Oracle® Multitenant
Administrator's Guide

23c
F46742-05
September 2023

Oracle Multitenant Administrator’s Guide, 23c

F46742-05

Primary Authors: Randy Urbano, Lance Ashdown, Donna Keesling, James Spiller

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

Contributors: Penny Avril, Thomas Baby, Hermann Baer, Yasin Baskan, Dominique Jeunot, 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

Part I Creating CDBs

2 Preparing to Create a CDB

3 Creating a CDB: Basic Steps

4 Creating a CDB: Advanced Topics

7 Creating a PDB from Scratch

8 Cloning a PDB

9 Relocating a PDB

10 Plugging In an Unplugged PDB

11 Creating a PDB as a Proxy PDB

12 Administering a PDB Snapshot Carousel

13 Removing a PDB

14 Creating and Removing Application Containers and Seeds

16 Administering PDBs

17 Administering an Application Container

Preface

This document describes how to create and configure CDBs, PDBs, and application containers.

Audience

This document explains how to administer containers *as* containers, for example, how to create CDBs and PDBs, start them up and shut them down, and perform cross-container operations. Specifically, this document is intended for database administrators who perform the following tasks:

- Create CDBs, PDBs, and application containers
- Relocate, unplug, and plug in PDBs and application containers
- Install and maintain applications in application containers
- Perform cross-container operations

> **Note:**
>
> *Oracle Database Administrator's Guide* describes traditional administrative tasks that you perform within an existing container, including managing database storage, schema objects, resources, and task scheduling.

To use this document, you must be familiar with relational database concepts. You must 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 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.

1

Introduction to Multitenant Administration

You can create and administer multitenant container databases (CDBs), pluggable databases (PDBs), and application containers.

- Changes in Oracle Database Release 23c for Oracle Multitenant Administrator's Guide
  The following features are new in this release.
- Multitenant Architecture
  The **multitenant architecture** enables an Oracle database to be a CDB.
- Benefits of the Multitenant Architecture
  Creating separate PDBs and application containers within a single CDB provides benefits for manageability and performance.
- Overview of Multitenant Administration
  Become familiar with basic concepts related to configuring and managing a multitenant environment.

Changes in Oracle Database Release 23c for Oracle Multitenant Administrator's Guide

The following features are new in this release.

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

- Hybrid read-only mode for pluggable databases
- Control PDB Open Order
- Oracle DBCA Support for Standard Edition High Availability

Hybrid read-only mode for pluggable databases

Starting with Oracle Database 23c, you can configure pluggable databases (PDBs) to operate in a new mode called hybrid read-only.

Hybrid read-only mode enables the PDB to operate as either read-write or read-only, depending on the user who is connected to the PDB. For common users, the PDB will be in both read-only and read-write mode. For local users, the PDB will be restricted to read-only mode.

To accommodate this enhancement, the following change has been made:

- The `ALTER PLUGGABLE DATABASE` statement has a new clause, `HYBRID READ ONLY`.

The `V$CONTAINER_TOPOLOGY` dynamic view has a new column, `IS_HYBRID_READ_ONLY`. The output of the `V$PDBS` dynamic view is also affected by this feature. For local users, the `OPEN_MODE` column shows `READ ONLY`; for common users, this column shows `READ WRITE`.

The benefit of using the hybrid read-only mode is that it enables database and application administrators to patch and maintain an application in a safe mode for open PDBs without the risk of local users, including higher privileged ones, interfering with the ongoing maintenance operation of the PDB.

Related Topics

- Opening a Pluggable Database in Hybrid Read-Only Mode
  Hybrid Read Only open mode is a special open mode in which PDB operates as Read Write as well as Read Only depending on which user is connected.
- Viewing the Open Mode of Each PDB
  The `V$PDBS` view provides information about the PDBs associated with the current database instance.
- Modifying a PDB with the ALTER PLUGGABLE DATABASE Statement
  To modify the attributes of a single PDB, use the `ALTER PLUGGABLE DATABASE` statement.
- About the Open Mode of a PDB
  When a PDB is mounted, you can open it in read/write, read-only, hybrid read-only or `MIGRATE` mode. You can also mount a PDB without opening it.
- Summary of PDB Open Modes
  Depending on the options that you specify in `ALTER PLUGGABLE DATABASE OPEN`, the PDB opens in different modes.
- 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.
- 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.
- Starting Up a PDB Using the STARTUP Command
  When the current container is a PDB, the SQL*Plus `STARTUP` command opens the PDB.
- 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.
- 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.
- Starting and Stopping PDBs in Oracle RAC
  You can use SRVCTL commands to manage PDBs.

Control PDB Open Order

Starting with Oracle Database 23c, customers can define a startup order for PDBs where the most important PDBs are started first. The database administrator can define a priority for each PDB. The priority is applied to PDB opening order and upgrade order as follows:

- Restoring PDB states when opening the CDB
- Setting PDB states when using the PDB OPEN ALL statement
- Setting the order for PDB database upgrade operations
- Starting PDBs in an ADG switchover or failover

This feature allows critical PDBs to start and open before less important PDBs, reducing the time for the critical applications to become usable.

Related Topics

- 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.
- To Set the Priority of a PDB
  Use the ALTER PLUGGABLE DATABASE <databasename> Priority <value> set the priority.

Oracle DBCA Support for Standard Edition High Availability

Starting with Oracle Database 23c, Oracle Database Configuration Assistant (Oracle DBCA) enables you to create a Standard Edition High Availability, single-instance Oracle Database. This database is a Single instance database with failover capability. This database uses ASM or ACFS for the database storage files(NAS is not supported).

Oracle DBCA automatically registers your database and allows you to select the cluster nodes that you want to configure for your Standard Edition High Availability, single instance database deployment. The SPFILE and password files are automatically created in ASM or ACFS based on db storage location.

Related Topics

- About Standard Edition High Availability
  In this release, you can install Oracle Database Standard Edition 2 in high availability mode.
- Requirements for Using Standard Edition High Availability With Oracle Databases
  To use Standard Edition High Availability, deploy Oracle Database Standard Edition 2 in accordance with these configuration requirements.
- Create Standard Edition High Availability Database Using DBCA
  Oracle Database Configuration Assistant (Oracle DBCA) enables you to create a Standard Edition High Availability (SEHA), single-instance Oracle Database.

Multitenant Architecture

The **multitenant architecture** enables an Oracle database to be a CDB.

Every Oracle database must contain or be able to be contained by another database. For example, a CDB contains PDBs, and an application container contains application PDBs. A PDB is contained by a CDB or application container, and an application container is contained by a CDB.

Starting in Oracle Database 21c, a multitenant container database is the only supported architecture. In previous releases, Oracle supported non-container databases (non-CDBs).

- CDBs
  A CDB contains one or more user-created PDBs and application containers.
- PDBs
  A **PDB** is a portable collection of schemas, schema objects, and nonschema objects that appears to an application as a separate database.
- Application Containers
  An **application container** is an optional, user-created container within a CDB that stores data and metadata for one or more applications.

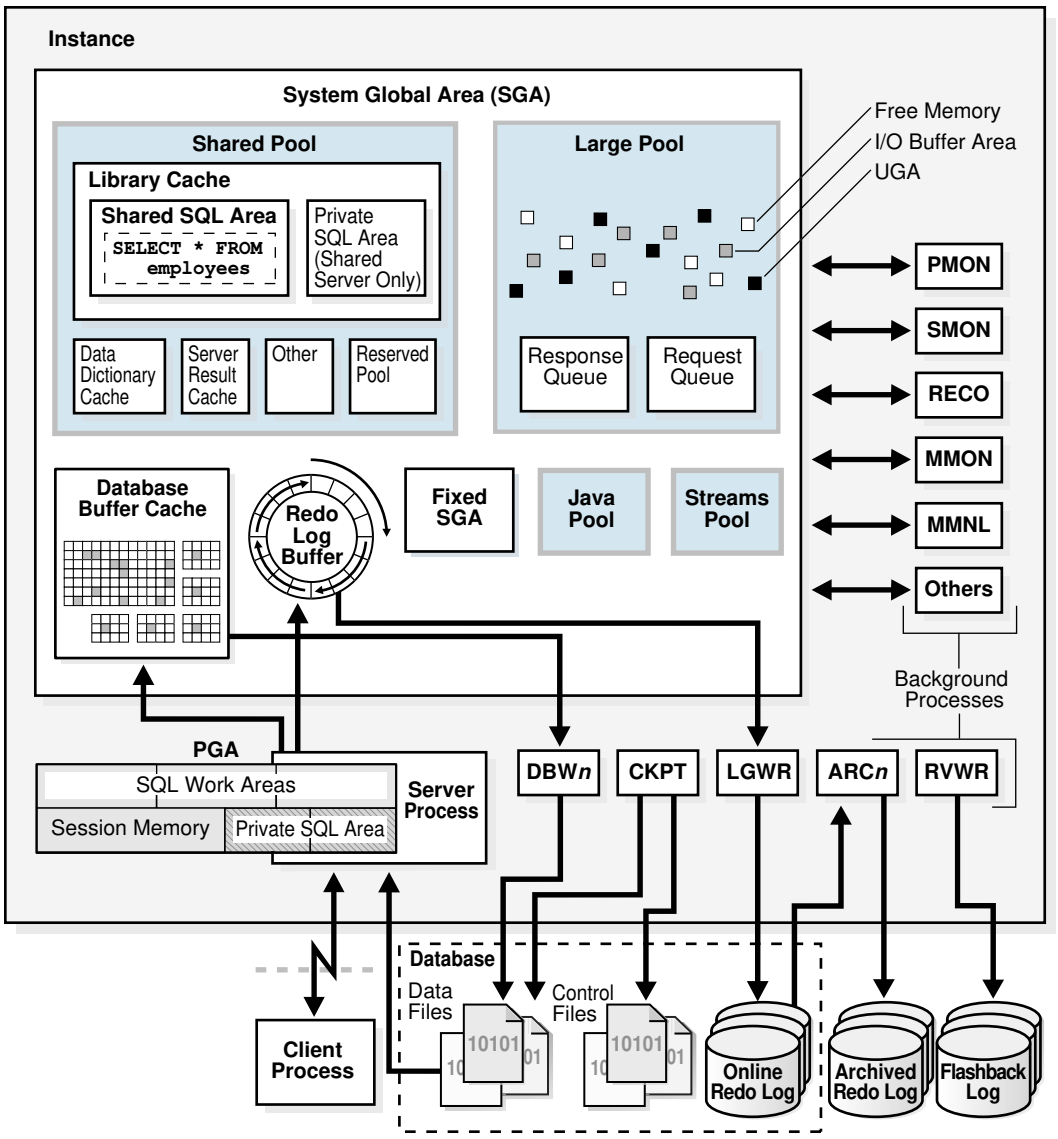
CDBs

A CDB contains one or more user-created PDBs and application containers.

At the physical level, a CDB is a set of files: control file, online redo log files, and data files. The database instance manages the files that make up the CDB.

The following figure shows a CDB and an associated database instance.

Figure 1-1 Database Instance and CDB



PDBs

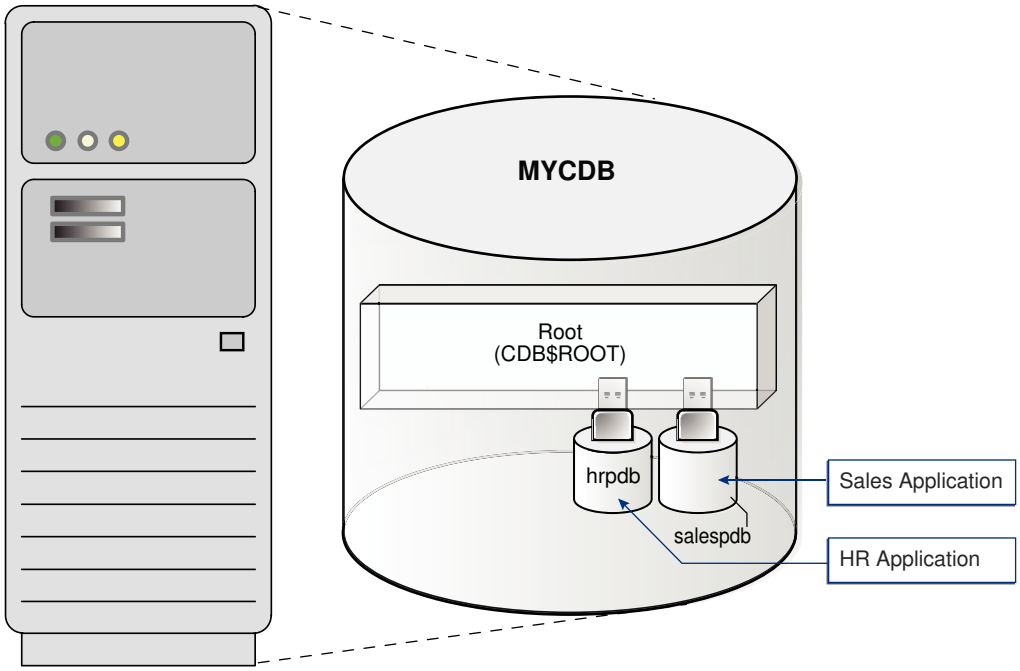
A **PDB** is a portable collection of schemas, schema objects, and nonschema objects that appears to an application as a separate database.

At the physical level, each PDB has its own set of data files that store the data for the PDB. The CDB includes all the data files for the PDBs contained within it, and a set of system data files that store metadata for the CDB itself.

To move or archive a PDB, you can unplug it. An unplugged PDB consists of the PDB data files and a metadata file. An unplugged PDB is not usable until it is plugged in to a CDB.

The following figure shows a CDB named `MYCDB`.

Figure 1-2 PDBs in a CDB



Physically, `MYCDB` is an Oracle database, in the sense of a set of data files associated with an instance. `MYCDB` has one database instance, although multiple instances are possible in Oracle Real Application Clusters, and one set of database files.

`MYCDB` contains two PDBs: `hrpdb` and `salespdb`. As shown in Figure 1-2, these PDBs appear to their respective applications as separate, independent databases. An application has no knowledge of whether it is connecting to a CDB or PDB.

To administer the CDB itself or any PDB within it, you can connect to the CDB root. The root is a collection of schemas, schema objects, and nonschema objects to which all PDBs and application containers belong.

Application Containers

An **application container** is an optional, user-created container within a CDB that stores data and metadata for one or more applications.

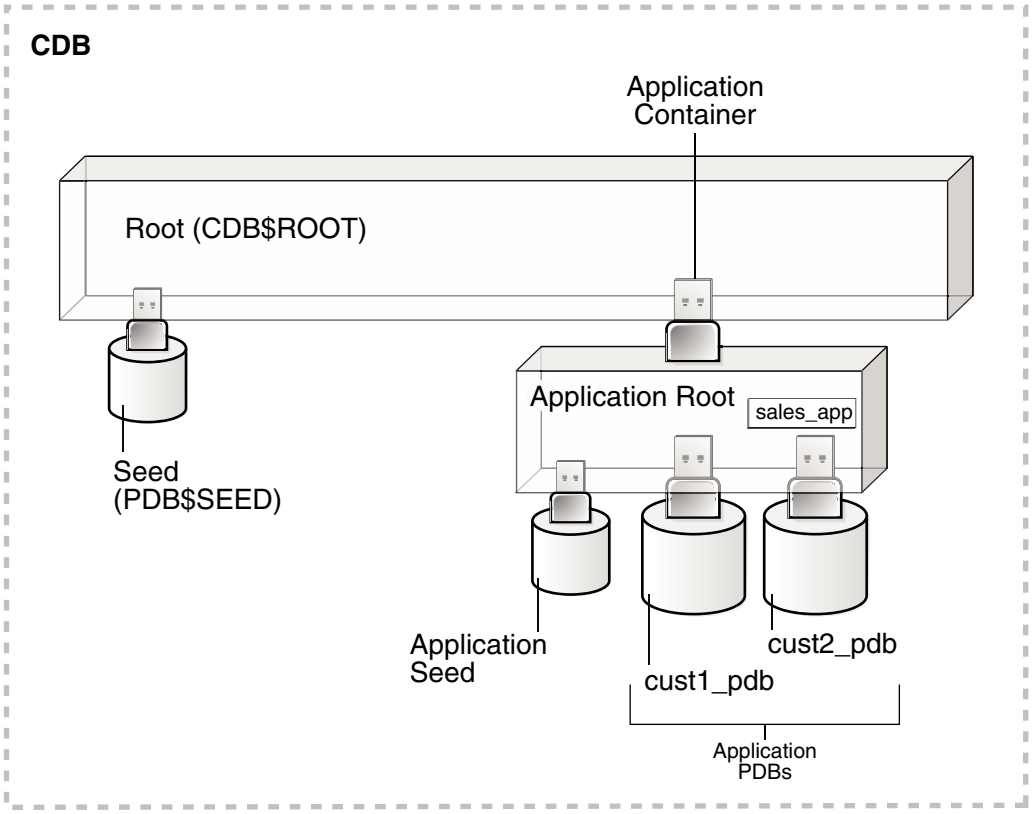
In this context, an application (also called the *master application definition*) is a named, versioned set of common data and metadata stored in the application root. For example, the application might include definitions of tables, views, user accounts, and PL/SQL packages that are common to a set of PDBs.

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 application PDBs, and enables these PDBs to share metadata and data. At the physical level, an application container has its own set of data files, just like a PDB.

For example, a SaaS deployment can use multiple application PDBs, each for a separate customer, which share application metadata and data. For example, in the following figure, `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 1-3 SaaS Use Case



Benefits of the Multitenant Architecture

Creating separate PDBs and application containers within a single CDB provides benefits for manageability and performance.

- Benefits of Consolidating Data into a Single CDB
  Database consolidation is the process of consolidating data from multiple databases on separate hosts into one CDB on one host. The multitenant architecture enables you to consolidate data and code *without altering existing schemas or applications*.
- Benefits of the Multitenant Architecture for Manageability
  The multitenant architecture improves manageability by storing the data *and* metadata specific to a PDB in the PDB itself.

Benefits of Consolidating Data into a Single CDB

Database consolidation is the process of consolidating data from multiple databases on separate hosts into one CDB on one host. The multitenant architecture enables you to consolidate data and code *without altering existing schemas or applications*.

Consolidating data into a single CDB 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 in a single CDB on a single host can 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 CDB, PDBs appears to applications as separate databases. For example, if user error loses critical data in a single PDB, then the 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 account can connect to any container on which it has sufficient privileges, whereas a local user account is restricted to a specific PDB. Administrators can divide duties as follows:

  – An administrator uses a common user account to manage a CDB or application container.

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

- Ease of performance tuning

  It is easier to collect performance metrics for a single CDB on one host than for multiple databases on multiple hosts. For example, it is easier to size one SGA than 100 SGAs.

- Fewer database patches and upgrades

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

See Also:

- "Overview of Multitenant Administration"
- *Oracle Database Security Guide* to learn about common user accounts

Benefits of the Multitenant Architecture for Manageability

The multitenant architecture improves manageability by storing the data *and* metadata specific to a PDB in the PDB itself.

By storing its own dictionary metadata, a PDB becomes easier to manage as a 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 CDB 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 Oracle 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.

- 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 definition 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 (the Resource Manager)

  In the multitenant environment, PDBs contend for shared resources. To address resource contention, usage, and monitoring issues, use the Resource Manager.

See Also:

- "Administering an Application Container"
- *Oracle Database Administrator's Guide* to learn more about the Resource Manager
- *Oracle Database Concepts* to learn more about data dictionary separation

Overview of Multitenant Administration

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

- Users, Roles, and Objects in a Multitenant Environment
  The container architecture enables database administrators to assume different roles. The key to the separation of duties is the distinction between common and local users, roles, and objects.
- Tasks and Tools for a Multitenant Environment
  This manual explains how to create and perform operations on containers using command-line tools such as SQL*Plus or SQL Developer.
- Overview of Container Creation
  You create a CDB using `CREATE DATABASE`, and then create PDBs and application containers using `CREATE PLUGGABLE DATABASE`.

Users, Roles, and Objects in a Multitenant Environment

The container architecture enables database administrators to assume different roles. The key to the separation of duties is the distinction between common and local users, roles, and objects.

- 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.
- About Common and Local User Accounts
  A database user account has a password and specific database privileges.
- 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.
- Common and Local Objects
  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.
- Separation of Duties in CDB and PDB Administration
  Some database administrators manage an entire CDB, while others manage individual PDBs.

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

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 a separate database that is distinct from any other PDBs.
- 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

About Common and Local User Accounts

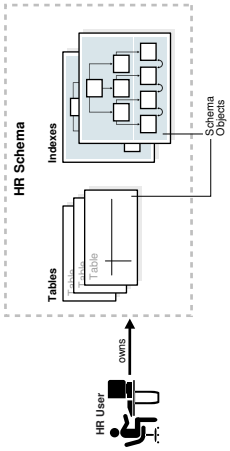
A database user account has a password and specific database privileges.

User Accounts and Schemas

Each user account owns a single schema, which has the same name as the user. The schema contains the data for the user owning the schema. For example, the `hr` user account owns the `hr` schema, which contains schema objects such as the `employees` table. In a production database, the schema owner usually represents a database application rather than a person.

Within a schema, each schema object of a particular type has a unique name. For example, `hr.employees` refers to the table `employees` in the `hr` schema. The following figure depicts a schema owner named `hr` and schema objects within the `hr` schema.

Figure 1-4 HR Schema

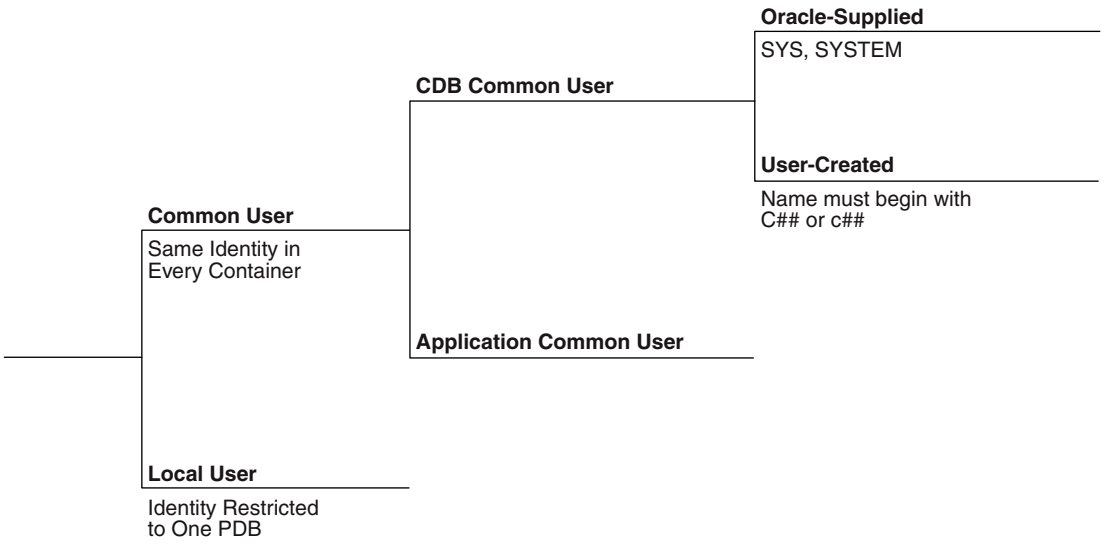


Common and Local User Accounts

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 1-5 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.

- Common User Accounts
  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.
- Local User Accounts
  A **local user** is a database user that is not common and can operate only within a single PDB.

Common User Accounts

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.

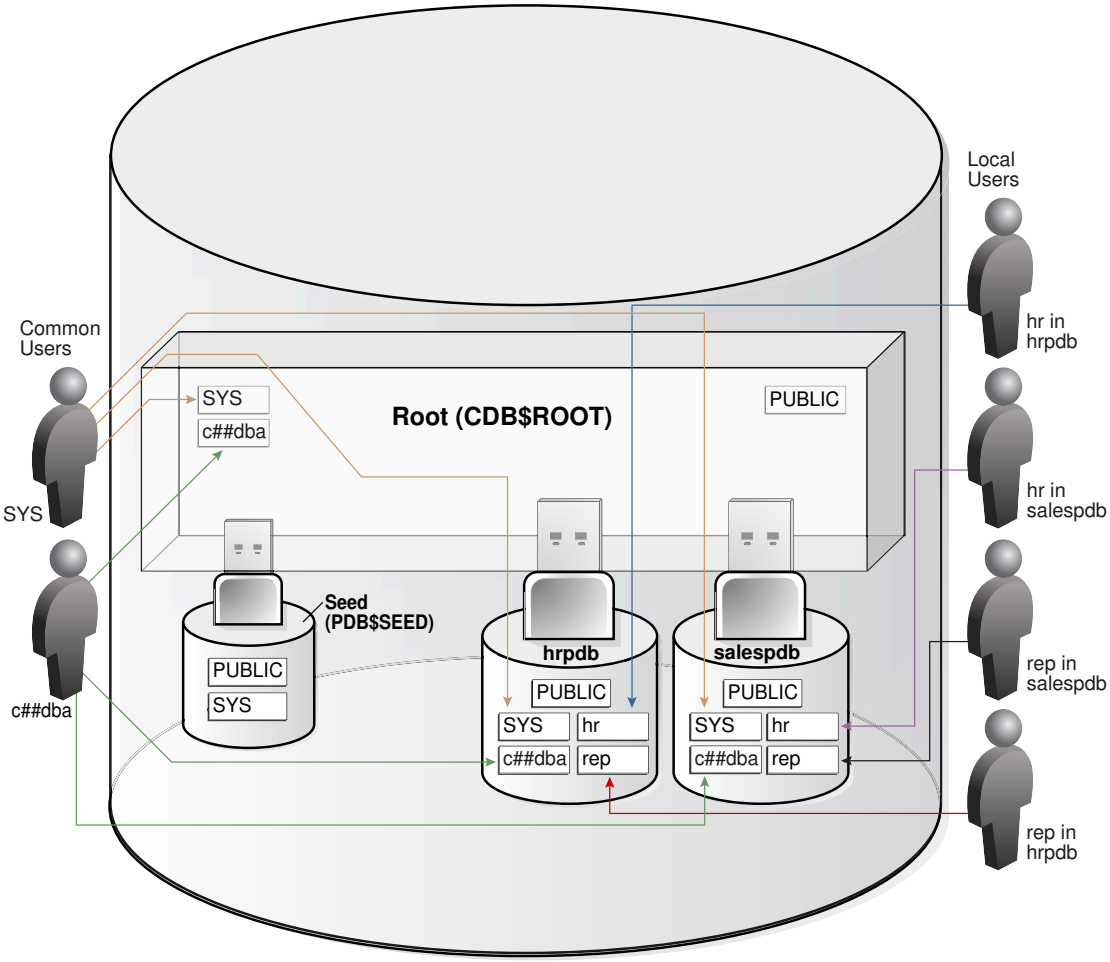
Every common user can connect to and perform operations within the root of its container, and within any PDB in which it has sufficient privileges. Some administrative tasks must be performed by a common user. Examples include creating a PDB and unplugging a PDB.

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.

The following figure 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 1-6 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.
- Characteristics of Common Users
  Every common user is either Oracle-supplied or user-created.
- SYS and SYSTEM Accounts
  All Oracle databases include default common user accounts with administrative privileges.

Characteristics of Common Users

Every common user is either Oracle-supplied or user-created.

Common user accounts 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.

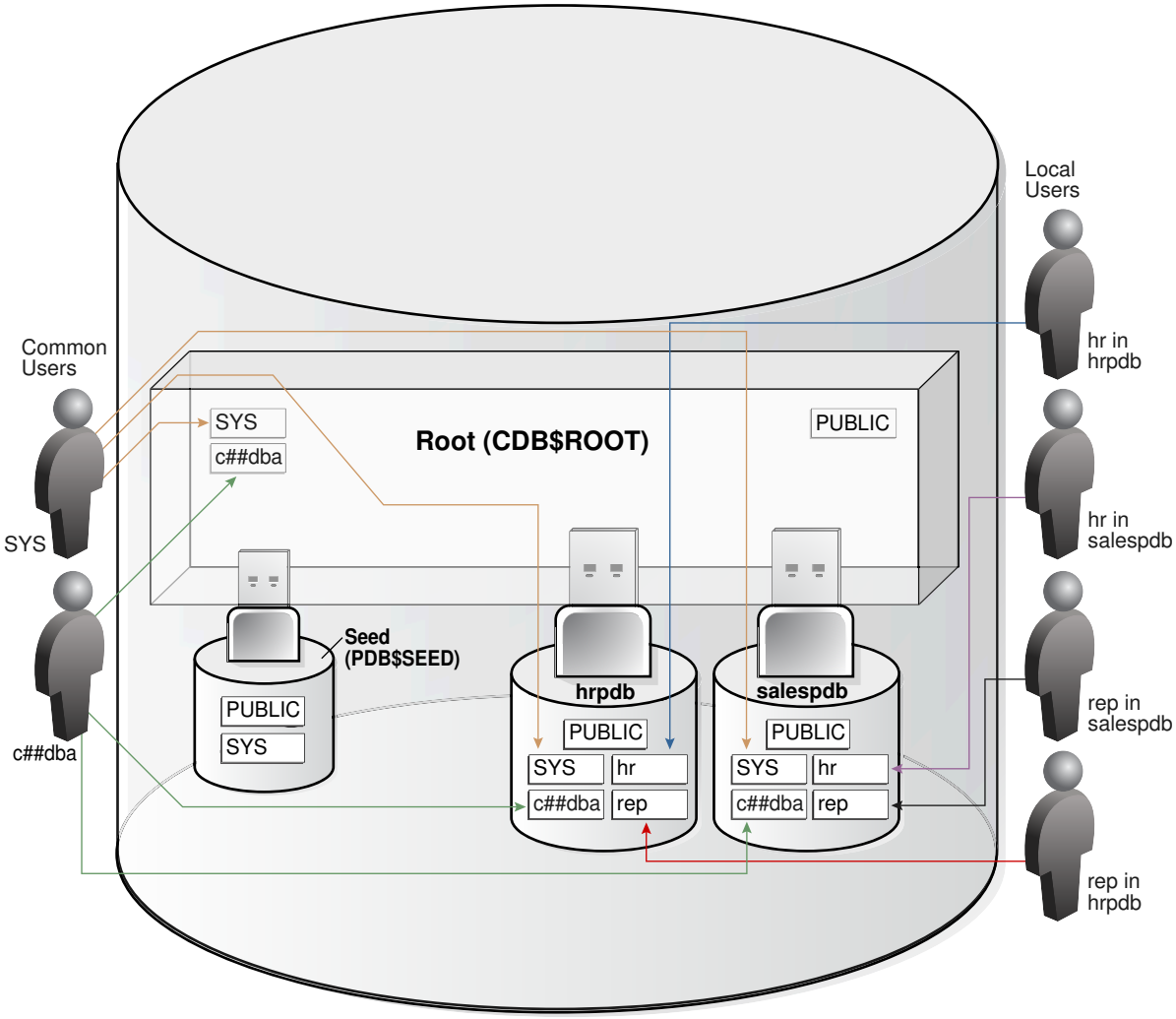
  Thus, an application common user is restricted to its own application container. For example, the application common user created in the `saas_sales` application can connect only to the application root and the PDBs in the `saas_sales` application container.
- The names of user-created CDB common users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

The following figure 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 1-7 Users and Schemas in a CDB



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

SYS and SYSTEM Accounts

All Oracle databases include default common user accounts with administrative privileges.

Administrative accounts are highly privileged and are intended only for DBAs authorized to perform tasks such as starting and stopping the database, managing memory and storage, creating and managing database users, and so on.

The `SYS` common user account is automatically created when a database is created. This account can perform all database administrative functions. The `SYS` schema stores the base tables and views for the data dictionary. These base tables and views are critical for the operation of Oracle Database. Tables in the `SYS` schema are manipulated only by the database and must never be modified by any user.

The `SYSTEM` administrative account is also automatically created when a database is created. The `SYSTEM` schema stores additional tables and views that display administrative information, and internal tables and views used by various Oracle Database options and tools. Never use the `SYSTEM` schema to store tables of interest to nonadministrative users.

See Also:

- *Oracle Database Security Guide* to learn about user accounts
- *Oracle Database Administrator's Guide* to learn about `SYS`, `SYSTEM`, and other administrative accounts

Local User Accounts

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 the example shown in "Characteristics of Common Users", 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. "Characteristics of Common Users" 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 "Characteristics of Common Users". Each row describes an action that occurs after the action in the preceding row. Common user SYSTEM creates local users in two PDBs.

Table 1-1 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.

`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`.

- 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).
- Local Roles in a CDB
  A **local role** exists only in a single PDB, and is thus completely independent of local roles in any other PDBs.

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.

Local Roles in a CDB

A **local role** exists only in a single PDB, and is thus completely independent of local roles in any other PDBs.

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.

Common and Local Objects

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:**
>
> *Oracle Database Security Guide* to learn more about privilege management for common objects

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 CDB DBAs can create, unplug, plug in, and drop PDBs. They can also specify the temporary tablespace and the default tablespace for the CDB root, and they can change the open mode of PDBs.

DBAs can also connect to a specific PDB as a local PDB administrator. The PDB DBA performs 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.

Tasks and Tools for a Multitenant Environment

This manual explains how to create and perform operations on containers using command-line tools such as SQL*Plus or SQL Developer.

- Tasks for a Multitenant Environment
  This section summarizes the tasks required to manage a multitenant environment.
- Tools for a Multitenant Environment
  You can use various tools to configure and administer a multitenant environment.

Tasks for a Multitenant Environment

This section summarizes the tasks required to manage a multitenant environment.

This manual explains how to administer containers *as* containers, for example, how to create CDBs and PDBs, start them up and shut them down, and perform cross-container operations. *Oracle Database Administrator's Guide* describes traditional administrative tasks that you perform within an existing container, including managing database storage, schema objects, resources, and task scheduling.

To achieve the goals described in "Benefits of the Multitenant Architecture", you must complete the following general tasks:

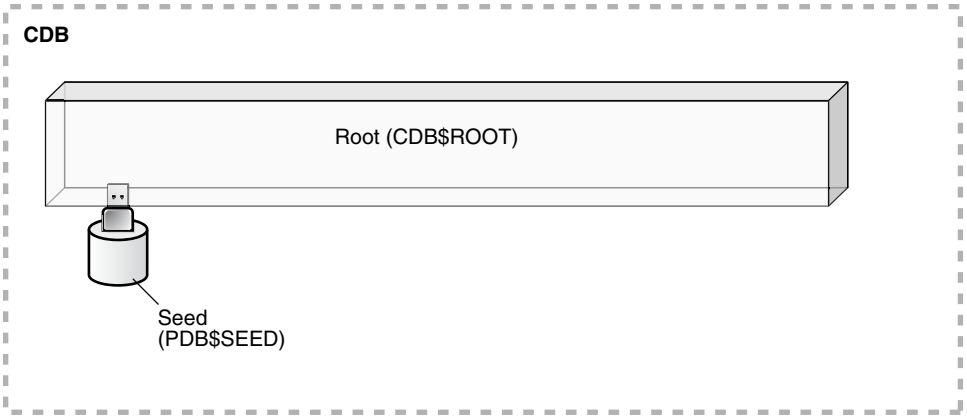
Task 1 Plan for the Multitenant Environment
Creating and configuring any database requires careful planning. A CDB requires special considerations. For example, consider the following factors when you plan for a CDB:

- The number of PDBs that will be plugged into each CDB
- The resources required to support the planned CDB
- Container management policies run as an aggregate on the entire CDB or run 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 "Preparing to Create a CDB" 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 ... ENABLE PLUGGABLE DATABASE` command. In either case, you must specify the configuration details for each CDB.
See "Creating a CDB with DBCA" and "Creating a Database with the CREATE DATABASE Statement" for detailed information about creating a CDB.
After a CDB is created, it consists of the root and `PDB$SEED`, as shown in the following figure. The CDB root contains only Oracle maintained objects and data structures, and `PDB$SEED` is a generic seed database for cloning purposes.

Figure 1-8 A Newly Created CDB

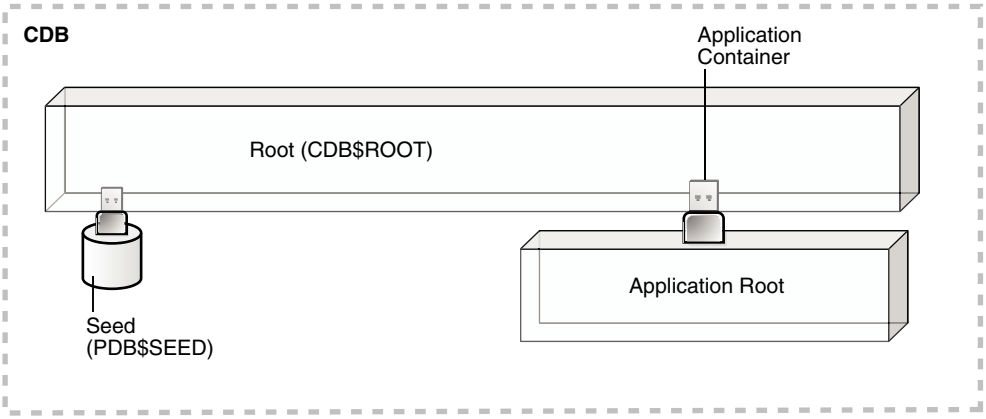


Task 3 Optionally, Create Application Containers

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 1-9 An Application Container



See "About Application Containers".

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 PDBs and Application Containers" for information about creating PDBs, plugging in PDBs, and unplugging PDBs.

The following figure shows a CDB with several PDBs.

Figure 1-10 A CDB with PDBs

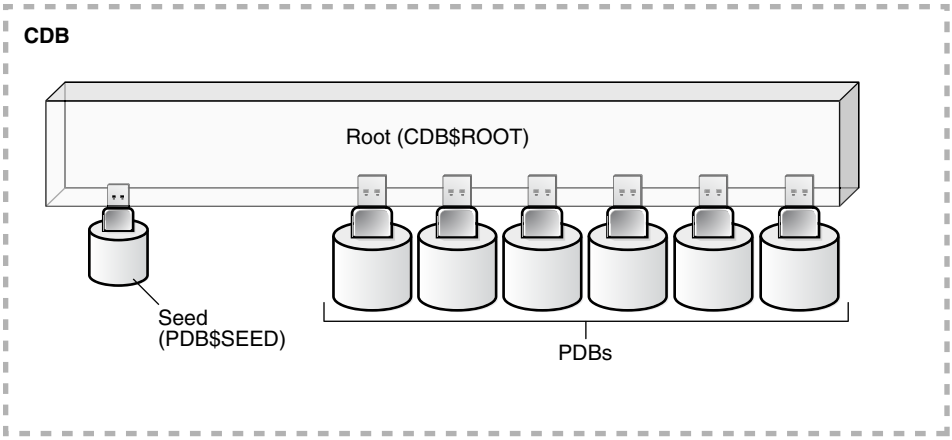
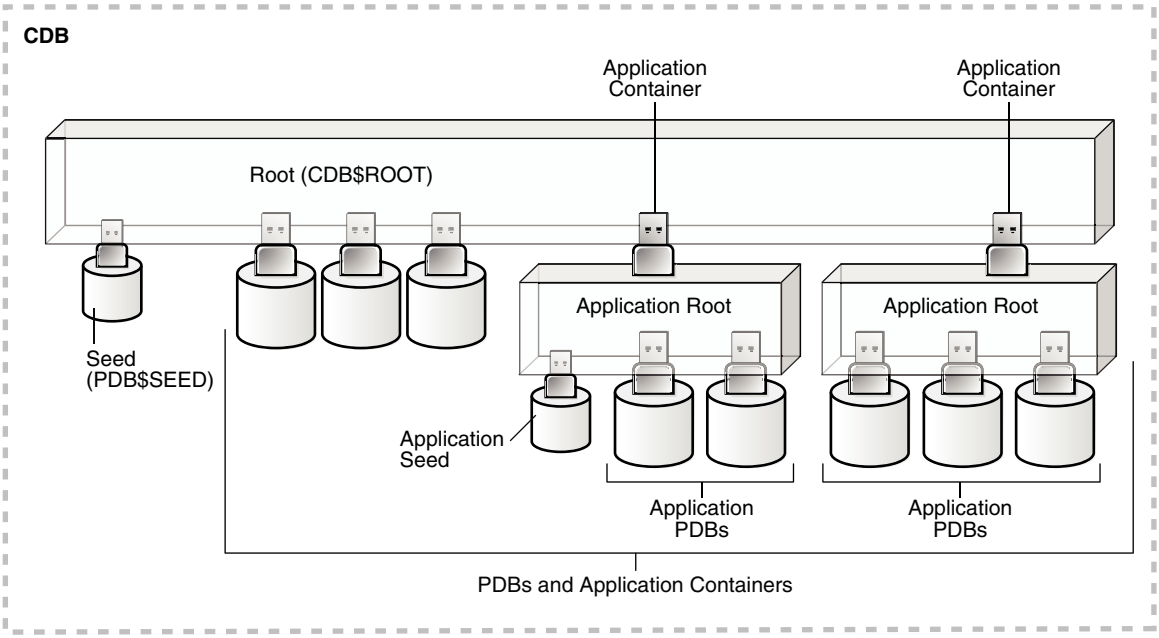


Figure 1-11 shows a CDB with PDBs, application containers, and application PDBs.

Figure 1-11 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 CDB root, and some attributes of PDBs.. 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 Containers in a CDB".

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.
You can also use Oracle Scheduler to schedule jobs in a CDB and in individual PDBs. See *Oracle Database Administrator's Guide*.

Task 6 Administer and Monitor PDBs and Application PDBs
See " Administering PDBs" and "Monitoring Containers in a CDB".

Tools for a Multitenant Environment

You can use various tools to configure and administer a multitenant environment.

Table 1-2 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 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"
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.

Overview of Container Creation

You create a CDB using `CREATE DATABASE`, and then create PDBs and application containers using `CREATE PLUGGABLE DATABASE`.

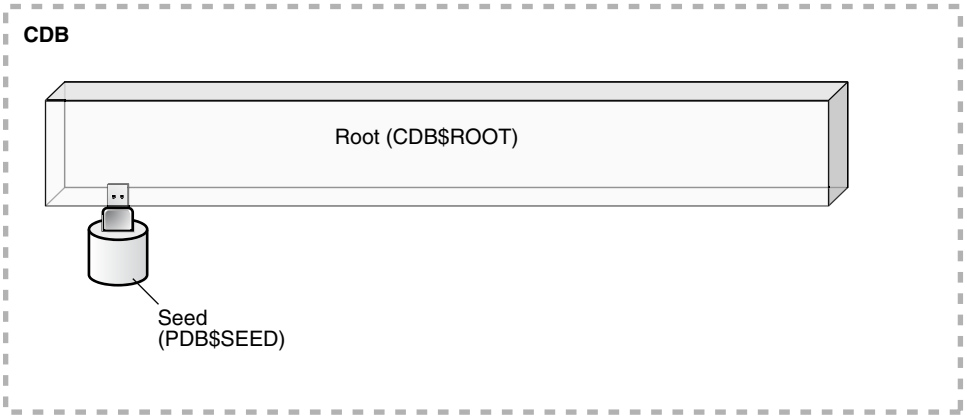
- Creation of a CDB
  The `CREATE DATABASE` statement creates a new CDB.
- Creation of a PDB or Application Container
  The `CREATE PLUGGABLE DATABASE` SQL statement creates a PDB. Specifying the `AS APPLICATION CONTAINER` clause creates an application container.

Creation of a CDB

The `CREATE DATABASE` statement creates a new CDB.

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 1-12 CDB with Seed PDB



See Also:

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

Creation of a PDB or Application Container

The `CREATE PLUGGABLE DATABASE` SQL statement creates a PDB. Specifying the `AS APPLICATION CONTAINER` clause creates an application container.

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

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

> **Note:**
>
> In the following topics, the term "PDB" refers to a PDB, application container, or application PDB.

- Creation of a PDB by Cloning
  One technique for creating a PDB is called *cloning*.
- 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.
- 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.
- Creation of a PDB as a Proxy PDB
  A **proxy PDB** provides access to different PDB, called the **referenced PDB**, in a remote CDB.

> **See Also:**
>
> "Creating 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, or a remote or local PDB.

- Creation of a PDB from a Seed
  You can use the `CREATE PLUGGABLE DATABASE` statement to create a PDB from a seed.
- Creation of a PDB by Cloning a PDB
  To clone a PDB from another PDB, use the `CREATE PLUGGABLE DATABASE` statement with the `FROM` clause.

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.

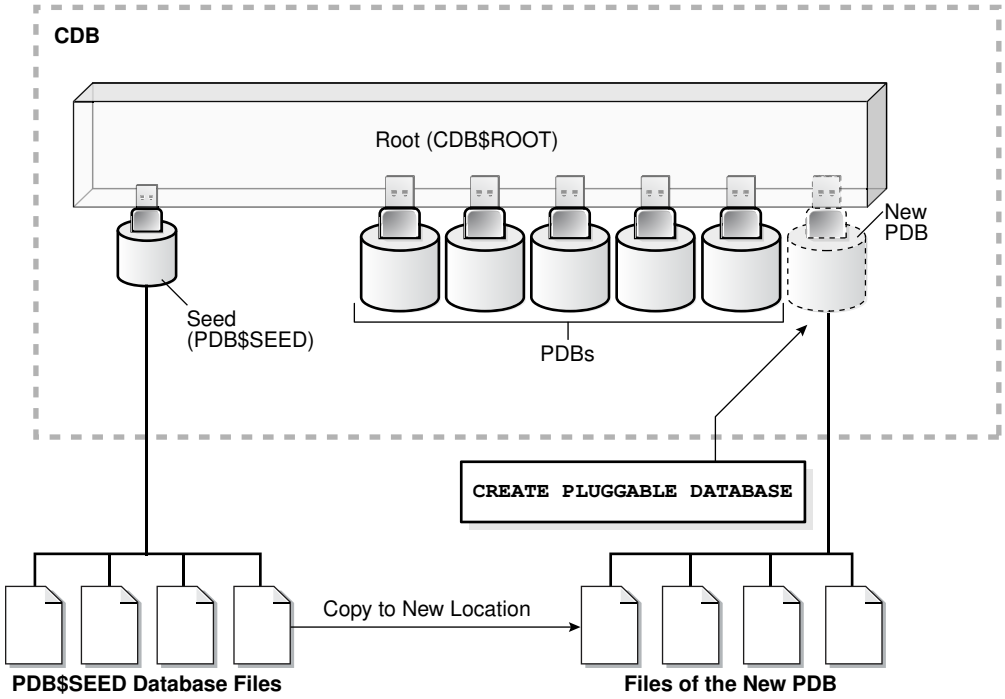
A seed PDB is either of the following:

- 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 1-13 Creation from PDB$SEED



Example 1-1 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

To clone a PDB from another PDB, use the `CREATE PLUGGABLE DATABASE` statement with the `FROM` clause.

In this technique, the source is 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.

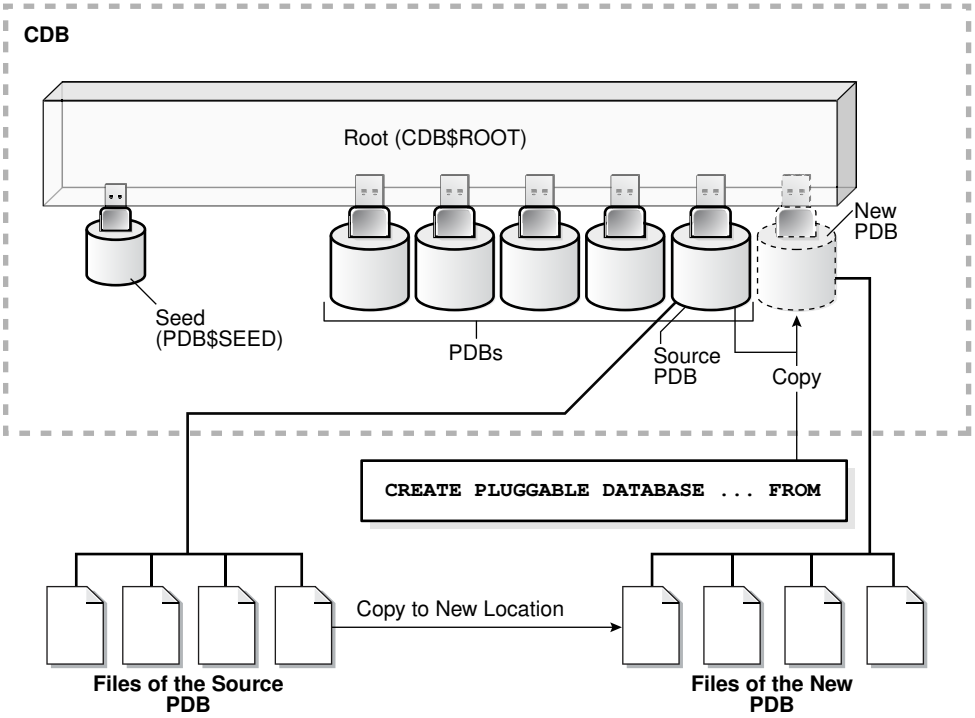
This technique is useful for quickly creating PDBs for testing and development. For example, you might test a new or modified application on a cloned PDB before deploying the application in a production PDB. If a PDB is in local undo mode, then the source PDB can be open in read/write mode during the 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 you create the cloned PDB 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 1-14 Cloning a PDB



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

Example 1-2 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;
```

- Clones from PDB Snapshots
  Create a **clone from a PDB snapshot** by specifying `USING SNAPSHOT` clause of the `CREATE PLUGGABLE DATABASE` command.
- 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.
- Refreshable Clone PDBs
  A **refreshable clone PDB** is a read-only clone that can periodically synchronize with its source PDB.

> **See Also:**
>
> - "Cloning a PDB"
> - "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

If you create a clone from a PDB snapshot using the `SNAPSHOT COPY` clause, then the PDB is a snapshot copy PDB and is based on a copy of the underlying storage system.

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" to learn how to clone a PDB using the `REFRESH MODE` clause

Creation of a PDB by Plugging In an Unplugged PDB

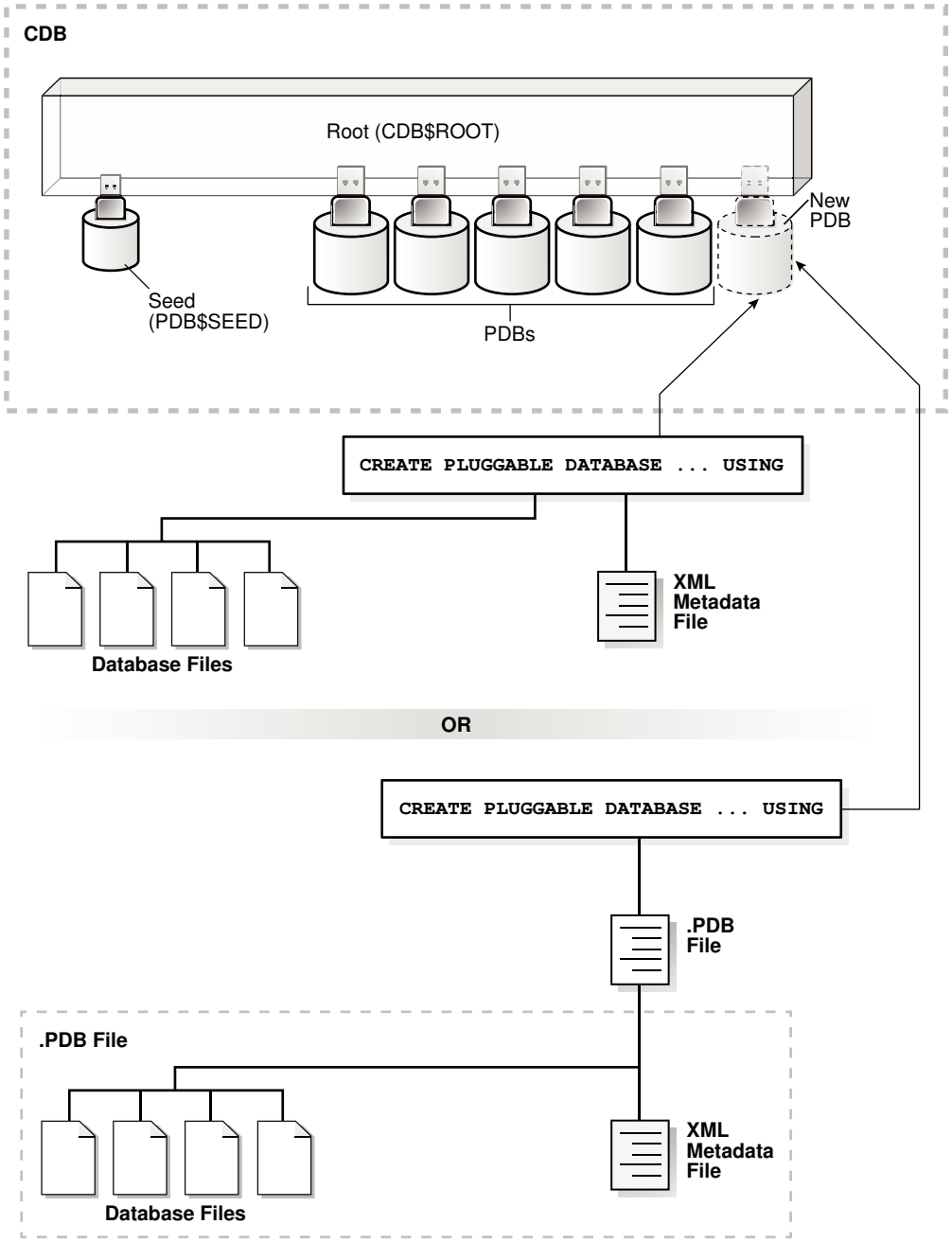
An **unplugged PDB** is a self-contained set of data files, and an XML metadata file that specifies the locations of the PDB files. To plug in an unplugged PDB, use the `CREATE PLUGGABLE DATABASE` statement with the `USING` clause.

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 1-15 Plugging In an Unplugged PDB



Example 1-3 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 by Relocating

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

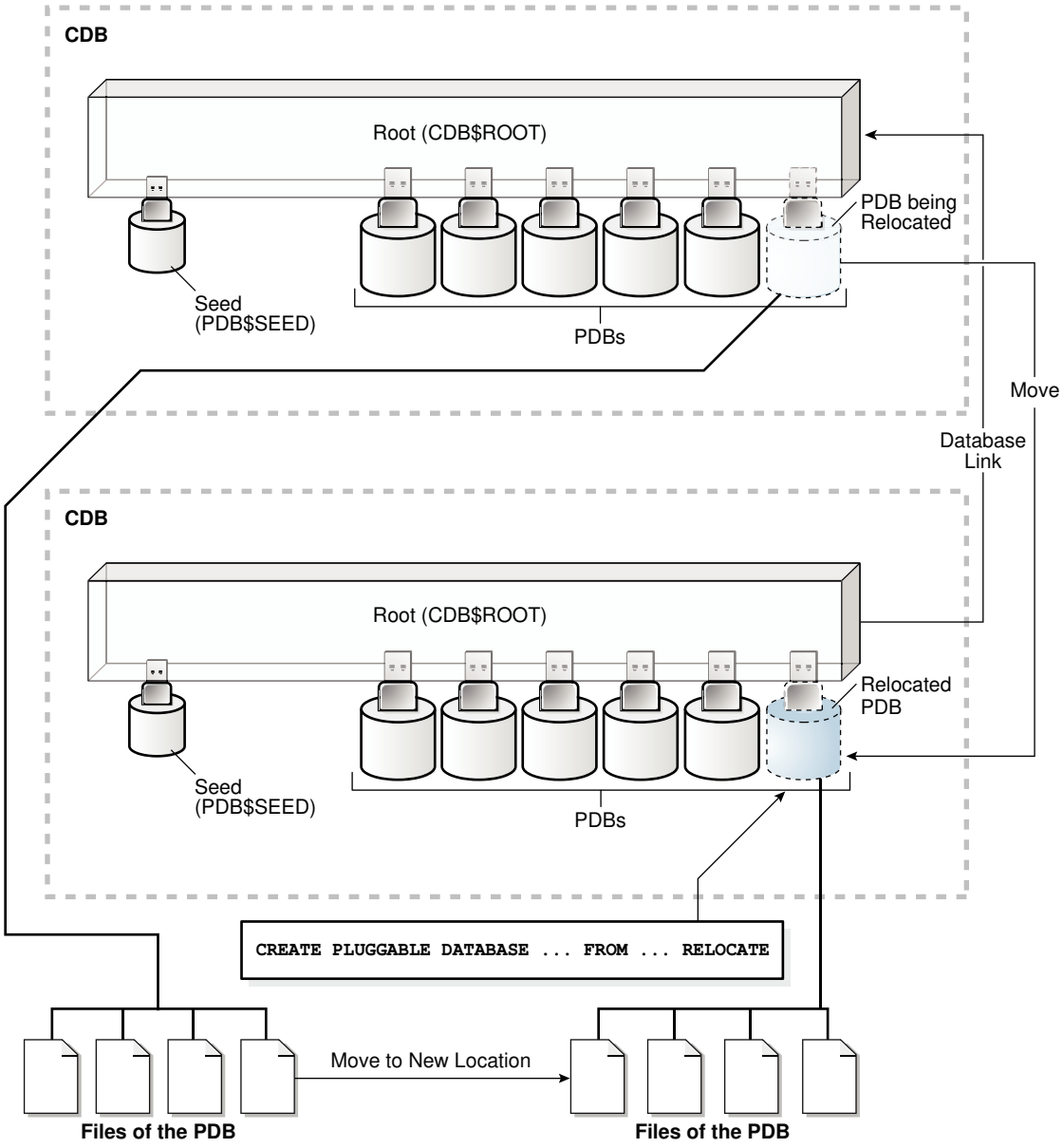
This technique has the following advantages:

- 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 1-16 Relocating a PDB



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

Example 1-4 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:**
>
> "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.

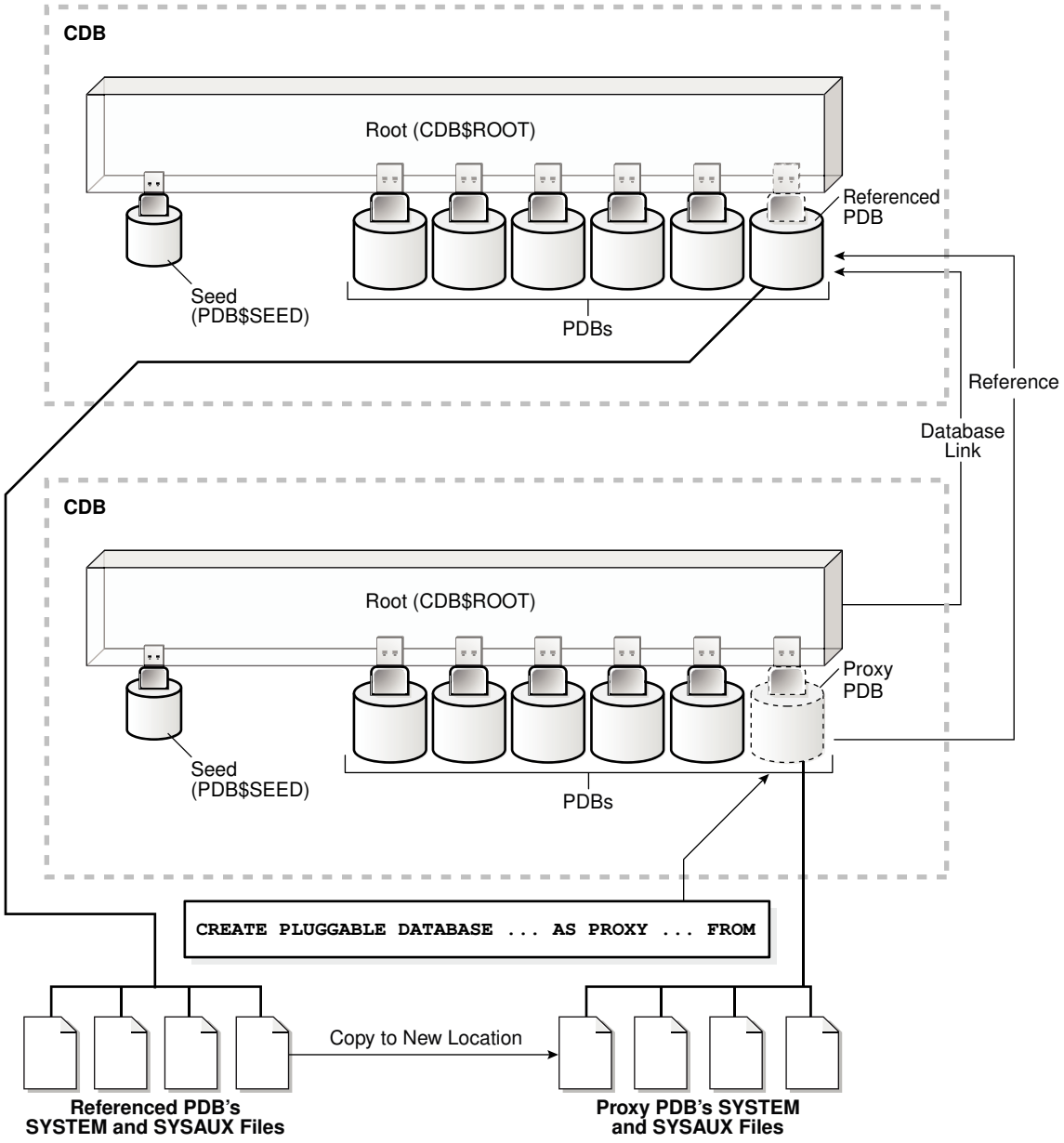
A typical use case is a proxy PDB that references an application root replica. If multiple CDBs have the same application definition (for example, same tables and PL/SQL packages), then you can create a proxy PDB in the application container of the master application root. The referenced PDB for the 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 1-17 Creating a Proxy PDB



Example 1-5 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"

Part I
Creating CDBs

You can create CDBs using the `CREATE DATABASE` command.

- Preparing to Create a CDB
  Before creating the CDB, you must make many important decisions: physical layout, database character set, database block sizes, and so on.
- Creating a CDB: Basic Steps
  After you plan your CDB, you can create it with a graphical tool or a SQL command.
- Creating a CDB: Advanced Topics
  This chapter covers creating a CDB in greater detail.
- Configuring a CDB Fleet
  A **CDB fleet** is a collection of CDBs and hosted PDBs that you can manage as one logical CDB.

2

Preparing to Create a CDB

Before creating the CDB, you must make many important decisions: physical layout, database character set, database block sizes, and so on.

- Prerequisites for a Multitenant Environment
  Prerequisites must be met for a multitenant environment.
- Deciding When to Create a CDB
  You can create the CDB either during or after Oracle Database software installation.
- Deciding How to Configure the CDB
  Prepare to create the CDB by research and careful planning.

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

Deciding When to Create a CDB

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. 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 techniques for creating a CDB are:

- With the Database Configuration Assistant (DBCA), a graphical tool.

  See "Creating a CDB with DBCA".
- With the `CREATE DATABASE ... ENABLE PLUGGABLE DATABASE` SQL command.

  See "Creating a Database with the CREATE DATABASE Statement".

Deciding How to Configure the CDB

Prepare to create the CDB by research and careful planning.

- Plan the PDBs
  Plan the tables and indexes for the pluggable databases (PDBs) and estimate the amount of space they require.
- Plan the Physical Layout
  Plan the layout of the underlying operating system files your CDB will comprise.
- Learn How to Manage Initialization Parameters
  Familiarize yourself with the initialization parameters that can be included in an initialization parameter file.
- Select the Character Set
  You must choose a character set for the CDB.
- Decide Which Time Zones to Support
  Consider which time zones your CDB must support.
- Select the Database and Redo Log Block Sizes
  Select the standard database block size for the CDB.
- Plan the SYSTEM and SYSAUX Tablespaces
  There is a separate `SYSAUX` and `SYSTEM` tablespace for the CDB root and for each PDB.

- Plan the Temporary Tablespaces
  Plan to use default temporary tablespaces.
- Choose the Undo Mode
  Plan to use an undo tablespace to manage your undo data.
- Plan the Services for Your Application
  Plan for the database services required to meet the needs of your applications.
- 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.
- Plan for Oracle RAC
  If you plan to use Oracle RAC, then plan for an Oracle RAC environment.

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 PDBs and Application Containers"
- *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 online redo log for a single-instance CDB, or one online redo log for each instance of an Oracle Real Application Clusters (Oracle RAC) CDB. Also, for Oracle RAC, all data files and online 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 upgrading a non-CDB to a PDB, 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.

- Default CDB Character Set
  It is important to select the right character set for your CDB. Oracle recommends AL32UTF8 as the CDB character set.
- Different Character Sets for CDB and PDBs
  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.

See Also:

Oracle Database Globalization Support Guide

Default CDB Character Set

It is important to select the right character set for your CDB. Oracle recommends AL32UTF8 as the CDB character set.

AL32UTF8 is Oracle's name for the UTF-8 encoding of the Unicode standard. The Unicode standard is the universal character set that supports most of the currently spoken languages of the world. The use of the Unicode standard is indispensable for any multilingual technology, including database processing.

After a CDB is created and accumulates production data, changing the database character set is a time consuming and complex project. Therefore, it is very important to select the right

character set at installation time. Even if the database does not currently store multilingual data but is expected to store multilingual data within a few years, the choice of AL32UTF8 for the database character set is usually the only good decision. The universality and flexibility of Unicode typically outweighs some additional cost associated with it, such as slightly slower text processing compared to single-byte character sets and higher storage space requirements for non-ASCII text compared to non-Unicode character sets.

If you do not want to use AL32UTF8, and you are not restricted in your choice by a vendor requirement, then Oracle suggests that you use one of the character sets listed as recommended for the database. The recommended character sets were selected based on the requirements of modern client operating systems. Oracle Universal Installer (OUI) presents the recommended list only, and Database Configuration Assistant (DBCA) must be used separately to choose a non-recommended character set. In addition, the default database creation configuration in DBCA allows the selection of the recommended character sets only. You must use the advanced configuration mode of DBCA or the `CREATE DATABASE` statement to select a non-recommended character set.

If no character set choice is presented in an OUI or a DBCA installation mode, then AL32UTF8 is used as the database character set, unless a custom database template with another character set has been selected.

Note:

- AL32UTF8 is the proper implementation of the Unicode encoding UTF-8. AL32UTF8 is used as the default database character set while creating a database using Oracle Universal Installer (OUI) as well as Oracle Database Configuration Assistant (DBCA).
- You can only select an ASCII-based character set for the database on an ASCII-based platform.

Caution:

Do not use UTF8 as the database character set unless required unless explicitly requested by your application vendor. Despite having a very similar name, UTF8 is not a proper implementation of the Unicode encoding UTF-8. If the UTF8 character set is used where UTF-8 processing is expected, data loss and security issues may occur. This is especially true for Web related data, such as XML and URL addresses.

AL32UTF8 and UTF8 character sets are not compatible with each other as they have different maximum character widths. AL32UTF8 has a maximum character width of 4 bytes, whereas UTF8 has a maximum character width of 3 bytes.

> **See Also:**
>
> *Oracle Database Globalization Support Guide* for information about the character sets recommended for the database

Different Character Sets for CDB and PDBs

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 you create 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`.

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:**
>
> "Specifying the Database Time Zone File" 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.

- "Specifying Database Block Sizes" to learn how to specify 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:

- "About the SYSAUX Tablespace" for information about the `SYSAUX` tablespace
- "Creating a Default Tablespace" 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"
- "Creating a Default Temporary Tablespace" 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"
- "Managing Application Workloads with Database Services"
- *Oracle Database Administrator's Guide* to learn about 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"
- " Starting Up and Shutting Down a CDB"

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

3

Creating a CDB: Basic Steps

After you plan your CDB, you can create it with a graphical tool or a SQL command.

Database creation prepares several operating system files to work together as CDB. You only need to create a CDB once, regardless of how many data files it has or how many instances access it. You can create a CDB to erase information in an existing CDB and create a new CDB with the same name and physical structure.

- Creating a CDB with DBCA
  Oracle Database Configuration Assistant (DBCA) is a tool for creating and configuring a CDB.
- Creating a Database with the CREATE DATABASE Statement
  Using the `CREATE DATABASE ... ENABLE PLUGGABLE DATABASE` SQL statement is a more manual approach to creating a database than using Oracle Database Configuration Assistant (DBCA). One advantage of using this statement over using DBCA is that you can create databases from within scripts.
- Considerations After Creating a CDB
  After you create a CDB, the instance is left running, and the database is open and available for normal database use. You may want to perform specific actions after creating a database.
- Database Data Dictionary Views
  You can query data dictionary views for information about your database content and structure.

See Also:

- Your platform-specific Oracle Real Application Clusters (Oracle RAC) installation guide for information about creating a database in an Oracle RAC environment
- *Oracle Clusterware Administration and Deployment Guide* for information on creating a database using Fleet Patching and Provisioning (it was called as Rapid Home Provisioning in the earlier database releases)

Creating a CDB with DBCA

Oracle Database Configuration Assistant (DBCA) is a tool for creating and configuring a CDB.

- About Creating a CDB with DBCA
  Oracle strongly recommends using the Database Configuration Assistant (DBCA) to create a CDB, because it is a more automated approach, and your CDB is ready to use when DBCA completes.

- After Creating a CDB
  After creation, a CDB consists of the root and the PDB seed.

About Creating a CDB with DBCA

Oracle strongly recommends using the Database Configuration Assistant (DBCA) to create a CDB, because it is a more automated approach, and your CDB is ready to use when DBCA completes.

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:

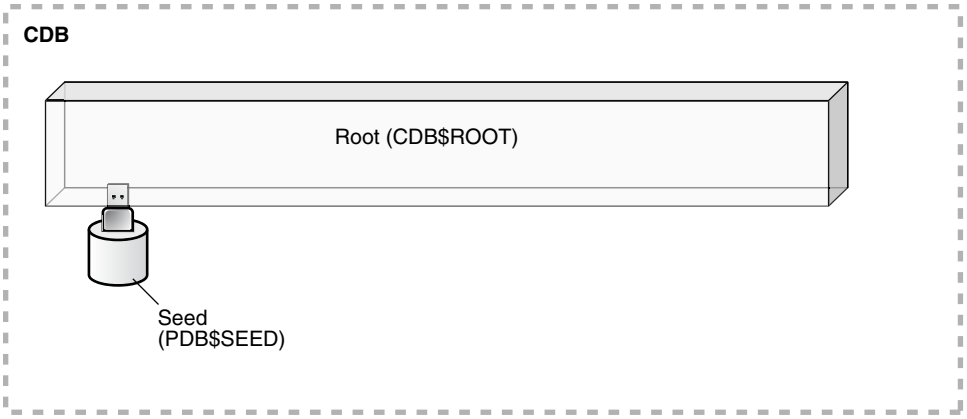
- *Oracle Database Administrator's Guide* to learn how to create a database with DBCA
- The DBCA online help

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.

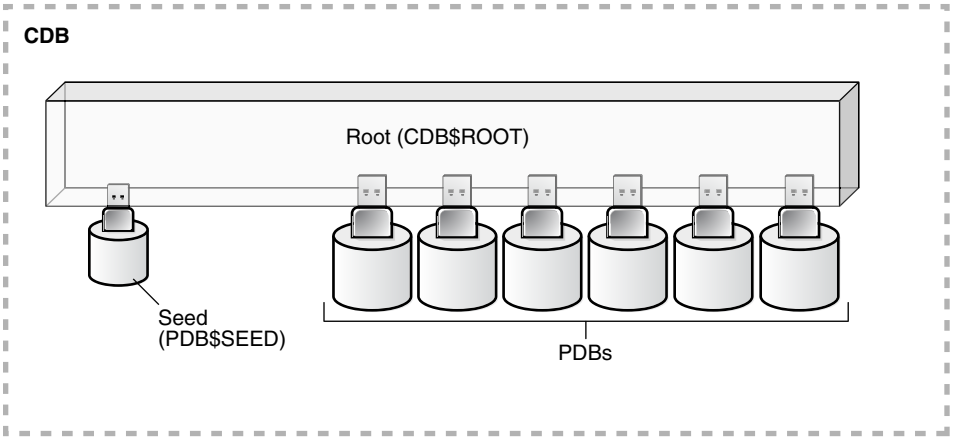
Figure 3-1 A Newly Created CDB



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.

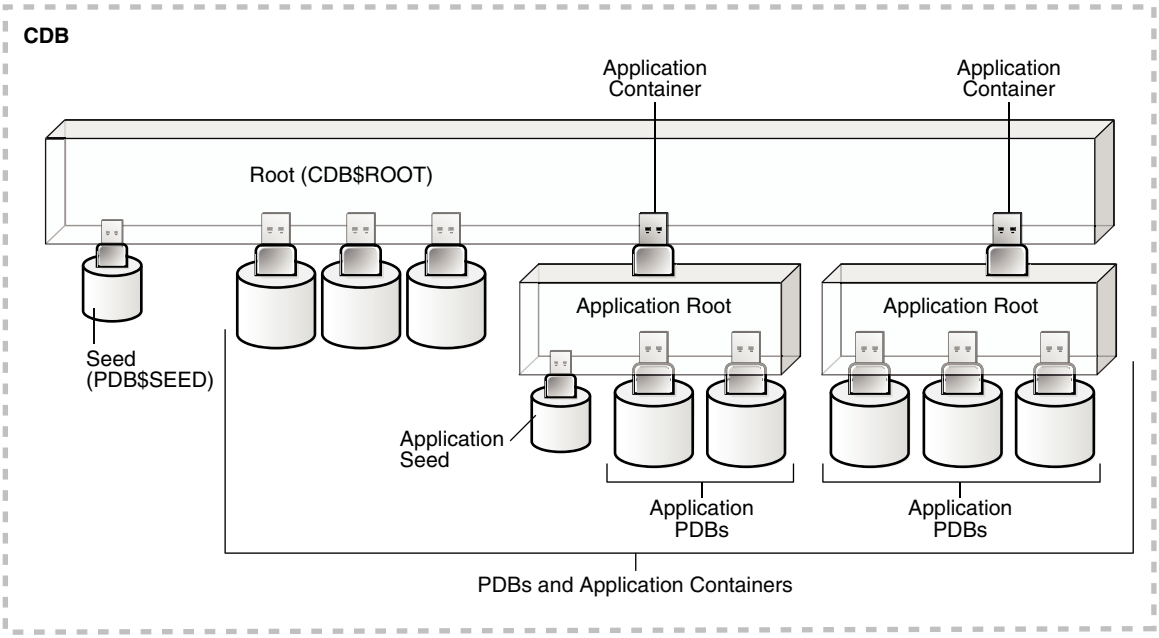
The following graphic shows a CDB with PDBs.

Figure 3-2 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 3-3 Application Containers in a 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

  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.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for information about RMAN

Creating a Database with the CREATE DATABASE Statement

Using the `CREATE DATABASE ... ENABLE PLUGGABLE DATABASE` SQL statement is a more manual approach to creating a database than using Oracle Database Configuration Assistant (DBCA). One advantage of using this statement over using DBCA is that you can create databases from within scripts.

- About CDB Creation with SQL Statements
  This section explains how to create a CDB manually, without using DBCA.
- Step 1: Specify an Instance Identifier (SID)
  The `ORACLE_SID` environment variable is used to distinguish this instance from other Oracle Database instances that you may create later and run concurrently on the same host computer.
- Step 2: Ensure That the Required Environment Variables Are Set
  Depending on your platform, before you can start SQL*Plus (as required in a later step), you may have to set environment variables, or at least verify that they are set properly.
- Step 3: Choose a Database Administrator Authentication Method
  You must be authenticated and granted appropriate system privileges in order to create a CDB.
- Step 4: Create the Initialization Parameter File
  When an Oracle instance starts, it reads an initialization parameter file.
- Step 5: (Windows Only) Create an Instance
  On the Windows platform, before you can connect to a database instance, you must manually create it if it does not already exist. The `ORADIM` command creates an instance by creating a new Windows service.
- Step 6: Connect to the Instance
  Start SQL*Plus and connect to your Oracle Database instance with the `SYSDBA` administrative privilege.
- Step 7: Create a Server Parameter File
  The server parameter file enables you to change initialization parameters with the `ALTER SYSTEM` command and persist the changes across a database shutdown and startup. You create the server parameter file from your edited text initialization file.
- Step 8: Start the Database Instance
  Start an instance without mounting a CDB.
- Step 9: Issue the CREATE DATABASE Statement
  To create the new database, use the `CREATE DATABASE` statement.
- Step 10: Run Scripts to Build Data Dictionary Views
  Run the scripts necessary to build data dictionary views, synonyms, and PL/SQL packages in the CDB root.
- Step 11: (Optional) Run Scripts to Install Additional Options
  You may want to run other scripts. The scripts that you run are determined by the features and options you choose to use or install.

- Step 12: Back Up the Database
  Take a full backup of the database to ensure that you have a complete set of files from which to recover if a media failure occurs.
- Step 13: (Optional) Enable Automatic Instance Startup
  You might want to configure the Oracle database instance to start automatically when its host computer restarts.

About CDB Creation with SQL Statements

This section explains how to create a CDB manually, without using DBCA.

> **Note:**
>
> "Specifying CREATE DATABASE Statement Clauses" provides more detailed information about the SQL clauses described in this chapter.

- About Oracle RAC and Oracle ASM
  The instructions in this section apply to *single-instance installations only*.
- About Enabling PDBs
  To create a CDB with the `CREATE DATABASE` command, the `ENABLE_PLUGGABLE_DATABASE` initialization parameter must be set to `true`.
- 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`.
- 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.
- 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.

> **See Also:**
>
> *Oracle Database Concepts* for information about the files in a CDB

About Oracle RAC and Oracle ASM

The instructions in this section apply to *single-instance installations only*.

See the Oracle Real Application Clusters (Oracle RAC) installation guide for your platform for instructions for creating an Oracle RAC database.

Note:

- *Single-instance* does not mean that only one Oracle instance can reside on a single host computer. In fact, multiple Oracle instances (and their associated databases) can run on a single host computer. A **single-instance database** is a database that is accessed by only one Oracle instance at a time, as opposed to an Oracle RAC database, which is accessed concurrently by multiple Oracle instances on multiple nodes.
- Starting in Oracle Database 12*c* Release 2 (12.2), read-only and read/write instances can coexist within a single Oracle RAC database. This configuration is useful for the scalability of parallel queries.

Tip:

If you are using Oracle Automatic Storage Management (Oracle ASM) to manage your disk storage, then you must start the Oracle ASM instance and configure your disk groups before performing these steps. See *Oracle Automatic Storage Management Administrator's Guide*.

See Also:

- *Oracle Real Application Clusters Administration and Deployment Guide*for more information on Oracle RAC
- *Oracle Clusterware Administration and Deployment Guide* for information about configuring read-only and read/write instances that coexist within a single Oracle RAC database

About Enabling PDBs

To create a CDB with the `CREATE DATABASE` command, the `ENABLE_PLUGGABLE_DATABASE` initialization parameter must be set to `true`.

The `CREATE DATABASE` command creates a CDB with the CDB root and `PDB$SEED`. You must create all other containers manually.

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.

- The ENABLE PLUGGABLE DATABASE SEED FILE_NAME_CONVERT Clause
  The `ENABLE PLUGGABLE DATABASE SEED FILE_NAME_CONVERT` clause of the `CREATE DATABASE` statement specifies how to generate the names of the `PDB$SEED` files using the names of the CDB root files.
- Oracle Managed Files
  When Oracle Managed Files is enabled, it can determine the names and locations of the `PDB$SEED` files.
- 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.

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

The ENABLE PLUGGABLE DATABASE SEED FILE_NAME_CONVERT Clause

The `ENABLE PLUGGABLE DATABASE 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 3-1 SEED FILE_NAME_CONVERT Clause

This `ENABLE PLUGGABLE DATABASE 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.

```
ENABLE PLUGGABLE DATABASE SEED FILE_NAME_CONVERT = ('/oracle/dbs/', '/oracle/
pdbseed/')
```

See Also:

Oracle Database SQL Language Reference for the syntax of the `ENABLE PLUGGABLE DATABASE 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 3-2 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`.

Step 1: Specify an Instance Identifier (SID)

The `ORACLE_SID` environment variable is used to distinguish this instance from other Oracle Database instances that you may create later and run concurrently on the same host computer.

1. Decide on a unique Oracle system identifier (SID) for your instance.
2. Open a command window.

> **Note:**
>
> Use this command window for the subsequent steps.

3. Set the `ORACLE_SID` environment variable.

Restrictions related to the valid characters in an `ORACLE_SID` are platform-specific. On some platforms, the SID is case-sensitive.

> **Note:**
>
> It is common practice to set the SID to be equal to the database name. The maximum number of characters for the database name is eight.

The following example for UNIX and Linux operating systems sets the SID for the instance that you will connect to in Step 6: Connect to the Instance:

- Bourne, Bash, or Korn shell:

```
ORACLE_SID=mynewdb
export ORACLE_SID
```

- C shell:

```
setenv ORACLE_SID mynewdb
```

The following example sets the SID for the Windows operating system:

```
set ORACLE_SID=mynewdb
```

See Also:

- *Oracle Database Concepts* for background information about the Oracle instance
- *Oracle Database Reference* to learn more about the `DB_NAME` initialization parameter

Step 2: Ensure That the Required Environment Variables Are Set

Depending on your platform, before you can start SQL*Plus (as required in a later step), you may have to set environment variables, or at least verify that they are set properly.

- Set required environment variables.

For example, on most platforms, `ORACLE_SID` and `ORACLE_HOME` must be set. In addition, it is advisable to set the `PATH` variable to include the *ORACLE_HOME*/bin directory. On the UNIX and Linux platforms, you must set these environment variables manually. On the Windows platform, OUI automatically assigns values to `ORACLE_HOME` and `ORACLE_SID` in the Windows registry. If you did not create a database upon installation, OUI does not set `ORACLE_SID` in the registry, and you will have to set the `ORACLE_SID` environment variable when you create your database later.

Step 3: Choose a Database Administrator Authentication Method

You must be authenticated and granted appropriate system privileges in order to create a CDB.

- Decide on an authentication method.

You can be authenticated as an administrator with the required privileges in the following ways:

- With a password file
- With operating system authentication

To be authenticated with a password file, create the password file. To be authenticated with operating system authentication, ensure that you log in to the host computer with a user account that is a member of the appropriate operating system user group. On the UNIX and Linux platforms, for example, this is typically the `dba` user group. On the Windows platform, the user installing the Oracle software is automatically placed in the required user group.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about password files and operating system authentication

Step 4: Create the Initialization Parameter File

When an Oracle instance starts, it reads an initialization parameter file.

The parameter file can be a text file, which can be created and modified with a text editor, or a binary file, which is created and dynamically modified by the database. The binary file, which is preferred, is called a **server parameter file**. In this step, you create a text initialization parameter file. In a later step, you create a server parameter file from the text file.

- Create the initialization parameter file.

One way to create the text initialization parameter file is to edit the sample presented in "Sample Initialization Parameter File".

If you create the initialization parameter file manually, ensure that it contains at least the parameters listed in the following table. All other parameters not listed have default values.

Table 3-1 Recommended Minimum Initialization Parameters

Parameter Name	Mandatory	Notes
`DB_NAME`	Yes	Database identifier for the name of the CDB root. Must correspond to the value used in the `CREATE DATABASE` statement. Maximum 8 characters. It is common practice to set the SID to the name of the CDB 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*.
`DB_DOMAIN`	Yes	Specifies the network domain where the database is created. 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 CDB 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.
`ENABLE_PLUGGABLE_DATABASE`	Yes	Specifies that the database is a CDB. Must be set to `TRUE`.
`CONTROL_FILES`	No	Strongly recommended. If not provided, then the database instance creates one control file in the same location as the initialization parameter file. Providing this parameter enables you to multiplex control files.

Table 3-1 (Cont.) Recommended Minimum Initialization Parameters

Parameter Name	Mandatory	Notes
MEMORY_TARGET	No	Sets the total amount of memory used by the instance and enables automatic memory management. You can choose other initialization parameters instead of this one for more manual control of memory usage.
DB_CREATE_FILE_DEST	No	Defines the base directory for Oracle Managed Files that the CDB creates and automatically names. To use Oracle Managed Files, the initialization parameter DB_CREATE_FILE_DEST must be set.

For convenience, store your initialization parameter file in the Oracle Database default location, using the default file name. Then when you start your database, it will not be necessary to specify the PFILE clause of the STARTUP command, because Oracle Database automatically looks in the default location for the initialization parameter file.

For more information about initialization parameters and the initialization parameter file, including the default name and location of the initialization parameter file for your platform, see "About Initialization Parameters and Initialization Parameter Files".

See Also:

- "Specifying Initialization Parameters"
- *Oracle Database Reference* for details on all initialization parameters

Step 5: (Windows Only) Create an Instance

On the Windows platform, before you can connect to a database instance, you must manually create it if it does not already exist. The ORADIM command creates an instance by creating a new Windows service.

To create a database instance:

- Enter the following command at a Windows command prompt:

  ```
  oradim -NEW -SID sid -STARTMODE MANUAL -PFILE file
  ```

  Replace the following placeholders with appropriate values:

  - *sid* - The desired SID (for example mynewdb)
  - *file* - The full path to the text initialization parameter file

> **Caution:**
>
> Do not set the -STARTMODE argument to AUTO at this point, because this causes the new instance to start and attempt to mount the database, which does not exist yet. You can change this parameter to AUTO, if desired, in Step 13: (Optional) Enable Automatic Instance Startup.

Most Oracle Database services log on to the system using the privileges of the Oracle Home User. The service runs with the privileges of this user. The ORADIM command prompts you for the password to this user account. You can specify other options using ORADIM.

> **See Also:**
>
> *Oracle Database Platform Guide for Microsoft Windows* for more information on the ORADIM command and the Oracle Home User

Step 6: Connect to the Instance

Start SQL*Plus and connect to your Oracle Database instance with the SYSDBA administrative privilege.

- To authenticate with a password file, enter the following commands, and then enter the SYS password when prompted:

```
$ sqlplus /nolog
SQL> CONNECT SYS AS SYSDBA
```

- To authenticate with operating system authentication, enter the following commands:

```
$ sqlplus /nolog
SQL> CONNECT / AS SYSDBA
```

SQL*Plus outputs the following message:

```
Connected to an idle instance.
```

> **Note:**
>
> SQL*Plus may output a message similar to the following:
>
> ```
> Connected to:
> Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
> Version 19.1.0.0.0
> ```
>
> If so, the instance is already started. You may have connected to the wrong instance. Exit SQL*Plus with the EXIT command, check that ORACLE_SID is set properly, and repeat this step.

Step 7: Create a Server Parameter File

The server parameter file enables you to change initialization parameters with the `ALTER SYSTEM` command and persist the changes across a database shutdown and startup. You create the server parameter file from your edited text initialization file.

- Run the following SQL*Plus command:

```
CREATE SPFILE FROM PFILE;
```

This SQL*Plus command reads the text initialization parameter file (PFILE) with the default name from the default location, creates a server parameter file (SPFILE) from the text initialization parameter file, and writes the SPFILE to the default location with the default SPFILE name.

You can also supply the file name and path for both the PFILE and SPFILE if you are not using default names and locations.

Tip:

The CDB must be restarted before the server parameter file takes effect.

Note:

Although creating a server parameter file is optional at this point, it is recommended. If you do not create a server parameter file, the instance continues to read the text initialization parameter file whenever it starts.

Important—If you are using Oracle Managed Files and your initialization parameter file does not contain the `CONTROL_FILES` parameter, then you must create a server parameter file now so the database can save the names and locations of the control files that it creates during the `CREATE DATABASE` statement. See "Specifying Oracle Managed Files at Database Creation" for more information.

See Also:

- "Managing Initialization Parameters Using a Server Parameter File"
- *Oracle Database SQL Language Reference* for more information on the `CREATE SPILE` command

Step 8: Start the Database Instance

Start an instance without mounting a CDB.

- Run the `STARTUP` command with the `NOMOUNT` clause.

Typically, you do this only during CDB creation or while performing maintenance on the database. In this example, because the initialization parameter file or server parameter file is stored in the default location, you are not required to specify the `PFILE` clause:

```
STARTUP NOMOUNT
```

At this point, the instance memory is allocated and its processes are started. The CDB itself does not yet exist.

> **See Also:**
>
> - " Starting Up and Shutting Down a CDB" for information about using the `STARTUP` command
> - "Managing Initialization Parameters Using a Server Parameter File"

Step 9: Issue the CREATE DATABASE Statement

To create the new database, use the `CREATE DATABASE` statement.

- Run the `CREATE DATABASE` statement with the `ENABLE PLUGGABLE DATABASE` clause.

The following topics show sample statements, using Oracle Managed Files and user-specified files.

- Creating a CDB Without Using Oracle Managed Files: Example
  The following statement creates a CDB named `newcdb`. This name must agree with the `DB_NAME` parameter in the initialization parameter file.
- 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.

Creating a CDB Without Using Oracle Managed Files: Example

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 `ENABLE_PLUGGABLE_DATABASE` initialization parameter is set to `true`.

- 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 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.

  Tip:

  If your `CREATE DATABASE` statement fails, and if you did not complete Step 7, then ensure that there is not a pre-existing server parameter file (SPFILE) for this database 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. See "Managing Initialization Parameters Using a Server Parameter File" 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 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 CDB 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 Oracle recommends 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 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 SEED` clause is required for the *`tablespace_datafile`* clauses.

> **Note:**
>
> If you do not specify the `SYSTEM` and `SYSAUX` clauses, which are optional, then the `ENABLE PLUGGABLE DATABASE SEED` clause is not required.

- The `SYSTEM DATAFILES` clause specifies attributes of the PDB seed's `SYSTEM` tablespace data files that differ from the root's.
- The `SYSAUX DATAFILES` clause specifies attributes of the PDB seed's `SYSAUX` tablespace data files that differ from the root's.

Step 10: Run Scripts to Build Data Dictionary Views

Run the scripts necessary to build data dictionary views, synonyms, and PL/SQL packages in the CDB root.

Perform these actions by running the supplied `catcdb.sql` script, which installs all components required by a CDB. The at-sign (`@`) is shorthand for the command that runs a SQL*Plus script. The question mark (`?`) is a SQL*Plus variable indicating the Oracle home directory.

Before you run the `catcdb.sql` SQL script, ensure that you set the following environment variables:

- `CATCDB_SYS_PASSWD` - administrator password (SYS)
- `CATCDB_SYSTEM_PASSWD` - administrator password (SYSTEM)
- `CATCDB_TEMP` - temporary tablespace name

Follow these steps:

1. Run the `catcdb.sql` SQL script.

   Enter the following in SQL*Plus to run the script:

   ```
   @?/rdbms/admin/catcdb.sql
   ```

2. 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
```

3. 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
```

Step 11: (Optional) Run Scripts to Install Additional Options

You may want to run other scripts. The scripts that you run are determined by the features and options you choose to use or install.

- Run scripts to install additional options.

Many of the scripts available to you are described in the *Oracle Database Reference*.

If you plan to install other Oracle products to work with this database, then see the installation instructions for those products. Some products require you to create additional data dictionary tables. Usually, command files are provided to create and load these tables into the database data dictionary.

See your Oracle documentation for the specific products that you plan to install for installation and administration instructions.

Step 12: Back Up the Database

Take a full backup of the database to ensure that you have a complete set of files from which to recover if a media failure occurs.

- Back up the CDB.

For information on backing up a CDB, see *Oracle Database Backup and Recovery User's Guide*.

Step 13: (Optional) Enable Automatic Instance Startup

You might want to configure the Oracle database instance to start automatically when its host computer restarts.

- Configure the Oracle instance to start automatically when its host computer restarts.

See your operating system documentation for instructions. For example, on Windows, use the following command to configure the database service to start the instance upon computer restart:

```
ORADIM -EDIT -SID sid -STARTMODE AUTO -SRVCSTART SYSTEM [-SPFILE]
```

You must use the `-SPFILE` argument if you want the instance to read an SPFILE upon automatic restart.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* to learn more about Oracle Restart
> - *Oracle Database Platform Guide for Microsoft Windows* for more information on the `ORADIM` command.

Considerations After Creating a CDB

After you create a CDB, the instance is left running, and the database is open and available for normal database use. You may want to perform specific actions after creating a database.

- Database Security
  You can use the default Oracle Database features to configure security in several areas for your Oracle database.
- Transparent Data Encryption
  Transparent Data Encryption enables encryption of database columns before storing them in the data file, or enables encryption of entire tablespaces.
- A Secure External Password Store
  Consider using client-side Oracle wallets to reduce exposing authentication and signing credentials over networks.
- Transaction Guard and Application Continuity
  Transaction Guard uses a logical transaction ID to prevent the possibility of a client application submitting duplicate transactions after a recoverable error. Application Continuity enables the replay, in a nondisruptive and rapid manner, of a request against the database after a recoverable error that makes the database session unavailable.
- File System Server Support in the Database
  An Oracle database can be configured to store file system objects and access them from any NFS client. The database stores both the files and their metadata. The database responds to file system requests from the NFS daemon process in the operating system (OS) kernel.
- The Oracle Database Sample Schemas
  Oracle Database includes sample schemas that help you to become familiar with Oracle Database functionality. Some Oracle Database documentation and training materials use the sample schemas in examples.

Database Security

You can use the default Oracle Database features to configure security in several areas for your Oracle database.

The following are some of the areas in which you can configure security for your database:

- **User accounts:** When you create user accounts, you can secure them in a variety of ways. You can also create password profiles to better secure password policies for your site.
- **Authentication methods:** Oracle Database provides several ways to configure authentication for users and database administrators. For example, you can authenticate users on the database level, from the operating system, and on the network.
- **Privileges and roles**: You can use privileges and roles to restrict user access to data.

> **Note:**
>
> - A newly created database has at least three user accounts that are important for administering your database: `SYS`, `SYSTEM`, and `SYSMAN`. Additional administrative accounts are provided that should be used only by authorized users.
> - To prevent unauthorized access and protect the integrity of your database, it is important that a new password is specified to the `SYS` user when the database is created.
> - Most Oracle Database supplied user accounts, except `SYS` and sample schemas are *schema only* accounts, that is, these accounts are created without passwords. You can assign passwords to these accounts whenever you want them to be authenticated, but Oracle recommends that for better security, you should change these accounts back to schema only accounts, when you do not need to authenticate them anymore.
>
> To find the status of an account, query the `ACCOUNT_STATUS` column of the `DBA_USERS` data dictionary view. If the account is schema only, then the status is `NONE`.

See Also:

- *Oracle Database Security Guide* for a complete list of predefined user accounts created with each new Oracle Database installation
- *Oracle Database Security Guide* to learn how to add new users and change passwords
- *Oracle Database SQL Language Reference* for the syntax of the `ALTER USER` statement used for unlocking database user accounts
- *Oracle Database Enterprise User Security Administrator's Guide*for information about Oracle Identity Management
- *Oracle Database Security Guide* for security guidelines for configuring a database

Transparent Data Encryption

Transparent Data Encryption enables encryption of database columns before storing them in the data file, or enables encryption of entire tablespaces.

If users attempt to circumvent the database access control mechanisms by looking inside data files directly with operating system tools, Transparent Data Encryption prevents such users from viewing sensitive information.

Users who have the `CREATE TABLE` privilege can choose one or more columns in a table to be encrypted. The data is encrypted in the data files. Database users with appropriate privileges can view the data in unencrypted format.

See Also:

- *Oracle Database Administrator's Guide* to learn about encrypting columns
- *Oracle Database Administrator's Guide* to learn about encrypted tablespaces
- *Oracle Database Advanced Security Guide* to learn more about Transparent Data Encryption

A Secure External Password Store

Consider using client-side Oracle wallets to reduce exposing authentication and signing credentials over networks.

For large-scale deployments where applications use password credentials to connect to databases, it is possible to store such credentials in a client-side Oracle wallet. An Oracle wallet is a secure software container that is used to store authentication and signing credentials.

Storing database password credentials in a client-side Oracle wallet eliminates the need to embed usernames and passwords in application code, batch jobs, or scripts. Client-side storage reduces the risk of exposing passwords in the clear in scripts and application code. It

also simplifies maintenance, because you need not change your code each time usernames and passwords change. In addition, not having to change application code also makes it easier to enforce password management policies for these user accounts.

When you configure a client to use the external password store, applications can use the following syntax to connect to databases that use password authentication:

```
CONNECT /@database_alias
```

You need not specify database login credentials in this `CONNECT` command. Instead your system looks for database login credentials in the client wallet.

See Also:

- *Oracle Database Security Guide*
- *Oracle Database Enterprise User Security Administrator's Guide*

Transaction Guard and Application Continuity

Transaction Guard uses a logical transaction ID to prevent the possibility of a client application submitting duplicate transactions after a recoverable error. Application Continuity enables the replay, in a nondisruptive and rapid manner, of a request against the database after a recoverable error that makes the database session unavailable.

Transaction Guard is a reliable protocol and API that application developers can use to provide a known outcome for the last open transaction on a database session that becomes unavailable. After an outage, the commit message that is sent from the database to the client is not durable. If the connection breaks between an application (the client) and an Oracle database (the server), then the client receives an error message indicating that the communication failed. This error message does not inform the client about the success or failure of commit operations or procedure calls.

Transaction Guard uses a concept called the logical transaction identifier (LTXID), a globally unique identifier that identifies the transaction from the application's perspective. When a recoverable outage occurs, the application uses the LTXID to determine the outcome of the transaction. This outcome can be returned to the client instead of the ambiguous communication error. The user can decide whether to resubmit the transaction. The application also can be coded to resubmit the transaction if the states are correct.

Application Continuity masks outages from end users and applications by recovering the in-flight database sessions following recoverable outages, for both unplanned and planned outages. After a successful replay, the application can continue using a new session where the original database session left off. Application Continuity performs this recovery so that the outage appears to the application as a delayed execution.

Application Continuity is enabled at the service level and is invoked for outages that are recoverable. These outages typically are related to underlying software, foreground, hardware, communications, network, or storage layers. Application Continuity supports queries, `ALTER SESSION` statements, Java and OCI APIs, PL/SQL,

DDL, and the last uncommitted transaction before the failure. Application Continuity determines whether the last in-flight transaction committed or not, and whether the last user call completed or not, using Transaction Guard.

See Also:

- *Oracle Database Concepts* for a conceptual overview of Transaction Guard and Application Continuity
- *Oracle Database Development Guide* for complete information about Transaction Guard and Application Continuity

File System Server Support in the Database

An Oracle database can be configured to store file system objects and access them from any NFS client. The database stores both the files and their metadata. The database responds to file system requests from the NFS daemon process in the operating system (OS) kernel.

When you configure the Oracle File System (OFS) server in a database and create a file system, you can store unstructured data, such as emails, videos, audio files, credit card bills, documents, photo images, and so on, inside the database. You can manipulate and manage these unstructured objects without using SQL. Instead, you can use operating system utilities for NFS support.

To enable NFS access in the database, set the `OFS_THREADS` initialization parameter to configure a sufficient number of OFS threads to process the NFS requests. The `OFS_THREADS` initialization parameter controls the number of OFS threads to create when the first file system is mounted with the database. The number of threads specified by the `OFS_THREADS` parameter are created only once for the database instance and subsequent file systems do not create any additional threads. The default value of the `OFS_THREADS` initialization parameter is 4. At database startup, OFSD background process is the sole OFS process that is spawned by the database server.

You can use the `DBMS_FS` package to create a file system in the database using a specified database object. You can also use this package to mount and unmount a specified file system.

See Also:

- *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about the Oracle File System (OFS)
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_FS` package

The Oracle Database Sample Schemas

Oracle Database includes sample schemas that help you to become familiar with Oracle Database functionality. Some Oracle Database documentation and training materials use the sample schemas in examples.

The schemas and installation instructions are described in detail in *Oracle Database Sample Schemas*.

> **Note:**
>
> Oracle strongly recommends that you do not install the sample schemas in a production database.

Database Data Dictionary Views

You can query data dictionary views for information about your database content and structure.

You can view information about your database content and structure using the following views:

View	Description
DATABASE_PROPERTIES	Displays permanent database properties
GLOBAL_NAME	Displays the global database name
V$DATABASE	Contains database information from the control file

4

Creating a CDB: Advanced Topics

This chapter covers creating a CDB in greater detail.

- Specifying CREATE DATABASE Statement Clauses
  When you execute a `CREATE DATABASE` statement, Oracle Database performs several operations. The actual operations performed depend on the clauses that you specify in the `CREATE DATABASE` statement and the initialization parameters that you have set.
- Specifying Initialization Parameters
  You can add or edit basic initialization parameters before you create your new database.
- Managing Initialization Parameters Using a Server Parameter File
  Initialization parameters for the Oracle Database have traditionally been stored in a text initialization parameter file. For better manageability, you can choose to maintain initialization parameters in a binary server parameter file that is persistent across database startup and shutdown.
- Managing Application Workloads with Database Services
  A database service is a named representation of one or more database instances. Services enable you to group database workloads and route a particular work request to an appropriate instance.
- Managing Standard Edition High Availability for Oracle Databases
  The Standard Edition High Availability feature provides protection against unplanned outages for Oracle Database Standard Edition 2 single instance databases using Oracle Clusterware.
- Cloning a Database
  This section describes various methods of cloning an Oracle database.
- Dropping a Database
  Dropping a CDB involves removing its data files, online redo logs, control files, and initialization parameter files.

Specifying CREATE DATABASE Statement Clauses

When you execute a `CREATE DATABASE` statement, Oracle Database performs several operations. The actual operations performed depend on the clauses that you specify in the `CREATE DATABASE` statement and the initialization parameters that you have set.

- About CREATE DATABASE Statement Clauses
  You can use the `CREATE DATABASE` clauses to simplify the creation and management of your database.
- Protecting Your Database: Specifying Passwords for SYS and SYSTEM Users
  To protect your database, specify passwords for `SYS` and `SYSTEM` users.
- Creating a Locally Managed SYSTEM Tablespace
  During database creation, create a locally managed SYSTEM tablespace. A locally managed tablespace uses a bitmap stored in each data file to manage the extents.

- Specify Data File Attributes for the SYSAUX Tablespace
  The SYSAUX tablespace is created by default, but you can specify its data file attributes during database creation.
- Using Automatic Undo Management: Creating an Undo Tablespace
  Automatic undo management uses an undo tablespace.
- Creating a Default Tablespace
  Oracle strongly recommends that you create a default tablespace. Oracle Database assigns to this tablespace any non-`SYSTEM` users for whom you do not explicitly specify a different tablespace.
- Creating a Default Temporary Tablespace
  When you create a default temporary tablespace, Oracle Database assigns it as the temporary tablespace for users who are not explicitly assigned a temporary tablespace.
- Specifying Oracle Managed Files at Database Creation
  You can minimize the number of clauses and parameters that you specify in your `CREATE DATABASE` statement by using the Oracle Managed Files feature.
- Supporting Bigfile Tablespaces During Database Creation
  Oracle Database simplifies management of tablespaces and enables support for extremely large databases by letting you create **bigfile tablespaces**.
- Specifying the Database Time Zone and Time Zone File
  Oracle Database datetime and interval data types and time zone support make it possible to store consistent information about the time of events and transactions.
- Specifying FORCE LOGGING Mode
  Some data definition language statements (such as `CREATE TABLE`) allow the `NOLOGGING` clause, which causes some database operations not to generate redo records in the database redo log. The `NOLOGGING` setting can speed up operations that can be easily recovered outside of the database recovery mechanisms, but it can negatively affect media recovery and standby databases.

About CREATE DATABASE Statement Clauses

You can use the `CREATE DATABASE` clauses to simplify the creation and management of your database.

When you execute a `CREATE DATABASE` statement, Oracle Database performs at least these operations:

- Creates the data files for the database
- Creates the control files for the database
- Creates the online redo logs for the database and establishes the `ARCHIVELOG` mode
- Creates the `SYSTEM` tablespace
- Creates the `SYSAUX` tablespace
- Creates the data dictionary
- Sets the character set that stores data in the database
- Sets the database time zone
- Mounts and opens the database for use

Protecting Your Database: Specifying Passwords for SYS and SYSTEM Users

To protect your database, specify passwords for `SYS` and `SYSTEM` users.

- In the `CREATE DATABASE` statement, include clauses that specify the password for users `SYS` and `SYSTEM`.

The clauses of the `CREATE DATABASE` statement used for specifying the passwords for users `SYS` and `SYSTEM` are:

- `USER SYS IDENTIFIED BY` *`password`*
- `USER SYSTEM IDENTIFIED BY` *`password`*

When choosing a password, keep in mind that passwords are case-sensitive. Also, there may be password formatting requirements for your database.

> **See Also:**
>
> *Oracle Database Security Guide* for information about the Oracle guidelines for creating secure passwords

Creating a Locally Managed SYSTEM Tablespace

During database creation, create a locally managed SYSTEM tablespace. A locally managed tablespace uses a bitmap stored in each data file to manage the extents.

- Specify the `EXTENT MANAGEMENT LOCAL` clause in the `CREATE DATABASE` statement to create a locally managed `SYSTEM` tablespace.

If you do not specify the `EXTENT MANAGEMENT LOCAL` clause, then by default the database creates a dictionary-managed `SYSTEM` tablespace. Dictionary-managed tablespaces are deprecated.

If you create your database with a locally managed `SYSTEM` tablespace, and if you are not using Oracle Managed Files, then ensure that the following conditions are met:

- You specify the `DEFAULT TEMPORARY TABLESPACE` clause in the `CREATE DATABASE` statement.
- You include the `UNDO TABLESPACE` clause in the `CREATE DATABASE` statement.

See Also:

- *Oracle Database SQL Language Reference* for more specific information about the use of the `DEFAULT TEMPORARY TABLESPACE` and `UNDO TABLESPACE` clauses when `EXTENT MANAGEMENT LOCAL` is specified for the `SYSTEM` tablespace
- *Oracle Database Administrator's Guide* to learn about locally managed tablespaces

Specify Data File Attributes for the SYSAUX Tablespace

The SYSAUX tablespace is created by default, but you can specify its data file attributes during database creation.

To specify data file attributes for the `SYSAUX` tablespace:

- Include the `SYSAUX DATAFILE` clause in the `CREATE DATABASE` statement.

If you include a `DATAFILE` clause for the `SYSTEM` tablespace, then you must specify the `SYSAUX DATAFILE` clause as well, or the `CREATE DATABASE` statement will fail. This requirement does not exist if the Oracle Managed Files feature is enabled (see "Specifying Oracle Managed Files at Database Creation").

- About the SYSAUX Tablespace
  The `SYSAUX` tablespace is always created at database creation.

About the SYSAUX Tablespace

The `SYSAUX` tablespace is always created at database creation.

The `SYSAUX` tablespace serves as an auxiliary tablespace to the `SYSTEM` tablespace. Because it is the default tablespace for many Oracle Database features and products that previously required their own tablespaces, it reduces the number of tablespaces required by the database. It also reduces the load on the `SYSTEM` tablespace.

You can specify only data file attributes for the `SYSAUX` tablespace, using the `SYSAUX DATAFILE` clause in the `CREATE DATABASE` statement. Mandatory attributes of the `SYSAUX` tablespace are set by Oracle Database and include:

- `PERMANENT`
- `READ WRITE`
- `EXTENT MANAGEMENT LOCAL`
- `SEGMENT SPACE MANAGEMENT AUTO`

You cannot alter these attributes with an `ALTER TABLESPACE` statement, and any attempt to do so will result in an error. You cannot drop or rename the `SYSAUX` tablespace.

The size of the `SYSAUX` tablespace is determined by the size of the database components that occupy `SYSAUX`. You can view a list of these components by querying the `V$SYSAUX_OCCUPANTS` view. Based on the initial sizes of these components, the

SYSAUX tablespace must be at least 400 MB at the time of database creation. The space requirements of the SYSAUX tablespace will increase after the database is fully deployed, depending on the nature of its use and workload.

The SYSAUX tablespace has the same security attributes as the SYSTEM tablespace.

> **See Also:**
>
> *Oracle Database Administrator's Guide* to learn how to manage the SYSAUX tablespace

Using Automatic Undo Management: Creating an Undo Tablespace

Automatic undo management uses an undo tablespace.

- To enable automatic undo management, set the UNDO_MANAGEMENT initialization parameter to AUTO in your initialization parameter file. Alternatively, omit this parameter so that the database defaults to automatic undo management.

In this mode, undo data is stored in an undo tablespace and is managed by Oracle Database. To define and name the undo tablespace yourself, you must include the UNDO TABLESPACE clause in the CREATE DATABASE statement at database creation time. If you omit this clause, and automatic undo management is enabled, then the database creates a default undo tablespace named SYS_UNDOTBS.

> **Note:**
>
> If you decide to define the undo tablespace yourself, then ensure that its block size matches the highest data file block size for the database.

> **See Also:**
>
> - "Specifying the Method of Undo Space Management"
> - *Oracle Database Administrator's Guide* for information about the creation and use of undo tablespaces

Creating a Default Tablespace

Oracle strongly recommends that you create a default tablespace. Oracle Database assigns to this tablespace any non-SYSTEM users for whom you do not explicitly specify a different tablespace.

To specify a default tablespace for the database:

- Include the DEFAULT TABLESPACE clause in the CREATE DATABASE statement

If you do not specify the `DEFAULT TABLESPACE` clause, then the `SYSTEM` tablespace is the default tablespace for non-`SYSTEM` users.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the syntax of the `DEFAULT TABLESPACE` clause of `CREATE DATABASE` and `ALTER DATABASE`

Creating a Default Temporary Tablespace

When you create a default temporary tablespace, Oracle Database assigns it as the temporary tablespace for users who are not explicitly assigned a temporary tablespace.

To create a default temporary tablespace for the CDB:

- Include the `DEFAULT TEMPORARY TABLESPACE` clause in the `CREATE DATABASE` statement.

You can explicitly assign a temporary tablespace or tablespace group to a user in the `CREATE USER` statement. However, if you do not do so, and if no default temporary tablespace has been specified for the database, then by default these users are assigned the `SYSTEM` tablespace as their temporary tablespace. It is not good practice to store temporary data in the `SYSTEM` tablespace, and it is cumbersome to assign every user a temporary tablespace individually. Therefore, Oracle recommends that you use the `DEFAULT TEMPORARY TABLESPACE` clause of `CREATE DATABASE`.

> **Note:**
>
> When you specify a locally managed `SYSTEM` tablespace, the `SYSTEM` tablespace *cannot* be used as a temporary tablespace. In this case you must create a default temporary tablespace. This behavior is explained in "Creating a Locally Managed SYSTEM Tablespace".

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for the syntax of the `DEFAULT TEMPORARY TABLESPACE` clause of `CREATE DATABASE` and `ALTER DATABASE`
> - *Oracle Database Administrator's Guide* for information about creating and using temporary tablespaces

Specifying Oracle Managed Files at Database Creation

You can minimize the number of clauses and parameters that you specify in your `CREATE DATABASE` statement by using the Oracle Managed Files feature.

- Specify either a directory or Oracle Automatic Storage Management (Oracle ASM) disk group in which your files are created and managed by Oracle Database.

By including any of the initialization parameters `DB_CREATE_FILE_DEST`, `DB_CREATE_ONLINE_LOG_DEST_n`, or `DB_RECOVERY_FILE_DEST` in your initialization parameter file, you instruct Oracle Database to create and manage the underlying operating system files of your database. Oracle Database will automatically create and manage the operating system files for the following database structures, depending on which initialization parameters you specify and how you specify clauses in your `CREATE DATABASE` statement:

- Tablespaces and their data files
- Temporary tablespaces and their temp files
- Control files
- Online redo log files
- Archived redo log files
- Flashback logs
- Block change tracking files
- RMAN backups

The following `CREATE DATABASE` statement shows briefly how the Oracle Managed Files feature works, assuming you have specified required initialization parameters:

```
CREATE DATABASE mynewdb
     USER SYS IDENTIFIED BY sys_password
     USER SYSTEM IDENTIFIED BY system_password
     EXTENT MANAGEMENT LOCAL
     UNDO TABLESPACE undotbs1
     DEFAULT TEMPORARY TABLESPACE tempts1
     DEFAULT TABLESPACE users
     ENABLE PLUGGABLE DATABASE
       SEED
       SYSTEM DATAFILES SIZE 125M AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
       SYSAUX DATAFILES SIZE 100M;
```

- The `SYSTEM` tablespace is created as a locally managed tablespace. Without the `EXTENT MANAGEMENT LOCAL` clause, the `SYSTEM` tablespace is created as dictionary managed, which is not recommended.
- No `DATAFILE` clause is specified, so the database creates an Oracle managed data file for the `SYSTEM` tablespace.
- No `LOGFILE` clauses are included, so the database creates two Oracle managed redo log file groups.
- No `SYSAUX DATAFILE` is included, so the database creates an Oracle managed data file for the `SYSAUX` tablespace.

- No `DATAFILE` subclause is specified for the `UNDO TABLESPACE` and `DEFAULT TABLESPACE` clauses, so the database creates an Oracle managed data file for each of these tablespaces.
- No `TEMPFILE` subclause is specified for the `DEFAULT TEMPORARY TABLESPACE` clause, so the database creates an Oracle managed temp file.
- If no `CONTROL_FILES` initialization parameter is specified in the initialization parameter file, then the database also creates an Oracle managed control file.
- If you are using a server parameter file, then the database automatically sets the appropriate initialization parameters.

> **See Also:**
>
> – "Specifying a Fast Recovery Area" for information about setting initialization parameters that create a Fast Recovery Area
> – *Oracle Database Administrator's Guide* for information about the Oracle Managed Files feature and how to use it
> – *Oracle Automatic Storage Management Administrator's Guide* for information about Automatic Storage Management

Supporting Bigfile Tablespaces During Database Creation

Oracle Database simplifies management of tablespaces and enables support for extremely large databases by letting you create **bigfile tablespaces**.

Bigfile tablespaces can contain only one file, but that file can have up to 4G blocks. The maximum number of data files in an Oracle Database is limited (usually to 64K files). Therefore, bigfile tablespaces can significantly enhance the storage capacity of an Oracle Database.

This section discusses the clauses of the `CREATE DATABASE` statement that let you include support for bigfile tablespaces.

- Specifying the Default Tablespace Type
  The `SET DEFAULT...TABLESPACE` clause of the `CREATE DATABASE` statement determines the default type of tablespace for this database in subsequent `CREATE TABLESPACE` statements.
- Overriding the Default Tablespace Type
  The `SYSTEM` and `SYSAUX` tablespaces are always created with the default tablespace type. However, you optionally can explicitly override the default tablespace type for the `UNDO` and `DEFAULT TEMPORARY` tablespace during the `CREATE DATABASE` operation.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for more information about bigfile tablespaces

Specifying the Default Tablespace Type

The `SET DEFAULT...TABLESPACE` clause of the `CREATE DATABASE` statement determines the default type of tablespace for this database in subsequent `CREATE TABLESPACE` statements.

- Specify either `SET DEFAULT BIGFILE TABLESPACE` or `SET DEFAULT SMALLFILE TABLESPACE`.

If you omit this clause, then the default is a **smallfile tablespace**, which is the traditional type of Oracle Database tablespace. A smallfile tablespace can contain up to 1022 files with up to 4M blocks each.

The use of bigfile tablespaces further enhances the Oracle Managed Files feature, because bigfile tablespaces make data files completely transparent for users. SQL syntax for the `ALTER TABLESPACE` statement has been extended to allow you to perform operations on tablespaces, rather than the underlying data files.

The `CREATE DATABASE` statement shown in "Specifying Oracle Managed Files at Database Creation" can be modified as follows to specify that the default type of tablespace is a bigfile tablespace:

```
CREATE DATABASE mynewdb
     USER SYS IDENTIFIED BY sys_password
     USER SYSTEM IDENTIFIED BY system_password
     SET DEFAULT BIGFILE TABLESPACE
     UNDO TABLESPACE undotbs1
     DEFAULT TEMPORARY TABLESPACE tempts1
     ENABLE PLUGGABLE DATABASE
        SEED
        SYSTEM DATAFILES SIZE 125M AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
        SYSAUX DATAFILES SIZE 100M;
```

To dynamically change the default tablespace type after database creation, use the `SET DEFAULT TABLESPACE` clause of the `ALTER DATABASE` statement:

```
ALTER DATABASE SET DEFAULT BIGFILE TABLESPACE;
```

You can determine the current default tablespace type for the database by querying the `DATABASE_PROPERTIES` data dictionary view as follows:

```
SELECT PROPERTY_VALUE FROM DATABASE_PROPERTIES
   WHERE PROPERTY_NAME = 'DEFAULT_TBS_TYPE';
```

Overriding the Default Tablespace Type

The `SYSTEM` and `SYSAUX` tablespaces are always created with the default tablespace type. However, you optionally can explicitly override the default tablespace type for the `UNDO` and `DEFAULT TEMPORARY` tablespace during the `CREATE DATABASE` operation.

- Specify an `UNDO TABLESPACE` clause or a `DEFAULT TEMPORARY TABLESPACE` clause that overrides the default tablespace type.

For example, you can create a bigfile UNDO tablespace in a database with the default tablespace type of smallfile as follows:

```
CREATE DATABASE mynewdb
...
     BIGFILE UNDO TABLESPACE undotbs1
        DATAFILE '/u01/oracle/oradata/mynewdb/undotbs01.dbf'
        SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

You can create a smallfile DEFAULT TEMPORARY tablespace in a database with the default tablespace type of bigfile as follows:

```
CREATE DATABASE mynewdb
   SET DEFAULT BIGFILE TABLESPACE
...
     SMALLFILE DEFAULT TEMPORARY TABLESPACE tempts1
        TEMPFILE '/u01/oracle/oradata/mynewdb/temp01.dbf'
        SIZE 20M REUSE
...
```

Specifying the Database Time Zone and Time Zone File

Oracle Database datetime and interval data types and time zone support make it possible to store consistent information about the time of events and transactions.

- Setting the Database Time Zone
  You can set the database time zone with the SET TIME_ZONE clause of the CREATE DATABASE statement.
- About the Database Time Zone Files
  Two time zone files are included in a subdirectory of the Oracle home directory. The time zone files contain the valid time zone names.
- Specifying the Database Time Zone File
  All databases that share information must use the same time zone data file.

Setting the Database Time Zone

You can set the database time zone with the SET TIME_ZONE clause of the CREATE DATABASE statement.

- Set the database time zone when the database is created by using the SET TIME_ZONE clause of the CREATE DATABASE statement.

If you do not set the database time zone, then it defaults to the time zone of the host operating system.

You can change the database time zone for a session by using the SET TIME_ZONE clause of the ALTER SESSION statement.

> **See Also:**
>
> *Oracle Database Globalization Support Guide* for more information about setting the database time zone

About the Database Time Zone Files

Two time zone files are included in a subdirectory of the Oracle home directory. The time zone files contain the valid time zone names.

The following information is also included for each time zone:

- Offset from Coordinated Universal Time (UTC)
- Transition times for Daylight Saving Time
- Abbreviations for standard time and Daylight Saving Time

The default time zone file is *ORACLE_HOME*/oracore/zoneinfo/timezlrg_11.dat. A smaller time zone file with fewer time zones can be found in *ORACLE_HOME*/oracore/zoneinfo/timezone_11.dat.

To view the time zone names in the file being used by your database, use the following query:

```
SELECT * FROM V$TIMEZONE_NAMES;
```

> **See Also:**
>
> *Oracle Database Globalization Support Guide* for more information about managing and selecting time zone files

Specifying the Database Time Zone File

All databases that share information must use the same time zone data file.

The database server always uses the large time zone file by default.

To use the small time zone file on the client and know that all your data will refer only to regions in the small file:

- Set the `ORA_TZFILE` environment variable on the client to the full path name of the timezone *`version`*.dat file on the client, where *`version`* matches the time zone file version that is being used by the database server.

If you are already using the default larger time zone file on the client, then it is not practical to change to the smaller time zone file, because the database may contain data with time zones that are not part of the smaller file.

Specifying FORCE LOGGING Mode

Some data definition language statements (such as `CREATE TABLE`) allow the `NOLOGGING` clause, which causes some database operations not to generate redo records in the

database redo log. The NOLOGGING setting can speed up operations that can be easily recovered outside of the database recovery mechanisms, but it can negatively affect media recovery and standby databases.

Oracle Database lets you force the writing of redo records even when NOLOGGING has been specified in DDL statements. The database never generates redo records for temporary tablespaces and temporary segments, so forced logging has no affect for objects.

- Using the FORCE LOGGING Clause
  You can force the writing of redo records even when NOLOGGING is specified in DDL statements.
- Performance Considerations of FORCE LOGGING Mode
  FORCE LOGGING mode results in some performance degradation.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about operations that can be done in NOLOGGING mode

Using the FORCE LOGGING Clause

You can force the writing of redo records even when NOLOGGING is specified in DDL statements.

To put the database into FORCE LOGGING mode:

- Include the FORCE LOGGING clause in the CREATE DATABASE statement.

If you do not specify this clause, then the database is not placed into FORCE LOGGING mode.

Use the ALTER DATABASE statement to place the database into FORCE LOGGING mode after database creation. This statement can take a considerable time for completion, because it waits for all unlogged direct writes to complete.

You can cancel FORCE LOGGING mode using the following SQL statement:

```
ALTER DATABASE NO FORCE LOGGING;
```

Independent of specifying FORCE LOGGING for the database, you can selectively specify FORCE LOGGING or NO FORCE LOGGING at the tablespace level. However, if FORCE LOGGING mode is in effect for the database, it takes precedence over the tablespace setting. If it is not in effect for the database, then the individual tablespace settings are enforced. Oracle recommends that either the entire database is placed into FORCE LOGGING mode, or individual tablespaces be placed into FORCE LOGGING mode, but not both.

The FORCE LOGGING mode is a persistent attribute of the database. That is, if the database is shut down and restarted, it remains in the same logging mode. However, if you re-create the control file, the database is not restarted in the FORCE LOGGING mode unless you specify the FORCE LOGGING clause in the CREATE CONTROL FILE statement.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about using the `FORCE LOGGING` clause for tablespace creation.

Performance Considerations of FORCE LOGGING Mode

`FORCE LOGGING` mode results in some performance degradation.

If the primary reason for specifying `FORCE LOGGING` is to ensure complete media recovery, and there is no standby database active, then consider the following:

- How many media failures are likely to happen?
- How serious is the damage if unlogged direct writes cannot be recovered?
- Is the performance degradation caused by forced logging tolerable?

If the database is running in `NOARCHIVELOG` mode, then generally there is no benefit to placing the database in `FORCE LOGGING` mode. Media recovery is not possible in `NOARCHIVELOG` mode, so if you combine it with `FORCE LOGGING`, the result may be performance degradation with little benefit.

Starting with Oracle Database 18c, the following two new nologging clauses are introduced, which enable non-logged operations to be carried out and have Active Data Guard standby databases receive all the data, thus preventing performance degradation caused by large redo log generation by the `FORCE LOGGING` mode:

- `STANDBY NOLOGGING FOR DATA AVAILABILITY`
- `STANDBY NOLOGGING FOR LOAD PERFORMANCE`

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for more information about these `STANDBY NOLOGGING` clauses

Specifying Initialization Parameters

You can add or edit basic initialization parameters before you create your new database.

- About Initialization Parameters and Initialization Parameter Files
  When an Oracle instance starts, it reads initialization parameters from an initialization parameter file. This file must at a minimum specify the `DB_NAME` parameter. All other parameters have default values.
- Determining the Global Database Name
  The global database name consists of the user-specified local database name and the location of the database within a network structure.

- Specifying a Fast Recovery Area
  The Fast Recovery Area is a location in which Oracle Database can store and manage files related to backup and recovery. It is distinct from the database area, which is a location for the current database files (data files, control files, and online redo logs).
- Specifying Control Files
  Every database has a control file, which contains entries that describe the structure of the database (such as its name, the timestamp of its creation, and the names and locations of its data files and redo files). The `CONTROL_FILES` initialization parameter specifies one or more names of control files, separated by commas.
- Specifying Database Block Sizes
  The `DB_BLOCK_SIZE` initialization parameter specifies the standard block size for the database.
- Specifying the Maximum Number of Processes
  The `PROCESSES` initialization parameter determines the maximum number of operating system processes that can be connected to Oracle Database concurrently.
- Specifying the DDL Lock Timeout
  You can specify the amount of time that blocking DDL statements wait for locks.
- Specifying the Method of Undo Space Management
  Every Oracle Database must have a method of maintaining information that is used to undo changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. Collectively these records are called **undo data**.
- Specifying the Database Compatibility Level
  The `COMPATIBLE` initialization parameter controls the database compatibility level.
- Setting the License Parameter
  If you use named user licensing, Oracle Database can help you enforce this form of licensing. You can set a limit on the number of users created in the database. Once this limit is reached, you cannot create more users.

See Also:

- *Oracle Database Administrator's Guide* for a discussion of the initialization parameters that pertain to memory management
- *Oracle Database Reference* for descriptions of all initialization parameters including their default settings

About Initialization Parameters and Initialization Parameter Files

When an Oracle instance starts, it reads initialization parameters from an initialization parameter file. This file must at a minimum specify the `DB_NAME` parameter. All other parameters have default values.

The initialization parameter file can be either a read-only text file, a `PFILE`, or a read/write binary file.

The binary file is called a **server parameter file**. A server parameter file enables you to change initialization parameters with `ALTER SYSTEM` commands and to persist the changes across a shutdown and startup. It also provides a basis for self-tuning by Oracle Database. For these reasons, it is recommended that you use a server parameter file. You can create one manually from your edited text initialization file, or automatically by using Database Configuration Assistant (DBCA) to create your database.

Before you manually create a server parameter file, you can start an instance with a text initialization parameter file. Upon startup, the Oracle instance first searches for a server parameter file in a default location, and if it does not find one, searches for a text initialization parameter file. You can also override an existing server parameter file by naming a text initialization parameter file as an argument of the `STARTUP` command.

Default file names and locations for the text initialization parameter file are shown in the following table:

Platform	Default Name	Default Location

UNIX and Linux	`init`*`ORACLE_SID`*`.ora`
	For example, the initialization parameter file for the `mynewdb` database is named:
	`initmynewdb.ora`
		`ORACLE_BASE_CONFIG``/dbs`

Windows	`init`*`ORACLE_SID`*`.ora`	*`ORACLE_HOME`*`\database`

If you are creating an Oracle database for the first time, Oracle suggests that you minimize the number of parameter values that you alter. As you become more familiar with your database and environment, you can dynamically tune many initialization parameters using the `ALTER SYSTEM` statement. If you are using a text initialization parameter file, then your changes are effective only for the current instance. To make them permanent, you must update them manually in the initialization parameter file, or they will be lost over the next shutdown and startup of the database. If you are using a server parameter file, then initialization parameter file changes made by the `ALTER SYSTEM` statement can persist across shutdown and startup.

- Sample Initialization Parameter File
  Oracle Database provides generally appropriate values in a sample text initialization parameter file. You can edit these Oracle-supplied initialization parameters and add others, depending upon your configuration and options and how you plan to tune the database.
- Text Initialization Parameter File Format
  The text initialization parameter file specifies the values of parameters in name/value pairs.
- Expressions in Initialization Parameter Settings
  Set the value of an initialization parameter to the desired numeric value, text value, or expression.

> **See Also:**
>
> - "Determining the Global Database Name" for information about the `DB_NAME` parameter
> - "Managing Initialization Parameters Using a Server Parameter File"
> - "About Initialization Parameter Files and Startup"

Sample Initialization Parameter File

Oracle Database provides generally appropriate values in a sample text initialization parameter file. You can edit these Oracle-supplied initialization parameters and add others, depending upon your configuration and options and how you plan to tune the database.

The sample text initialization parameter file is named `init.ora` and is found in the following location on most platforms:

ORACLE_HOME/dbs

The following is the content of the sample file:

```
###########################################################################
#######
# Example INIT.ORA file
#
# This file is provided by Oracle Corporation to help you start by
providing
# a starting point to customize your RDBMS installation for your site.
#
# NOTE: The values that are used in this file are only intended to be
used
# as a starting point. You may want to adjust/tune those values to your
# specific hardware and needs. You may also consider using Database
# Configuration Assistant tool (DBCA) to create INIT file and to size
your
# initial set of tablespaces based on the user input.
###########################################################################
########

# Change '<ORACLE_BASE>' to point to the oracle base (the one you
specify at
# install time)

db_name='ORCL'
memory_target=1G
processes = 150
db_block_size=8192
db_domain=''
db_recovery_file_dest='<ORACLE_BASE>/flash_recovery_area'
db_recovery_file_dest_size=2G
```

```
diagnostic_dest='<ORACLE_BASE>'
dispatchers='(PROTOCOL=TCP) (SERVICE=ORCLXDB)'
open_cursors=300
remote_login_passwordfile='EXCLUSIVE'
undo_tablespace='UNDOTBS1'
# You may want to ensure that control files are created on separate physical
# devices
control_files = (ora_control1, ora_control2)
compatible ='12.0.0'
enable_pluggable_database=TRUE
```

Text Initialization Parameter File Format

The text initialization parameter file specifies the values of parameters in name/value pairs.

The text initialization parameter file (PFILE) must contain name/value pairs in one of the following forms:

- For parameters that accept only a single value:

  parameter_name=value

- For parameters that accept one or more values (such as the `CONTROL_FILES` parameter):

  parameter_name=(*value*[,*value*] ...)

Parameter values of type string must be enclosed in single quotes ('). Case (upper or lower) in file names is significant only if case is significant on the host operating system.

For parameters that accept multiple values, to enable you to easily copy and paste name/value pairs from the alert log, you can repeat a parameter on multiple lines, where each line contains a different value.

```
control_files='/u01/app/oracle/oradata/orcl/control01.ctl'
control_files='/u01/app/oracle/oradata/orcl/control02.ctl'
control_files='/u01/app/oracle/oradata/orcl/control03.ctl'
```

If you repeat a parameter that does not accept multiple values, then only the last value specified takes effect.

See Also:

- *Oracle Database Administrator's Guide* to learn about the alert log
- *Oracle Database Reference* for more information about the content and syntax of the text initialization parameter file

Expressions in Initialization Parameter Settings

Set the value of an initialization parameter to the desired numeric value, text value, or expression.

Starting with Oracle Database Release 21c, you can use expressions when setting the value of initialization parameters. The expressions can contain other initialization parameters and one or more of the following:

- arithmetic operators (+, -, *, /)
- environment variables
- MIN or MAX function

Examples:

```
SGA_TARGET = SYSTEM_MEMORY * 0.4
PARALLEL_MAX_SERVERS = MAX(100, PROCESSES * 0.4)
DB_CREATE_FILE_DEST=$ORACLE_HOME/oracle/database_files
```

Determining the Global Database Name

The global database name consists of the user-specified local database name and the location of the database within a network structure.

- Set the DB_NAME and DB_DOMAIN initialization parameters.

The DB_NAME initialization parameter determines the local name component of the database name, and the DB_DOMAIN parameter, which is optional, indicates the domain (logical location) within a network structure. The combination of the settings for these two parameters must form a database name that is unique within a network.

For example, to create a database with a global database name of test.us.example.com, edit the parameters of the new parameter file as follows:

```
DB_NAME = test
DB_DOMAIN = us.example.com
```

You can rename the GLOBAL_NAME of your database using the ALTER DATABASE RENAME GLOBAL_NAME statement. However, you must also shut down and restart the database after first changing the DB_NAME and DB_DOMAIN initialization parameters and recreating the control files. Recreating the control files is easily accomplished with the command ALTER DATABASE BACKUP CONTROLFILE TO TRACE. See *Oracle Database Backup and Recovery User's Guide* for more information.

- DB_NAME Initialization Parameter
  The DB_NAME initialization parameter specifies a database identifier.
- DB_DOMAIN Initialization Parameter
  In a distributed database system, the DB_DOMAIN initialization parameter specifies the logical location of the database within the network structure.

> **See Also:**
>
> *Oracle Database Utilities* for information about using the DBNEWID utility, which is another means of changing a database name

DB_NAME Initialization Parameter

The DB_NAME initialization parameter specifies a database identifier.

DB_NAME must be set to a text string of no more than 8 characters. The database name must start with an alphabetic character. During database creation, the name provided for DB_NAME is recorded in the data files, redo log files, and control file of the database. If during database instance startup the value of the DB_NAME parameter (in the parameter file) and the database name in the control file are different, then the database does not start.

DB_DOMAIN Initialization Parameter

In a distributed database system, the DB_DOMAIN initialization parameter specifies the logical location of the database within the network structure.

DB_DOMAIN is a text string that specifies the network domain where the database is created. If the database you are about to create will ever be part of a distributed database system, then give special attention to this initialization parameter before database creation. This parameter is optional.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for more information about distributed databases

Specifying a Fast Recovery Area

The Fast Recovery Area is a location in which Oracle Database can store and manage files related to backup and recovery. It is distinct from the database area, which is a location for the current database files (data files, control files, and online redo logs).

Specify the Fast Recovery Area with the following initialization parameters:

- DB_RECOVERY_FILE_DEST: Location of the Fast Recovery Area. This can be a directory, file system, or Automatic Storage Management (Oracle ASM) disk group.

  In an Oracle Real Application Clusters (Oracle RAC) environment, this location must be on a cluster file system, Oracle ASM disk group, or a shared directory configured through NFS.
- DB_RECOVERY_FILE_DEST_SIZE: Specifies the maximum total bytes to be used by the Fast Recovery Area. This initialization parameter must be specified before DB_RECOVERY_FILE_DEST is enabled.

In an Oracle RAC environment, the settings for these two parameters must be the same on all instances.

You cannot enable these parameters if you have set values for the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST parameters. You must disable those parameters before setting up the Fast Recovery Area. You can instead set values for the LOG_ARCHIVE_DEST_*n* parameters.

Oracle recommends using a Fast Recovery Area, because it can simplify backup and recovery operations for your database.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* to learn how to create and use a Fast Recovery Area

Specifying Control Files

Every database has a control file, which contains entries that describe the structure of the database (such as its name, the timestamp of its creation, and the names and locations of its data files and redo files). The `CONTROL_FILES` initialization parameter specifies one or more names of control files, separated by commas.

- Set the `CONTROL_FILES` initialization parameter.

When you execute the `CREATE DATABASE` statement, the control files listed in the `CONTROL_FILES` parameter are created.

If you do not include `CONTROL_FILES` in the initialization parameter file, then Oracle Database creates a control file in the same directory as the initialization parameter file, using a default operating system–dependent file name. If you have enabled Oracle Managed Files, the database creates Oracle managed control files.

If you want the database to create new operating system files when creating database control files, the file names listed in the `CONTROL_FILES` parameter must not match any file names that currently exist on your system. If you want the database to reuse or overwrite existing files when creating database control files, ensure that the file names listed in the `CONTROL_FILES` parameter match the file names that are to be reused, and include a `CONTROLFILE REUSE` clause in the `CREATE DATABASE` statement.

Oracle strongly recommends you use at least two control files stored on separate physical disk drives for each database.

> **See Also:**
>
> - *Oracle Database Administrator's Guide*
> - "Specifying Oracle Managed Files at Database Creation"

Specifying Database Block Sizes

The `DB_BLOCK_SIZE` initialization parameter specifies the standard block size for the database.

- Set the `DB_BLOCK_SIZE` initialization parameter.

This block size is used for the `SYSTEM` tablespace and by default in other tablespaces. Oracle Database can support up to four additional nonstandard block sizes.

- DB_BLOCK_SIZE Initialization Parameter
  The most commonly used block size should be picked as the standard block size. In many cases, this is the only block size that you must specify.

- Nonstandard Block Sizes
  You can create tablespaces of nonstandard block sizes.

DB_BLOCK_SIZE Initialization Parameter

The most commonly used block size should be picked as the standard block size. In many cases, this is the only block size that you must specify.

- Set the `DB_BLOCK_SIZE` initialization parameter.

Typically, `DB_BLOCK_SIZE` is set to either 4K or 8K. If you do not set a value for this parameter, then the default data block size is operating system specific, which is generally adequate.

You cannot change the block size after database creation except by re-creating the database. If the database block size is different from the operating system block size, then ensure that the database block size is a multiple of the operating system block size. For example, if your operating system block size is 2K (2048 bytes), the following setting for the `DB_BLOCK_SIZE` initialization parameter is valid:

```
DB_BLOCK_SIZE=4096
```

A larger data block size provides greater efficiency in disk and memory I/O (access and storage of data). Therefore, consider specifying a block size larger than your operating system block size if the following conditions exist:

- Oracle Database is on a large computer system with a large amount of memory and fast disk drives. For example, databases controlled by mainframe computers with vast hardware resources typically use a data block size of 4K or greater.
- The operating system that runs Oracle Database uses a small operating system block size. For example, if the operating system block size is 1K and the default data block size matches this, the database may be performing an excessive amount of disk I/O during normal operation. For best performance in this case, a database block should consist of multiple operating system blocks.

> **See Also:**
>
> Your operating system specific Oracle documentation for details about the default block size.

Nonstandard Block Sizes

You can create tablespaces of nonstandard block sizes.

To create tablespaces of nonstandard block sizes:

- Specify the `BLOCKSIZE` clause in a `CREATE TABLESPACE` statement.

These nonstandard block sizes can have any of the following power-of-two values: 2K, 4K, 8K, 16K or 32K. Platform-specific restrictions regarding the maximum block size apply, so some of these sizes may not be allowed on some platforms.

To use nonstandard block sizes, you must configure subcaches within the buffer cache area of the SGA memory for all of the nonstandard block sizes that you intend to use. The initialization parameters used for configuring these subcaches are described in *Oracle Database Administrator's Guide*.

The ability to specify multiple block sizes for your database is especially useful if you are transporting tablespaces between databases. You can, for example, transport a tablespace that uses a 4K block size from an OLTP environment to a data warehouse environment that uses a standard block size of 8K.

> **Note:**
>
> A 32K block size is valid only on 64-bit platforms.

> **Caution:**
>
> Oracle recommends against specifying a 2K block size when 4K sector size disks are in use, because performance degradation can occur. For an explanation, see *Oracle Database Administrator's Guide*.

Specifying the Maximum Number of Processes

The `PROCESSES` initialization parameter determines the maximum number of operating system processes that can be connected to Oracle Database concurrently.

- Set the `PROCESSES` initialization parameter.

The value of this parameter must be a minimum of one for each background process plus one for each user process. The number of background processes will vary according the database features that you are using. For example, if you are using Advanced Queuing or the file mapping feature, then you will have additional background processes. If you are using Automatic Storage Management, then add three additional processes for the database instance.

If you plan on running 50 user processes, a good estimate would be to set the `PROCESSES` initialization parameter to 70.

Specifying the DDL Lock Timeout

You can specify the amount of time that blocking DDL statements wait for locks.

A data definition language (DDL) statement is either nonblocking or blocking, and both types of DDL statements require exclusive locks on internal structures. If these locks are unavailable when a DDL statement runs, then nonblocking and blocking DDL statements behave differently:

- Nonblocking DDL waits until every concurrent DML transaction that references the object affected by the DDL either commits or rolls back.
- Blocking DDL fails, though it might have succeeded if it had been executed subseconds later when the locks become available.

To enable blocking DDL statements to wait for locks, specify a **DDL lock timeout**—the number of seconds a DDL command waits for its required locks before failing.

- To specify a DDL lock timeout, set the `DDL_LOCK_TIMEOUT` parameter.

The permissible range of values for DDL_LOCK_TIMEOUT is 0 to 1,000,000. The default is 0. You can set DDL_LOCK_TIMEOUT at the system level, or at the session level with an ALTER SESSION statement.

> **Note:**
>
> The DDL_LOCK_TIMEOUT parameter does not affect nonblocking DDL statements.

> **See Also:**
>
> - *Oracle Database Reference*
> - *Oracle Database Development Guide*
> - *Oracle Database SQL Language Reference*

Specifying the Method of Undo Space Management

Every Oracle Database must have a method of maintaining information that is used to undo changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. Collectively these records are called **undo data**.

To set up an environment for automatic undo management using an undo tablespace.

- Set the UNDO_MANAGEMENT initialization parameter to AUTO, which is the default.

- UNDO_MANAGEMENT Initialization Parameter
  The UNDO_MANAGEMENT initialization parameter determines whether an instance starts in automatic undo management mode, which stores undo in an undo tablespace. Set this parameter to AUTO to enable automatic undo management mode. AUTO is the default if the parameter is omitted or is null.
- UNDO_TABLESPACE Initialization Parameter
  The UNDO_TABLESPACE initialization parameter enables you to override that default undo tablespace for an instance.

> **See Also:**
>
> *Oracle Database Administrator's Guide*

UNDO_MANAGEMENT Initialization Parameter

The UNDO_MANAGEMENT initialization parameter determines whether an instance starts in automatic undo management mode, which stores undo in an undo tablespace. Set this parameter to AUTO to enable automatic undo management mode. AUTO is the default if the parameter is omitted or is null.

UNDO_TABLESPACE Initialization Parameter

The UNDO_TABLESPACE initialization parameter enables you to override that default undo tablespace for an instance.

When an instance starts up in automatic undo management mode, it attempts to select an undo tablespace for storage of undo data. If the database was created in automatic undo management mode, then the default undo tablespace (either the system-created SYS_UNDOTBS tablespace or the user-specified undo tablespace) is the undo tablespace used at instance startup. You can override this default for the instance by specifying a value for the UNDO_TABLESPACE initialization parameter. This parameter is especially useful for assigning a particular undo tablespace to an instance in an Oracle Real Application Clusters environment.

If no undo tablespace is specified by the UNDO_TABLESPACE initialization parameter, then the first available undo tablespace in the database is chosen. If no undo tablespace is available, then the instance starts without an undo tablespace, and undo data is written to the SYSTEM tablespace. You should avoid running in this mode.

> **Note:**
>
> When using the CREATE DATABASE statement to create a database, do not include an UNDO_TABLESPACE parameter in the initialization parameter file. Instead, include an UNDO TABLESPACE clause in the CREATE DATABASE statement.

Specifying the Database Compatibility Level

The COMPATIBLE initialization parameter controls the database compatibility level.

- Set the COMPATIBLE initialization parameter to a release number.
- About the COMPATIBLE Initialization Parameter
  The COMPATIBLE initialization parameter enables or disables the use of features in the database that affect file format on disk. For example, if you create an Oracle Database 19c database, but specify COMPATIBLE=12.0.0 in the initialization parameter file, then features that require Oracle Database 19c compatibility generate an error if you try to use them. Such a database is said to be at the 12.0.0 compatibility level.

About the COMPATIBLE Initialization Parameter

The COMPATIBLE initialization parameter enables or disables the use of features in the database that affect file format on disk. For example, if you create an Oracle Database 19c database, but specify COMPATIBLE=12.0.0 in the initialization parameter file, then features that require Oracle Database 19c compatibility generate an error if you try to use them. Such a database is said to be at the 12.0.0 compatibility level.

You can advance the compatibility level of your database by changing the COMPATIBLE initialization parameter. If you do, then there is no way to start the database using a

lower compatibility level setting, except by doing a point-in-time recovery to a time before the compatibility was advanced.

The default value for the `COMPATIBLE` parameter is the release number of the most recent major release.

> **Note:**
>
> - When you set this parameter in a server parameter file (SPFILE) using the `ALTER SYSTEM` statement, you must specify `SCOPE=SPFILE`, and you must restart the database for the change to take effect.
> - The `COMPATIBLE` initialization parameter must be specified as at least three decimal numbers separated by a dot, such as `19.0.0`.

> **See Also:**
>
> - *Oracle Database Upgrade Guide* for a detailed discussion of database compatibility and the `COMPATIBLE` initialization parameter
> - *Oracle Database Reference*
> - *Oracle Database Backup and Recovery User's Guide* for information about point-in-time recovery of your database

Setting the License Parameter

If you use named user licensing, Oracle Database can help you enforce this form of licensing. You can set a limit on the number of users created in the database. Once this limit is reached, you cannot create more users.

> **Note:**
>
> This mechanism assumes that each person accessing the database has a unique user name and that no people share a user name. Therefore, so that named user licensing can help you ensure compliance with your Oracle license agreement, do not allow multiple users to log in using the same user name.

To limit the number of users created in a database, set the `LICENSE_MAX_USERS` initialization parameter in the database initialization parameter file.

The following example sets the `LICENSE_MAX_USERS` initialization parameter:

```
LICENSE_MAX_USERS = 200
```

> **Note:**
>
> Oracle no longer offers licensing by the number of concurrent sessions. Therefore the `LICENSE_MAX_SESSIONS` and `LICENSE_SESSIONS_WARNING` initialization parameters are no longer needed and have been deprecated.

Managing Initialization Parameters Using a Server Parameter File

Initialization parameters for the Oracle Database have traditionally been stored in a text initialization parameter file. For better manageability, you can choose to maintain initialization parameters in a binary server parameter file that is persistent across database startup and shutdown.

- What Is a Server Parameter File?
  A server parameter file can be thought of as a repository for initialization parameters that is maintained on the system running the Oracle Database server. It is, by design, a server-side initialization parameter file.
- Migrating to a Server Parameter File
  If you are currently using a text initialization parameter file, then you can migrate to a server parameter file.
- Server Parameter File Default Names and Locations
  Oracle recommends that you allow the database to give the SPFILE the default name and store it in the default location. This eases administration of your database. For example, the `STARTUP` command assumes this default location to read the SPFILE.
- Creating a Server Parameter File
  You use the `CREATE SPFILE` statement to create a server parameter file. You must have the `SYSDBA`, `SYSOPER`, or `SYSBACKUP` administrative privilege to execute this statement.
- The SPFILE Initialization Parameter
  The `SPFILE` initialization parameter contains the name of the current server parameter file.
- Changing Initialization Parameter Values
  You can change initialization parameter values to affect the operation of a database instance.
- Clearing Initialization Parameter Values
  You can use the `ALTER SYSTEM RESET` statement to clear an initialization parameter value. When you do so, the initialization parameter value is changed to its default value or its startup value.
- Exporting the Server Parameter File
  You can use the `CREATE PFILE` statement to export a server parameter file (SPFILE) to a text initialization parameter file.
- Backing Up the Server Parameter File
  You can create a backup of your server parameter file (SPFILE) by exporting it. If the backup and recovery strategy for your database is implemented using Recovery Manager (RMAN), then you can use RMAN to create a backup of the

SPFILE. The SPFILE is backed up automatically by RMAN when you back up your database, but RMAN also enables you to specifically create a backup of the currently active SPFILE.

- Recovering a Lost or Damaged Server Parameter File
  You can recover the server parameter file (SPFILE). If your server parameter file (SPFILE) becomes lost or corrupted, then the current instance may fail, or the next attempt at starting the database instance may fail.
- Methods for Viewing Parameter Settings
  You can view parameter settings using several different methods.

What Is a Server Parameter File?

A server parameter file can be thought of as a repository for initialization parameters that is maintained on the system running the Oracle Database server. It is, by design, a server-side initialization parameter file.

Initialization parameters stored in a server parameter file are persistent, in that any changes made to the parameters while an instance is running can persist across instance shutdown and startup. This arrangement eliminates the need to manually update initialization parameters to make persistent any changes effected by `ALTER SYSTEM` statements. It also provides a basis for self-tuning by the Oracle Database server.

A server parameter file is initially built from a text initialization parameter file using the `CREATE SPFILE` statement. (It can also be created directly by the Database Configuration Assistant.) The server parameter file is a binary file that cannot be edited using a text editor. Oracle Database provides other interfaces for viewing and modifying parameter settings in a server parameter file.

> **Note:**
>
> Although you can open the binary server parameter file with a text editor and view its text, *do not* manually edit it. Doing so will corrupt the file. You will not be able to start your instance, and if the instance is running, it could fail.

When you issue a `STARTUP` command with no `PFILE` clause, the Oracle instance searches an operating system–specific default location for a server parameter file from which to read initialization parameter settings. If no server parameter file is found, the instance searches for a text initialization parameter file. If a server parameter file exists but you want to override it with settings in a text initialization parameter file, you must specify the `PFILE` clause when issuing the `STARTUP` command.

> **See Also:**
>
> "Starting Up a CDB"

Migrating to a Server Parameter File

If you are currently using a text initialization parameter file, then you can migrate to a server parameter file.

To migrate to a server parameter file:

1. If the initialization parameter file is located on a client system, then transfer the file (for example, FTP) from the client system to the server system.

   > **Note:**
   >
   > If you are migrating to a server parameter file in an Oracle Real Application Clusters environment, you must combine all of your instance-specific initialization parameter files into a single initialization parameter file. Instructions for doing this and other actions unique to using a server parameter file for instances that are part of an Oracle Real Application Clusters installation are discussed in *Oracle Real Application Clusters Administration and Deployment Guide* and in your platform-specific Oracle Real Application Clusters Installation Guide.

2. Create a server parameter file in the default location using the `CREATE SPFILE FROM PFILE` statement. See "Creating a Server Parameter File" for instructions.

   This statement reads the text initialization parameter file to create a server parameter file. The database does not have to be started to issue a `CREATE SPFILE` statement.

3. Start up or restart the instance.

   The instance finds the new SPFILE in the default location and starts up with it.

Server Parameter File Default Names and Locations

Oracle recommends that you allow the database to give the SPFILE the default name and store it in the default location. This eases administration of your database. For example, the `STARTUP` command assumes this default location to read the SPFILE.

The following table shows the default name and location for both the text initialization parameter file (PFILE) and server parameter file (SPFILE) for the UNIX, Linux, and Windows platforms, both with and without the presence of Oracle Automatic Storage Management (Oracle ASM). The table assumes that the SPFILE is a file.

Table 4-1 PFILE and SPFILE Default Names and Locations on UNIX, Linux, and Windows

Platform	PFILE Default Name	SPFILE Default Name	PFILE Default Location	SPFILE Default Location
UNIX and Linux	init*ORACLE_SID*.ora	spfile*ORACLE_SID*.ora	*ORACLE_BASE_CONFIG*/dbs or the same location as the data files	**Without Oracle ASM**: *ORACLE_BASE_CONFIG*/dbs or the same location as the data files **When Oracle ASM is present**: In the same disk group as the data files (assuming the database was created with DBCA)
Windows	init*ORACLE_SID*.ora	spfile*ORACLE_SID*.ora	*Oracle_Home*\database	**Without Oracle ASM**: *OH*\database **When Oracle ASM is present**: In the same disk group as the data files (assuming the database was created with DBCA)

> **Note:**
>
> Upon startup, the instance first searches for an SPFILE named spfile*ORACLE_SID*.ora, and if not found, searches for spfile.ora. Using spfile.ora enables all Real Application Cluster (Oracle RAC) instances to use the same server parameter file.
>
> If neither SPFILE is found, the instance searches for the text initialization parameter file init*ORACLE_SID*.ora.

If you create an SPFILE in a location other than the default location, you must create in the default PFILE location a "stub" PFILE that points to the server parameter file. For more information, see "Starting Up a Database".

When you create the database with DBCA when Oracle ASM is present, DBCA places the SPFILE in an Oracle ASM disk group, and also causes this stub PFILE to be created.

Creating a Server Parameter File

You use the CREATE SPFILE statement to create a server parameter file. You must have the SYSDBA, SYSOPER, or SYSBACKUP administrative privilege to execute this statement.

To create a server parameter file:

- Run the CREATE SPFILE statement.

> **Note:**
>
> When you use the Database Configuration Assistant to create a database, it automatically creates a server parameter file for you.

The `CREATE SPFILE` statement can be executed before or after instance startup. However, if the instance has been started using a server parameter file, an error is raised if you attempt to re-create the same server parameter file that is currently being used by the instance.

You can create a server parameter file (SPFILE) from an existing text initialization parameter file or from memory. Creating the SPFILE from memory means copying the current values of initialization parameters in the running instance to the SPFILE.

The following example creates a server parameter file from text initialization parameter file `/u01/oracle/dbs/init.ora`. In this example no `SPFILE` name is specified, so the file is created with the platform-specific default name and location shown in Table 4-1.

```
CREATE SPFILE FROM PFILE='/u01/oracle/dbs/init.ora';
```

The next example illustrates creating a server parameter file and supplying a name and location.

```
CREATE SPFILE='/u01/oracle/dbs/test_spfile.ora'
       FROM PFILE='/u01/oracle/dbs/test_init.ora';
```

The next example illustrates creating a server parameter file in the default location from the current values of the initialization parameters in memory.

```
CREATE SPFILE FROM MEMORY;
```

Whether you use the default SPFILE name and default location or specify an SPFILE name and location, if an SPFILE of the same name already exists in the location, it is overwritten without a warning message.

When you create an SPFILE from a text initialization parameter file, comments specified on the same lines as a parameter setting in the initialization parameter file are maintained in the SPFILE. All other comments are ignored.

The SPFILE Initialization Parameter

The `SPFILE` initialization parameter contains the name of the current server parameter file.

When the default server parameter file is used by the database—that is, you issue a `STARTUP` command and do not specify a `PFILE` parameter—the value of `SPFILE` is internally set by the server. The SQL*Plus command `SHOW PARAMETERS SPFILE` (or any other method of querying the value of a parameter) displays the name of the server parameter file that is currently in use.

Changing Initialization Parameter Values

You can change initialization parameter values to affect the operation of a database instance.

About Changing Initialization Parameter Values

The ALTER SYSTEM statement enables you to set, change, or restore to default the values of initialization parameters. If you are using a text initialization parameter file, the ALTER SYSTEM statement changes the value of a parameter only for the current instance, because there is no mechanism for automatically updating text initialization parameters on disk. You must update them manually to be passed to a future instance. Using a server parameter file overcomes this limitation.

There are two kinds of initialization parameters:

- **Dynamic initialization parameters** can be changed for the current Oracle Database instance. The changes take effect immediately.
- **Static initialization parameters** cannot be changed for the current instance. You must change these parameters in the text initialization file or server parameter file and then restart the database before changes take effect.

Setting or Changing Initialization Parameter Values

With a server parameter file, use the SET clause of the ALTER SYSTEM statement to set or change initialization parameter values.

- Run an ALTER SYSTEM SET statement.

For example, the following statement changes the maximum number of failed login attempts before the connection is dropped. It includes a comment, and explicitly states that the change is to be made only in the server parameter file.

```
ALTER SYSTEM SET SEC_MAX_FAILED_LOGIN_ATTEMPTS=3
                 COMMENT='Reduce from 10 for tighter security.'
                 SCOPE=SPFILE;
```

The next example sets a complex initialization parameter that takes a list of attributes. Specifically, the parameter value being set is the LOG_ARCHIVE_DEST_n initialization parameter. This statement could change an existing setting for this parameter or create a new archive destination.

```
ALTER SYSTEM
     SET LOG_ARCHIVE_DEST_4='LOCATION=/u02/oracle/rbdb1/',MANDATORY,'REOPEN=2'
         COMMENT='Add new destination on Nov 29'
         SCOPE=SPFILE;
```

When a value consists of a list of parameters, you cannot edit individual attributes by the position or ordinal number. You must specify the complete list of values each time the parameter is updated, and the new list completely replaces the old list.

The SCOPE Clause in ALTER SYSTEM SET Statements

The optional SCOPE clause in ALTER SYSTEM SET statements specifies the scope of an initialization parameter change.

SCOPE Clause	Description
SCOPE = SPFILE	The change is applied in the server parameter file only. The effect is as follows: • No change is made to the current instance. • For both dynamic and static parameters, the change is effective at the next startup and is persistent. This is the only SCOPE specification allowed for static parameters.
SCOPE = MEMORY	The change is applied in memory only. The effect is as follows: • The change is made to the current instance and is effective immediately. • For dynamic parameters, the effect is immediate, but it is not persistent because the server parameter file is not updated. For static parameters, this specification is not allowed.
SCOPE = BOTH	The change is applied in both the server parameter file and memory. The effect is as follows: • The change is made to the current instance and is effective immediately. • For dynamic parameters, the effect is persistent because the server parameter file is updated. For static parameters, this specification is not allowed.

It is an error to specify SCOPE=SPFILE or SCOPE=BOTH if the instance did not start up with a server parameter file. The default is SCOPE=BOTH if a server parameter file was used to start up the instance, and MEMORY if a text initialization parameter file was used to start up the instance.

For dynamic parameters, you can also specify the DEFERRED keyword. When specified, the change is effective only for future sessions.

When you specify SCOPE as SPFILE or BOTH, an optional COMMENT clause lets you associate a text string with the parameter update. The comment is written to the server parameter file.

Clearing Initialization Parameter Values

You can use the ALTER SYSTEM RESET statement to clear an initialization parameter value. When you do so, the initialization parameter value is changed to its default value or its startup value.

The ALTER SYSTEM RESET statement includes a SCOPE clause. When executed in a non-CDB or a multitenant container database (CDB) root, the ALTER SYSTEM RESET statement and SCOPE clause behave differently than when the statement is executed in a pluggable database (PDB), an application root, or an application PDB.

The startup value of a parameter is the value of the parameter in memory after the instance's startup or PDB open has completed. This value can be seen in the VALUE and DISPLAY_VALUE columns in the V$SYSTEM_PARAMETER view immediately after startup. The startup value can be different from the value in the spfile or the default value (if the parameter is not set in the spfile), since the value of the parameter can be adjusted internally at startup.

The SCOPE values for the ALTER SYSTEM RESET statement behave as follows in a non-CDB and in the CDB$ROOT of a CDB:

- SCOPE=SPFILE: If an instance is using spfile, removes the parameter from the spfile; the default value takes effect upon the next instance startup.
- SCOPE=MEMORY: The startup value takes effect immediately. However, the change is not stored in instance's spfile and will be lost upon instance restart.
- SCOPE=BOTH: If an instance is using spfile, removes the parameter from the spfile; the default value takes effect immediately and the change is available across instance restart.

> **Note:**
>
> SCOPE=BOTH changes the way SCOPE=MEMORY behaves. After SCOPE=BOTH is issued, SCOPE=MEMORY always resets the parameter to the default value.

The SCOPE values for the ALTER SYSTEM RESET statement behave as follows in a PDB, an application root, or an application PDB:

- SCOPE=SPFILE: Removes the parameter from the container's spfile; the container will inherit the parameter value from its root upon the next PDB open.
- SCOPE=MEMORY: There are two cases:
  - The parameter is present in container's spfile when the container is opened. The parameter value is updated to the startup value for the parameter. This change is not stored in container’s spfile and will be lost upon the next container open.
  - The parameter is not present in container’s spfile when the container is opened. The container starts inheriting the parameter value from its root.
- SCOPE=BOTH: Removes the parameter from the container’s spfile; the container will inherit the parameter value from its root.

> **Note:**
>
> - SCOPE=BOTH changes the way SCOPE=MEMORY behaves. After SCOPE=BOTH is issued, the container always inherits the parameter value from its root when SCOPE=MEMORY is issued.
> - In a case where a container inherits a parameter value from its root, a PDB inherits the value from CDB$ROOT. In an application container, an application PDB inherits the parameter value from its application root, and an application root inherits the parameter value from CDB$ROOT.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the `ALTER SYSTEM` command

Exporting the Server Parameter File

You can use the `CREATE PFILE` statement to export a server parameter file (SPFILE) to a text initialization parameter file.

- Run a `CREATE PFILE` statement.

Exporting the server parameter file might be necessary for several reasons:

- For diagnostic purposes, listing all of the parameter values currently used by an instance. This is analogous to the SQL*Plus `SHOW PARAMETERS` command or selecting from the `V$PARAMETER` or `V$PARAMETER2` views.
- To modify the server parameter file by first exporting it, editing the resulting text file, and then re-creating it using the `CREATE SPFILE` statement

The exported file can also be used to start up an instance using the `PFILE` clause.

You must have the `SYSDBA`, `SYSOPER`, or `SYSBACKUP` administrative privilege to execute the `CREATE PFILE` statement. The exported file is created on the database server system. It contains any comments associated with the parameter in the same line as the parameter setting.

The following example creates a text initialization parameter file from the SPFILE:

```
CREATE PFILE FROM SPFILE;
```

Because no names were specified for the files, the database creates an initialization parameter file with a platform-specific name, and it is created from the platform-specific default server parameter file.

The following example creates a text initialization parameter file from a server parameter file, but in this example the names of the files are specified:

```
CREATE PFILE='/u01/oracle/dbs/test_init.ora'
  FROM SPFILE='/u01/oracle/dbs/test_spfile.ora';
```

> **Note:**
>
> An alternative is to create a PFILE from the current values of the initialization parameters in memory. The following is an example of the required command:
>
> ```
> CREATE PFILE='/u01/oracle/dbs/test_init.ora' FROM MEMORY;
> ```

Backing Up the Server Parameter File

You can create a backup of your server parameter file (SPFILE) by exporting it. If the backup and recovery strategy for your database is implemented using Recovery Manager (RMAN), then you can use RMAN to create a backup of the SPFILE. The SPFILE is backed up automatically by RMAN when you back up your database, but RMAN also enables you to specifically create a backup of the currently active SPFILE.

- Back up the server parameter file either by exporting it or by using RMAN.

> **See Also:**
>
> - "Exporting the Server Parameter File"
> - *Oracle Database Backup and Recovery User's Guide*

Recovering a Lost or Damaged Server Parameter File

You can recover the server parameter file (SPFILE). If your server parameter file (SPFILE) becomes lost or corrupted, then the current instance may fail, or the next attempt at starting the database instance may fail.

There are several ways to recover the SPFILE:

- If the instance is running, issue the following command to re-create the SPFILE from the current values of initialization parameters in memory:

  ```
  CREATE SPFILE FROM MEMORY;
  ```

  This command creates the SPFILE with the default name and in the default location. You can also create the SPFILE with a new name or in a specified location. See "Creating a Server Parameter File" for examples.
- If you have a valid text initialization parameter file (PFILE), re-create the SPFILE from the PFILE with the following statement:

  ```
  CREATE SPFILE FROM PFILE;
  ```

  This command assumes that the PFILE is in the default location and has the default name. See "Creating a Server Parameter File" for the command syntax to use when the PFILE is not in the default location or has a nondefault name.
- Restore the SPFILE from backup.

  See "Backing Up the Server Parameter File" for more information.
- If none of the previous methods are possible in your situation, perform these steps:

  1. Create a text initialization parameter file (PFILE) from the parameter value listings in the alert log.

When an instance starts up, the initialization parameters used for startup are written to the alert log. You can copy and paste this section from the text version of the alert log (without XML tags) into a new PFILE.

See *Oracle Database Administrator's Guide* for more information.

2. Create the SPFILE from the PFILE.

   See "Creating a Server Parameter File" for instructions.

Read/Write Errors During a Parameter Update

If an error occurs while reading or writing the server parameter file during a parameter update, the error is reported in the alert log and all subsequent parameter updates to the server parameter file are ignored. At this point, you can take one of the following actions:

- Shut down the instance, recover the server parameter file and described earlier in this section, and then restart the instance.
- Continue to run the database if you do not care that subsequent parameter updates will not be persistent.

Methods for Viewing Parameter Settings

You can view parameter settings using several different methods.

Method	Description
`SHOW PARAMETERS`	This SQL*Plus command displays the values of initialization parameters in effect for the current session.
`SHOW SPPARAMETERS`	This SQL*Plus command displays the values of initialization parameters in the server parameter file (SPFILE).
`CREATE PFILE`	This SQL statement creates a text initialization parameter file (PFILE) from the SPFILE or from the current in-memory settings. You can then view the PFILE with any text editor.
`V$PARAMETER`	This view displays the values of initialization parameters in effect for the current session.
`V$PARAMETER2`	This view displays the values of initialization parameters in effect for the current session. It is easier to distinguish list parameter values in this view because each list parameter value appears in a separate row.
`V$SYSTEM_PARAMETER`	This view displays the values of initialization parameters in effect for the instance. A new session inherits parameter values from the instance-wide values.
`V$SYSTEM_PARAMETER2`	This view displays the values of initialization parameters in effect for the instance. A new session inherits parameter values from the instance-wide values. It is easier to distinguish list parameter values in this view because each list parameter value appears in a separate row.
`V$SPPARAMETER`	This view displays the current contents of the SPFILE. The view returns `FALSE` values in the `ISSPECIFIED` column if an SPFILE is not being used by the instance.

> **See Also:**
>
> *Oracle Database Reference* for a complete description of views

Managing Application Workloads with Database Services

A database service is a named representation of one or more database instances. Services enable you to group database workloads and route a particular work request to an appropriate instance.

- Database Services
  A database service represents a single database. This database can be a single-instance database or an Oracle Real Application Clusters (Oracle RAC) database with multiple concurrent database instances. A global database service is a service provided by multiple databases synchronized through data replication.
- Global Data Services
  Starting with Oracle Database 12*c*, you can use Global Data Services (GDS) for workload management involving multiple Oracle databases.
- Reset Database Session State to Prevent Application State Leaks
  When you use the `RESET_STATE` service attribute, the session state set by an application in a request is cleared when the request ends.
- Database Service Data Dictionary Views
  You can query data dictionary views to find information about database services.

Database Services

A database service represents a single database. This database can be a single-instance database or an Oracle Real Application Clusters (Oracle RAC) database with multiple concurrent database instances. A global database service is a service provided by multiple databases synchronized through data replication.

- About Database Services
  Database services divide workloads for a single database into mutually disjoint groupings.
- Database Services and Performance
  Database services offer an extra dimension in performance tuning.
- Oracle Database Features That Use Database Services
  Several Oracle Database features support database services.
- Creating Database Services
  There are a few ways to create database services, depending on your database configuration.

About Database Services

Database services divide workloads for a single database into mutually disjoint groupings.

Each database service represents a workload with common attributes, service-level thresholds, and priorities. The grouping is based on attributes of work that might include the application function to be used, the priority of execution for the application function, the job

class to be managed, or the data range used in the application function or job class. For example, the Oracle E-Business Suite defines a database service for each responsibility, such as general ledger, accounts receivable, order entry, and so on. When you configure database services, you give each service a unique name, associated performance goals, and associated importance. The database services are tightly integrated with Oracle Database and are maintained in the data dictionary.

Connection requests can include a database service name. Thus, middle-tier applications and client/server applications use a service by specifying the database service as part of the connection in TNS connect data. If no database service name is included and the Net Services file listener.ora designates a default database service, then the connection uses the default database service.

Database services enable you to configure a workload for a single database, administer it, enable and disable it, and measure the workload as a single entity. You can do this using standard tools such as the Database Configuration Assistant (DBCA), Oracle Net Configuration Assistant, and Oracle Enterprise Manager Cloud Control (Cloud Control). Cloud Control supports viewing and operating services as a whole, with drill down to the instance-level when needed.

In an Oracle Real Application Clusters (Oracle RAC) environment, a database service can span one or more instances and facilitate workload balancing based on transaction performance. This capability provides end-to-end unattended recovery, rolling changes by workload, and full location transparency. Oracle RAC also enables you to manage several database service features with Cloud Control, the DBCA, and the Server Control utility (`SRVCTL`).

Database services describe applications, application functions, and data ranges as either functional services or data-dependent services. Functional services are the most common mapping of workloads. Sessions using a particular function are grouped together. In contrast, data-dependent routing routes sessions to database services based on data keys. The mapping of work requests to database services occurs in the object relational mapping layer for application servers and TP monitors. For example, in Oracle RAC, these ranges can be completely dynamic and based on demand because the database is shared.

In addition to database services that are used by applications, Oracle Database also supports two internal database services: `SYS$BACKGROUND` is used by the background processes only, and `SYS$USERS` is the default database service for user sessions that are not associated with services.

Using database services requires no changes to your application code. Client-side work can connect to a named database service. Server-side work, such as Oracle Scheduler, parallel execution, and Oracle Database Advanced Queuing, set the database service name as part of the workload definition. Work requests executing under a database service inherit the performance thresholds for the service and are measured as part of the service.

See Also:

- *Oracle Database Concepts*
- *Oracle Real Application Clusters Administration and Deployment Guide* for information about using services in an Oracle RAC environment
- *Oracle Database Net Services Administrator's Guide* for information on connecting to a service
- The Cloud Control online help

Database Services and Performance

Database services offer an extra dimension in performance tuning.

Tuning by "service and SQL" can replace tuning by "session and SQL" in the majority of systems where all sessions are anonymous and shared. With database services, workloads are visible and measurable. Resource consumption and waits are attributable by application. Additionally, resources assigned to database services can be augmented when loads increase or decrease. This dynamic resource allocation enables a cost-effective solution for meeting demands as they occur. For example, database services are measured automatically, and the performance is compared to service-level thresholds. Performance violations are reported to Cloud Control, enabling the execution of automatic or scheduled solutions.

Oracle Database Features That Use Database Services

Several Oracle Database features support database services.

The Automatic Workload Repository (AWR) manages the performance of services. AWR records database service performance, including execution times, wait classes, and resources consumed by services. AWR alerts warn when database service response time thresholds are exceeded. The dynamic views report current service performance metrics with one hour of history. Each database service has quality-of-service thresholds for response time and CPU consumption.

In addition, the Database Resource Manager can map database services to consumer groups. Therefore, you can automatically manage the priority of one database service relative to others. You can use consumer groups to define relative priority in terms of either ratios or resource consumption.

You also can specify an edition attribute for a database service. Editions make it possible to have two or more versions of the same objects in the database. When you specify an edition attribute for a database service, all subsequent connections that specify the database service use this edition as the initial session edition.

Specifying an edition as a database service attribute can make it easier to manage resource usage. For example, database services associated with an edition can be placed on a separate instance in an Oracle RAC environment, and the Database Resource Manager can manage resources used by different editions by associating resource plans with the corresponding database services.

For Oracle Scheduler, you optionally assign a database service when you create a job class. During execution, jobs are assigned to job classes, and job classes can run within database

services. Using database services with job classes ensures that the work executed by the job scheduler is identified for workload management and performance tuning.

For parallel query and parallel DML, the query coordinator connects to a database service just like any other client. The parallel query processes inherit the database service for the duration of the execution. At the end of query execution, the parallel execution processes revert to the default database service.

> **See Also:**
>
> *Oracle Database Administrator's Guide* to learn about the following topics:
>
> - "Managing Resources with Oracle Database Resource Manager"
> - "Specifying Session-to-Consumer Group Mapping Rules"
> - "Setting the Edition Attribute of a Database Service"
> - "Scheduling Jobs with Oracle Scheduler"

Creating Database Services

There are a few ways to create database services, depending on your database configuration.

> **Note:**
>
> Starting with Oracle Database 19c, customer use of the `SERVICE_NAMES` parameter is deprecated. It can be desupported in a future release. To manage your services, Oracle recommends that you use the `SRVCTL` or `GDSCTL` command line utilities, or the `DBMS_SERVICE` package.

> **Note:**
>
> This section describes creating services locally.

To create a database service:

- If your single-instance database is being managed by Oracle Restart, use the `SRVCTL` utility to create the database service.

  ```
  srvctl add service -db db_unique_name -service service_name
  ```

- If your single-instance database is not being managed by Oracle Restart, do one of the following:
  - Append the desired database service name to the `SERVICE_NAMES` parameter.
  - Call the `DBMS_SERVICE.CREATE_SERVICE` package procedure.

- (Optional) Define database service attributes with Cloud Control or with `DBMS_SERVICE.MODIFY_SERVICE`.

Oracle Net Listener (the listener) receives incoming client connection requests and manages the traffic of these requests to the database server. The listener handles connections for registered services, and it supports dynamic service registration.

See Also:

- "Global Data Services"
- *Oracle Database Administrator's Guide* for information about Oracle Restart
- *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 a service in an Oracle RAC environment
- *Oracle Database Net Services Administrator's Guide* for more information about Oracle Net Listener and services

Global Data Services

Starting with Oracle Database 12*c*, you can use Global Data Services (GDS) for workload management involving multiple Oracle databases.

GDS enables administrators to automatically and transparently manage client workloads across replicated databases that offer common services. These common services are known as global services.

GDS enables you to integrate multiple databases in various locations into private GDS configurations that can be shared by global clients. Benefits include the following:

- Enables central management of global resources
- Provides global scalability, availability, and run-time load balancing
- Allows you to dynamically add databases to the GDS configuration and dynamically migrate global services
- Extends service management, load balancing, and failover capabilities for distributed environments of replicated databases that use features such as Oracle Active Data Guard, Oracle GoldenGate, and so on
- Provides high availability through seamless failover of global services across databases (located both locally or globally)
- Provides workload balancing both within and between data centers through services, connection load balancing, and runtime load balancing
- Allows efficient utilization of the resources of the GDS configuration to service client requests

> **See Also:**
>
> *Oracle Database Global Data Services Concepts and Administration Guide* for an overview of Global Data Services

Reset Database Session State to Prevent Application State Leaks

When you use the `RESET_STATE` service attribute, the session state set by an application in a request is cleared when the request ends.

The service attribute `RESET_STATE` is recommended for all stateless applications to prevent the leakage of a session state to later reuses.

Stateless Applications

A stateless application is an application program that does not use the session state of one request, such as context and PL/SQL states that were set by a prior usage of that session, in another web request or similar connection pool usage. The necessary state to handle the request is contained within the request, as a part of the request itself; in the URL, query-string parameters, request body, or headers.

In a cloud environment, it is preferable for applications to be stateless for scalability and portability. Being stateless enables greater scalability because the server does not have to maintain, update, or communicate a session state. For example, load balancers do not have to consider session affinity for stateless systems. Most modern web applications are stateless.

Prevent Application State Leaking to Later Usages

Setting the session state in a request leaves the session with the current state, meaning that subsequent usages of that session can see the current session state. For example, when an application borrows and returns a connection to a connection pool, if the sessions state is not cleared, the next usage of that connection can see the session state of the previous usage.

`RESET_STATE` is an attribute of the database service. When you use the `RESET_STATE` service attribute, the session state set by an application in a request is cleared when the database request ends. When `RESET_STATE` is used, an application can depend on the state being reset at the end of a request. Without `RESET_STATE`, application developers must cancel their cursors and clear the session state that has been set before returning their connections to a pool for reuse.

`RESET_STATE` is used with applications that are stateless between requests. These type of applications use the session state in a request, and do not rely on that session state in the later requests. The necessary session state that the request needs is contained within the request itself. REST, ORDS, Oracle Application Express (APEX) are examples of stateless applications.

`RESET_STATE` is available for all applications that use Oracle and third party connection pools with request boundaries. Setting `RESET_STATE` to `LEVEL1` enables automatic resetting of session states at an explicit end of request. `RESET_STATE` does not apply to implicit request boundaries, such as those with DRCP implicit statement caching.

Using the `RESET_STATE` attribute has the following impact at the end of a request:

- Cursors are canceled.
- PL/SQL global variables are cleared.
- Temporary tables that have a session-based duration are truncated.
- Temporary LOBs that have a session-based duration are cleared.
- Session local sequences are reset.

`RESET_STATE` is a very important database feature that enables your developers to rely on the session state being clean when a session is returned to a connection pool with request boundaries. This can be an Oracle connection pool or a custom connection pool with added request boundaries. `RESET_STATE` also improves your protection when using Transparent Application Continuity (TAC), because the session state is clean at the beginning of a new request.

Database Service Data Dictionary Views

You can query data dictionary views to find information about database services.

You can find information about database services in the following views:

- `DBA_SERVICES`
- `ALL_SERVICES` or `V$SERVICES`
- `V$ACTIVE_SERVICES`
- `V$SERVICE_STATS`
- `V$SERVICE_EVENT`
- `V$SERVICE_WAIT_CLASS`
- `V$SERV_MOD_ACT_STATS`
- `V$SERVICEMETRIC`
- `V$SERVICEMETRIC_HISTORY`

The following additional views also contain some information about database services:

- `V$SESSION`
- `V$ACTIVE_SESSION_HISTORY`
- `DBA_RSRC_GROUP_MAPPINGS`
- `DBA_SCHEDULER_JOB_CLASSES`
- `DBA_THRESHOLDS`

The `ALL_SERVICES` view includes a `GLOBAL_SERVICE` column, and the `V$SERVICES` and `V$ACTIVE_SERVICES` views contain a `GLOBAL` column. These views and columns enable you to determine whether a database service is a global service.

Managing Standard Edition High Availability for Oracle Databases

The Standard Edition High Availability feature provides protection against unplanned outages for Oracle Database Standard Edition 2 single instance databases using Oracle Clusterware.

> **Note:**
>
> Standard Edition High Availability is supported in this release of Oracle Database.

> **Note:**
>
> The `srvctl` commands used with Standard Edition High Availability are different from those used with Oracle Restart. For Standard Edition High Availability, refer to the `srvctl` commands documented in the *Oracle Real Application Clusters Administration and Deployment Guide*.

- About Standard Edition High Availability
  In this release, you can install Oracle Database Standard Edition 2 in high availability mode.
- Requirements for Using Standard Edition High Availability With Oracle Databases
  To use Standard Edition High Availability, deploy Oracle Database Standard Edition 2 in accordance with these configuration requirements.
- Enabling Standard Edition High Availability for Oracle Databases
  You enable Standard Edition High Availability to provide cluster-based failover for an Oracle Database Standard Edition 2 database.
- Create Standard Edition High Availability Database Using DBCA
  Oracle Database Configuration Assistant (Oracle DBCA) enables you to create a Standard Edition High Availability (SEHA), single-instance Oracle Database.
- Relocating a Standard Edition High Availability Database to Another Node
  To manage planned outages, you can relocate an Oracle Database Standard Edition 2 database that uses Standard Edition High Availability to another configured node.
- Adding a Node to a Standard Edition High Availability Database
  Adding new nodes to an existing Standard Edition High Availability configuration provides enhanced failover capabilities to your Standard Edition 2 database.
- Removing a Configured Node from a Standard Edition High Availability Database
  Use `srvctl` commands to remove a node from the list of nodes configured for a Standard Edition High Availability database.
- Starting and Stopping Standard Edition High Availability Databases
  Use `srvctl` commands to start or stop an Oracle Database that is configured for Standard Edition High Availability.
- Deactivating Standard Edition High Availability for Oracle Databases
  When you deactivate Standard Edition High Availability for a single instance Oracle Database, the database is no longer part of a high availability failover configuration.

Related Topics

- *Oracle Database Installation Guide for Linux*
- *Oracle Real Application Clusters Administration and Deployment Guide*

About Standard Edition High Availability

In this release, you can install Oracle Database Standard Edition 2 in high availability mode.

Standard Edition High Availability provides cluster-based failover for single-instance Standard Edition Oracle Databases using Oracle Clusterware.

Oracle Standard Edition High Availability benefits from the cluster capabilities and storage solutions that are already part of Oracle Grid Infrastructure, such as Oracle Clusterware, Oracle Automatic Storage Management (Oracle ASM) and Oracle Advanced Cluster File System (Oracle ACFS).

Using integrated, shared, and concurrently mounted storage, such as Oracle ASM and Oracle ACFS for database files as well as for unstructured data, enables Oracle Grid Infrastructure to restart an Oracle Database on a failover node much faster than any cluster solution that relies on failing over and remounting volumes and file systems.

Standard Edition High Availability is supported on Linux x86-64.

> **Note:**
>
> This section is specific to Standard Edition High Availability, which provides cluster-based database failover for Standard Edition Oracle Databases 23c and later. For more information about high availability options for Oracle Database, see *Oracle Clusterware Administration and Deployment Guide*.

Requirements for Using Standard Edition High Availability With Oracle Databases

To use Standard Edition High Availability, deploy Oracle Database Standard Edition 2 in accordance with these configuration requirements.

- The database is created in a cluster running Oracle Grid Infrastructure for a Standalone Cluster, with its database files placed in Oracle Automatic Storage Management (Oracle ASM) or Oracle Advanced Cluster File System (Oracle ACFS).
- When using the Database Configuration Assistant, do not create a listener when creating an Oracle Database Standard Edition 2 database that you want to configure for Standard Edition High Availability.
- Register the database with Single Client Access Name (SCAN) listeners as remote listeners, and node listeners as the local listener.
- Create a database service. Use this service, instead of the default database service, when you connect applications or database clients to the database.
- Ensure that the server parameter file (`spfile`) and password file are on Oracle ASM or Oracle ACFS. If the `spfile` and password file were placed on a local file system when the database was created or configured, then move these files to Oracle ASM or Oracle ACFS.

Refer to the database installation documentation for additional requirements that must be met.

Related Topics

- *Oracle Database Installation Guide for Linux*

Enabling Standard Edition High Availability for Oracle Databases

You enable Standard Edition High Availability to provide cluster-based failover for an Oracle Database Standard Edition 2 database.

> **Note:**
>
> The steps in this topic must be performed after you install the Oracle Database software binaries to configure Standard Edition High Availability, as described in the *Oracle Database Installation Guide for Linux*, and create a database. If you have an existing Standard Edition 2 database that runs on one cluster node, and you want to enable Standard Edition High Availability for this database, you need to add a node to the configuration.

> **Note:**
>
> Skip this topic, if you use the following topic *Create Standard Edition High Availability Database Using DBCA*

Prerequisites

- Ensure that the initialization parameter `local_listener` is not set. This is to enable node listeners to be automatically used and database connections to be made over the node virtual IP address.

  Use the following command to display the current listener configuration:

  ```
  SQL> SHOW PARAMETER LOCAL_LISTENER;
  ```

  If the output of the above command shows a local listener name, then reset the local listener using the following command:

  ```
  SQL> ALTER SYSTEM RESET LOCAL_LISTENER SCOPE = BOTH;
  ```

  The database must be restarted for the listener configuration change to take effect. However, if the database is relocated to another node, as part of verifying the Standard Edition High Availability configuration, then the database need not be restarted.
- When the database files are stored in Oracle Advanced Cluster File System (Oracle ACFS), the Oracle ACFS file system must be registered in Oracle Clusterware and the dependency of the database resource on the corresponding Oracle ACFS resources must be configured using `srvctl` commands. For example:

  ```
  $ srvctl modify database -db se2cdb2 -acfspath /u01/app/oradata
  ```

To enable Standard Edition High Availability for an Oracle Database Standard Edition 2 database:

1. If the database was created using the `CREATE DATABASE` command, then register the database with Oracle Clusterware.

   Databases created using the `CREATE DATABASE` command are not automatically registered with Oracle Clusterware. Use the `srvctl add database` command to register the database.

   For example, the following command registers the single instance database, whose unique name is `se2cdb`, with the nodes `node2` and `node3`:

   ```
   $ srvctl add database -db se2cdb -oraclehome $ORACLE_HOME
   -dbtype SINGLE -spfile +DATA/SE2CDB1/PARAMETERFILE/spfile.276.1030845691
   -node node2,node3
   ```

2. If the database is already registered with Oracle Clusterware, use the `srvctl modify database` command to enable Standard Edition High Availability.

   For example, the following command enables Standard Edition High Availability on a Standard Edition 2 database whose database unique name is `se2cdb`:

   ```
   $ srvctl modify database -db se2cdb -node node2,node3
   ```

> **Note:**
>
> The `-fixed` option to create a fixed single instance database is not supported with Standard Edition High Availability.

After you enable Standard Edition High Availability for a database:

- When there is an unplanned outage of the node which runs the database instance, the instance is restarted on an available node in the configured node list.
- When there is an unplanned termination of the database instance, an attempt is made to restart the instance on the current node. If the restart fails, a failover is initiated to an available node in the configured node list.
- When the node which runs the database instance completely loses connectivity to the public network, the instance is relocated to an available node in the configured node list.

> **Note:**
>
> The order of nodes in the node list determines the node on which the database is started. Oracle Clusterware attempts to start the database on the first node in the node list. If the first node is unavailable, it moves to the next node in the node list.
>
> Oracle Clusterware also uses the order of nodes in the node list to determine the failover target, if the current node fails. Failover targets are considered starting with the first node in the list and until a suitable candidate is found and unless other circumstances in the cluster prevent this order to determine the failover node.

To verify the Standard Edition High Availability configuration, especially its current configured node list, use the `srvctl config database` command. For example:

```
$ srvctl config database -db se2cdb
...
Type: SINGLE
...
Configured nodes:
        node2, node3
```

Notice from the output that the type of database is single, but there are multiple configured nodes. This indicates that Standard Edition High Availability is enabled.

Further verification can be performed by relocating the database to another configured node.

Related Topics

- *Oracle Database Installation Guide for Linux*
- Adding a Node to a Standard Edition High Availability Database
  Adding new nodes to an existing Standard Edition High Availability configuration provides enhanced failover capabilities to your Standard Edition 2 database.

Create Standard Edition High Availability Database Using DBCA

Oracle Database Configuration Assistant (Oracle DBCA) enables you to create a Standard Edition High Availability (SEHA), single-instance Oracle Database.

After you complete installing Standard Edition High Availability database software you can use Oracle DBCA, in either interactive or silent mode, to create a single-instance Oracle Database. For details see *Deploying Standard Edition High Availability*.

A Standard Edition High Availability database:

- is a single instance database with failover capability.
- uses Oracle ASM or Oracle ACFS for the database storage files (NAS is not supported).

DBCA provides these capabilities for the Standard Edition High Availability single instance database:

- DBCA provides an interface to select node list for placement of the Standard Edition High Availability database.
- DBCA creates a service for the Standard Edition High Availability database.
- DBCA creates spfile and password file in Oracle ASM or Oracle ACFS based on db storage location.

NOT_SUPPORTED

Creating a Standard Edition High Availability database with DBCA

1. Start Oracle Database Configuration Assistant (Oracle DBCA) in interactive mode.

   ```
   cd $ORACLE_HOME/bin
    ./dbca
   ```

2. In the **Select Database Operation** screen, select **Create a database** and click **Next.**
3. In the **Select Database Creation Mode** screen, select **Advanced configuration** and click **Next**.
4. In the **Select Database Deployment Type** page, for the **Database type** field select **Oracle SEHA (SI) database.** Select **Automatic** as the Database Management policy, and select an appropriate template for your database. Click **Next**.
5. Select the database template, then Click **Next** .
6. In the **Select List of Nodes** page, select each node you wish to use for your single instance cluster. Click **Next**.
7. On the **Specify Database Identification Details** page, provide a **Global database name** and a **SID prefix.**
8. Respond to the configuration screens and prompts as needed to complete the database creation process. Configuration screens vary depending on the configuration option that you select.

NOT_SUPPORTED

Silent Mode Options for Standard Edition High Availability

`-databaseConfigType SEHA`

`[-SEHAServiceName` *`Service name for the service to be created for Standard Edition High Availability database`*`]`. This option is mandatory when the databaseConfigType is SEHA]

Sample command line for Oracle ASM:

```
dbca -silent -createDatabase -createAsContainerDatabase true
-templateName $ORACLE_HOME/assistants/dbca/templates/General_Purpose.dbc -
gdbname test
-sehaNodeList node1,node2 -storageType ASM -diskgroupName +DATA -
databaseConfigType SEHA
-SEHAServiceName dbtestsvc
```

Sample command line for Oracle ACFS:

```
dbca -silent -createDatabase -createAsContainerDatabase true
-templateName $ORACLE_HOME/assistants/dbca/templates/General_Purpose.dbc -
gdbname testdb
-sehaNodeList node1,node2 -storageType FS -datafileDestination /acfsmount/
oradata -databaseConfigType SEHA
-SEHAServiceName dbtestsvc
```

Relocating a Standard Edition High Availability Database to Another Node

To manage planned outages, you can relocate an Oracle Database Standard Edition 2 database that uses Standard Edition High Availability to another configured node.

The node to which the database is being relocated must be part of the configured node list for this database.

To relocate an active Oracle Database Standard Edition 2 database from its current node to another configured node:

- Use the `srvctl relocate` command.

  This command performs an offline relocation. It shuts down the database on the existing node and then starts it on the new node.

For example, the following command relocates the Standard Edition 2 database named `se2cdb`, that uses Standard Edition High Availability, to the node `node5`:

```
$ srvctl relocate database -db se2cdb -node node5
```

> **Note:**
>
> The `-abort` and `-revert` options of `srvctl relocate database` command are not supported with Standard Edition High Availability.

Adding a Node to a Standard Edition High Availability Database

Adding new nodes to an existing Standard Edition High Availability configuration provides enhanced failover capabilities to your Standard Edition 2 database.

You may need to add nodes in certain scenarios. Assume that you configured two nodes for your Standard Edition High Availability database. Subsequently, a new node is added to the cluster and you want to include this new node in your database configuration. You can do this by adding the node to the Standard Edition High Availability configuration. Another scenario is when you want to enable Standard Edition High Availability for an existing Standard Edition 2 database.

To add a node to a Standard Edition High Availability database:

1. Extend the Oracle Database Oracle home to the new node. The steps depend on the storage used.

   For a local (non-shared) Oracle home:

   a. Log in to the first cluster node (on which you configured Standard Edition High Availability) as the Oracle installation owner user account (`oracle`).

   b. Navigate to the *`ORACLE_HOME`*`/addnode` directory and run the `addnode.sh` script on Linux or `addnode.bat` on Windows.

      For example, to extend the Oracle home to `node3` on Linux:

      ```
      $ ./addnode.sh -silent "CLUSTER_NEW_NODES={node3}"
      ```

   For a shared Oracle home using Oracle ACFS:

   a. Start the Oracle ACFS resource on the new node by running the following command as `root` from the `Grid_home/bin` directory:

      ```
      # srvctl start filesystem -device volume_device [-node node_name]
      ```

> **Note:**
>
> Make sure the Oracle ACFS resources, including Oracle ACFS registry resource and Oracle ACFS file system resource where the Oracle home is located, are online on the newly added node.

b. Log in to the first cluster node (on which you configured Standard Edition High Availability) as the Oracle installation owner user account (`oracle`).

c. Attach the Oracle home on the first cluster node to the node that you want to add.

```
$ $ORACLE_HOME/addnode/addnode.sh -silent
CLUSTER_NEW_NODES=new_node_name
```

2. On the new node (that must be added), as the `root` user, and run the *`ORACLE_HOME`*`/root.sh` script.

```
# /u01/app/oracle/product/21.0.0/dbhome_1/root.sh
```

3. On any configured node, including the node that is being added, as the `oracle` user, add the new node by using the `srvctl modify database` command.

   The `-node` argument must list the existing configured nodes and the new node that is being added.

   For example, the following command adds the node `node3` to the database with unique name `se2cdb`. Existing configured nodes are `node1` and `node2`.

```
$ srvctl modify database -db se2cdb -node node1,node2,node3
```

You can optionally verify the new configuration as described in *Enabling Standard Edition High Availability for Oracle Databases*.

Related Topics

- Enabling Standard Edition High Availability for Oracle Databases
  You enable Standard Edition High Availability to provide cluster-based failover for an Oracle Database Standard Edition 2 database.

Removing a Configured Node from a Standard Edition High Availability Database

Use `srvctl` commands to remove a node from the list of nodes configured for a Standard Edition High Availability database.

Navigate to the Oracle Database home directory. On Linux, log in to the database host computer as the Oracle installation owner user account (`oracle`). On Windows, log in to the database host computer as Administrator.

To remove a configured node from a database that uses Standard Edition High Availability:

1. List the existing configured nodes by using the `srvctl config database` command.

2. If the database is currently running on the node that you want to remove, relocate the database to another configured node by using the `srvctl relocate database` command.
3. Remove the node by using the `srvctl modify database` command with the `-node` argument.

   The `-node` argument must list all the configured nodes, except the node that must be removed.

Example 4-1 Removing a Configured Node from a Standard Edition High Availability Database

The example assumes that the database with unique name `sec2cdb` uses Standard Edition High Availability and the configured nodes are `node1`, `node2`, and `node3`. The database is currently running on `node3`. To remove `node2` from the list of configured nodes for this database, log in as a user who installed the Oracle Database home and run the following command:

```
$ srvctl modify database -db sec2cdb -node node1,node3
```

Related Topics

- Relocating a Standard Edition High Availability Database to Another Node
  To manage planned outages, you can relocate an Oracle Database Standard Edition 2 database that uses Standard Edition High Availability to another configured node.

Starting and Stopping Standard Edition High Availability Databases

Use `srvctl` commands to start or stop an Oracle Database that is configured for Standard Edition High Availability.

Navigate to the Oracle Database home directory. On Linux, log in to the database host computer as the Oracle installation owner user account (`oracle`). On Windows, log in to the database host computer as Administrator.

To start up a Standard Edition High Availability database:

- Use the `srvctl start database` command.

  Optionally, include the `-node` argument to specify the node on which the database must be started.

To stop a Standard Edition High Availability database:

- Use the `srvctl stop database` command

Example 4-2 Starting Up a Standard Edition High Availability Database on a Specified Node

This example starts up a database with the unique name `se2cdb` on the node named `node3`.

```
$ srvctl start database -db sec2cdb -node node3
```

Example 4-3 Stopping a Standard Edition High Availability Database

This example stops a database instance that is configured to use Standard Edition High Availability. The unique name of the database is `sec2cdb`.

```
$ srvctl stop database -db sec2cdb
```

Related Topics

- *Oracle Real Application Clusters Administration and Deployment Guide*

Deactivating Standard Edition High Availability for Oracle Databases

When you deactivate Standard Edition High Availability for a single instance Oracle Database, the database is no longer part of a high availability failover configuration.

To deactivate Standard Edition High Availability for an Oracle Database:

- Use the `srvctl modify` command and include only one node in the `-node` argument.

Example 4-4 Deactivating the Use of Standard Edition High Availability for an Oracle Database

This example deactivates the use of Standard Edition High Availability for the database with unique name `se2cdb` and configures only one node, `node1`, for this database:

```
srvctl modify database -db se2cdb -node node1
```

All previously configured nodes are removed and the database is now a single-instance database that is registered with Oracle Clusterware.

Cloning a Database

This section describes various methods of cloning an Oracle database.

- Cloning a Database with CloneDB in a Non-multitenant Environment
  CloneDB enables you to clone a database in a non-multitenant environment multiple times without copying the data files into several different locations. Instead, CloneDB uses copy-on-write technology, so that only the blocks that are modified require additional storage on disk.
- Cloning a Database in a Multitenant Environment
  You can clone a database in a multitenant environment.
- Cloning a Database with Oracle Automatic Storage Management (Oracle ASM)
  Oracle Automatic Storage Management (Oracle ASM) provides support for cloning a pluggable database (PDB) in a multitenant container database (CDB). Oracle ASM does not support cloning a non-CDB.

Cloning a Database with CloneDB in a Non-multitenant Environment

CloneDB enables you to clone a database in a non-multitenant environment multiple times without copying the data files into several different locations. Instead, CloneDB uses copy-on-write technology, so that only the blocks that are modified require additional storage on disk.

About Cloning a Database with CloneDB

It is often necessary to clone a production database for testing purposes or other purposes.

Common reasons to clone a production database include the following:

- Deployment of a new application, or an update of an existing application, that uses the database
- A planned operating system upgrade on the system that runs the database
- New storage for the database installation
- Reporting
- Analysis of older data

Before deploying a new application, performing an operating system upgrade, or using new storage, thorough testing is required to ensure that the database works properly under the new conditions. Cloning can be achieved by making copies of the production data files in one or more test environments, but these copies typically require large amounts of storage space to be allocated and managed.

With CloneDB, you can clone a database multiple times without copying the data files into several different locations. Instead, Oracle Database creates the files in the CloneDB database using copy-on-write technology, so that only the blocks that are modified in the CloneDB database require additional storage on disk.

Cloning a database in this way provides the following advantages:

- It reduces the amount of storage required for testing purposes.
- It enables the rapid creation of multiple database clones for various purposes.

The CloneDB databases use the data files of a database backup. Using the backup data files ensures that the production data files are not accessed by the CloneDB instances and that the CloneDB instances do not compete for the production database's resources, such as CPU and I/O resources.

> **Note:**
>
> - The instructions in this section about cloning a database with CloneDB are not applicable for a database in a multitenant environment.
> - The CloneDB feature is not intended for performance testing.

> **See Also:**
>
> "Cloning a Database in a Multitenant Environment" for more information about cloning a database in a multitenant environment

Cloning a Database with CloneDB

You can clone a database with CloneDB.

Before cloning a database, the following prerequisites must be met:

- Each CloneDB database must use Direct NFS Client, and the backup of the production database must be located on an NFS volume.

  Direct NFS Client enables an Oracle database to access network attached storage (NAS) devices directly, rather than using the operating system kernel NFS client. This CloneDB database feature is available on platforms that support Direct NFS Client.

  See *Oracle Grid Infrastructure Installation Guide* for your operating system for information about Direct NFS Client.

- At least 2 MB of additional System Global Area (SGA) memory is required to track the modified blocks in a CloneDB database.

  See *Oracle Database Administrator's Guide*.

- Storage for the database backup and for the changed blocks in each CloneDB database is required.

  The storage required for the database backup depends on the method used to perform the backup. A single full RMAN backup requires the most storage. Storage snapshots carried out using the features of a storage appliance adhere to the requirements of the storage appliance. A single backup can support multiple CloneDB databases.

  The amount of storage required for each CloneDB database depends on the write activity in that database. Every block that is modified requires an available block of storage. Therefore, the total storage requirement depends on the number of blocks modified in the CloneDB database over time.

This section describes the steps required to create one CloneDB database and uses these sample databases and directories:

- The Oracle home for the production database `PROD1` is /u01/prod1/oracle.
- The files for the database backup are in /u02/oracle/backup/prod1.
- The Oracle home for CloneDB database `CLONE1` is /u03/clone1/oracle.

To clone a database with CloneDB:

1. Create a backup of your production database. You have the following backup options:
   - An online backup

     If you perform an online backup, then ensure that your production database is in `ARCHIVELOG` mode and that all of the necessary archived redo log files are saved and accessible to the CloneDB database environment.
   - A full offline backup

If you perform a full offline backup, then ensure that the backup files are accessible to the CloneDB database environment.

- A backup that copies the database files

  If you specify `BACKUP AS COPY` in RMAN, then RMAN copies each file as an image copy, which is a bit-for-bit copy of a database file created on disk. Image copies are identical to copies created with operating system commands such as `cp` on Linux or `COPY` on Windows, but are recorded in the RMAN repository and so are usable by RMAN. You can use RMAN to make image copies while the database is open. Ensure that the copied database files are accessible to the CloneDB database environment.

See *Oracle Database Backup and Recovery User's Guide* for information about backing up a database.

2. Create a text initialization parameter file (PFILE) if one does not exist.

   If you are using a server parameter file (SPFILE), then run the following statement on the production database to create a PFILE:

   ```
   CREATE PFILE FROM SPFILE;
   ```

3. Create SQL scripts for cloning the production database.

   You will use one or more SQL scripts to create a CloneDB database in a later step. To create the SQL scripts, you can either use an Oracle-supplied Perl script called clonedb.pl, or you can create a SQL script manually.

   To use the clonedb.pl Perl script, complete the following steps:

   a. Set the following environment variables at an operating system prompt:

      `MASTER_COPY_DIR` - Specify the directory that contains the backup created in Step 1. Ensure that this directory contains only the backup of the data files of the production database.

      `CLONE_FILE_CREATE_DEST` - Specify the directory where CloneDB database files will be created, including data files, log files, control files.

      `CLONEDB_NAME` - Specify the name of the CloneDB database.

      `S7000_TARGET` - If the NFS host providing the file system for the backup and the CloneDB database is a Sun Storage 7000, then specify the name of the host. Otherwise, do not set this environment variable. Set this environment variable only if cloning must be done using storage snapshots. You can use S7000 storage arrays for Direct NFS Client without setting this variable.

   b. Run the clonedb.pl Perl script.

      The script is in the $ORACLE_HOME/rdbms/install directory and has the following syntax:

      ```
      $ORACLE_HOME/perl/bin/perl $ORACLE_HOME/rdbms/install/clonedb.pl
      prod_db_pfile [sql_script1] [sql_script2]
      ```

      Specify the following options:

      `prod_db_pfile` - Specify the full path of the production database's PFILE.

      `sql_script1` - Specify a name for the first SQL script generated by clonedb.pl. The default is crtdb.sql.

sql_script2 - Specify a name for the second SQL script generated by clonedb.pl. The default is dbren.sql.

The clonedb.pl script copies the production database's PFILE to the CloneDB database's directory. It also creates two SQL scripts that you will use to create the CloneDB database.

c. Check the two SQL scripts that were generated by the clonedb.pl Perl script, and make changes if necessary.

d. Modify the initialization parameters for the CloneDB database environment, and save the file.

Change any initialization parameter that is specific to the CloneDB database environment, such as parameters that control SGA size, PGA target, the number of CPUs, and so on. The `CLONEDB` parameter must be set to `TRUE`, and the initialization parameter file includes this parameter. See *Oracle Database Reference* for information about initialization parameters.

e. In SQL*Plus, connect to the CloneDB database with `SYSDBA` administrative privilege.

f. Run the SQL scripts generated by the clonedb.pl Perl script.

For example, if the scripts use the default names, then run the following scripts at the SQL prompt:

```
crtdb.sql
dbren.sql
```

To create a SQL script manually, complete the following steps:

a. Connect to the database with `SYSDBA` or `SYSBACKUP` administrative privilege.

See *Oracle Database Administrator's Guide*.

b. Generate a backup control file script from your production database by completing the following steps:

Run the following SQL statement:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

This statement generates a trace file that contains the SQL statements that create the control file. The trace file containing the `CREATE CONTROLFILE` statement is stored in a directory determined by the `DIAGNOSTIC_DEST` initialization parameter. Check the database alert log for the name and location of this trace file.

c. Open the trace file generated in Step 3b, and copy the `STARTUP NOMOUNT` and `CREATE CONTROLFILE` statements in the trace file to a new SQL script.

d. Edit the new SQL script you created in Step 3c in the following ways:

Change the name of the database to the name of the CloneDB database you are creating. For example, change `PROD1` to `CLONE1`.

Change the locations of the log files to a directory in the CloneDB database environment. For example, change/u01/prod1/oracle/dbs/t_log1.f to /u03/clone1/oracle/dbs/t_log1.f.

Change the locations of the data files to the backup location. For example, change /u01/prod1/oracle/dbs/t_db1.f to /u02/oracle/backup/prod1/t_db1.f.

The following is an example of the original statements generated by the `ALTER DATABASE BACKUP CONTROLFILE TO TRACE` statement:

```
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE "PROD1" NORESETLOGS  ARCHIVELOG
    MAXLOGFILES 32
    MAXLOGMEMBERS 2
    MAXDATAFILES 32
    MAXINSTANCES 1
    MAXLOGHISTORY 292
LOGFILE
  GROUP 1 '/u01/prod1/oracle/dbs/t_log1.f'  SIZE 25M BLOCKSIZE
512,
  GROUP 2 '/u01/prod1/oracle/dbs/t_log2.f'  SIZE 25M BLOCKSIZE
512
-- STANDBY LOGFILE
DATAFILE
  '/u01/prod1/oracle/dbs/t_db1.f',
  '/u01/prod1/oracle/dbs/t_ax1.f',
  '/u01/prod1/oracle/dbs/t_undo1.f',
  '/u01/prod1/oracle/dbs/t_xdb1.f',
  '/u01/prod1/oracle/dbs/undots.dbf'
CHARACTER SET WE8ISO8859P1;
```

The following is an example of the modified statements in the new SQL script:

```
STARTUP NOMOUNT PFILE=/u03/clone1/oracle/dbs/clone1.ora
CREATE CONTROLFILE REUSE DATABASE "CLONE1" RESETLOGS  ARCHIVELOG
    MAXLOGFILES 32
    MAXLOGMEMBERS 2
    MAXDATAFILES 32
    MAXINSTANCES 1
    MAXLOGHISTORY 292
LOGFILE
  GROUP 1 '/u03/clone1/oracle/dbs/t_log1.f'  SIZE 25M BLOCKSIZE
512,
  GROUP 2 '/u03/clone1/oracle/dbs/t_log2.f'  SIZE 25M BLOCKSIZE
512
-- STANDBY LOGFILE
DATAFILE
  '/u02/oracle/backup/prod1/t_db1.f',
  '/u02/oracle/backup/prod1/t_ax1.f',
  '/u02/oracle/backup/prod1/t_undo1.f',
  '/u02/oracle/backup/prod1/t_xdb1.f',
  '/u02/oracle/backup/prod1/undots.dbf'
CHARACTER SET WE8ISO8859P1;
```

If you have a storage level snapshot taken on a data file, then you can replace the RMAN backup file names with the storage snapshot names.

e. After you edit the SQL script, save it to a location that is accessible to the CloneDB database environment.

Make a note of the name and location of the new SQL script. You will run the script in a subsequent step. In this example, assume the name of the script is create_clonedb1.sql

f. Copy the text initialization parameter file (PFILE) from the production database environment to the CloneDB database environment.

For example, copy the text initialization parameter file from /u01/prod1/oracle/dbs to /u03/clone1/oracle/dbs. The name and location of the file must match the name and location specified in the `STARTUP NOMOUNT` command in the modified SQL script. In the example in Step 3d, the file is /u03/clone1/oracle/dbs/clone1.ora.

g. Modify the initialization parameters for the CloneDB database environment, and save the file.

Add the `CLONEDB` parameter, and ensure that this parameter is set to `TRUE`. Change any other initialization parameter that is specific to the CloneDB database environment, such as parameters that control SGA size, PGA target, the number of CPUs, and so on. See *Oracle Database Reference* for information about initialization parameters.

h. In SQL*Plus, connect to the CloneDB database with `SYSDBA` administrative privilege.

i. Run the SQL script you saved in Step 3e.

For example, enter the following in SQL*Plus:

```
@create_clonedb1.sql
```

j. For each data file in the backup location, run the `CLONEDB_RENAMEFILE` procedure in the `DBMS_DNFS` package and specify the appropriate location in the CloneDB database environment.

For example, run the following procedure if the backup data file is /u02/oracle/backup/prod1/t_db1.f and the CloneDB database data file is /u03/clone1/oracle/dbs/t_db1.f:

```
BEGIN
  DBMS_DNFS.CLONEDB_RENAMEFILE(
    srcfile  => '/u02/oracle/backup/prod1/t_db1.f',
    destfile => '/u03/clone1/oracle/dbs/t_db1.f');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_DNFS` package.

4. If you created your CloneDB database from an online backup, then recover the CloneDB database. This step is not required if you performed a full offline backup or a `BACKUP AS COPY` backup.

For example, run the following SQL statement on the CloneDB database:

```
RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL CANCEL;
```

This statement prompts for the archived redo log files for the period when the backup was performed.

5. Open the database by running the following SQL statement:

```
ALTER DATABASE OPEN RESETLOGS;
```

The CloneDB database is ready for use.

To create additional CloneDB databases of the production database, repeat Steps 3-5 for each CloneDB database.

After Cloning a Database with CloneDB

After a CloneDB database is created, you can use it in almost any way you use your production database. Initially, a CloneDB database uses a minimal amount of storage for each data file. Changes to rows in a CloneDB database cause storage space to be allocated on demand.

You can use the same backup files to create multiple CloneDB databases. This backup can be taken either by RMAN or by storage level snapshots. If you have a storage level snapshot taken on a data file, then you can replace the RMAN backup file names with the storage snapshot names.

You can use the `V$CLONEDFILE` view to show information about each data file in the CloneDB database. This information includes the data file name in the backup, the corresponding data file name in the CloneDB database, the number of blocks read from the backup file, and the number of requests issued against the backup file.

Because CloneDB databases use the backup files as their backend storage, the backup files must be available to each CloneDB database for it to run. If the backup files become unavailable, then the CloneDB databases return errors.

When your use of a CloneDB database is complete, you can destroy the CloneDB database environment. You can delete all of the files in the CloneDB database environment without affecting the production database environment or the backup environment.

> **See Also:**
>
> *Oracle Database Reference* for more information about the `V$CLONEDFILE` view

Cloning a Database in a Multitenant Environment

You can clone a database in a multitenant environment.

Refer to *Oracle Multitenant Administrator's Guide* for more information about cloning a database in a multitenant environment.

Cloning a Database with Oracle Automatic Storage Management (Oracle ASM)

Oracle Automatic Storage Management (Oracle ASM) provides support for cloning a pluggable database (PDB) in a multitenant container database (CDB). Oracle ASM does not support cloning a non-CDB.

See the following guides for more information:

- *Oracle Automatic Storage Management Administrator's Guide*
- *Oracle Multitenant Administrator's Guide*

Dropping a Database

Dropping a CDB involves removing its data files, online redo logs, control files, and initialization parameter files.

> **WARNING:**
>
> Dropping a CDB deletes all its data.

To drop a database:

- Submit the following statement:

```
DROP DATABASE;
```

The `DROP DATABASE` statement first deletes all control files and all other database files listed in the control file. It then shuts down the database instance.

To use the `DROP DATABASE` statement successfully, the database must be mounted in exclusive and restricted mode.

The `DROP DATABASE` statement has no effect on archived redo log files, nor does it have any effect on copies or backups of the database. It is best to use RMAN to delete such files.

If you used the Database Configuration Assistant to create your database, you can use that tool to delete (drop) your database and remove the files.

> **See Also:**
>
> *Oracle Database Administrator's Guide*

5

Configuring 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.
- Purpose of a CDB Fleet
  A CDB fleet provides the database infrastructure for scalability and centralized management of many CDBs.
- 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`.
- 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.

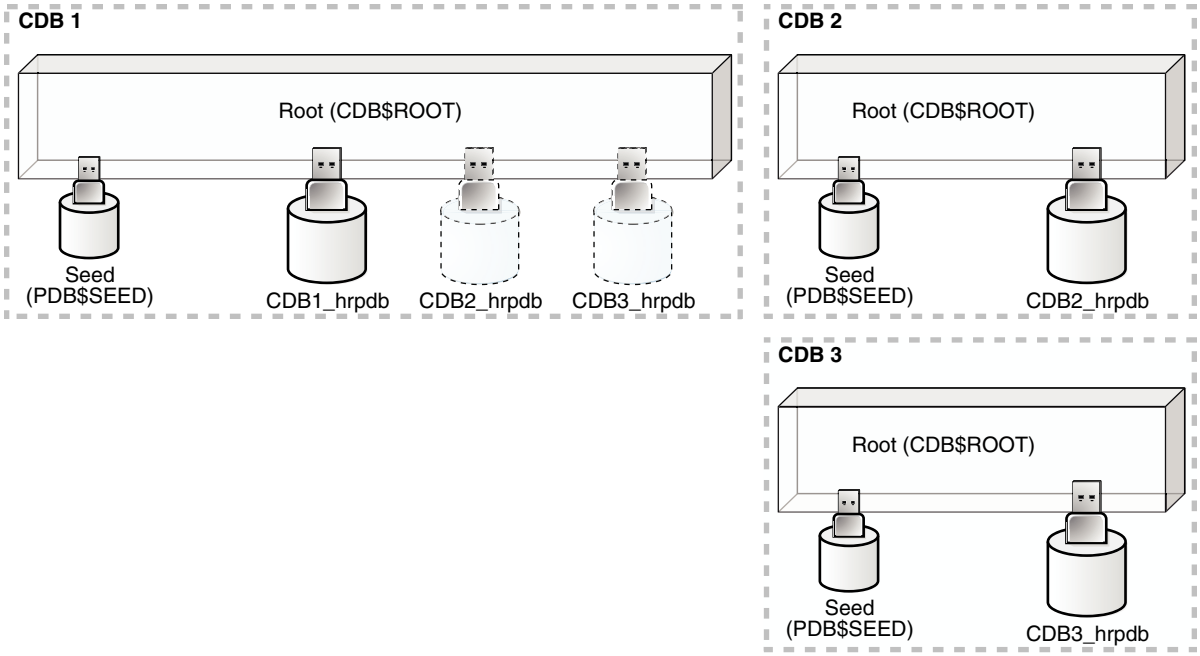
About CDB Fleets

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 5-1 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 Containers in a CDB"
- "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 Containers in a CDB"
> - "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 5-1 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 5-2 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

Part II
Creating PDBs and Application Containers

To create PDBs and application containers, use the `CREATE PLUGGABLE DATABASE` command.

For example, you can create a PDB from scratch, cloning an existing PDB, 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.

- Overview of PDB Creation
  A CDB supports multiple techniques for creating PDBs.
- 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`).
- Cloning a PDB
  You can create a PDB by cloning a local or remote PDB.
- Relocating a PDB
  You can move a PDB to a different CDB or application container.
- Plugging In an Unplugged PDB
  You can create a PDB by plugging an unplugged PDB into 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.
- 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.
- Removing a PDB
  You can remove a plugged-in PDB from a CDB by unplugging it, dropping it, or relocating it.
- 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.

Related Topics

- Tools for a Multitenant Environment
  You can use various tools to configure and administer a multitenant environment.

6

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.

- Current Container and PDB Creation
  You can use the `CREATE PLUGGABLE DATABASE` statement to create PDBs, application containers, application seeds, and application PDBs.
- Techniques for Creating a PDB
  You can create a PDB with various techniques, all of which require the `CREATE PLUGGABLE DATABASE` statement.
- PDB Storage
  However you choose to create a PDB, you must decide on the tablespaces and files that will store the data.
- Service Name Conversion
  An important aspect of PDB creation is managing the renaming of database services.
- 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.
- General Prerequisites for PDB Creation
  Before creating a PDB, you must meet certain prerequisites.

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.

Techniques for Creating a PDB

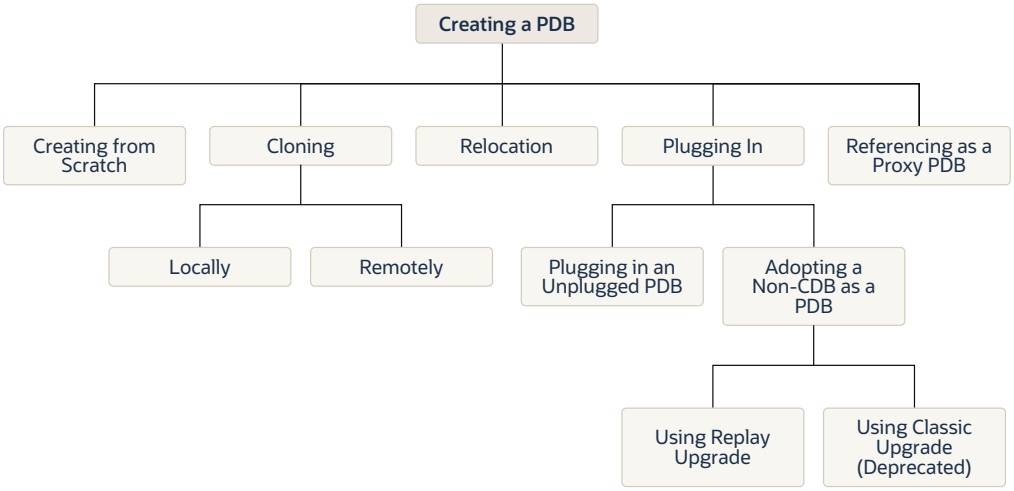
You can create a PDB with various techniques, all of which require the `CREATE PLUGGABLE DATABASE` statement.

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.

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

Figure 6-1 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 6-1 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 PDB or XML.	"Creating a PDB from Scratch"
Clone an existing PDB	Create a PDB by cloning a source PDB. A source can be a PDB in the local CDB, a PDB in a remote CDB, or a PDB in a local or remote application container. 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"
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"
Adopting a non-CDB as a PDB using Replay Upgrade	When adopting a non-CDB from a previous release as a PDB in an Oracle Database 21c CDB, the upgrade occurs automatically when the PDB is opened normally. The Replay Upgrade feature automatically captures necessary `CREATE OR REPLACE` statements, replays the statements only for changed objects, and converts the data dictionary. The replay mechanism is the same one used in application synchronization. If you disable Replay Upgrade by executing `ALTER DATABASE UPGRADE SYNC OFF`, which is not recommended, then you can run `catctl.pl` with the `-t` option for classic upgrade. In this case, you must resolve any compatibility errors manually.	*Oracle Database Upgrade Guide* to learn how to adopt a non-CDB as a PDB using Replay Upgrade

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 and Seeds"
- "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

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.
- Default Tablespace
  The `DEFAULT TABLESPACE` clause of the `CREATE PLUGGABLE DATABASE` statement specifies the default tablespace for the new PDB.
- 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.
- PDB File Locations
  In the `CREATE PLUGGABLE DATABASE` statement, you can specify the locations of files used by the new PDB.

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 6-1 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 6-2 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 assigns the default tablespace to any non-`SYSTEM` users who do not have a different 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 6-3 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.

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 6-4 USER_TABLESPACES Clause That Includes One Tablespace

Assume that the 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 6-5 USER_TABLESPACES Clause That Includes a List of Tablespaces

Assume that the 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 6-6 USER_TABLESPACES Clause That Includes All Tablespaces Except for Listed Ones

Assume that the 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 6-7 USER_TABLESPACES in a Different Creation Mode

This example shows a full CREATE PLUGGABLE DATABASE statement that plugs in a PDB and only includes the tbs3 user tablespace from the PDB. 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 6-2 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.

- 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.
- 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 PATH_PREFIX Clause
  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.
- 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.

> **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 6-8 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 15-34"
> - *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 6-9 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*

The PATH_PREFIX Clause

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.

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 6-10 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:

- "Users, Roles, and Objects in 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 6-11 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 6-3 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

Table 6-3 (Cont.) Clauses for Creating a PDB

Question	Yes	No	Clause Can Be Used Only When
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
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 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 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 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 6-3 (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 6-3 (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 6-3 (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 6-3 (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 6-3 (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 6-3 (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 6-3 (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 6-4 Prerequisites for Creating PDBs

Prerequisite	See Also
The CDB must exist.	"Creating CDBs"
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 User Accounts"
The current user must have the `CREATE PLUGGABLE DATABASE` system privilege.	*Oracle Database Advanced Security Guide* to learn about system privileges

Table 6-4 (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`

7

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.

- 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.
- 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.
- Creating a PDB: Examples
  These examples create a new PDB named `salespdb` and a `salesadm` local administrator given different factors.

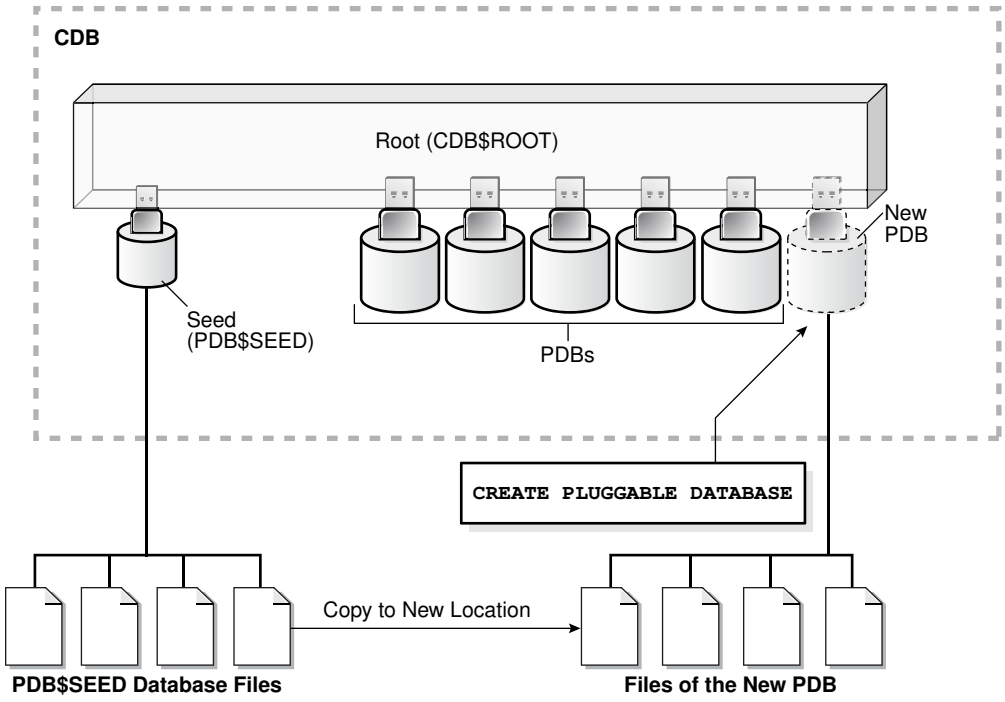
> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about the `CREATE PLUGGABLE DATABASE` statement

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.

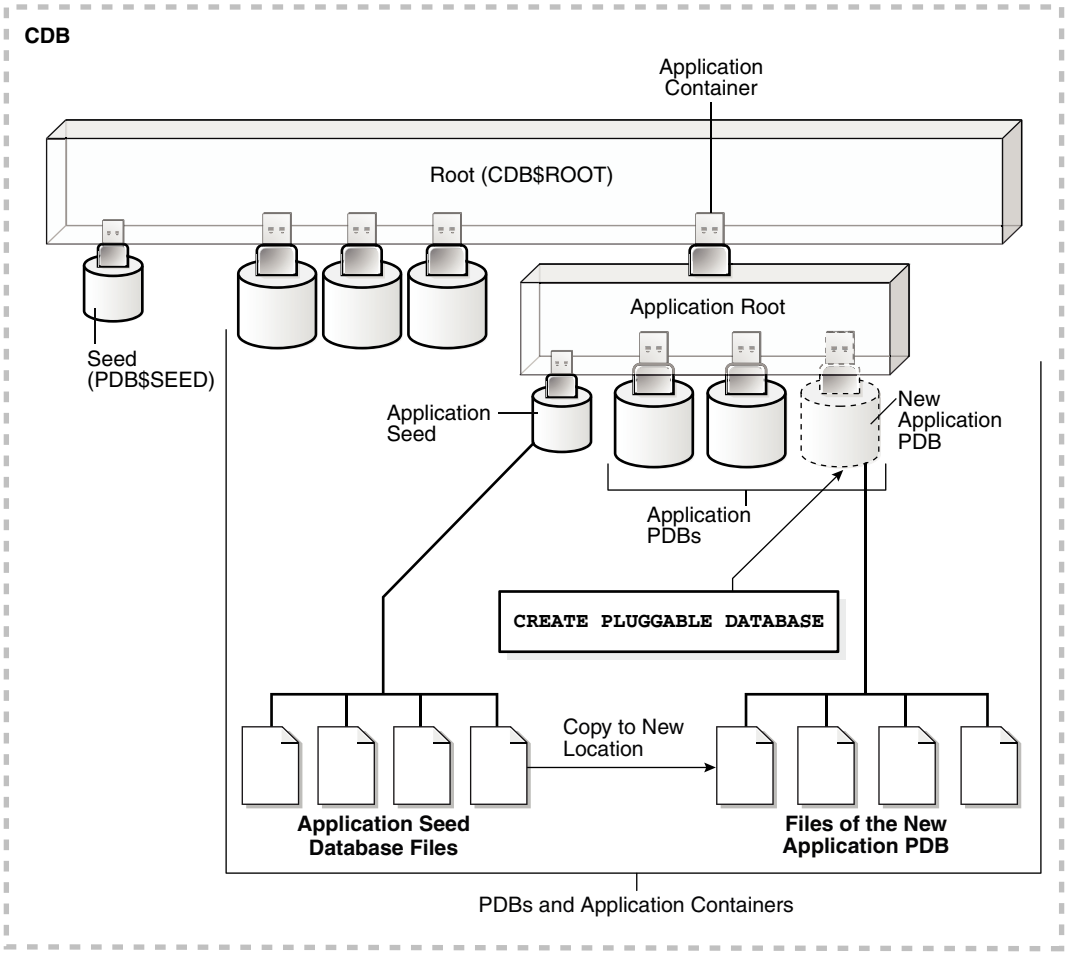
The statement copies these files to a new location and associates them with the new PDB. The following figure illustrates how this technique creates a new PDB in a CDB with the CDB root as the current container.

Figure 7-1 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 7-2 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 6-3. 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.
- Creating a PDB and Granting Predefined Oracle Roles to the PDB Administrator: Example
  This example uses the ROLES parameter to grant a predefined role.
- Creating a PDB Using Multiple Clauses: Example
  This example creating a PDB using the STORAGE, DEFAULT TABLESPACE, PATH_PREFIX, and FILE_NAME_CONVERT clauses.

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 15-34" 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

8

Cloning a PDB

You can create a PDB by cloning a local or remote PDB.

- About Cloning a PDB
  Cloning means creating a new PDB from a source PDB.
- 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.
- 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 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.
- Cloning PDBs from PDB Snapshots
  You can create PDBs from PDB snapshots by executing the `CREATE PLUGGABLE DATABASE … USING SNAPSHOT` statement.
- 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.
- 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.

About Cloning a PDB

Cloning means creating a new PDB from a source PDB.

A typical use case is development testing. You can create one or more clones of a PDB 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.

- How Cloning Works
  This technique creates a new PDB from a source PDB. The process automatically plugs the new PDB into the CDB.
- User Interface for PDB Cloning
  All forms of PDB cloning use the `CREATE PLUGGABLE DATABASE` statement.

> **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. 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 a local or remote PDB.

The target PDB is the copy of the source PDB. The copy is called a clone PDB.

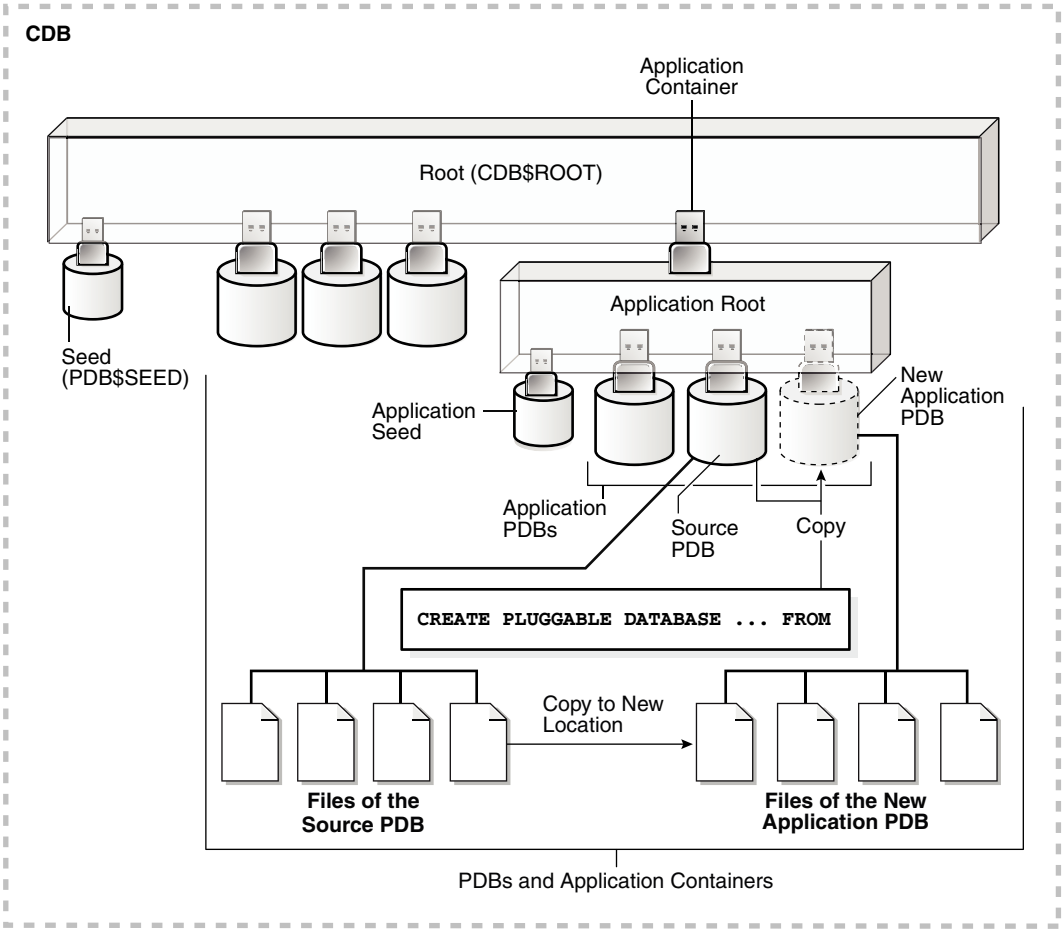
The `CREATE PLUGGABLE DATABASE` statement copies the files associated with the source to a new location and associates the files with the target PDB. When the CDB is in `ARCHIVELOG` mode and local undo mode, the source PDB can be open in read/write mode and operational during the cloning process. This technique is known as **hot cloning**.

> **Note:**
>
> If you clone a PDB, and if that PDB has encrypted data or a TDE master encryption key has been set, you must provide the keystore password of the target keystore by including the `KEYSTORE IDENTIFIED BY` *`keystore_password`* clause in the `CREATE PLUGGABLE DATABASE ... FROM` SQL statement. You must provide the target keystore password so that a check can be made to see if additional keys must be imported before the clone can be used. You can determine whether the source PDB has encrypted data or a TDE master encryption key set in the keystore by querying the `V$ENCRYPTION_KEYS` dynamic 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 8-1 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 8-1 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.	"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.
- 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.
- After Cloning a Local PDB
  Certain rules regarding users and tablespaces apply after cloning a local PDB.

- Cloning a Local PDB: Examples
  The following examples clone a local source PDB named `pdb1` to a target PDB named `pdb2` given different factors.

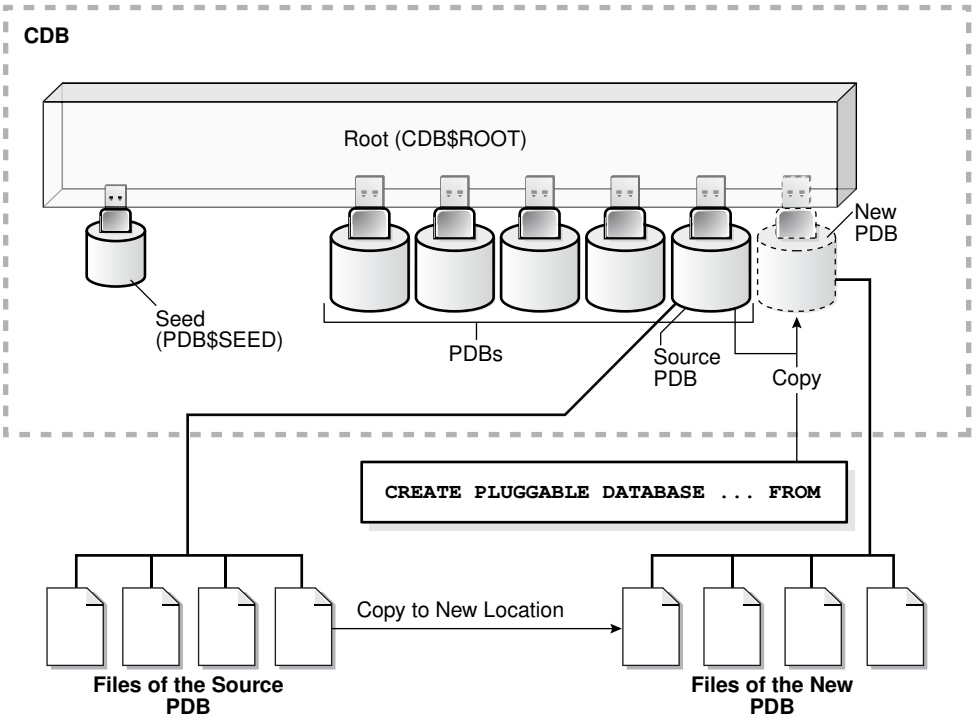
About Cloning a Local PDB

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 8-2 Clone a Local PDB



Before cloning a PDB, address the questions that apply to cloning a PDB in "Table 6-3". 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.
- Cloning a Local PDB Using DBCA: Example
  This example clones a PDB using the silent mode of DBCA. Hot cloning is supported.
- 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.
- 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.
- 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.

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 15-34".
- 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 15-34.
- 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.
- 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.
- After Cloning a Remote PDB
  Certain rules regarding users and tablespaces apply after cloning a remote PDB.
- Cloning a Remote PDB: Examples
  These examples clone a remote PDB given different factors.

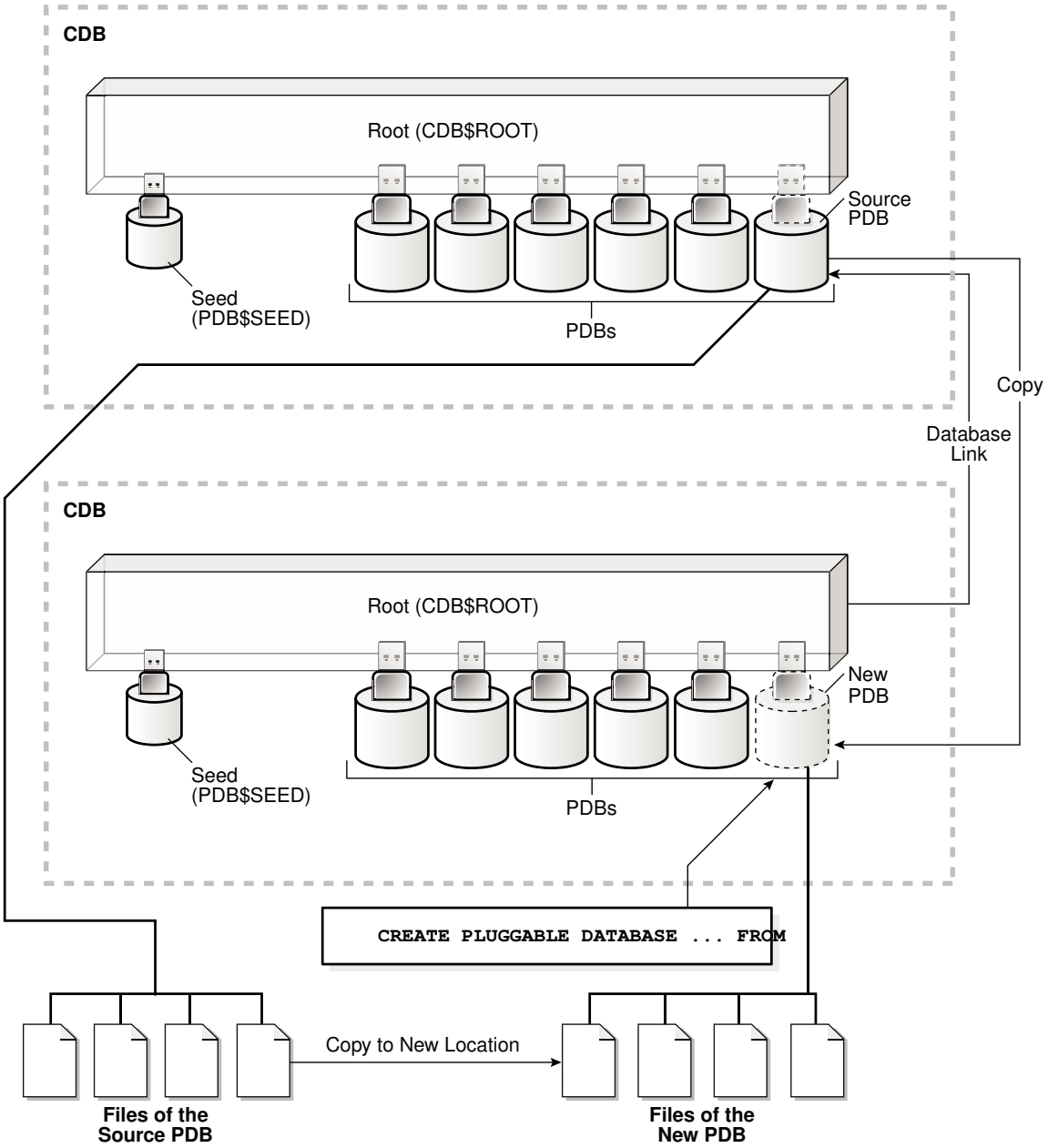
About Cloning a Remote PDB

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 8-3 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`.

   > **Note:**
   >
   > **For the case where the source PDB is open read-only and resides on a standby database:** If the PDB is open on the primary database, and recovery is running on the standby database, and the datafiles of the source PDB are being recovered, then remote cloning of the source PDB is not possible. The media recovery on the source standby database must be stopped first, and then the remote clone of the source PDB can be performed.

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 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.
- 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`.

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 is renamed `clonepdb1`.

This following silent command clones `rempdb1` to `loccdb1`:

```
./dbca -silent
  -createPluggableDatabase
  -createFromRemotePDB
  -sourceDB loccdb1
  -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

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.
- 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.
- 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.

- Creating a Refreshable Clone PDB: Scenario
  This scenario creates a refreshable clone named `pdb1_ref_cln` from a remote PDB named `pdb1`.
- About Creating Refreshable Clone PDBs with DBCA
  Oracle Database Configuration Assistant (DBCA) supports cloning of a remote PDB as a refreshable PDB.
- Creating a Refreshable Clone PDB Using DBCA: Example
  This example uses DBCA to clone a remote PDB named `pdb1` to a refreshable PDB, where it is renamed `refreshpdb1`.

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.
- A change to a tablespace encryption algorithm (for example, from AES128 to AES256) is not applied to a refreshable PDB after the algorithm has been changed in the source PDB. After you create the refreshable PDB, you must update its tablespace encryption algorithm manually.

Example 8-1 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"

About Creating Refreshable Clone PDBs with DBCA

Oracle Database Configuration Assistant (DBCA) supports cloning of a remote PDB as a refreshable PDB.

When a PDB is created as a refreshable PDB, the changes of the source PDB periodically propagate to the refreshable PDB. The refreshable PDB can be configured to refresh manually or automatically during creation. For refreshable PDBs, the database link that connects to the remote database is not dropped at the end. The link is required to perform the refresh operation. After creation, the refreshable PDB is left in MOUNTED mode. This is because the refresh operation works only if the refreshable PDB is closed.

Table 8-2 Silent Mode Options

Option	Description
`-createAsRefreshablePDB` *`true/false`*	Specify `true` to create the pluggable database as a refreshable PDB.
`-refreshMode` *`AUTO|MANUAL`*	Specify the refresh mode of the pluggable database.
`-refreshInterval` *`time_interval`*	Specify the time interval in minutes to perform automatic refresh of the PDB. If no refresh interval is provided, then manual refresh is configured.

Creating a Refreshable Clone PDB Using DBCA: Example

This example uses DBCA to clone a remote PDB named `pdb1` to a refreshable PDB, where it is renamed `refreshpdb1`.

Prerequisites

This scenario assumes the following:

- The user in the local database has the `CREATE PLUGGABLE DATABASE` privilege in the root container.
- The remote PDB is in local undo mode.
- The remote and local PDBs are in `ARCHIVELOG` mode.
- The common user in the remote PDB to whom the database link connects has the `CREATE PLUGGABLE DATABASE`, `SESSION`, and `SYSOPER` privilege.
- The local and remote PDBs 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 is renamed `refreshpdb1`.

This following silent command clones `rempdb1` to `loccdb1`:

```
./dbca -silent
  -createPluggableDatabase
  -createFromRemotePDB
  -sourceDB loccdb1
  -remotePDBName rempdb1
  -remoteDBConnString remcdb1host:1521/reminst
  -remoteDBSYSDBAUserName remSYS
      -remoteDBSYSDBAUserPassword remsyspwd
  -dbLinkUsername c##adminuser_remcdb1
      -dbLinkUserPassword pwd4dblinkusr
  -sysDBAUserName locSYS
      -sysDBAPassword locsyspwd
  -pdbName refreshpdb1
  -createAsRefreshablePDB true
  -refreshMode AUTO
  -refreshInterval 60
```

> **See Also:**
>
> *Oracle Database Administrator's Guide* for syntax and semantics of DBCA commands

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`.

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.

- PDB Snapshot Carousel
  A PDB snapshot carousel is a library of up to 8 snapshots.
- Creation of a PDB with the USING SNAPSHOT Clause
  The `USING SNAPSHOT` clause of the `CREATE PLUGGABLE DATABASE` statement creates an active PDB from a read-only PDB snapshot.

PDB Snapshot Carousel

A PDB snapshot carousel is a library of up to 8 snapshots.

The carousel enables you to clone a PDB to a specific SCN or point in time. A typical use case is to restore a PDB snapshot from the carousel, typically the most recent snapshot, and then recover it to the required SCN or timestamp.

> **See Also:**
>
> "Administering a PDB Snapshot Carousel"

Creation of a PDB with the USING SNAPSHOT Clause

The `USING SNAPSHOT` clause of the `CREATE PLUGGABLE DATABASE` statement creates an active PDB from a read-only PDB snapshot.

To view the available PDB snapshots, query the `DBA_PDB_SNAPSHOTS` data dictionary view. To clone a PDB from a snapshot, specify one of the following values in the `USING SNAPSHOT` clause:

- The unique name of the PDB snapshot
- The PDB snapshot SCN in the following form:

  ```
  USING SNAPSHOT AT SCN scn
  ```

- The PDB snapshot timestamp in the following form:

  ```
  USING SNAPSHOT AT TIME timestamp
  ```

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 snapshot clone PDB.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the syntax and semantics of the `USING SNAPSHOT` clause

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.

Note that on Oracle ACFS, the PDB files do not appear sparse. Instead, they use a storage snapshot so that the snapshot copy files share storage with the source files.

- 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.
- Creating a Snapshot Copy PDB: Scenario
  This scenario create a snapshot copy PDB by specify the `SNAPSHOT COPY` clause in `CREATE PLUGGABLE DATABASE`.
- 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.

See Also:

About Oracle ACFS and Database Data Files in the *Oracle Advanced Cluster File System Guide*

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.
- Restrictions for Snapshot Copy PDBs
  You cannot drop the storage snapshot on which a snapshot copy PDB is based.

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 files share storage with their source. 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.

For storage-managed snapshots, the new snapshot PDB is created and mounted only on the local node where you run the command. For Oracle RAC databases, you must manually mount the new snapshot file system and open the PDB on other nodes.

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 8-2 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.

   If you are creating the PDB in a different CDB, issue the `ALTER PLUGGABLE DATABASE ... PREPARE MIRROR COPY` statement with the `FOR DATABASE` *`database_name`* clause where *`database_name`* is the name of the target CDB.
3. Create a clone PDB from the source PDB by issuing the `CREATE PLUGGABLE DATABASE ... FROM ... USING MIRROR COPY` statement.

   If you are creating the PDB in a different CDB, include the database link to the source CDB in the `FROM` clause. Before issuing the `CREATE PLUGGABLE DATABASE` command, you must create a database link that can connect to the source CDB from where the `ALTER PLUGGABLE DATABASE ... PREPARE MIRROR COPY` command was issued.
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`

9

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.
- 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.
- 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.
- 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.
- Relocating a PDB Using CREATE PLUGGABLE DATABASE
  The `CREATE PLUGGABLE DATABASE ... RELOCATE` statement moves a PDB to a different container.
- Relocating a PDB: Examples
  The examples in this section demonstration relocation using SQL and DBCA.

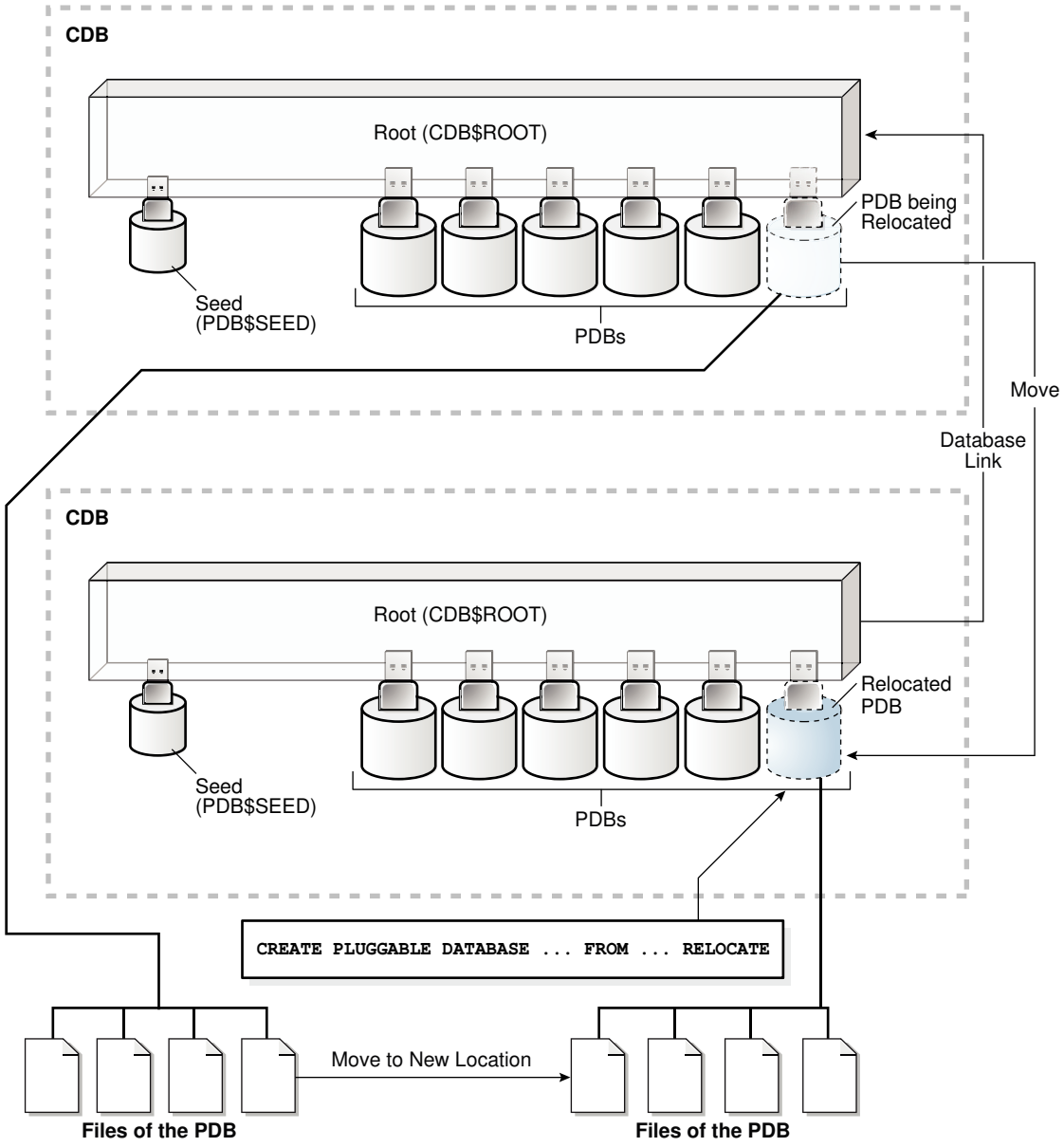
About PDB Relocation

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 9-1 Relocate a PDB into the Root Container

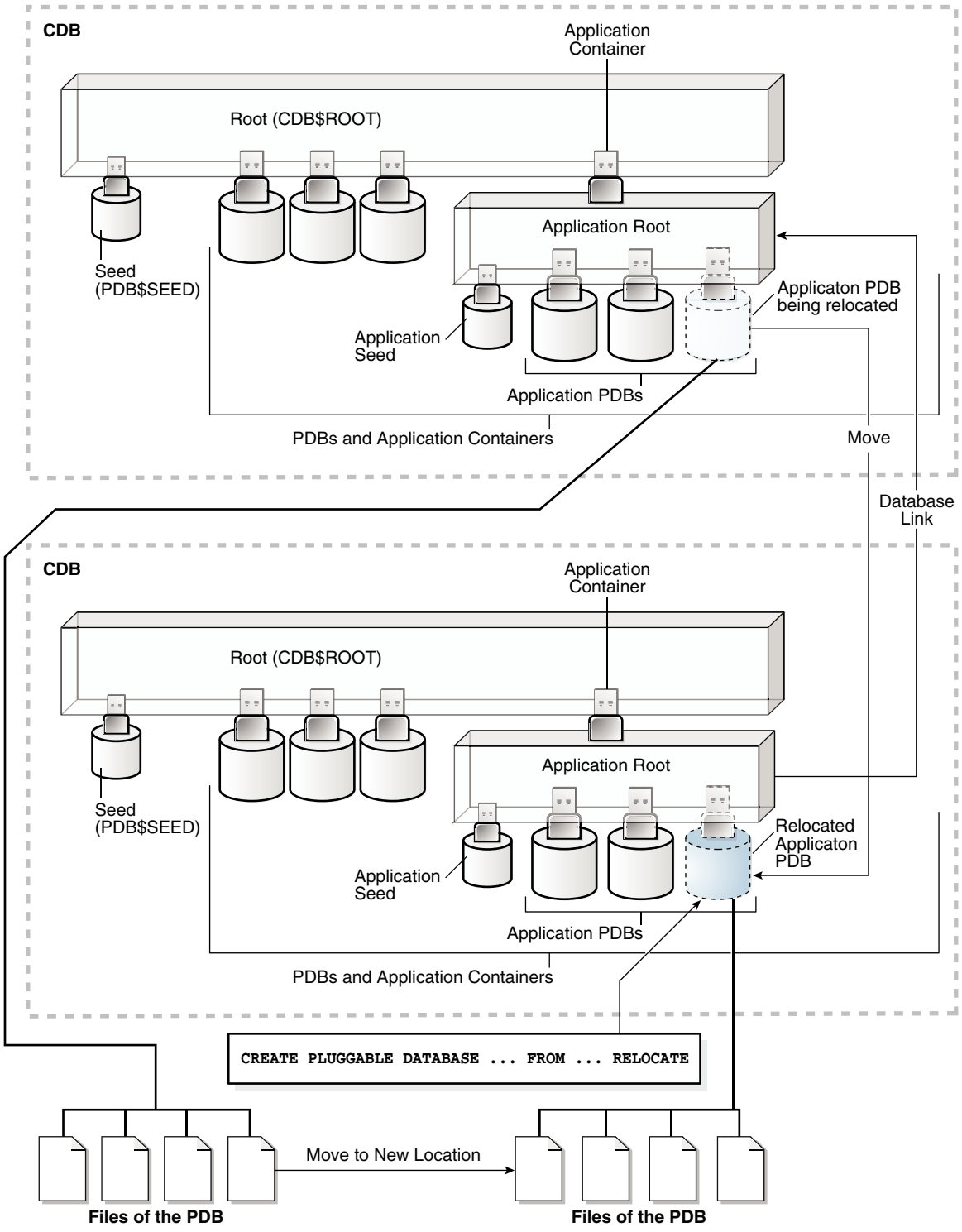


When the target PDB is an application PDB or application root, you have the following options:

- 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 9-2 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.
- Stages of PDB Relocation
  The details of PDB relocation vary depending on the listener networks.

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 21c, 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.
- 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.

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 9-1 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 CDB for the PDB being relocated.
`-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 6-3". 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.
- 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`.

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

10

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).
- Plugging In an Unplugged PDB
  Plug in a PDB with the `CREATE PLUGGABLE DATABASE ... USING` statement.
- After Plugging in an Unplugged PDB
  Certain rules regarding users and tablespaces apply after plugging in an unplugged PDB.
- 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.

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.
- Source File Locations When Plugging In an Unplugged PDB
  Use the `CREATE PLUGGABLE DATABASE ... USING` statement to plug an unplugged PDB into a CDB.

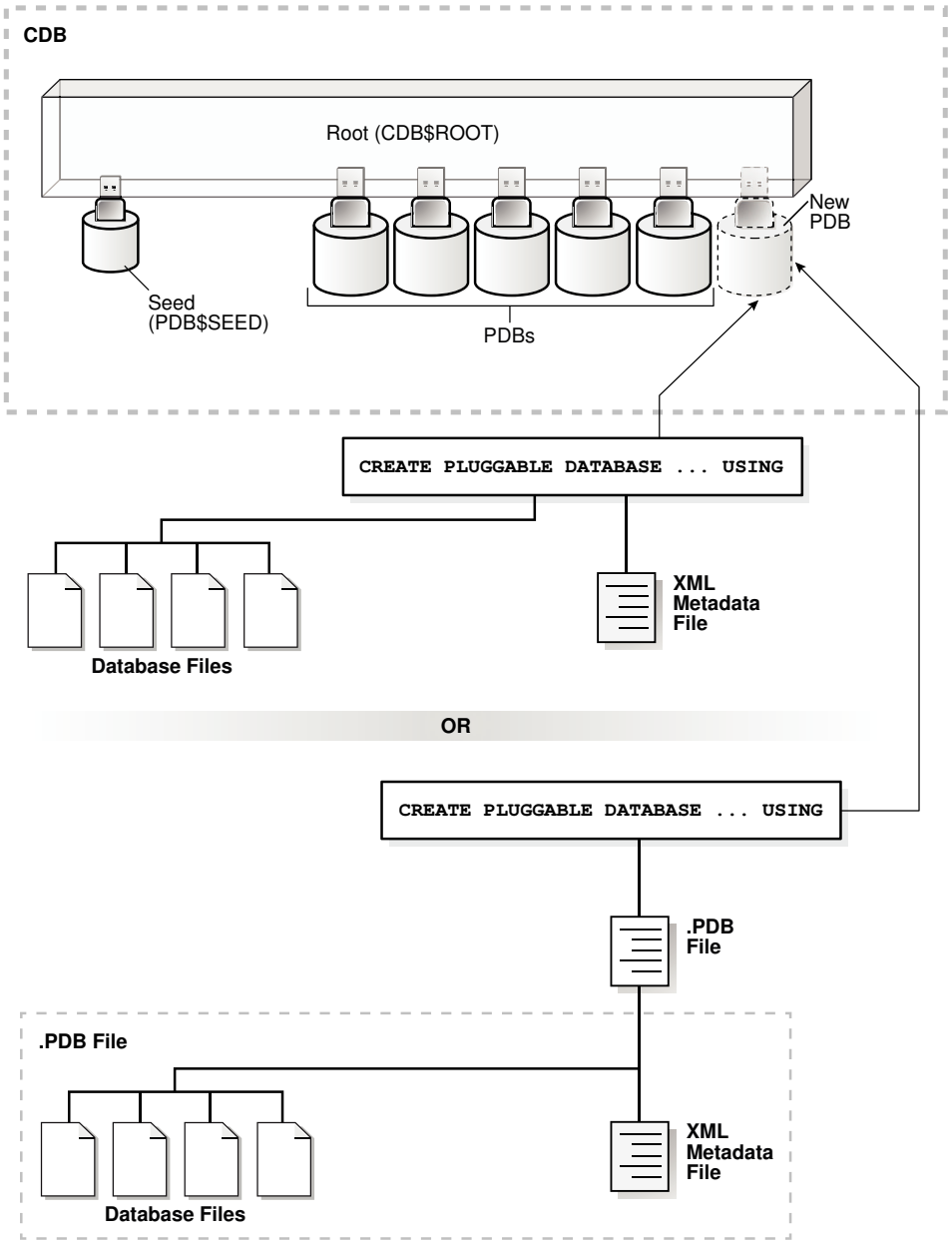
About the XML File and Archive File

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.

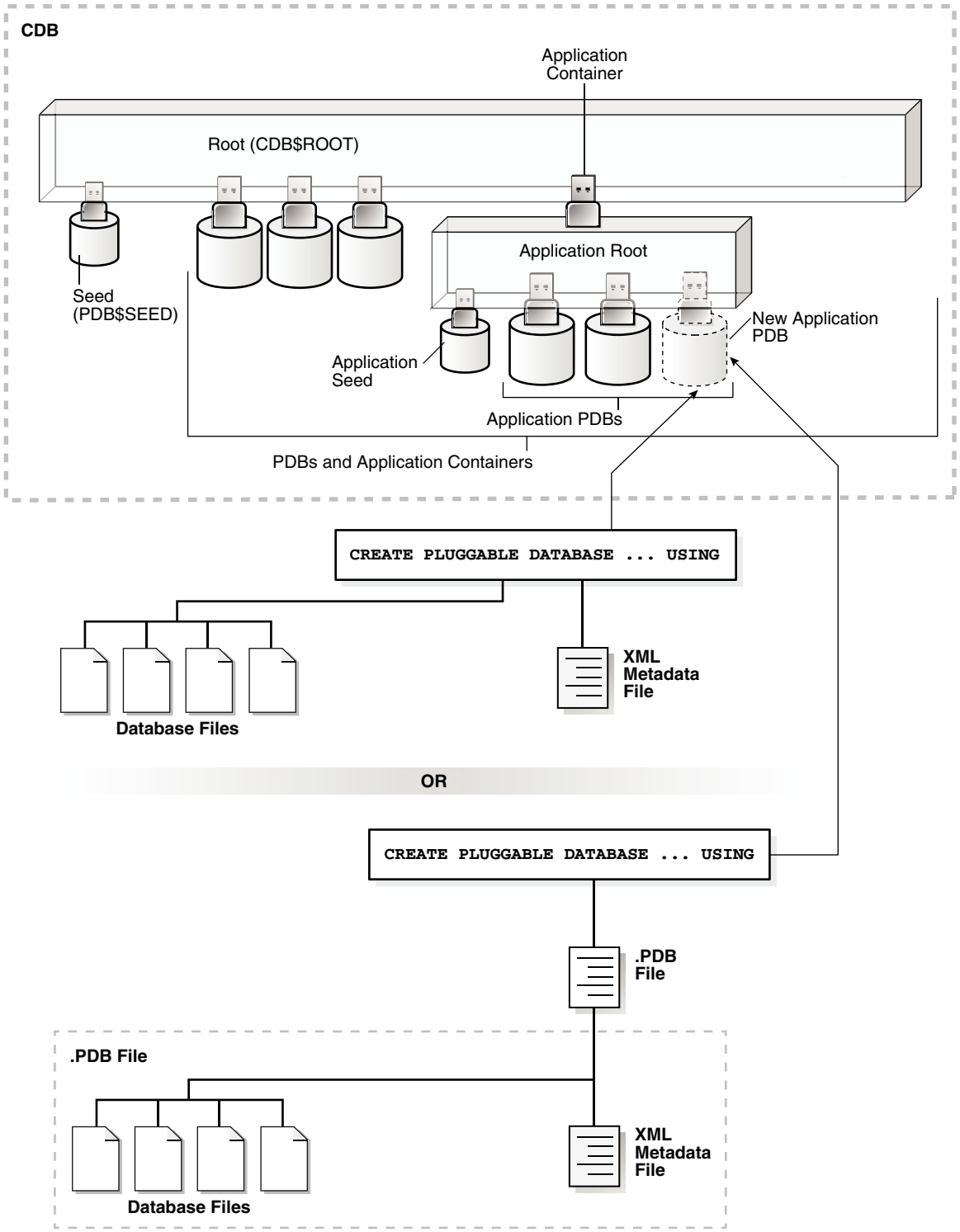
The following figure illustrates how to plug in an unplugged PDB.

Figure 10-1 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 10-2 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 6-3. 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.
- 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.

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 10-1 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 10-2 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

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.

> **Note:**
>
> If you are plugging in a PDB that includes data that was encrypted with Transparent Data Encryption, then follow the instructions in *Oracle Database Advanced Security Guide* for united mode and *Oracle Database Advanced Security Guide* for isolated mode.

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`.

   Opening a PDB upgrades it automatically when a version mismatch occurs between the PDB and the CDB root. The Replay Upgrade on PDB Open optimization, which is the default, avoids manual error correction by re-executing statements stored in capture tables. The mechanism is the same used in application synchronization. When the PDB is opened, the database automatically performs a Replay Upgrade.
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.

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 10-3 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 10-4 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 10-5 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 10-6 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 10-7 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 10-8 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 10-9 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);
```

11

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.
- Creating a Proxy PDB
  Create a proxy PDB by referencing a PDB in a different 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.

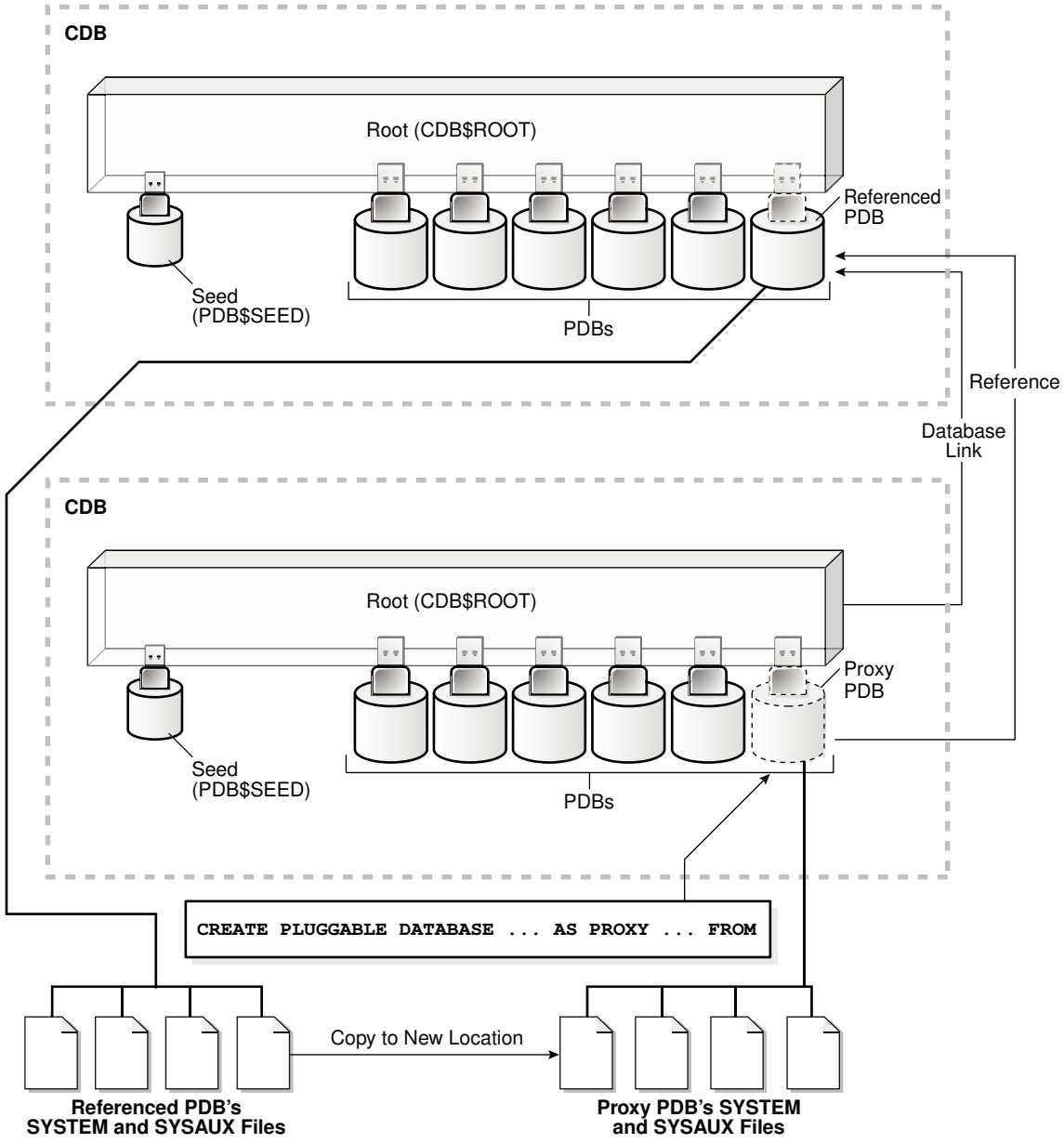
The `CREATE PLUGGABLE DATABASE` statement creates a proxy PDB by referencing a PDB in a different CDB, which is called the **referenced PDB**. You can use a proxy PDB when you want a local context for a remote PDB. In addition, when application containers in different CDBs have the same application, 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.

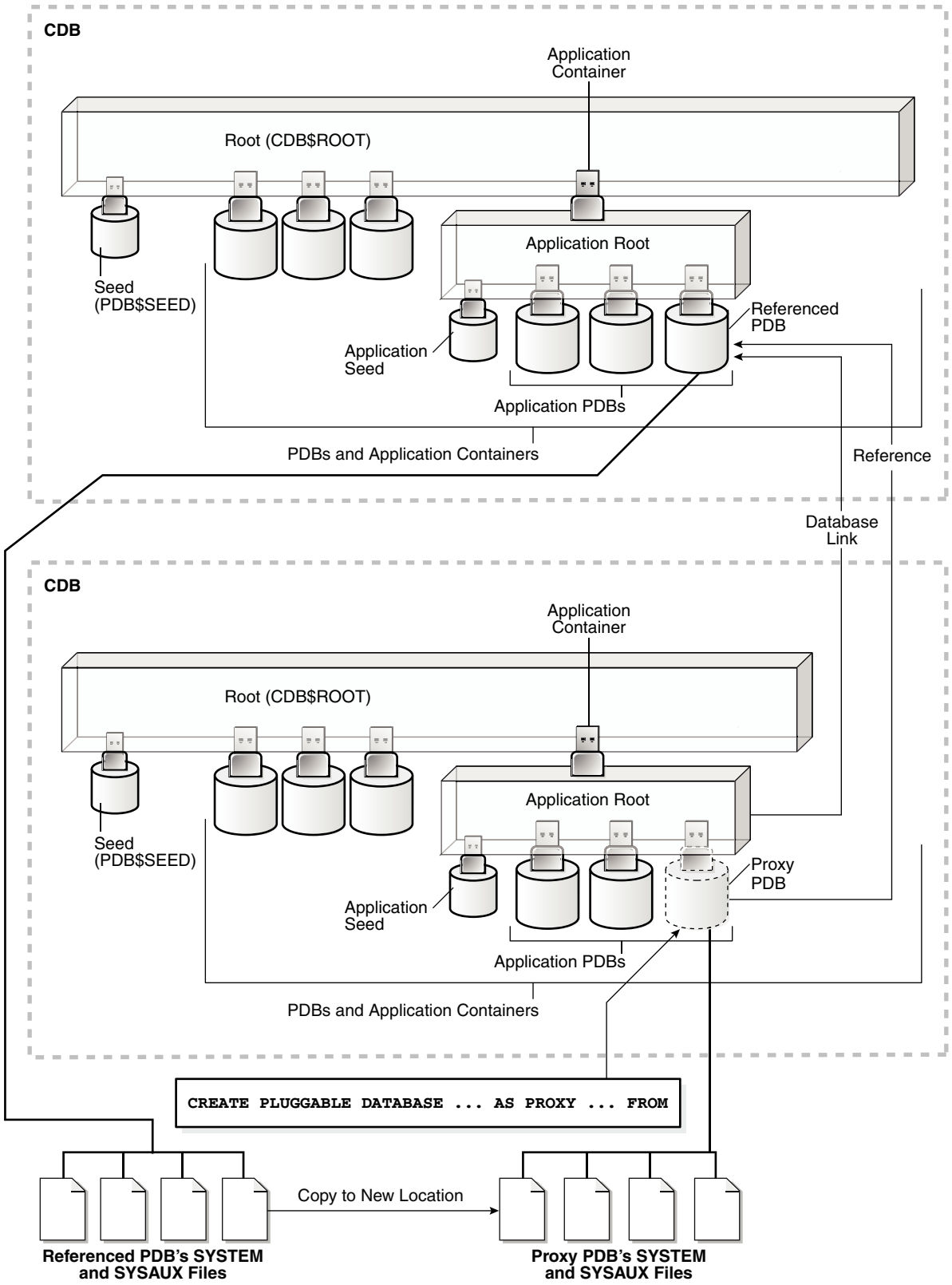
Figure 11-1 Create a Remote Proxy PDB



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 11-2 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 6-3". The table describes which `CREATE PLUGGABLE DATABASE` clauses you must specify based on different factors.

- 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.
- Proxy PDBs and Database Links
  A database link is required when you create a proxy 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.

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.
- PORT Clause
  The `PORT` clause of the `CREATE PLUGGABLE DATABASE` statement specifies the port number of the listener for the PDB being created.

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 11-1 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 11-2 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 11-3 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

12

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.
- Setting the Maximum Number of Snapshots in a PDB Snapshot Carousel
  You can set the maximum number of PDB snapshots for a PDB.
- Configuring Automatic PDB Snapshots
  Configure a PDB for automatic snapshots by using the `SNAPSHOT MODE EVERY` clause when creating or altering a PDB.
- Creating PDB Snapshots Manually
  To create a PDB snapshot manually, specify the `SNAPSHOT` *`snapshot_name`* clause in `ALTER PLUGGABLE DATABASE` or `CREATE PLUGGABLE DATABASE`.
- Dropping a PDB Snapshot
  You can drop a PDB snapshot by running an `ALTER PLUGGABLE DATABASE` statement with the `DROP SNAPSHOT` clause.
- Viewing Metadata for PDB Snapshots
  The data dictionary views `DBA_PDB_SNAPSHOTS` and `DBA_PDB_SNAPSHOTFILE` show the metadata for PDB 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.

- 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.
- How PDB Snapshot Carousel Works
  The carousel for a specific PDB is a circular library of copies for this PDB.
- User Interface for PDB Snapshot Carousel
  The `SNAPSHOT MODE` clause controls creation of snapshots, and determines whether creation is manual, automatic, or disabled.

> **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.

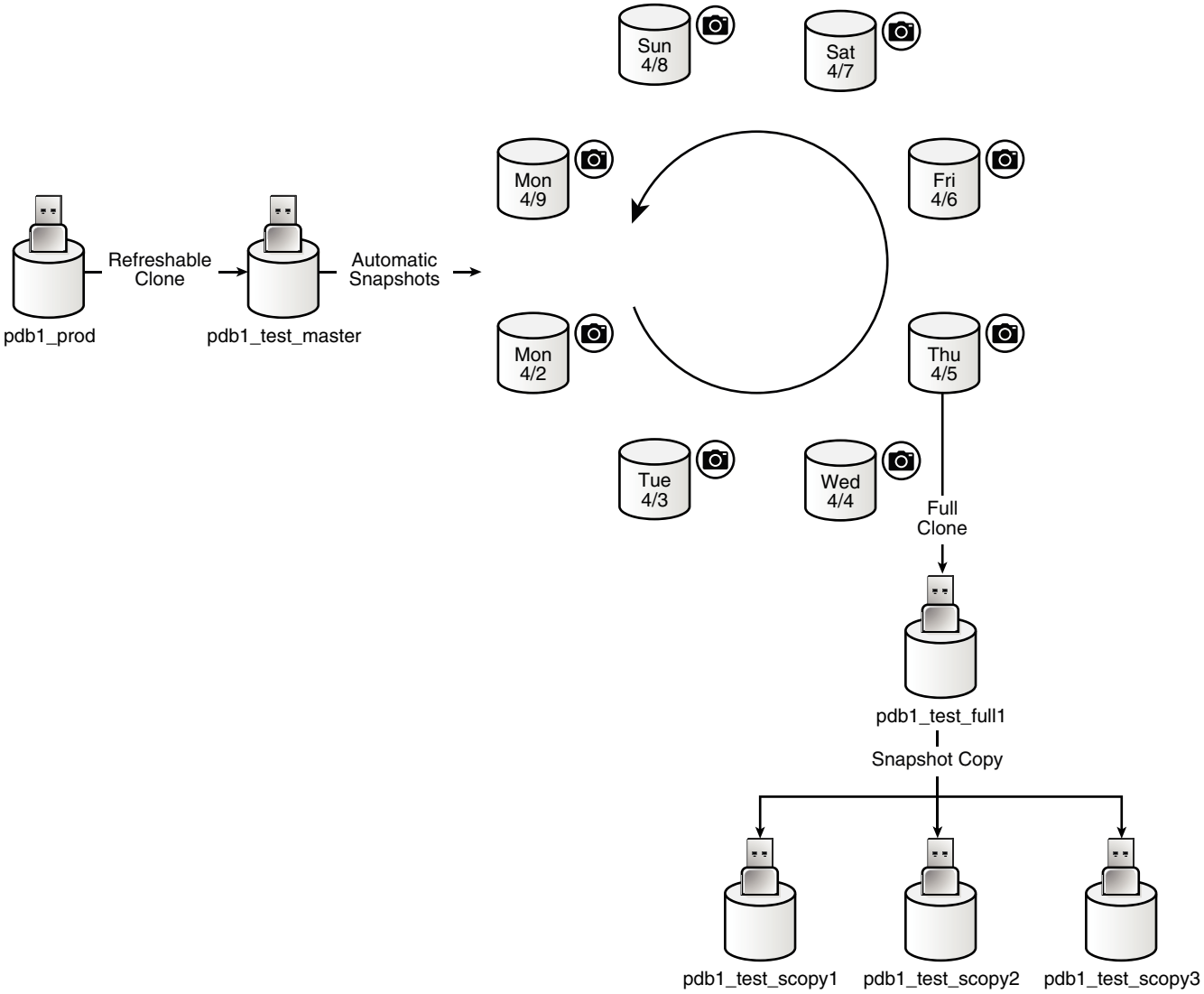
2. Run `ALTER PLUGGABLE DATABASE pdb1_test_master SNAPSHOT MODE EVERY 24 HOURS`, which configures the PDB to generate automatic snapshots of `pdb1_test_master` every day.

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 12-1 Automatic Snapshots of a Refreshable Clone PDB

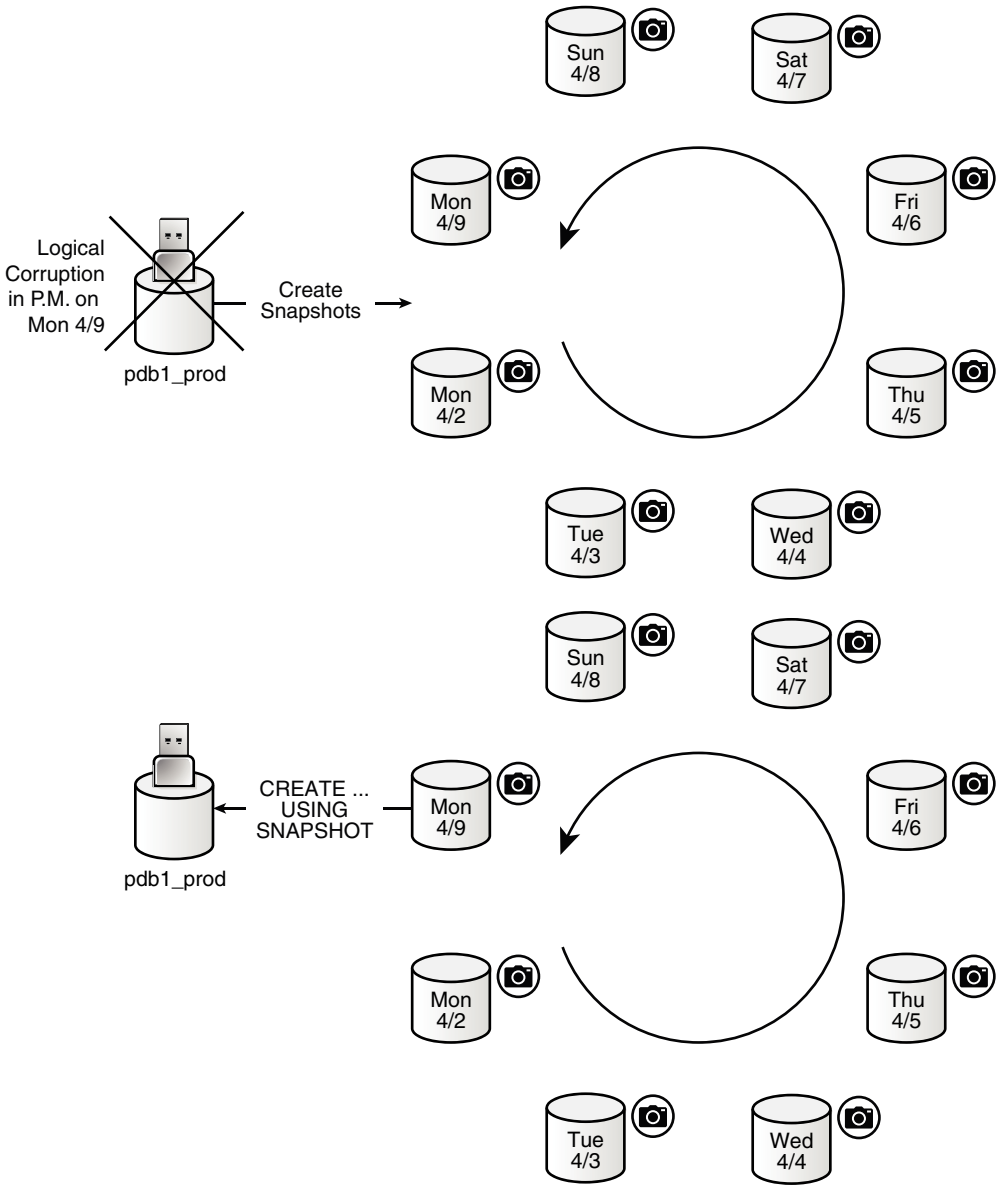


Point-in-Time Restore with PDB Snapshot Carousel

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 12-2 Restore a Production PDB Using a Snapshot



See Also:

- "About Cloning a PDB"
- *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.
- Contents of a PDB Snapshot Carousel
  The PDB snapshot carousel is the set of all existing snapshots for a PDB.

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
```

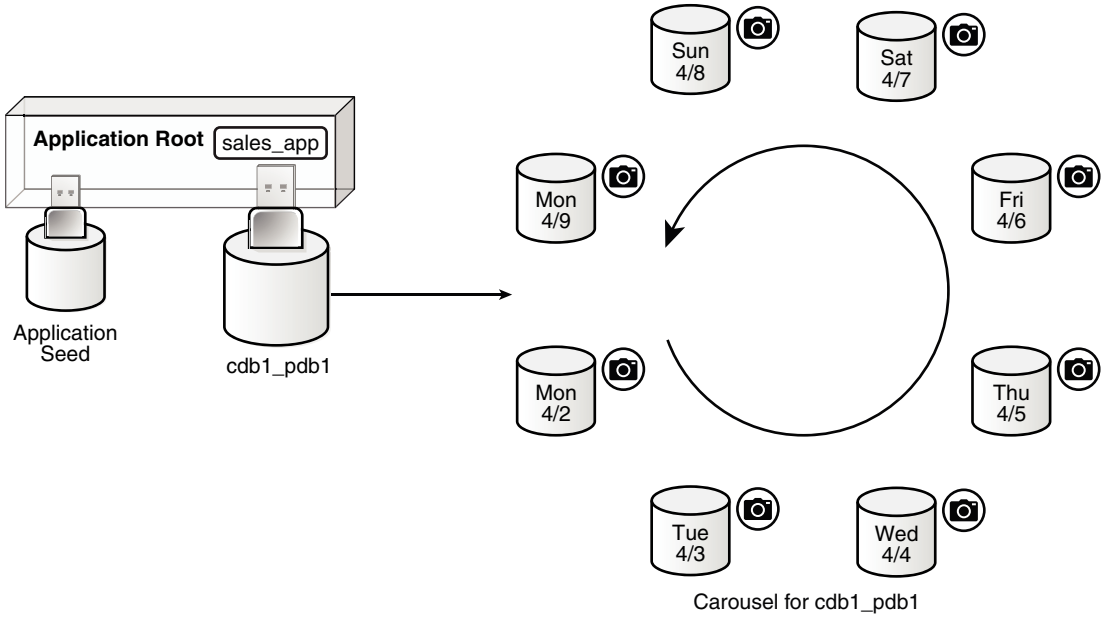
Contents of a PDB Snapshot Carousel

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 12-3 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 12-1 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 12-2 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 12-3 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 12-4 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 12-5 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 12-6 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 12-7 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 12-8 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
```

13

Removing a PDB

You can remove a plugged-in PDB from a CDB by unplugging it, dropping it, or relocating it.

- Unplugging a PDB from a CDB
  Just as you can plug a PDB into a CDB, you can unplug a PDB from a CDB.
- Dropping a PDB
  Drop a PDB when you want to move the PDB to a new CDB or when you no longer need 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.
- Unplugging a PDB
  Unplug a PDB with a `ALTER PLUGGABLE DATABASE ... UNPLUG INTO` statement.

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

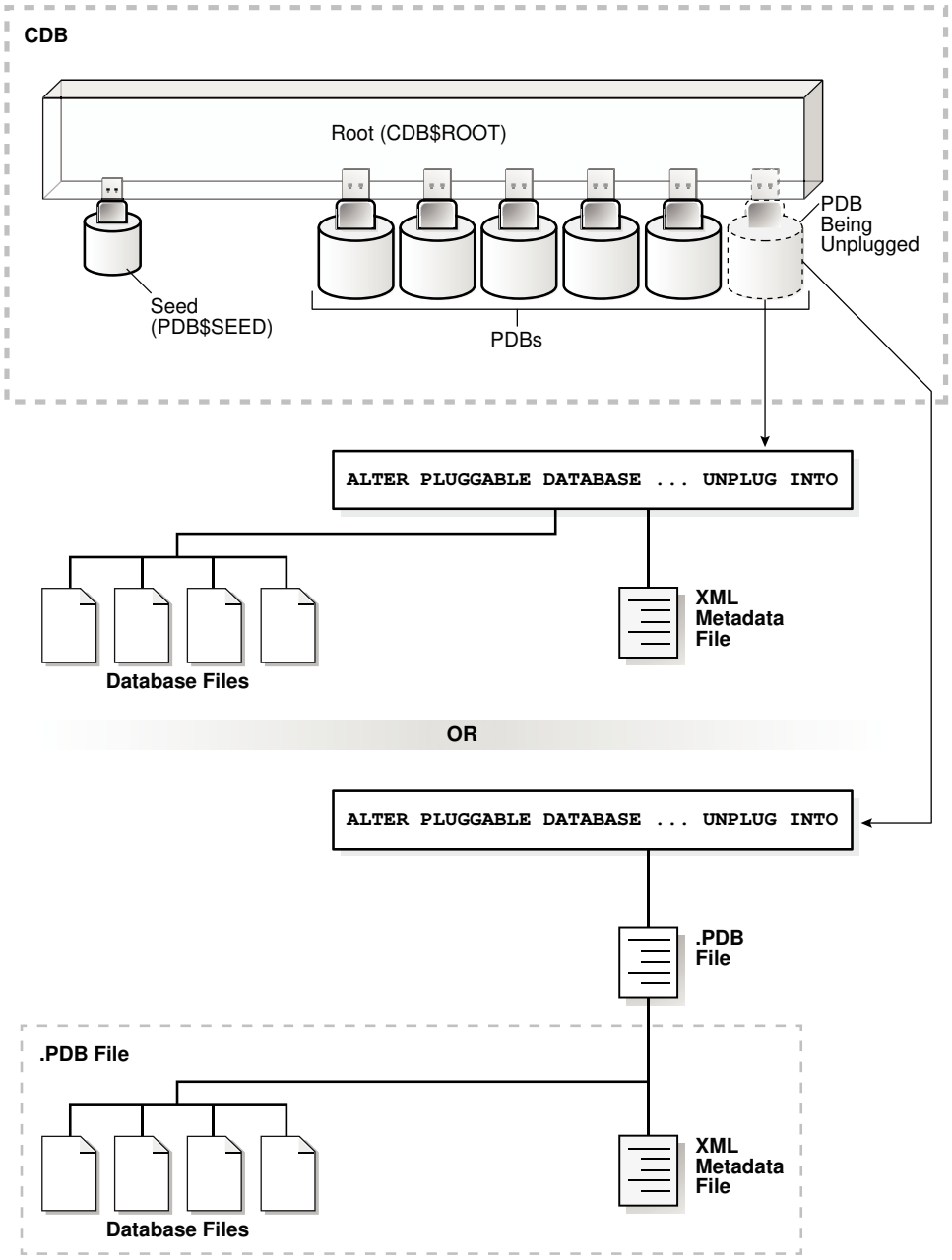
To unplug a PDB, connect to its CDB root or application root and use the `ALTER PLUGGABLE DATABASE` statement to specify either of the following:

- XML file

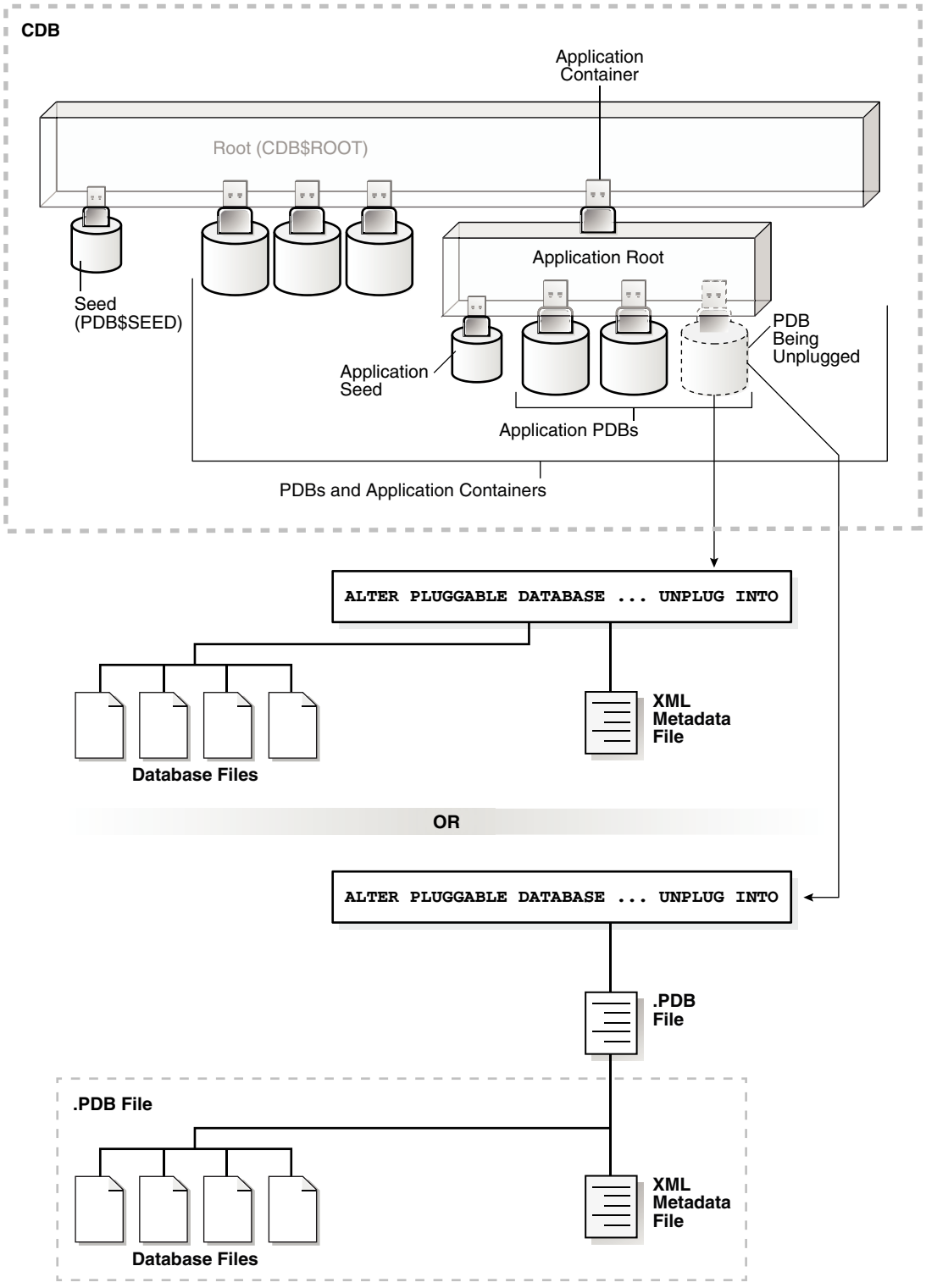
  An XML file (.xml extension) contains metadata about the PDB after it is unplugged. This metadata contains the required information to enable a `CREATE PLUGGABLE DATABASE` statement on a target CDB to plug in the PDB.
- .pdb file

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 13-1 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* for united mode and *Oracle Database Advanced Security Guide* for isolated mode.

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 13-1 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 13-2 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 13-3 Dropping PDB salespdb While Keeping Its Data Files

```
DROP PLUGGABLE DATABASE salespdb
  KEEP DATAFILES;
```

Example 13-4 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

14

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.

- 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.
- Creating Application Containers
  You can create application containers in several different ways, including using the PDB seed, cloning an existing PDB, and plugging in an unplugged PDB by using the `CREATE PLUGGABLE DATABASE` statement.
- Unplugging an Application Container from a CDB
  You can unplug an application container from a CDB.
- 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.
- Creating Application Seeds
  You can create application seeds in several different ways, including using the PDB seed, cloning an existing PDB, and plugging in an unplugged PDB by using the `CREATE PLUGGABLE DATABASE` statement.
- Unplugging an Application Seed from an Application Container
  You can unplug an application seed from an application container.
- 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.
- Creating an Application PDB
  You create an application PDB by running the `CREATE PLUGGABLE DATABASE` statement with an application root as the current container.

See Also:

- "About Application Containers"
- "Administering an Application Container"

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.

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.

For example, you might create multiple sales-related PDBs within one application container, with these PDBs sharing an application 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.

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

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.

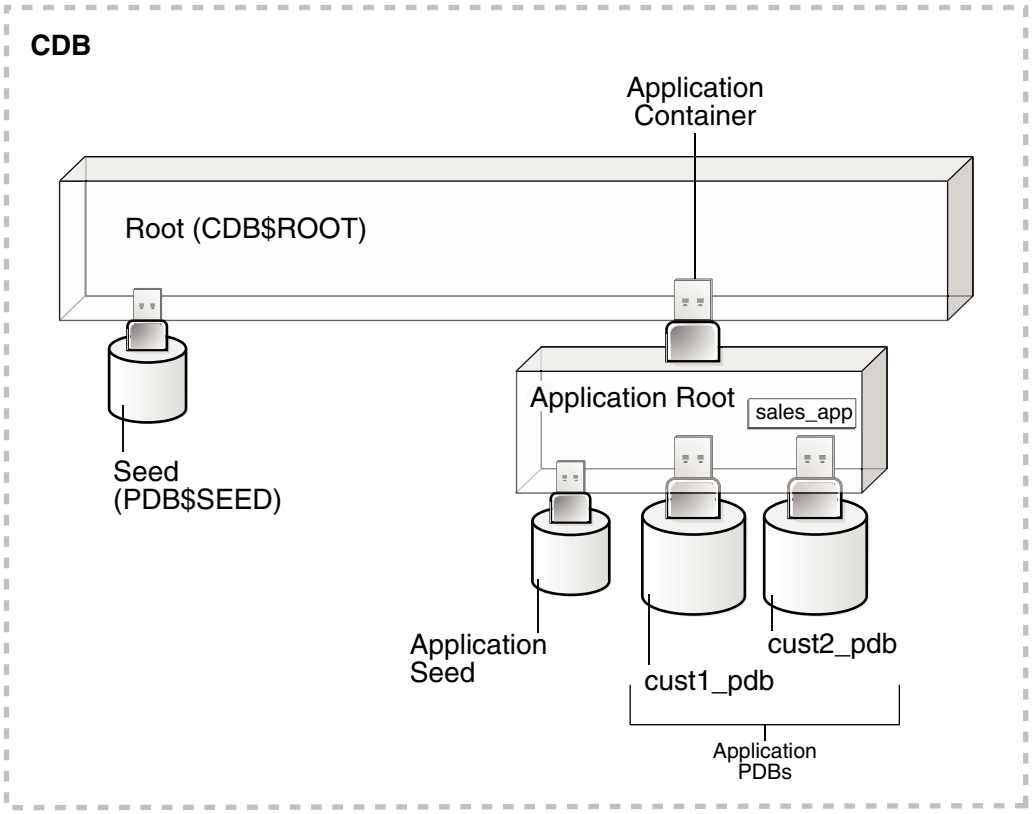
  For example, if the `products` table exists in every application PDB, then you can connect to the application root and query the products in all application PDBs using a single `SELECT` statement.
- You can unplug a PDB from an application root, and then plug it in to an application root in a higher Oracle database release. Thus, PDBs are useful in an Oracle database upgrade.

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 14-1 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.

Example 14-1 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.

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.

Creating Application Containers

You can create application containers in several different ways, including using the PDB seed, cloning an existing PDB, and plugging in an unplugged PDB by using the `CREATE PLUGGABLE DATABASE` statement.

- About Creating an Application Container
  The `CREATE PLUGGABLE DATABASE ... AS APPLICATION CONTAINER` statement creates a new application container.
- Preparing for Application Containers
  Prerequisites must be met before creating an application container.
- Creating an Application Container
  You can create an application container using the `CREATE PLUGGABLE DATABASE` statement with the `AS APPLICATION CONTAINER` clause.

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
- 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 14-2 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 14-3 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 15-34".
- 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 14-4 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 an Application Container
  You can install and 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.
- Unplugging an Application Container
  Unplug an application container by using an `ALTER PLUGGABLE DATABASE ... UNPLUG INTO` statement.

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 14-5 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 14-6 Dropping Application Container salesact While Keeping Its Data Files

```
DROP PLUGGABLE DATABASE salesact
  KEEP DATAFILES;
```

Example 14-7 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 Application Seeds

You can create application seeds in several different ways, including using the PDB seed, cloning an existing PDB, and plugging in an unplugged PDB by using the `CREATE PLUGGABLE DATABASE` statement.

- 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.
- Preparing for an Application Seed
  Prerequisites must be met before creating an application seed.
- Creating an Application Seed
  You create an application seed by including the `AS SEED` clause in the `CREATE PLUGGABLE DATABASE` statement.

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 a CDB: Basic Steps".
- 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 14-8 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 14-9 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 14-10 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
  To unplug an application seed, run the `ALTER PLUGGABLE DATABASE ... UNPLUG INTO` statement.

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 14-11 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 14-12 Dropping Application Seed salesact$SEED While Keeping Its Data Files

```
DROP PLUGGABLE DATABASE salesact$SEED
  KEEP DATAFILES;
```

Example 14-13 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 PDBs and Application Containers
  To create PDBs and application containers, use the `CREATE PLUGGABLE DATABASE` command.
- Administering an Application Container
  You can install and administer the applications installed in application containers.

Part III
Administering a Multitenant Environment

You can administer containers in a multitenant environment using SQL*Plus or Enterprise Manager Cloud Control (Cloud Control).

This manual explains how to administer containers *as* containers, for example, how to create CDBs and PDBs, start them up and shut them down, and perform cross-container operations. *Oracle Database Administrator's Guide* describes traditional administrative tasks that you perform within an existing container, including managing database storage, schema objects, resources, and task scheduling.

- 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.
- Administering PDBs
  Administering PDBs includes tasks such as connecting to a PDB, modifying a PDB, and managing services associated with PDBs.
- Administering an Application Container
  You can install and administer the applications installed in application containers.

15
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.

- About CDB Administration
  Some administrative tasks apply to the entire CDB, whereas others apply to specific containers.
- 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.
- Starting Up and Shutting Down a CDB
  When you start up a CDB, you create an instance and then determine the state of the CDB. Shutting down a currently running Oracle Database instance can optionally close and dismount a CDB.
- 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.
- Modifying Containers When Connected to the CDB Root
  You can modify the entire CDB or the root with the `ALTER DATABASE` statement.
- 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.
- Monitoring Containers in a CDB
  You can view metadata about CDBs, PDBs, and application containers using SQL*Plus or SQL Developer.

> **See Also:**
>
> "Tools for a Multitenant Environment"

About CDB Administration

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.
- About Administrative Tasks in a CDB
  Common users perform administrative tasks for 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.
- 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.
- About Managing Database Objects in a CDB
  In a CDB, different containers can contain different database objects.
- 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).
- About Restricting PDB Users for Enhanced Security
  There are several ways to restrict PDB users for enhanced security.

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"
- *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.

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

The following table describes some CDB administrative tasks and provides pointers to the relevant documentation.

Table 15-1 Administrative Tasks for 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.	" Starting Up and Shutting Down a CDB" 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

Table 15-1 (Cont.) Administrative Tasks for CDBs

Task	Description	Additional Information
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 or more control files.	*Oracle Database Administrator's Guide* for information about managing control files
Managing the online redo log and the archived redo log files	A CDB has one or more online redo log files and one or more 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. Note the following: • 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 15-1 (Cont.) Administrative Tasks for CDBs

Task	Description	Additional Information
Moving data between containers	You can move data between containers within a CDB using the same methods that you would use to move data between 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 a 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:**
>
> *Oracle Database Concepts* 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 15-2 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 15-2 (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 15-2 (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 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 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.	*Oracle Database SQL Tuning Guide*
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

Table 15-2 (Cont.) Manageability Features in a CDB

Manageability Feature	Data Location	Data Visibility	Additional Information
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*
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*

Table 15-2 (Cont.) Manageability Features in a CDB

Manageability Feature	Data Location	Data Visibility	Additional Information
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 15-2 (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 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 Tablespaces in a CDB
  A tablespace can be associated with only one container. Therefore, a tablespace can be associated with the root or with one PDB.
- About Managing Temporary Tablespaces in a CDB
  Each container in a CDB has its own default temporary tablespace (or tablespace group).

About Managing Tablespaces in a CDB

A tablespace can be associated with only one container. Therefore, a tablespace can be associated with the root or with one PDB.

Each container in a CDB must have its own default tablespace, and default tablespaces cannot be shared between containers. Users connected to the container who are not explicitly assigned a tablespace use the default 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 CDB root and PDBs contain schemas, and schemas contain schema objects. The CDB root and PDBs can also contain nonschema objects, such as users, roles, tablespaces, directories, and editions.

The CDB 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 CDB root and in a PDB.

You can create common user accounts in the CDB root to administer PDBs and application containers. User-created common user accounts can create database objects in the CDB root. Oracle recommends that, in the CDB root, schemas owned by user-created common user accounts contain only database triggers and the objects used in their definitions. A user-created common user account can also own any type of local object in a PDB.

You can create local user accounts in a PDB. A local user in a PDB can create schema objects and nonschema objects in the PDB. You cannot create local user accounts in the CDB root.

In a CDB, names are resolved in the context of the dictionary of the user's current container.

See Also:

- "About the Current Container"
- *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.
- 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.

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

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 operating system 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 operating system 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.

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.
- Accessing a Container in a CDB
  Access a container in a CDB with SQL*Plus by issuing a `CONNECT` or `ALTER SESSION` command.

About Container Access in a CDB

You can use SQL*Plus to access the root or a PDB in a CDB.

- Services in a CDB
  Clients access the root or a PDB through database services.
- 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.
- 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.
- How the Multitenant Option Affects Password Files for Administrative Users
  The password information for the local and common administrative users is stored in different locations.

> **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.

> **Important:**
>
> Do not use the default service name; instead, create user-defined services.

> **See Also:**
>
> - "Managing Services for PDBs"
> - "Example 15-36"

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

The password information for the local and common administrative users is stored in different locations.

- **For CDB common 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.
- 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.

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 CDB Root Using the SQL*Plus CONNECT Command
  You can connect to the CDB root in several ways.
- 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.

Connecting to the CDB Root Using the SQL*Plus CONNECT Command

You can connect to the CDB root in several ways.

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 CDB root:

- The user must be a common user.
- The user must be granted `CREATE SESSION` privilege in the CDB 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 15-1 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 15-2 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 15-3 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 15-4 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 CDB, each tenant can have its own PDB. 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 15-5 Switching to the PDB salespdb and Using the salesrep Service

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

Example 15-6 Switching to the PDB salespdb and Using the Default Service

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

Example 15-7 Switching to the CDB Root

```
ALTER SESSION SET CONTAINER = CDB$ROOT;
```

Example 15-8 Switching to the PDB Seed

```
ALTER SESSION SET CONTAINER = PDB$SEED;
```

Example 15-9 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

Starting Up and Shutting Down a CDB

When you start up a CDB, you create an instance and then determine the state of the CDB. Shutting down a currently running Oracle Database instance can optionally close and dismount a CDB.

- Starting Up a CDB
  When you start up a CDB, you create an instance of that database and you determine the state of the database.
- Altering Database Availability
  You can alter the availability of a database. You may want to do this in order to restrict access for maintenance reasons or to make the database read only.
- Shutting Down a CDB
  You can shut down a CDB with SQL*Plus or Oracle Restart.
- Quiescing a CDB
  A quiesced CDB allows only DBA transactions, queries, fetches, or PL/SQL statements.
- Suspending and Resuming a Database
  The `ALTER SYSTEM SUSPEND` statement halts all input and output (I/O) to data files (file header and file data) and control files. The suspended state lets you back up a database without I/O interference. When the database is suspended all preexisting I/O operations are allowed to complete and any new database accesses are placed in a queued state. Use the `ALTER SYSTEM RESUME` statement to resume normal database operations.
- Delaying Instance Abort
  The `INSTANCE_ABORT_DELAY_TIME` initialization parameter specifies the amount of time, in seconds, to delay shutting down a database when an error causes the instance to abort.

> **See Also:**
>
> *Oracle Real Application Clusters Administration and Deployment Guide* for additional information specific to an Oracle Real Application Clusters environment

Starting Up a CDB

When you start up a CDB, you create an instance of that database and you determine the state of the database.

Normally, you start up an instance by mounting and opening the CDB. This operation makes the CDB available for any valid user to connect to and perform typical data access operations.

- About Database Startup Options
  When Oracle Restart is not in use, you can start up a database instance with SQL*Plus, Recovery Manager, or Oracle Enterprise Manager Cloud Control (Cloud Control). If your database is being managed by Oracle Restart, then Oracle recommends starting the database with SRVCTL.
- Specifying Initialization Parameters at Startup
  To start a database instance, the CDB must read instance configuration parameters (the initialization parameters) from either a server parameter file (`SPFILE`) or a text initialization parameter file (PFILE).
- About Automatic Startup of Database Services
  When your database is managed by Oracle Restart, you can configure startup options for each individual database service (service).
- Preparing to Start Up an Instance
  You must perform some preliminary steps before attempting to start an instance of your CDB using SQL*Plus.
- Starting Up an Instance
  You can start up an instance using SQL*Plus or Oracle Restart.

About Database Startup Options

When Oracle Restart is not in use, you can start up a database instance with SQL*Plus, Recovery Manager, or Oracle Enterprise Manager Cloud Control (Cloud Control). If your database is being managed by Oracle Restart, then Oracle recommends starting the database with SRVCTL.

Oracle Database Administrator's Guide for information about Oracle Restart

- Starting Up a Database Using SQL*Plus
  You can start a SQL*Plus session, connect to Oracle Database with administrator privileges, and then issue the `STARTUP` command. Using SQL*Plus in this way is the only method described in detail in this book.
- Starting Up a Database Using Recovery Manager
  You can also use Recovery Manager (RMAN) to execute `STARTUP` and `SHUTDOWN` commands. You may prefer to do this if your are within the RMAN environment and do not want to invoke SQL*Plus.
- Starting Up a Database Using Cloud Control
  You can use Cloud Control to administer your database, including starting it up and shutting it down. Cloud Control combines a GUI console, agents, common services, and tools to provide an integrated and comprehensive systems management platform for managing Oracle products. Cloud Control enables you to perform the functions discussed in this book using a GUI interface, rather than command line operations.
- Starting Up a Database Using SRVCTL
  When Oracle Restart is installed and configured for your database, Oracle recommends that you use SRVCTL to start the database.

Starting Up a Database Using SQL*Plus

You can start a SQL*Plus session, connect to Oracle Database with administrator privileges, and then issue the `STARTUP` command. Using SQL*Plus in this way is the only method described in detail in this book.

- Run the SQL*Plus `STARTUP` command.

Related Topics

- *SQL*Plus User's Guide and Reference*

Starting Up a Database Using Recovery Manager

You can also use Recovery Manager (RMAN) to execute `STARTUP` and `SHUTDOWN` commands. You may prefer to do this if your are within the RMAN environment and do not want to invoke SQL*Plus.

- Run an RMAN `STARTUP` command.

> **See Also:**
>
> *Oracle Database Backup and Recovery Reference* for information about the RMAN `STARTUP` command

Starting Up a Database Using Cloud Control

You can use Cloud Control to administer your database, including starting it up and shutting it down. Cloud Control combines a GUI console, agents, common services, and tools to provide an integrated and comprehensive systems management platform for managing Oracle products. Cloud Control enables you to perform the functions discussed in this book using a GUI interface, rather than command line operations.

- In Cloud Control, start the database instance.

> **See Also:**
>
> The Cloud Control online help

Starting Up a Database Using SRVCTL

When Oracle Restart is installed and configured for your database, Oracle recommends that you use SRVCTL to start the database.

Starting the database instance with SRVCTL ensures that:

- Any components on which the database depends (such as Oracle Automatic Storage Management and the Oracle Net listener) are automatically started first, and in the proper order.

- The database is started according to the settings in its Oracle Restart configuration. An example of such a setting is the server parameter file location.
- Environment variables stored in the Oracle Restart configuration for the database are set before starting the instance.

To start a database instance with SRVCTL:

- Run the `srvctl start database` command.

Oracle Database Administrator's Guide to learn more about `srvctl start database`

Specifying Initialization Parameters at Startup

To start a database instance, the CDB must read instance configuration parameters (the initialization parameters) from either a server parameter file (`SPFILE`) or a text initialization parameter file (PFILE).

The CDB looks for these files in a default location. You can specify nondefault locations for these files, and the method for doing so depends on whether you start the database with SQL*Plus (when Oracle Restart is not in use) or with SRVCTL (when the database is being managed with Oracle Restart).

- About Initialization Parameter Files and Startup
  When you start the database instance, it attempts to read the initialization parameters from an `SPFILE` in a platform-specific default location. If it finds no `SPFILE`, then it searches for a text initialization parameter file.
- Starting Up with SQL*Plus with a Nondefault Server Parameter File
  With SQL*Plus, you can use the `PFILE` clause to start an instance with a nondefault server parameter file.
- Starting Up with SRVCTL with a Nondefault Server Parameter File
  If your database is being managed by Oracle Restart, then you can specify the location of a nondefault SPFILE by setting or modifying the SPFILE location option in the Oracle Restart configuration for the database.

> **See Also:**
>
> " Creating a CDB: Basic Steps" for more information about initialization parameters, initialization parameter files, and server parameter files

About Initialization Parameter Files and Startup

When you start the database instance, it attempts to read the initialization parameters from an `SPFILE` in a platform-specific default location. If it finds no `SPFILE`, then it searches for a text initialization parameter file.

In the platform-specific default location, Oracle Database locates your initialization parameter file by examining file names in the following order:

1. The location specified by the `-spfile` option in the SRVCTL commands `srvctl add database` or `srvctl modify database`

   You can check the current setting with the `srvctl config database` command.

2. spfile*ORACLE_SID*.ora
3. spfile.ora
4. init*ORACLE_SID*.ora

The first three files are SPFILEs and the fourth is a text initialization parameter file. If DBCA created the SPFILE in an Oracle Automatic Storage Management disk group, then the database searches for the SPFILE in the disk group.

When AS COPY is not specified in a CREATE SPFILE statement and the database is defined as a resource in Oracle Clusterware, if you specify both the spfile_name and the FROM PFILE clause, then this statement automatically updates the SPFILE name and location in the database resource. When AS COPY is specified in a CREATE SPFILE statement, the SPFILE is copied, and the database resource is not updated.

> **Note:**
>
> The spfile.ora file is included in this search path because in an Oracle Real Application Clusters environment one server parameter file is used to store the initialization parameter settings for all instances. There is no instance-specific location for storing a server parameter file.

If you (or the Database Configuration Assistant) created a server parameter file, but you want to override it with a text initialization parameter file, then you can do so with SQL*Plus, specifying the PFILE clause of the STARTUP command to identify the initialization parameter file:

```
STARTUP PFILE = /u01/oracle/dbs/init.ora
```

Nondefault Server Parameter Files

A nondefault server parameter file (SPFILE) is an SPFILE that is in a location other than the default location. It is not usually necessary to start an instance with a nondefault SPFILE. However, should such a need arise, both SRVCTL (with Oracle Restart) and SQL*Plus provide ways to do so. These are described later in this section.

Initialization Files and Oracle Automatic Storage Management

A database that uses Oracle Automatic Storage Management (Oracle ASM) usually has a nondefault SPFILE. If you use the Database Configuration Assistant (DBCA) to configure a database to use Oracle ASM, DBCA creates an SPFILE for the database instance in an Oracle ASM disk group, and then causes a text initialization parameter file (PFILE) to be created in the default location in the local file system to point to the SPFILE, as explained in the next section.

> **See Also:**
>
> - "Table 4-1" lists PFILE and SPFILE default names and locations.
> - *Oracle Real Application Clusters Administration and Deployment Guide* for more information about the server parameter file for an Oracle Real Application Clusters environment
> - *Oracle Database Administrator's Guide* for the SRVCTL Command Reference for Oracle Restart

Starting Up with SQL*Plus with a Nondefault Server Parameter File

With SQL*Plus, you can use the `PFILE` clause to start an instance with a nondefault server parameter file.

To start up with SQL*Plus with a nondefault server parameter file:

1. Create a one-line text initialization parameter file that contains only the `SPFILE` parameter. The value of the parameter is the nondefault server parameter file location.

   For example, create a text initialization parameter file `/u01/oracle/dbs/spf_init.ora` that contains only the following parameter:

   ```
   SPFILE = /u01/oracle/dbs/test_spfile.ora
   ```

   > **Note:**
   >
   > You cannot use the `IFILE` initialization parameter within a text initialization parameter file to point to a server parameter file. In this context, you must use the `SPFILE` initialization parameter.

2. Start up the instance pointing to this initialization parameter file.

   ```
   STARTUP PFILE = /u01/oracle/dbs/spf_init.ora
   ```

The `SPFILE` must reside on the database host computer. Therefore, the preceding method also provides a means for a client system to start a database that uses an `SPFILE`. It also eliminates the need for a client system to maintain a client-side initialization parameter file. When the client system reads the initialization parameter file containing the `SPFILE` parameter, it passes the value to the server where the specified `SPFILE` is read.

Starting Up with SRVCTL with a Nondefault Server Parameter File

If your database is being managed by Oracle Restart, then you can specify the location of a nondefault SPFILE by setting or modifying the SPFILE location option in the Oracle Restart configuration for the database.

To start up with SRVCTL with a nondefault server parameter file:

1. Prepare to run SRVCTL as described in *Oracle Database Administrator's Guide*.

2. Enter the following command:

```
srvctl modify database -db db_unique_name -spfile spfile_path
```

where *db_unique_name* must match the DB_UNIQUE_NAME initialization parameter setting for the database.

3. Enter the following command:

```
srvctl start database -db db_unique_name [options]
```

> **See Also:**
>
> *Oracle Database Administrator's Guide* for the SRVCTL Command Reference for Oracle Restart

About Automatic Startup of Database Services

When your database is managed by Oracle Restart, you can configure startup options for each individual database service (service).

If you set the management policy for a service to AUTOMATIC (the default), the service starts automatically when you start the database with SRVCTL. If you set the management policy to MANUAL, the service does not automatically start, and you must manually start it with SRVCTL. A MANUAL setting does not prevent Oracle Restart from monitoring the service when it is running and restarting it if a failure occurs.

In an Oracle Data Guard (Data Guard) environment in which databases are managed by Oracle Restart, you can additionally control automatic startup of services by assigning Data Guard roles to the services in their Oracle Restart configurations. A service automatically starts upon manual database startup only if the management policy of the service is AUTOMATIC and if one of its assigned roles matches the current role of the database.

> **Note:**
>
> When using Oracle Restart, Oracle strongly recommends that you use SRVCTL to create database services.

> **See Also:**
>
> srvctl add service and srvctl modify service in *Oracle Database Administrator's Guide* for the syntax for setting the management policy of and Data Guard roles for a service

Preparing to Start Up an Instance

You must perform some preliminary steps before attempting to start an instance of your CDB using SQL*Plus.

> **Note:**
>
> The following instructions are for installations where Oracle Restart is not in use.

To prepare for starting an instance:

1. Ensure that any Oracle components on which the database depends are started.

   For example, if the CDB stores data in Oracle Automatic Storage Management (Oracle ASM) disk groups, ensure that the Oracle ASM instance is running and the required disk groups are mounted. Also, it is preferable to start the Oracle Net listener before starting the CDB.
2. If you intend to use operating system authentication, log in to the database host computer as a member of the OSDBA group.
3. Ensure that environment variables are set so that you connect to the desired Oracle instance.
4. Start SQL*Plus without connecting to the CDB root:

   ```
   SQLPLUS /NOLOG
   ```
5. Connect to the CDB root as `SYSOPER`, `SYSDBA`, `SYSBACKUP`, or `SYSDG`. For example:

   ```
   CONNECT username AS SYSDBA

   –or–

   CONNECT / AS SYSDBA
   ```

Now you are connected to the CDB root and ready to start up an instance of your database.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* to learn about operating system authentication
> - *Oracle Database Administrator's Guide* for information about setting environment variables to connect to an Oracle instance
> - *Oracle Database Administrator's Guide* if your database is being managed by Oracle Restart
> - *SQL*Plus User's Guide and Reference* for descriptions and syntax for the `CONNECT`, `STARTUP`, and `SHUTDOWN` commands

Starting Up an Instance

You can start up an instance using SQL*Plus or Oracle Restart.

- About Starting Up an Instance
  When Oracle Restart is not in use, you use the SQL*Plus `STARTUP` command to start up an Oracle Database instance. If your database is being managed by Oracle Restart, Oracle recommends that you use the `srvctl start database` command.
- Starting an Instance, and Mounting and Opening a Database
  Normal database operation means that an instance is started and the database is mounted and open. This mode allows any valid user to connect to the database and perform data access operations.
- Starting an Instance Without Mounting a Database
  You can start an instance without mounting a database. Typically, you do so only during database creation.
- Starting an Instance and Mounting a Database
  You can start an instance and mount a CDB without opening it, allowing you to perform specific maintenance operations.
- Restricting Access to an Instance at Startup
  You can start an instance, and optionally mount and open a database, in restricted mode so that the instance is available only to administrative personnel (not general database users).
- Forcing an Instance to Start
  In unusual circumstances, you might experience problems when attempting to start a database instance, and you can force a database instance to start.
- Starting an Instance, Mounting a Database, and Starting Complete Media Recovery
  If you know that media recovery is required, then you can start an instance, mount a database to the instance, and have the recovery process automatically start.
- Automatic Database Startup at Operating System Start
  Many sites use procedures to enable automatic startup of one or more Oracle Database instances and databases immediately following a system start.
- Starting Remote Instances
  If your local Oracle Database server is part of a distributed database, then you might want to start a remote instance and database.

About Starting Up an Instance

When Oracle Restart is not in use, you use the SQL*Plus `STARTUP` command to start up an Oracle Database instance. If your database is being managed by Oracle Restart, Oracle recommends that you use the `srvctl start database` command.

With SQL*Plus and Oracle Restart, you can start a database instance in various modes:

- `NOMOUNT`—Start the instance without mounting a CDB. This does not allow access to the database and usually would be done only for database creation or the re-creation of control files.

- `MOUNT`—Start the instance and mount the CDB, but leave it closed. This state allows for certain DBA activities, but does not allow general access to the database.
- `OPEN`—Start the instance, and mount and open the CDB. This can be done in unrestricted mode, allowing access to all users, or in restricted mode, allowing access for database administrators only.
- `FORCE`—Force the instance to start after a startup or shutdown problem.
- `OPEN RECOVER`—Start the instance and have complete media recovery begin immediately.

> **Note:**
>
> You cannot start a database instance if you are connected to the database through a shared server process.

The following scenarios describe and illustrate the various states in which you can start up an instance. Some restrictions apply when combining clauses of the `STARTUP` command or combining startup options for the `srvctl start database` command.

> **Note:**
>
> It is possible to encounter problems starting up an instance if control files, database files, or online redo logs are not available. If one or more of the files specified by the `CONTROL_FILES` initialization parameter does not exist or cannot be opened when you attempt to mount a database, Oracle Database returns a warning message and does not mount the database. If one or more of the data files or online redo logs is not available or cannot be opened when attempting to open a database, the database returns a warning message and does not open the database.

> **See Also:**
>
> - *SQL*Plus User's Guide and Reference* for details on the `STARTUP` command syntax
> - *Oracle Database Administrator's Guide* for instructions for starting a database that is managed by Oracle Restart

Starting an Instance, and Mounting and Opening a Database

Normal database operation means that an instance is started and the database is mounted and open. This mode allows any valid user to connect to the database and perform data access operations.

The following command starts an instance, reads the initialization parameters from the default location, and then mounts and opens the database.

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`STARTUP`	`srvctl start database -db db_unique_name`

where *db_unique_name* matches the `DB_UNIQUE_NAME` initialization parameter.

Starting an Instance Without Mounting a Database

You can start an instance without mounting a database. Typically, you do so only during database creation.

Use one of the following commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`STARTUP NOMOUNT`	`srvctl start database -db db_unique_name -startoption nomount`

Starting an Instance and Mounting a Database

You can start an instance and mount a CDB without opening it, allowing you to perform specific maintenance operations.

For example, the CDB must be mounted but not open during the following tasks:

- Starting with Oracle Database 12*c* Release 1 (12.1.0.2), putting a database instance in force full database caching mode. For more information, see *Oracle Database Administrator's Guide*.
- Enabling and disabling redo log archiving options. For more information, see *Oracle Database Administrator's Guide*.
- Performing full database recovery. For more information, see *Oracle Database Backup and Recovery User's Guide*.

The following command starts an instance and mounts the database, but leaves the database closed:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`STARTUP MOUNT`	`srvctl start database -db db_unique_name -startoption mount`

Restricting Access to an Instance at Startup

You can start an instance, and optionally mount and open a database, in restricted mode so that the instance is available only to administrative personnel (not general database users).

Use this mode of instance startup when you must accomplish one of the following tasks:

- Perform an export or import of data
- Perform a data load (with SQL*Loader)
- Temporarily prevent typical users from using data
- Perform certain migration or upgrade operations

Typically, all users with the `CREATE SESSION` system privilege can connect to an open database. Opening a database in restricted mode allows database access only to users with both the `CREATE SESSION` and `RESTRICTED SESSION` system privilege. Only database administrators should have the `RESTRICTED SESSION` system privilege. Further, when the instance is in restricted mode, a database administrator cannot access the instance remotely through an Oracle Net listener, but can only access the instance locally from the system that the instance is running on.

The following command starts an instance (and mounts and opens the database) in restricted mode:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`STARTUP RESTRICT`	`srvctl start database -db` *`db_unique_name`* `-startoption restrict`

You can use the restrict mode in combination with the mount, nomount, and open modes.

Later, use the `ALTER SYSTEM` statement to disable the `RESTRICTED SESSION` feature:

```
ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

See Also:

- *Oracle Database Administrator's Guide* to learn how to use the `ALTER SYSTEM` statement to restrict access after you open the database in nonrestricted mode
- *Oracle Database SQL Language Reference* for more information on the `ALTER SYSTEM` statement

Forcing an Instance to Start

In unusual circumstances, you might experience problems when attempting to start a database instance, and you can force a database instance to start.

You should not force a database to start unless you are faced with the following:

- You cannot shut down the current instance with the `SHUTDOWN NORMAL`, `SHUTDOWN IMMEDIATE`, or `SHUTDOWN TRANSACTIONAL` commands.
- You experience problems when starting an instance.

If one of these situations arises, you can usually solve the problem by starting a new instance (and optionally mounting and opening the database) using one of these commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`STARTUP FORCE`	`srvctl start database -db db_unique_name -startoption force`

If an instance is running, the force mode shuts it down with mode `ABORT` before restarting it. In this case, the alert log shows the message "`Shutting down instance (abort)`" followed by "`Starting ORACLE instance (normal)`."

> **See Also:**
>
> "Shutting Down with the Abort Mode" to understand the side effects of aborting the current instance

Starting an Instance, Mounting a Database, and Starting Complete Media Recovery

If you know that media recovery is required, then you can start an instance, mount a database to the instance, and have the recovery process automatically start.

To do so, use one of these commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`STARTUP OPEN RECOVER`	`srvctl start database -db db_unique_name -startoption "open,recover"`

If you attempt to perform recovery when no recovery is required, Oracle Database issues an error message.

Automatic Database Startup at Operating System Start

Many sites use procedures to enable automatic startup of one or more Oracle Database instances and databases immediately following a system start.

The procedures for performing this task are specific to each operating system. For information about automatic startup, see your operating system specific Oracle documentation.

The preferred (and platform-independent) method of configuring automatic startup of a database is Oracle Restart.

> **See Also:**
>
> *Oracle Database Administrator's Guide* to learn about Oracle Restart

Starting Remote Instances

If your local Oracle Database server is part of a distributed database, then you might want to start a remote instance and database.

Procedures for starting and stopping remote instances vary widely depending on communication protocol and operating system.

Altering Database Availability

You can alter the availability of a database. You may want to do this in order to restrict access for maintenance reasons or to make the database read only.

- Mounting a Database to an Instance
  When you perform specific administrative operations, the database must be started and mounted to an instance, but closed. You can achieve this scenario by starting the instance and mounting the database.
- Opening a Closed Database
  When a database is mounted but closed, you can make it available for general use by opening it.
- Opening a Database in Read-Only Mode
  Opening a database in read-only mode enables you to query an open database while eliminating any potential for online data content changes.
- Restricting Access to an Open Database
  When a database is in restricted mode, only users with the `RESTRICTED SESSION` privilege can initiate new connections. Users connecting as `SYSDBA` or connecting with the `DBA` role have this privilege.

Mounting a Database to an Instance

When you perform specific administrative operations, the database must be started and mounted to an instance, but closed. You can achieve this scenario by starting the instance and mounting the database.

- To mount a database to a previously started, but not opened instance, use the SQL statement `ALTER DATABASE` with the `MOUNT` clause as follows:

```
ALTER DATABASE MOUNT;
```

> **See Also:**
>
> "Starting an Instance and Mounting a Database" for a list of operations that require the database to be mounted and closed (and procedures to start an instance and mount a database in one step)

Opening a Closed Database

When a database is mounted but closed, you can make it available for general use by opening it.

- To open a mounted database, use the `ALTER DATABASE` SQL statement with the `OPEN` clause:

```
ALTER DATABASE OPEN;
```

After executing this statement, any valid Oracle Database user with the `CREATE SESSION` system privilege can connect to the database.

Opening a Database in Read-Only Mode

Opening a database in read-only mode enables you to query an open database while eliminating any potential for online data content changes.

While opening a database in read-only mode guarantees that data files and redo log files are not written to, it does not restrict database recovery or operations that change the state of the database without generating redo. For example, you can take data files offline or bring them online since these operations do not affect data content.

If a query against a database in read-only mode uses temporary tablespace, for example to do disk sorts, then the issuer of the query must have a locally managed tablespace assigned as the default temporary tablespace. Otherwise, the query will fail.

The following statement opens a database in read-only mode:

```
ALTER DATABASE OPEN READ ONLY;
```

You can also open a database in read/write mode as follows:

```
ALTER DATABASE OPEN READ WRITE;
```

However, read/write is the default mode.

> **Note:**
>
> You cannot use the `RESETLOGS` clause with a `READ ONLY` clause.

Limitations of a Read-only Database

- An application must not write database objects while executing against a read-only database. For example, an application writes database objects when it inserts, deletes, updates, or merges rows in a database table, including a global temporary table. An application writes database objects when it manipulates a database sequence. An application writes database objects when it locks rows, when it runs `EXPLAIN PLAN`, or when it executes DDL. Many of the functions and procedures in Oracle-supplied PL/SQL packages, such as `DBMS_SCHEDULER`, write database objects. If your application calls any of these functions and procedures, or if it

performs any of the preceding operations, your application writes database objects and hence is not read-only.

- When executing on a read-only database, you must commit or roll back any in-progress transaction that involves one database link before you use another database link. This is true even if you execute a generic `SELECT` statement on the first database link and the *transaction* is currently read-only.
- You cannot compile or recompile PL/SQL stored procedures on a read-only database. To minimize PL/SQL invalidation because of remote procedure calls, use `REMOTE_DEPENDENCIES_MODE=SIGNATURE` in any session that does remote procedure calls on a read-only database.
- You cannot invoke a remote procedure (even a read-only remote procedure) from a read-only database if the remote procedure has never been called on the database. This limitation applies to remote procedure calls in anonymous PL/SQL blocks and in SQL statements. You can either put the remote procedure call in a stored procedure, or you can invoke the remote procedure in the database before it becomes read only.

See Also:

Oracle Database SQL Language Reference for more information about the `ALTER DATABASE` statement

Restricting Access to an Open Database

When a database is in restricted mode, only users with the `RESTRICTED SESSION` privilege can initiate new connections. Users connecting as `SYSDBA` or connecting with the `DBA` role have this privilege.

To place an already running instance in restricted mode:

- Run the SQL statement `ALTER SYSTEM` with the `ENABLE RESTRICTED SESSION` clause.

When you place a running instance in restricted mode, no user sessions are terminated or otherwise affected. Therefore, after placing an instance in restricted mode, consider terminating all current user sessions before performing administrative tasks.

To lift an instance from restricted mode, use `ALTER SYSTEM` with the `DISABLE RESTRICTED SESSION` clause.

See Also:

- *Oracle Database Administrator's Guide* for directions for ending user sessions
- *Oracle Database Administrator's Guide* to learn some reasons for placing an instance in restricted mode

Shutting Down a CDB

You can shut down a CDB with SQL*Plus or Oracle Restart.

- About Shutting Down the Database
  When Oracle Restart is not in use, you can shut down a database instance with SQL*Plus by connecting as `SYSOPER`, `SYSDBA`, `SYSBACKUP`, or `SYSDG` and issuing the `SHUTDOWN` command. If your database is being managed by Oracle Restart, the recommended way to shut down the database is with the `srvctl stop database` command.
- Shutting Down with the Normal Mode
  When you shut down a database with the normal mode, the database waits for all connected users to disconnect before shutting down. Normal mode is the default mode of shutdown.
- Shutting Down with the Immediate Mode
  When you shut down a database with the immediate mode, Oracle Database terminates any executing SQL statements and disconnects users. Active transactions are terminated and uncommitted changes are rolled back.
- Shutting Down with the Transactional Mode
  When you shut down a database with transactional mode, the database prevents users from starting new transactions, but waits for all current transactions to complete before shutting down. This mode can take a significant amount of time depending on the nature of the current transactions.
- Shutting Down with the Abort Mode
  You can shut down a database instantaneously by terminating the database instance.
- Shutdown Timeout
  Shutdown modes that wait for users to disconnect or for transactions to complete have a limit on the amount of time that they wait.

About Shutting Down the Database

When Oracle Restart is not in use, you can shut down a database instance with SQL*Plus by connecting as `SYSOPER`, `SYSDBA`, `SYSBACKUP`, or `SYSDG` and issuing the `SHUTDOWN` command. If your database is being managed by Oracle Restart, the recommended way to shut down the database is with the `srvctl stop database` command.

Control is not returned to the session that initiates a database shutdown until shutdown is complete. Users who attempt connections while a shutdown is in progress receive a message like the following:

```
ORA-01090: shutdown in progress - connection is not permitted
```

> **Note:**
>
> You cannot shut down a database if you are connected to the database through a shared server process.

There are several modes for shutting down a database: normal, immediate, transactional, and abort. Some shutdown modes wait for certain events to occur (such as transactions completing or users disconnecting) before actually bringing down the database. There is a one-hour timeout period for these events.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about Oracle Restart

Shutting Down with the Normal Mode

When you shut down a database with the normal mode, the database waits for all connected users to disconnect before shutting down. Normal mode is the default mode of shutdown.

To shut down a database in normal situations, use one of these commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`SHUTDOWN [NORMAL]`	`srvctl stop database -db` *`db_unique_name`* `-stopoption normal`

The `NORMAL` clause of the SQL*Plus `SHUTDOWN` command is optional because this is the default shutdown method. For SRVCTL, if the `-stopoption` option is omitted, the shutdown operation proceeds according to the stop options stored in the Oracle Restart configuration for the database. The default stop option is `immediate`.

Normal database shutdown proceeds with the following conditions:

- No new connections are allowed after the statement is issued.
- Before the database is shut down, the database waits for all currently connected users to disconnect from the database.

The next startup of the database will not require any instance recovery procedures.

Shutting Down with the Immediate Mode

When you shut down a database with the immediate mode, Oracle Database terminates any executing SQL statements and disconnects users. Active transactions are terminated and uncommitted changes are rolled back.

Use immediate database shutdown only in the following situations:

- To initiate an automated and unattended backup
- When a power shutdown is going to occur soon
- When the database or one of its applications is functioning irregularly and you cannot contact users to ask them to log off or they are unable to log off

To shut down a database immediately, use one of the following commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`SHUTDOWN IMMEDIATE`	`srvctl stop database -db db_unique_name -stopoption immediate`

Immediate database shutdown proceeds with the following conditions:

- No new connections are allowed, nor are new transactions allowed to be started, after the statement is issued.
- Any uncommitted transactions are rolled back. (If long uncommitted transactions exist, this method of shutdown might not complete quickly, despite its name.)
- Oracle Database does not wait for users currently connected to the database to disconnect. The database implicitly rolls back active transactions and disconnects all connected users.

The next startup of the database will not require any instance recovery procedures.

Shutting Down with the Transactional Mode

When you shut down a database with transactional mode, the database prevents users from starting new transactions, but waits for all current transactions to complete before shutting down. This mode can take a significant amount of time depending on the nature of the current transactions.

When you want to perform a planned shutdown of an instance while allowing active transactions to complete first, use one of the following commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`SHUTDOWN TRANSACTIONAL`	`srvctl stop database -db db_unique_name -stopoption transactional`

Transactional database shutdown proceeds with the following conditions:

- No new connections are allowed, nor are new transactions allowed to be started, after the statement is issued.
- After all transactions have completed, any client still connected to the instance is disconnected.
- At this point, the instance shuts down just as it would when a `SHUTDOWN IMMEDIATE` statement is submitted.

The next startup of the database will not require any instance recovery procedures.

A transactional shutdown prevents clients from losing work, and at the same time, does not require all users to log off.

Shutting Down with the Abort Mode

You can shut down a database instantaneously by terminating the database instance.

If possible, perform this type of shutdown *only* in the following situations:

- The database or one of its applications is functioning irregularly *and* none of the other types of shutdown works.
- You must shut down the database instantaneously (for example, if you know a power shutdown is going to occur in one minute).
- You experience problems when starting a database instance.

When you must do a database shutdown by aborting transactions and user connections, use one of the following commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`SHUTDOWN ABORT`	`srvctl stop database -db db_unique_name -stopoption abort`

An aborted database shutdown proceeds with the following conditions:

- No new connections are allowed, nor are new transactions allowed to be started, after the statement is issued.
- Current client SQL statements being processed by Oracle Database are immediately terminated.
- Uncommitted transactions are not rolled back.
- Oracle Database does not wait for users currently connected to the database to disconnect. The database implicitly disconnects all connected users.

The next startup of the database *will* require automatic instance recovery procedures.

Shutdown Timeout

Shutdown modes that wait for users to disconnect or for transactions to complete have a limit on the amount of time that they wait.

If all events blocking the shutdown do not occur within one hour, the shutdown operation aborts with the following message: `ORA-01013: user requested cancel of current operation`. This message is also displayed if you interrupt the shutdown process, for example by pressing `CTRL-C`. Oracle recommends that you do not attempt to interrupt an instance shutdown. Instead, allow the shutdown process to complete, and then restart the instance.

After `ORA-01013` occurs, you must consider the instance to be in an unpredictable state. You must therefore continue the shutdown process by resubmitting a `SHUTDOWN` command. If subsequent `SHUTDOWN` commands continue to fail, you must submit a `SHUTDOWN ABORT` command to bring down the instance. You can then restart the instance.

Quiescing a CDB

A quiesced CDB allows only DBA transactions, queries, fetches, or PL/SQL statements.

- About Quiescing a Database
  Occasionally you might want to put a database in a state that allows only DBA transactions, queries, fetches, or PL/SQL statements. Such a state is referred to as a **quiesced state**, in the sense that no ongoing non-DBA transactions, queries, fetches, or PL/SQL statements are running in the system.

- Placing a Database into a Quiesced State
  When you place a database in quiesced state, non-DBA active sessions will continue until they become inactive. An active session is one that is currently inside of a transaction, a query, a fetch, or a PL/SQL statement; or a session that is currently holding any shared resources (for example, enqueues). No inactive sessions are allowed to become active.
- Restoring the System to Normal Operation
  When you restore the system to normal operation, all non-DBA activity is allowed to proceed.
- Viewing the Quiesce State of an Instance
  You can view the quiesce state of an instance by querying the `V$INSTANCE` view.

About Quiescing a Database

Occasionally you might want to put a database in a state that allows only DBA transactions, queries, fetches, or PL/SQL statements. Such a state is referred to as a **quiesced state**, in the sense that no ongoing non-DBA transactions, queries, fetches, or PL/SQL statements are running in the system.

> **Note:**
>
> In this discussion of quiesce database, a DBA is defined as user `SYS` or `SYSTEM`. Other users, including those with the `DBA` role, are not allowed to issue the `ALTER SYSTEM QUIESCE DATABASE` statement or proceed after the database is quiesced.

The quiesced state lets administrators perform actions that cannot safely be done otherwise. These actions include:

- Actions that fail if concurrent user transactions access the same object, for example, changing the schema of a database table or adding a column to an existing table where a no-wait lock is required.
- Actions whose undesirable intermediate effect can be seen by concurrent user transactions, for example, a multistep procedure for reorganizing a table when the table is first exported, then dropped, and finally imported. A concurrent user who attempts to access the table after it was dropped, but before import, would not have an accurate view of the situation.

Without the ability to quiesce the database, you would need to shut down the database and reopen it in restricted mode. This is a serious restriction, especially for systems requiring 24 x 7 availability. Quiescing a database is much a smaller restriction, because it eliminates the disruption to users and the downtime associated with shutting down and restarting the database.

When the database is in the quiesced state, it is through the facilities of the Database Resource Manager that non-DBA sessions are prevented from becoming active. Therefore, while this statement is in effect, any attempt to change the current resource plan will be queued until after the system is unquiesced.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for more information about the Database Resource Manager

Placing a Database into a Quiesced State

When you place a database in quiesced state, non-DBA active sessions will continue until they become inactive. An active session is one that is currently inside of a transaction, a query, a fetch, or a PL/SQL statement; or a session that is currently holding any shared resources (for example, enqueues). No inactive sessions are allowed to become active.

For example, If a user issues a SQL query in an attempt to force an inactive session to become active, the query will appear to be hung. When the database is later unquiesced, the session is resumed, and the blocked action is processed.

- To place a database into a quiesced state, issue the following SQL statement:

```
ALTER SYSTEM QUIESCE RESTRICTED;
```

Once all non-DBA sessions become inactive, the `ALTER SYSTEM QUIESCE RESTRICTED` statement completes, and the database is in a quiesced state. In an Oracle Real Application Clusters environment, this statement affects all instances, not just the one that issues the statement.

The `ALTER SYSTEM QUIESCE RESTRICTED` statement may wait a long time for active sessions to become inactive. You can determine the sessions that are blocking the quiesce operation by querying the `V$BLOCKING_QUIESCE` view. This view returns only a single column: `SID` (Session ID). You can join it with `V$SESSION` to get more information about the session, as shown in the following example:

```
select bl.sid, user, osuser, type, program
from v$blocking_quiesce bl, v$session se
where bl.sid = se.sid;
```

If you interrupt the request to quiesce the database, or if your session terminates unusually before all active sessions are quiesced, then Oracle Database automatically reverses any partial effects of the statement.

For queries that are carried out by successive multiple Oracle Call Interface (OCI) fetches, the `ALTER SYSTEM QUIESCE RESTRICTED` statement does not wait for all fetches to finish. It only waits for the current fetch to finish.

For both dedicated and shared server connections, all non-DBA logins after this statement is issued are queued by the Database Resource Manager, and are not allowed to proceed. To the user, it appears as if the login is hung. The login will resume when the database is unquiesced.

The database remains in the quiesced state even if the session that issued the statement exits. A DBA must log in to the database to issue the statement that specifically unquiesces the database.

> **Note:**
>
> You cannot perform a cold backup when the database is in the quiesced state, because Oracle Database background processes may still perform updates for internal purposes even while the database is quiesced. In addition, the file headers of online data files continue to appear to be accessible. They do not look the same as if a clean shutdown had been performed. However, you can still take online backups while the database is in a quiesced state.

> **See Also:**
>
> - *Oracle Database Reference* for more information about the `V$BLOCKING_QUIESCE` view
> - *Oracle Database Reference* for more information about the `V$SESSION` view

Restoring the System to Normal Operation

When you restore the system to normal operation, all non-DBA activity is allowed to proceed.

- To restore the database to normal operation, issue the following SQL statement:

```
ALTER SYSTEM UNQUIESCE;
```

In an Oracle Real Application Clusters environment, this statement is not required to be issued from the same session, or even the same instance, as that which quiesced the database. If the session issuing the `ALTER SYSTEM UNQUIESCE` statement terminates unusually, then the Oracle Database server ensures that the unquiesce operation completes.

Viewing the Quiesce State of an Instance

You can view the quiesce state of an instance by querying the `V$INSTANCE` view.

To view the quiesce state of an instance:

- Query the `ACTIVE_STATE` column of the `V$INSTANCE` view.

The column has one of these values:

- `NORMAL`: Normal unquiesced state.
- `QUIESCING`: Being quiesced, but some non-DBA sessions are still active.
- `QUIESCED`: Quiesced; no non-DBA sessions are active or allowed.

Suspending and Resuming a Database

The `ALTER SYSTEM SUSPEND` statement halts all input and output (I/O) to data files (file header and file data) and control files. The suspended state lets you back up a database without I/O interference. When the database is suspended all preexisting I/O operations are allowed to complete and any new database accesses are placed in a queued state. Use the `ALTER SYSTEM RESUME` statement to resume normal database operations.

To suspend database operations:

- Run the `ALTER SYSTEM SUSPEND` statement.

To resume database operations:

- Run the `ALTER SYSTEM RESUME` statement.

The suspend command is not specific to an instance. In an Oracle Real Application Clusters environment, when you issue the suspend command on one system, internal locking mechanisms propagate the halt request across instances, thereby quiescing all active instances in a given cluster. However, if a new instance is started while another instance is being suspended, then the new instance is not suspended.

The `SUSPEND` and `RESUME` commands can be issued from different instances. For example, if instances 1, 2, and 3 are running, and you issue an `ALTER SYSTEM SUSPEND` statement from instance 1, then you can issue a `RESUME` statement from instance 1, 2, or 3 with the same effect.

The suspend/resume feature is useful in systems that allow you to mirror a disk or file and then split the mirror, providing an alternative backup and restore solution. If you use a system that cannot split a mirrored disk from an existing database while writes are occurring, then you can use the suspend/resume feature to facilitate the split.

The suspend/resume feature is not a suitable substitute for normal shutdown operations, because copies of a suspended database can contain uncommitted updates.

> **Note:**
>
> Do not use the `ALTER SYSTEM SUSPEND` statement as a substitute for placing a tablespace in hot backup mode. Precede any database suspend operation by an `ALTER TABLESPACE BEGIN BACKUP` statement.

The following statements illustrate `ALTER SYSTEM SUSPEND/RESUME` usage. The `V$INSTANCE` view is queried to confirm database status.

```
SQL> ALTER SYSTEM SUSPEND;
System altered
SQL> SELECT DATABASE_STATUS FROM V$INSTANCE;
DATABASE_STATUS
---------
SUSPENDED

SQL> ALTER SYSTEM RESUME;
System altered
SQL> SELECT DATABASE_STATUS FROM V$INSTANCE;
DATABASE_STATUS
```

```
---------
ACTIVE
```

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for details about backing up a database using the database suspend/resume feature

Delaying Instance Abort

The `INSTANCE_ABORT_DELAY_TIME` initialization parameter specifies the amount of time, in seconds, to delay shutting down a database when an error causes the instance to abort.

Some errors cause the Oracle database instance to abort. You can use the `INSTANCE_ABORT_DELAY_TIME` initialization parameter to specify the amount of time to delay shutting down the instance. A database administrator can use the delay time to get information about the error and minimize problems that can result when an instance aborts. For example, a database administrator might use the delay time to get diagnostics, redirect connections using Transparent Application Failover (TAF), and flush the buffer cache. A message is written to the alert log when a delayed abort is initiated.

> **Caution:**
>
> Do not set the `INSTANCE_ABORT_DELAY_TIME` value too high. Since the instance is closing because of an error, some processes or resources might be corrupted or unavailable, which can make complex actions impossible.

To delay instance abort:

- Set the `INSTANCE_ABORT_DELAY_TIME` initialization parameter to the number of seconds to delay shutting down an instance when an error causes it to abort.

  This parameter is set to `0` by default.

Example 15-10 Setting the INSTANCE_ABORT_DELAY_TIME Initialization Parameter

```
ALTER SYSTEM SET INSTANCE_ABORT_DELAY_TIME=60;
```

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.

- Modifying a CDB with ALTER SYSTEM
  To modify a CDB at the system level, use the `ALTER SYSTEM` statement.

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 15-11 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 15-12 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.

- 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.
- 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.
- Modifying the CDB Root Using ALTER DATABASE
  To modify only the root of a CDB, use 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

  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 15-3 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 15-13 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 15-14 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.
- 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.
- 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.

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.
- 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.

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 15-3 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 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 15-15 Changing the Default Tablespace for the Root

This `ALTER DATABASE` statement uses a `DEFAULT TABLESPACE` clause to set the default tablespace to `root_tbs` for the root.

```
ALTER DATABASE DEFAULT TABLESPACE root_tbs;
```

Example 15-16 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 15-17 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 15-18 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.
- 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.

- 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.
- 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.

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.
- 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.

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 15-19 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 15-20 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.

- 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.
- 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.

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 15-4.

Table 15-4 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.

Table 15-4 (Cont.) DDL Statements and the CONTAINER Clause in a CDB

DDL Statement	CONTAINER = CURRENT	CONTAINER = ALL
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 15-4.

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 15-21 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 15-4.

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 15-22 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.
- Syntax and Parameters for catcon.pl
  The `catcon.pl` script is a Perl script that must be run at an operating system prompt.
- Running the catcon.pl Script
  Examples illustrate running the `catcon.pl` script.

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.

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]
[--num_procs number]
[--user_scripts]
[--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 15-5 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*
`--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.

Table 15-5 (Cont.) catcon.pl Parameters

Parameter	Short Name	Description
`--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.
`--num_procs`	`-n`	Specifies how many SQL*Plus processes `catcon.pl` will spawn to execute statements and/or scripts supplied by the caller. This overrides the number that would be spawned by `catcon.pl` based on number of PDBs in a CDB and the value of the `CPU_COUNT` initialization parameter.
`--user_scripts`	`-S`	Specifies that all scripts and/or statements supplied by the caller will not run in `CDB$ROOT`, `PDB$SEED`, or App Root Clones. All objects, such as tables and views, created by the scripts and/or statements will not be marked as Oracle-maintained.
`--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 15-23 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 15-24 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 15-25 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 15-26 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 15-27 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 15-28 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

Monitoring Containers in a CDB

You can view metadata about CDBs, PDBs, and application containers using SQL*Plus or SQL Developer.

- 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.
- Viewing Information About the Containers in a CDB
  The `V$CONTAINERS` view provides information about all containers in a CDB, including the root and all PDBs.
- Viewing Information About PDBs
  The `CDB_PDBS` view and `DBA_PDBS` view provide information about the PDBs associated with a CDB, including the status of each PDB.
- Viewing the Open Mode of Each PDB
  The `V$PDBS` view provides information about the PDBs associated with the current database instance.
- 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.
- 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.
- Determining the Current Container ID or Name
  You can determine your current container ID or container name in a CDB.
- 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.

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.

- 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.
- 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.
- Views for a CDB
  You can query a set of views for information about a CDB and its PDBs.

> **See Also:**
>
> *Oracle Database Concepts* 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.

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 15-6 describes the meanings of the values in the `CON_ID` column.

Table 15-6 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.
- When a container is opened in restricted mode, it is ignored in queries on `CDB_` views.

See Also:

- "About the Current Container"
- *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 15-7 describes data dictionary views that are useful for monitoring a CDB and its PDBs.

Table 15-7 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*
`{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*

Table 15-7 (Cont.) Views for a CDB

View	Description	More Information
{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*
{CDB|DBA}_CDB_RSRC_PLAN_DIRECTIVES	Displays information about all the CDB resource plan directives.	*Oracle Database Reference*
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*
{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*

Table 15-7 (Cont.) Views for a CDB

View	Description	More Information
{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$DATABASE	Displays information about the database from the control file. If the database is a CDB, then CDB-related information is included.	*Oracle Database Reference*
[G]V$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*
[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*

Table 15-7 (Cont.) Views for a CDB

View	Description	More Information
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*

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 15-29 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:**
>
> - "Users, Roles, and Objects in 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 15-30 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 15-31 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 15-32 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 15-33 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 15-34 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 15-35 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 15-36 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"
- *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.
- 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.
- 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.

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 15-37 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 15-38 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"
> - *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 15-39 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 15-8 to determine the container ID, container name, DBID, GUID, and UID of a container.

Table 15-8 Functions That Return 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.
CON_ID_TO_DBID(*container_id*)	Returns the container's DBID based on the container ID.
CON_ID_TO_GUID(*container_id*)	Returns the container's globally unique identifier (GUID) based on the container ID.
CON_ID_TO_UID(*container_id*)	Returns the container's unique identifier (UID) based on the container ID.

The V$CONTAINERS view shows the name, DBID, UID, and GUID for each container in a CDB.

Example 15-40 Returning the Container ID Based on the Container Name

```
SELECT CON_NAME_TO_ID('HRPDB') FROM DUAL;
```

Example 15-41 Returning the Container ID Based on the Container DBID

```
SELECT CON_DBID_TO_ID(2226957846) FROM DUAL;
```

Example 15-42 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.

- 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.

> **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 15-43 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"

16

Administering PDBs

Administering PDBs includes tasks such as connecting to a PDB, modifying a PDB, and managing services associated with PDBs.

- About PDB Administration
  Administering a pluggable database (PDB) involves a subset of the tasks required to administer a CDB.
- Managing Connections to a PDB
  You manage connections for a PDB in the same way as for a CDB, with some special considerations.
- Modifying a PDB at the System Level
  You can use the `ALTER SYSTEM` statement to modify a PDB.
- Modifying a PDB at the Database Level
  You can modify a PDB using the `ALTER PLUGGABLE DATABASE` statement.
- 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.

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 CDB.

In this subset of tasks, most are the same for a PDB and a 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).

- Tasks Common to PDBs and CDBs
  Most administrative tasks are the same for a PDB and a CDB.
- Tasks Specific to CDBs
  Some administrative tasks cannot be performed when the current container is a PDB.

> **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 CDBs

Most administrative tasks are the same for a PDB and a 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 CDB.

Table 16-1 Administrative Tasks Common to PDBs and 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
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 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 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 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.
- Managing Services for PDBs
  You can create, modify, or remove services for a PDB.
- Modifying the Listener Settings of a Referenced PDB
  A PDB that is referenced by a proxy PDB is called a referenced PDB.

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 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 16-1 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.

- About Services for PDBs
  Each PDB has a default service, but you can create your own using SRVCTL or `DBMS_SERVICE`.
- Managing Services for a PDB Using SRVCTL and DBMS_SERVICE
  You can create, modify, or remove a service with a `PDB` property.

> **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.
- Default and User-Defined Services
  Creating a PDB creates a new default service for the PDB automatically.
- Tools for Managing Services
  Oracle recommends using the SRVCTL utility to create and modify services. Alternatively, you can use the `DBMS_SERVICE` package.

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 15-36"
> - *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 16-2 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 16-3 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 16-4 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 16-5 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 16-6 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 16-7 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 15-36"
- *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.

- 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.
- 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.

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 the database link used during PDB creation, 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.

Beginning with Oracle Database 19c, version 19.10, you can execute the `ALTER PLUGGABLE DATABASE CONTAINERS HOST` command in the CDB root, an application root, or a PDB by including the PDB name.

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 16-8 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 16-9 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"

Example 16-10 Using the PDB Name When Altering the Listener Host Name

This example changes the host name for the PDB named `PDB01` to `myhost.example.com`.

```
ALTER PLUGGABLE DATABASE PDB01 CONTAINERS HOST='myhost.example.com';
```

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 the database link used during PDB creation, 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.

Beginning with Oracle Database 19c, version 19.10, you can execute the `ALTER PLUGGABLE DATABASE CONTAINERS PORT` command in the CDB root, an application root, or a PDB by including the PDB name.

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 16-11 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 16-12 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;
```

Example 16-13 Using the PDB Name When Altering the Listener Port Number

This example changes the port number for the PDB named `PDB01` to `1543`.

```
ALTER PLUGGABLE DATABASE PDB01 CONTAINERS PORT=1543;
```

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.
- Modifying a PDB with ALTER SYSTEM
  To modify a PDB at the system level, use the `ALTER SYSTEM` statement.

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.

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 16-14 Enable Restricted Sessions in a PDB

To restrict sessions in a PDB, issue the following statement:

```
ALTER SYSTEM ENABLE RESTRICTED SESSION;
```

Example 16-15 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 PDB-Level Modifications
  The `ALTER PLUGGABLE DATABASE` for a PDB is analogous to the `ALTER DATABASE` for a CDB.
- Modifying a PDB with the ALTER PLUGGABLE DATABASE Statement
  To modify the attributes of a single PDB, use the `ALTER PLUGGABLE DATABASE` statement.
- 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.
- Managing Refreshable Clone PDBs
  A **refreshable clone PDB** is a read-only clone that can periodically synchronize with its source PDB.

About PDB-Level Modifications

The `ALTER PLUGGABLE DATABASE` for a PDB is analogous to the `ALTER DATABASE` for a 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.
- Logging and Recovery Clauses
  Use `ALTER PLUGGABLE DATABASE` to set logging and recovery and recovery modes at the PDB level.
- Miscellaneous Clauses
  You can use `ALTER PLUGGABLE DATABASE` to modify the open mode, global name, time zone, and default edition.

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 16-16 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 hybrid read only 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 HYBRID 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 16-17 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 16-18 Changing the Default Tablespaces for a PDB

- This `ALTER PLUGGABLE DATABASE` statement uses a `DEFAULT TABLESPACE` clause to set the default 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 16-19 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 16-20 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 16-21 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 16-22 Setting the Force Logging Mode of a PDB

This statement enables force logging mode for the PDB:

```
ALTER PLUGGABLE DATABASE ENABLE FORCE LOGGING;
```

Example 16-23 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 PDB-Level Modifications" 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.
- Switching Over a Refreshable Clone PDB
  You can switch the roles of a source PDB and its refreshable clone 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 16-24 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, hybrid read-only or `MIGRATE` mode. You can also mount a PDB without opening it.
- 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.
- Setting Read-Only Access for a PDB User
  You can set the access of a local user to a PDB to `READ ONLY` or `READ WRITE` with the `ALTER USER` or `CREATE USER` statement.
- 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.
- 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.
- Starting and Stopping PDBs in Oracle RAC
  You can use SRVCTL commands to manage PDBs.

About the Open Mode of a PDB

When a PDB is mounted, you can open it in read/write, read-only, hybrid read-only or `MIGRATE` mode. You can also mount a PDB without opening it.

- Summary of PDB Open Modes
  Depending on the options that you specify in `ALTER PLUGGABLE DATABASE OPEN`, the PDB opens in different modes.
- Opening a Pluggable Database in Hybrid Read-Only Mode
  Hybrid Read Only open mode is a special open mode in which PDB operates as Read Write as well as Read Only depending on which user is connected.

- 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`.
- Compatibility Checks When a PDB Is Opened
  When a PDB is opened, Oracle Database checks the compatibility of the PDB with the CDB.
- How to Disable or Enable Replay Upgrade
  By default, the Oracle Multitenant Replay Upgrade (Replay Upgrade) method is enabled for upgrades on PDBs and CDBs. However, you can enable or disable the use of the Replay Upgrade method.

Summary of PDB Open Modes

Depending on the options that you specify in `ALTER PLUGGABLE DATABASE OPEN`, the PDB opens in different modes.

You can view the current open mode of PDBs by querying the `V$PDBS.OPEN_MODE` column. The following table describes the possible PDB open modes.

Table 16-2 PDB Mount and Open Modes

Mode	Description	Notes
Read/Write	When you run `ALTER PLUGGABLE DATABASE OPEN READ WRITE`, the PDB 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.
Read-Only	When you run `ALTER PLUGGABLE DATABASE OPEN READ ONLY`, the PDB allows queries but not 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.

Table 16-2 (Cont.) PDB Mount and Open Modes

Mode	Description	Notes
Hybrid Read-only	When you run `ALTER PLUGGABLE DATABASE OPEN HYBRID READ ONLY`, the PDB allows common users to issuing DML and user transactions to proceed and allows these users to generate redo logs.	Common users shall successfully execute DDL, DML and DCL statements. Both Common and Local users shall successfully execute DQL statements (Queries and any other Read operations) Local users shall fail to execute DDL, DML and DCL statements. Any Write operation by Local user shall fail
Migrate	When you run `ALTER PLUGGABLE DATABASE OPEN UPGRADE`, the PDB is in `MIGRATE` mode. You can run database upgrade scripts on the PDB.	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 you run `ALTER PLUGGABLE DATABASE CLOSE` in an open PDB, then the PDB is mounted. The PDB does not allow changes to any objects. In this state, the PDB is accessible only to database administrators. The PDB 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.

See Also:

Oracle Database SQL Language Reference to learn more about the `ALTER PLUGGABLE DATABASE OPEN` command

Important:

Do not set the PDB state in Oracle Real Application Clusters (Oracle RAC) deployments. Setting the PDB state for Oracle RAC conflicts with the database agent running PDB open/close operations.

Opening a Pluggable Database in Hybrid Read-Only Mode

Hybrid Read Only open mode is a special open mode in which PDB operates as Read Write as well as Read Only depending on which user is connected.

Opening a pluggable database in hybrid read-only mode enables you to query an open database while eliminating any potential for online data content changes.

- When a CDB Common user connects to a PDB open in Hybrid Read Only mode, the PDB appears to be open in Read Write mode. Write to PDB would be permitted to Common user.
- When a PDB Local user or Application Common user connects to that PDB, it appears to be open in Read Only mode.

The benefit of using the hybrid read-only mode is that it enables database and application administrators to patch and maintain an application in a safe mode for open PDBs without the risk of local users, including higher privileged ones, interfering with the ongoing maintenance operation of the PDB.

The following statement opens a database in read-only mode:

```
ALTER DATABASE OPEN HYBRID READ ONLY;
```

Table 16-3 Effective Open Mode in User Session

Open Mode	Effective open mode in user session - CDB common user	Effective open mode in user session - Application common use	Effective open mode in user session - PDB local user
Read Write	Read Write	Read Write	Read Write
Read Only	Read Only	Read Only	Read Only
Hybrid Read Only	Read Write	Read Only	Read Only

> **See Also:**
>
> Oracle Database SQL Language Reference for more information about the `ALTER PLUGGABLE DATABASE` statement

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.
- SERVICES Clause
  You can use the *services* clause to specify the services that are started when a single PDB is opened.
- INSTANCES Clause
  In an Oracle RAC CDB, you can use the *instances* clause to specify the instances on which the PDB is modified.

- 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.
- To Set the Priority of a PDB
  Use the ALTER PLUGGABLE DATABASE <databasename> Priority <value> set the priority.

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.

> **Note:**

> **Note:**
>
> When a priority is set for any of the PDBs The PDBs will open in priority order the lowest priority (1) going first.

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 16-4 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.
`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 HYBRID READ ONLY [RESTRICTED][FORCE]`	Opens the PDB in hybrid read-only mode

Table 16-4 (Cont.) ALTER PLUGGABLE DATABASE Clauses That Modify the Mode of a PDB

Clause	Description
`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.

To Set the Priority of a PDB

Use the ALTER PLUGGABLE DATABASE <databasename> Priority <value> set the priority.

PDB priority concept is introduced in Oracle Databse 23c for different operations, including open, state restoration and upgrade.

Note: the PRIORITY clause was introduced in Oracle Database12cR2 to enable you to specify a priority for upgrading PDBs (ALTER PLUGGABLE DATABASE xxx UPGRADE PRIORITY n;). This syntax was documented in the Upgrade Guide.

To manage different kinds of PDBs, the following ordering rules are applied:

- PDBs are processed in an ascending order of priority. A PDB with a lower priority value will be processed before a PDB with a higher priority value.
- PDBs with the same priority may be processed in any order. However, if App PDBs and the App Root have the same priority or have no priority, App PDBs will still be opened after the App Root.- PDBs have no priority are considered to be the lowest priority.
- PDB priority for a given PDB is applicable to all RAC instances, i.e, priority is NOT specific to a given RAC instance.
- Priority will not be copied from source PDB to target PDB by plug/unplug or refreshable clone.
- App PDBs cannot have a higher priority than App Root.
- App Root Clones have the same priority as App Roots, and cannot be explicitly given a PDB priority.
- CDB$ROOT and PDB$SEED are exempt for PDB priority, their priority is determined internally by Oracle RDBMS.

The priority is determined by the integer value assigned. A priority of 1 being the first PDB opened or upgraded, followed by other PDBs in ascending priority order. If no priority is assigned all PDBs can be processed in any order. All PDBs with the same priority will be processed in any order.

ALTER PLUGGABLE DATABASE <PDB name> PRIORITY <value>

where

- PDB name is required
- PRIORITY <value> - <value> is an integer between 1 and 4096

How priority affects the behavior the following statements

- ALTER PLUGGABLE DATABASE x OPEN ...
- ALTER PLUGGABLE DATABASE x CLOSE ...
- ALTER PLUGGABLE DATABASE x SAVE STATE ...

Where x is ALL or a list of PDBs or the ALL EXCEPT clause - basically any syntax that specifies more than one PDB.

When a OPEN statement applies to more than one PDB, the priority ordering rules are applied.

Compatibility Checks When a PDB Is Opened

When a PDB is opened, Oracle Database checks the compatibility of the PDB with the CDB.

Opening a PDB upgrades it automatically when a version mismatch occurs between the PDB and the CDB root. The Replay Upgrade on PDB Open optimization, which is the default, avoids manual error correction by re-executing statements stored in

capture tables. The mechanism is the same used in application synchronization. Oracle Database 21c uses Replay Upgrade on PDB Open in the following scenarios:

- You plug in a PDB that was unplugged from a CDB in a previous release. When the PDB is opened, the database automatically performs a Replay Upgrade.
- A CDB from a previous release was upgraded to Oracle Database 21c, but a PDB in the CDB was not upgraded. If you open this PDB without the `OPEN UPGRADE` option, then the CDB automatically performs a Replay Upgrade of the PDB.

The Replay Upgrade on PDB Open feature requires that database properties `PDB_UPGRADE_SYNC` and `UPGRADE_PDB_ON_OPEN` be set to the default value of `true`. If either property is `false`, then a classic upgrade is required before you can open the PDB. If a problem occurs during replay upgrade or classic upgrade, then the CDB records a compatibility violation.

A 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 the `PDB_PLUG_IN_VIOLATIONS` view.

See Also:

- "Modifying the Open Mode of PDBs" to learn how to modify the open mode of one or more PDBs when the current container is the root
- *Oracle Database Reference* to learn about the `PDB_PLUG_IN_VIOLATIONS` view

How to Disable or Enable Replay Upgrade

By default, the Oracle Multitenant Replay Upgrade (Replay Upgrade) method is enabled for upgrades on PDBs and CDBs. However, you can enable or disable the use of the Replay Upgrade method.

To disable the Parallel Upgrade Utility (`catctl.pl`) default of performing a Replay Upgrade, run the following command, on either `CDB$ROOT` or a particular PDB:

```
ALTER DATABASE UPGRADE SYNC OFF
```

To re-enable the Replay Upgrade behavior, enter the following command

```
ALTER DATABASE UPGRADE SYNC ON
```

You can also select a non-replay upgrade by setting the Parallel Upgrade Utility (`catctl.pl`) parameter `-t`, which forces a non-replay upgrade that uses the classic scripting method.

> **Note:**
>
> You can manage use of the Replay Upgrade method on the entire CDB, or on individual PDBs, depending on whether you are connected to `CDB$ROOT`, or to a particular PDB:
>
> - If `UPGRADE SYNC` is set to `OFF` in `CDB$ROOT`, then the Replay Upgrade method is not used for any PDBs plugged into the CDB.
> - If `UPGRADE SYNC` is set to `ON` in `CDB$ROOT`, but set to `OFF` for a PDB, then the Replay Upgrade method is not used for the PDB where `UPGRADE SYNC` is `OFF`, but the Replay Upgrade method is used for all other PDBs plugged into the CDB.
> - If `UPGRADE SYNC` is set to `ON` in `CDB$ROOT`, and set to `ON` for all PDBs (the default), then the Replay Upgrade method is used for all PDBs plugged into the CDB.

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 16-5 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
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
Hybrid Read-only	Yes	Yes	Read-only or read/write	
Hybrid Read-only	Yes	No	Read-only or read/write	Mounted, read-only, or read/write
Hybrid Read-only	No	Yes	Read-only or read/write	Mounted or read-only
Hybrid Read-only	No	No	Read-only or read/write	Mounted, read-only, or read/write
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 16-25 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 16-26 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 16-27 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

Setting Read-Only Access for a PDB User

You can set the access of a local user to a PDB to READ ONLY or READ WRITE with the ALTER USER or CREATE USER statement.

To set read-only access for a PDB user

You can set read-only access to a PDB user using the READ ONLY clause in ALTER USER or CREATE USER statements. After read-only access is enabled for a PDB user, whenever that user connects to the PDB, the session operates as if the database is open in read-only mode and the user cannot perform any write operation. This statement can be executed by anyone with the ALTER USER or CREATE USER privilege. Note that you can view the state of a local user in the *_USERS view.

Example 16-28 Enabling Read-Only Access for a PDB User

To enable read-only access for a PDB local user, use the `ALTER USER` statement with the `READ ONLY` clause. You can use the `READ ONLY` clause with the `CREATE USER` statement also.

```
alter user user1 read only;
```

Example 16-29 Revoking Read-Only Access for a PDB User with the `READ WRITE` Clause

To revoke read-only access for a PDB local user, use the `ALTER USER` statement with the `READ WRITE` clause.

```
alter user user1 read write;
```

Example 16-30 Setting Read-Only Access for a PDB User with the `CREATE USER` Clause

To set read-only access for a PDB local user, you can also use the `READ ONLY` clause with the `CREATE USER` statement.

```
create user u1 identified by u1 read only;
```

See Also:

- CREATE USER
- ALTER USER
- About Privileges and Roles
- Configuring Hybrid Read-Only Users

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 16-31 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 16-32 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 16-33 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 16-34 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 16-35 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.
- Starting Up a PDB Using the STARTUP Command
  When the current container is a PDB, the SQL*Plus `STARTUP` command opens the PDB.
- 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.
- Shutting Down a PDB Using the SHUTDOWN Command
  When the current container is a PDB, the SQL*Plus `SHUTDOWN` command closes 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 16-6 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 16-36 Opening a PDB in Read/Write Mode with the STARTUP Command

```
STARTUP OPEN
```

Example 16-37 Opening a PDB in Read-Only Mode with the STARTUP Command

```
STARTUP OPEN READ ONLY
```

Example 16-38 Opening a PDB in Read-Only Restricted Mode with the STARTUP Command

```
STARTUP RESTRICT OPEN READ ONLY
```

Example 16-39 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 16-40 Opening a PDB in Read/Write Mode with the STARTUP Command

```
STARTUP PLUGGABLE DATABASE hrpdb OPEN
```

Example 16-41 Opening a PDB in Read/Write Restricted Mode with the STARTUP Command

```
STARTUP PLUGGABLE DATABASE hrpdb RESTRICT
```

Example 16-42 Opening a PDB in Read-Only Restricted Mode with the STARTUP Command

```
STARTUP PLUGGABLE DATABASE hrpdb OPEN READ ONLY RESTRICT
```

Example 16-43 Opening a PDB in Read-Only Mode with the STARTUP Command

```
STARTUP PLUGGABLE DATABASE hrpdb OPEN READ ONLY
```

Example 16-44 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 16-45 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

You can use SRVCTL commands to manage PDBs.

> **Note:**
>
> Starting with Oracle Database 21c, installation of non-CDB Oracle Database architecture is no longer supported. The policy-managed database deployment option is desupported in Oracle Database 23c.

Starting with Oracle Database 21c, PDBs are a resource managed by Oracle Clusterware. Consider a admin-managed CDB called `raccont` that has a PDB called `spark`.

> **Note:**
>
> If you attempt to create the service without first creating the PDB, then you will get an error message indicating you must create the PDB resource first.

If the `spark` PDB was created with cardinality set to 1, or 2, or ALL, then if you create a service named `plug` for the PDB, the service can use the `-cardinality` argument, too. If the `spark` PDB was created without specifying the `-cardinality` argument, then new services you create for the PDB use the `-preferred` or `-available` arguments, not the `-cardinality` argument.

Because PDBs are managed as an Oracle Clusterware resource, typical Oracle RAC-based management practices apply. For this reason, if the PDB `spark` is in the online state when Oracle Clusterware is shut down on a server hosting this service, then the PDB 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.

To start a Pluggable Database:

```
$ srvctl start pdb -db db_name -pdb pdb_name [-startoption
start_options]
```

To start a Pluggable Database on specific nodes:

```
$ srvctl start pdb -db db_name -pdb pdb_name -node node_list
   [-startoption start_options]
```

To stop a PDB and all its services on all nodes within a database using the IMMEDIATE option:

```
$ srvctl stop pdb -db db_name -pdb pdb_name -stopoption IMMEDIATE -
drain_timeout 0
  -stopsvcoption IMMEDIATE
```

To stop a Pluggable Database on specific nodes:

```
$ srvctl stop pdb -db db_name -pdb pdb_name -node node_list
   [-stopoption stop_options] [-stopsvcoption stop_service_options
   [-drain_timeout timeout]
```

If you do not want the `spark` PDB to restart when the Oracle RAC database is restarted on all, or on a specific node, use the following command:

```
srvctl disable pdb -db raccont -pdb spark [-node node_name]
```

To view the status of the PDB service `plug`, use the following command:

```
srvctl status service -db raccont -service plug -verbose
```

To view the status of the PDB `spark`, use the following command:

```
srvctl status pdb -db raccont -pdb plug -detail
```

To modify the configuration of the PDB, use the following command:

```
srvctl modify pdb -db db_unique_name -pdb pdb_name
     [-cardinality {num_of_instances | ALL}]
     [-maxcpu max_cpu_usage] [-mincpuunit min_cpu_usage]
     [-rank rank] [-startoption start_options]
     [-stopoption stop_options] [-policy policy]
```

> **Note:**
>
> You can modify the `-cardinality` parameter only if you had set the `-cardinality` parameter when creating the PDB.

Related Topics

- *Oracle Real Application Clusters Administration and Deployment Guide*

17

Administering an Application Container

You can install and administer the applications installed in application containers.

> **Note:**
>
> You can complete the tasks in this chapter using SQL*Plus or Oracle SQL Developer.

- 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 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.
- Managing Applications in an Application Container
  You install, upgrade, or patch an application in an application container.
- 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.
- 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.
- 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.
- 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 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.

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

- 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.
- Application Maintenance
  In this context, **application maintenance** refers to installing, uninstalling, upgrading, or patching an application.
- 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.
- Implicitly Created Applications
  In addition to user-created applications, application containers can also contain 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.

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. These containers include the application root, application PDBs that belong to the application root, and an optional application seed that belongs to the application root.	*Oracle Database Security Guide*
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 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"

- Transparent Data Encryption and Application Containers
  Best Practices to use TDE with Application Containers.

Transparent Data Encryption and Application Containers

Best Practices to use TDE with Application Containers.

> **Note:**
>
> If Transparent Data Encryption is enabled in the application root, then an external password store must be configured."

If TDE is enabled in Application Root, then **our recommendation is to configure a SEPS (Secure External Password Store) keystore** to store the password for the TDE wallet. If a SEPS keystore is configured, then the `APPLICATION BEGIN UPGRADE` statement does not need a `KEYSTORE` clause. The Application Root Clone will then be created with the `KEYSTORE IDENTIFIED BY EXTERNAL STORE` clause.

If a SEPS keystore is not configured, then the `APPLICATION BEGIN UPGRADE` statement needs to include a '**`KEYSTORE IDENTIFIED BY <password>`**' clause, otherwise creation of the Application Root Clone will fail. If the clause is specified, then the Application Root Clone will be created with the `'KEYSTORE IDENTIFIED BY <password>'` clause.

Whether or not a SEPS keystore is configured, the TDE wallet should always be configured as an **auto-login** wallet. This is so that the wallet is opened automatically in Application Root Clone on access. Without an auto-login wallet, customer does not have a way to open the wallet in the Application Root Clone as SET CONTAINER to it and connections to it are disallowed.

Related Topics

- Transparent Data Encryption
  Transparent Data Encryption enables encryption of database columns before storing them in the data file, or enables encryption of entire tablespaces.
- A Secure External Password Store
  Consider using client-side Oracle wallets to reduce exposing authentication and signing credentials over networks.

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.
- 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.
- Application Upgrade
  An **application upgrade** is a major change to an installed application.
- Application Patch
  An **application patch** is a minor change to an application.

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 17-1 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 17-2 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';
```

- How an Application Upgrade Works
  During an application upgrade, the application remains available. To make this availability possible, Oracle Database clones the application root.
- Applications at Different Versions
  Different application PDBs might use different versions of the application.

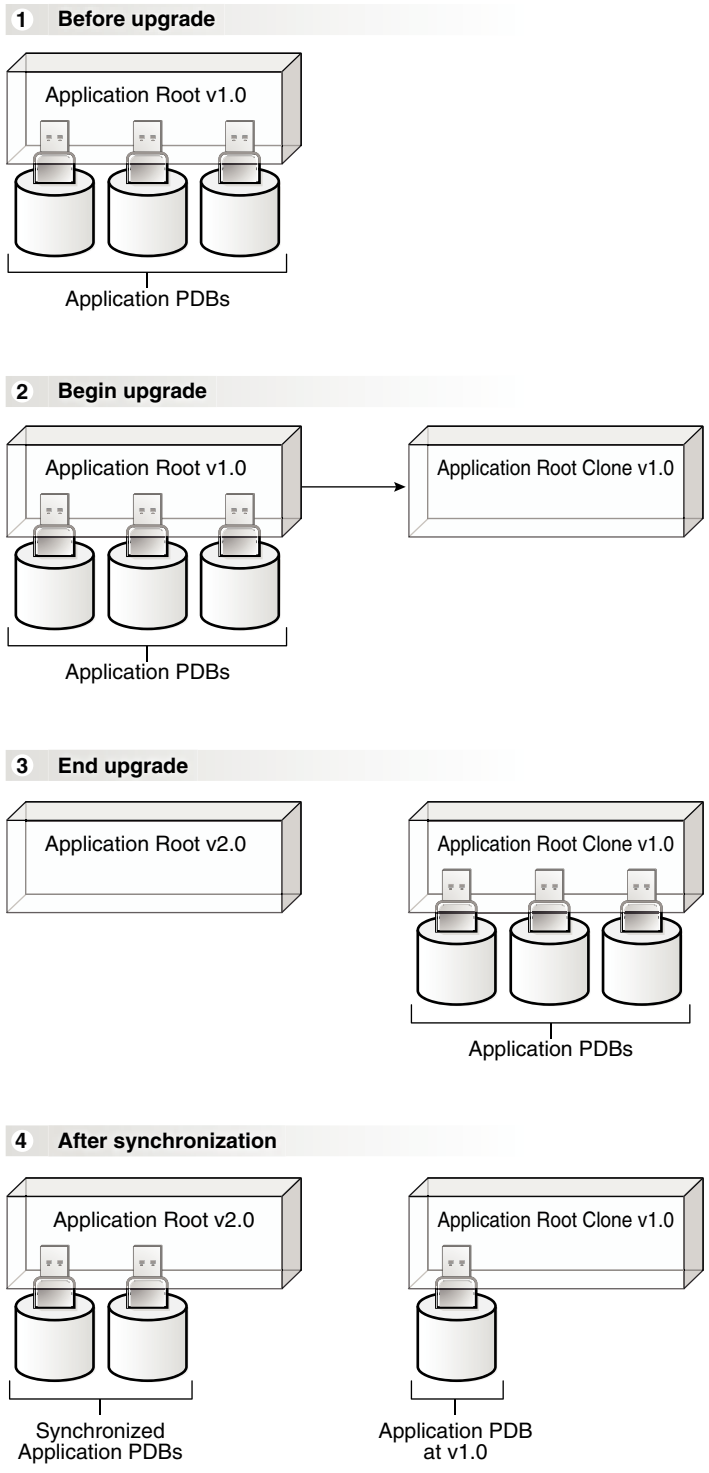
> **See Also:**
>
> "Upgrading Applications in an Application Container"

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 17-1 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 17-3 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.
- Synchronization of Multiple Applications
  You can list multiple applications by name or specify the `ALL` keyword.

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 list multiple applications by name or specify the `ALL` keyword.

Applications Specified by Name

If you list multiple application names before `SYNC`, then the database synchronizes the specified applications. The following example synchronizes both `apexapp` and `ordsapp`:

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

When specifying multiple applications by name, the `SYNC TO PATCH patchno` and `SYNC TO version` clauses are not supported.

Applications Specified by ALL

If you specify `ALL SYNC`, then the database synchronizes all applications, including those implicitly created. The following statement synchronizes all applications:

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

You can synchronize all except a specified subset of applications, as in the following statement:

```
ALTER PLUGGABLE DATABASE APPLICATION ALL EXCEPT apexapp, ordsapp SYNC;
```

When using `ALL`, the `SYNC TO PATCH patchno` and `SYNC TO version` clauses are not supported.

Order of Replay During Synchronization

When specifying multiple applications using `ALL` or a list of names, the replay order for application `BEGIN` and `END` blocks is the same as the capture order. Assume that you upgrade applications in the following order:

1. `apexapp` from `1.0` to `2.0`
2. `ordsapp` from `1.0` to `2.0`
3. `apexapp` from `2.0` to `3.0`

The statement `ALTER PLUGGABLE DATABASE APPLICATION apexapp, ordsapp SYNC` replays the statements in the same sequence. If objects in `apexapp` and `ordsapp` depend on one another, then the ordering of replay is important for functional correctness. Executing `ALTER PLUGGABLE DATABASE APPLICATION apexapp SYNC` and then `ALTER PLUGGABLE DATABASE APPLICATION ordsapp SYNC` would replay statements in the following sequence:

1. `apexapp` from `1.0` to `2.0`

2. `apexapp` from `2.0` to `3.0`
3. `ordsapp` from `1.0` to `2.0`

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

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.

- 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.
- Installing Applications in an Application Container
  You can install an application in an application container.

- Upgrading Applications in an Application Container
  Major changes to an application constitute application upgrades. You can upgrade an application in an application container.
- Patching Applications in an Application Container
  Minor changes to an application constitute application patches.
- Migrating an Existing Application to an Application Container
  You can migrate an application that is installed in a PDB to 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.
- 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.
- 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.
- Uninstalling Applications from an Application Container
  You can uninstall an application in an application container.

Related Topics

-

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.
- Application Versions
  The application container also manages the versions of the application and the patches to 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.

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 17-4 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.
- 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.

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.
- 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.

About Upgrading Applications in an Application Container

You issue `ALTER PLUGGABLE DATABASE APPLICATION` statements to upgrade an application in the application root.

- 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.

- How an Application Upgrade Works
  When you upgrade an application, Oracle Database automatically clones the application root.
- User Interface for Application Upgrade
  To upgrade an application definition in the application root, use the `ALTER PLUGGABLE DATABASE APPLICATION ... UPGRADE` command.

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.

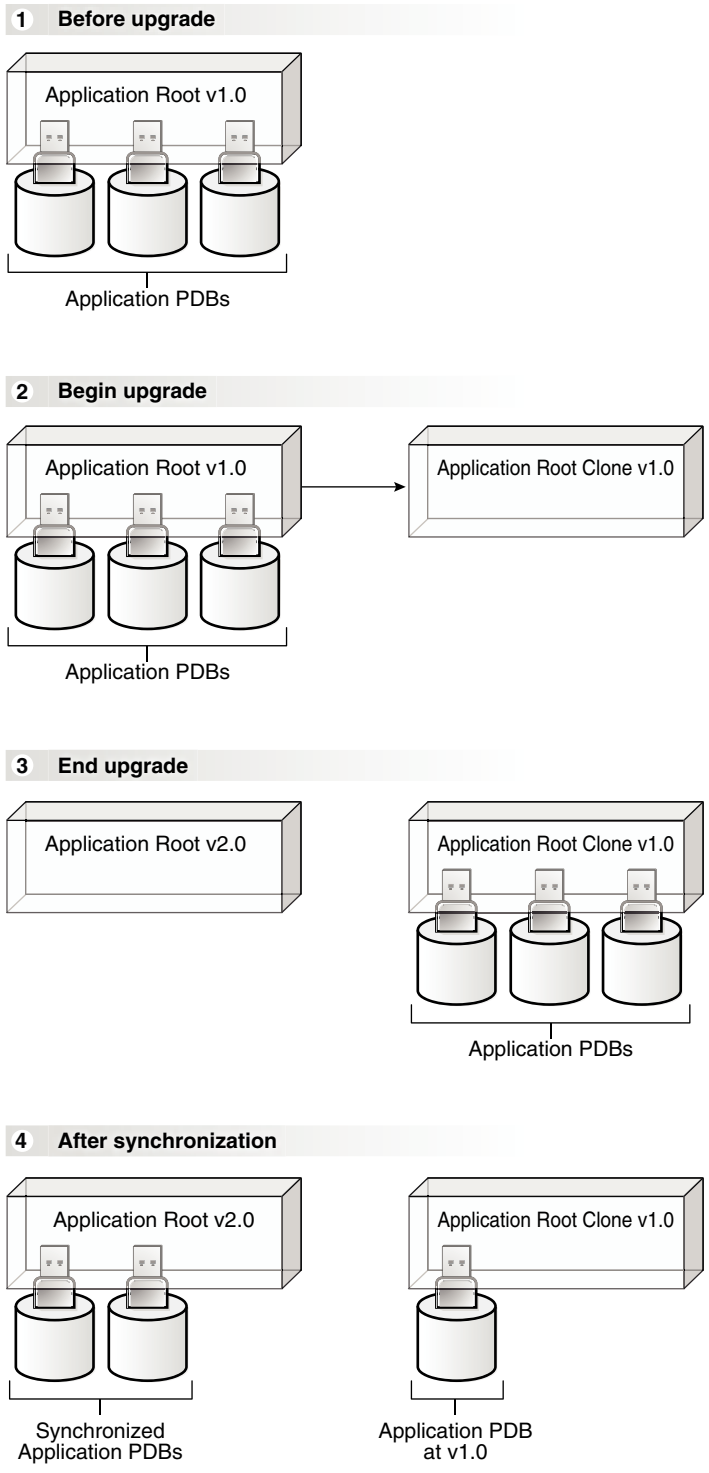
During the upgrade, application PDBs point to the root clone. Applications continue to run during the upgrade. Application PDBs can perform DML on metadata-linked and extended data-linked tables and views. Application PDBs can query metadata-linked objects, extended data-linked objects, and data-linked 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 17-2 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.

- 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.

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.
- 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.
- 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.

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 PDBs and Application Containers
  To create PDBs and application containers, use the `CREATE PLUGGABLE DATABASE` 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.

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 multiple applications as follows, where *app1* and *app2* are the names of different applications:

  ```
  ALTER PLUGGABLE DATABASE APPLICATION app1, app2 SYNC;
  ```

- Synchronize all applications as follows:

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

- Synchronize all applications except a specified subset as follows, where *app1* and *app2* are the applications to be excluded:

  ```
  ALTER PLUGGABLE DATABASE APPLICATION ALL EXCEPT app1, app2 SYNC;
  ```

Prerequisites and Restrictions

- The current user must have `ALTER PLUGGABLE DATABASE` system privilege.
- When specifying multiple applications using `ALL` or a list of names, the `SYNC TO` clause is not supported.
- Specifying multiple applications using `ALL` or a list of names 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 17-5 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 17-6 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 17-7 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 17-8 Synchronizing Multiple Applications in an Application PDB

This example synchronizes the applications `salesapp` and `eusalesapp` in an application PDB with the latest application changes in the application root.

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp, eusalesapp SYNC;
```

Example 17-9 Synchronizing All 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 17-10 Synchronizing All Applications Minus a Subset

This example synchronizes all of the applications in an application PDB except for `salesapp`.

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

Example 17-11 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;
```

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.

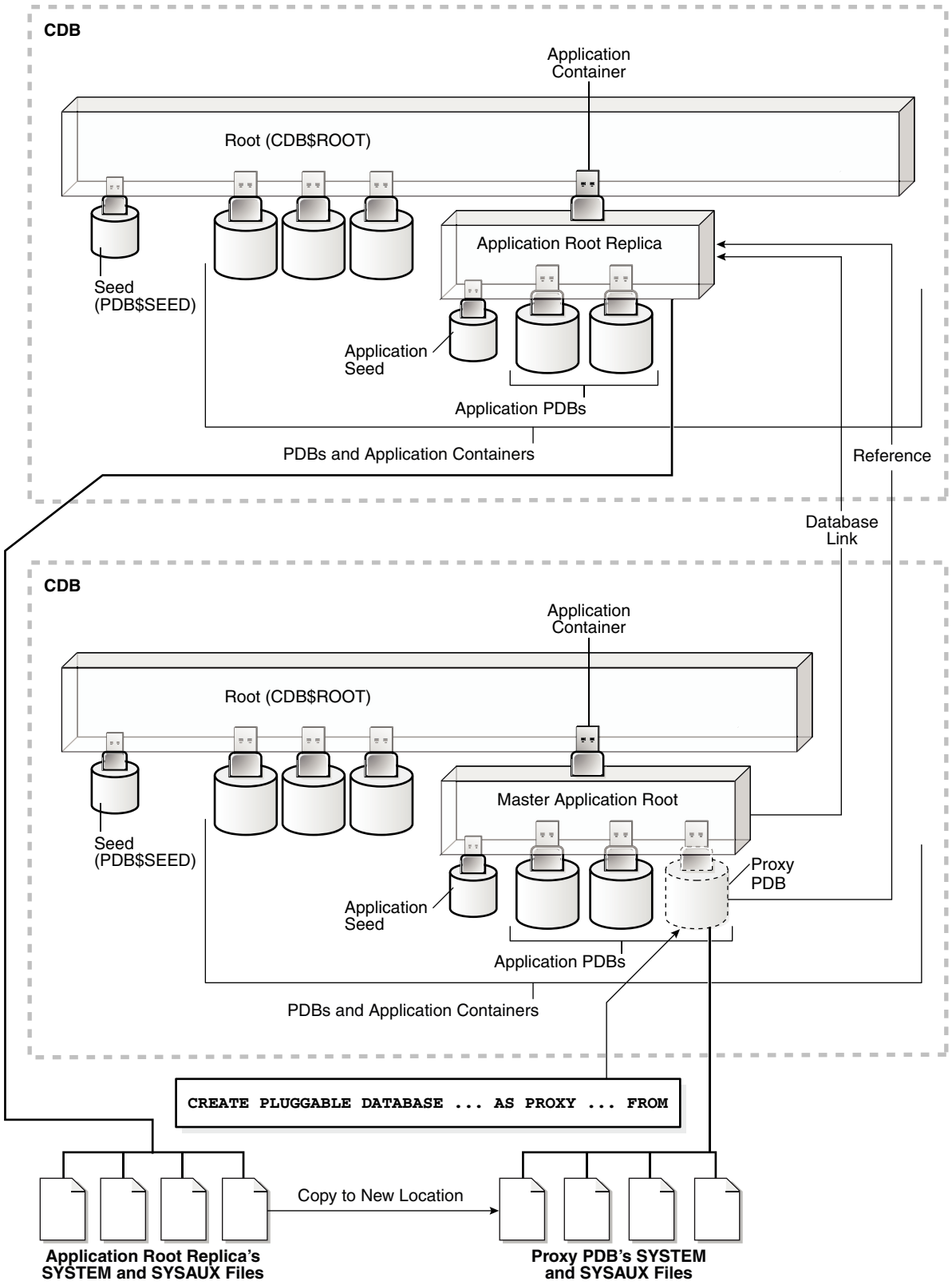
An application might be installed in several application containers. Installing, upgrading, and patching the application are more efficient when you use proxy PDBs.

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 17-3 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 17-12 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 create application containers in several different ways, including using the PDB seed, cloning an existing PDB, and plugging in an unplugged PDB by using the `CREATE PLUGGABLE DATABASE` statement.
- 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 17-13 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 17-14 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 17-15 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.
- 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.

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.
- Restrictions for Application Common Objects
  Some restrictions apply to application common objects.
- 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.
- 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.
- 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.

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
- 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.
- About Metadata-Linked Application Common Objects
  For metadata-linked application common objects, the metadata for the object is stored once in the application root.
- About 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.
- 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.

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 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 17-16 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 17-17 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 17-18 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 17-19 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 17-20 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.

- Issuing DML on Metadata-Linked Common Objects
  You can issue DML on metadata-linked application objects as normal.
- 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.

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.
- 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.

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.
- 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.

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 17-21 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 17-22 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.
- Creating a Container Map
  Create a container map by creating a map object and setting the `CONTAINER_MAP` database property to the map object.

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.
- List-Partitioned Container Map: Example
  This example uses a container map to route queries to PDBs that store data for a geographical region.
- Range-Partitioned Container Map: Example
  This example uses a container map to route queries to PDBs that store data for a particular department.

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

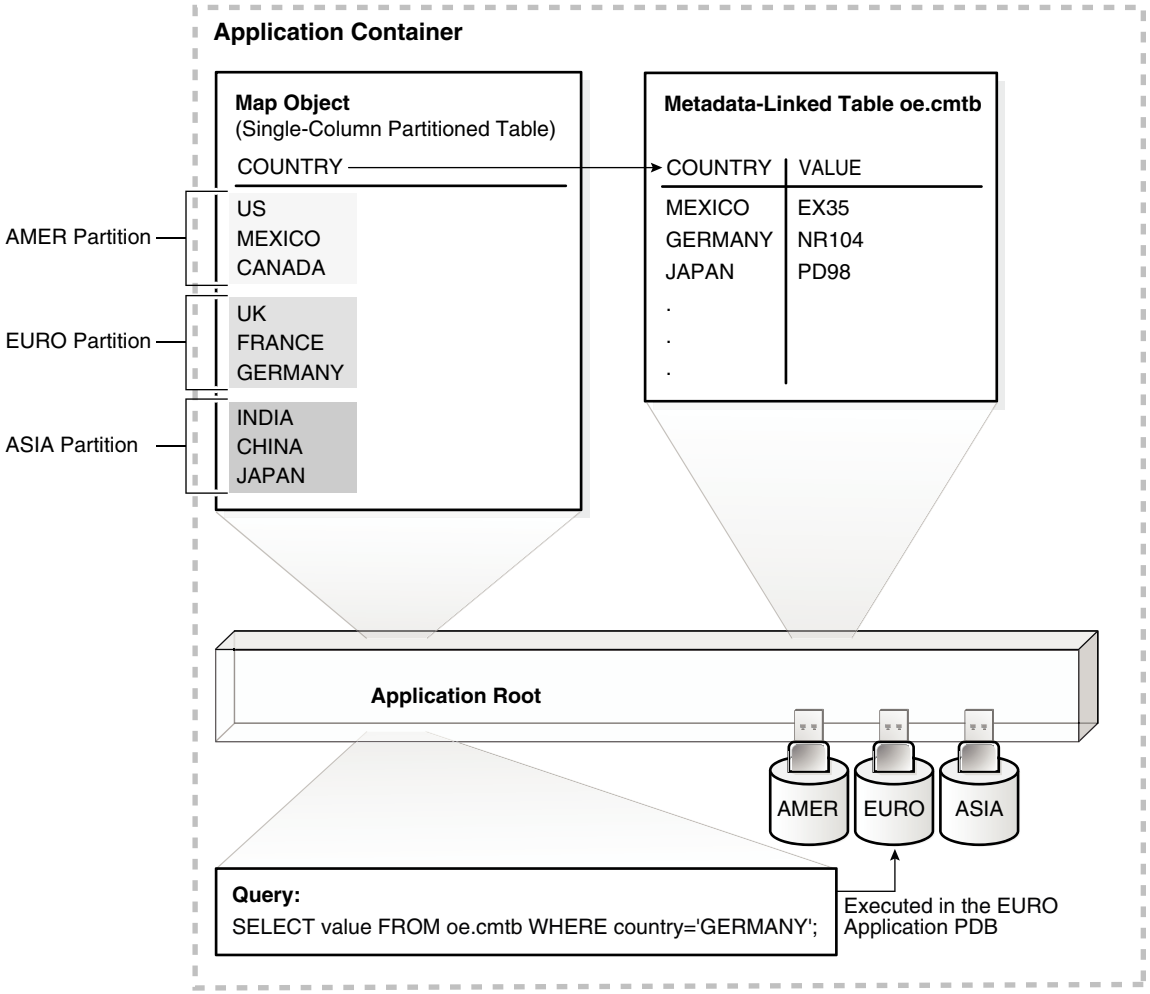
`sh.sales` does *not* include a `cname` column, the map table routes the following query to the appropriate PDB: `SELECT * FROM CONTAINERS(sh.sales) WHERE cname = 'US' ORDER BY time_id`.

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 17-4 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 17-23 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.

Viewing Information About Applications in Application Containers

Several views provide information about the applications in application containers in a CDB.

- Viewing Information About Applications
  The `DBA_APPLICATIONS` view provides information about the applications in an application container.
- 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.
- Viewing Information About Application Statements
  The `DBA_APP_STATEMENTS` view provides information about SQL statements issued during application installation, upgrade, and patch operations
- Viewing Information About Application Versions
  The `DBA_APP_VERSIONS` view provides information about the versions for applications in an application container.
- Viewing Information About Application Patches
  The `DBA_APP_PATCHES` view provides information about the patches for applications in an application container.
- Viewing Information About Application Errors
  The `DBA_APP_ERRORS` view provides information about errors raised when an application PDB synchronizes with an application in the application root.

- Listing the Shared Database Objects in an Application Container
  The `DBA_OBJECTS` view can list the shared database objects in an application container.
- 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.

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.
- Administering an Application Container
  You can install and 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 17-24 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 an Application Container
  You can install and 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 17-25 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 an Application Container"

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 17-26 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 an Application Container"
> - "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 17-27 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 an Application Container"

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 17-28 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 an Application Container"

Viewing Information About Application Errors

The DBA_APP_ERRORS view provides information about 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 errors raised during the last synchronization for each application by querying the DBA_APP_ERRORS view. You can view errors raised during the last 10 synchronizations for each application by querying the DBA_APP_ERRORS_HISTORY 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 or the DBA_APP_ERRORS_HISTORY view.

Example 17-29 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 an Application Container"

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 17-30 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 17-31 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 IV

Database Configuration Assistant Command Reference for Silent Mode

This section provides detailed information about the syntax and options for the Database Configuration Assistant (DBCA) silent mode commands.

- DBCA Overview
  This chapter gives an overview of DBCA command-line syntax, templates, and user authentication.
- DBCA Silent Mode Commands
  This section lists all the DBCA silent mode commands along with their syntax and parameter description.
- DBCA Exit Codes
  The outcome of running DBCA commands in silent mode is reported as an exit code.

18
DBCA Overview

This chapter gives an overview of DBCA command-line syntax, templates, and user authentication.

- DBCA Command-Line Syntax Overview
  This section provides an overview of the command-line syntax of DBCA in silent mode.
- About DBCA Templates
  You can use DBCA to create a database from a template supplied by Oracle or from a template that you create.
- Database User Authentication in DBCA Commands Using Oracle Wallet
  You can use Oracle wallet as a secure external password store for authenticating database users in DBCA silent mode commands.

DBCA Command-Line Syntax Overview

This section provides an overview of the command-line syntax of DBCA in silent mode.

DBCA silent mode has the following command syntax:

```
dbca  [-silent] [command [options]] [-h|-help]
```

> **Note:**
>
> On Windows, you must run DBCA as an Administrator if user access control (UAC) is enabled.

The following table describes the DBCA silent mode command syntax.

Table 18-1 DBCA Silent Mode Command Syntax Description

Option	Description
`-silent`	Specify `-silent` to run DBCA in silent mode. In silent mode, DBCA uses values that you specify as command-line options to create or modify a database.
`command options`	Specify a DBCA command and valid options for the command.

Table 18-1 (Cont.) DBCA Silent Mode Command Syntax Description

Option	Description

`-h | -help`	Displays help for DBCA.
	You can display help for a specific command by entering the following:
	`dbca command -help`
	For example, to display the help for the `-createDatabase` command, enter the following:
	`dbca -createDatabase -help`

The following example illustrates how to create a database with the silent mode of DBCA:

```
dbca -silent -createDatabase -templateName General_Purpose.dbc
                             -gdbname oradb.example.com
                             -sid oradb
                             -characterSet AL32UTF8
                             -memoryPercentage 30

Enter SYSTEM user password:
password
Enter SYS user password:
password
Copying database files
1% complete
3% complete
...
```

To ensure completely silent operation, you can redirect stdout to a file. If you do this, however, you may have to supply passwords for the administrative users in command-line arguments or the response file.

> **Note:**
>
> If you use Oracle wallet as a secure external password store for storing passwords for the administrative users, then you do not have to supply passwords for these users in the command-line arguments or in the response file. See "Database User Authentication in DBCA Commands Using Oracle Wallet" for more information.

To view brief help for DBCA command-line arguments, enter the following command:

```
dbca -help
```

For more detailed argument information, including defaults, view the response file template found on your distribution media. See the Oracle Database installation guide for your platform to get information about the name and location of the response file template.

> **See Also:**
>
> "DBCA Silent Mode Commands"

About DBCA Templates

You can use DBCA to create a database from a template supplied by Oracle or from a template that you create.

A DBCA template is an XML file that contains information required to create a database. Oracle ships templates for the following two workload types:

- General purpose OR online transaction processing
- Data warehouse

Select the template suited to the type of workload your database will support. If you are not sure which to choose, then use the "General purpose OR online transaction processing" template. You can also create custom templates to meet your specific workload requirements.

> **Note:**
>
> The General Purpose or online transaction processing template and the data Warehouse template create a database with the `COMPATIBLE` initialization parameter set to 12.1.0.2.0.

Database User Authentication in DBCA Commands Using Oracle Wallet

You can use Oracle wallet as a secure external password store for authenticating database users in DBCA silent mode commands.

Oracle wallet is a secure software container external to Oracle Database, which can be used to store authentication credentials of Oracle Database users. You can use the following DBCA silent mode command parameters to use Oracle wallet for authenticating database users:

- `useWalletForDBCredentials` : Specify `true` to use Oracle wallet for database user authentication, else specify `false`. Default is `false`.

  If `true` is specified, then provide the following additional parameters:

  - `dbCredentialsWalletLocation`: Directory in which the Oracle wallet files are stored.
  - (Optional) `dbCredentialsWalletPassword`: Password for the Oracle wallet account user. If the Oracle wallet is *auto-login* enabled, then you need not specify this password.

You can store the following keys and associated passwords in the Oracle wallet that can be used by DBCA in silent mode for authenticating users:

- `oracle.dbsecurity.sysPassword`: `SYS` user password

- `oracle.dbsecurity.systemPassword`: `SYSTEM` user password
- `oracle.dbsecurity.pdbAdminPassword`: Pluggable database (PDB) administrator password
- `oracle.dbsecurity.dbsnmpPassword`: `DBSNMP` user password
- `oracle.dbsecurity.asmsnmpPassword`: `ASMSNMP` user password
- `oracle.dbsecurity.lbacsysPassword`: `LBACSYS` user password
- `oracle.dbsecurity.sysdbaUserPassword`: `SYSDBA` role user password for the database that you are creating or configuring
- `oracle.dbsecurity.oracleHomeUserPassword`: Oracle home user password
- `oracle.dbsecurity.dvUserPassword`: Oracle Data Vault user password
- `oracle.dbsecurity.dvAccountManagerPassword`: Oracle Data Vault account manager password
- `oracle.dbsecurity.emPassword`: Enterprise Manager administrator password
- `oracle.dbsecurity.asmPassword`: `ASM` user password
- `oracle.dbsecurity.asmsysPassword`: `ASMSYS` user password
- `oracle.dbsecurity.walletPassword`: Oracle wallet account user password for authenticating with a directory service
- `oracle.dbsecurity.userDNPassword`: Directory service user password
- `oracle.dbsecurity.srcDBsysdbaUserPassword`: `SYSDBA` role user password for the database that you are using as a source to perform certain operations, such as duplicating a database
- `oracle.dbsecurity.dbLinkUserPassword`: Database link user password

Note:

If you are using Oracle Unified Directory (OUD), then the OUD account passwords should be stored in the wallet using the following keys:

- `oracle.dbsecurity.walletPassword`
- `oracle.dbsecurity.userDNPassword`

See Also:

Oracle Database Security Guide for information about configuring Oracle wallet as a secure external password store using the `mkstore` command-line utility

19

DBCA Silent Mode Commands

This section lists all the DBCA silent mode commands along with their syntax and parameter description.

- addInstance
  The `addInstance` command adds a database instance to an administrator-managed Oracle RAC database.
- configureDatabase
  The `configureDatabase` command configures a database.
- configurePluggableDatabase
  The `configurePluggableDatabase` command configures a pluggable database (PDB).
- createCloneTemplate
  The `createCloneTemplate` command creates a clone (seed) database template from an existing database.
- createDatabase
  The `createDatabase` command creates a database.
- createDuplicateDB
  The `createDuplicateDB` command creates a duplicate of an Oracle database.
- createPluggableDatabase
  The `createPluggableDatabase` command creates a pluggable database (PDB) in a multitenant container database (CDB).
- createTemplateFromDB
  The `createTemplateFromDB` command creates a database template from an existing database.
- createTemplateFromTemplate
  The `createTemplateFromTemplate` command creates a database template from an existing database template.
- deleteDatabase
  The `deleteDatabase` command deletes a database.
- deleteInstance
  The `deleteInstance` command deletes a database instance from an administror-managed Oracle RAC database.
- deletePluggableDatabase
  The `deletePluggableDatabase` command deletes a PDB.
- deleteTemplate
  The `deleteTemplate` command deletes a database template.
- executePrereqs
  The `executePrereqs` command executes the prerequisites checks and reports the results. This command can be used to check the environment before running `dbca` to create a database.

- generateScripts
  The `generateScripts` command generates scripts, which can be used to create a database.
- relocatePDB
  The `relocatePDB` command relocates a PDB from a *remote* CDB to a *local* CDB.
- unplugDatabase
  The `unplugDatabase` command unplugs a pluggable database (PDB) from a multitenant container database (CDB).

addInstance

The `addInstance` command adds a database instance to an administrator-managed Oracle RAC database.

Syntax and Parameters

Use the `dbca -addInstance` command with the following syntax:

```
dbca -addInstance
   -gdbName global_database_name
   -nodeName database_instance_node_name
   [-updateDirService {true | false}
      -dirServiceUserName directory_service_user_name
      -dirServicePassword directory_service_user_password]
   [-instanceName database_instance_name]
   [-sysDBAUserName SYSDBA_user_name]
   [-sysDBAPassword SYSDBA_user_password]
   [-useWalletForDBCredentials {true | false}
      -dbCredentialsWalletPassword wallet_account_password
      -dbCredentialsWalletLocation wallet_files_directory]
```

Table 19-1 addInstance Parameters

Parameter	Required/ Optional	Description
`-gdbName` *`global_database_name`*	Required	Global database name in the form *database_name.domain_name*.
`-nodeName` *`database_instance_node_name`*	Required	Node name of the database instance.
`-instanceName` *`database_instance_name`*	Optional	Database instance name.
`-sysDBAUserName` *`SYSDBA_user_name`*	Optional	User name of the database user having the `SYSDBA` privileges.
`-sysDBAPassword` *`SYSDBA_user_password`*	Optional	Password of the database user having the `SYSDBA` privileges.

Table 19-1 (Cont.) addInstance Parameters

Parameter	Required/ Optional	Description
-updateDirService {true | false}	Optional	Specify true to register the database with a directory service, else specify false. When true is specified, the following additional parameters are required: • -dirServiceUserName: User name for the directory service. • –dirServicePassword: Password for the directory service user.
-useWalletForDBCredentials {true | false}	Optional	Specify true to use Oracle Wallet for database credentials, else specify false. Default is false. When true is specified, the following additional parameters can be provided: • -dbCredentialsWalletPassword: Password for the Oracle Wallet account. • -dbCredentialsWalletLocation: Directory location for the Oracle Wallet files. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • oracle.dbsecurity.walletPassword • oracle.dbsecurity.userDNPassword

configureDatabase

The configureDatabase command configures a database.

Syntax and Parameters

Use the dbca -configureDatabase command with the following syntax:

```
dbca -configureDatabase
   -sourceDB database_sid
   [-addDBOption database_options]
   [-configureOML4PY
      [-oml4pyConfigTablespace tablespace_for_OML4PY_configuration]
      [-enableOml4pyEmbeddedExecution {true | false}]]
   [-configureOracleR
      -oracleRConfigTablespace tablespace_for_Oracle_R_configuration]
   [-dvConfiguration {true | false}
      -dvUserName Database_Vault_owner_name
      -dvUserPassword Database_Vault_owner_password
      [-dvAccountManagerName Database_Vault_account_manager_name]
      [-dvAccountManagerPassword Database_Vault_account_manager_password]]
   [-moveDatabaseFiles
      -datafileDestination data_files_directory
      -sourceDB database_sid
      [-initParams initialization_parameters_list
        [-initParamsEscapeChar initialization_parameters_escape_character]]
      [-recoveryAreaDestination fast_recovery_area_directory
```

```
      [-recoveryAreaSize fast_recovery_area_size]]
   [-useOMF {true | false}]]
[-olsConfiguration {true | false}
   -configureWithOID configure_with_OID_flag]
[-regenerateDBPassword {true | false}]
[-registerWithDirService {true | false}
   -dirServiceUserName directory_service_user_name
   [-databaseCN database_common_name]
   [-dirServiceCertificatePath certificate_file_path]
   [-dirServiceUser directory_service_user_name]
   [-dirServicePassword directory_service_password]
   [-ldapDirectoryAccessType ldap_directory_access_type]
   [-useSYSAuthForLDAPAccess use_sys_user_for_ldap_access_flag]
   [-walletPassword wallet_password]]
[-sysDBAPassword SYSDBA_user_password]
[-sysDBAUserName SYSDBA_user_name]
[-unregisterWithDirService {true | false}
   -dirServiceUserName directory_service_user_name
   [-dirServicePassword directory_service_password]
   [-walletPassword wallet_password]]
[-useWalletForDBCredentials {true | false}
   -dbCredentialsWalletPassword wallet_account_password
   -dbCredentialsWalletLocation wallet_files_directory]
```

Table 19-2 configureDatabase Parameters

Parameter	Required/ Optional	Description
`-sourceDB` *database_sid*	Required	The database system identifier (SID) of the database being configured.
`-addDBOption` *database_options*	Optional	Specify one or more of the following Oracle Database options in the form of a comma separated list: • `JSERVER`: Oracle JServer JAVA Virtual Machine • `ORACLE_TEXT`: Oracle Text • `IMEDIA`: Oracle Locator (fully supported) and Oracle Multimedia (desupported) • `CWMLITE`: Oracle OLAP with Oracle Warehouse Builder (OWB) • `SPATIAL`: Oracle Spatial and Graph • `OMS`: Oracle Management Server • `APEX`: Oracle Application Express • `DV`: Oracle Database Vault Example: `-addDBOption JSERVER,ORACLE_TEXT,OMS`
`-configureOML4PY`	Optional	Specify this parameter to configure Oracle Machine Learning for Python in the database. Additionally, you specify the following parameters: • `-oml4pyConfigTablespace` to configure the tablespace of the `PYQSYS` schema for Oracle Machine Learning for Python. The default tablespace is `SYSAUX`. • `-enableOml4pyEmbeddedExecution` to enable the embedded Python component of Oracle Machine Learning for Python. The default value is `TRUE`.

Table 19-2 (Cont.) configureDatabase Parameters

Parameter	Required/ Optional	Description
`-configureOracleR`	Optional	Specify this parameter to configure Oracle R in the database. Additionally, you can specify the `-oracleRConfigTablespace` parameter to assign a tablespace for the Oracle R configuration, such as `SYSAUX` tablespace.
`-dvConfiguration {true | false}`	Optional	Specify `true` to enable and configure Database Vault, or specify `false`. Default is `false`. When `true` is specified, the following additional Database Vault parameters are required: • `-dvUserName`: Specify the Database Vault owner username. • `-dvUserPassword`: Specify Database Vault owner password. • `-dvAccountManagerName`: Specify a separate Database Vault account manager. • `-dvAccountManagerPassword`: Specify the Database Vault account manager password.
`-moveDatabaseFiles`	Optional	Specify this parameter to move database files from one storage location to another storage location. For example, to move database files from ASM to FS, or from FS to ASM. Specify the following additional parameters: • `-datafileDestination`: Destination directory for all the database files • `-sourceDB`: Database system identifier (SID) for a single instance database or database unique name for an Oracle RAC database • `-initParams`: Database initialization parameters in the form of comma separated list of `name=value` pairs Additionally, you can specify the `-initParamsEscapeChar` parameter for using a specific escape character between multiple values of an initialization parameter. If an escape character is not specified, backslash (/) is used as the default escape character. • `-recoveryAreaDestination`: Destination directory for the Fast Recovery Area, which is a backup and recovery area. Specify `NONE` to disable Fast Recovery Area. Additionally, you can specify the Fast Recovery Area size in megabytes using the parameter `-recoveryAreaSize`. This parameter is optional. • `-useOMF`: Specify `true` to use Oracle-Managed Files (OMF), else specify `false`.
`-olsConfiguration {true | false}`	Optional	Specify `true` to enable and configure Oracle Label Security, else specify `false`. Default is `false`. When `true` is specified, you can additionally specify the `-configureWithOID` parameter to configure Oracle Label Security with Oracle Internet Directory (OID). This parameter is optional.

Table 19-2 (Cont.) configureDatabase Parameters

Parameter	Required/ Optional	Description
`-regenerateDBPassword {true | false}`	Optional	Specify `true` to regenerate Oracle Internet Directory (OID) server registration password, else specify `false`. Default is `false`.
`-registerWithDirService {true | false}`	Optional	Specify `true` to register with a Lightweight Directory Access Protocol (LDAP) service, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters are required: • `-dirServiceUserName`: User name for the LDAP service. • `-dirServicePassword`: Password for the LDAP service. • `-databaseCN`: Database common name. • `-dirServiceCertificatePath`: Directory service certificate file path. • `-dirServiceUser`: Directory service user name. • `-ldapDirectoryAccessType {PASSWORD | SSL}`: LDAP directory access type. • `-useSYSAuthForLDAPAccess {true | false}`: Specify whether to use SYS user authentication for LDAP access. • `-walletPassword`: Password for the database wallet.
`-sysDBAPassword` *`SYSDBA_user_password`*	Optional	Password of a user having `SYSDBA` privileges.
`-sysDBAUserName` *`SYSDBA_user_name`*	Optional	User name of a user having `SYSDBA` privileges.
`-unregisterWithDirService {true | false}`	Optional	Specify `true` to unregister with a Lightweight Directory Access Protocol (LDAP) service, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters are required: • `-dirServiceUserName`: User name for the LDAP service. • `-dirServicePassword`: Password for the LDAP service. • `-walletPassword`: Password for the database wallet.

Table 19-2 (Cont.) configureDatabase Parameters

Parameter	Required/ Optional	Description
-useWalletForDBCredentials {true | false}	Optional	Specify `true` to use Oracle Wallet for database credentials, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters can be provided: • `-dbCredentialsWalletLocation`: Directory location for the Oracle Wallet files. • `-dbCredentialsWalletPassword`: Password for the Oracle Wallet account. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • `oracle.dbsecurity.walletPassword` • `oracle.dbsecurity.userDNPassword`

configurePluggableDatabase

The `configurePluggableDatabase` command configures a pluggable database (PDB).

Syntax and Parameters

Use the `dbca -configurePluggableDatabase` command with the following syntax:

```
dbca -configurePluggableDatabase
   -pdbName pdb_name
   -sourceDB cdb_sid
   [-configureOML4PY
      [-oml4pyConfigTablespace tablespace_for_OML4PY_configuration]
      [-enableOml4pyEmbeddedExecution {true | false}]]
   [-configureOracleR
      [-oracleRConfigTablespace tablespace_for_Oracle_R_configuration]]
   [-dvConfiguration {true | false}
      -dvUserName Database_Vault_owner_name
      -dvUserPassword Database_Vault_owner_password
      [-dvAccountManagerName Database_Vault_account_manager_name]
      [-dvAccountManagerPassword Database_Vault_account_manager_password]]
   [-lbacsysPassword LBACSYS_user_password]]
   [-olsConfiguration {true | false}
      [-configureWithOID configure_with_OID_flag]]
   [-pdbTimezone {{+|-}hh:mi|time_zone_region}]
   [-registerWithDirService {true | false}
      -dirServiceUserName directory_service_user_name
      [-dirServicePassword directory_service_user_password]
      [-walletPassword wallet_password]
      [-databaseCN database_common_name]
      [-dirServiceCertificatePath certificate_file_path]
      [-dirServiceUser active_directory_account_user_name]]
   [-unregisterWithDirService {true | false}
```

```
   -dirServiceUserName directory_service_user_name
   [-dirServicePassword directory_service_user_password]
   [-walletPassword wallet_password]]
[-useWalletForDBCredentials {true | false}
   -dbCredentialsWalletPassword wallet_account_password
   -dbCredentialsWalletLocation wallet_files_directory]
```

Table 19-3 configurePluggableDatabase Parameters

Parameter	Required/ Optional	Description
-pdbName *pdb_name*	Required	Name of the PDB.
-sourceDB *cdb_sid*	Required	The database system identifier (SID) of the CDB.
-configureOML4PY	Optional	Specify this parameter to configure Oracle Machine Learning for Python in the database. Additionally, you specify the following parameters: • -oml4pyConfigTablespace to configure the tablespace of the PYQSYS schema for Oracle Machine Learning for Python. The default tablespace is SYSAUX. • -enableOml4pyEmbeddedExecution to enable the embedded Python component of Oracle Machine Learning for Python. The default value is TRUE.
-configureOracleR	Optional	Specify this parameter to configure Oracle R for the PDB. Additionally, you can specify the -oracleRConfigTablespace parameter to assign a tablespace for the Oracle R configuration, for example, SYSAUX tablespace.
-dvConfiguration {true | false}	Optional	Specify true to enable and configure Database Vault for the PDB, else specify false. Default is false. When true is specified, the following additional Database Vault parameters are required: • -dvUserName: Specify the Database Vault owner user name. • -dvUserPassword: Specify Database Vault owner password. • -dvAccountManagerName: Specify a separate Database Vault account manager. • -dvAccountManagerPassword: Specify the Database Vault account manager password.
-lbacsysPassword	Optional	Specify the LBACSYS user password, if you want to configure OLS with a directory service.
-olsConfiguration {true | false}	Optional	Specify true to enable and configure Oracle Label Security (OLS) for the PDB, else specify false. Default is false. When true is specified, you can additionally specify the -configureWithOID parameter to configure Oracle Label Security (OLS) with Oracle Internet Directory (OID). This parameter is optional.

Table 19-3 (Cont.) configurePluggableDatabase Parameters

Parameter	Required/ Optional	Description
-pdbTimezone{{+|-}hh:mi| time_zone_region}	Optional	Use this parameter to specify the time zone of the PDB. You can specify the time zone in two ways: • By specifying a displacement from UTC (Coordinated Universal Time—formerly Greenwich Mean Time). The valid range of *hh:mi* is -12:00 to +14:00. • By specifying a time zone region. To see a listing of valid time zone region names, query the TZNAME column of the V$TIMEZONE_NAMES dynamic performance view.
-registerWithDirService{true | false}	Optional	Specify true to register the PDB with a Lightweight Directory Access Protocol (LDAP) service, else specify false. Default is false. When true is specified, the following additional parameters can be provided: • -dirServiceUserName: User name for the LDAP service. • -dirServicePassword: Password for the LDAP service user. • -walletPassword: Password for the database wallet. • -databaseCN: Database common name. • -dirServiceCertificatePath: Directory service certificate file path. • -dirServiceUser: Active Directory account user name.
unregisterWithDirService {true | false}	Optional	Specify true to unregister the PDB with the Lightweight Directory Access Protocol (LDAP) service, else specify false. Default is false. When true is specified, the following additional parameters can be provided: • -dirServiceUserName: User name for the LDAP service. • -dirServicePassword: Password for the LDAP service user. • -walletPassword: Password for the database wallet.
-useWalletForDBCredentials {true | false}	Optional	Specify true to use Oracle Wallet for database credentials, else specify false. Default is false. When true is specified, the following additional parameters can be provided: • -dbCredentialsWalletPassword: Password for the Oracle Wallet account. • -dbCredentialsWalletLocation: Directory location for the Oracle Wallet files. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • oracle.dbsecurity.walletPassword • oracle.dbsecurity.userDNPassword

createCloneTemplate

The `createCloneTemplate` command creates a clone (seed) database template from an existing database.

Syntax and Parameters

Use the `dbca -createCloneTemplate` command with the following syntax:

```
dbca -createCloneTemplate
   -sourceSID source_database_sid | -sourceDB source_database_name
   -templateName new_database_template_name
   [-promptForWalletPassword]
   [-rmanParallelism parallelism_integer_value]
   [-maxBackupSetSizeInMB maximum_backup_set_size_in_MB]
   [-dataFileBackup {true | false}]
   [-datafileJarLocation data_files_backup_directory]
   [-sysDBAUserName SYSDBA_user_name]
   [-sysDBAPassword SYSDBA_user_password]
   [-useWalletForDBCredentials {true | false}
       -dbCredentialsWalletPassword wallet_account_password
       -dbCredentialsWalletLocation wallet_files_directory]
   [-uploadToCloud
       -opcLibPath OPC_library_path
       -opcConfigFile OPC_configuration_file_name
       [-rmanEncryptionPassword rman_encryption_password ]
   [-compressBackup { true | false } ]
   [-walletPassword database_wallet_password]
```

Table 19-4 createCloneTemplate Parameters

Parameter	Required/ Optional	Description
`-sourceSID` *source_database_sid* or `-sourceDB` *source_database_name*	Required	Specify either the source database system identifier (SID) or the source database name.
`-templateName` *new_database_template_name*	Required	Name of the new database template.
`-sysDBAUserName` *SYSDBA_user_name*	Optional	User name of a user having the `SYSDBA` privileges.
`-sysDBAPassword` *SYSDBA_user_password*	Optional	Password of the user having the `SYSDBA` privileges.
`-maxBackupSetSizeInMB` *maximum_backup_set_size_in_MB*	Optional	Maximum backup set size in megabytes.

Table 19-4 (Cont.) createCloneTemplate Parameters

Parameter	Required/ Optional	Description
`-rmanParallelism` *parallelism_integer_value*	Optional	Parallelism integer value for RMAN operations.
`-datafileJarLocation` *data_files_backup_directory*	Optional	Complete directory path to store data files as a backup in a compressed format.
`-dataFileBackup {true | false}`	Optional	Specify `true` to take the data files backup, else specify `false`.
`-useWalletForDBCredentials {true | false}`	Optional	Specify `true` to use Oracle Wallet for database credentials, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters can be provided: • `-dbCredentialsWalletPassword`: Password for the Oracle Wallet account. • `-dbCredentialsWalletLocation`: Directory location for the Oracle Wallet files. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • `oracle.dbsecurity.walletPassword` • `oracle.dbsecurity.userDNPassword`

Table 19-4 (Cont.) createCloneTemplate Parameters

Parameter	Required/ Optional	Description
`-uploadToCloud`	Optional	Creates a clone template and uploads it to Oracle Cloud Infrastructure. The structure and data of the database is stored in the template. DBCA can then use this template to create new databases. To create a template to Oracle Cloud Infrastructure, you must subscribe to the Oracle Database Backup Cloud Service and install the Oracle Database Cloud Backup Module for OCI. Recovery Manager (RMAN) creates a backup containing the details of the clone template. The backup must be encrypted, so you must provide the RMAN encryption password to encrypt backups. This option is only supported on Linux. • `opcLibPath`: Directory in which the Oracle Database Cloud Backup Module for OCI is stored. The backup module is a system backup to tape (SBT) library that is used to integrate on-premise databases with Oracle Cloud Infrastructure. The file name is `libopc.so` on Linux. • `opcConfigFile`: Name, with complete location, of the Oracle Database Cloud Backup Module for OCI configuration file. This file is created when you install the backup module. • `rmanEncryptionPassword`: Password used to encrypt the RMAN backups to Cloud that contian the clone template. See *Administering Oracle Database Backup Cloud Service* for information about installing and configuring the backup module.
`-compressBackup`	Optional	Compresses the backup containing the clone template files.
`-walletPassword`	Optional	Password of the TDE wallet that contains the keys used to encrypt backups. Specify this parameter if Transparent Data Encryption (TDE) must be used to encrypt backups.

createDatabase

The `createDatabase` command creates a database.

Syntax and Parameters

Use the `dbca -createDatabase` command with the following syntax:

```
dbca -createDatabase
    -gdbName global_database_name
    -responseFile | (-gdbName,-templateName)
    -responseFile response_file_directory
    -templateName database_template_name
    [-adminManaged]
    [-characterSet database_character_set]
    [-configureTDE {true | false}
          [-primaryDBTdeWallet value]
          [-sourceTdeWalletPassword value]
```

```
      [-tdeWalletRoot tde_wallet_root_init_parameter]
      [-pdbTDEPassword pdb_tde_wallet_password]
      [-tdeWalletModeForPDB pdb_keystore_type]
      [-tdeAlgorithm TDE_algorithm]
      [-tdeWalletLoginType type_of_wallet_login]
      [-sourcePdbTDEPassword source_pdb_TDE_wallet_password]
      [-tdeWalletPassword TDE_wallet_password]]
[-createAsContainerDatabase {true | false}
    [-numberOfPDBs number_of_pdbs]
    [-pdbName pdb_name]
    [-pdbStorageMAXSizeInMB maximum_storage_size_of_the_pdb]
    [-pdbStorageMAXTempSizeInMB maximum_temporary_storage_size_of_the_pdb]
    [-useLocalUndoForPDBs {true | false}]
    [-pdbAdminPassword pdb_administrator_password]
    [-pdbOptions pdb_options]]
[-createListener new_database_listener]
[-customScripts list_of_custom_sql_scripts]
[-databaseConfigType {SINGLE | RAC | RACONENODE}
        [-RACOneNodeServiceName service_name_for_RAC_One_Node_database]]
[-databaseType {MULTIPURPOSE | DATA_WAREHOUSING | OLTP}]
[-datafileDestination data_files_directory]
[-datafileJarLocation data_files_backup_directory]
[-dbOptions database_options]
[-dvConfiguration {true | false}
    -dvUserName Database_Vault_owner_name
    -dvUserPassword Database_Vault_owner_password
    [-dvAccountManagerName Database_Vault_account_manager_name
    -dvAccountManagerPassword Database_Vault_account_manager_password]]
[-emConfiguration {CENTRAL | NONE}
    [-dbsnmpPassword DBSNMP_user_password]
    [-omsHost Oracle_Management_Server_host_name]
    [-omsPort Oracle_Management_Server_port_number]
    [-emUser EM_administrator_user_name]
    [-emPassword EM_administrator_user_password]
[-enableArchive {true | false}
    [-archiveLogMode {AUTO | MANUAL}]
    [-archiveLogDest archive_log_files_directory]]
[-encryptPDBTablespaces ALL|tablespace_name:{true | false}]
[-encryptTablespaces ALL|tablespace_name:{true | false}]
[-initParams initialization_parameters_list
    [-initParamsEscapeChar initialization_parameters_escape_character]]
[-listeners listeners_list]
[-managementPolicy [AUTOMATIC|RANK]]
[-memoryMgmtType {AUTO | AUTO_SGA | CUSTOM_SGA}]
[-memoryPercentage | -totalMemory]
[-memoryPercentage percentage_of_total_memory_to_assign_to_oracle_database]
[-nationalCharacterSet database_national_character_set]
[-nodelist database_nodes_list]
[-olsConfiguration {true | false}
    [-configureWithOID configure_with_OID_flag]]
[-oracleHomeUserName Oracle_Home_user_name]
[-oracleHomeUserPassword Oracle_Home_user_password]
[-policyManaged
    -serverPoolName server_pool_names
    [-pqPoolName pq_pool_name]
    [-createServerPool new_server_pool_name]
        [-pqPoolName new_pq_pool_name]
        [-force]
        [-pqCardinality pq_cardinality_of_the_new_server_pool]
        [-cardinality cardinality_of_the_new_server_pool]]
[-recoveryAreaDestination recovery_files_directory
```

```
    [-recoveryAreaSize fast_recovery_area_size]]
[-redoLogFileSize maximum_redo_log_file_size]
[-registerWithDirService {true | false}
    [-dirServiceUserName directory_service_user_name]
    [-dirServicePassword directory_service_password]
    [-databaseCN database_common_name]
    [-dirServiceCertificatePath certificate_file_path]
    [-dirServiceUser directory_service_user_name]
    [-ldapDirectoryAccessType ldap_directory_access_type]
    [-useSYSAuthForLDAPAccess use_sys_user_for_ldap_access_flag]
    [-walletPassword wallet_password]]
[-runCVUChecks {true | false}]
[-sampleSchema {true | false}]
[-sid database_system_identifier]
[-storageType {FS | ASM }
    [-asmsnmpPassword ASMSNMP_password]
    -datafileDestination database_files_directory]
[-sysPassword SYS_user_password]
[-systemPassword SYSTEM_user_password]
[-templateFromCloud
    -opcLibPath OPC_library_path
    -opcConfigFile OPC_config_file_name
    [-rmanDecryptionPassword rman_decryption_password]]
[-totalMemory total_memory_to_assign_to_oracle_database_in_MB]
[-useOMF {true | false}]
[-useWalletForDBCredentials { true | false}
     -dbCredentialsWalletLocation directory_containing_wallet_files
     [-dbCredentialsWalletPassword password_to_open_wallet]]
[-variables variables_list]
[-variablesFile variables_file]
```

Table 19-5 createDatabase Parameters

Parameter	Required/ Optional	Description
`-gdbName` *global_database_name*	Required	Global database name in the form *database_name.domain_name*.
`-responseFile` *response_file_directory*	Required	Absolute directory path of the response file.
`-templateName` *database_template_name*	Required	Name of an existing database template in the default location or the complete path to a database template that is not in the default location.
`-adminManaged`	Optional	Administrator-managed database.
`-characterSet` *database_character_set*	Optional	Character set of the database.

Table 19-5 (Cont.) createDatabase Parameters

Parameter	Required/ Optional	Description
`configureTDE`	Optional	Specify `true` to configure TDE during the database creation. Only software wallets are supported. You can create a wallet for the entire CDB or for a PDB. • `primaryDBTdeWallet`: This option is not applicable when creating a database. • `sourceTdeWalletPassword`: If the template that is being used is from a database that uses encryption, or if you are duplicating a database, specify the password of the wallet in the source database. • `tdeWalletModeForPDB`: Specify `UNITED` to create a wallet for the entire CDB. Use `ISOLATED` to create a wallet for a PDB. • `tdeAlgorithm`: Algorithm used to encrypt data. Can be one of the following: 3DES168, AES128, AES192, AES256. • `tdeWalletLoginType`: Type of software wallet. `PASSWORD` or `AUTO_LOGIN` or `LOCAL_AUTO_LOGIN`. • `tdeWalletLocation`: Location in which the TDE wallet is stored. • `tdeWalletPassword`: The password used to open the wallet. This parameter is mandatory. Note: Isolated wallets are supported only in Oracle Cloud or Exadata environments.
`-createAsContainerDatabase {true | false}`	Optional	Specify `true` to create a CDB. Specifying `false` is not supported starting with Oracle Database Release 20.3. When `true` is specified, the following additional parameters are optional: • `-numberOfPDBs`: Number of PDBs to create. The default is 0 (zero). • `-pdbName`: Base name of each PDB. A number is appended to each name if `-numberOfPDBs` is greater than 1. This parameter must be specified if `-numberOfPDBs` is greater than 0 (zero). • `-pdbStorageMAXSizeInMB`: Maximum storage size for the PDBs in megabytes. • `-pdbStorageMAXTempSizeInMB`: Maximum temporary storage size for the PDBs in megabytes. • `-useLocalUndoForPDBs {true | false}`: Specify whether local undo should be used for the PDBs. • `-pdbAdminPassword`: PDB administrator password. • `-pdbOptions`: Specify PDB options as comma separated list in `name:value` format. Example: `JSERVER:true`, `DV:false`
`-createListener` *`new_database_listener`*	Optional	Database listener to register the database in the form *listener_name:port*.
`-customScripts` *`custom_scripts_list`*	Optional	Specify a comma separated list of SQL scripts that needs to be run after the database creation. The scripts are run in the order they are listed.

Table 19-5 (Cont.) createDatabase Parameters

Parameter	Required/ Optional	Description
`-databaseConfigType {SINGLE | RAC | RACONENODE}`	Optional	Specify one of the following database configuration types: • `SINGLE`: Single individual database. • `RAC`: Oracle RAC database. • `RACONENODE`: Oracle RAC One Node database. For Oracle RAC One Node database, you can specify the service name using the `-RACOneNodeServiceName` parameter.
`-databaseType {MULTIPURPOSE | DATA_WAREHOUSING | OLTP}`	Optional	Specify `MULTIPURPOSE` if the database is for both OLTP and data warehouse purposes. Specify `DATA_WAREHOUSING` if the primary purpose of the database is a data warehouse. Specify `OLTP` if the primary purpose of the database is online transaction processing.
`-datafileDestination` *`data_files_directory`*	Optional	Complete path to the location of the database data files.
`-datafileJarLocation` *`data_files_backup_directory`*	Optional	Absolute directory path of the database backup data files stored in a compressed RMAN backup format (files with .dfb extensions).
`-dbOptions` *`database_options`*	Optional	Specify database options as comma separated list of `name:value` pairs. Example: `JSERVER:true,DV:false`
`-dvConfiguration {true | false}`	Optional	Specify `true` to enable and configure Database Vault, else specify `false`. Default is `false`. When `true` is specified, the following additional Database Vault parameters are required: • `-dvUserName`: Specify Database Vault owner name. • `-dvUserPassword`: Specify Database Vault owner password. • `-dvAccountManagerName`: Specify Database Vault account manager name. • `-dvAccountManagerPassword`: Specify Database Vault account manager password.
`-emConfiguration {CENTRAL | NONE}`	Optional	Enterprise Manager configuration settings. When `CENTRAL` is specified, specify the following additional parameters: • `-dbsnmpPassword`: DBSNMP user password. • `-omsHost`: Oracle Management Server host name. • `-omsPort`: Oracle Management Server port number. • `-emUser`: User name for Enterprise Manager administrator. • `-emPassword`: Password for Enterprise Manager administrator.

Table 19-5 (Cont.) createDatabase Parameters

Parameter	Required/ Optional	Description
-enableArchive {true | false}	Optional	Specify true to enable log file archive, else specify false. Default is false. When true is specified, the following additional parameters can be provided: • -archiveLogMode {AUTO | MANUAL}: Specify either the automatic archive mode or the manual archive mode. Default is automatic archive mode. • -archiveLogDest: Directory path for storing the archive log files.
-encryptPDBTablespaces	Optional	If you have configured TDE using -configureTDE, you can use this parameter to encrypt all of some of the tablespaces at the PDB and CDB level. Specify the following additional parameters: • ALL: Encrypt all tablespaces • *tablespace_name*: Comma-separated list of tablespaces with a value of TRUE or FALSE for each
-encryptTablespaces	Optional	If you have configured TDE using -configureTDE, you can use this parameter to encrypt all or some of the tablespaces at the CDB level. Specify the following additional parameters: • ALL: Encrypt all tablespaces • *tablespace_name*: Comma-separated list of tablespaces with a value of TRUE or FALSE for each
-initParams *initialization_parameters_list*	Optional	A comma-separated list of name=value pairs of initialization parameter values for the database. You can additionally provide the -initParamsEscapeChar parameter for using a specific escape character between multiple values of an initialization parameter. If an escape character is not specified, backslash (/) is used as the default escape character.
-listeners *listeners_list*	Optional	A comma-separated list of listeners for the database.
-managementPolicy	Optional	Use this parameter to set the database management policy. Optionally, specify the management policy type: • AUTOMATIC • RANK If -adminManaged is selected, -managementPolicy defaults to AUTOMATIC.

Table 19-5 (Cont.) createDatabase Parameters

Parameter	Required/ Optional	Description
-memoryMgmtType {AUTO | AUTO_SGA | CUSTOM_SGA}	Optional	Specify one of the following memory management types: • AUTO: Automatic memory management for SGA and PGA. • AUTO_SGA: Automatic shared memory management for SGA. • CUSTOM_SGA: Manual shared memory management for SGA. **Note:** If the total physical memory of a database instance is greater than 4 GB, then you cannot specify the Automatic Memory Management option AUTO during the database installation and creation. Oracle recommends that you specify the Automatic Shared Memory Management option AUTO_SGA in such environments.
-memoryPercentage *percentage_of_total_memory_to_assign_to_oracle_database* or -totalMemory *total_memory_to_assign_to_oracle_database_in_MB*	Optional	Specify either -memoryPercentage or -totalMemory. • -memoryPercentage The percentage of physical memory that can be used by the database. • -totalMemory. Total amount of physical memory, in megabytes, that can be used by the database.
-nationalCharacterSet *database_national_character_set*	Optional	National character set of the database.
-nodelist *database_nodes_list*	Optional	List of database nodes separated by comma.
-olsConfiguration {true | false}	Optional	Specify true to enable and configure Oracle Label Security (OLS), else specify false. Default is false. When true is specified, you can additionally specify the -configureWithOID parameter, if you want to configure Oracle Label Security (OLS) with Oracle Internet Directory (OID).
-oracleHomeUserName *Oracle_Home_user_name* -oracleHomeUserPassword *Oracle_Home_user_password*	Optional	Oracle Home user name and password.

Table 19-5 (Cont.) createDatabase Parameters

Parameter	Required/ Optional	Description
`-policyManaged`	Optional	Policy-managed database. You can specify the following additional parameters: • `-serverPoolName`: Specify the single server pool name when creating a new server pool or specify a comma separated list of existing server pools. • `-pqPoolName`: Specify the PQ pool name. • `-createServerPool`: Specify this parameter for creating a new server pool. : – `-pqPoolName`: Specify the PQ pool name. – `-force`: Specify this parameter to create the server pool by force when adequate free servers are not available. – `-pqCardinality`: Specify the PQ cardinality of the new server pool. – `-cardinality`: Specify the cardinality of the new server pool.
`-recoveryAreaDestination` *fast_recovery_area_directory*	Optional	Destination directory for the Fast Recovery Area, which is a backup and recovery area. Specify `NONE` to disable Fast Recovery Area. Additionally, you can specify the Fast Recovery Area size in megabytes using the parameter `-recoveryAreaSize`. This parameter is optional.
`-redoLogFileSize` *maximum_size_of_redo_log_file*	Optional	Size of each online redo log in megabytes.
`-registerWithDirService {true | false}`	Optional	Specify `true` to register with a Lightweight Directory Access Protocol (LDAP) service, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters are required: • `-dirServiceUserName`: Username for the LDAP service. • `-dirServicePassword`: Password for the LDAP service. • `-databaseCN`: Database common name. • `-dirServiceCertificatePath`: Directory path to the certificate file to use when configuring SSL between the database and the directory service. • `-dirServiceUser`: Directory service user name. • `-ldapDirectoryAccessType {PASSWORD | SSL}`: LDAP directory access type. • `-useSYSAuthForLDAPAccess {true | false}`: Specify whether to use SYS user authentication for LDAP acces. • `-walletPassword`: Password for the database wallet.

Table 19-5 (Cont.) createDatabase Parameters

Parameter	Required/ Optional	Description
-runCVUChecks {true | false}	Optional	Specify true to run Cluster Verification Utility checks periodically for Oracle RAC databases, else specify false. Default is false.
-sampleSchema {true | false}	Optional	Specify true to include the HR sample schema (EXAMPLE tablespace) in your database. Oracle guides and educational materials contain examples based on the sample schemas. Oracle strongly recommends that you do not install the sample schemas in a production database. Specify false to create the database without the HR sample schema. Default is false.
-sid *database_system_identifier*	Optional	Database system identifier (SID). The SID uniquely identifies the instance that runs the database. If it is not specified, then it defaults to the database name.
-storageType {FS | ASM}	Optional	Specify the storage type of either FS or ASM. • FS: File system storage type. When FS is specified, your database files are managed by the file system of your operating system. You can specify the directory path where the database files are to be stored using a database template or the -datafileDestination parameter. Oracle Database can create and manage the actual files. • ASM: Oracle Automatic Storage Management (Oracle ASM) storage type. When ASM is specified, your database files are placed in Oracle ASM disk groups. Oracle Database automatically manages database file placement and naming. When ASM is specified, you can also specify the ASMSNMP password using the -asmsnmpPassword parameter. This parameter is optional.
-sysPassword *SYS_user_password*	Optional	SYS user password for the new database.
-systemPassword *SYSTEM_user_password*	Optional	SYSTEM user password for the new database.

Table 19-5 (Cont.) createDatabase Parameters

Parameter	Required/ Optional	Description
`-templateFromCloud`	Optional	Creates a database using the clone template that is stored in Oracle Cloud Infrastructure. This option is supported only for Linux. • `opcLibPath`: Directory in which the Oracle Database Cloud Backup Module for OCI is stored. The backup module is a system backup to tape (SBT) library that is used to integrate an on-premise database with Oracle Cloud Infrastructure. You must install the Oracle Database Cloud Backup Module for OCI before running this command. • `opcConfigFile`: Name, with complete location, of the Oracle Database Cloud Backup Module for OCI configuration file. This file is created when you install the backup module. • `rmanDecryptionPassword`: Password that must be used to decrypt the RMAN template file stored in Oracle Cloud Infrastructure. This is the same password that was used when creating an RMAN backup of the template. See *Administering Oracle Database Backup Cloud Service* for information about installing and configuring the backup module.
`-useOMF` `{true | false}`	Optional	Specify `true` to use Oracle-Managed Files (OMF), else specify `false`. Default is `false`.
`-useWalletForDBCredentials` `{true | false}`	Optional	Specify `true` to use Oracle Wallet for database credentials, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters can be provided: • `-dbCredentialsWalletLocation`: Directory location for the Oracle Wallet files. • `-dbCredentialsWalletPassword`: Password for the Oracle Wallet account. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • `oracle.dbsecurity.walletPassword` • `oracle.dbsecurity.userDNPassword`
`-variables` *`variables_list`*	Optional	A comma-separated list of `name=value` pairs for the variables in the database template.
`-variablesFile` *`variables_file`*	Optional	Name of the variables file with the complete directory path in the database template.

See Also:

Oracle Database Sample Schemas

createDuplicateDB

The `createDuplicateDB` command creates a duplicate of an Oracle database.

Prerequisites

The following are the prerequisites for using the `createDuplicateDB` command:

- The database to be duplicated is in the *archivelog* mode.
- If the database to be duplicated is in a remote server, then there must be connectivity from the system where DBCA is running to the remote server.

Syntax and Parameters

Use the `dbca -createDuplicateDB` command with the following syntax:

```
dbca -createDuplicateDB
    -gdbName global_database_name
    -primaryDBConnectionString easy_db_connection_string
    -sid database_system_identifier
    [-initParams initialization_parameters
        [-initParamsEscapeChar initialization_parameters_escape_character]]
    [-sysPassword SYS_user_password]
    [-policyManaged | -adminManaged]
    [-policyManaged
        -serverPoolName server_pool_names
        [-pqPoolName pq_pool_name]
        [-createServerPool new_server_pool_name
            [-pqPoolName new_pq_pool_name]
            [-force]
            [-pqCardinality pq_cardinality_of_the_new_server_pool]
            [-cardinality cardinality_of_the_new_server_pool]]]
    [-adminManaged]
    [-nodelist database_nodes_list]
    [-datafileDestination data_files_directory]
    [-recoveryAreaDestination recovery_files_directory
        [-recoveryAreaSize fast_recovery_area_size]]
    [-databaseConfigType {SINGLE | RAC | RACONENODE}
        [-RACOneNodeServiceName service_name_for_RAC_One_Node_database]]
    [-useOMF {true | false}]
    [-storageType {FS | ASM}
        [-asmsnmpPassword ASMSNMP_password]
        -datafileDestination database_files_directory]
    [-createListener new_database_listener]
    [-createAsStandby
        [-dbUniqueName db_unique_name_for_standby_database]]
    [-customScripts custom_sql_scripts_to_run_after_database_creation]
    [-useWalletForDBCredentials {true | false}
        -dbCredentialsWalletPassword wallet_account_password
        -dbCredentialsWalletLocation wallet_files_directory]
    [-configureTDE <true | false> ]
          [-primaryDBTdeWallet value]
          [-sourceTdeWalletPassword value]
          [-tdeWalletRoot tde_wallet_root_init_parameter]
          [-pdbTDEPassword pdb_tde_wallet_password]
          [-tdeWalletModeForPDB pdb_keystore_type]
          [-tdeAlgorithm TDE_algorithm]
          [-tdeWalletLoginType type_of_wallet_login]
```

```
[-sourcePdbTDEPassword source_pdb_TDE_wallet_password]
[-tdeWalletPassword TDE_wallet_password]
```

Table 19-6 createDuplicateDB Parameters

Parameter	Required/ Optional	Description
`-gdbName` *global_database_name*	Required	Global database name of the duplicate database in the form *database_name.domain_name*.
`-primaryDBConnectionString` *easy_db_connection_string*	Required	Easy connection string to connect to the database to be duplicated. Easy connection string must be in the following format: *"host[:port][/service_name][:server][/instance_name]"*
`-sid` *database_system_identifier*	Required	Database system identifier (SID) of the duplicate database. The SID uniquely identifies the instance that runs the database. If it is not specified, then it defaults to the database name.
`-initParams` *initialization_parameters_list*	Optional	A comma-separated list of `name=value` pairs of initialization parameter values for the database. You can additionally provide the `-initParamsEscapeChar` parameter for using a specific escape character between multiple values of an initialization parameter. If an escape character is not specified, backslash (/) is used as the default escape character.
`-sysPassword` *SYS_user_password*	Optional	`SYS` user password.
`-policyManaged`	Optional	Policy-managed database. **Note:** You can specify either policy-managed database or administrator-managed database. You can specify the following additional parameters: • `-serverPoolName`: Specify the single server pool name when creating a new server pool or specify a comma separated list of existing server pools. • `-pqPoolName`: Specify the PQ pool name. • `-createServerPool`: Specify this parameter for creating a new server pool. – `-pqPoolName`: Specify the PQ pool name. – `-force`: Specify this parameter to create the server pool by force when adequate free servers are not available. – `-pqCardinality`: Specify the PQ cardinality of the new server pool. – `-cardinality`: Specify the cardinality of the new server pool.
`-adminManaged`	Optional	Administrator-managed database. **Note:** You can specify either policy-managed database or administrator-managed database.
`-nodelist` *database_nodes_list*	Optional	For administrator-managed database, specify database nodes separated by comma.

Table 19-6 (Cont.) createDuplicateDB Parameters

Parameter	Required/ Optional	Description
-datafileDestination *data_files_directory*	Optional	Complete directory path for database data files.
-recoveryAreaDestination *fast_recovery_area_directory*	Optional	Destination directory for the Fast Recovery Area, which is a backup and recovery area. Specify NONE to disable Fast Recovery Area. Additionally, you can specify the Fast Recovery Area size in megabytes using the parameter -recoveryAreaSize. This parameter is optional.
-databaseConfigType {SINGLE | RAC | RACONENODE}	Optional	Specify one of the following database configuration types: • SINGLE: Single individual database. • RAC: Oracle RAC database. • RACONENODE: Oracle RAC One Node database. For Oracle RAC One Node database, you can specify the service name using the -RACOneNodeServiceName parameter.
-useOMF {true | false}	Optional	Specify true to use Oracle-Managed Files (OMF), else specify false. Default is false.
-storageType {FS | ASM}	Optional	Specify the storage type of either FS or ASM. • FS: File system storage type. When FS is specified, your database files are managed by the file system of your operating system. You can specify the directory path where the database files are to be stored using a database template or the -datafileDestination parameter. Oracle Database can create and manage the actual files. • ASM: Oracle Automatic Storage Management (Oracle ASM) storage type. When ASM is specified, your database files are placed in Oracle ASM disk groups. Oracle Database automatically manages database file placement and naming. When ASM is specified, you can also specify the ASMSNMP password using the -asmsnmpPassword parameter. This parameter is optional.
-createListener *new_database_listener*	Optional	Database listener to register the database in the form *listener_name:port*.
-createAsStandby	Optional	Specifies that the duplicate database is a standby database for the primary database. Optionally, use the -dbUniqueName parameter to set the unique database name for the standby database. If the -dbUniqueName parameter is not specified, then the value of the DB_NAME initialization parameter is used.
-customScripts *custom_sql_scripts_to_run_after_database_creation*	Optional	A comma separated list of SQL scripts that should be run after the duplicate database is created. The scripts are run in the order listed.

Table 19-6 (Cont.) createDuplicateDB Parameters

Parameter	Required/ Optional	Description
-useWalletForDBCredentials {true | false}	Optional	Specify `true` to use Oracle Wallet for database credentials, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters can be provided: • `-dbCredentialsWalletPassword`: Password for the Oracle Wallet account. • `-dbCredentialsWalletLocation`: Directory location for the Oracle Wallet files. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • `oracle.dbsecurity.walletPassword` • `oracle.dbsecurity.userDNPassword`
`configureTDE`	Optional	Specify true to configure TDE during the database creation. Only software wallets are supported. You can create a wallet for the entire CDB or for a PDB. • `primaryDBTdeWallet`: This option is not applicable when creating a database. • `sourceTdeWalletPassword`: If the template that is being used is from a database that uses encryption, specify the password of the wallet in the source database. • `tdeWalletModeForPDB`: Specify UNITED to create a wallet for the entire CDB. Use ISOLATED to create a wallet for a PDB. • `tdeAlgorithm`: Algorithm used to encrypt data. Can be one of the following: 3DES168, AES128, AES192, AES256. • `tdeWalletLoginType`: Type of software wallet. PASSWORD or AUTO_LOGIN or LOCAL_AUTO_LOGIN. • `tdeWalletLocation`: Location in which the TDE wallet is stored. • `tdeWalletPassword`: The password used to open the wallet. This parameter is mandatory. Note: Isolated wallets are supported only in Oracle Cloud or Exadata environments.

Related Topics

- *Oracle Data Guard Concepts and Administration*

createPluggableDatabase

The `createPluggableDatabase` command creates a pluggable database (PDB) in a multitenant container database (CDB).

Syntax and Parameters

Use the `dbca -createPluggableDatabase` command with the following syntax:

```
dbca -createPluggableDatabase
   -pdbName name_of_the_pdb_to_create
   -sourceDB cdb_sid
   [-configureTDE {true | false}
          [-primaryDBTdeWallet value]
          [-sourceTdeWalletPassword value]
          [-tdeWalletRoot tde_wallet_root_init_parameter]
          [-pdbTDEPassword pdb_tde_wallet_password]
          [-tdeWalletModeForPDB pdb_keystore_type]
          [-tdeAlgorithm TDE_algorithm]
          [-tdeWalletLoginType type_of_wallet_login]
          [-sourcePdbTDEPassword source_pdb_TDE_wallet_password]
          [-tdeWalletPassword TDE_wallet_password]]
   [-createAsClone {true | false}]
   [-createFromRemotePDB
      -pdbName name_of_the_local_pdb_to_create
      -sourceDB database_name_of_the_local_pdb
      -remotePDBName name_of_the_remote_pdb
      -remoteDBConnString db_connection_string_of_the_remote_pdb
      -sysDBAUserName name_of_the_sysdba_user
      -sysDBAPassword password_of_the_sysdba_user
      -dbLinkUsername name_of_the_dblink_user_of_the_remote_pdb
      -dbLinkUserPassword
password_of_the_dblink_user_of_the_remote_pdb]
   [-createPDBFrom {DEFAULT | FILEARCHIVE | RMANBACKUP | USINGXML |
PDB}
      [-pdbArchiveFile pdb_archive_file_name_with_directory_path]
      [-PDBBackUpfile pdb_backup_file_name_with_directory_path]
      [-PDBMetadataFile pdb_metadata_file_name_with_directory_path]
      [-pdbAdminUserName pdb_administrator_name]
      [-pdbAdminPassword pdb_administrator_password]
      [-createNewPDBAdminUser {true | false}]
      [-sourceFileNameConvert method_to_locate_pdb_files]
      [-fileNameConvert names_of_pdb_files]
      [-pdbStorageMAXSizeInMB maximum_storage_size_for_the_pdb_in_MB]
      [-pdbStorageMAXTempSizeInMB
maximum_temporary_storage_size_for_the_pdb_in_MB]
      [-workArea
directory_to_unzip_PDB_archive_files_for_FILEARCHIVE_option]
      [-copyPDBFiles  {true | false}]
      [-sourcePDB name_of_the_pdb_to_clone]
   [-createUserTableSpace {true | false)]
   [-customScripts custom_sql_scripts_to_run_after_PDB_creation]
   [-dvConfiguration {true | false}
```

```
      -dvUserName Database_Vault_owner_name
      -dvUserPassword Database_Vault_owner_password
      [-dvAccountManagerName Database_Vault_account_manager_name]
      [-dvAccountManagerPassword Database_Vault_account_manager_password]]
   [-encryptPDBTablespaces ALL|tablespace_name:{true | false}]
    [-encryptTablespaces ALL|tablespace_name:{true | false}]
   [-lbacsysPassword LBACSYS_user_password]
   [-pdbDatafileDestination pdb_data_files_directory]
   [-pdbStorageMAXSizeInMB maximum_storage_size_for_the_pdb_in_MB]
   [-pdbStorageMAXTempSizeInMB
maximum_temporary_storage_size_for_the_pdb_in_MB]
   [-pdbTimezone {{+|-}hh:mi|time_zone_region}]
   [-pdbUseMultipleBackup number_of_pdb_backups_to_create]
   [-registerWithDirService {true | false}
      -dirServiceUserName directory_service_user_name
      [-dirServicePassword directory_service_user_password]
      [-databaseCN directory_service_database_common_name]
      [-dirServiceCertificatePath certificate_file_directory_path]
      [-dirServiceUser active_directory_account_user_name]
      [-walletPassword wallet_password]]
   [-useMetaDataFileLocation {true | false}]
   [-useWalletForDBCredentials {true | false}
      -dbCredentialsWalletPassword wallet_account_password
      -dbCredentialsWalletLocation wallet_files_directory]
```

Table 19-7 createPluggableDatabase Parameters

Parameter	Required/ Optional	Description
-pdbName *name_of_the_pdb_to_create*	Required	Name of the new PDB to create. **Note:** For Oracle RAC databases, the PDB name must be unique in the cluster.
-sourceDB *cdb_sid*	Required	The database system identifier (SID) of the CDB.

Table 19-7 (Cont.) createPluggableDatabase Parameters

Parameter	Required/ Optional	Description
`configureTDE`	Optional	Specify `true` to configure TDE during the database creation. Only software wallets are supported. You can create a wallet for the entire CDB or for a PDB. • `primaryDBTdeWallet`: This option is not applicable when creating a database. • `sourceTdeWalletPassword`: If the template that is being used is from a database that uses encryption, or if you are duplicating a database, specify the password of the wallet in the source database. • `tdeWalletModeForPDB`: Specify `UNITED` to create a wallet for the entire CDB. Use `ISOLATED` to create a wallet for a PDB. • `tdeAlgorithm`: Algorithm used to encrypt data. Can be one of the following: 3DES168, AES128, AES192, AES256. • `tdeWalletLoginType`: Type of software wallet. `PASSWORD` or `AUTO_LOGIN` or `LOCAL_AUTO_LOGIN`. • `tdeWalletLocation`: Location in which the TDE wallet is stored. • `tdeWalletPassword`: The password used to open the wallet. This parameter is mandatory. Note: Isolated wallets are supported only in Oracle Cloud or Exadata environments.
`-createAsClone {true | false}`	Optional	Specify `true` if the files you plan to use to create the new PDB are the same files that were used to create an existing PDB. Specifying `true` ensures that Oracle Database generates unique PDB DBID, GUID, and other identifiers expected for the new PDB. Specify `false`, the default, if the files you plan to use to create the new PDB are not the same files that were used to create an existing PDB.

Table 19-7 (Cont.) createPluggableDatabase Parameters

Parameter	Required/ Optional	Description
`-createFromRemotePDB`	Optional	Create a PDB by cloning a remote PDB. Specify the following parameters: • `-pdbName`: Name of the local PDB to create. • `-sourceDB`: Database name of the local PDB. • `-remotePDBName`: Name of the remote PDB to clone. • `-remoteDBConnString`: Database connection string of the remote PDB. • `-sysDBAUserName`: Name of the SYSDBA user. • `-sysDBAPassword`: Password of the SYSDBA user. • `-dbLinkUsername`: Name of the database link user of the remote PDB. • `-dbLinkUserPassword`: Password of the database link user of the remote PDB. **Note:** • The database user of the local CDB must have the `CREATE PLUGGABLE DATABASE` privileges in the root container. • The remote CDB must be in the *local undo* mode. • The remote PDB must be in the *archivelog* mode. • The database user of the remote PDB to which the database link connects to must have the `CREATE PLUGGABLE DATABASE` and `CREATE SESSION` privileges.

Table 19-7 (Cont.) createPluggableDatabase Parameters

Parameter	Required/ Optional	Description
`-createPDBFrom {DEFAULT | FILEARCHIVE | RMANBACKUP | USINGXML | PDB}`	Optional	Specify `DEFAULT` to create the PDB from the CDB's seed. When you specify `DEFAULT`, the following additional parameters are required: • `-pdbAdminUserName`: The user name of the PDB's local administrator. • `-pdbAdminPassword`: The password for the PDB's local administrator. Specify `FILEARCHIVE` to create the PDB from an unplugged PDB's files. When you specify `FILEARCHIVE`, the following additional parameters are required: • `-pdbArchiveFile`: Complete path and name for unplugged PDB's archive file. The archive file contains all of the files for the PDB, including its XML metadata file and its data files. Typically, the archive file has a `.gz` extension. • `-createNewPDBAdminUser`: Specify `true` to create a new PDB administrator or `false` to avoid creating a new PDB administrator. • `-workArea`: Specify the directory location where the PDB archive files need to be unzipped. Specify `RMANBACKUP` to create the PDB from a Recovery Manager (RMAN) backup. When you specify `RMANBACKUP`, the following additional parameters are required: • `-pdbBackUpfile`: Complete path and name for the PDB backup file. • `-pdbMetadataFile`: Complete path and name for the PDB's XML metadata file. Specify `USINGXML` to create the PDB from an unplugged PDB's XML metadata file. When you specify `USINGXML`, the following additional parameter is required: • `-pdbMetadataFile`: Complete path and name for the PDB's XML metadata file. Specify `PDB` to create a new PDB by cloning an existing PDB. When you specify `PDB`, the following additional parameter is required: • `-sourcePDB`: Name of an existing PDB to clone. Specify the following optional parameters, if required: • `-sourceFileNameConvert`: This parameter specifies how to locate PDB files listed in the PDB XML metadata file. See `SOURCE_FILE_NAME_CONVERT` clause of the `CREATE PLUGGABLE DATABASE` statement described in *Oracle Multitenant Administrator's Guide*. • `-fileNameConvert`: This parameter specifies the names of the PDB's files. See `FILE_NAME_CONVERT` clause of the `CREATE PLUGGABLE DATABASE` statement described in *Oracle Multitenant Administrator's Guide*.

Table 19-7 (Cont.) createPluggableDatabase Parameters

Parameter	Required/ Optional	Description
		• `-pdbStorageMAXSizeInMB`: Specify the maximum storage size for the PDB in megabytes. See information about PDB storage described in *Oracle Multitenant Administrator's Guide*. • `-pdbStorageMAXTempSizeInMB`: Specify the maximum temporary storage size for the PDB in megabytes. • `-copyPDBFiles {true | false}`: Specify `true` if the PDB data files need to be copied, else specify `false`.
`-createUserTableSpace {true | false}`	Optional	Specify `true` if a default user tablespace needs to be created in the new PDB.
`-customScripts` *`lcustom_sql_scripts_to_run_after_PDB_creation`*	Optional	Specify a list of custom SQL scripts to run after the PDB creation.
`-dvConfiguration {true | false}`	Optional	Specify `true` to enable and configure Database Vault, else specify `false`. Default is `false`. When `true` is specified, the following additional Database Vault parameters are required: • `-dvUserName`: Specify the Database Vault owner name. • `-dvUserPassword`: Specify Database Vault owner password. • `-dvAccountManagerName`: Specify a separate Database Vault account manager name. • `-dvAccountManagerPassword`: Specify the Database Vault account manager password.
`-encryptPDBTablespaces`	Optional	If you have configured TDE using `-configureTDE`, you can use this parameter to encrypt all of some of the tablespaces at the PDB and CDB level. Specify the following additional parameters: • `ALL`: Encrypt all tablespaces • *`tablespace_name`*: Comma-separated list of tablespaces with a value of `TRUE` or `FALSE` for each
`-encryptTablespaces`	Optional	If you have configured TDE using `-configureTDE`, you can use this parameter to encrypt all or some of the tablespaces at the CDB level. Specify the following additional parameters: • `ALL`: Encrypt all tablespaces • *`tablespace_name`*: Comma-separated list of tablespaces with a value of `TRUE` or `FALSE` for each
`-lbacsysPassword` *`LBACSYS_user_password`*	Optional	Specify the `LBACSYS` user password if you want to configure OLS with a directory service.

Table 19-7 (Cont.) createPluggableDatabase Parameters

Parameter	Required/ Optional	Description
`-pdbDatafileDestination` *pdb_data_files_directory*	Optional	Compete directory path to the new PDB data files. When this parameter is not specified, either Oracle Managed Files or the `PDB_FILE_NAME_CONVERT` initialization parameter specifies how to generate the names and locations of the files. If you use both Oracle Managed Files and the `PDB_FILE_NAME_CONVERT` initialization parameter, then Oracle Managed Files takes precedence. When this parameter is not specified, Oracle Managed Files is not enabled, and the `PDB_FILE_NAME_CONVERT` initialization parameter is not set, by default a path to a subdirectory with the name of the PDB in the directory for the root's files is used.
`-pdbStorageMAXSizeInMB` *maximum_storage_size_for_the_pdb_in_MB*	Optional	Specify the maximum storage size for the PDB in megabytes.
`-pdbStorageMAXTempSizeInMB` *maximum_temporary_storage_size_for_the_pdb_in_MB*	Optional	Specify the maximum temporary storage size for the PDB in megabytes.
`-pdbTimezone{{+|-}hh:mi|time_zone_region}`	Optional	Use this parameter to specify the time zone of the PDB. You can specify the time zone in two ways: • By specifying a displacement from UTC (Coordinated Universal Time—formerly Greenwich Mean Time). The valid range of *hh:mi* is -12:00 to +14:00. • By specifying a time zone region. To see a listing of valid time zone region names, query the `TZNAME` column of the `V$TIMEZONE_NAMES` dynamic performance view.
`-pdbUseMultipleBackup` *number_of_pdb_backups_to_create*	Optional	Specify the number of PDB backups to create.
`-registerWithDirService {true | false}`	Optional	Specify `true` to register the PDB with a Lightweight Directory Access Protocol (LDAP) service, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters are required: • `-dirServiceUserName`: User name for the LDAP service. • `-dirServicePassword`: Password for the LDAP service. • `-dirServiceUser`: User name for the Active Directory account. • `-dirServiceCertificatePath`: Certificate file path of the directory service. • `-databaseCN`: Common name of the directory service database. • `-walletPassword`: Password for the database wallet.

Table 19-7 (Cont.) createPluggableDatabase Parameters

Parameter	Required/ Optional	Description
-useMetaDataFileLocation {true | false}	Optional	Specify `true` to use the data file path defined in XML metadata file within a PDB archive when extracting data files. Specify `false`, the default, to not use the data file path defined in XML metadata file within a PDB archive when extracting data files.
-useWalletForDBCredentials {true | false}	Optional	Specify `true` to use Oracle Wallet for database credentials, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters can be provided: • `-dbCredentialsWalletLocation`: Directory location for the Oracle Wallet files. • `-dbCredentialsWalletPassword`: Password for the Oracle Wallet account. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • `oracle.dbsecurity.walletPassword` • `oracle.dbsecurity.userDNPassword`

createTemplateFromDB

The `createTemplateFromDB` command creates a database template from an existing database.

Syntax and Parameters

Use the `dbca -createTemplateFromDB` command with the following syntax:

```
dbca -createTemplateFromDB
   -sourceDB source_database_sid
   -templateName new_database_template_name
   -sysDBAUserName SYSDBA_user_name
   -sysDBAPassword SYSDBA_user_password
   [-maintainFileLocations {true | false}]
   [-connectionString easy_connect_string]
   [-useWalletForDBCredentials {true | false}
       -dbCredentialsWalletPassword wallet_account_password
       -dbCredentialsWalletLocation wallet_files_directory]
```

Table 19-8 createTemplateFromDB Parameters

Parameter	Required/ Optional	Description
-sourceDB *source_database_sid*	Required	The source database system identifier (SID).

Table 19-8 (Cont.) createTemplateFromDB Parameters

Parameter	Required/ Optional	Description
`-templateName` *new_database_template_name*	Required	Name of the new database template.
`-sysDBAUserName` *SYSDBA_user_name*	Required	User name of a user that has `SYSDBA` privileges.
`-sysDBAPassword` *SYSDBA_user_password*	Required	Password of the user that has `SYSDBA` privileges.
`-maintainFileLocations {true | false}`	Optional	Specify `true` to use the file locations of the database in the template. Specify `false`, the default, to use different file locations in the template. The file locations are determined by Oracle Flexible Architecture (OFA).
`-connectionString` *easy_connect_string*	Optional	Easy connect string for connecting to a remote database in the following format: *"host[:port][/service_name][:server][/instance_name]"*
`-useWalletForDBCredentials {true | false}`	Optional	Specify `true` to use Oracle Wallet for database credentials, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters can be provided: • `-dbCredentialsWalletPassword`: Password for the Oracle Wallet account. • `-dbCredentialsWalletLocation`: Directory location for the Oracle Wallet files. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • `oracle.dbsecurity.walletPassword` • `oracle.dbsecurity.userDNPassword`

createTemplateFromTemplate

The `createTemplateFromTemplate` command creates a database template from an existing database template.

Syntax and Parameters

Use the `dbca -createTemplateFromTemplate` command with the following syntax:

```
dbca -createTemplateFromTemplate
    -sourcetemplateName existing_template_name
    -templateName new_template_name
    [-variables variables_list]
    [-characterSet database_character_set]
    [-nationalCharacterSet database_national_character_set]
```

```
    [-recoveryAreaDestination fast_recovery_area_directory]
        -recoveryAreaSize fast_recovery_area_size]
    [-datafileDestination data_files_directory]
    [-useOMF {true | false}]
    [-datafileJarLocation database_backup_files_directory]
    [-memoryPercentage
percentage_of_total_memory_to_assign_to_oracle_database]
    [-totalMemory total_memory_to_assign_to_oracle_database]
    [-dbOptions database_options]
    [-variablesFile variables_file]
    [-redoLogFileSize redo_log_file_size]
    [-initParams initialization_parameters_list]
        [-initParamsEscapeChar
escape_character_for_initialization_parameters]
    [-storageType {FS | ASM}
        [-asmsnmpPassword ASMSNMP_password]
        -datafileDestination data_files_directory]
    [-enableArchive {true | false}
         -archiveLogMode {AUTO | MANUAL}
         -archiveLogDest archive_logs_directory
    [-memoryMgmtType {AUTO | AUTO_SGA | CUSTOM_SGA}]
    [-useWalletForDBCredentials {true | false}
         -dbCredentialsWalletPassword wallet_account_password
         -dbCredentialsWalletLocation wallet_files_directory]
```

Table 19-9 createTemplateFromTemplate Parameters

Parameter	Required/ Optional	Description
-sourceTemplateName *existing_template_name*	Required	Name of an existing database template in the default location or the complete path to a database template that is not in the default location.
-templateName *new_template_name*	Required	Name for a new database template.
-variables *variables_list*	Optional	A comma-separated list of `name=value` pairs for the variables in the database template.
-characterSet *database_character_set*	Optional	Character set of the database.
-nationalCharacterSet *database_national_character_set*	Optional	National character set of the database.
-recoveryAreaDestination *fast_recovery_area_directory*	Optional	Directory path for the Fast Recovery Area, which is a backup and recovery area.
-datafileDestination *data_files_directory*	Optional	Directory path for the data files.
-useOMF {true | false}	Optional	Specify `true` to use Oracle-Managed Files (OMF), else specify `false`.

Table 19-9 (Cont.) createTemplateFromTemplate Parameters

Parameter	Required/ Optional	Description
`-datafileJarLocation` *database_backup_files_directory*	Optional	Location of the database offline backup (for clone database creation only). The data files for the seed database are stored in compressed RMAN backup format in a file with a .dfb extension.
`-memoryPercentage` *percentage_of_total_memory_to_assign_to_oracle_database* or `-totalMemory` *total_memory_to_assign_to_oracle_database*	Optional	Specify either `-memoryPercentage` or `-totalMemory`. • `-memoryPercentage` The percentage of physical memory that can be used by the database. • `-totalMemory` The amount of physical memory in megabytes that can be used by the database.
`-dbOptions` *database_options*	Optional	Specify database options as comma separated list of `name:value` pairs. Example: `JSERVER:true,DV:false`
`-variablesFile` *variables_file*	Optional	File name with complete directory path to the file that contains the variables and their values in the database template.
`-redoLogFileSize` *redo_log_file_size*	Optional	Size of each online redo log file in megabytes.
`-initParams` *initialization_parameters_list*	Optional	A comma-separated list of `name=value` pairs of the database initialization parameters and their values.
`-storageType` `{FS | ASM}`	Optional	Specify `FS` for file system and `ASM` for Oracle Automatic Storage Management (Oracle ASM) system. When `FS` is specified, your database files are managed by the file system of your operating system. You specify the directory path where the database files are to be stored using the `-datafileDestination` parameter. When `ASM` is specified, your database files are placed in the Oracle ASM disk groups. Oracle Database automatically manages database file placement and naming. You also specify the `ASMSNMP` password for ASM monitoring using the `-asmsnmpPassword` parameter.
`-enableArchive` `{true | false}`	Optional	Specify `true` to enable log file archive. Default is `false`. When `true` is specified, the following additional parameters can be provided: • `-archiveLogMode {AUTO | MANUAL}`: Specify either the automatic archive mode (`AUTO`) or the manual archive mode (`MANUAL`). Default is automatic archive mode (`AUTO`). • `-archiveLogDest`: Directory path for storing the archive log files.

Table 19-9 (Cont.) createTemplateFromTemplate Parameters

Parameter	Required/ Optional	Description
-memoryMgmtType {AUTO | AUTO_SGA | CUSTOM_SGA}	Optional	Specify one of the following memory management types: • AUTO: Automatic memory management for SGA and PGA. • AUTO_SGA: Automatic shared memory management for SGA. • CUSTOM_SGA: Manual shared memory management for SGA. **Note:** If the total physical memory of a database instance is greater than 4 GB, then you cannot specify the Automatic Memory Management option AUTO during the database installation and creation. Oracle recommends that you specify the Automatic Shared Memory Management option AUTO_SGA in such environments.
-useWalletForDBCredentials {true | false}	Optional	Specify true to use Oracle Wallet for database credentials, else specify false. Default is false. When true is specified, the following additional parameters can be provided: • -dbCredentialsWalletPassword: Password for the Oracle Wallet account. • -dbCredentialsWalletLocation: Directory location for the Oracle Wallet files. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • oracle.dbsecurity.walletPassword • oracle.dbsecurity.userDNPassword

deleteDatabase

The deleteDatabase command deletes a database.

Syntax and Parameters

Use the dbca -deleteDatabase command with the following syntax:

```
dbca -deleteDatabase
   -sourceDB database_name_or_sid
   [-sysDBAUserName SYSDBA_user_name]
   [-sysDBAPassword SYSDBA_user_password]
   [-forceArchiveLogDeletion]
   [-deRegisterEMCloudControl
     [-omsHost Oracle_Management_Server_host_name
      -omsPort Oracle_Management_Server_port_number
      -emUser EM_administrator_user_name
      -emPassword EM_administrator_password]]
   [-unregisterWithDirService {true | false}
      -dirServiceUserName directory_service_user_name
```

```
   [-dirServicePassword directory_service_user_password
   [-walletPassword wallet_password]]
[-sid database_system_identifier]
[-useWalletForDBCredentials {true | false}
    -dbCredentialsWalletPassword wallet_account_password
    -dbCredentialsWalletLocation wallet_files_directory]
```

Table 19-10 deleteDatabase Parameters

Parameter	Required/ Optional	Description
`-sourceDB` *database_name_or_sid*	Required	Database unique name for an Oracle RAC database or database system identifier (SID) for a single instance database.
`-sysDBAUserName` *SYSDBA_user_name*	Optional	User name of the user having the `SYSDBA` privileges.
`-sysDBAPassword` *SYSDBA_password*	Optional	Password of the user having the `SYSDBA` privileges.
`-forceArchiveLogDeletion`	Optional	Specify this parameter to delete the database archive logs.
`-deRegisterEMCloudControl`	Optional	Specify this parameter along with the following parameters to unregister the database with Enterprise Manager Cloud Control: • `-omsHost`: Oracle Management Server host name. • `-omsPort`: Oracle Management Server port number. • `-emUser`: User name for Enterprise Manager administrator. • `-emPassword`: Password for Enterprise Manager administrator.
`-unregisterWithDirService {true | false}`	Optional	Specify this parameter along with the following parameters to unregister the database with the directory service: • `-dirServiceUserName`: User name for the directory service. • `-dirServicePassword`: Password for the directory service user. • `-walletPassword`: Password for the database wallet.
`-sid` *database_system_identifier*	Optional	Database system identifier (SID).

Table 19-10 (Cont.) deleteDatabase Parameters

Parameter	Required/ Optional	Description
-useWalletForDBCredentials {true | false}	Optional	Specify `true` to use Oracle Wallet for database credentials, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters can be provided: • `-dbCredentialsWalletLocation`: Directory location for the Oracle Wallet files. • `-dbCredentialsWalletPassword`: Password for the Oracle Wallet account. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • `oracle.dbsecurity.walletPassword` • `oracle.dbsecurity.userDNPassword`

deleteInstance

The `deleteInstance` command deletes a database instance from an administror-managed Oracle RAC database.

Syntax and Parameters

Use the `dbca -deleteInstance` command with the following syntax:

```
dbca -deleteInstance
   -gdbName global_database_name
   -instanceName database_instance_name
   [-nodeName database_instance_node_name]
   [-updateDirService {true | false}
      -dirServiceUserName directory_service_user_name
      -dirServicePassword directory_service_user_password]
   [-sysDBAUserName SYSDBA_user_name]
   [-sysDBAPassword SYSDBA_user_password]
   [-useWalletForDBCredentials {true | false}
      -dbCredentialsWalletPassword wallet_account_password
      -dbCredentialsWalletLocation wallet_files_directory]
```

Table 19-11 deleteInstance Parameters

Parameter	Required/ Optional	Description
-gdbName *global_database_name*	Required	Global database name in the form *database_name.domain_name*.
-instanceName *database_instance_name*	Required	Database instance name.

Table 19-11 (Cont.) deleteInstance Parameters

Parameter	Required/ Optional	Description
`-nodeName` *node_name_of_database_instance*	Optional	Node name of the database instance.
`-sysDBAUserName` *SYSDBA_user_name*	Optional	User name of the database user having the `SYSDBA` privileges.
`-sysDBAPassword` *SYSDBA_user_password*	Optional	Password of the database user having the `SYSDBA` privileges.
`-updateDirService` `{true | false}`	Optional	Specify `true` to unregister the database with the directory service, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters are required: • `-dirServiceUserName`: User name for the directory service. • `–dirServicePassword`: Password for the directory service user.
`-useWalletForDBCredentials` `{true | false}`	Optional	Specify `true` to use Oracle Wallet for database credentials, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters can be provided: • `-dbCredentialsWalletPassword`: Password for the Oracle Wallet account. • `-dbCredentialsWalletLocation`: Directory location for the Oracle Wallet files. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • `oracle.dbsecurity.walletPassword` • `oracle.dbsecurity.userDNPassword`

deletePluggableDatabase

The `deletePluggableDatabase` command deletes a PDB.

Syntax and Parameters

Use the `dbca -deletePluggableDatabase` command with the following syntax:

```
dbca -deletePluggableDatabase
   -sourceDB cdb_sid
   -pdbName pdb_name
   [-unregisterWithDirService {true | false} ]
      -dirServiceUserName directory_service_user_name
      [-dirServicePassword directory_service_user_password ]
      [-walletPassword wallet_password]
   [-useWalletForDBCredentials {true | false} ]
```

```
-dbCredentialsWalletPassword wallet_account_password
[-dbCredentialsWalletLocation wallet_files_directory]
```

Table 19-12 deletePluggableDatabase Parameters

Parameter	Required/ Optional	Description
-sourceDB *cdb_sid*	Required	The database system identifier (SID) of the CDB.
-pdbName *pdb_name*	Required	Name of the PDB to delete.
-useWalletForDBCredentials {true | false}	Optional	Specify `true` to use Oracle Wallet for database credentials, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters must be provided: • `-dbCredentialsWalletPassword` (Optional): Password for the Oracle Wallet account. If you omit this parameter, DBCA prompts for the password. • `-dbCredentialsWalletLocation`: Directory location for the Oracle Wallet files. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • `oracle.dbsecurity.walletPassword` • `oracle.dbsecurity.userDNPassword`
-unregisterWithDirService {true | false}	Optional	Specify this parameter along with the following parameters to unregister the database with the directory service: • `-dirServiceUserName`: User name for the directory service. • `-dirServicePassword`: Password for the directory service user. • `-walletPassword`: Password for the database wallet.

deleteTemplate

The `deleteTemplate` command deletes a database template.

Syntax and Parameters

Use the `dbca -deleteTemplate` command with the following syntax:

```
dbca -deleteTemplate
   -templateName name_of_an_existing_database_template
   [-useWalletForDBCredentials {true | false}
      -dbCredentialsWalletPassword wallet_account_password
      -dbCredentialsWalletLocation wallet_files_directory]
   [-templateFromCloud
      -opcLibPath OPC_library_path
      -opcConfigFile OPC_config_file_name
      [-rmanDecryptionPassword rman_decryption_password]]
```

Table 19-13 deleteTemplate Parameters

Parameter	Required/ Optional	Description
`-templateName` *`name_of_an_existing_database_template`*	Required	Name of an existing database template to delete.
`-templateFromCloud`	Optional	Indicates that the template is a Cloud template. • `opcLibPath`: Provide the directory containing the `odbsrmt.py` script for the delete template operation or provide the directory containing `libopc.so`. • `opcConfigFile`: Name, with complete location, of the Oracle Database Cloud Backup Module for OCI configuration file. • `rmanDecryptionPassword`: Password that must be used to decrypt the RMAN template file stored in Oracle Cloud Infrastructure. This is the same password that was used when creating an RMAN backup of the template.
`-useWalletForDBCredentials` `{true | false}`	Optional	Specify `true` to use Oracle Wallet for database credentials, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters can be provided: • `-dbCredentialsWalletPassword`: Password for the Oracle Wallet account. • `-dbCredentialsWalletLocation`: Directory location for the Oracle Wallet files. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • `oracle.dbsecurity.walletPassword` • `oracle.dbsecurity.userDNPassword`

executePrereqs

The `executePrereqs` command executes the prerequisites checks and reports the results. This command can be used to check the environment before running `dbca` to create a database.

Syntax and Parameters

Use the `dbca -executePrereqs` command with the following syntax:

```
dbca -executePrereqs
   -databaseConfigType {SINGLE | RAC | RACONENODE}
        [-RACOneNodeServiceName RAC_node_service_name]
   [-nodelist database_nodes_list]
```

Table 19-14 executePrereqs Parameters

Parameter	Required/ Optional	Description
`-databaseConfigType {SINGLE | RAC | RACONENODE}`	Required	Specify one of the following database configuration types: • `SINGLE`: Single individual database. • `RAC`: Oracle RAC database. • `RACONENODE`: Oracle RAC One Node database. For Oracle RAC One Node database, you can specify the service name using the `-RACOneNodeServiceName` parameter.
`-nodelist` *database_nodes_list*	Optional	List of database nodes separated by comma.

generateScripts

The `generateScripts` command generates scripts, which can be used to create a database.

Syntax and Parameters

Use the `dbca -generateScripts` command with the following syntax:

```
dbca -generateScripts
    -templateName database_template_name
    -gdbName global_database_name
    [-sid database_system_identifier]
    [-scriptDest sql_scripts_directory]
    [-createAsContainerDatabase {true | false}
        [-numberOfPDBs number_of_pdbs_to_create]
        [-pdbName pdb_name]
        [-pdbStorageMAXSizeInMB maximum_storage_size_of_the_pdb]
        [-pdbStorageMAXTempSizeInMB
maximum_temporary_storage_size_of_the_pdb]
        [-useLocalUndoForPDBs {true | false}]
        [-pdbAdminPassword pdb_administrator_password]
        [-pdbOptions pdb_options]
    [-sysPassword SYS_user_password]
    [-systemPassword SYSTEM_user_password]
    [-emConfiguration {CENTRAL | NONE}
        [-dbsnmpPassword DBSNMP_user_password]
        [-omsHost EM_Management_Server_host_name]
        [-omsPort EM_Management_Server_port_number]
        [-emUser EM_administrator_name]
        [-emPassword EM_administrator_password]
    [-dvConfiguration {true | false}
        -dvUserName Database_Vault_owner_user_name
        -dvUserPassword Database_Vault_owner_user_password
        [-dvAccountManagerName Database_Vault_account_manager_name
        -dvAccountManagerPassword Database_Vault_account_manager_password]]
    [-olsConfiguration {true | false}
        [-configureWithOID configure_with_OID_flag]]
```

```
    [-datafileDestination data_files_directory]
    [-redoLogFileSize maximum_redo_log_file_size_in_MB]
    [-recoveryAreaDestination fast_recovery_area_directory
        [-recoveryAreaSize fast_recovery_area_size]]
    [-datafileJarLocation data_files_backup_directory]
    [-responseFile response_file_directory]
    [-storageType {FS | ASM}
        [-asmsnmpPassword ASMSNMP_password]
        -datafileDestination data_files_directory]
    [-runCVUChecks {true | false}]
    [-nodelist database_nodes_list]
    [-enableArchive {true | false}
        [-archiveLogMode {AUTO | MANUAL}]
        [-archiveLogDest archive_log_files_directory]]
    [-memoryMgmtType {AUTO | AUTO_SGA | CUSTOM_SGA}]
    [-createListener
new_database_listener_to_register_the_database_with]
    [-useOMF {true | false}]
    [-dbOptions database_options]
    [-customScripts custom_sql_scripts_to_run_after_database_creation]
    [-policyManaged | -adminManaged]
    [-policyManaged
        -serverPoolName server_pool_names
        [-pqPoolName pq_pool_name]
        [-createServerPool new_server_pool_name]
            [-pqPoolName new_pq_pool_name]
            [-force]
            [-pqCardinality pq_cardinality_of_the_new_server_pool]
            [-cardinality cardinality_of_the_new_server_pool]]
    [-adminManaged]
    [-databaseConfigType {SINGLE | RAC | RACONENODE}
            [-RACOneNodeServiceName
service_name_for_RAC_one_node_database]]
    [-characterSet database_character_set]
    [-nationalCharacterSet database_national_character_set]
    [-registerWithDirService {true | false}
        [-dirServiceUserName directory_service_user_name]
        [-dirServicePassword directory_service_user_password]
        [-databaseCN database_common_name]
        [-dirServiceCertificatePath certificate_file_path]
        [-dirServiceUser directory_service_user_name]
        [-ldapDirectoryAccessType ldap_directory_access_type]
        [-useSYSAuthForLDAPAccess use_sys_user_for_ldap_access_flag]
        [-walletPassword wallet_password]]
    [-listeners list_of_listeners_to_register_the_database_with]
    [-variablesFile variables_file]
    [-variables variables_list]
    [-initParams initialization_parameters_list
        [-initParamsEscapeChar
initialization_parameters_escape_character]]
    [-sampleSchema {true | false}]
    [-memoryPercentage
percentage_of_total_memory_to_assign_to_the_database]
    [-totalMemory total_memory_to_assign_to_the_database_in_MB]
    [-databaseType {MULTIPURPOSE | DATA_WAREHOUSING | OLTP}]
```

```
[-useWalletForDBCredentials {true | false}
    -dbCredentialsWalletPassword wallet_account_password
    -dbCredentialsWalletLocation wallet_files_directory]
[-configureTDE <true | false> ]
      [-primaryDBTdeWallet value]
      [-sourceTdeWalletPassword value]
      [-tdeWalletRoot tde_wallet_root_init_parameter]
      [-pdbTDEPassword pdb_tde_wallet_password]
      [-tdeWalletModeForPDB pdb_keystore_type]
      [-tdeAlgorithm TDE_algorithm]
      [-tdeWalletLoginType type_of_wallet_login]
      [-sourcePdbTDEPassword source_pdb_TDE_wallet_password]
      [-tdeWalletPassword TDE_wallet_password]
```

Table 19-15 generateScripts Parameters

Parameter	Required/ Optional	Description
`-templateName` *database_template_name*	Required	Name of an existing database template in the default location or the complete path of a template that is not in the default location.
`-gdbName` *global_database_name*	Required	Global database name in the form *database_name.domain_name*.
`-sid` *database_system_identifier*	Optional	Database system identifier (SID). The SID uniquely identifies the instance that runs the database. If it is not specified, then it defaults to the database name.
`-scriptDest` *scripts_directory*	Optional	Complete directory path to store the scripts.
`-createAsContainerDatabase` `{true | false}`	Optional	Specify `true` to create a CDB. Specifying `false` is not supported starting with Oracle Database Release 21c. When `true` is specified, the following optional parameters can be provided: • `-numberOfPDBs`: Number of PDBs to create. Default is 0 (zero). • `-pdbName`: Name of each PDB. A number is appended to each PDB name if `-numberOfPDBs` value is greater than 1. This parameter must be specified if `-numberOfPDBs` value is greater than 0 (zero). • `-pdbStorageMAXSizeInMB`: Maximum storage size for a PDB in megabytes. • `-pdbStorageMAXTempSizeInMB`: Maximum temporary storage size for a PDB in megabytes. • `-useLocalUndoForPDBs {true | false}`: Flag indicating whether local undo should be used for the PDBs. • `-pdbAdminPassword`: PDB administrator password. • `-pdbOptions`: PDB options in the form of comma separated list. Each option must be specified in the `name:value` format. Example: `JSERVER:true,DV:false`

Table 19-15 (Cont.) generateScripts Parameters

Parameter	Required/ Optional	Description
`-sysPassword` *SYS_user_password*	Optional	`SYS` user password for the new database.
`-systemPassword` *SYSTEM_user_password*	Optional	`SYSTEM` user password for the new database.
`-emConfiguration` `{CENTRAL | NONE}`	Optional	Enterprise Manager configuration settings. When `CENTRAL` is specified, specify the following additional parameters: • `-dbsnmpPassword`: DBSNMP user password. • `-omsHost`: Oracle Management Server host name. • `-omsPort`: Oracle Management Server port number. • `-emUser`: User name for Enterprise Manager administrator. • `-emPassword`: Password for Enterprise Manager administrator.
`-dvConfiguration` `{true | false}`	Optional	Specify `true` to enable and configure Database Vault, else specify `false`. Default is `false`. When `true` is specified, the following additional Database Vault parameters are required: • `-dvUserName`: Database Vault owner name. • `-dvUserPassword`: Database Vault owner password. • `-dvAccountManagerName`: Database Vault account manager name. • `-dvAccountManagerPassword`: Database Vault account manager password.
`-olsConfiguration` `{true | false}`	Optional	Specify `true` to enable and configure Oracle Label Security (OLS), else specify `false`. Default is `false`. When `true` is specified, you can additionally specify the `-configureWithOID` parameter to configure Oracle Label Security (OLS) with Oracle Internet Directory (OID). This parameter is optional.
`-datafileDestination` *data_files_directory*	Optional	Complete path to the location of the database's data files.
`-redoLogFileSize` *maximum_size_of_online_redo_log*	Optional	Size of each online redo log file in megabytes.
`-recoveryAreaDestination` *fast_recovery_area_directory*	Optional	Directory for the Fast Recovery Area, which is a backup and recovery area. Specify `NONE` to disable the Fast Recovery Area. Additionally, you can specify the Fast Recovery Area size in megabytes using the parameter `-recoveryAreaSize`. This parameter is optional.
`-datafileJarLocation` *data_files_backup_directory*	Optional	Directory of the database backup data files in a compressed RMAN backup format (files with .dfb extensions).

Table 19-15 (Cont.) generateScripts Parameters

Parameter	Required/ Optional	Description
`-responseFile` *response_file_directory*	Optional	Directory path of the response file.
`-storageType {FS | ASM}`	Optional	Specify the storage type of either `FS` or `ASM`. • `FS`: File system storage type. When `FS` is specified, your database files are managed by the file system of your operating system. You can specify the directory path where the database files are to be stored using a database template or the `-datafileDestination` parameter. Oracle Database can create and manage the actual files. • `ASM`: Oracle Automatic Storage Management (Oracle ASM) storage type. When `ASM` is specified, your database files are placed in Oracle ASM disk groups. Oracle Database automatically manages database file placement and naming. When `ASM` is specified, you can also specify the ASMSNMP password using the `-asmsnmpPassword` parameter. This parameter is optional.
`-runCVUChecks {true | false}`	Optional	Specify `true` to run Cluster Verification Utility checks periodically for Oracle RAC databases, else specify `false`. Default is `false`.
`-nodelist` *database_nodes_list*	Optional	List of database nodes separated by comma.
`-enableArchive {true | false}`	Optional	Specify `true` to enable log file archive, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters can be provided: • `-archiveLogMode {AUTO | MANUAL}`: Specify either the automatic archive mode or the manual archive mode. Default is automatic archive mode. • `-archiveLogDest`: Directory for storing the archive log files.
`-memoryMgmtType {AUTO | AUTO_SGA | CUSTOM_SGA}`	Optional	Specify one of the following memory management types: • `AUTO`: Automatic memory management for SGA and PGA. • `AUTO_SGA`: Automatic shared memory management for SGA. • `CUSTOM_SGA`: Manual shared memory management for SGA. **Note:** If the total physical memory of a database instance is greater than 4 GB, then you cannot specify the Automatic Memory Management option `AUTO` during the database installation and creation. Oracle recommends that you specify the Automatic Shared Memory Management option `AUTO_SGA` in such environments.

Table 19-15 (Cont.) generateScripts Parameters

Parameter	Required/ Optional	Description
`-createListener` *new_database_listener*	Optional	Database listener to register the database with in the form *listener_name:port*.
`-useOMF` `{true | false}`	Optional	Specify `true` to use Oracle-Managed Files (OMF), else specify `false`.
`-dbOptions` *database_options*	Optional	Specify database options as a comma separated list of `name:value` pairs. Example: `JSERVER:true,DV:false`
`-customScripts` *custom_sql_scripts_list*	Optional	Specify a comma separated list of SQL scripts that need to be run after the database creation. The scripts are run in the order they are listed.
`-policyManaged`	Optional	Policy-managed database. You can specify the following additional parameters: • `-serverPoolName`: Specify the single server pool name when creating a new server pool or specify a comma separated list of existing server pools. • `-pqPoolName`: Specify the PQ pool name. • `-createServerPool`: Specify this parameter for creating a new server pool. – `-pqPoolName`: Specify the PQ pool name. – `-force`: Specify this parameter to create the server pool by force when adequate free servers are not available. – `-pqCardinality`: Specify the PQ cardinality of the new server pool. – `-cardinality`: Specify the cardinality of the new server pool.
`-adminManaged`	Optional	Administrator-managed database.
`-databaseConfigType` `{SINGLE | RAC | RACONENODE}`	Optional	Specify one of the following database configuration types: • `SINGLE`: Single individual database. • `RAC`: Oracle RAC database. • `RACONENODE`: Oracle RAC One Node database. For Oracle RAC One Node database, you can specify the service name using the `-RACOneNodeServiceName` parameter.
`-characterSet` *database_character_set*	Optional	Character set of the database.
`-nationalCharacterSet` *database_national_character_set*	Optional	National character set of the database.

Table 19-15 (Cont.) generateScripts Parameters

Parameter	Required/ Optional	Description
`-registerWithDirService {true | false}`	Optional	Specify `true` to register with a Lightweight Directory Access Protocol (LDAP) service, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters are required: • `-dirServiceUserName`: User name for the LDAP service. • `-dirServicePassword`: Password for the LDAP service. • `-databaseCN`: Database common name. • `-dirServiceCertificatePath`: Directory service certificate file path. • `-dirServiceUser`: Directory service user name. • `-ldapDirectoryAccessType {PASSWORD | SSL}`: LDAP directory access type. • `-useSYSAuthForLDAPAccess {true | false}`: Specify whether to use SYS user authentication for LDAP acces. • `-walletPassword`: Password for the database wallet.
`-listeners` *`listeners_list`*	Optional	A comma-separated list of listeners for the database.
`-variablesFile` *`variables_file`*	Optional	Directory path to the file that contains the variables and their values for the database template.
`-variables` *`variables_list`*	Optional	A comma-separated list of `name=value` pairs of variables for the database template.
`-initParams` *`initialization_parameters_list`*	Optional	A comma-separated list of `name=value` pairs of initialization parameter values of the database. You can additionally provide the `-initParamsEscapeChar` parameter for using a specific escape character between multiple values of an initialization parameter. If an escape character is not specified, backslash (/) is used as the default escape character.
`-sampleSchema {true | false}`	Optional	Specify `true` to include the `HR` sample schema (`EXAMPLE` tablespace) in your database, else specify `false`. Default is `false`. Oracle guides and educational materials contain examples based on the sample schemas. Oracle strongly recommends that you do not install the sample schemas in a production database.
`-memoryPercentage` *`percentage_of_total_memory_assigned_to_the_database`*	Optional	The percentage of physical memory that can be used by the database.
`-totalMemory` *`total_memory_assigned_to_the_database_in_MB`*	Optional	Total amount of physical memory, in megabytes, that can be used by the database.

Table 19-15 (Cont.) generateScripts Parameters

Parameter	Required/ Optional	Description
-databaseType {MULTIPURPOSE | DATA_WAREHOUSING | OLTP}	Optional	Specify MULTIPURPOSE if the database is for both OLTP and data warehouse purposes. Specify DATA_WAREHOUSING if the primary purpose of the database is a data warehouse. Specify OLTP if the primary purpose of the database is online transaction processing.
-useWalletForDBCredentials {true | false}	Optional	Specify true to use Oracle Wallet for database credentials, else specify false. Default is false. When true is specified, the following additional parameters can be provided: • -dbCredentialsWalletLocation: Directory location for the Oracle Wallet files. • -dbCredentialsWalletPassword: Password for the Oracle Wallet account. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • oracle.dbsecurity.walletPassword • oracle.dbsecurity.userDNPassword
configureTDE	Optional	Specify true to configure TDE during the database creation. Only software wallets are supported. You can create a wallet for the entire CDB or for a PDB. • primaryDBTdeWallet: This option is not applicable when creating a database. • sourceTdeWalletPassword: If the template that is being used is from a database that uses encryption, or if you are duplicating a database, specify the password of the wallet in the source database. • tdeWalletModeForPDB: Specify UNITED to create a wallet for the entire CDB. Use ISOLATED to create a wallet for a PDB. • tdeAlgorithm: Algorithm used to encrypt data. Can be one of the following: 3DES168, AES128, AES192, AES256. • tdeWalletLoginType: Type of software wallet. PASSWORD or AUTO_LOGIN or LOCAL_AUTO_LOGIN. • tdeWalletLocation: Location in which the TDE wallet is stored. • tdeWalletPassword: The password used to open the wallet. This parameter is mandatory. Note: Isolated wallets are supported only in Oracle Cloud or Exadata environments.

relocatePDB

The `relocatePDB` command relocates a PDB from a *remote* CDB to a *local* CDB.

Prerequisites

The following are the prerequisites for running the `relocatePDB` command:

- The database user in the local PDB must have the `CREATE PLUGGABLE DATABASE` privilege in the local CDB root container.
- The remote CDB must be in the *local undo* mode.
- The remote and local PDBs must be in the *archivelog* mode.
- The database user in the remote PDB that the database link connects to must have the `CREATE PLUGGABLE DATABASE`, `SESSION`, and `SYSOPER` privileges.
- The local and remote PDBs must have the same options installed, or the remote PDB must have a subset of the options installed on the local PDB.

Syntax and Parameters

Use the `dbca -relocatePDB` command with the following syntax:

```
dbca -relocatePDB
   -pdbName name_of_the_local_pdb_to_create
   -sourceDB database_name_of_the_local_pdb
   -remotePDBName name_of_the_remote_pdb_to_relocate
   -remoteDBConnString db_connection_string_of_the_remote_pdb
   -sysDBAUserName name_of_the_sysdba_user
   -sysDBAPassword password_of_the_sysdba_user
   -dbLinkUsername name_of_the_dblink_user_of_the_remote_pdb
   -dbLinkUserPassword password_of_the_dblink_user_of_the_remote_pdb
```

Table 19-16 relocatePDB Parameters

Parameter	Required/ Optional	Description
`-pdbName` *name_of_the_local_pdb_to_create*	Required	Name of the local PDB to create after relocating the remote PDB.
`-sourceDB` *database_name_of_the_local_pdb*	Required	Database name of the local PDB.
`-remotePDBName` *name_of_the_remote_pdb_to_relocate*	Required	Name of the remote PDB to relocate.
`-remoteDBConnString` *db_connection_string_of_the_remote_pdb*	Required	Database connection string of the remote PDB.
`-sysDBAUserName` *name_of_the_sysdba_user*	Required	Name of the SYSDBA user.

Table 19-16 (Cont.) relocatePDB Parameters

Parameter	Required/ Optional	Description
`-sysDBAPassword` *password_of_the_sysdba_user*	Required	Password of the SYSDBA user.
`-dbLinkUsername` *name_of_the_dblink_user_of_the_remote_pdb*	Required	Name of the database link user of the remote PDB.
`-dbLinkUserPassword` *password_of_the_dblink_user_of_the_remote_pdb*	Required	Password of the database link user of the remote PDB.

unplugDatabase

The `unplugDatabase` command unplugs a pluggable database (PDB) from a multitenant container database (CDB).

Syntax and Parameters

Use the `dbca -unplugDatabase` command with the following syntax:

```
dbca -unplugDatabase
   -sourceDB cdb_sid
   -pdbName pdb_name
   [-unregisterWithDirService {true | false}
      -dirServiceUserName directory_service_user_name
      -dirServicePassword directory_service_user_password
      -walletPassword wallet_password]
   [-archiveType {TAR | RMAN | NONE}
      [-rmanParallelism parallelism_integer_value]
      [-pdbArchiveFile pdb_archive_file_directory]
      [-PDBBackUpfile pdb_backup_file_directory]
      [-PDBMetadataFile pdb_metadata_file_directory]
      [-rmanParallelism parallelism_integer_value]]
   [-useWalletForDBCredentials {true | false}
      -dbCredentialsWalletPassword wallet_account_password
      -dbCredentialsWalletLocation wallet_files_directory]
```

Table 19-17 unplugDatabase Parameters

Parameter	Required/ Optional	Description
`-sourceDB` *cdb_sid*	Required	The database system identifier (SID) of the CDB.
`-pdbName` *pdb_name*	Required	Name of the PDB.

Table 19-17 (Cont.) unplugDatabase Parameters

Parameter	Required/ Optional	Description
`-archiveType {TAR | RMAN | NONE}`	Optional	Specify `TAR` to store the unplugged PDB files in a tar file. Specify `RMAN` to store the unplugged PDB files in an RMAN backup. Specify `NONE` to store the unplugged PDB files without using a tar file or an RMAN backup. Specify any of the following parameters: • `-pdbArchiveFile`: Specify absolute file path and name for the PDB Archive file. • `-pdbBackUpfile`: Specify absolute file path and name for the PDB backup file when archive type is `RMAN`. specify comma separated file paths, if there are multiple backups to be taken when creating the PDB. • `-pdbMetadataFile`: Specify absolute file path and name for the PDB metadata file when archive type is `RMAN` or `NONE`. • `-rmanParallelism`: Specify the RMAN parallelism integer value.
`-unregisterWithDirService {true | false}`	Optional	Specify `true` to unregister the PDB from the LDAP service, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters are required: • `-dirServiceUserName`: User name for the LDAP service. • `-dirServicePassword`: Password for the LDAP service user. • `-walletPassword`: Password for the database wallet.
`-useWalletForDBCredentials {true | false}`	Optional	Specify `true` to use Oracle Wallet for database credentials, else specify `false`. Default is `false`. When `true` is specified, the following additional parameters can be provided: • `-dbCredentialsWalletPassword`: Password for the Oracle Wallet account. • `-dbCredentialsWalletLocation`: Directory location for the Oracle Wallet files. **Note:** If you are using Oracle Unified Directory (OUD), then the OUD passwords should be stored in the wallet using the following keys: • `oracle.dbsecurity.walletPassword` • `oracle.dbsecurity.userDNPassword`

20
DBCA Exit Codes

The outcome of running DBCA commands in silent mode is reported as an exit code.

The following table shows the exit codes that DBCA returns to the operating system.

Table 20-1 Exit Codes for Database Configuration Assistant

Exit Code	Description
0	Command execution successful
6	Command execution successful but with warnings
-1	Command execution failed
-2	Invalid input from user
-4	Command canceled by user

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. Starting in Oracle Database 21c, every Oracle database is a 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 PDBs and application containers.

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 21c, every database must be a 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 separate database.

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.

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.

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

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W