

Oracle Database 12c: Managing Multitenant Architecture

Student Guide

D79128GC10

Edition 1.0

July 2013

D82583

ORACLE®

Authors

Dominique Jeunot
Jean-François Verrier

Technical Contributors and Reviewers

Bill Millar
Branislav Valny
Gerlinde Frenzen
Joel Goodman
Harald Van Breederode
Maria Billings
Randy Urbano

Editors

Raj Kumar
Anwesha Ray

Graphic Designer

Rajiv Chandrabhanu

Publishers

Jobi Varghese
Srividya Rameshkumar

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Disclaimer

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

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

Restricted Rights Notice

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

U.S. GOVERNMENT RIGHTS

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

Trademark Notice

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

Contents

1 Introduction

- Objectives 1-2
- Course Structure 1-3
- Suggested Schedule 1-4
- Enterprise Manager Cloud Control and Other Tools 1-5
- Target Discovery: Multitenant Container Database 1-6
- Enterprise Manager Cloud Control 1-7
- Enterprise Manager Database Express 1-8
- Configuring Enterprise Manager Database Express 1-9
- Database Configuration Assistant 1-10
- SQL*Plus 1-11
- Oracle SQL Developer: Connections 1-12
- Oracle SQL Developer: DBA Tasks 1-13
- Quiz 1-14
- Summary 1-15
- Practice 1 Overview: Using Enterprise Manager Cloud Control 1-16

2 Basics of Multitenant Container Database and Pluggable Databases

- Course Structure 2-2
- Objectives 2-3
- Challenges 2-4
- Non-CDB Architecture 2-5
- New Multitenant Architecture: Benefits 2-6
- Other Benefits of Multitenant Architecture 2-8
- Configurations 2-10
- Oracle Container Database 2-11
- A Pristine Installation 2-12
- User Data Is Added 2-13
- Separating SYSTEM and User Data 2-14
- SYSTEM Objects in the USER Container 2-15
- Naming the Containers 2-16
- Provisioning a Pluggable Database 2-17
- Interacting Within Multitenant Container Database 2-18
- Multitenant Container Database Architecture 2-19
- Containers 2-20

Questions: root Versus PDBs 2-21
Questions: PDBs Versus root 2-22
Terminology 2-23
Common and Local Users 2-24
Common and Local Privileges and Roles 2-25
Shared and Non-shared Objects 2-26
Data Dictionary Views 2-27
Impacts 2-28
Quiz 2-30
Summary 2-33
Multitenant Architecture General Architecture Poster 2-34
Practice 2 Overview: Exploring a Multitenant Container Database 2-35

3 Creating a Multitenant Container Database and Pluggable Databases

Course Structure 3-2
Objectives 3-3
Goals 3-4
Tools 3-5
Steps to Create a Multitenant Container Database 3-6
Create a Multitenant Container Database: Using SQL*Plus 3-7
Create a Multitenant Container Database: Using DBCA 3-9
New Clause: SEED FILE_NAME_CONVERT 3-10
New Clause : ENABLE PLUGGABLE DATABASE 3-11
After CDB Creation: What's New in CDB 3-12
Data Dictionary Views: DBA_xxx 3-13
Data Dictionary Views: CDB_xxx 3-14
Data Dictionary Views: Examples 3-15
Data Dictionary Views: V\$xxx Views 3-16
EM Cloud Control: Summary 3-17
EM Cloud Control: Storage Example 3-18
After CDB Creation: To do List 3-20
Automatic Diagnostic Repository 3-21
Automatic Diagnostic Repository: alert.log File 3-22
Quiz 3-23
Practice 3 Overview: Creating a CDB and PDBs 3-25
Provisioning New Pluggable Databases 3-26
Tools 3-27
Method 1: Create New PDB from PDB\$SEED 3-28
Steps: With FILE_NAME_CONVERT 3-29
Steps: Without FILE_NAME_CONVERT 3-30
Method 1: Using SQL Developer 3-31

Synchronization	3-33
Method 2: Plug a Non-CDB into CDB	3-34
Plug a Non-CDB into CDB Using DBMS_PDB	3-35
Plug a Non-CDB into CDB Using Replication	3-36
Method 3: Clone PDBs	3-37
Clone PDBs: Using SQL Developer	3-38
Method 4: Plug Unplugged PDB into CDB	3-40
Method 4: Flow	3-41
Unplugged PDB: Using SQL Developer	3-43
Plug Unplugged PDB: Using SQL Developer	3-44
Unplug and Plug PDB with Encrypted Data	3-45
Plug Sample Schemas PDB: Using DBCA	3-46
Dropping a PDB	3-47
Migrating pre-12.1 or 12.1 non-CDB to CDB	3-48
Quiz	3-49
Summary	3-51
Practice 3 Overview: Creating a CDB and PDBs	3-52

4 Managing a Multitenant Container Database and Pluggable Databases

Course Structure	4-2
Objectives	4-3
Connection	4-4
Connection with SQL*Developer	4-6
Creating Services	4-7
Switching Connection	4-8
Starting Up a CDB Instance	4-9
Mounting a CDB	4-10
Opening a CDB	4-11
Opening a PDB	4-12
Closing a PDB	4-13
Shutting Down a CDB Instance	4-14
Database Event Triggers: Automatic PDB Opening	4-15
Changing PDB Mode	4-16
Changing PDB Mode: With SQL Developer	4-17
Modifying a PDB Settings	4-18
Instance Parameter Change Impact	4-19
Instance Parameter Change Impact: Example	4-20
Using ALTER SYSTEM Statement on PDB	4-21
Quiz	4-22
Summary	4-24
Practice 4 Overview: Managing a CDB and PDBs	4-25

5 Managing Tablespaces in CDB and PDBs

- Course Structure 5-2
- Objectives 5-3
- Tablespaces in PDBs 5-4
- Creating Permanent Tablespaces in a CDB 5-5
- Assigning Default Tablespaces 5-6
- Creating Local Temporary Tablespaces 5-7
- Assigning Default Temporary Tablespaces 5-8
- Quiz 5-9
- Summary 5-11
- Practice 5 Overview: Managing Tablespaces and Users in CDB and PDBs 5-12

6 Managing Security in CDB and PDBs

- Course Structure 6-2
- Objectives 6-3
- Users, Roles, and Privileges 6-4
- Local Users, Roles, and Privileges 6-5
- Creating a Local User 6-6
- Common Users 6-7
- Creating a Common User 6-8
- Common and Local Schemas 6-9
- Common and Local Privileges 6-10
- Granting Common and Local Privileges 6-11
- Granting and Revoking Privileges 6-12
- Creating Common and Local Roles 6-13
- Granting Common or Local Privileges / Roles to Roles 6-15
- Granting Common and Local Roles to Users 6-16
- Granting and Revoking Roles 6-17
- Creating Shared and Nonshared Objects 6-18
- Enabling Common Users to Access Data in Specific PDBs 6-19
- Finding Information About CONTAINER_DATA Attributes 6-20
- Creating Common and Local Profiles 6-21
- Assigning Common and Local Profiles to Users 6-23
- Assigning Profiles 6-24
- Restriction on Definer's Rights 6-25
- Quiz 6-26
- Summary 6-28
- Practice 6 Overview: Managing Users, Roles, and Privileges in CDB and PDBs 6-29

7 Backup, Recovery, Flashback CDB and PDBs

- Course Structure 7-2
- Objectives 7-3
- New Syntax and Clauses in RMAN 7-5
- CDB Backup: Whole CDB Backup 7-6
- CDB Backup: User-Managed Hot CDB Backup 7-7
- CDB Backup: Partial CDB Backup 7-8
- PDB Backup: Whole PDB Backup 7-9
- PDB Backup: Partial PDB Backup 7-10
- Using RMAN Backup to Plug an Unplugged PDB 7-11
- Recovery 7-12
- Instance Failure 7-13
- NOARCHIVELOG Mode 7-14
- Media Failure: CDB or PDB Tempfile Recovery 7-15
- Media Failure: PDB Tempfile Recovery 7-16
- Media Failure: Control File Loss 7-17
- Media Failure: Redo Log File Loss 7-18
- Media Failure: root SYSTEM or UNDO datafile 7-19
- Media Failure: root SYSAUX Datafile 7-20
- Media Failure: PDB SYSTEM Datafile 7-21
- Media Failure: PDB Nonsystem Datafile 7-22
- Using Data Recovery Advisor 7-23
- Data Failures 7-24
- Data Recovery Advisor RMAN Command-Line Interface 7-25
- Media Failure: PITR 7-26
- Flashback CDB 7-28
- Duplicating Pluggable Databases 7-30
- Checking for Block Corruption 7-31
- Special Situations 7-32
- New Data Dictionary View and Column 7-33
- Quiz 7-34
- Summary 7-36
- Practice 7 Overview: Managing CDB and PDBs Backup and Recovery 7-37

8 Performance

- Course Structure 8-2
- Objectives 8-3
- Tuning a Multitenant Container Database 8-4
- Tuning Methodology 8-5
- General Tuning Session 8-7
- Sizing the CDB 8-9

Testing the Estimates	8-10
Allocating Resources in the CDB	8-11
Initialization Parameters in a CDB	8-12
Enterprise Manager Cloud Control 12c: Setting Initialization Parameters	8-13
Tuning CDB Memory	8-14
Enterprise Manager Cloud Control 12c: Memory Advisors	8-15
Memory Advisors	8-16
Limiting PGA Usage	8-17
Tuning SQL	8-18
AWR SQL Reports	8-19
ADDM Tasks	8-20
SQL Advisors	8-21
SQL Tuning Advisor	8-22
Quiz	8-23
Resource Manager and Pluggable Databases	8-24
Managing Resources Between PDBs	8-25
CDB Resource Plan Basics: Share	8-26
CDB Resource Plan Basics: Limits	8-29
CDB Resource Plan: Full Example	8-30
Creating a CDB Resource Plan	8-31
Maintaining a CDB Resource Plan	8-32
Creating CDB Resource Plan: SQL Example	8-33
Viewing CDB Resource Plan Directives	8-36
Enabling a CDB Resource Plan	8-37
Managing Resources Within a PDB	8-38
Managing PDB Resource Plans	8-39
Putting It Together	8-40
Considerations	8-41
Consolidated Database Replay Use Cases	8-42
Use Cases: Source Workloads	8-43
The Big Picture	8-44
Step 1	8-45
Step 2	8-46
Step 3	8-47
Step 4	8-48
Consolidated Replay Steps	8-49
Quiz	8-50
Summary	8-51
Practice 8 Overview: Performance	8-52

9 Miscellaneous

- Course Structure 9-2
- Objectives 9-3
- Using Oracle Data Pump with PDBs 9-4
- Exporting from non-CDB and Importing into PDB 9-5
- Exporting and Importing Between PDBs 9-6
- Exporting from PDB and Importing into non-CDB 9-7
- Full Transportable Export/Import: Overview 9-8
- Full Transportable Export/Import: Usage 9-9
- Full Transportable Export/Import: Example 9-11
- Transporting a Database Over the Network: Example 9-12
- Using SQL*Loader with PDBs 9-13
- Quiz 9-14
- Auditing Actions in a CDB and PDBs 9-15
- Creating an Audit Policy at CDB/PDB Level 9-16
- Enabling / Disabling the Audit Policy 9-17
- Viewing the Audit Policy 9-18
- Viewing the Audit Records CDB_UNIFIED_AUDIT_TRAIL 9-19
- Dropping the Audit Policy 9-20
- Audit Cleanup 9-21
- Quiz 9-22
- Securing Data with Transparent Data Encryption 9-23
- The Keystore and Master Keys 9-24
- Creating the Keystore 9-25
- Opening the Keystore 9-26
- Setting Master Encryption Keys 9-27
- Unplug and Plug PDB with Encrypted Data 9-28
- Securing Data with Oracle Database Vault 9-29
- Quiz 9-30
- Using Xstreams with a CDB and PDB 9-31
- Creating a Standby of a CDB 9-32
- Quiz 9-34
- Scheduling Operations in a PDB 9-35
- Jobs Coordinator and Resources 9-36
- Mining Statements of a PDB Using LogMiner 9-37
- Summary 9-38
- Practice 9 Overview: Miscellaneous 9-39

A Consolidated Database Replay Procedures and Tables

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

B New Processes, Views, Parameters, Packages and Privileges

- New Views B-3
- New Parameters and Packages B-4

1

Introduction



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this course, you should be able to:

- Distinguish non-CDB and multitenant container databases
- Define when to use multitenant container databases
- Explain the multitenant architecture
- Create and administer a multitenant container database
- Create and administer a pluggable database
- Configure the storage within a multitenant container database and its pluggable databases
- Secure access and data within a multitenant container database and its pluggable databases
- Implement a backup, recovery, and flashback strategy
- Employ monitoring procedures
- Allocate resources within a multitenant container database and its pluggable databases
- Move data between non-CDB and pluggable databases



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the “Oracle Database 12c: Administration Workshop” and “Oracle Database 12c: Backup and Recovery Workshop” courses, you learnt how to create, administer, monitor, back up, and recover non-CDBs (old database architecture).

In this course, you distinguish non-CDBs from the new types of databases that are multitenant container databases and pluggable databases, and define when it is appropriate or not to use multitenant container databases.

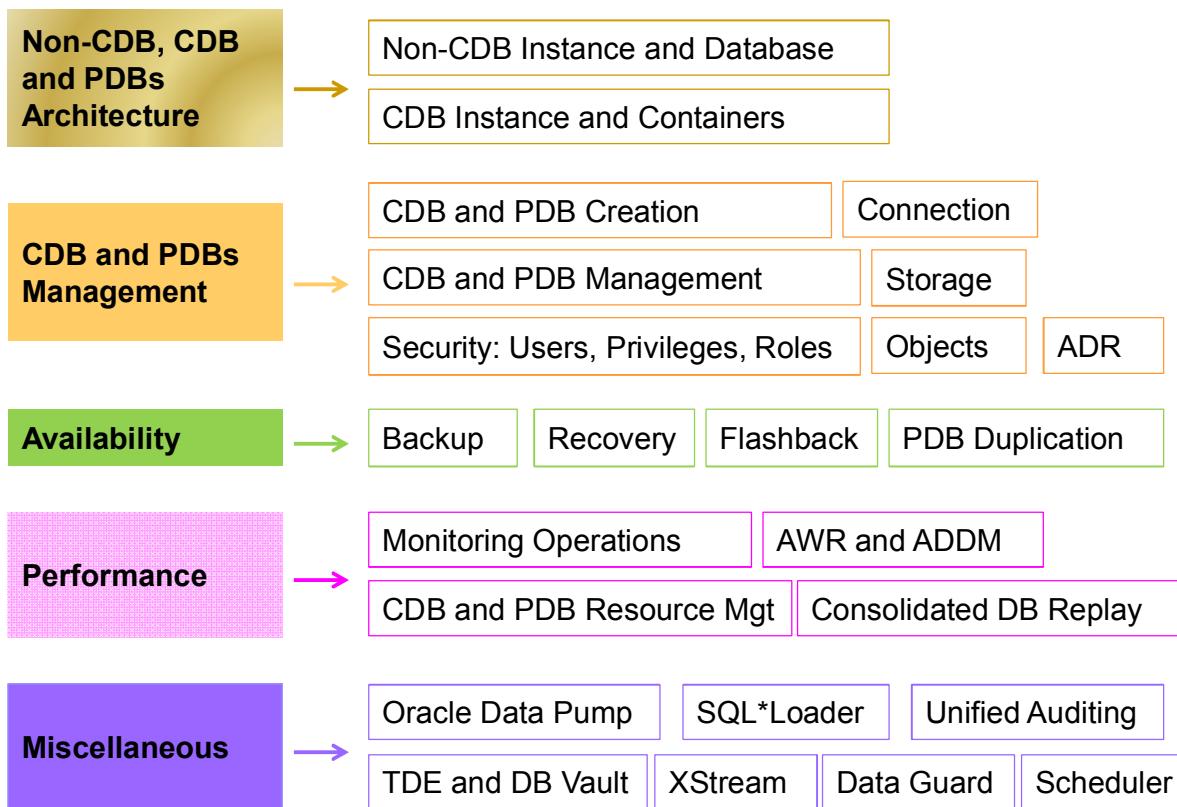
You create new multitenant container databases and pluggable databases, and learn how to administer these types of databases.

You also configure the multitenant container database and pluggable databases to support applications and perform tasks such as securing access and data in the multitenant container database and its pluggable databases, creating different types of users, and defining storage structures within the multitenant container database and its pluggable databases. This course uses fictional applications.

Database administration does not end after you configure your multitenant container database and pluggable databases. You also learn how to protect data by designing a backup and recovery strategy, how to monitor operations to ensure that it operates smoothly and manage resources allocation within the multitenant container database and its pluggable databases.

And finally you move data from non-CDBs to pluggable databases, from pluggable databases to non-CDBs, and from pluggable databases to pluggable databases.

Course Structure



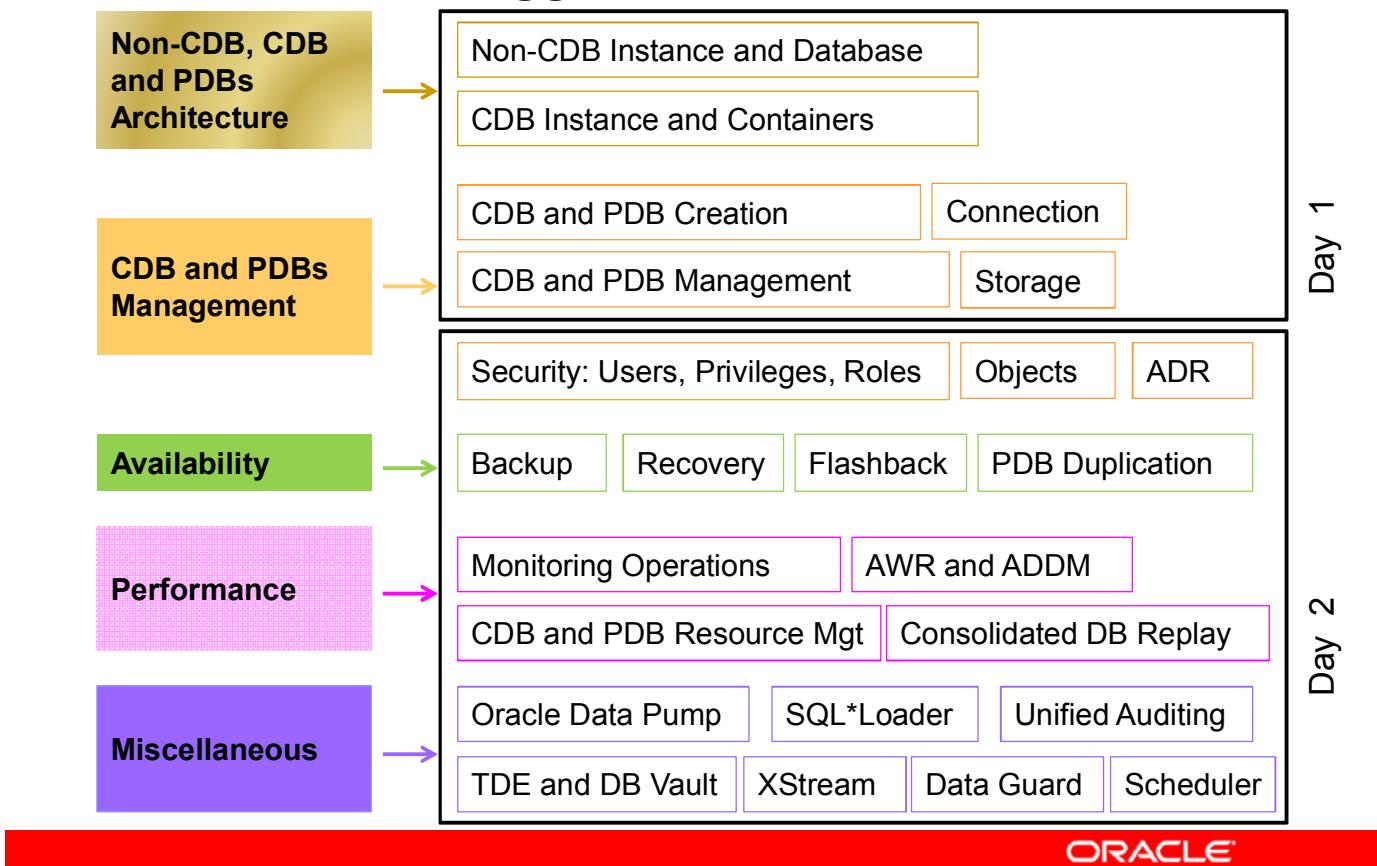
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The lessons in the course explain how to:

- Distinguish *non-CDBs*—not multitenant container databases—from the new types of databases that are multitenant container databases. You will see the use of the term *CDB*, pluggable databases, and also the term *PDB*.
- Understand the multitenant architecture of a CDB and its different containers
- Create a CDB and PDBs, start up, and shut down a CDB and a PDB
- Connect to a CDB and to any of its PDBs
- Configure the CDB and its PDBs to support applications:
 - Create tablespaces, datafiles in the CDB and its PDBs.
 - Create different types of users.
 - Secure access and data, granting privileges in the CDB and its PDBs.
- Design a backup, recovery, and flashback strategy at the CDB and PDB level
- Monitor all operations in the CDB and PDBs using the AWR and ADDM
- Tune the applications using SQL tuning advisors and consolidated database replay
- Manage resources allocation between PDBs and within a PDB
- Export and import data from a non-CDB to a PDB, from a PDB to a non-CDB, from a PDB to another PDB, and load data in a PDB
- Perform auditing in a CDB and PDBs and use other products

Suggested Schedule



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The first day will cover non-CDBs, the multitenant architecture and the first part of management.

The second day will cover the second part of management, availability, performance and miscellaneous topics of multitenant container databases and pluggable databases.

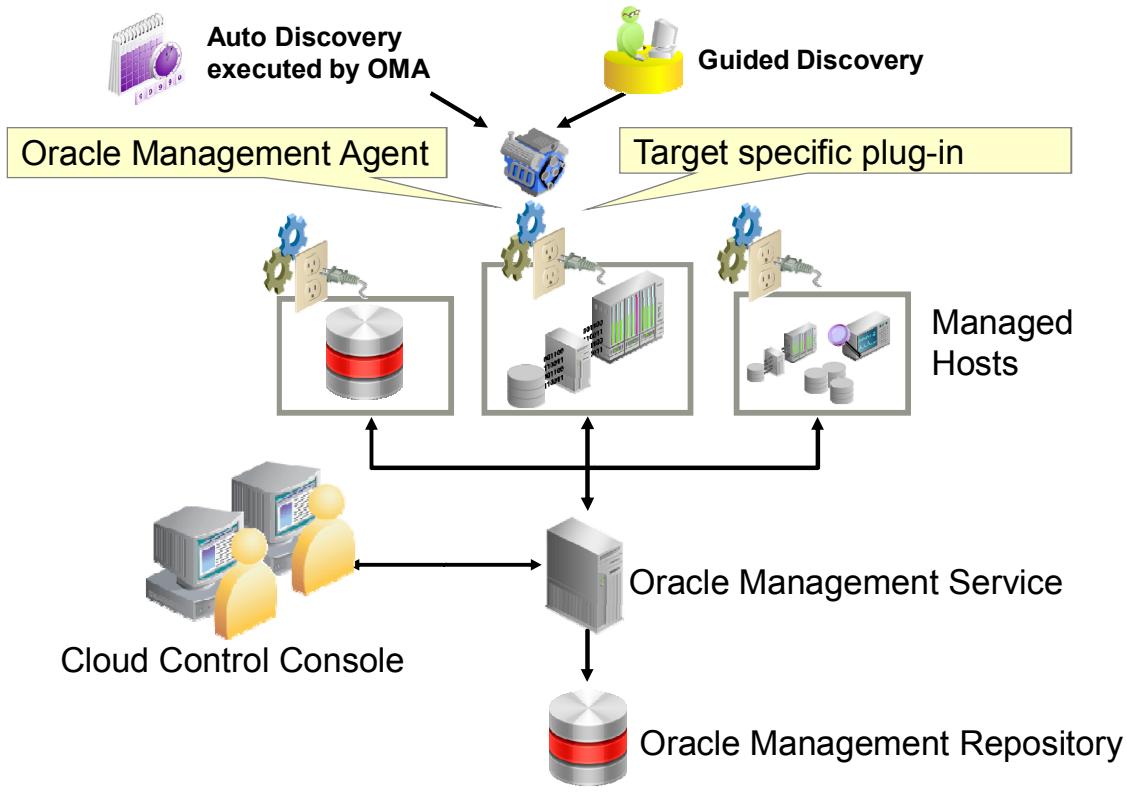
Enterprise Manager Cloud Control and Other Tools



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Target Discovery: Multitenant Container Database



ORACLE

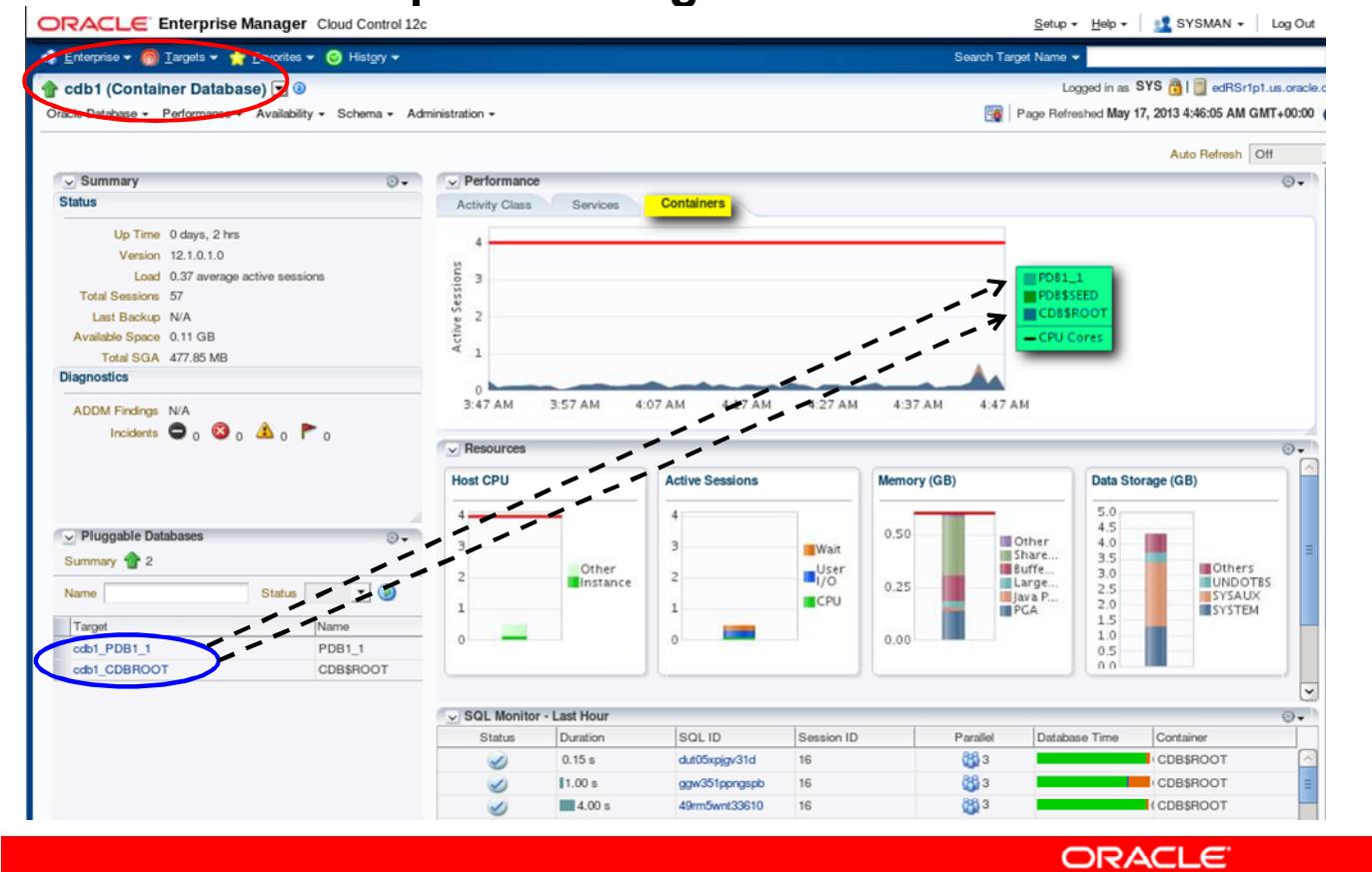
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The diagram in the slide shows that Enterprise Manager Cloud Control (EM Cloud Control) is composed of four main components:

- The Oracle Management Agent (OMA) runs on hosts, gathering metric data about those host environments as well as using plug-ins to monitor availability, configuration, and performance and to manage targets running on the host.
- The agents communicate with the Oracle Management Service (OMS) to upload metric data collected by them and their plug-ins.
- In turn, the OMS stores the data it collects in the Oracle Management Repository (OMR) where it can be accessed by the OMS for automated and manual reporting and monitoring. The OMS also communicates with the agents to orchestrate the management of their monitored targets.
- As well as coordinating the agents, the OMS runs the Cloud Control Console web pages that are used by administrators and users to report on, monitor, and manage the computing environment that is visible to Cloud Control via the agents and their plug-ins.

After the agent has been installed on a host, it needs to look for targets that it can manage. Guided discovery allows you to nominate a family of target types that you want to search for, such as multitenant container databases, non-CDBs, and listeners, and then the agents where you want that search to be executed. You can also configure auto discovery to run at regular intervals and get an agent to search for known targets unattended, and promote discovered targets to become managed targets.

Enterprise Manager Cloud Control



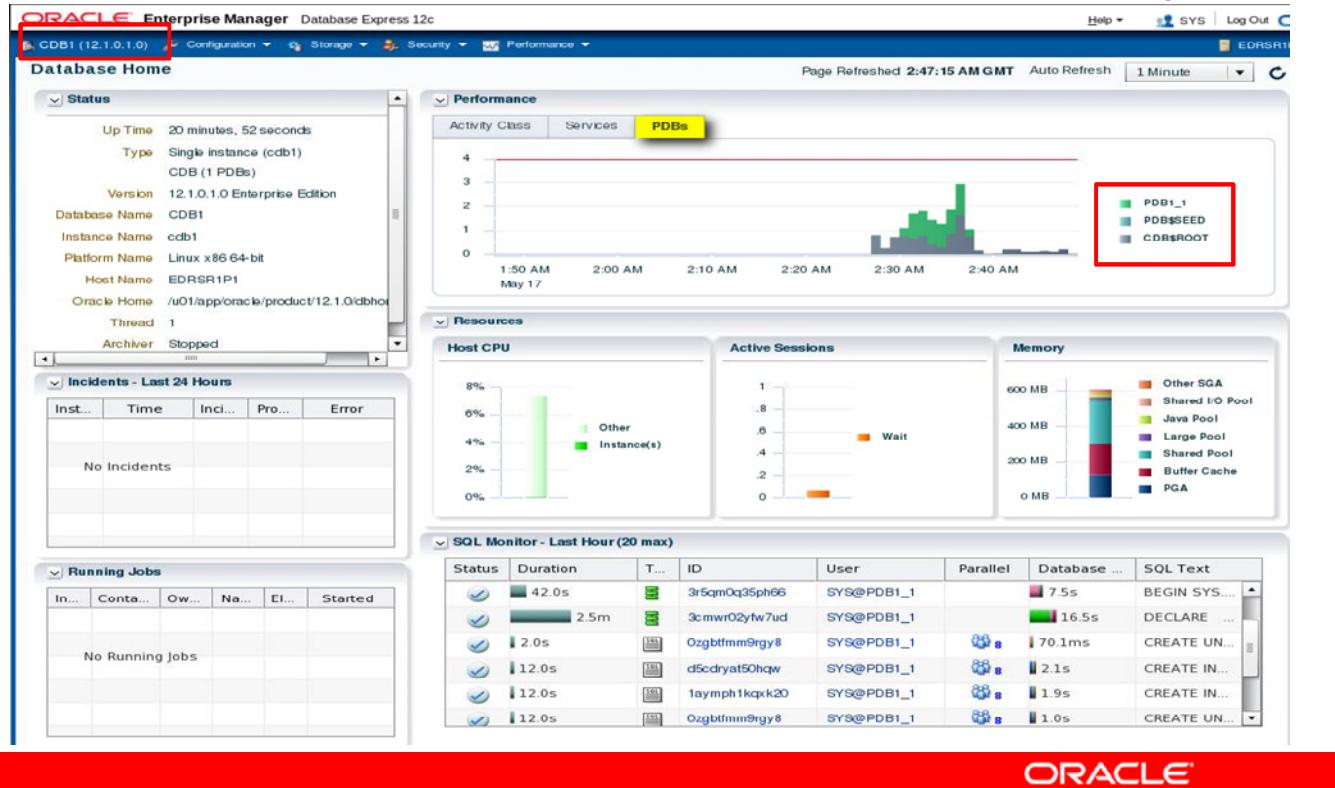
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

The screenshot in the slide shows the Cloud Control homepage from where you can perform any database administrator operation on CDBs and PDBs, such as creating users, granting privileges, backing up datafiles, creating tablespaces, SQL monitoring and tuning as you will see in the following lessons.

Enterprise Manager Database Express

Available when the multitenant container database is open



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

With Oracle Database 12c, Enterprise Manager Database Control is no longer available. Enterprise Manager Database Express replaces it.

The screenshot in the slide shows the Enterprise Manager Database Express homepage that presents an overall view of a multitenant container database instance status and activity as it does for a non-CDB instance.

However, if the DBA needs to perform administrative tasks for a PDB, such as creating users, tablespaces, and granting privileges for a specific PDB, Enterprise Manager Cloud Control is the tool to use.

Enterprise Manager Database Express is built on the Common Reporting Framework in the database. Because of this, Enterprise Manager Database Express is only available when the multitenant container database is open. This means that Enterprise Manager Database Express cannot be used to start up the database. Other operations that require that the database change state, such as enable or disable ARCHIVELOG mode, are also not available in Enterprise Manager Database Express.

Configuring Enterprise Manager Database Express

Configure an HTTP or HTTPS listener port for each database instance:

- Verify DISPATCHERS parameter.

```
dispatchers=(PROTOCOL=TCP) (SERVICE=sampleXDB)
```

- Use DBMS_XDB_CONFIG.setHTTPsPort procedure.

```
exec DBMS_XDB_CONFIG.setHTTPsPort(5500)
```

- Connect to the Enterprise Manager Database Express console with:

```
https://hostname:5500/em
```

- Use a different port for each instance.
- Browser requires Flash Plugin.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Enterprise Manager Database Express is configurable with a single click in Database Configuration Assistant (DBCA).

Enterprise Manager Database Express requires that the XMLDB components are installed. All Oracle version 12.1.0 databases will have XMLDB installed.

To activate Enterprise Manager Database Express in a database verify that the DISPATCHERS initialization parameter has at least one dispatcher configured for the XMLDB service with the TCP protocol.

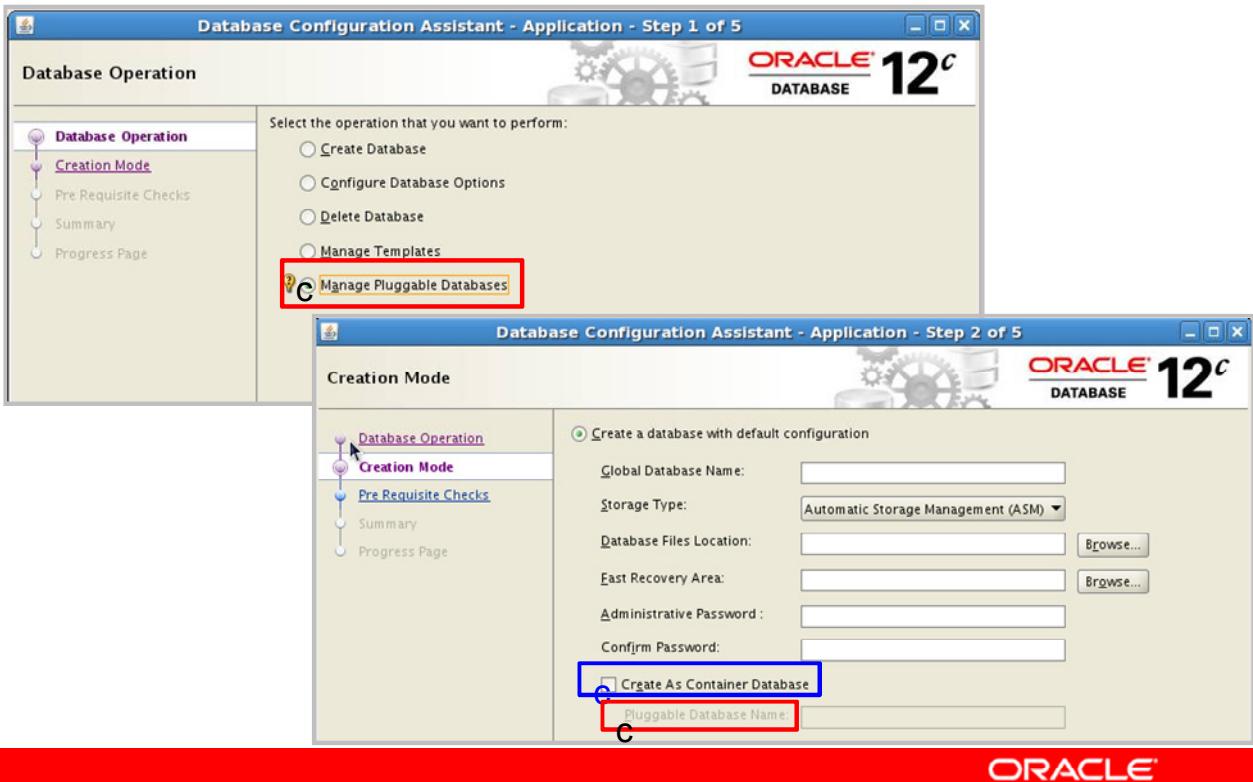
Use the setHTTPsPort (secured with SSL) or the setHTTPPort procedure in the DBMS_XDB_CONFIG package to configure a used port on the server. Connect to the EM Database Express console with the URL shown in the slide if you configured the port using the setHTTPsPort procedure (default used by DBCA). Substitute the host name of the server, and the port number used in the setHTTPsPort procedure. If you configured the port using the setHTTPPort procedure, use a URL such as `http://hostname:5500/em`.

If you have multiple database instances to monitor on the same machine, each one will require a different port. To find the port used for each database instance, use either of the following statements depending on the procedure used for the configuration:

```
SQL> SELECT dbms_xdb_config.gethttpsport FROM DUAL;  
SQL> SELECT dbms_xdb_config.gethttpport FROM DUAL;
```

Database Configuration Assistant

Multitenant Container database and Pluggable database



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The screenshot in the slide shows that Database Configuration Assistant in Oracle Database 12c allows the creation and management of new types of database:

- Multitenant container database
- Pluggable database

The creation and management of these new types of database are covered in the following lessons.

SQL*Plus

Use SQL*Plus to perform any DBA operations:

1. Set ORACLE_HOME=/u01/app/oracle/product/db_home1.
2. Set ORACLE_SID=cdb1.

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

```
SQL> CREATE TABLESPACE tbs1  
2  DATAFILE '/u01/app/oradata/CDB1/tbs01.dbf' SIZE 325M ;
```

```
SQL> CREATE USER u1 IDENTIFIED BY u1;
```

```
SQL> SELECT * FROM DBA_TABLESPACES;  
SQL> SELECT * FROM DBA_USERS;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use SQL*Plus to perform any administration operation on a multitenant container database or on a pluggable database, as you would do for any non-CDB.

Set the ORACLE_SID environment variable to the instance name and the ORACLE_HOME environment variable to the Oracle Database 12c directory. You can use either the export shell command or the oraenv Oracle utility, which prompts for the database SID and ORACLE_HOME if not already set. The following environment variables are set:

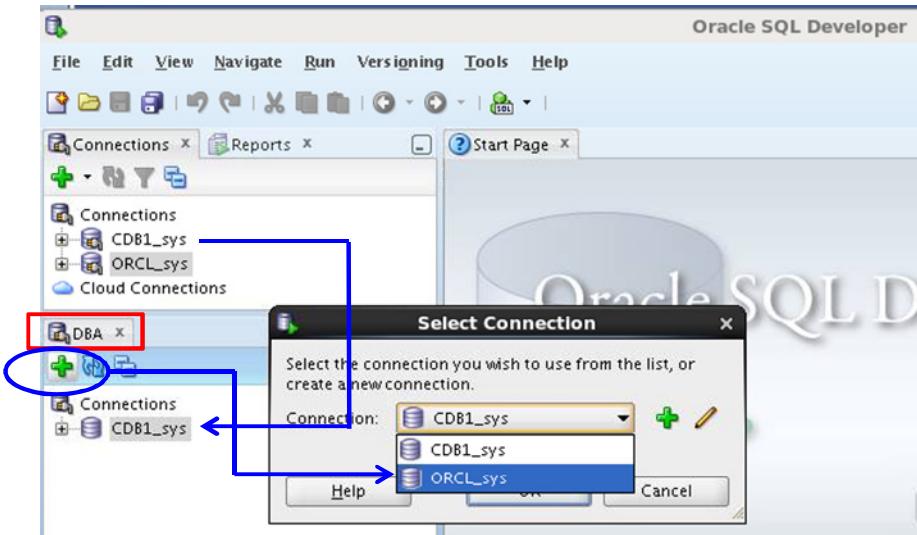
- ORACLE_SID: Oracle system identifier
- ORACLE_HOME: Top-level directory of the Oracle system hierarchy
- PATH: Old ORACLE_HOME/bin removed, new one added
- ORACLE_BASE: Top-level directory for storing database files

```
$ . oraenv  
ORACLE_SID = [oracle] ? orcl  
The Oracle base for  
ORACLE_HOME=/u01/app/oracle/product/12.1.0/dbhome_1 is  
/u01/app/oracle
```

The examples in the slide show how to connect to a CDB instance, start it up, create a tablespace, create a user, and finally view data dictionary information from the data dictionary views. In the following lessons, you will learn how to perform DBA operations on CDBs and PDBs using new commands and how to retrieve data dictionary information using new views.

Oracle SQL Developer: Connections

Perform DBA operations in the **DBA navigator** using **DBA connections**:



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

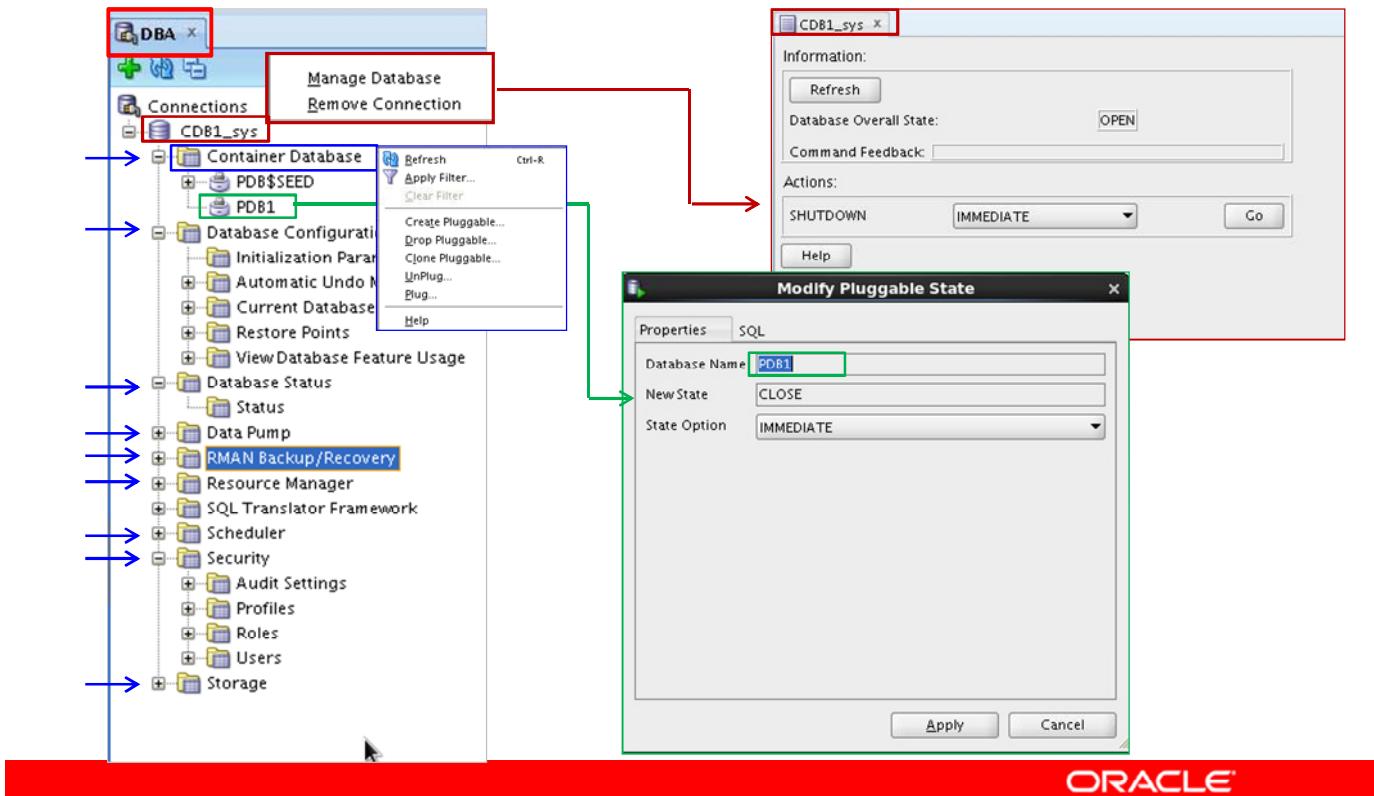
ORACLE

The screenshot in the slide shows that Oracle SQL Developer allows the execution of multitenant container and pluggable database administrative tasks.

SQL Developer enables users with database administrator privileges to view and edit certain information relevant to perform DBA operations. To perform DBA operations, use the DBA navigator, which is similar to the Connections navigator in that it has nodes for all defined database connections. If the DBA navigator is not visible, select View, then DBA. You should add only connections for which the associated database user has DBA privileges.

Oracle SQL Developer: DBA Tasks

Performing DBA tasks through **DBA** navigator:



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

The screenshot in the slide shows that the DBA operations in SQL Developer that can be performed are the following:

- Multitenant container database startup/shutdown and pluggable database open/close
- Pluggable database creation, cloning, plugging, unplugging, dropping
- Pluggable database open mode setting
- Multitenant container database configuration: Initialization Parameters, Automatic Undo Management, Current Database Properties, Restore Points, View Database Feature Usage
- Multitenant container database status view
- Data Pump Export and Import jobs
- RMAN Backup/Recovery actions
- Resource Manager configuration
- Scheduler setting
- Security configuration such as audit settings, profiles, roles, users
- Storage configuration for archive logs, control files, redo log groups, CDB and PDB datafiles, CDB and PDB tablespaces, CDB and PDB temporary tablespace groups

All the new tasks related to CDB and PDBs are covered in the following lessons.

Quiz

Oracle SQL Developer allows multitenant container database and pluggable database administration operations.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a

Summary

In this lesson, you should have learned how to:

- View multitenant container database and pluggable databases in the Oracle Enterprise Manager Cloud Control interface
- View multitenant container databases in Oracle Enterprise Manager Database Express
- View multitenant container database and pluggable databases in DBCA
- View new functionalities in SQL*Developer interface to administer multitenant container database and pluggable databases



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 1 Overview: Using Enterprise Manager Cloud Control

These practices cover the following topics:

- Accessing Enterprise Manager Cloud Control
- Setting the Databases page as the home page
- Adding a multitenant container database instance as a new target monitored by EM CC
- Creating a new, named credential to access the CDB
- Using the named credential to log in to the CDB



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

View the “Oracle Enterprise Manager 12c: Console Overview and Customization” demonstration (unless your instructor just demonstrated those topics) (8 mins).

Optional demonstrations on related topics are available:

- Oracle Enterprise Manager 12c: Create an Enterprise Manager Administrator (6 mins)
- Oracle Enterprise Manager 12c: Create and Use Named Credentials (6 mins)
- Oracle Enterprise Manager 12c: Create SSH Key Named Credentials (3 mins)
- Oracle Enterprise Manager 12c: Discover and Promote Unmanaged Hosts and Targets (3 mins)

Optional Oracle By Example (OBEs) on related topics are available:

- Oracle Enterprise Manager 12c Enterprise Ready Framework: Create and Use Credentials (60 mins)
- Oracle Enterprise Manager 12c Enterprise Ready Framework: Create a Super Administrator Account (5 mins)

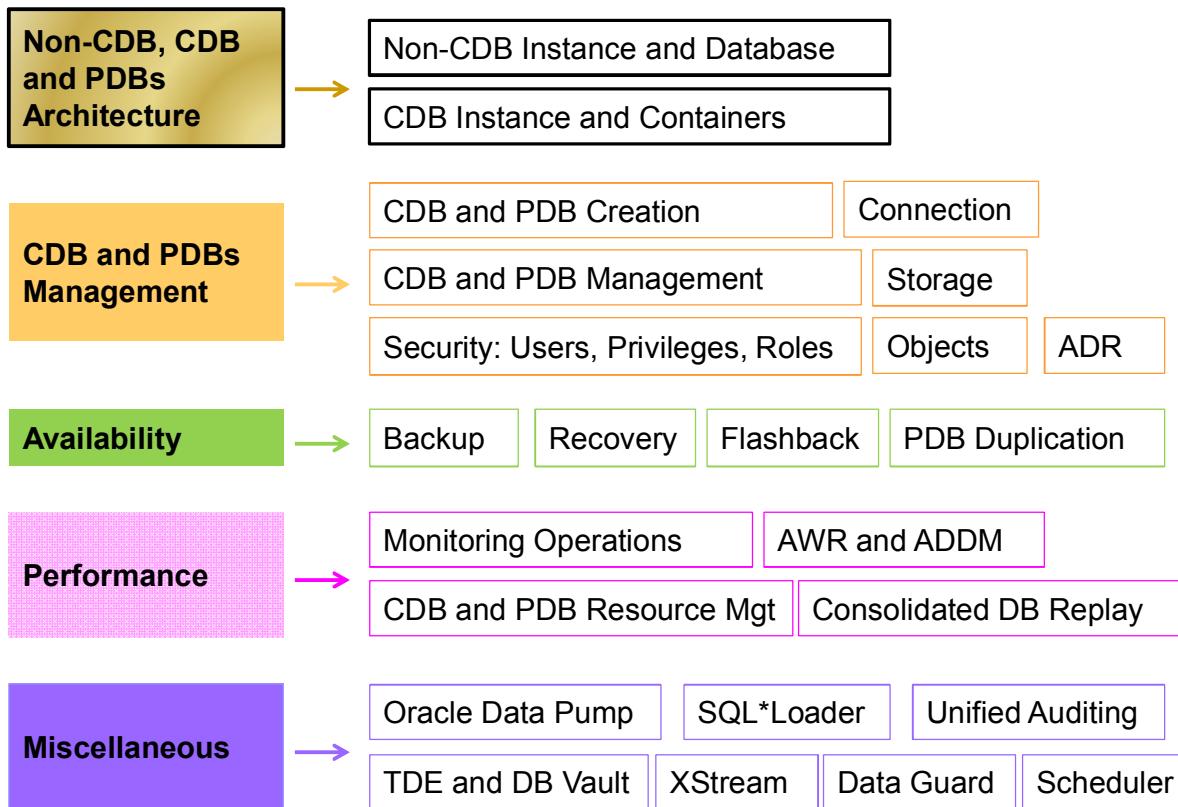
Basics of Multitenant Container Database and Pluggable Databases



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Course Structure



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This lesson explains how to:

- Distinguish non-CDBs (*non-CDB* is used here as a shorthand for the not multitenant container database) from the new types of databases that are multitenant container databases . You will see the use of the term *CDB*, pluggable databases, and also the term *PDB*.
- Understand the multitenant architecture of a CDB and its different containers

Objectives

After completing this lesson, you should be able to:

- Describe the multitenant architecture
- Describe the root and pluggable database containers
- Differentiate the root from a pluggable database
- Explain pluggable database plugging
- List impacts in various areas



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Note: For a complete understanding of Oracle Multitenant new option and usage, refer to the following guides in the Oracle documentation:

- *Oracle Database Administrator's Guide 12c Release 1 (12.1)*
- *Oracle Database Concepts 12c Release 1 (12.1)*

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

- *Oracle Database 12c New Features Demo Series* demonstrations:
 - *Multitenant Architecture*
 - *Basics of CDB and PDB Architecture*
- *Oracle By Example (OBE)*:
 - *Performing Basic Tasks on Multitenant Container Databases and Pluggable Databases*

Challenges

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

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



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

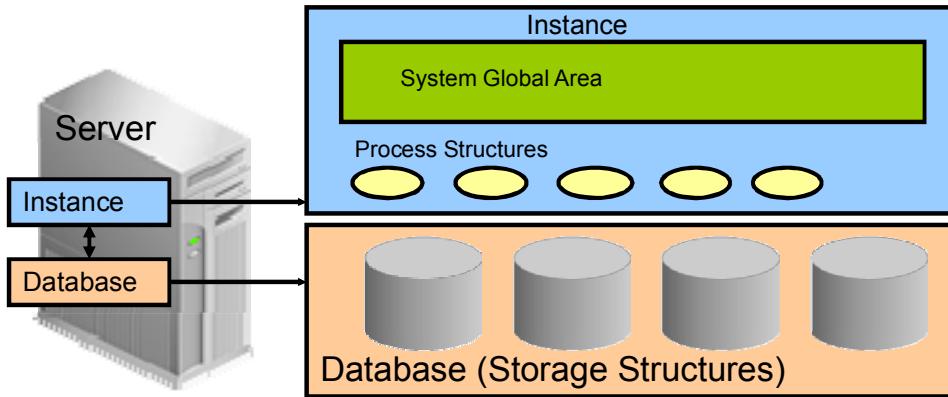
Oracle Database 12c brings a new architecture that lets you have many *pluggable databases* inside a single Oracle Database occurrence. What is the benefit of using the multitenant architecture in Oracle Database 12c?

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

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

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

Non-CDB Architecture



Multiple monolithic or non-CDBs share nothing:

- Too many background processes
- High shared/process memory
- Many copies of Oracle metadata



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In Oracle 11g database, the only kind of database that is supported is a non-CDB. The old architecture is referred to as the non-CDB architecture—the term *non-CDB* will be used as a shorthand for an occurrence of a pre-12.1 database that uses the pre-12.1 architecture—that requires its own instance and, therefore, its own background processes, memory allocation for the SGA, and it needs to store the Oracle metadata in its data dictionary. The database administrator can still create Oracle 12c non-CDBs with the same pre-12.1 architecture. These databases are not multitenant container databases or non-CDBs.

When you have to administer small departmental database applications, you have to create as many databases as applications and, therefore, multiply the number of instances, consequently the number of background processes, memory allocation for the SGAs, and provision enough storage for all data dictionaries of these databases.

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

New Multitenant Architecture: Benefits

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



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Consolidating many non-CDB databases onto a single platform reduces instance overhead, avoids redundant copies of data dictionaries, and consequently storage allocation, and benefits from fast provisioning, time saving upgrading, better security through separation of duties and application isolation. The new 12c database that consolidates databases together is a multitenant container database or CDB, and a database consolidated within a CDB, a pluggable database or PDB.

DBA resource costs are reduced with:

- *No application change and very fast provisioning:* A new database can be provisioned very quickly. A clone of a populated database can be created very quickly. A populated database can be quickly unplugged from its CDB on one platform and quickly plugged into a CDB on a different platform. A non-CDB can quickly be plugged into a CDB.
- *Fast upgrade and patching of the Oracle Database version:* The cost (time taken and human effort needed) to upgrade many PDBs is the cost of upgrading a single Oracle Database occurrence. You can also upgrade a single PDB by unplugging it and plugging it into a CDB at a different Oracle Database version.

The multitenant architecture maintains:

- *Secure separation of duties*: The administrator of an application can do all the required tasks by connecting to the particular PDB that implements its back end. However, someone who connects to a PDB cannot see other PDBs. To manage PDBs as entities (for example, to create or drop or unplug or plug one), the system administrator needs to connect to the CDB. For these specific tasks, new privileges need to be granted.
- *Isolation of applications* that may not be achieved manually unless using Database Vault for example. A good example of isolation is dictionary separation enabling Oracle Database to manage the multiple PDBs separately from each other and from the CDB itself.

Other Benefits of Multitenant Architecture

- Ensures **full backwards-compatibility** with non-CDBs
- Fully operates with RAC
- Is integrated with Enterprise Manager and Resource Manager
- Allows central management and administration of multiple databases
 - Backups / Disaster recovery
 - Patching and Upgrades

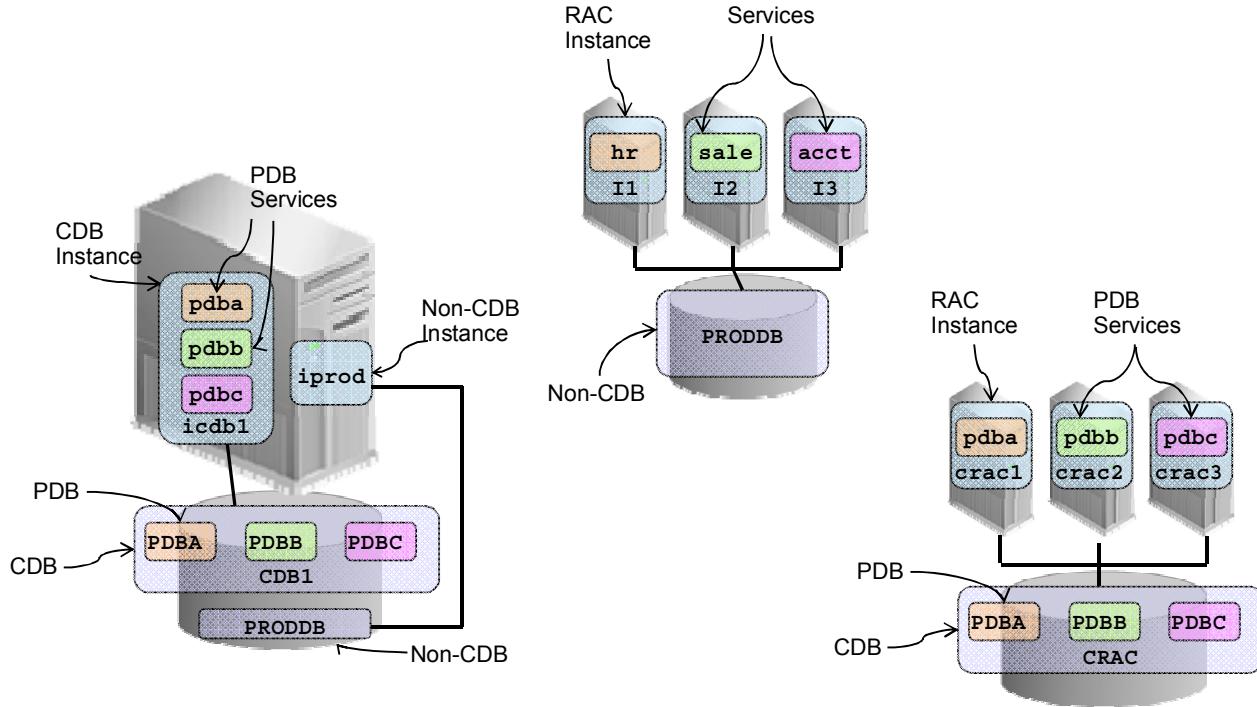


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- The multitenant architecture ensures the backwards-compatibility principle. An example is the data dictionary views. The DBA_OBJECTS view shows the same results in a PDB as in a non-CDB for a particular application.
- The multitenant architecture is designed to be fully interoperable with RAC. Each instance in a RAC opens the CDB as a whole. A session sees only the single PDB it connects to.
- Enterprise Manager integrates CDBs and models the separation of duties of the CDB administrator and the PDB administrator.
 - A CDB can be defined as a target. An Enterprise Manager user can be given the credentials to act as a CDB administrator in such a target.
 - A PDB can be set up as a subtarget of a CDB target. An Enterprise Manager user can be given the credentials to act as a PDB administrator in such a target. An Enterprise Manager user that has been set up with the credentials to act as a PDB administrator for a particular PDB is able to connect to that one PDB and is unaware of the existence of peer PDBs in the same CDB. Moreover, when the intention is to carry out the duties of an application administrator, this Enterprise Manager user is unaware that the environment is a CDB and not a non-CDB.

- Resource Manager is extended with new capabilities to allow the management of resources between the PDBs within a CDB. The backwards-compatibility principle implies that Resource Manager must function in exactly the same way within a PDB as it does in a non-CDB.
- When you upgrade a whole CDB with n PDBs, you achieve the effect of upgrading n non-CDBs for the cost of upgrading one non-CDB.

Configurations



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Which are the possible instance/database configurations in Oracle Database 12c?

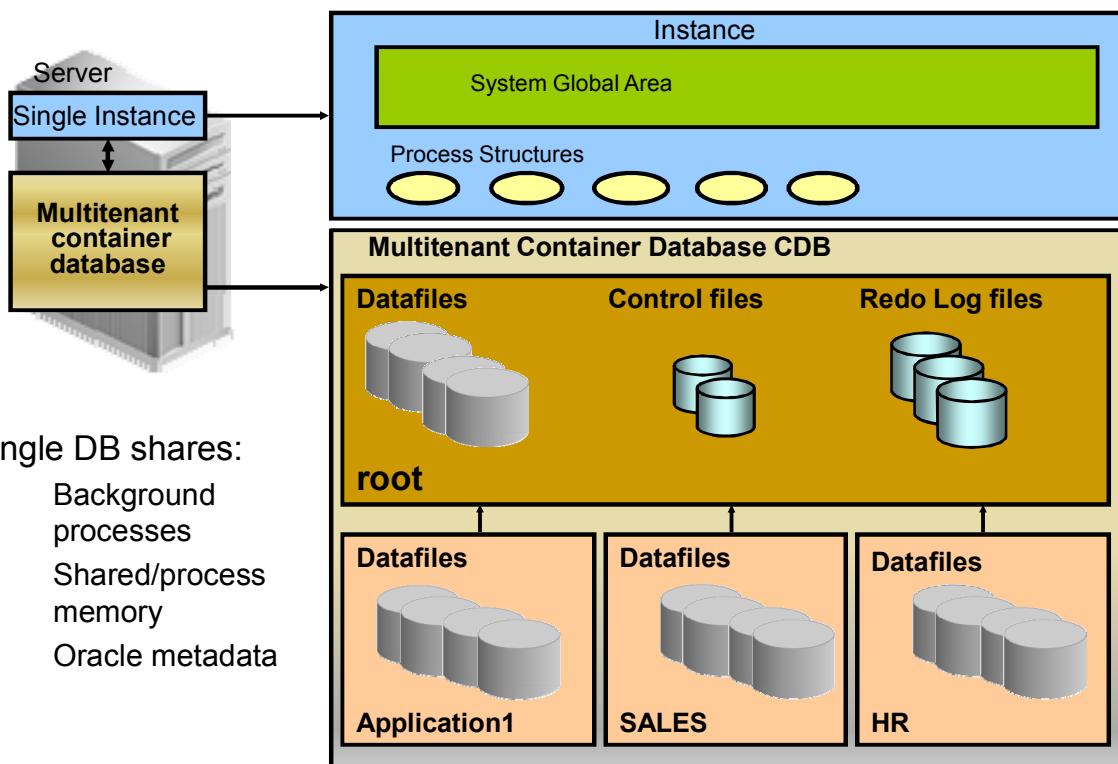
- Each database instance can be associated with one and only one non-CDB or multitenant container database.
- In a RAC environment, several instances can be associated to a non-CDB or multitenant container database.
- An instance is associated with an entire CDB.

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

In Oracle Database 12c, there are three possible configuration options:

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

Oracle Container Database



Single DB shares:

- Background processes
- Shared/process memory
- Oracle metadata

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

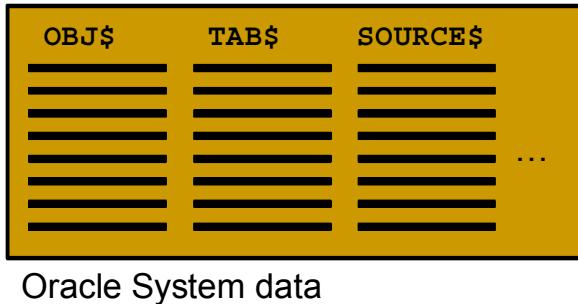
- Background processes
- Memory allocation
- Oracle metadata in several data dictionaries

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

When applications need to be patched or upgraded, the maintenance operation is performed only once on the CDB and consequently all applications are updated at the same time.

A Pristine Installation

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



Oracle System data

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

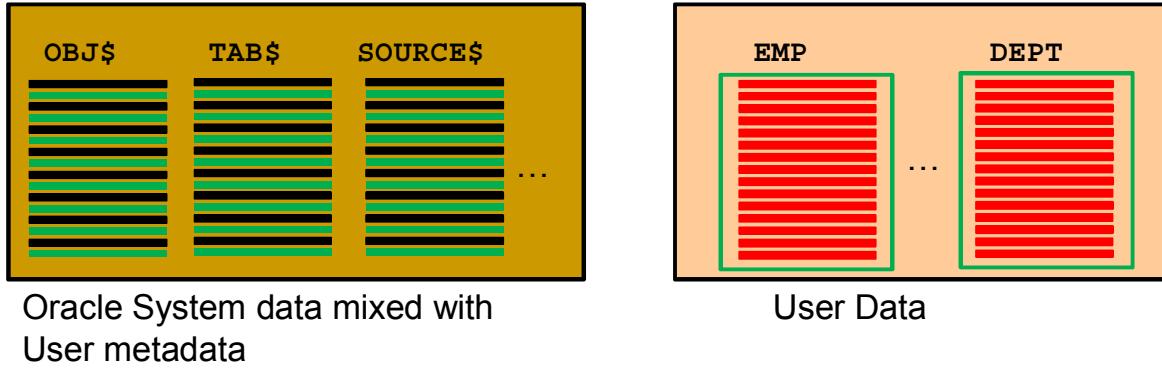
How is Oracle metadata shared between several applications in a non-CDB?
Immediately after the creation of the non-CDB, the only objects in the data dictionary are the Oracle-supplied objects. At this point, there is no user data.

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

User Data Is Added

In a non-CDB, user data is added:

- The metadata is mixed with the Oracle-supplied data in the data dictionary.



ORACLE

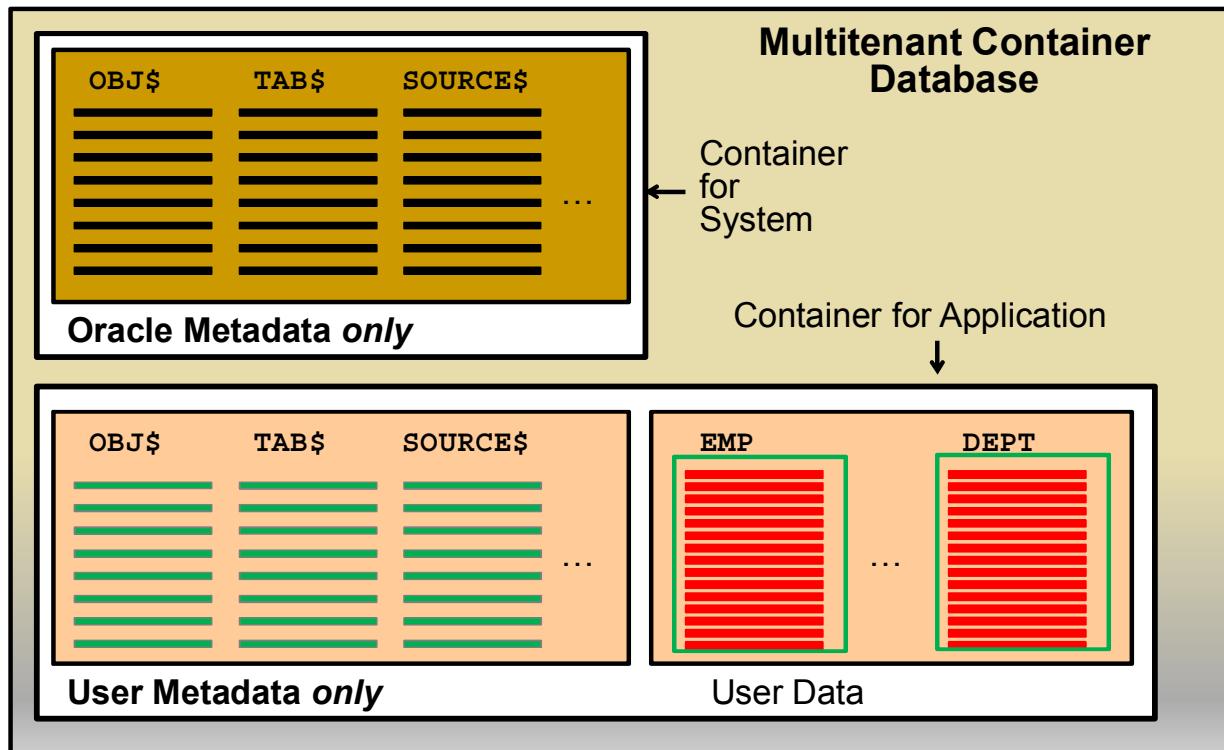
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

Best practice is to add this data in tablespaces dedicated to user data. Storing the data in separate tablespaces enables to protect, secure, and transport the data more easily. The metadata is though all mixed together in the data dictionary:

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

Separating SYSTEM and User Data



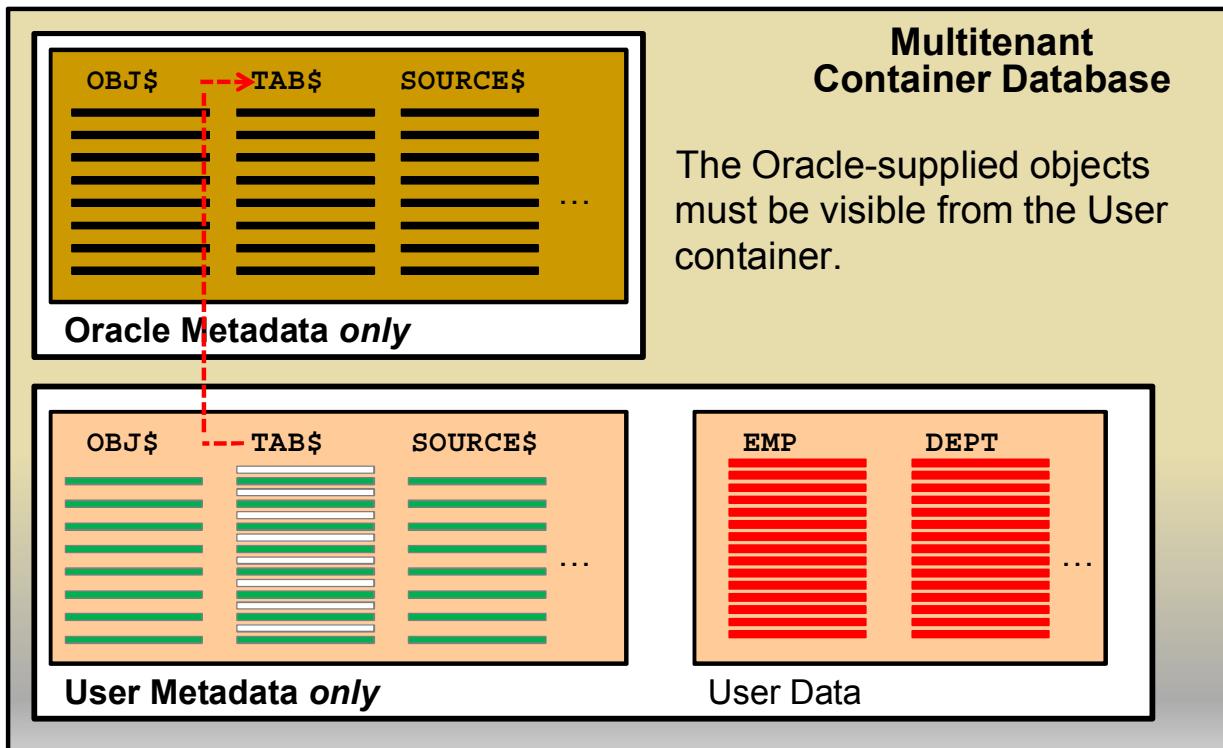
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In a multitenant container database, the concept of containers is introduced to separate the Oracle-supplied objects and the user data including the metadata into distinct containers. This can be thought of as an horizontal partitioning. There is a `SYSTEM` tablespace in each container holding a data dictionary.

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

SYSTEM Objects in the USER Container



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

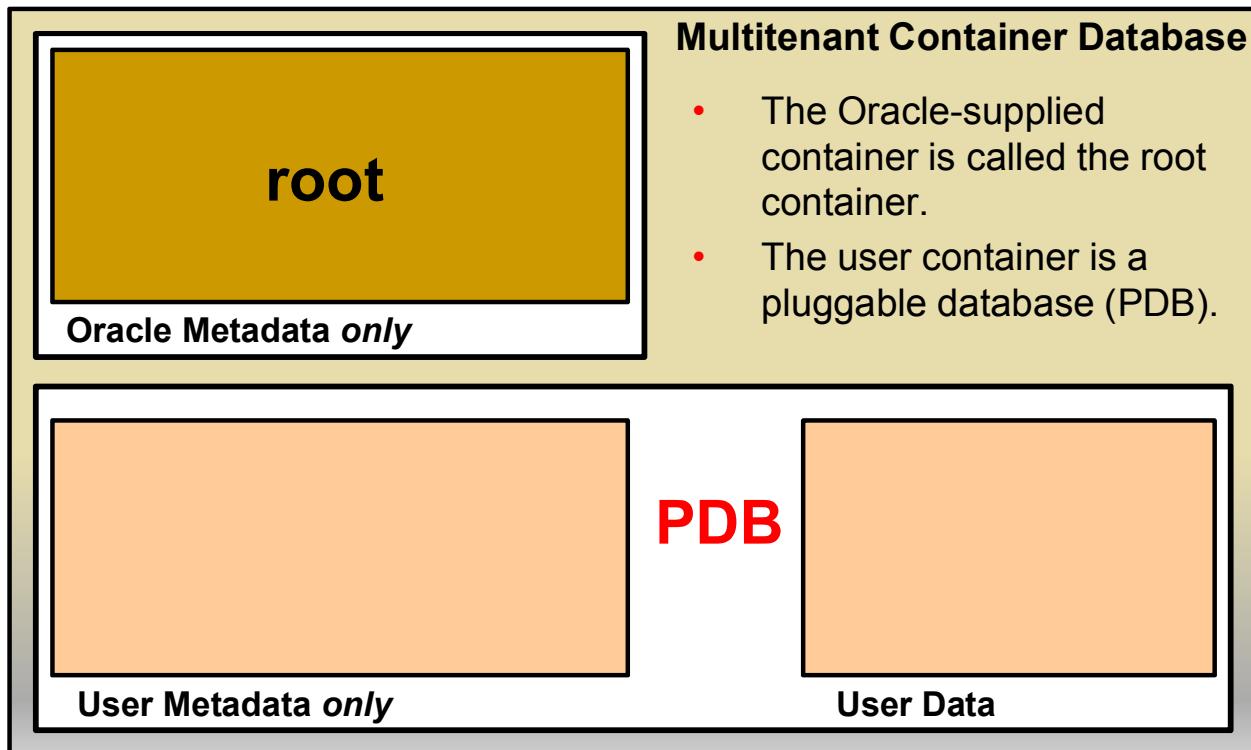
One of the goals of the multitenant architecture is that each container has a one-to-one relationship with an application.

Separating the metadata is the first step, the second is allowing the application or users inside the “user” container to access the Oracle-supplied objects.

The Oracle objects could have been duplicated in each user container, but that takes a lot of space, and would require every user container to be upgraded each time an Oracle-supplied object changes, for example, with patches.

A simple solution is to provide pointers from the user container to the Oracle-supplied objects to allow these “system” objects to be accessed without duplicating them in the user container. The user container has the pieces it needs to be a complete environment for a database application. The application can run in the container just as it does in a non-CDB.

Naming the Containers

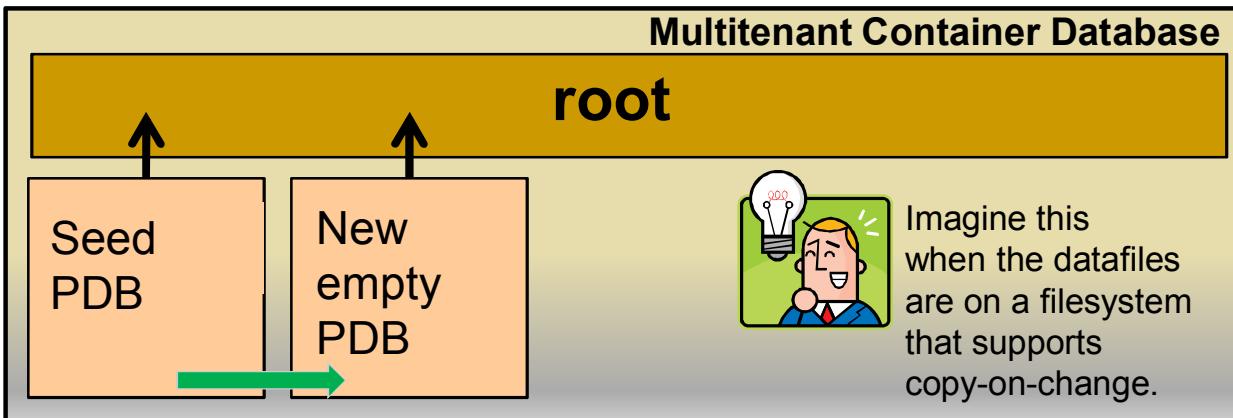


ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The Oracle-supplied objects reside in a container called the root container named `CDB$ROOT`. The user container is called a pluggable database (PDB) and has the name you give it when creating it or plugging it into the CDB.

Provisioning a Pluggable Database



Four methods:

- Create new PDB from PDB\$SEED pluggable database.
- Plug in a non-CDB.
- Clone a PDB from another PDB into the same or another CDB.
- Plug an unplugged PDB into the same or another CDB.

ORACLE

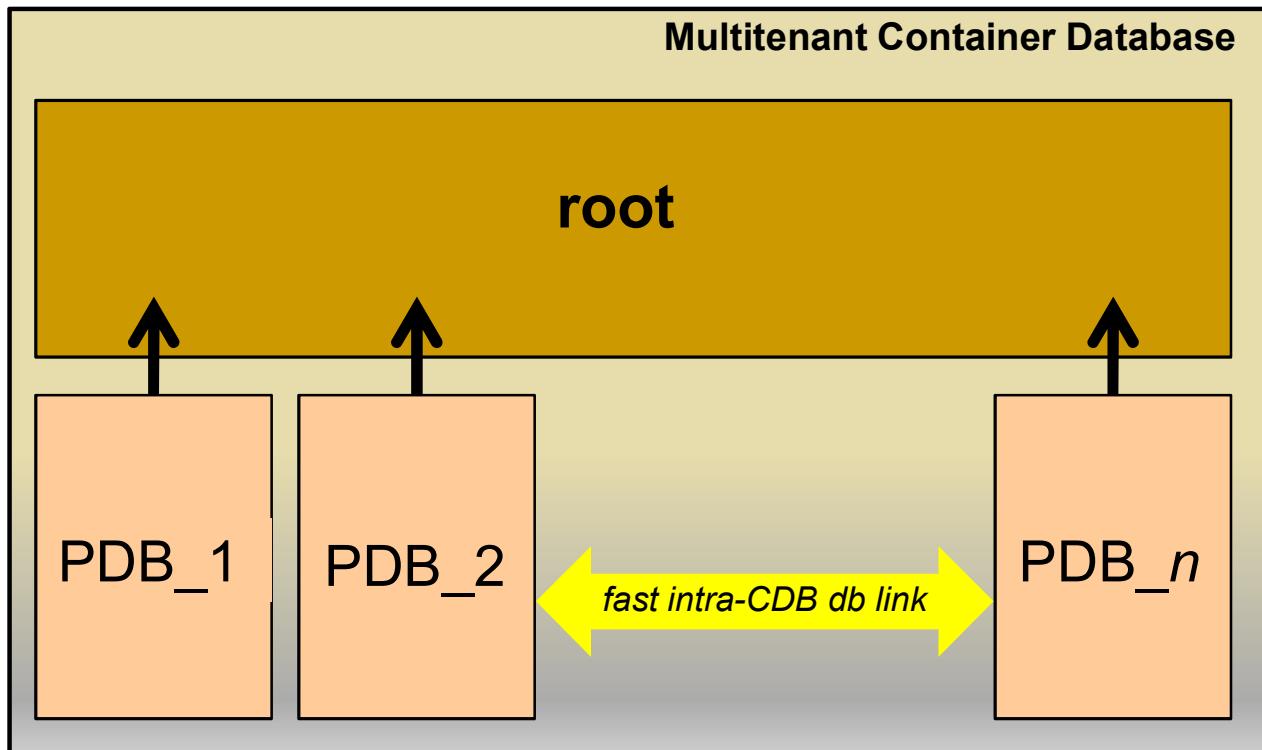
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

There are four methods to provision pluggable databases:

- Create a new PDB from PDB\$SEED pluggable database: for example for a brand new application implementation.
- Create a new PDB from a non-CDB: plug the non-CDBs in a CDB as PDBs, as part of migration strategy. It is also a good way to consolidate the non-CDBs into a CDB.
- Clone a PDB from another PDB into the same or another CDB: an example of this method is application testing.
- Plug an unplugged PDB into another CDB; for example, instead of upgrading a multitenant container database from one release to another, you can unplug a pluggable database from one Oracle Database release, and then plug it into a newly created multitenant container database from a higher release.

Interacting Within Multitenant Container Database



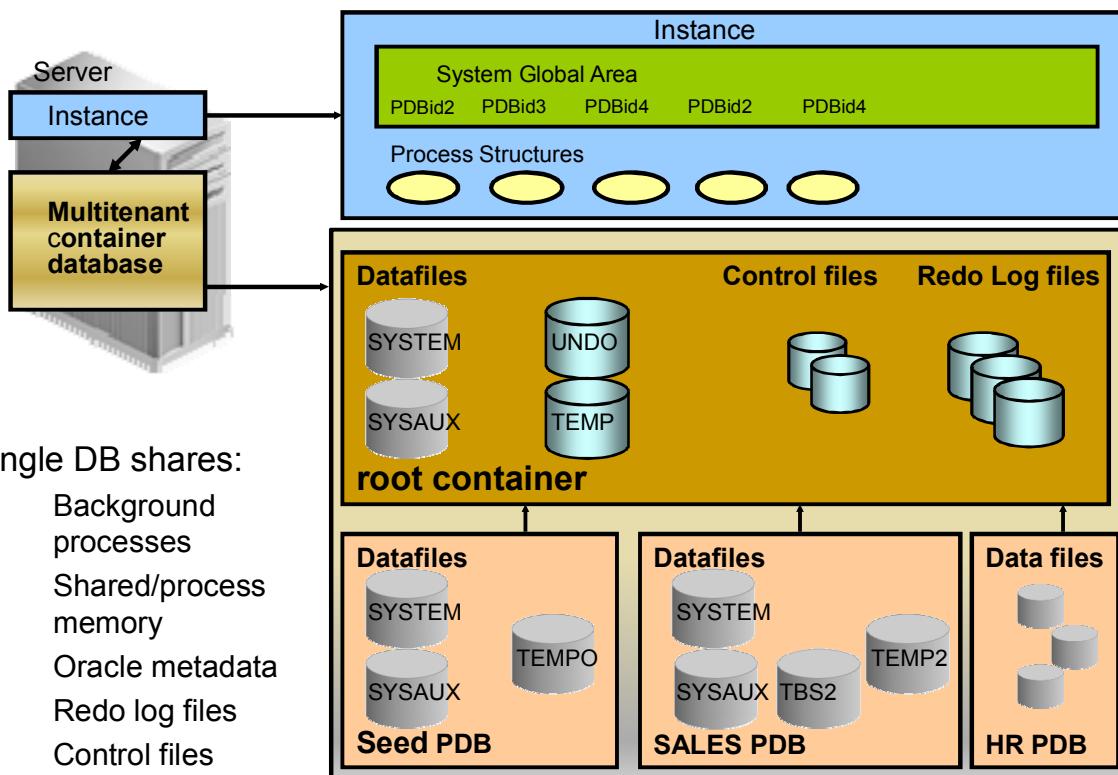
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

With a multitenant architecture that holds several PDBs, these once separated non-CDBs now may work with a single instance, sharing memory, disk and CPU resources, but maintaining application separation.

These databases shared data using database links. The database link still works, but now because the “link” communication does not leave the instance, the link is very fast.

Multitenant Container Database Architecture



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

- The redo log files are common for the whole CDB. The information it contains is annotated with the identity of the PDB where a change occurs. Oracle GoldenGate is enhanced to understand the format of the redo log for a CDB. All PDBs in a CDB share the ARCHIVELOG mode of the CDB.
- The control files are common for the whole CDB. The control files are updated to reflect any additional tablespace and data files of plugged PDBs.
- The UNDO tablespace is common for all containers.
- A temporary tablespace common to all containers is required. But each PDB can hold its own temporary tablespace for its own local users.
- Each container has its own data dictionary stored in its proper SYSTEM tablespace, containing its own metadata, and a SYSAUX tablespace.
- The PDBs can create tablespaces within the PDB according to application needs.
- Each datafile is associated with a specific container, named `CON_ID`.

Containers

Two types of containers in V\$CONTAINERS:

- The root container
 - The first **mandatory** container created at CDB creation
 - Oracle system-supplied common objects and metadata
 - Oracle system-supplied common users and roles
- Pluggable database containers (PDBs)
 - A container for an application:
 - Tablespaces (permanent and temporary)
 - Schemas / Objects / Privileges
 - Created / cloned / unplugged / plugged
 - Particular seed PDB\$SEED: fast provisioning of a new PDB
 - Limit of 253 PDBs in a CDB including the seed
 - Limit of 512 services in a CDB



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To summarize, a CDB is an Oracle database that contains the root and eventually several pluggable databases.

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

A non-CDB cannot contain PDBs.

The multitenant architecture enables an Oracle database to contain a portable collection of schemas, schema objects, and nonschema objects that appears to an Oracle Net client as a separate database. For the PDBs to exist and work, the CDB requires a particular type of container, the root container, generated at the creation of the CDB. The root is a system-supplied container that stores common users, which are users that can connect to multiple containers, and system-supplied metadata and data. For example, the source code for system-supplied PL/SQL packages is stored in the root. If the root container exists, you can create containers of the other type, the PDB.

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

A CDB can contain up to 253 PDBs, including the seed, and 252 user-defined PDBs can therefore be created, with IDs ranging from 3 to 254.

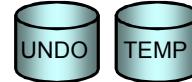
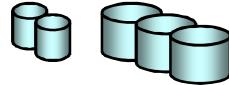
The V\$CONTAINERS view displays all containers including the root.

**CDBA**

Questions: root Versus PDBs

What belongs to the CDB and not to a specific container?

- Control files and redo log files
- UNDO and default TEMP tablespace
- System-supplied metadata
- Shared Oracle-supplied data
 - PL/SQL Oracle supplied packages (DBMS_SQL ...)
 - PDBs service names
- CDB dictionary views providing information across PDBs
- CDB RM plan



NAME	TYPE
TAB\$	2
USER\$	2

NAME
PDB_SALES
PDB_HR

TABLE_NAME	CON_ID
EMPLOYEES	1
TEST	2

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

What belongs to the CDB and not to a specific container?

1. Control files and redo log files

What is in the root that does not exist in PDBs ?

1. Shared UNDO and default database temporary tablespace
2. Oracle-supplied metadata
3. Shared Oracle-supplied data
4. CDB views providing information across PDBs
5. CDB resource manager plan allowing resource management between PDBs within a CDB

Note: A multitenant container database administrator (CDBA) is responsible for administering the CDB instance and the root container.



Questions: PDBs Versus root

What is in a PDB that is not in the root nor in another PDB?

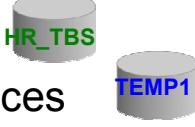
- Application tablespaces 
- Local temporary tablespaces
- Local users  and local roles 
 - Local users connect to the PDB where they exist
- Non-shared local metadata
- Non-shared application data with other PDBs
- PDB RM plan

Table SYS.OBJ\$

NAME	TYPE
EMPLOYEES	2
JOB\$	2

Table HR.EMPLOYEES

EMP_NAME
SMITH
JOHN

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

What is in a PDB that is not in the root?

1. Application tablespaces
2. Local temporary tablespaces
3. Local users and local roles
4. Non-shared local metadata
5. PDB resource manager plan allowing resource management within PDB

Note: A pluggable database administrator (PDBA) is responsible for administering its own PDB.

Terminology

- DBA, CDBA, and PDBA
- Common vs Local:
 - Users
 - Roles
 - Privileges
- CDB vs PDB level:
 - CDB Resource Manager plan vs PDB RM plan
 - Unified audit at CDB or PDB level
 - XStream at CDB or PDB level



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

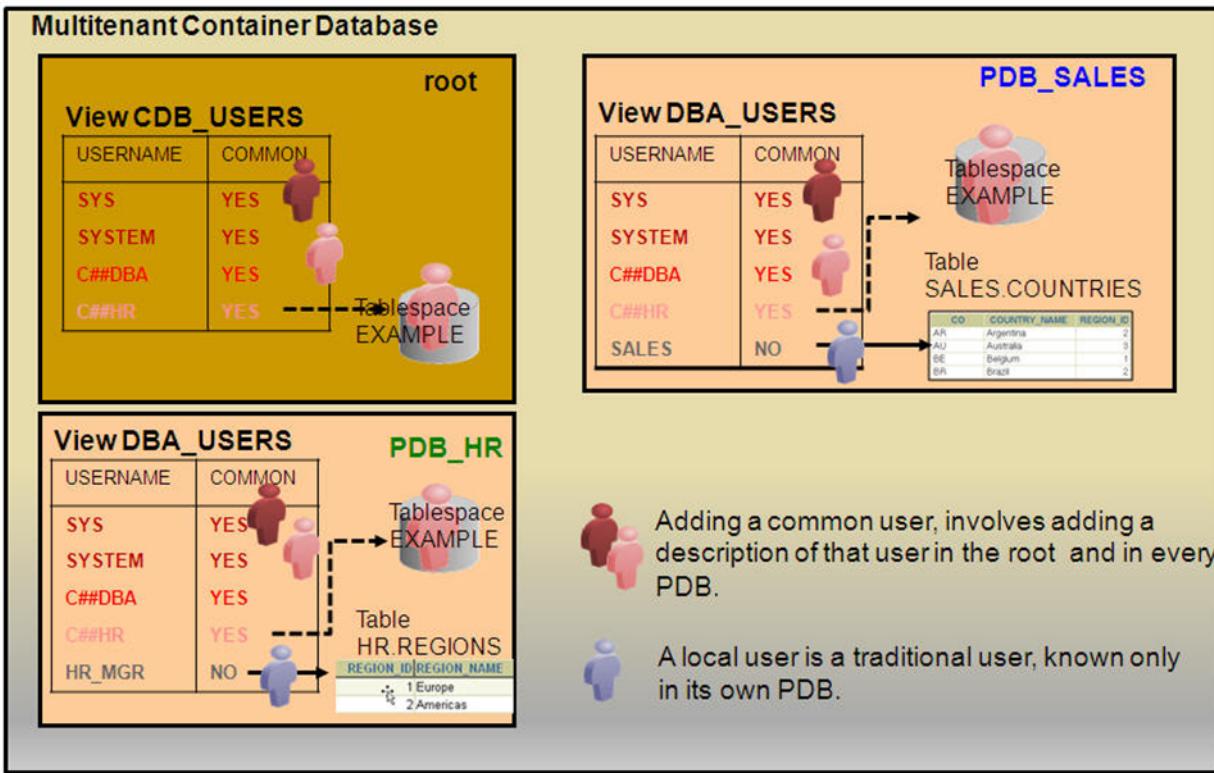
There are three types of database administrators.

- In a non-CDB, the DBA is responsible for all administrative tasks at the database level.
- In a CDB, there are two levels of administration:
 - A CDBA is responsible for administering the CDB instance and the root container.
 - A PDBA is responsible for administering its own PDB.

There is a new terminology for new entities.

- Common users / roles versus local users / roles
- **Common privileges:** Privileges granted “commonly” for all containers versus privileges granted locally within a PDB
- CDB resource management works at CDB level, and PDB resource management at PDB level.
- XStream Out is only available at CDB level and XStream In only at PDB level.
XStream consists of Oracle Database components and application programming interfaces (APIs) that enable client applications to receive data changes from an Oracle database and send data changes to an Oracle database. XStream Out provides Oracle Database components and APIs that enable you to share data changes made to an Oracle database with other systems. XStream In provides Oracle Database components and APIs that enable you to share data changes made to other systems with an Oracle database.

Common and Local Users



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Local User

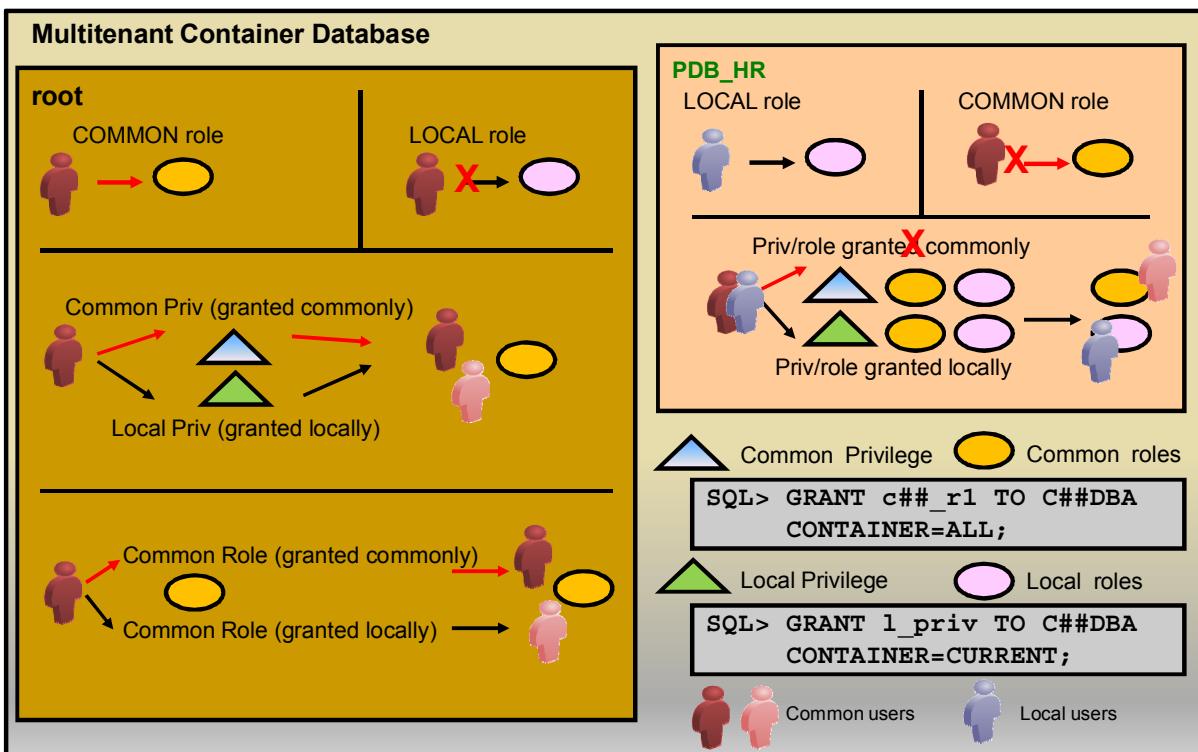
A user in a non-CDB maps to a *local user* in a PDB.

- A local user is defined in the PDB's own data dictionary—and so is not known outside of that PDB.
- A local user can connect only to the PDB where it is defined.
- A local user is specific to a particular PDB and owns a schema in this PDB.
- According to the privileges granted, a user can work on the application data within the PDB or with other PDBs' application using database links. Moreover, there cannot be any local users defined in the root.

Common User

- A *common user* is defined in the root's data dictionary.
- **Only common users can be defined in the root:** Creating a common user allows the CDB administrator to create at once a user that is replicated in each PDB.
- A common user is known, not only where it is defined in the root, but also in every PDB that belongs to the CDB.
- A common user can perform administrative tasks specific to the root or PDBs, such as plugging and unplugging PDBs, start up the CDB, or open a PDB when granted the proper privileges.

Common and Local Privileges and Roles



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Local Roles

A role in a non-CDB maps to a *local role* in a PDB. A local role is defined in the PDB's own data dictionary—and so it is not known outside of that PDB and can only be used within its PDB.

Common Roles

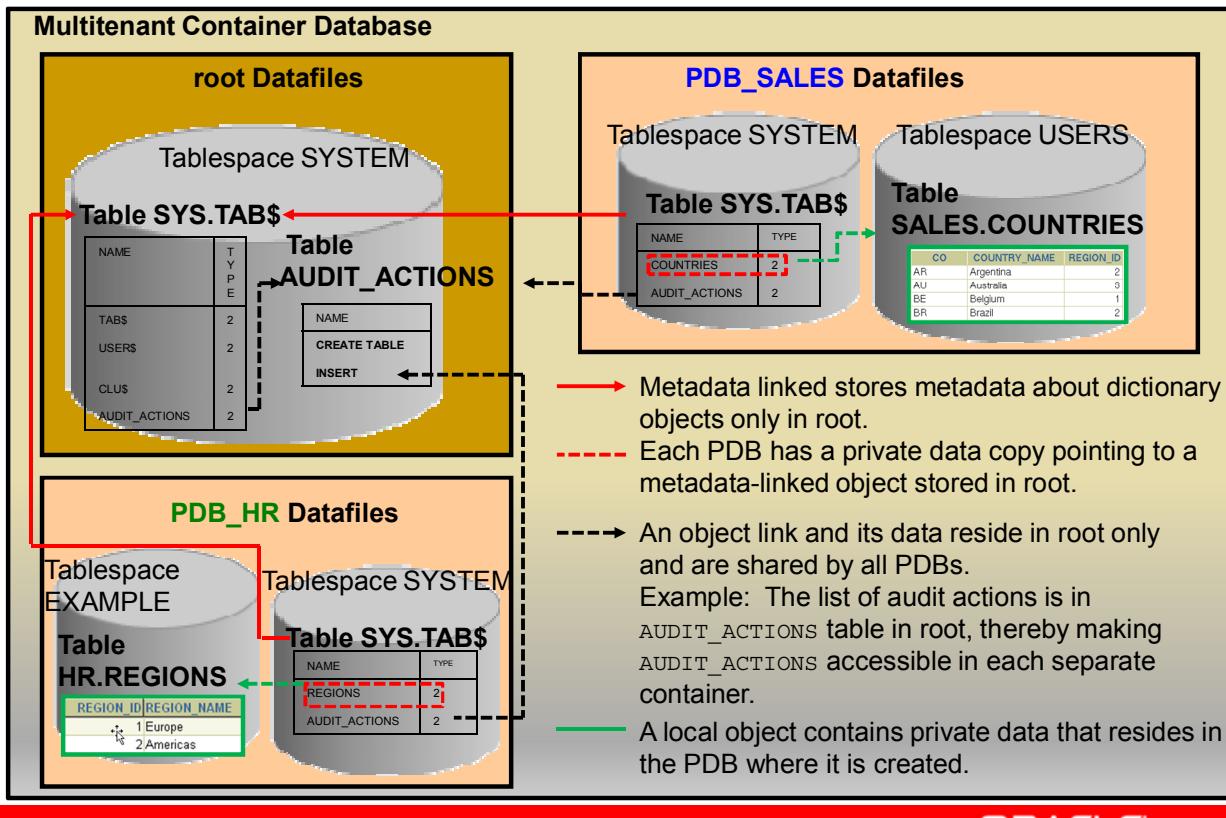
Besides the local role, you can create common roles that are defined in every container. This way, it is easy to create at once a role that is replicated in all PDBs. It is at the creation time that you specify the nature of the role: local or common. Common roles as well as common users can only be created in the root by common users. Moreover, there cannot be any local roles defined in the root. All Oracle-supplied predefined roles are common roles.

Local and Common Privileges

The privileges are commonly referred to as local or common privileges, but to be more precise a privilege is either granted locally with the clause CONTAINER=CURRENT or commonly with the clause CONTAINER=ALL.

The same rule applies to roles: common roles can be granted commonly or locally to common users or roles. Common roles may contain privileges that apply across the CDB, that is, commonly granted for all containers, and can also contain locally granted privileges that apply only to an individual PDB, whereas local roles do not contain any commonly granted privileges.

Shared and Non-shared Objects



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In a non-CDB, all objects are local objects. Common objects are being introduced in 12c to avoid having to store redundant representations of data and metadata across a CDB and to simplify the process of upgrading a CDB. Shared objects exist only in Oracle-supplied schemas.

A shared object may be metadata-linked or object-linked.

- Metadata-linked objects store metadata about dictionary objects only in the root. Each PDB has a private data copy of an object pointing to a metadata link stored in the root.
- An object-linked object and its data reside in the root only and are shared by all PDBs.

Example

The list of audit actions is in the **SYS.AUDIT_ACTIONS** table in the root, thereby making **AUDIT_ACTIONS** accessible in each separate container.

Data Dictionary Views

CDB_XXX All objects in the multitenant container database across all PDBs

DBA_XXX All of the objects in a container or pluggable database

ALL_XXX Objects accessible by the current user

USER_XXX Objects owned by the current user

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

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

DBA dictionary views providing information within PDB:

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For backwards-compatibility, DBA views show the same results in a PDB as in a non-CDB: DBA_OBJECTS shows the objects that exist in the PDB from which you run the query. This implies, in turn, that although the PDB and the root have separate data dictionaries, each data dictionary view in a PDB shows results fetched from both of these data dictionaries. The DBA_XXX views in the root shows, even in a populated CDB, only the Oracle-supplied system—as is seen in a freshly created non-CDB. This is another advantage of the new architecture. To support the duties of the CDB administrator, a new family of data dictionary views is supported with names such as CDB_XXX. Each DBA_XXX view has a CDB_XXX view counterpart with an extra column, *Con_ID*, that shows from which container the listed facts originate. Query the CDB_XXX views from the root and from any PDB. The CDB_XXX views are useful when queried from the root because the results from a particular CDB_XXX view are the union of the results from the DBA_XXX view counterpart over the root and all currently open PDBs. When a CDB_XXX view is queried from a PDB, it shows only the information that it shows in its DBA_XXX view counterpart. If you connect to the root and query CDB_USERS, you get the list of users, common and local, of each container. Now if you query DBA_USERS, you get the list of common users (you are aware that in the root, only common users exist). Now if you connect to a PDB, and query CDB_USERS or DBA_USERS, you get the same list of users, common and local, of the PDB.

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

Impacts

- One character set for all PDBs (Unicode recommended)
- PDB initialization parameters but a single SPFILE
- No PDB qualified database object names
 - ~~SELECT * FROM HR:apps.tab1~~
 - Use DB Links: SELECT * FROM [apps.tab1@HR](#)
- Oracle Data Guard at CDB level
- Oracle Database Vault per PDB only
- One master key per PDB to encrypt PDB data
- Unified audit both at CDB and PDB level
- Oracle Scheduler
- Oracle GoldenGate
- Oracle Streams
- Oracle XStream both at CDB and PDB level



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- There is only one character set and one single spfile for the CDB. The different parameter values of each PDB are stored in a dictionary table.
- There is no PDB-qualified object name. To access an object in another PDB, use a DB link.
- Oracle Data Guard works exactly the same in a CDB as it does in non-CDBs. This feature is not to be available on a per-PDB basis. Opening of a physical standby CDB will always open the root in READ ONLY mode and this also means that no PDB may be open in a mode other than READ ONLY.
- In Oracle Database Vault, each PDB has its own Database Vault metadata. Database Vault constructs, such as realms, are isolated within a PDB.
- Each PDB has its own master key used to encrypt data in the PDB. The master key must be transported from the source database wallet to the target database wallet when a PDB is moved from one host to another. For column encryption, each PDB maintains its own `ENC$` which is not a metadata-linked object.

- A unified audit configuration is visible and enforced across all PDBs. This provides the ability to create audit policies used by all PDBs and audit policies used exclusively for each PDB. An audit configuration that is not enforced across all PDBs means it applies only within a PDB and is not visible outside it. A unified audit framework enables administrators to avoid configuring auditing separately for each container.
- In general, all scheduler objects created by the user can be exported/imported into the PDB using data pump. Predefined scheduler objects will not get exported and that means that any changes made to these objects by the user will have to be made once again after the database has been imported into the pluggable database. However, this is how import/export works currently. A job defined in a PDB will run only if a PDB is open.
- XStream is a programmatic interface to allow a client application access to the changes in the database, known as XStream Outbound Server. XStream Inbound Server allows a client application to feed changes into the database and takes advantage of the apply process available in the database. Oracle GoldenGate is the logical replication and XStream is licensed via the Oracle GoldenGate license. Capturing changes from the database must always be from the root, due to the single unified redo log for the CDB. The XStream outbound can be configured to capture changes from a PDB or the entire CDB. Applying changes via Oracle GoldenGate is done per PDB. An XStream inbound server is configured to apply changes into a specific PDB and performs all of its work within the context of the PDB.
- Oracle Streams continues to be supported in the Oracle Database 12c non-CDBs but is no longer enhanced for new database features or datatypes. Oracle Streams cannot be used with CDBs.

Quiz

Are all assertions below TRUE or FALSE?

- a. Oracle-supplied metadata is in the root container.
- b. There can be several PDBs in one CDB.
- c. The seed PDB is always in read-only state.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: TRUE

Quiz

Which of the following are true?

- a. Only one SYSTEM tablespace per CDB
- b. Only one instance per PDB
- c. A set of redo log files per PDB
- d. Only one UNDO tablespace per CDB
- e. One SYSAUX tablespace per PDB



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: d, e

Quiz

You can create common users in a PDB.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

Summary

In this lesson, you should have learned how to:

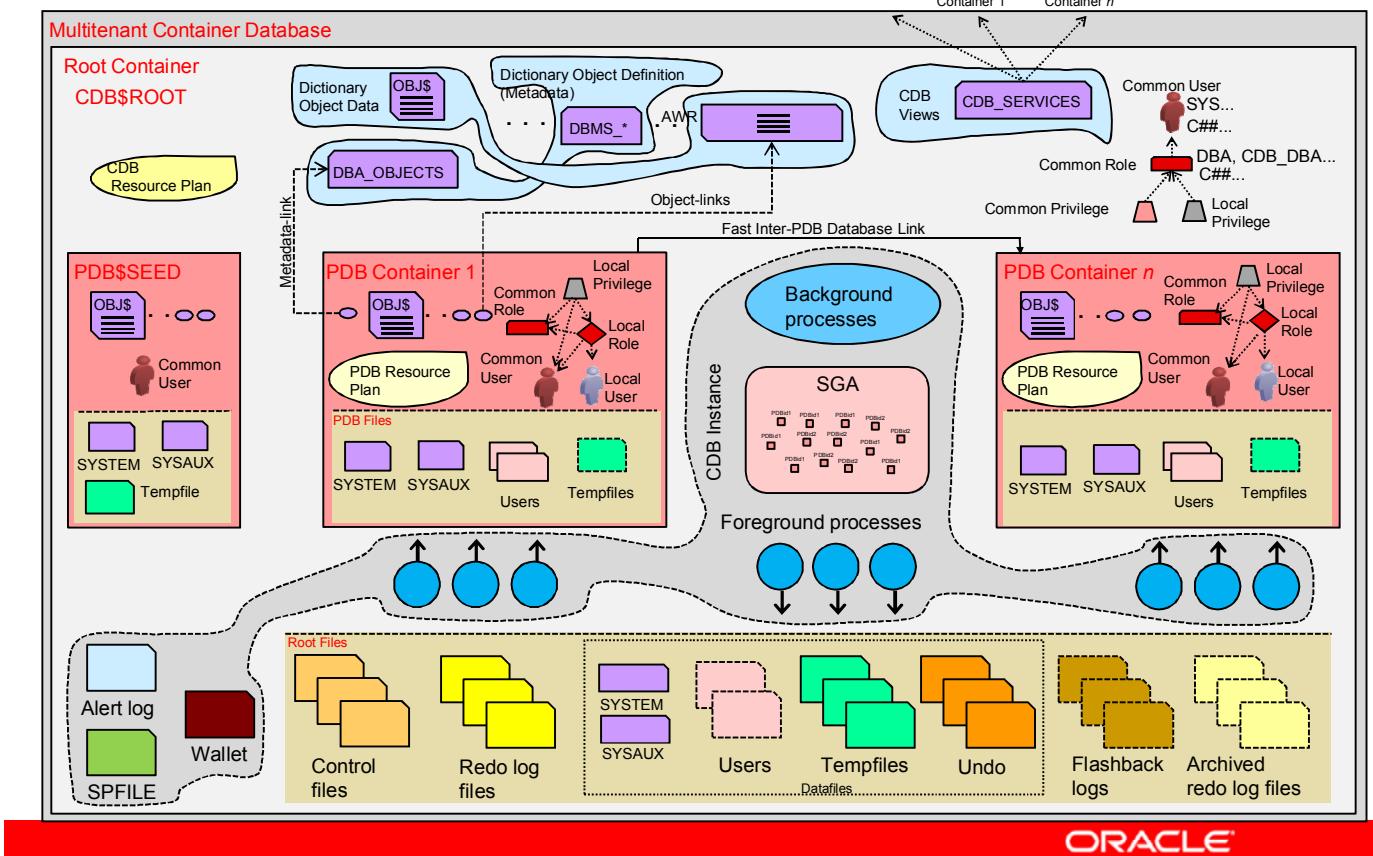
- Describe the multitenant architecture
- Describe the root and pluggable database containers
- Differentiate the root from a pluggable database
- Explain pluggable database plugging
- List impacts in various areas



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Multitenant Architecture

General Architecture Poster



Practice 2 Overview: Exploring a Multitenant Container Database

These practices cover the following topics:

- Exploring the CDB processes, files
- Displaying CDB_xxx views



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only



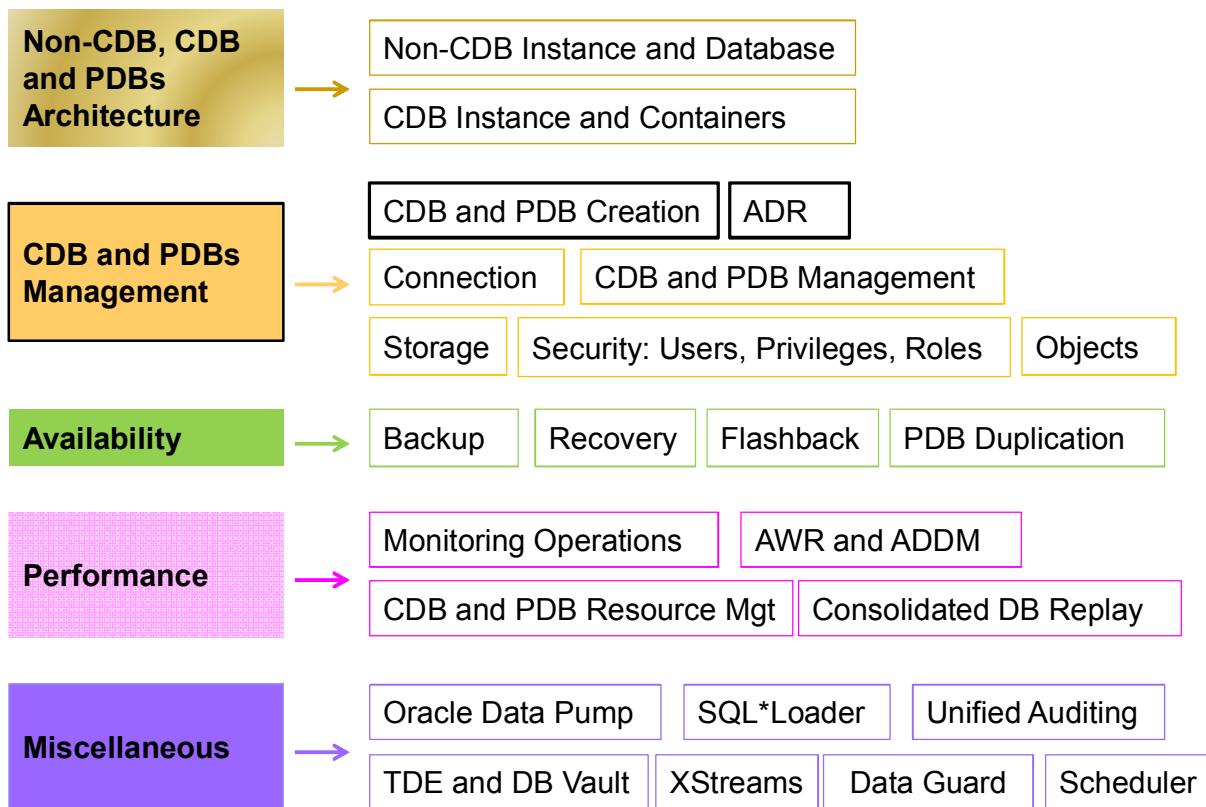
Creating a Multitenant Container Database and Pluggable Databases



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Course Structure



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This lesson explains how to:

- Create a CDB
- Create PDBs using different methods
- Migrate non-CDB to CDB

Objectives

After completing this lesson, you should be able to:

- Configure and create a CDB
- Create a PDB from PDB\$SEED
- Create a PDB from a non-CDB
- Clone a PDB into the same CDB
- Unplug and plug a PDB from one CDB to another or the same CDB
- Explore the instance
- Explore the structure of PDBs
- Explore the ADR
- Drop a PDB
- Migrate pre-12.1 and 12.1 non-CDB to CDB



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

- *Oracle Database Administrator's Guide 12c Release 1 (12.1)*
- *Oracle Database PL/SQL Packages and Types Reference 12c Release 1 (12.1) – "DBMS_PDB" chapter*

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

- *Oracle Database 12c New Features Demo Series* demonstrations:
 - *How to Create and Drop CDBs*
 - *How to Create and Drop PDBs: Create a PDB from seed*
 - *How to Create and Drop PDBs: Clone a PDB from another PDB*
 - *How to Create and Drop PDBs: Plug non-CDB into a CDB as a PDB*
 - *How to Create and Drop PDBs: Unplug a PDB and plug the PDB in another CDB*
 - *How to Create and Drop PDBs: Drop a PDB*
- *Oracle By Example (OBE)*:
 - *Unplugging and Plugging Pluggable Databases*
 - *Cloning Pluggable Databases*
 - *Plugging a non-CDB as a Pluggable Database into a Multitenant Container Database*

Goals

Create a multitenant container database:

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



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Why create multitenant container databases?

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

Tools

	SQL*Plus	OUI	DBCA	EM Cloud Control	SQL Developer	DBUA
Create a new CDB or PDB	Yes	Yes	Yes	Yes (PDB only)	Yes (PDB only)	
Explore CDB instance, architecture, PDBs	Yes			Yes	Yes	
Upgrade a 12.1 CDB to 12.x CDB	Yes					Yes



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

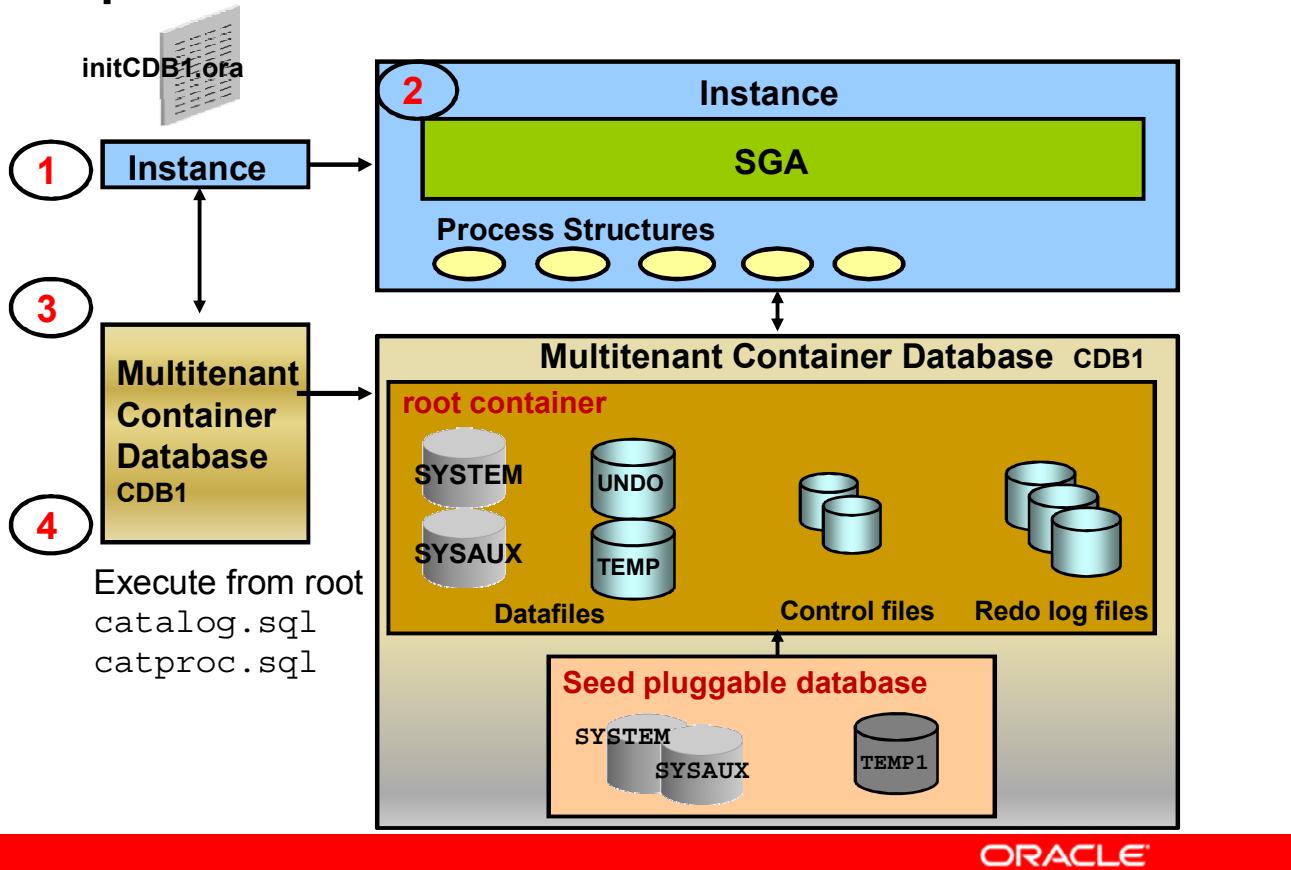
There are different tools to create and upgrade multitenant container databases. As shown in the slide, you can create a new CDB or new PDBs using either SQL*Plus or Database Configuration Assistant (DBCA) or during the installation of Oracle Database 12c. SQL Developer and Enterprise Manager Cloud Control allow you to create pluggable databases.

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

You can upgrade a 12.1 CDB to a 12.x CDB with Database Upgrade Assistant (DBUA).

Note: Enterprise Manager Database Express does not provide the capability to create a CDB nor a PDB nor explore PDBs architecture.

Steps to Create a Multitenant Container Database



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

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

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

Create a Multitenant Container Database: Using SQL*Plus

1. Instance startup:

- Set ORACLE_SID=CDB1.
- Set in initCDB1.ora:
 - Set CONTROL_FILES to CDB control file names.
 - Set DB_NAME to a CDB name.
 - Set ENABLE_PLUGGABLE_DATABASE to TRUE.

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

2. Create the database:

```
SQL> CREATE DATABASE CDB1 ENABLE PLUGGABLE DATABASE ...
2 SEED FILE_NAME_CONVERT ('/oracle/dbs','/oracle/seed');
```

- CDB\$ROOT container
- PDB\$SEED pluggable database

3. Close/open the seed PDB and run postcreation scripts.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The detailed steps to create a new CDB using SQL*Plus are the following:

- Before starting the instance, prepare an `init<SID>.ora` parameter file with the usual parameters: `DB_NAME`, `CONTROL_FILES` if OMF is not used, and `DB_BLOCK_SIZE`. The global database name of the root is the global database name of the CDB. A new parameter is required to define that the instance started is ready for a CBD and not a non-CDB creation. The `ENABLE_PLUGGABLE_DATABASE` parameter must be set to `TRUE`.
 Set the `ORACLE_SID` environment variable. Launch SQL*Plus, connect as an OS authenticated user belonging to the DBA OS group, and execute the `STARTUP NOMOUNT` command.
 If you are using Oracle ASM storage to manage your disk storage, then you must start the Oracle ASM instance and configure your disk groups before performing the next steps.
- Create the CDB using the `CREATE DATABASE` command with the new clause `ENABLE PLUGGABLE DATABASE`. The clause specifies that the database is a CDB and not a non-CDB. This creates the root container and the seed pluggable database. You may use another clause `SEED FILE_NAME_CONVERT` to specify the location of the seed's files. If you omit the clause, OMF determines the names and locations of the seed's files. The `FILE_NAME_CONVERT` specifies the source directory of the root datafiles copied to the target seed directory.

Omit the clause `SEED FILE_NAME_CONVERT` if you use the new `init.ora` parameter `PDB_FILE_NAME_CONVERT`, mapping names of the root datafiles to the seed datafiles. In the example, the `/oracle/dbs` and `/oracle/seed` directories must exist. The character set defined in the statement is still the single one for the CDB.

3. Run postcreation scripts:

- Set the session with a new parameter:

```
alter session set "_oracle_script"=true;
```

- Close and open the seed PDB:

```
alter pluggable database pdb$seed close;
```

```
alter pluggable database pdb$seed open;
```

- Execute `catalog.sql` and other postcreation scripts.

```
?/rdbms/admin/catalog.sql
```

```
?/rdbms/admin/catblock.sql
```

```
?/rdbms/admin/catproc.sql
```

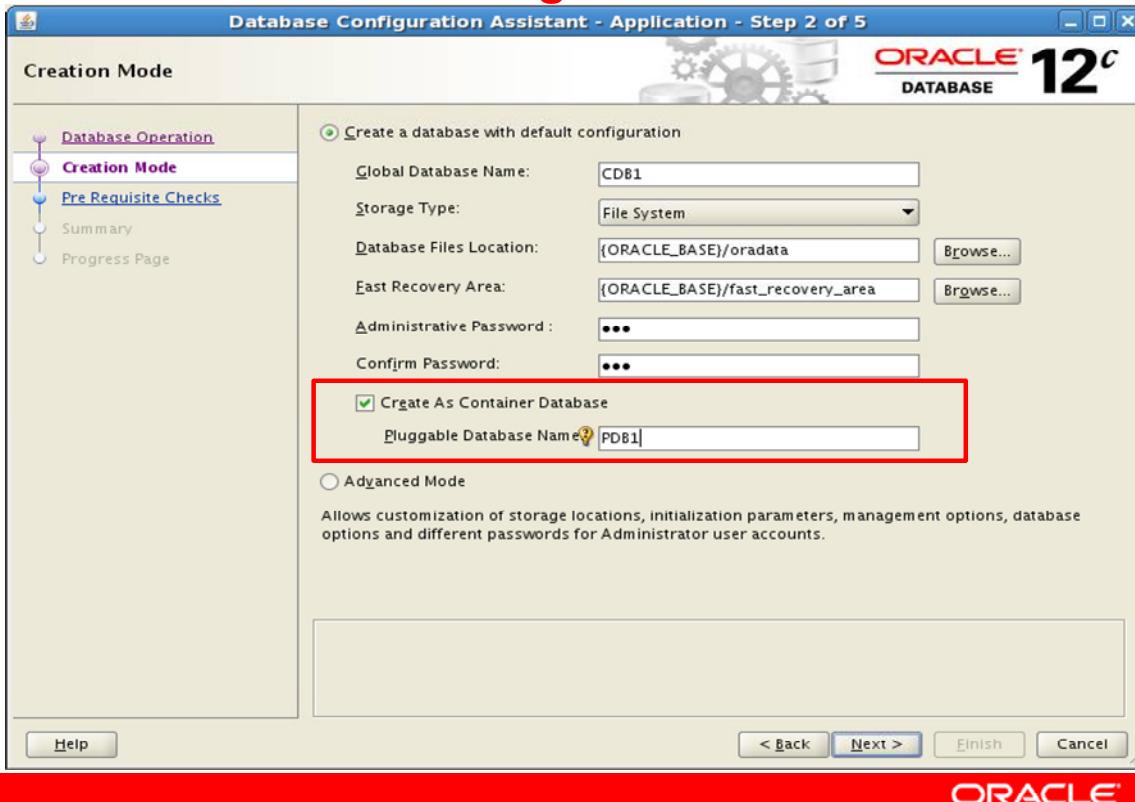
```
?/rdbms/admin/catoctk.sql
```

```
?/rdbms/admin/owminst.plb
```

```
?/sqlplus/admin/pupbld.sql
```

Note: To get the full list of scripts executed and the sequence of execution to follow, run DBCA to create CDB and in the last step, generate the scripts only. The scripts are created in the `$ORACLE_BASE/admin/<cdb_name>/scripts` directory. The shell script `<cdb_name>.sh` is the first script to read.

Create a Multitenant Container Database: Using DBCA



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The screenshot in the slide shows that creating the database with DBCA requires that the “Create As Container Database” check box be selected, else the database is created as a non-CDB.

You also have to provide a pluggable database name when using the “Create a database with default configuration” option. If you select the “Advanced Mode” option, you can create an empty multitenant container database with only the root and seed containers.

In Advanced Mode, you can register the CDB with Enterprise Manager Cloud Control and or configure the CDB for Enterprise Manager Database Express, and set passwords for SYS and SYSTEM users.

New Clause: **SEED FILE_NAME_CONVERT**

CREATE DATABASE new clauses:

```
SQL> CREATE DATABASE cdb1
  2  USER SYS IDENTIFIED BY p1 USER SYSTEM IDENTIFIED BY p2
  3  LOGFILE GROUP 1 ('/u01/app/oradata/CDB1/redo1a.log',
  4                      '/u02/app/oradata/CDB1/redo1b.log') SIZE 100M,
  5          GROUP 2 ('/u01/app/oradata/CDB1/redo2a.log',
  6                      '/u02/app/oradata/CDB1/redo2b.log') SIZE 100M
  7  CHARACTER SET AL32UTF8 NATIONAL CHARACTER SET AL16UTF16
  8  EXTENT MANAGEMENT LOCAL DATAFILE
  9          '/u01/app/oradata/CDB1/system01.dbf' SIZE 325M
10  SYSAUX DATAFILE  '/u01/app/oradata/CDB1/sysaux01.dbf' SIZE 325M
11  DEFAULT TEMPORARY TABLESPACE tempst1
12          TEMPFILE '/u01/app/oradata/CDB1/temp01.dbf' SIZE 20M
13  UNDO TABLESPACE undotbs
14          DATAFILE '/u01/app/oradata/CDB1/undotbs01.dbf' SIZE 200M
15  ENABLE PLUGGABLE DATABASE
16  SEED   FILE_NAME_CONVERT =
17          ('/u01/app/oradata/CDB1',
18          '/u01/app/oradata/CDB1/seed') ;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

New Clause : **ENABLE PLUGGABLE DATABASE**

Without **SEED FILE_NAME_CONVERT**:

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

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

SQL> CREATE DATABASE cdb2
  2  USER SYS IDENTIFIED BY p1 USER SYSTEM IDENTIFIED BY p2
  3  EXTENT MANAGEMENT LOCAL
  4  DEFAULT TEMPORARY TABLESPACE temp
  5  UNDO TABLESPACE undotbs
  6  DEFAULT TABLESPACE users
  7  ENABLE PLUGGABLE DATABASE;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle Managed Files

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

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

PDB_FILE_NAME_CONVERT Instance Parameter

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

- The **PDB_FILE_NAME_CONVERT** instance parameter maps names of existing files (the root datafiles in your case) to new file names (the seed datafiles in this case).
In the example, both **/u01/app/oradata/CDB1** and **/u01/app/oradata/seed** directories must exist.

After CDB Creation: What's New in CDB

A CDB has new characteristics compared to non-CDBs:

- **Two containers:**
 - The **root** (CDB\$ROOT)
 - The **seed** PDB (PDB\$SEED)
- **Several services:** One per container
 - Name of root service = name of the CDB (cdb1)
 - Maximum number of services: 512
 - Max nb of services per PDB<= max nb of services in CDB
- **Common users** in root and seed: SYS , SYSTEM ...
- **Common privileges** granted to common users
- **Predefined common roles**
- Tablespaces and datafiles associated with each container:
 - **root**: SYSTEM and SYSAUX
 - **seed**: SYSTEM and SYSAUX



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

- Two containers: the root and the seed PDB (Maximum number of PDBs: 252 plus seed)
- As many services as containers: the service name for the root container is the CDB name given at the CDB creation concatenated with the domain name. Each new PDB is assigned a service name: the service name is the PDB name given at PDB creation concatenated with domain name. If you create or plug a PDBtest PDB, its service name would be PDBtest concatenated with the domain name. You can find all service names maintained in a CDB in the CDB_SERVICES view. To connect to the CDB, you connect to the root, using either local OS authentication or the root service name. For example, if you set the ORACLE_SID to the CDB instance name and use CONNECT / AS SYSDBA, you are connected to the root under the common SYS user granted system privileges to manage and maintain all PDBs. To connect to a desired PDB, use either easyconnect or the tnsnames.ora file.
For example, CONNECT username/password@net_service_name .
- Common users: SYS, SYSTEM, created in all containers, the root, and the seed
- Common privileges granted to all users in all containers
- Predefined common roles in all containers, the root, and the seed
- Tablespaces: SYSTEM, SYSAUX associated with each container (maximum number of datafiles: 65534)

Data Dictionary Views: **DBA_xxx**

DBA_xxx All objects in the root or a pluggable database

ALL_xxx Objects accessible by the current user in a PDB

USER_xxx Objects owned by the current user in a PDB

DBA dictionary views providing information within PDB:

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

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



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

Examples of DBA views:

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

Data Dictionary Views: CDB_*

CDB_*

All objects in the CDB (new column CON_ID)

DBA_*

All objects in the root or a pluggable database

ALL_*

Objects accessible by the current user in a PDB

USER_*

Objects owned by the current user in a PDB

CDB dictionary views provide information across PDBs:

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

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

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

Examples of CDB views:

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

Data Dictionary Views: Examples

- Comparisons:

1

```
SQL> CONNECT / AS SYSDBA
SQL> SELECT role, common, con_id FROM cdb_roles;
```

2

```
SQL> SELECT role, common FROM dba_roles;
```

3

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

4

```
SQL> SELECT role, common FROM dba_roles;
```

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

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

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

Data Dictionary Views: V\$xxx Views

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

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

STATUS	CON_ID	
cr	1	→ root
free	1	
xcur	1	
xcur	2	→ seed PDB
cr	3	→ PDB1 PDB
xcur	3	

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

OBJECT_ID	ORACLE_USERNAME	LOCKED_MODE	CON_ID	
83711	SYS	3	3	← PDB1 PDB
83710	DOM	3	4	← PDB2 PDB

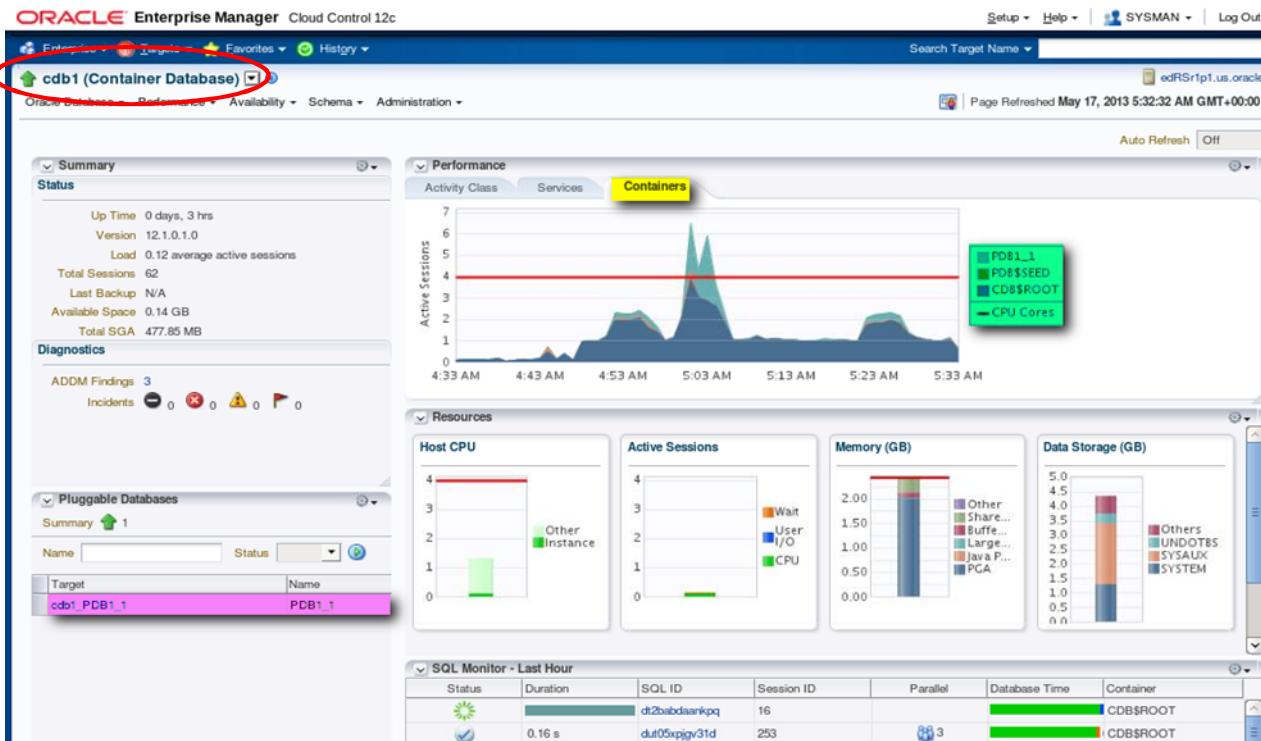
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

EM Cloud Control: Summary



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The screenshot in the slide shows that Enterprise Manager Cloud Control provides information related to CDBs and PDBs.

Moreover, the user interface (UI) allows you to perform any database administrative operation on CDBs and PDBs, such as creating users, granting privileges, backing up datafiles, creating tablespaces, SQL monitoring and tuning and so on.

In the example in the slide, the Pluggable Databases page shows on the right side of the window all PDBs of the `cdb1` CDB except the seed. On the left side of the window, the Performance page shows the Active Sessions of all containers.

EM Cloud Control: Storage Example

The screenshot illustrates two views of the Tablespace management interface in Oracle Enterprise Manager Cloud Control.

Left View (Container Database Scope):

- The top navigation bar shows "cdb1 (Container Database)".
- The main title is "Tablespaces".
- A yellow box highlights the "root and PDBs tablespaces" section, which lists tablespaces from multiple containers (CDB\$ROOT, PDB1_1).
- A yellow arrow points from this section to the right-hand view.

Right View (PDB Scope):

- The top navigation bar shows "cdb1 / PDB1_1".
- The main title is "Tablespaces".
- A yellow box highlights "PDB tablespaces", listing tablespaces specific to the PDB1_1 container.
- A yellow arrow points from the left view's "root and PDBs tablespaces" section to this list.
- A callout box states: "Scope limited to a specific PDB".

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the first example of the screenshot on the left side of the slide, the Tablespace page displays on the top of the page the Space Usage Summary of all containers of the cdb1 CDB and on the bottom of the page the list of all tablespaces of all containers of the CDB. In the second example of the screenshot on the right side of the slide, because the scope was limited to a PDB, the Tablespace page displays only the list of all tablespaces of one PDB of the CDB.

EM Cloud Control: Storage Example

Tablespaces

Search
Select an object type and optionally enter an object name to filter
Container _____ Object Name _____ Go

By default, the search returns all uppercase matches beginning with the string (%) in a double quoted string.

Select	Container	Name	Available Space Used(%)	Alerts	Available Space Used(%)
<input checked="" type="radio"/>	CDB\$ROOT	SYSAUX	1.0	✓	90.2 YES
<input type="radio"/>	CDB\$ROOT	SY	2.1	✓	94.8 YES
<input type="radio"/>	CDB\$ROOT	SY	0.8	✓	96.5 YES
<input type="radio"/>	CDB\$ROOT	TEMP	0.0	✓	2.3 YES
<input type="radio"/>	CDB\$ROOT	UNPLUGGED1	0.0	✓	26.2 YES
<input type="radio"/>	CDB\$ROOT	USERS	0.0	✓	26.2 YES
<input type="radio"/>	CDB\$ROOT	USERS	0.0	✓	26.2 YES
<input type="radio"/>	PDB1_1	EXAMPLE	1.0	✓	90.2 YES
<input type="radio"/>	PDB1_1	SYSAUX	2.1	✓	94.8 YES

Tablespaces

Search
Select an object type and optionally enter an object name to filter the data that is displayed in your result
Object Name _____ Go

By default, the search returns all uppercase matches beginning with the string you entered. To run an exact or case-sensitive (%) in a double quoted string.

Selection Mode Single

Select	Name	Available Space Used(%)	Alerts	Allocated Space Used(%)	Auto Extend
<input checked="" type="radio"/>	EXAMPLE	1.0	✓	90.2 YES	YES
<input type="radio"/>	SYSAUX	2.1	✓	94.8 YES	YES
<input type="radio"/>	SYSTEM	0.8	✓	96.5 YES	YES
<input type="radio"/>	TEMP	0.0	✓	2.3 YES	YES
<input type="radio"/>	USERS	0.0	✓	26.2 YES	YES

PDB tablespaces

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the first example in the screenshot on the left side of the slide, the Tablespaces page displays the list of all tablespaces of all containers of the CDB except the tablespaces of the seed. In the second example of the screenshot of the right side of the slide, because the scope was limited to a PDB, the Tablespaces page displays only the list of all tablespaces of one PDB of the CDB.

After CDB Creation: To do List

After CDB creation, the CDBA has to:

- Set a default tablespace for the root
- Set a default temporary tablespace for the entire CDB
- Start the listener
- Plug non-CDBs
- Test startup/shutdown procedures
- Create new event triggers to automate PDBs opening
- Create backup and recovery procedures

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

- Set a default tablespace
- Optionally create additional temporary tablespaces



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

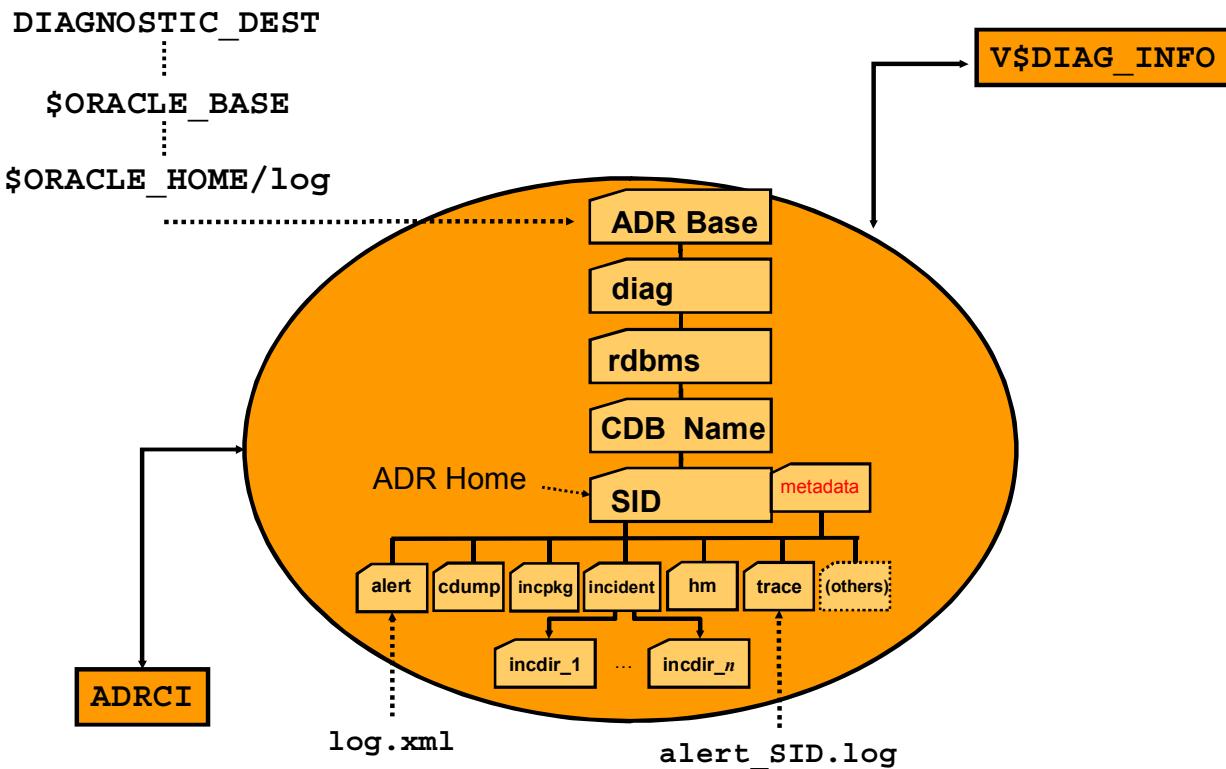
After the CDB is created, the container database administrator (CDBA) has to complete administrative tasks:

- Set a separate default tablespace for the root. In the root, there should not be any user data.
- Set a default temporary tablespace for the entire CDB.
- Start the listener.
- Plug non-CDBs if the initial plan was to consolidate several non-CDBs into a single one.
- Test startup and shutdown procedures.
- Create new event triggers to automate PDBs opening.
- Create backup and recovery procedures.

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

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

Automatic Diagnostic Repository



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

Automatic Diagnostic Repository: **alert.log** File

The `alert_CDB1.log` shows new DDL statements.

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

CREATE PLUGGABLE DATABASE pdb1 ... ;
ALTER PLUGGABLE DATABASE pdb1 UNPLUG INTO ... ;
ALTER PLUGGABLE DATABASE ALL OPEN ;
ALTER PLUGGABLE DATABASE pdb2 CLOSE IMMEDIATE ;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

Quiz

The seed pluggable database of a CDB owns several characteristics.

- a. It is always kept in READ ONLY mode.
- b. It is not a container.
- c. The seed can be dropped.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a

Quiz

You create a CDB. What is true about the seed pluggable database?

- a. Copy the seed datafiles yourself.
- b. Use the new clause `SEED FILE_NAME_CONVERT` in the `CREATE DATABASE` statement.
- c. The seed pluggable database is not required.
- d. The seed pluggable database does not require datafiles.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

Practice 3 Overview: Creating a CDB and PDBs

This practice covers creating a CDB with no PDBs.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Creating a CDB with no PDB with DBCA

Provisioning New Pluggable Databases

Four methods:

- Create a new PDB from the seed PDB
- Plug a non-CDB in a CDB
- Clone a PDB from another PDB:
 - Into the same CDB
- Plug an unplugged PDB into the same CDB or to another CDB



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

- Create a new PDB from the seed, the PDB\$SEED PDB, for example for a new application implementation. This type of PDB creation is nearly instantaneous.
- Plug non-CDBs in a CDB as PDBs, as part of the migration strategy. It is also a good way to consolidate several non-CDBs into a CDB.
- Clone a PDB from another PDB of the same CDB. For example, you want to test the application patch of your production. You first clone your production application in a cloned PDB and patch the cloned PDB to test.
- Plug an unplugged PDB into another CDB or into the same CDB. For example, you have to upgrade a PDB to the latest Oracle version, but you do not want to apply it on all PDBs. Instead of upgrading a CDB from one release to another, you can unplug a PDB from one Oracle Database release, and then plug it into a newly created CDB from a higher release. In case you unplugged a PDB inappropriately, you can still replug it into the same CDB.

Tools

To provision new PDBs, you can use:

- SQL*Plus
- SQL Developer
- Enterprise Manager Cloud Control
- DBCA
 - Copy from seed
 - Plugging from supplied templates (schema templates)
 - By unplugging / plugging method



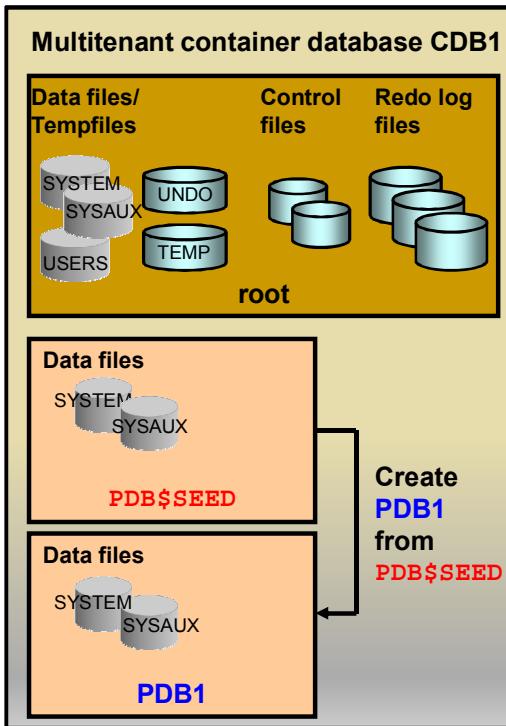
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

To create a new PDB from seed or by unplug/plug method, you can also use DBCA.

Method 1: Create New PDB from PDB\$SEED



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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

A new default service is also created for the PDB.

Steps: With FILE_NAME_CONVERT

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

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

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

2. Use views to verify:

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

Note: The STATUS of the PDB is NEW.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

Steps: Without FILE_NAME_CONVERT

Create a new PDB from seed without FILE_NAME_CONVERT:

- OMF: DB_CREATE_FILE_DEST = '/u01/app/oradata/CDB1/pdb1'

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

Or

- New parameter: PDB_FILE_NAME_CONVERT = '/u01/app/oradata/CDB1/seed','/u01/app/oradata/CDB1/pdb'

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



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If you use OMF or PDB_FILE_NAME_CONVERT, then first connect to the root of the CDB as a SYS.

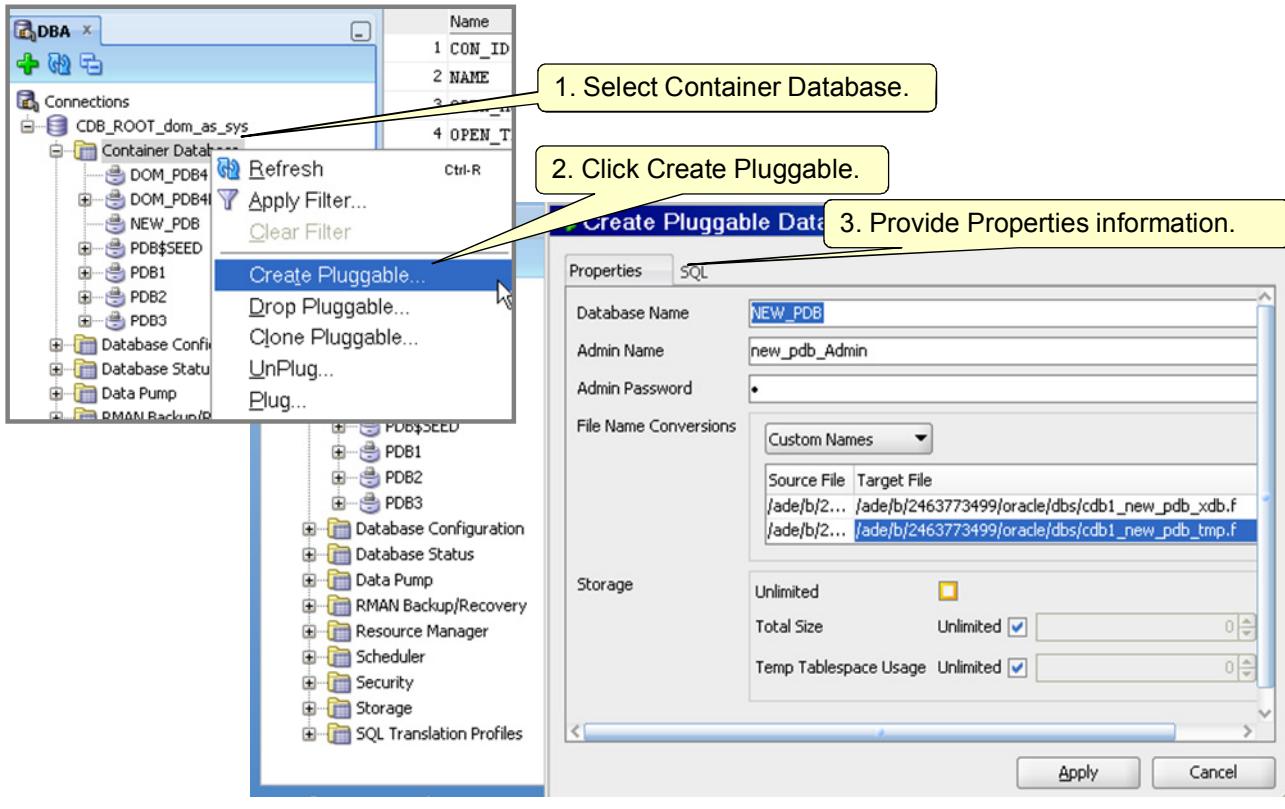
- With OMF, set in init.ora DB_CREATE_FILE_DEST instance parameter to a target directory for the datafiles of the new PDB.
- Without OMF, set the PDB_FILE_NAME_CONVERT new instance parameter to both the source directory of the seed datafiles and the target directory for the new PDB datafiles.

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

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

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

Method 1: Using SQL Developer



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use SQL*Developer to perform the same type of operations, such as create a new PDB from the seed, clone a PDB from another PDB from the CDB, or from another CDB.

First of all, click to view the CDB and its PDBs, click View option from the top menu then DBA.

If you want to create a new PDB from the seed, then select Container Database, then choose the Create Pluggable option, then provide the PDB name, the PDB administrator username and the target files location in the File Name Conversions attribute. Use Custom Names to provide the Target File location. The Source File displays the list of seed datafiles used as templates during the copy.

Method 1: Using SQL Developer

4. View SQL statement.

```
CREATE PLUGGABLE DATABASE NEW_PDB ADMIN USER new_pdb_Admin IDENTIFIED BY p
FILE_NAME_CONVERT=(
  '/ade/b/2463773499/oracle/dbs/cdbl_pdb0_db.f', '/ade/b/2463773499/oracle/dbs/cdbl_new_pdb_db.f',
  '/ade/b/2463773499/oracle/dbs/cdbl_pdb0_ax.f', '/ade/b/2463773499/oracle/dbs/cdbl_new_pdb_ax.f',
  '/ade/b/2463773499/oracle/dbs/cdbl_pdb0_xdb.f', '/ade/b/2463773499/oracle/dbs/cdbl_new_pdb_xdb.f',
  '/ade/b/2463773499/oracle/dbs/cdbl_pdb0_tmp.f', '/ade/b/2463773499/oracle/dbs/cdbl_new_pdb_tmp.f'
)
STORAGE UNLIMITED
```

5. After PDB is created:

- CON_ID defines the new container identifier in the CDB.
- OPEN_MODE defines the open status, by default MOUNTED .
- GUID defines the new global unique container identifier.

Name	Value
1 CON_ID	8
2 NAME	NEW_PDB
3 OPEN_MODE	MOUNTED
4 OPEN_TIME	05-JAN-12 09.03.39.600000000 AM
5 DBID	1221202379
6 CON_UID	84955
7 GUID	B5CBE804C06B46FBE0431C72E50A5FB4
8 CREATE_SCN	1669072
9 OPEN_TIME	05-JAN-12 09.03.39.600000000 AM

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

Before applying the creation, you can view the SQL statement.

After it is created, the PDB is assigned a CON_ID number, unique within the CDB, and a GUID global unique identifier.

Synchronization

- Customer-created common users or roles in root:
 - Cannot be created, modified, dropped when PDB is in READ-ONLY mode
 - Can be created, modified, dropped when PDB is in MOUNTED mode
- When opening the PDB:
 - In READ-ONLY mode, an error is returned
 - In READ-WRITE mode, synchronization with the target CDB is automatically completed
 - A compatibility check is automatically performed:
 - Any violation is reported in the PDB_PLUG_IN_VIOLATIONS view.
 - If there are no violation, the PDB status is changed to NORMAL.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

After a PDB is created using seed PDB or plugging or cloning methods, or even closed, you can view the status of the new or closed PDB by querying the STATUS column of the CDB_PDBS view.

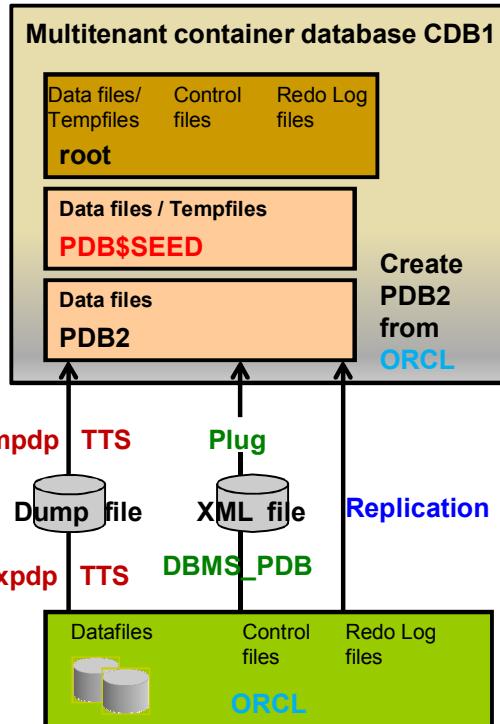
If common users and roles had been previously created, the new or closed PDB must be synchronized to retrieve the new common users and roles from the root. The synchronization is automatically performed if the PDB is opened in read write mode.

If you open the PDB in read-only mode, an error is returned.

When a PDB is opened, Oracle Database checks the compatibility of the PDB with the CDB.

Each compatibility violation is either a warning or an error. If a compatibility violation is a warning, then the warning is recorded in the alert log, but the PDB is opened normally without displaying a warning message. If a compatibility violation is an error, then an error message is displayed when the PDB is opened, and the error is recorded in the alert log. You must correct the condition that caused each error. When there are errors, access to the PDB is limited to users with the 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.

Method 2: Plug a Non-CDB into CDB



Three possible methods:

- TTS or TDB or full export/import
- XML file definition with DBMS_PDB
- Replication

Entities are created in the new PDB:

- Tablespaces: SYSTEM, SYSAUX
- A full catalog
- Common users: SYS, SYSTEM
- A local administrator (PDBA)
- A new default service

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

There are three possible methods to plug a non-CDB database into a CDB.

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

It is appropriate to use Oracle Data Pump when:

- Both source and target databases are different endian
- The source character set is not equal to the target character set, and is not a binary subset of the target

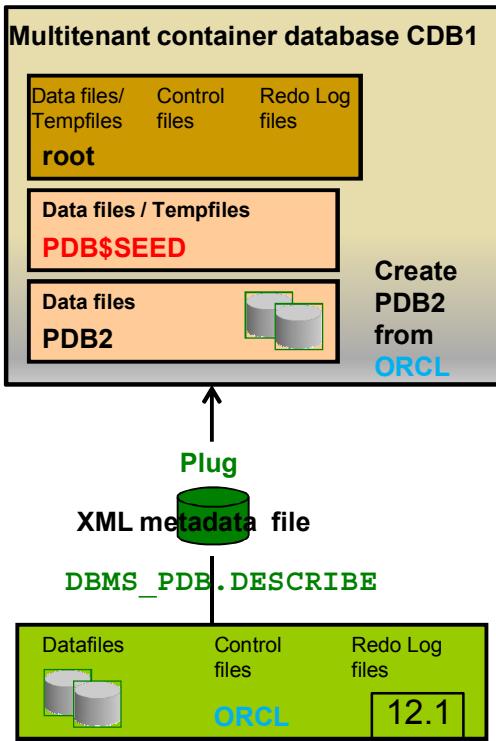
Either use transportable tablespace (TTS) or full conventional export / import or full transportable database (TDB) provided that in the last one any user-defined object resides in a single user-defined tablespace. Data Pump full transportable database does not support movement of XDB or AWR repositories. Only user-generated XML schemas are moved.

In other cases, using the DBMS_PDB package is the easiest option. The DBMS_PDB package constructs an XML file describing the non-CDB data files to plug the non-CDB into the CDB as a PDB. This method presupposes that the non-CDB is an Oracle 12c database.

If the DBMS_PDB package is not used, export/import is usually simpler than using GoldenGate replication, but export/import might require more down time during the switch from the non-CDB to the PDB.

To understand the steps of the first method, refer to the lesson titled “Miscellaneous.”

Plug a Non-CDB into CDB Using DBMS_PDB



1. Open `ORCL` in READ ONLY mode
2.

```
SQL> EXEC DBMS_PDB.DESCRIBE
  ('/tmp/ORCL.xml')
```
3. Connect to the target cdb as a common user with CREATE PLUGGABLE DATABASE privilege
4. Plug in the unplugged PDB `ORCL` as `PDB2`

```
SQL> CREATE PLUGGABLE DATABASE
  2 PDB2 USING '/tmp/ORCL.xml';
```
5. Run the `noncdb_to_pdb.sql` script


```
SQL> CONNECT sys@PDB2 AS SYSDBA
SQL> @$ORACLE_HOME/rdbms/admin/noncdb_to_pdb
```
6. Open PDB2

Note: The STATUS of the PDB is CONVERTING.

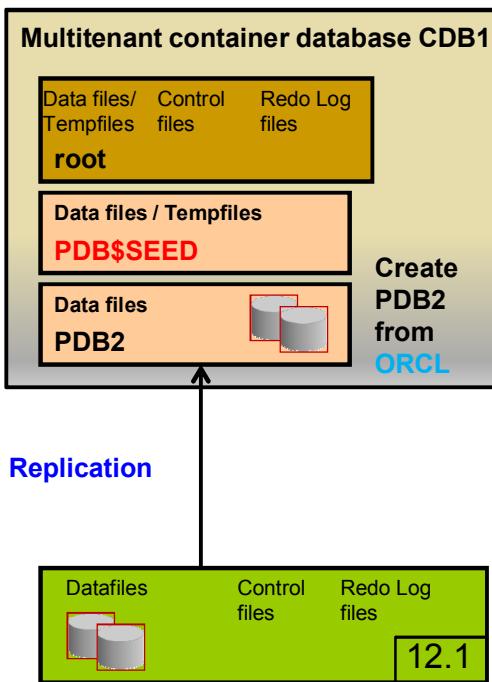
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The technique with DBMS_PDB package creates an unplugged PDB from an Oracle database 12c non-CDB. The unplugged PDB can then be plugged into a CDB as a new PDB. Running the DBMS_PDB.DESCRIBE procedure on the non-CDB generates an XML file that describes the future PDB. You can plug in the unplugged PDB in the same way that you can plug in any unplugged PDB, using the XML file and the non-CDB datafiles. The steps are the following:

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

Plug a Non-CDB into CDB Using Replication



Method using replication:

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

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

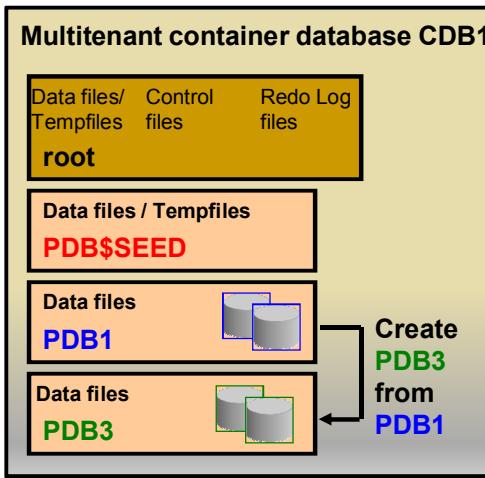
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

Method 3: Clone PDBs



PDB3 owns:

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

1. In init.ora, set `DB_CREATE_FILE_DEST= 'PDB3dir'` or `PDB_FILE_NAME_CONVERT= 'PDB1dir', 'PDB3dir'`.
2. Connect to the root to close **PDB1**.


```
SQL> ALTER PLUGGABLE DATABASE
  2  pdb1 CLOSE;
```
3. Then quiesce **PDB1**.


```
SQL> ALTER PLUGGABLE DATABASE
  2  pdb1 OPEN READ ONLY;
```
4. Clone **PDB3** from **PDB1**.


```
SQL> CREATE PLUGGABLE DATABASE
  2  pdb3 FROM pdb1;
```
5. Open **PDB3** in read write mode.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

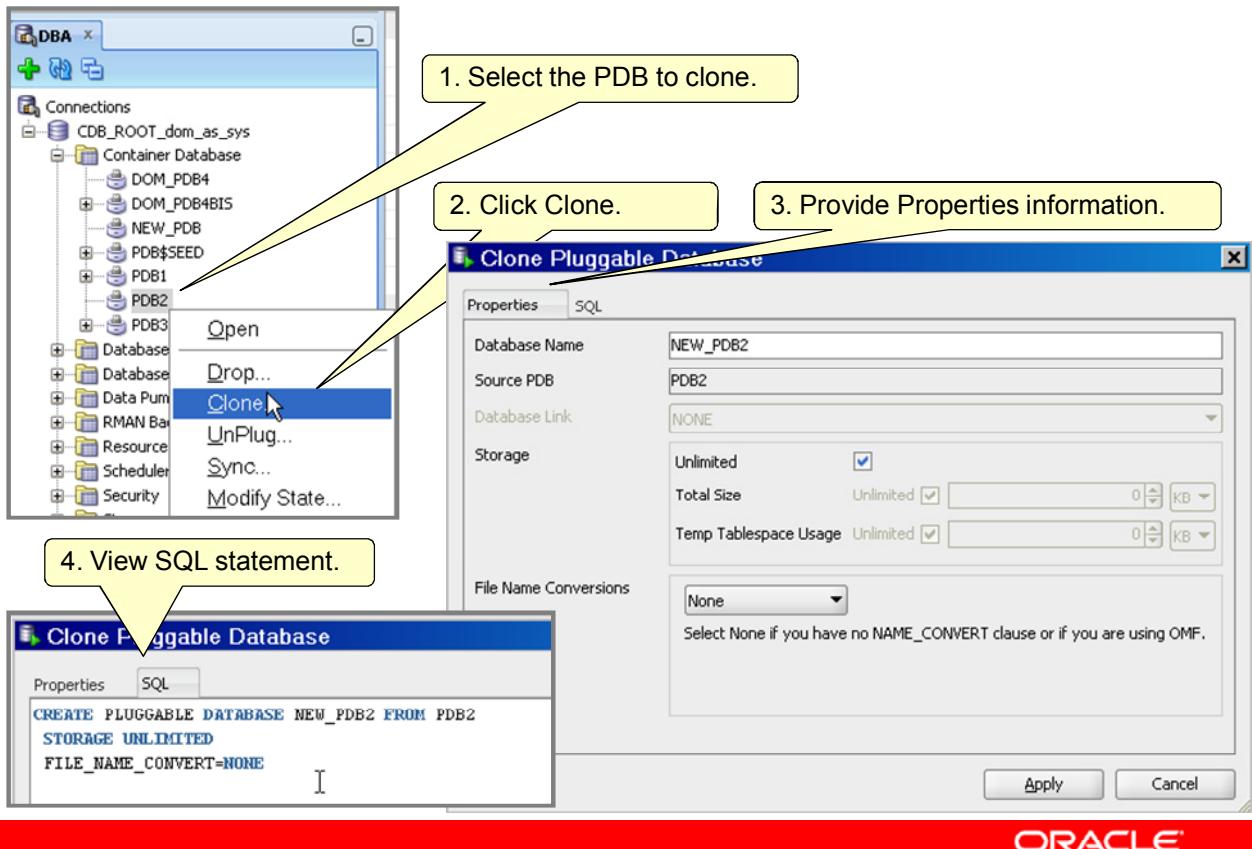
This technique copies a source PDB from a CDB and plugs the copy into a CDB. The source PDB is in the local CDB.

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

1. In `init.ora`, set `DB_CREATE_FILE_DEST= 'PDB3dir'` (OMF) or `PDB_FILE_NAME_CONVERT= 'PDB1dir', 'PDB3dir'` (non OMF).
2. Connect to the root of the CDB as a common user with the `CREATE PLUGGABLE DATABASE` privilege to close the PDB before opening it in read only mode, using the command `ALTER PLUGGABLE DATABASE CLOSE`.
3. Quiesce the source PDB used to clone, using the command `ALTER PLUGGABLE DATABASE pdb1 READ ONLY`.
4. Use the command `CREATE PLUGGABLE DATABASE` to clone the PDB `pdb3` from `pdb1`.
5. Then open the new `pdb3` with the command `ALTER PLUGGABLE DATABASE OPEN`.

If you do not use OMF in step 4, use the command `CREATE PLUGGABLE DATABASE` with the clause `FILE_NAME_CONVERT= ('pdb1dir', ' pdb3dir')` to define the directory of the source files to copy from `PDB1` and the target directory for the new files of `PDB3`.

Clone PDBs: Using SQL Developer



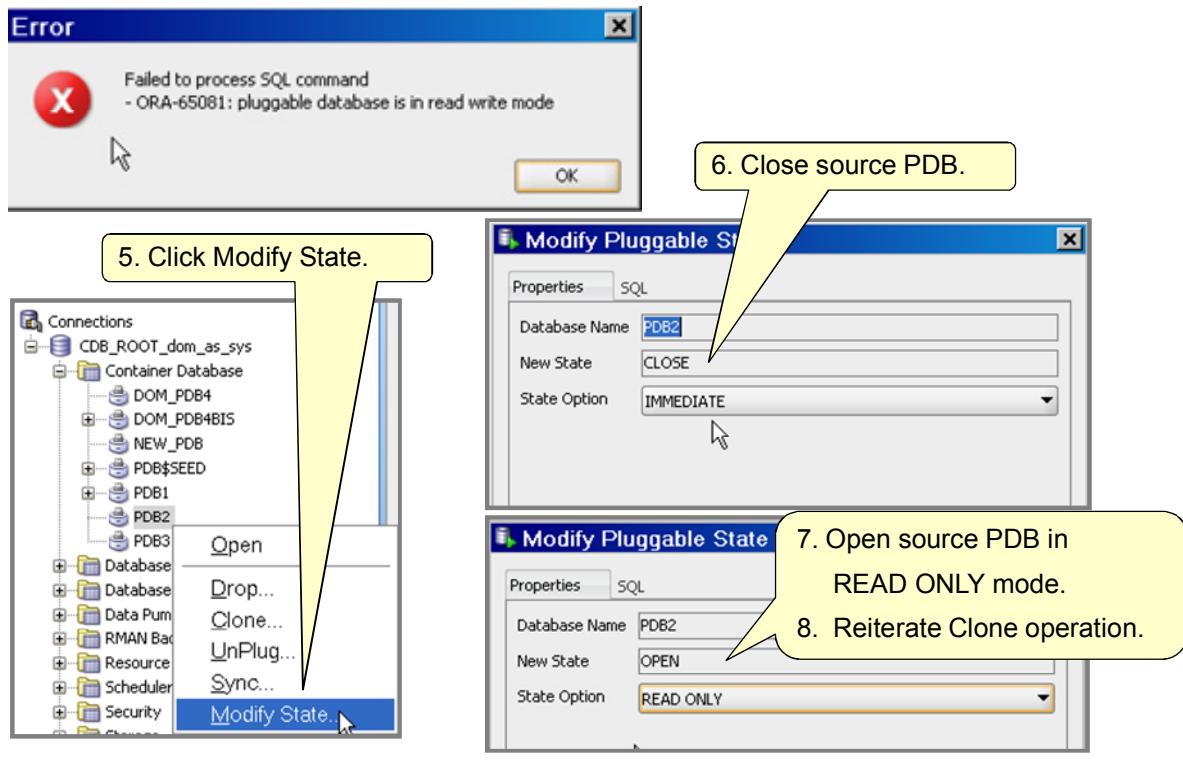
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use SQL Developer to clone a PDB from another PDB.

When you clone a PDB from another PDB, you first select the PDB to clone, then choose the Clone option, then provide the new PDB name, the PDB administrator username and the target files location in the File Name Conversions attribute. Use Custom Names to provide the Target File location. The Source File displays the list of the datafiles used as templates during the copy.

Clone PDBs: Using SQL Developer

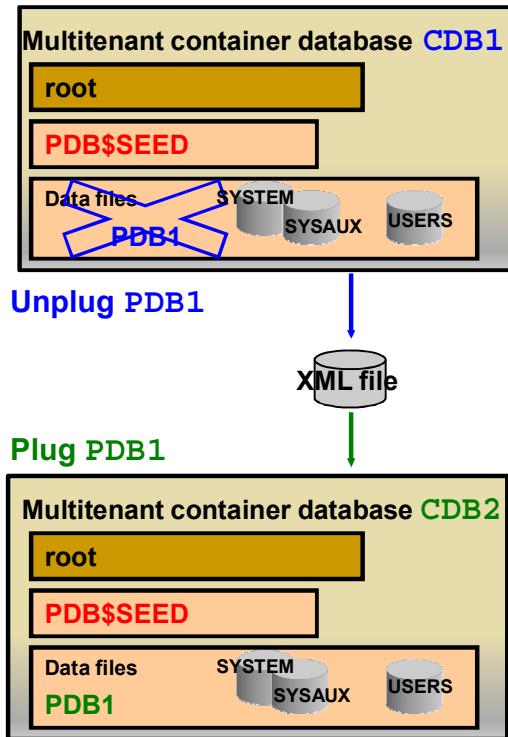
The source PDB must be in READ ONLY mode.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If the source PDB is still opened, you have to open it in READ ONLY mode. You first have to close it and then open it in READ ONLY mode. Then you can reiterate the clone operation.



Unplug **PDB1** from **CDB1**:

1. Connect to **CDB1** as a common user.
2. Verify that **PDB1** is closed.
3.

```
SQL> ALTER PLUGGABLE DATABASE
  2  pdb1 UNPLUG INTO 'xmlfile1';
```
4. Drop **PDB1** from **CDB1**

Plug **PDB1** into **CDB2**:

1. Connect to **CDB2** as a common user.
2. Use the DBMS_PDB package to check the compatibility of **PDB1** with **CDB2**.
3.

```
SQL> CREATE PLUGGABLE DATABASE
  2  pdb1 USING 'xmlfile1' NOCOPY;
```
4. Open **PDB1** in read write mode.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

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

```
SET SERVEROUTPUT ON
DECLARE compat BOOLEAN;
BEGIN
  compat := DBMS_PDB.CHECK_PLUG_COMPATIBILITY( pdb_descr_file
                                              => '/disk1/usr/salespdb.xml',
                                              pdb_name => 'PDB1');
  DBMS_OUTPUT.PUT_LINE('Is PDB compatible? = ' || compat);
END;
```

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

Method 4: Flow

Several clauses can be used in conjunction:

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

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

XML file accurately describes current locations of files?

If not, the SOURCE_FILE_NAME_CONVERT clause is required.

Are files are in correct location?

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

- FILE_NAME_CONVERT clause of CREATE PLUGGABLE DATABASE statement
- OMF: DB_CREATE_FILE_DEST parameter
- PDB_FILE_NAME_CONVERT parameter

Do you want to specify storage limits for PDB?

If yes, specify the STORAGE clause.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

Second question: Does the XML file accurately describe the current locations of the files? If not, the SOURCE_FILE_NAME_CONVERT clause is required.

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

Use one of these techniques to specify the target location:

- In the CREATE PLUGGABLE DATABASE statement, include a FILE_NAME_CONVERT clause that specifies the target locations based on the names of the source files.
- Enable OMF to determine the target location using the DB_CREATE_FILE_DEST initialization parameter.
- Specify the target locations in the PDB_FILE_NAME_CONVERT initialization parameter.

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

- FILE_NAME_CONVERT clause
- Oracle Managed Files using the DB_CREATE_FILE_DEST initialization parameter
- The PDB_FILE_NAME_CONVERT initialization parameter

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

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

```
CREATE PLUGGABLE DATABASE PDB1AS CLONE USING
'/disk1/usr/salespdb.xml' COPY;

CREATE PLUGGABLE DATABASE PDB1USING '/disk1/usr/salespdb.xml'
SOURCE_FILE_NAME_CONVERT = ('/disk1/oracle/sales',
'/disk2/oracle/sales') NOCOPY STORAGE (MAXSIZE 500M
MAX_SHARED_TEMP_SIZE 100M);

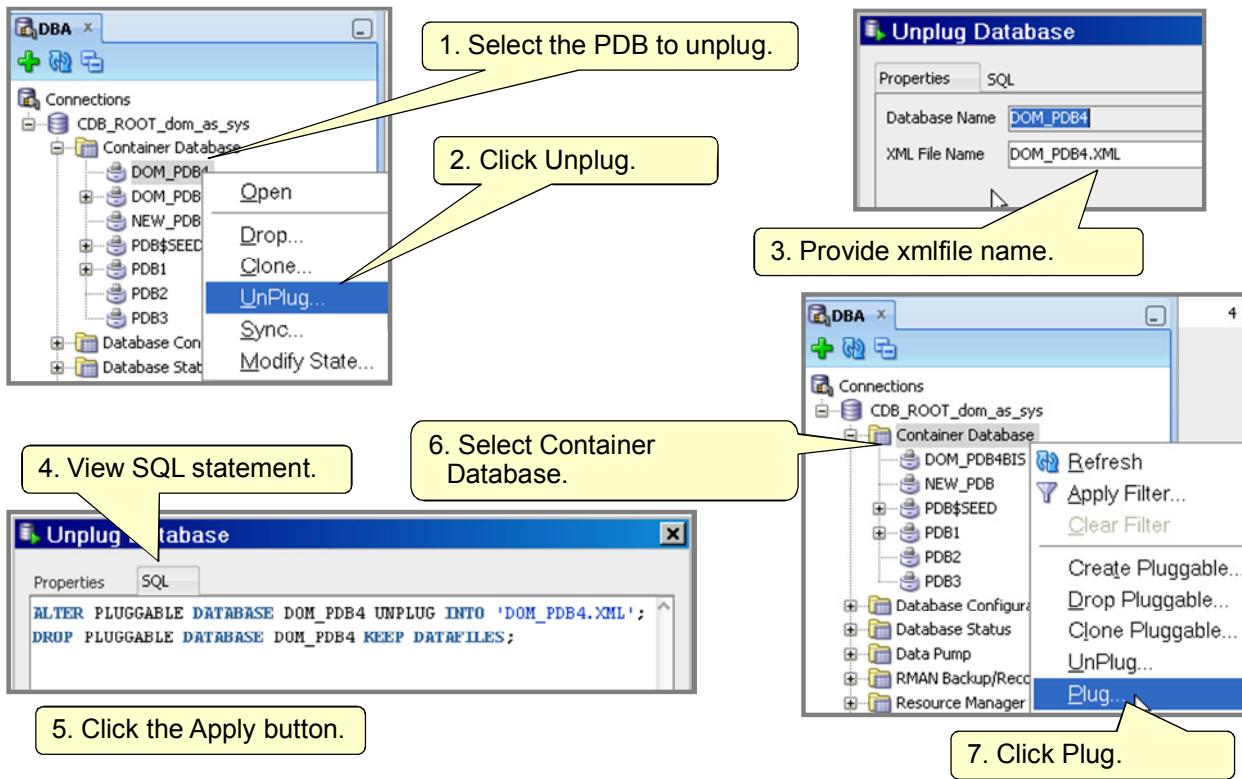
CREATE PLUGGABLE DATABASE PDB1USING '/disk1/usr/salespdb.xml'
COPY FILE_NAME_CONVERT = ('/disk1/oracle/sales',
'/disk2/oracle/sales');
```

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

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

PDB_NAME	STATUS
PDB1	NORMAL
PDB\$SEED	NORMAL
PDB3	UNPLUGGED

Unplugged PDB: Using SQL Developer



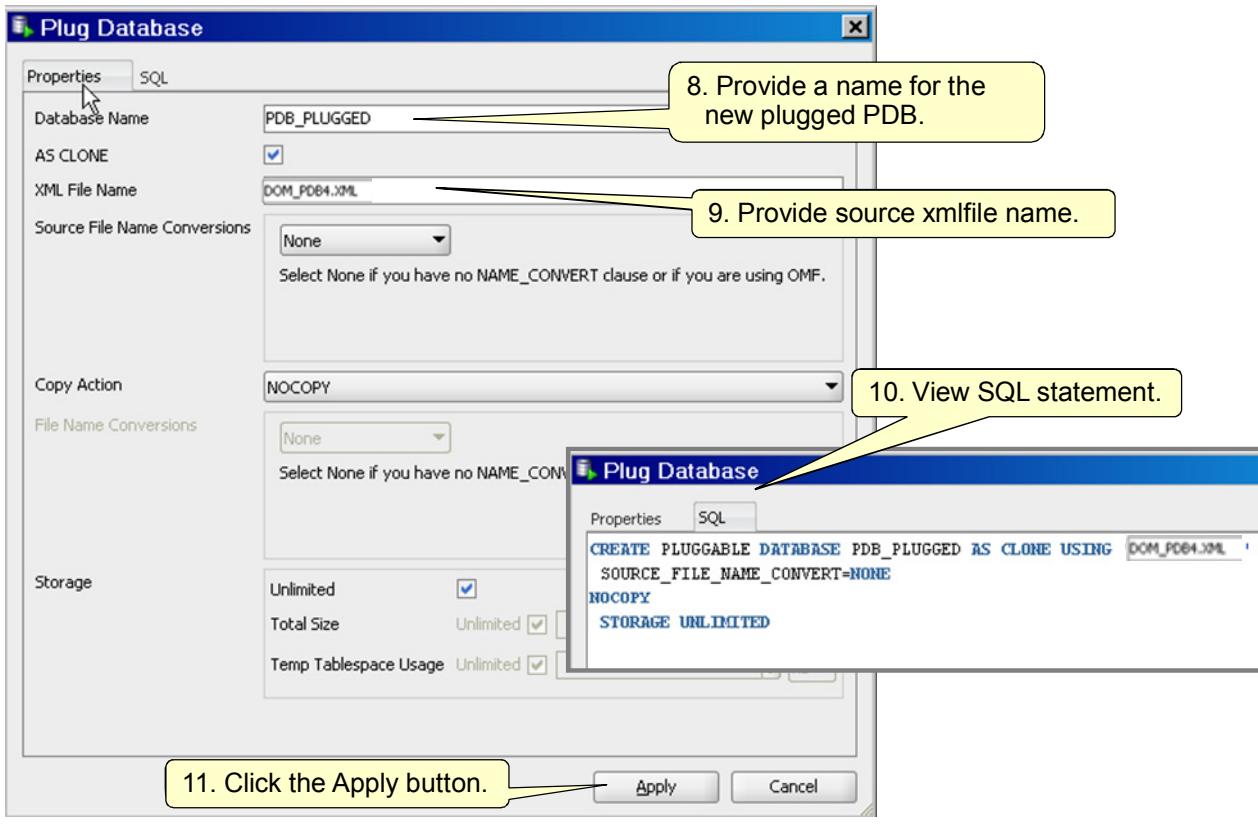
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use SQL Developer to perform the unplug and plug operations.

First select the PDB to unplug. Then click the Unplug option. Provide a name for the xmlfile file that will be generated. You can view the SQL statement before applying it. Then to plug the unplugged PDB, select the Container Database and choose the Plug option.

Plug Unplugged PDB: Using SQL Developer



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To plug the PDB, provide a name for the new plugged PDB and the XMLFILE source file. Before applying the statement, you can view it.

Unplug and Plug PDB with Encrypted Data

1. Export the master encryption key of the PDB to unplug.

```
SQL> ADMINISTER KEY MANAGEMENT
  2  EXPORT ENCRYPTION KEYS WITH SECRET my_secret
  3  TO '/tmp/export.p12' IDENTIFIED BY password
  4  CONTAINER = 'HR_PDB1';
```

2. Unplug and plug the PDB.
3. Import the master encryption key into the new PDB.

```
SQL> ADMINISTER KEY MANAGEMENT
  2  IMPORT ENCRYPTION KEYS WITH SECRET my_secret
  3  FROM '/tmp/export.p12' CONTAINER = 'HR_PDB2'
  4  IDENTIFIED BY password WITH BACKUP;
```

4. Open the keystore.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE
  2  OPEN IDENTIFIED BY password CONTAINER = CURRENT;
```

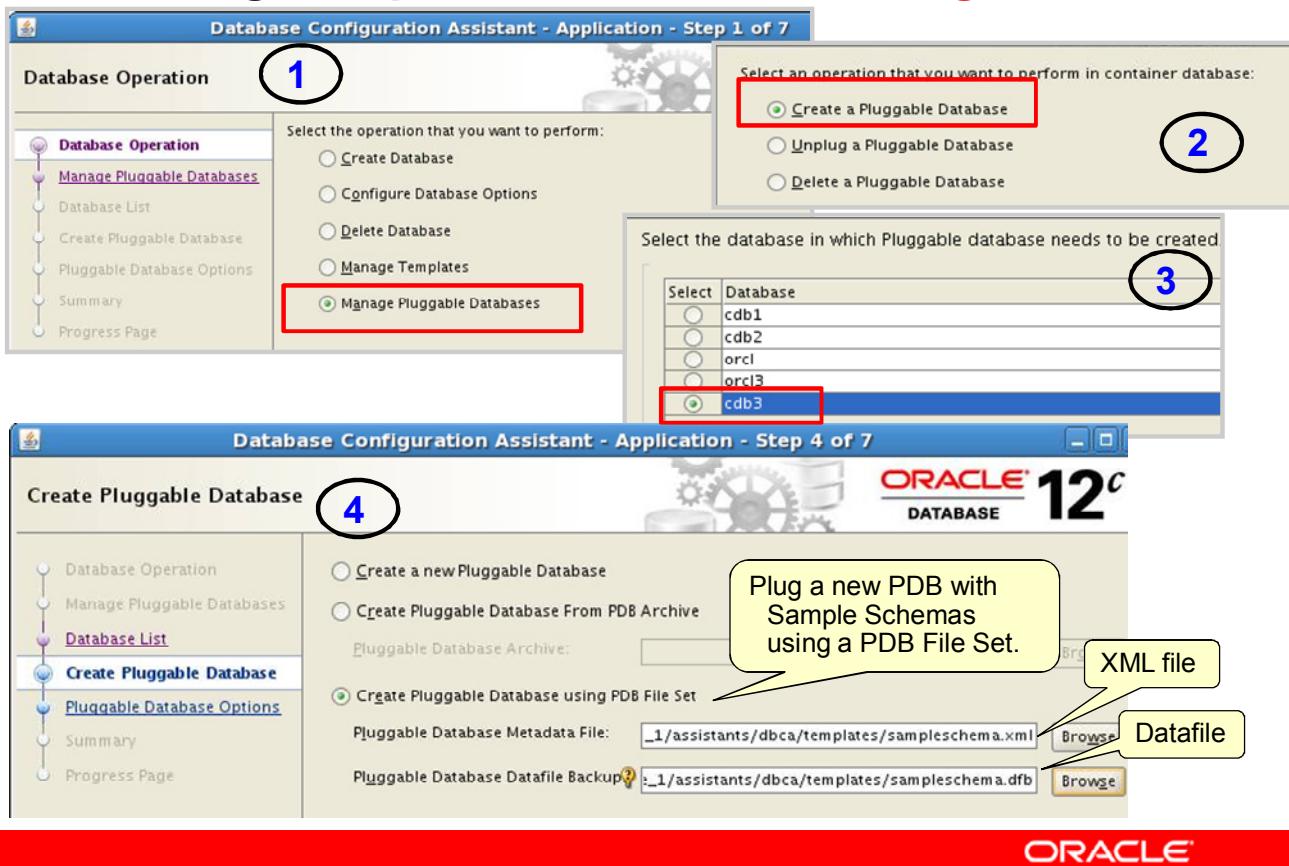
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The steps use the following sequence:

1. Export the Master Encryption Key from the PDB:
 - a. Log in to root as a user who has been granted the SYSKM system privilege.
 - b. Query the STATUS column of the V\$ENCRYPTION_WALLET view to find if the keystore is open.
 - c. Export the Master Encryption Key from the PDB using the ADMINISTER KEY MANAGEMENT EXPORT command.
2. Unplug the PDB and plug the PDB.
3. Import the Master Encryption Key into the PDB using the ADMINISTER KEY MANAGEMENT IMPORT command.
4. Open the keystore using the ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN command.

Plug Sample Schemas PDB: Using DBCA

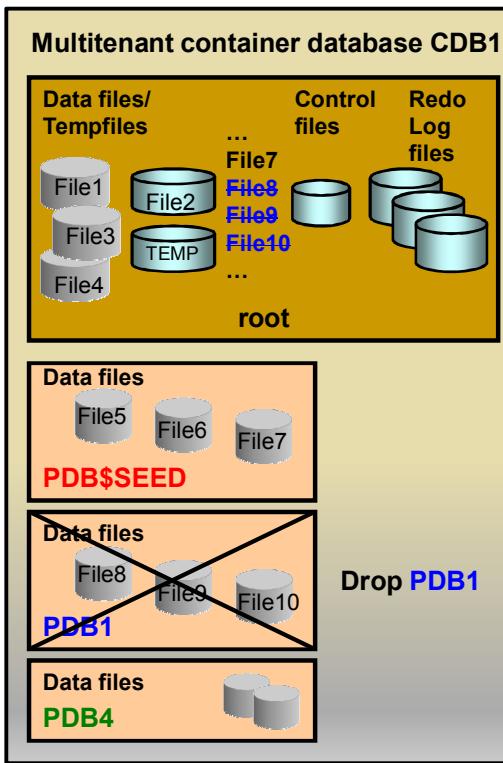


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can plug a new PDB with the sample schemas using DBCA.

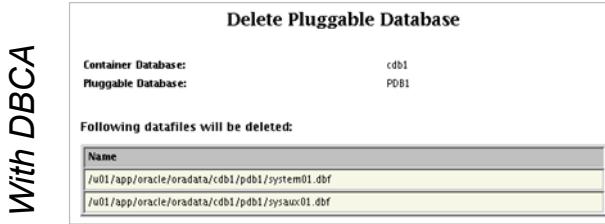
1. In DBCA, select "Manage Pluggable Databases".
2. Then choose "Create a Pluggable Database".
3. Select the CDB in which you intend to create the new PDB.
4. Click the "Create Pluggable Database using PDB File Set". Browse until you find both files:
 - Pluggable Database Metadata File:
\$ORACLE_HOME/assistants/dbca/templates/sampleschema.xml
 - Pluggable Database Datafile Backup:
\$ORACLE_HOME/assistants/dbca/templates/sampleschema.dfb
5. Define a name for the new PDB and a location for the data files.
You can also define a PDB User to create a new administrator for the PDB.
6. Click Next and Finish.

Dropping a PDB



```
SQL> ALTER PLUGGABLE DATABASE
  2  pdb1 CLOSE;
SQL> DROP PLUGGABLE DATABASE
  2  pdb1 [INCLUDING DATAFILES];
```

- Updates controlfiles
- If INCLUDING DATAFILES :
 - Removes **PDB1** datafiles
- If KEEP DATAFILES (default):
 - Retain datafiles
 - Can be plugged in another or same CDB
- Cannot drop seed PDB



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When you no longer need the data in a PDB, you can drop the PDB.

You can also unplug it and then drop it specifying `KEEP DATAFILES` to retain the datafiles associated with the PDB after the PDB is dropped. `KEEP DATAFILES` is the default behavior. Keeping data files may be useful in scenarios where an unplugged PDB is plugged into another CDB or replugged into the same CDB.

When you drop a PDB specifying `INCLUDING DATAFILES`, all of its datafiles listed in the control file are deleted.

With or without the clause `INCLUDING DATAFILES`, the `DROP PLUGGABLE DATABASE` statement modifies the control files to eliminate all references to the dropped PDB.

Backups are not removed, but you can use RMAN to remove them. This operation requires the `SYSBACKUP` privilege.

To perform the operation, use SQL*Plus or SQL Developer or DBCA or Enterprise Manager Cloud Control.

You cannot drop the seed pluggable database.

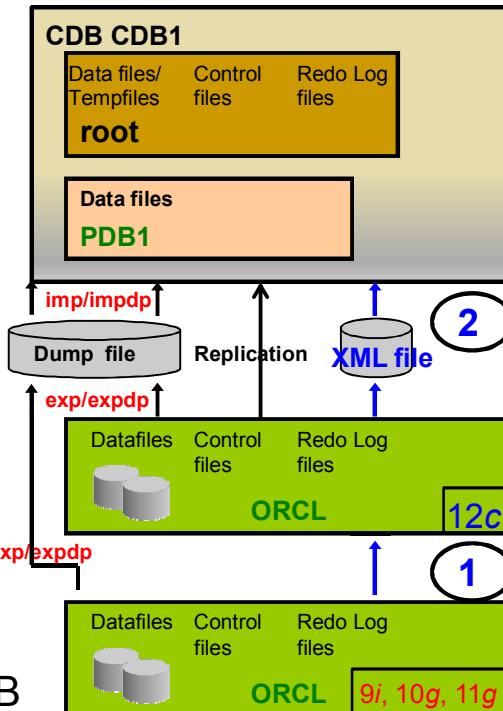
Migrating pre-12.1 or 12.1 non-CDB to CDB

There are two methods:

1. Upgrade an existing pre-12.1 non-CDB to 12c.
2. Plug-in non-CDB into a CDB.

Or

1. Precreate a PDB in CDB.
2. Use 11g expdp / 12c impdp.
or
Use 9i or 10g exp / 12c imp.
or
Use replication between non-CDB and PDB.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

There are two methods to migrate a non-container 11g database to a 12c CDB.

The first method consists of two steps:

1. Upgrade the 11g database (or any other previous Oracle Database release) to 12c non-CDB.
2. Plug in the 12c non-CDB into a CDB: Use `DBMS_PDB.DESCRIBE` procedure to generate the XML file to plug the data files into the CDB as a new PDB. This is the fastest solution.

The second method consists of two steps:

1. Precreate a PDB in the CDB from the seed PDB. This operation establishes an Oracle Database 12c dictionary in the newly created PDB.
2. Use either export/import (Data Pump or not) or replication to load the 9i, 10g, 11g data into the newly created PDB of the CDB.

Quiz

Which of the following are true about cloning a PDB into the same CDB? Select all that apply.

- a. It is not possible. You can only clone a PDB into another CDB.
- b. You can clone only one PDB into the same CDB.
- c. Cloning a PDB can use the source files copy method to the target PDB files.
- d. Cloning a PDB can use the clause NOCOPY if the target PDB files will use the source files.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b, c, d

Quiz

Which of the following are true about dropping a PDB?

- a. You can drop a PDB only if the PDB is closed.
- b. You can drop the seed PDB, but you will not be able to create any other PDB within the CDB.
- c. You can drop a PDB and keep the datafiles to be reused by another PDB.
- d. When you drop a PDB, the datafiles and redo logs files are automatically removed from the storage file system.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a, c

Summary

In this lesson, you should have learned how to:

- Configure and create a CDB
- Create a PDB from PDB\$SEED
- Create a PDB from a non-CDB
- Clone a PDB into the same CDB
- Unplug and plug a PDB from one CDB to another or the same CDB
- Explore the instance
- Explore the structure of PDBs
- Explore the ADR
- Drop a PDB
- Migrate pre-12.1 and 12.1 non-CDB to CDB



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 3 Overview: Creating a CDB and PDBs

These practices cover the following topics:

- Creating a new CDB
- Exploring the newly created CDB and its containers
- Creating a new PDB into the CDB using the seed
 - Using SQL*Plus and re-creating it using SQL Developer
- Cloning a PDB from a CDB into the same CDB
 - Using SQL*Plus or SQL Developer
- Plug a non-CDB into a CDB using SQL*Plus
- Merging two CDBs into a single one using SQL*Plus
- Dropping a CDB using SQL*Plus



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

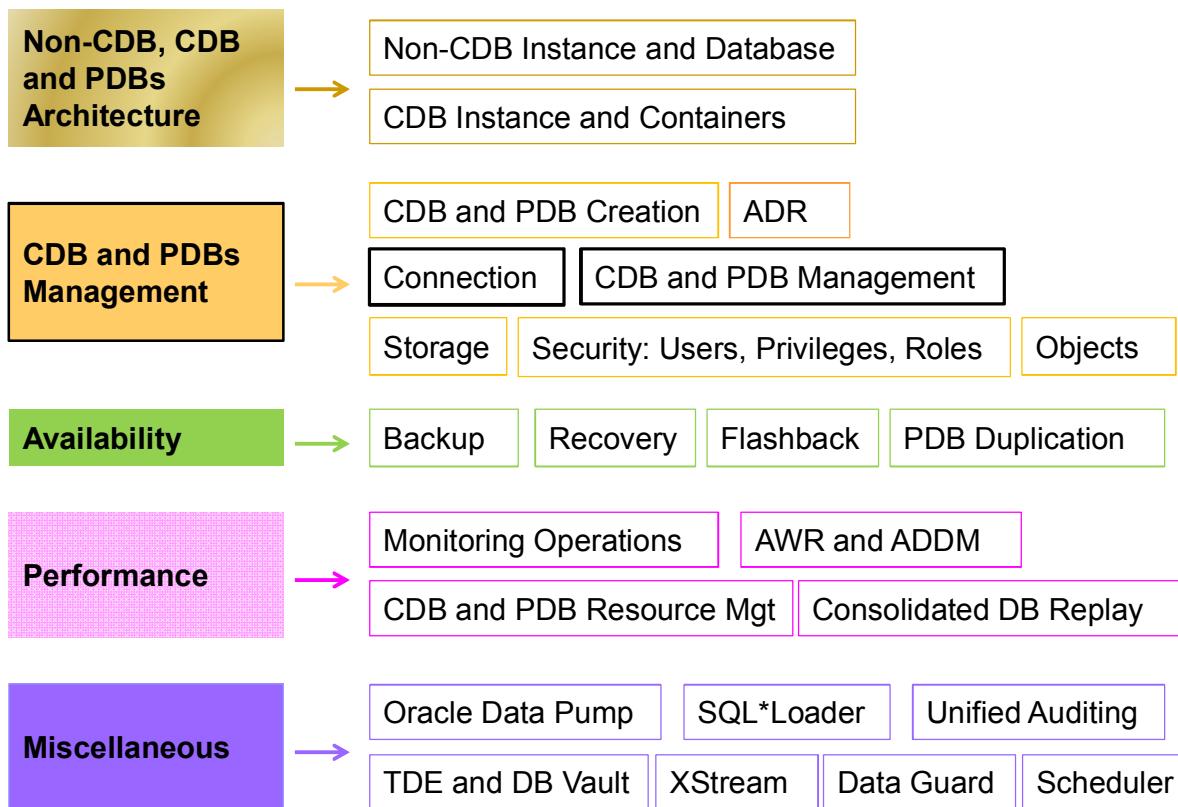
- Creating a CDB with no PDB with DBCA
- Creating a new PDB into a CDB using the seed with SQL*Plus or SQL Developer tools
- Cloning a PDB from a CDB into the same CDB with SQL*Plus or SQL Developer
- Plugging a non-CDB into a CDB
- Merging two CDBs into a single one with SQL*Plus
- Dropping a PDB and a CDB using SQL*Plus

Managing a Multitenant Container Database and Pluggable Databases

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Course Structure



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This lesson explains how to:

- Connect to a CDB and PDBs using services
- Administer CDB and PDBs

Objectives

After completing this lesson, you should be able to:

- Establish connections to CDB / PDB
- Start up and shut down a CDB
- Open and close PDBs
- Create event triggers to open PDBs
- Change the different modes and settings of PDBs
- Evaluate the impact of parameter value changes
- Change the way a PDB operates



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

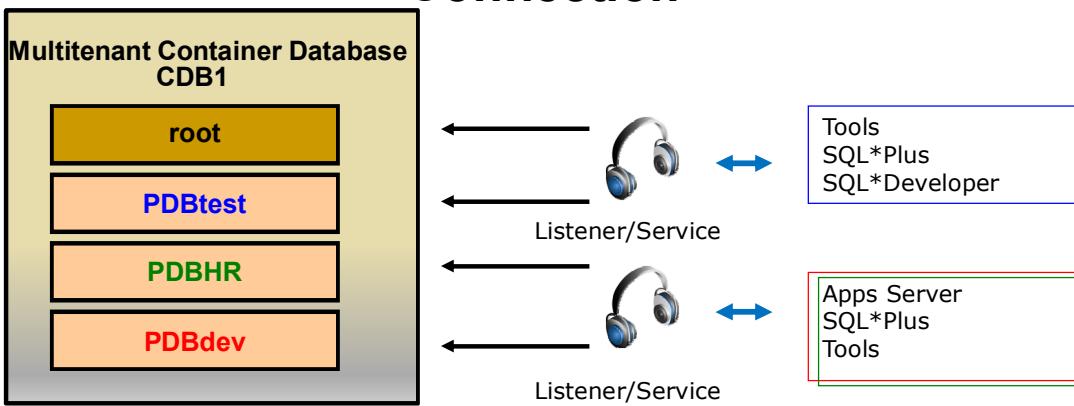
Note: For a complete understanding of the multitenant architecture and usage, refer to the following guides in the Oracle documentation:

- *Oracle Database Administrator's Guide 12c Release 1 (12.1)*

Refer to other sources of information:

- *Oracle Database 12c New Features Demo Series* demonstrations under Oracle Learning Library:
 - *CDB and PDBs Administration – Demo 1: Open and Close CDBs and PDBs*
 - *CDB and PDBs Administration – Demo 2: Change PDB mode*

Connection



1. Every PDB has a default service.

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

2. Service name has to be unique across CDBs.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

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

Using EasyConnect, you would enter the following connect string:

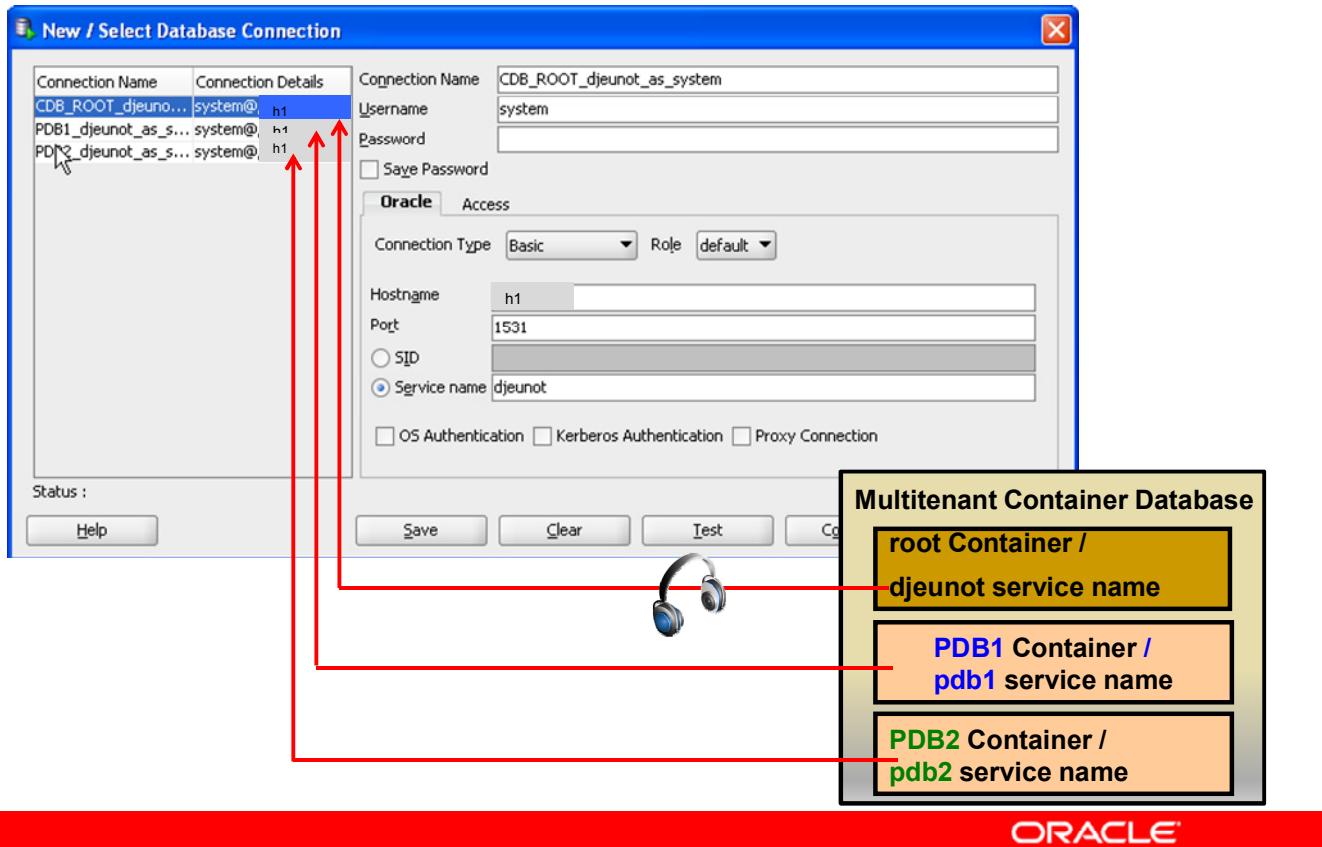
```
SQL> CONNECT username@hostname:portnumber/service_name  
SQL> CONNECT username@localhost:portnumber/service_name
```

Using the `tnsnames.ora` file, you would enter the following connect string:

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

To connect to a desired PDB, use either EasyConnect or the alias from the `tnsnames.ora` file, for example, as shown in the slide. In the examples used here, the net service name in the `tnsnames.ora` matches the service name.

Connection with SQL*Developer



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can also use SQL Developer to connect to the CDB, to the root and to any of the PDBs. As shown in the slide, in the left pane, there are three configured connection names, one to connect to the root, another one to connect to pdb1, and third one to connect to pdb2.

In the right pane, the connection name `CDB_root_djeunot_as_system` shows that the connection connects to the service name `djeunot` being the CDB name hence the root, as user `SYSTEM`.

Creating Services

- Using the **DBMS_SERVICE** package in an environment without Oracle Restart:

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

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

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

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

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

- Oracle Restart configuration automatically updated:

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Creating Additional Services by Using the DBMS_SERVICE Package

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

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

Creating Additional Services by Using the SRVCTL Utility

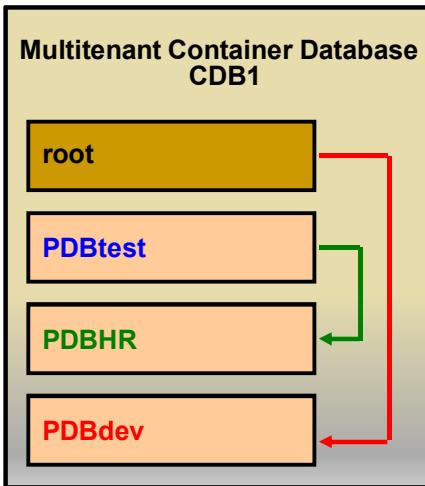
If your database is being managed by Oracle Restart or Oracle Clusterware, you can create or modify additional services for each PDB using the Server Control utility as follows:

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

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

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

Switching Connection



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

- Reconnect

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

- Use ALTER SESSION statement:

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

- Using CONNECT allows connection under common or local user.
- Using ALTER SESSION SET CONTAINER allows connection under common user only who is granted new system privilege SET CONTAINER.
 - AFTER LOGON triggers do not fire.
 - Transactions are still pending after switching containers.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

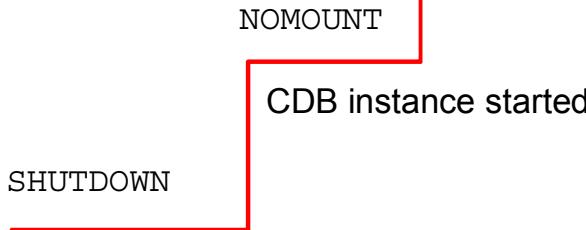
- Using the command CONNECT allows connections under common or local users.
- Using ALTER SESSION SET CONTAINER allows connections under a common user only who is granted the new system privilege SET CONTAINER.
 - Be aware that AFTER LOGON trigger do not fire.
 - Transactions that are not committed nor rolled back in the original container are still in a pending state while switching to another container and when switching back to the original container.

Starting Up a CDB Instance

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

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

```
no rows selected
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

Mounting a CDB

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

Or

```
SQL> ALTER DATABASE cdb1 MOUNT;
```

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

NAME	OPEN_MODE
PDB\$SEED	MOUNTED
PDB1	MOUNTED
PDB2	MOUNTED

NOMOUNT
Instance started
SHUTDOWN

MOUNT

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

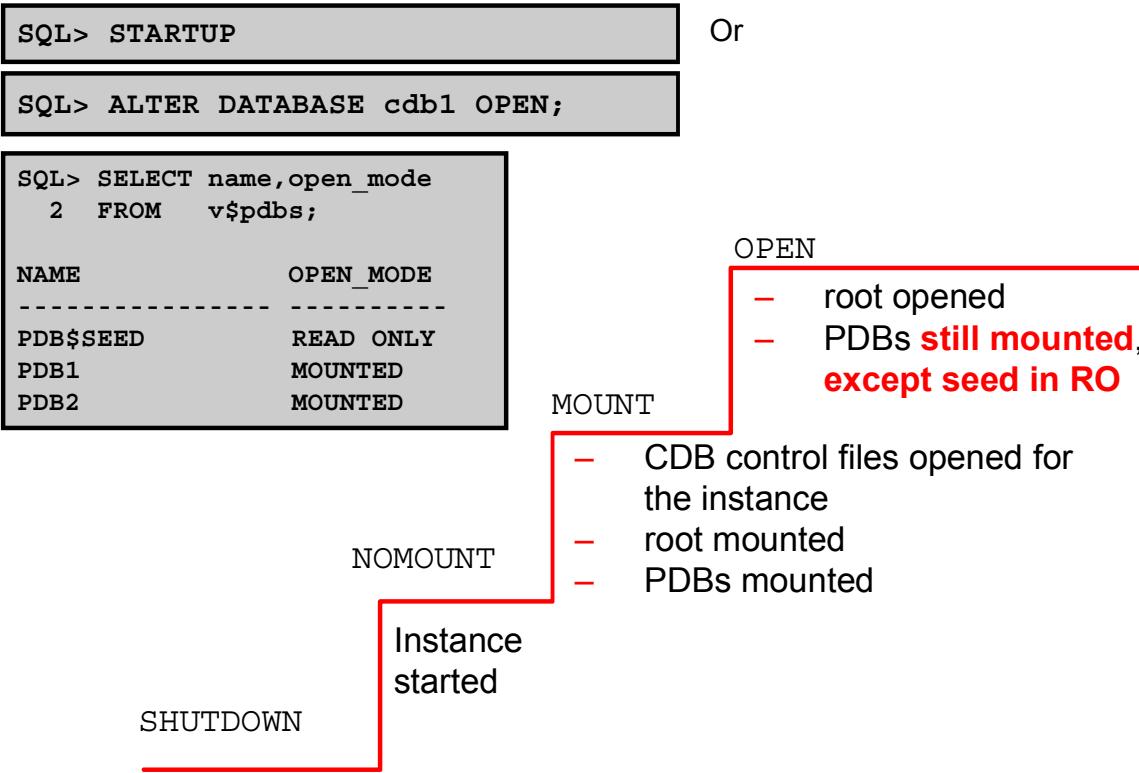
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

Opening a CDB



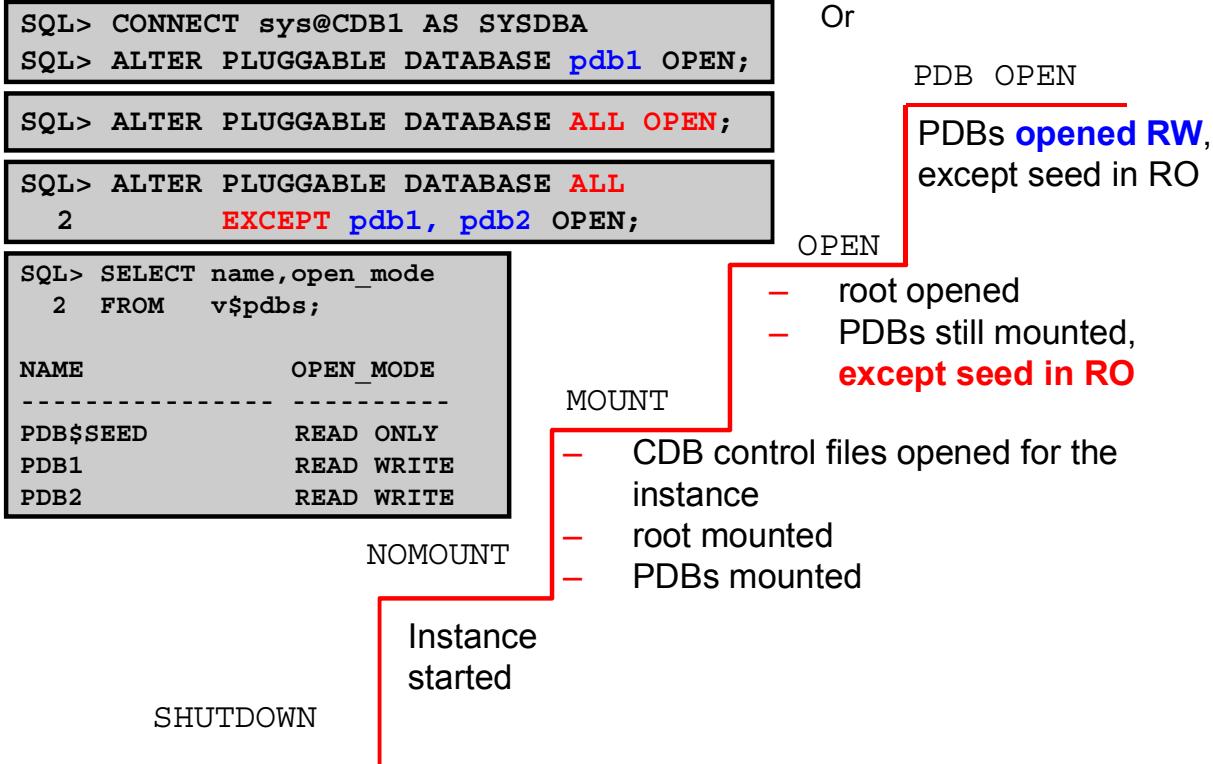
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

Opening a PDB



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

Closing a PDB

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

PDB CLOSE

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

CDB OPEN

PDBs closed

MOUNT

- root opened
- PDBs mounted, except seed still RO

NOMOUNT

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

SHUTDOWN

Instance started

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

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

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

Shutting Down a CDB Instance

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

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

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

- PDB closed



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

Database Event Triggers: Automatic PDB Opening

Trigger to automatically open PDBs after STARTUP

- AFTER STARTUP → ON DATABASE

New database event triggers

- AFTER CLONE → ON PLUGGABLE DATABASE
- BEFORE UNPLUG → ON PLUGGABLE DATABASE
 - Triggers are deleted after firing.
 - Any failure in AFTER CLONE or BEFORE UNPLUG trigger cause operation to fail.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

After opening a CDB, the PDBs are by default kept in mounted mode. If you want the PDBs opened automatically AFTER STARTUP ON DATABASE, you can create an database event trigger that opens all PDBs after STARTUP.

```
CREATE TRIGGER Open_All_PDBs
    after startup on database
begin
    execute immediate 'alter pluggable database all open';
end Open_All_PDBs;
/
```

In a Data Guard configuration, you can decide if a PDB needs to be opened or remain closed after a standby CDB role change. Instead of using the AFTER STARTUP ON DATABASE triggering event, use the AFTER DB_ROLE_CHANGE ON DATABASE triggering event that fires when database is opened for first time after role change.

The AFTER CLONE trigger fires in a PDB copy after it has been cloned. Any failure in an AFTER CLONE trigger causes the clone operation to fail.

The BEFORE UNPLUG trigger fires in the PDB before any unplug operation starts. Any failure in a BEFORE UNPLUG trigger causes the unplug operation to fail.

Such triggers are deleted after they fire. ON PLUGGABLE DATABASE must be specified when creating an AFTER CLONE or BEFORE UNPLUG trigger.

Changing PDB Mode

After closing a PDB, open in:

- Restricted mode

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

SQL> ALTER PLUGGABLE DATABASE OPEN RESTRICTED;

SQL> SELECT name, open_mode FROM v$pdbs;

NAME          OPEN_MODE
-----        -----
PDB1          RESTRICTED
```

- Read only mode

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

- Read write



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

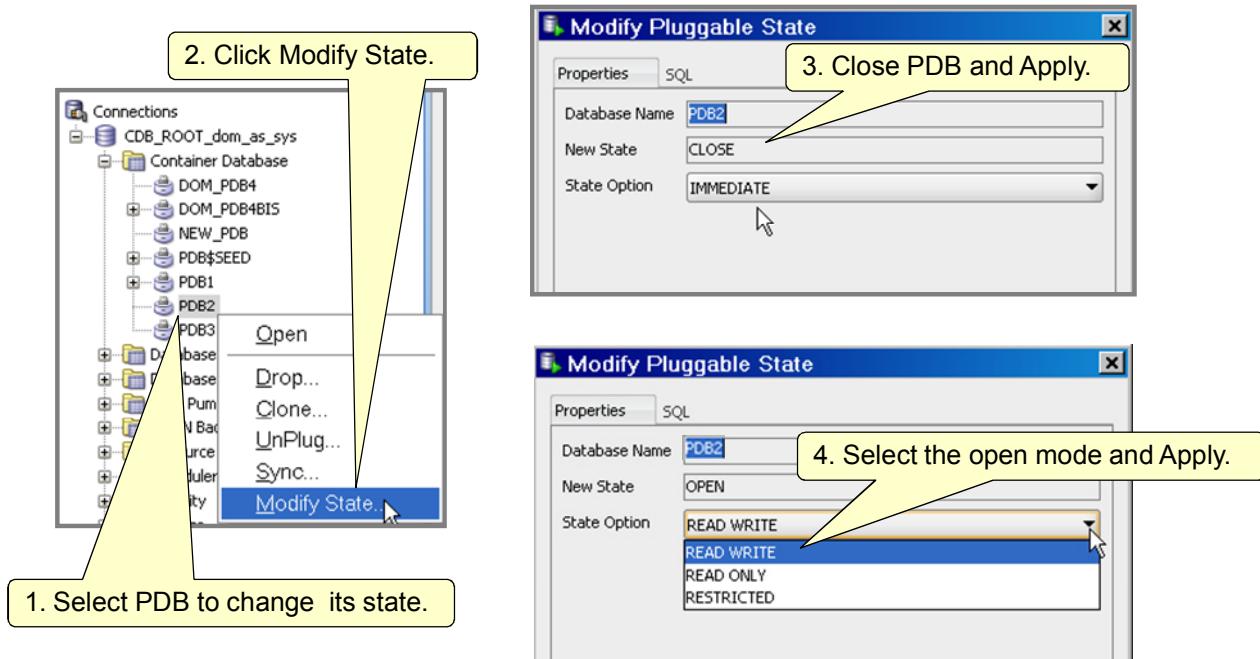
Use the V\$PDBS view to verify that the PDB is in RESTRICTED open mode.

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

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

Changing PDB Mode: With SQL Developer

Perform the same operations with SQL Developer.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If for example you want to clone a PDB and the source PDB is still opened, you have to open the PDB in READ ONLY mode. So, first close the PDB and then open the PDB in READ ONLY mode. Then you can reiterate the clone operation.

Modifying a PDB Settings

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

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

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

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

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

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

The first example uses a DATAFILE clause to bring the data file online.

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

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

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

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

Instance Parameter Change Impact

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

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

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



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

Instance Parameter Change Impact: Example

```
SQL> CONNECT sys@pdb2 AS SYSDBA  
  
SQL> ALTER SYSTEM SET ddl_lock_timeout=20 scope=BOTH;  
  
SQL> ALTER PLUGGABLE DATABASE CLOSE;  
SQL> ALTER PLUGGABLE DATABASE OPEN;
```

```
SQL> CONNECT / AS SYSDBA  
SQL> select VALUE, ISPDB_MODIFIABLE, CON_ID  
  2  from V$SYSTEM_PARAMETER  
  3  where name ='ddl_lock_timeout';  
  
VALUE          ISPDB      CON_ID  
-----  
0              TRUE       0  
10             TRUE       3  
20             TRUE       4
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

Using ALTER SYSTEM Statement on PDB

- Some statements change the way a PDB operates:

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

- Some ALTER SYSTEM statements can be executed in a PDB but affect the whole CDB:

ALTER SYSTEM CHECKPOINT	Affects all datafiles except those in read only or offline
-------------------------	--

- All other ALTER SYSTEM statements affect the entire CDB and must be run by a common user in the root.

ALTER SYSTEM SWITCH LOGFILE	Operation not allowed from within a pluggable database
-----------------------------	--

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

Quiz

When you start up a CDB, which is the sequence of operations performed automatically?

- a. Triggers may fire if they exist to open other PDBs.
- b. root container is opened (redo logs and root datafiles).
- c. Instance is started.
- d. Seed pluggable database is in READ ONLY mode.
- e. Other PDBs are still in MOUNTED mode.
- f. Control files are opened.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: False

The right sequence is as follows: c-f-b-d-e-a

Quiz

When you start up a CDB, all PDBs can be opened read write.

- a. False
- b. True



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a

Summary

In this lesson, you should have learned how to:

- Establish connections to CDB / PDB
- Start up and shut down a CDB
- Open and close PDBs
- Create event triggers to open PDBs
- Change the different modes and settings of PDBs
- Evaluate the impact of parameter value changes
- Change the way a PDB operates



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 4 Overview: Managing a CDB and PDBs

These practices cover the following topics:

- Starting up and shutting down a CDB
- Connecting to PDBs and displaying context
- Opening and closing PDBs
- Changing PDB behavior with the ALTER SYSTEM statement



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

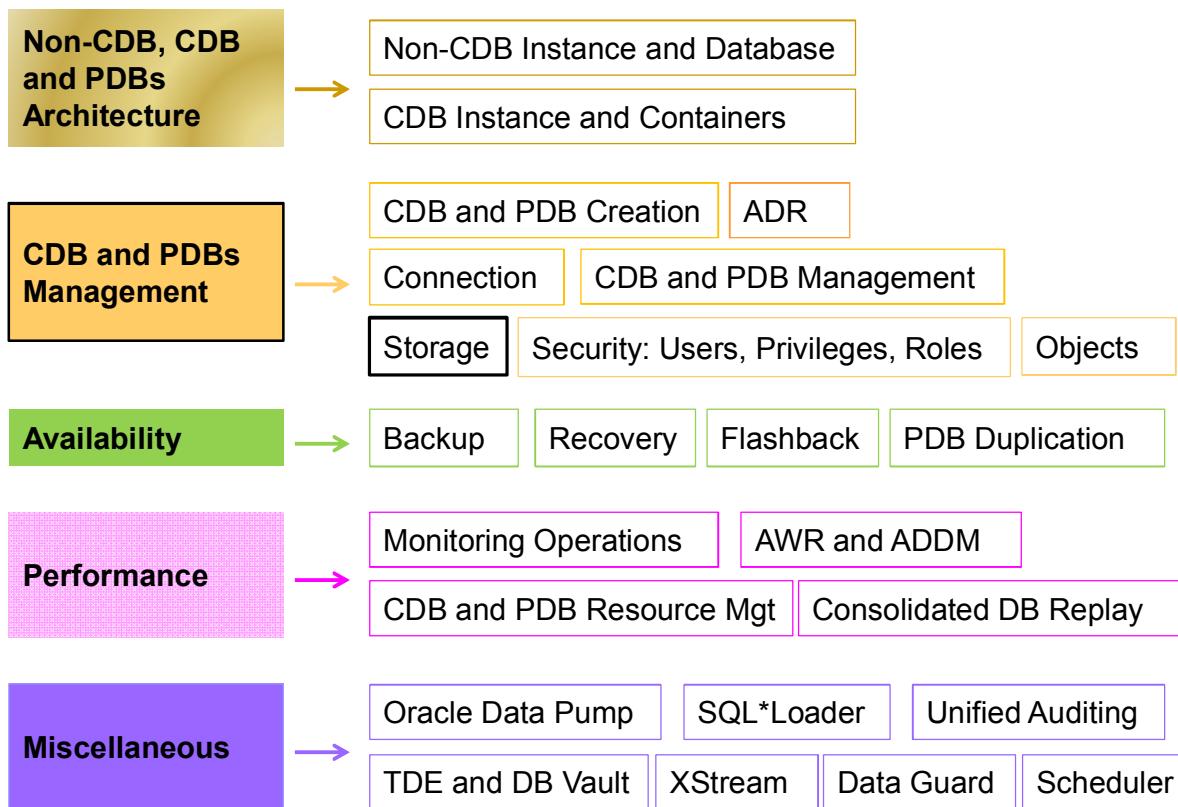
Oracle University and Error : You are not a Valid Partner use only

Managing Tablespaces in CDB and PDBs

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Course Structure



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This lesson explains how to:

- Manage the storage in a CDB and the storage for application schemas in PDBs
- Manage the temporary storage in a CDB and in PDBs

Objectives

After completing this lesson, you should be able to:

- Manage permanent tablespaces in CDB and PDBs
- Manage temporary tablespaces in CDB and PDBs



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Note: For a complete understanding of Oracle PDB new feature and usage, refer to the following guides in the Oracle documentation:

- *Oracle Database Administrator's Guide 12c Release 1 (12.1)*

Tablespaces in PDBs

- A tablespace in a PDB can contain objects associated with exactly one PDB.
- In CREATE DATABASE:
 - `USER_DATA TABLESPACE` replaces automatic creation of `USERS` tablespace by DBCA
- There is only one active `UNDO` tablespace per CDB.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In a non-CDB, all the tablespaces belong to one database. In the CDB, one set of tablespaces belong to the root container, and each PDB has its own set of tablespaces.

Common objects are created and their data stored in a tablespace in the root container. The common object is visible in the PDBs through links.

There are new clauses in the `CREATE DATABASE` command. The `USER_DATA TABLESPACE` allows you to specify a default tablespace other than `USERS` when using DBCA to create a database. This tablespace will also be used for XDB options.

The `UNDO` tablespace is common to all PDBs, that is, there is only one active `UNDO` tablespace per CDB.

Creating Permanent Tablespaces in a CDB

- Create a permanent tablespace in the root container:

```
SQL> CONNECT system@cdb1
SQL> CREATE TABLESPACE tbs_CDB_users DATAFILE
  2  '/u1/app/oracle/oradata/cdb/cdb_users01.dbf'
  3  SIZE 100M;
```

- Create a permanent tablespace in a PDB:

```
SQL> CONNECT system@PDB1
SQL> CREATE TABLESPACE tbs_PDB1_users DATAFILE
  2  '/u1/app/oracle/oradata/cdb/pdb1/users01.dbf'
  3  SIZE 100M;
```



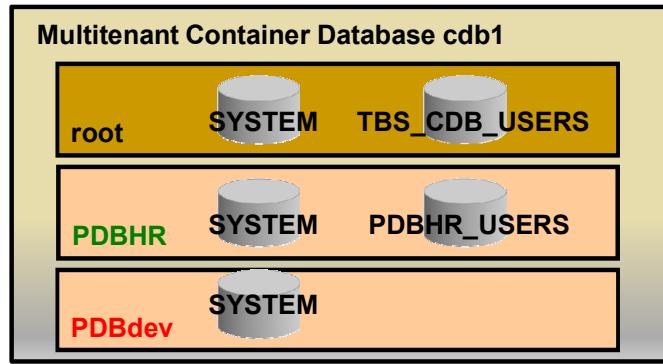
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The CREATE TABLESPACE command should be familiar. The change in its behavior in a CDB is that the tablespace is created in the container where the command is executed.

Separating the datafiles into different directories by PDB can help determine which files belong to which PDB, though it is not necessary.

You can use Oracle ASM storage to manage your disk storage.

Assigning Default Tablespaces



- In the CDB:

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

- In the PBD:

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

ORACLE

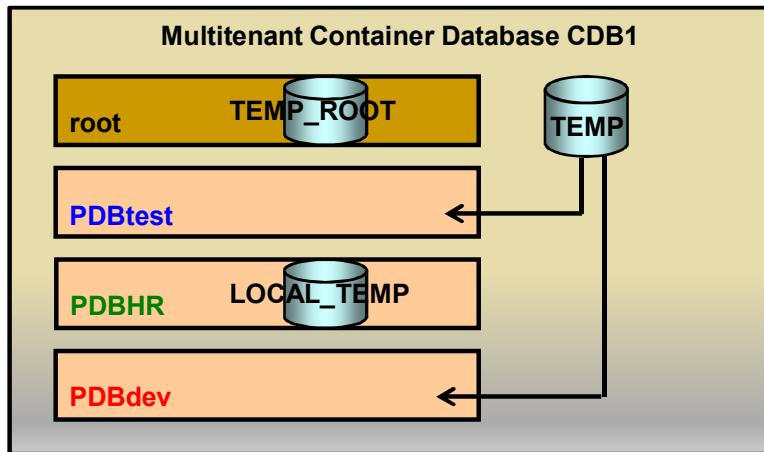
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

Creating Local Temporary Tablespaces

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



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A CDB can have only one default temporary tablespace or tablespace group. As with a non-CDB there can be other temporary tablespaces to which users can be assigned. The default temporary tablespace is a shared resource for all PDBs, that is, any user in any PDB will use the CDB default temporary tablespace unless that user is assigned to a specific temporary tablespace.

PDBs can have temporary tablespaces for use by users in the PDB. These temporary tablespaces will be transported with the PDB when it is unplugged.

Assigning Default Temporary Tablespaces

- In the CDB:

```
SQL> CONNECT system@cdb1
SQL> ALTER DATABASE
  2> DEFAULT TEMPORARY TABLESPACE temp_root;
```

- In the PBD:

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

or

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



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The default temporary tablespace for the CDB is set at the root container level. There is one default temporary tablespace (or tablespace group) for an entire CDB. There may be multiple temporary tablespaces, but only one can be the default.

A default temporary tablespace (or tablespace group) can be set for each PDB. A PDB may have multiple temporary tablespaces, but only one default per PDB.

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

The default temporary tablespace in the CDB is shared by all the PDBs. The amount of space a single PDB can use in the shared temporary tablespace can be set in the PDB by:

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

When you unplug a PDB from a CDB, its temporary tablespaces are also unplugged.

Quiz

You can create several temporary tablespaces in PDBs.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a

Quiz

You can create several default temporary tablespaces in a CDB.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a

One for the root and one for each PDB. The default temporary tablespace is a shared resource for all PDBs, that is, any user in any PDB will use the CDB default temporary tablespace unless that user is assigned to a specific temporary tablespace.

Summary

In this lesson, you should have learned how to manage any type of tablespaces in CDB and PDBs.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 5 Overview: Managing Tablespaces and Users in CDB and PDBs

These practices cover the following topics:

- Creating a tablespace in the root
- Creating a tablespace in a PDB



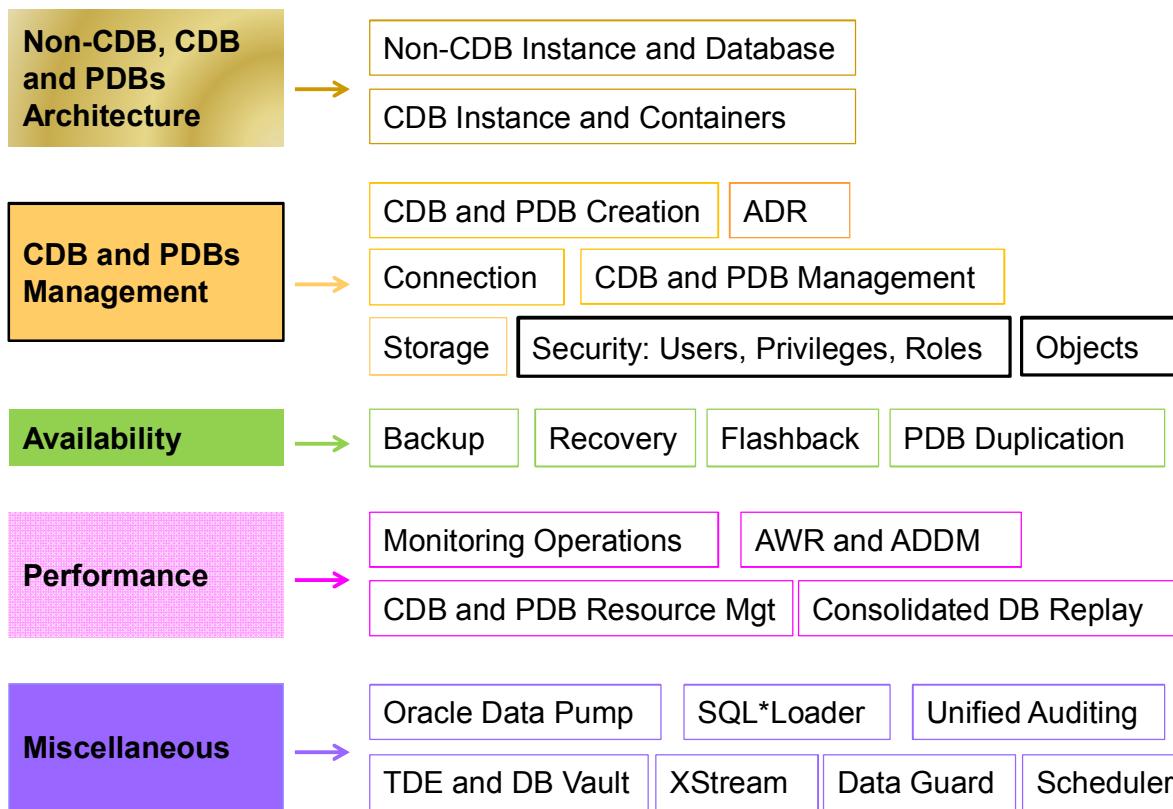
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Managing Security in CDB and PDBs

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Course Structure



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This lesson explains how to:

- Manage the users, privileges, and roles in a CDB and to any of its PDBs
- Manage types of objects in a CDB and PDBs

Objectives

After completing this lesson, you should be able to:

- Manage common and local users
- Manage common and local roles
- Manage common and local privileges
- Manage the CONTAINER_DATA attributes of common users
- Manage common and local profiles



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

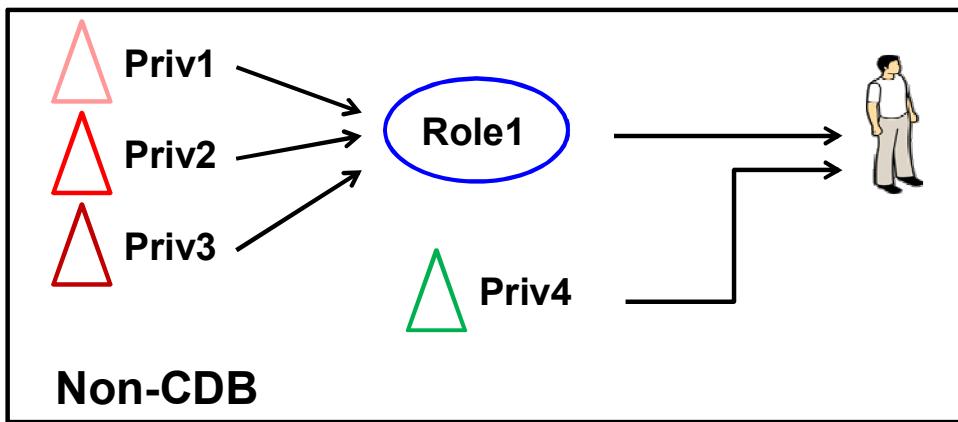
- *Oracle Database Administrator's Guide 12c Release 1 (12.1)*
- *Oracle Database Security Guide 12c Release 1 (12.1)*

Refer to other sources of information:

- *Oracle Database 12c New Features Demo Series* demonstrations under Oracle Learning Library:
 - *CDB and PDBs Administration – Manage Common and Local Users*
 - *CDB and PDBs Administration – Manage Common and Local Roles and Privileges*

Users, Roles, and Privileges

- Each user can exercise granted privileges in the context of a single database.
- A role is a collection of privileges.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

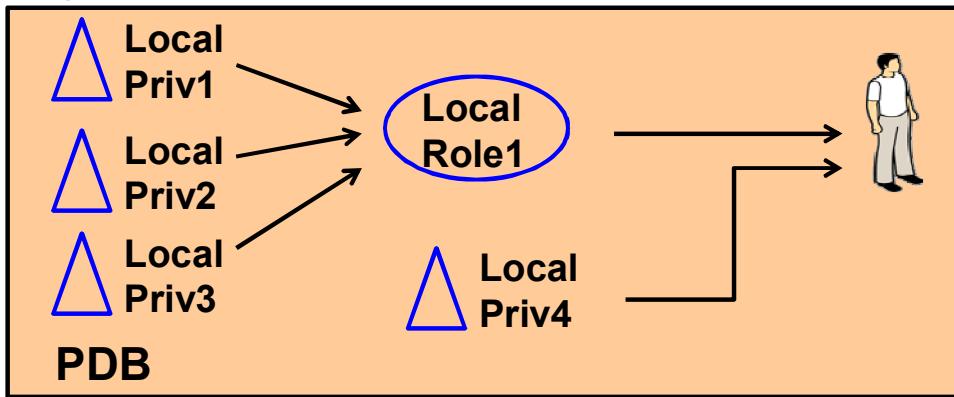
In a non-CDB, a user can perform operations authorized by system privileges, on objects allowed by object privileges. These privileges apply only to the database to which the user is connected.

A role is a set of privileges. The privileges can be exercised in the context of a specific database. Privileges can be granted either directly or through roles.

In the past, the phrase “in the context of a single database” has been implied. When you consider that there was only one database per instance, this seems a trivial assertion, but this is key to understanding local users, local privileges, and local roles in PDBs.

Local Users, Roles, and Privileges

- Each **local** user can exercise granted privileges in the context of a single **PDB**.
- A **local** role is a collection of privileges that are assigned at user login to a specific **PDB**.
- A **local** privilege is one that is granted in the context of a single **PDB**.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A local user exists in one and only one PDB. Even if multiple PDBs have local users with the same name and credentials, each local user is distinct.

In a PDB, the local users behave in exactly the same manner as users defined in the context of a non-CDB. A local user can perform DDL operations authorized by system privileges and DML and other operations on objects allowed by object privileges. The privileges apply only to the PDB in which the local user is defined. The differences between the rules for a non-CDB and a PDB are highlighted. As you can see, they are almost identical.

The commands to create local users and roles in a PDB are the same as for a non-CDB.

Creating a Local User

- A local user with proper privileges can create another local user.
- The syntax is the same as in non-CDB.

```
SQL> CONNECT system@pdb1
SQL> CREATE USER george IDENTIFIED BY x;
```

- You cannot create local users in the root.
- A local user connects only to the PDB where it exists.

```
SQL> CONNECT george@pdb1
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

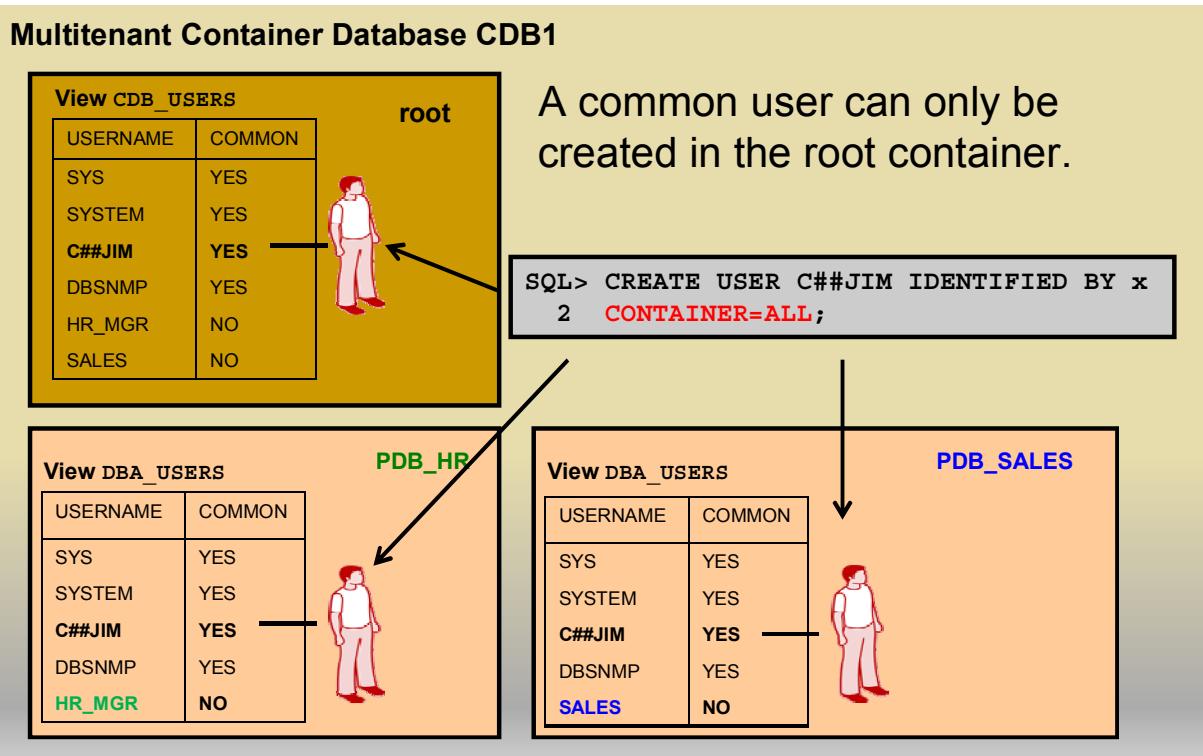
A local user can be created by another local user with the proper privilege, typically one with the DBA role in the PDB. The CREATE USER privilege is required.

The syntax is the same as in a traditional or non-CDB. This is in keeping with the application transparency goal for PDBs. The syntax for commands in a PDB are identical to commands in a non-CDB.

Note: The DROP USER and ALTER USER commands are identical, as the commands in a non-CDB.

You cannot create local users in the root.

Common Users



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A common user is a user that has the same username and authentication credentials across multiple PDBs, unlike the local user which exists in one and only one PDB.

A common user cannot have the same name as any local user across all of the PDBs. A common user can be created in the root container, and only in the root container: a common user is a database user that has the same identity in the root and in every existing and future PDB.

A local user cannot create a common user.

A common user with the proper privileges can create a common user using the CONTAINER=ALL clause. To create a common user, the common user must have the SET CONTAINER privilege for all the PDBs. A common user can connect to a specific PDB and with the proper privilege in the PDB create a local user with the CONTAINER = CURRENT clause.

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

Creating a Common User

- A common user can create common users and local users.
- The **CONTAINER** clause determines the type of user created.
- Create a **common** user in the root container:

```
SQL> CREATE USER c##jim IDENTIFIED BY x  
2 CONTAINER=ALL;
```

- Create a **local** user in a PDB:

```
SQL> CREATE USER l_fred IDENTIFIED BY y  
2 CONTAINER=CURRENT;
```

- A common user exists in each container:

```
SQL> CONNECT c##jim@cdb1 → jim connection in root  
SQL> CONNECT c##jim@pdb1 → jim connection in pdb1  
SQL> CONNECT c##jim@pdb2      jim connection in pdb2
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Only a common user with the `CREATE USER` and `SET CONTAINER` common privileges can create a common user. A common privilege is a privilege granted across all the PDBs.

A common user with the `CREATE USER` and `SET CONTAINER` privileges in the PDB can create a local user in that PDB.

The type of user created is determined by the `CONTAINER` clause.

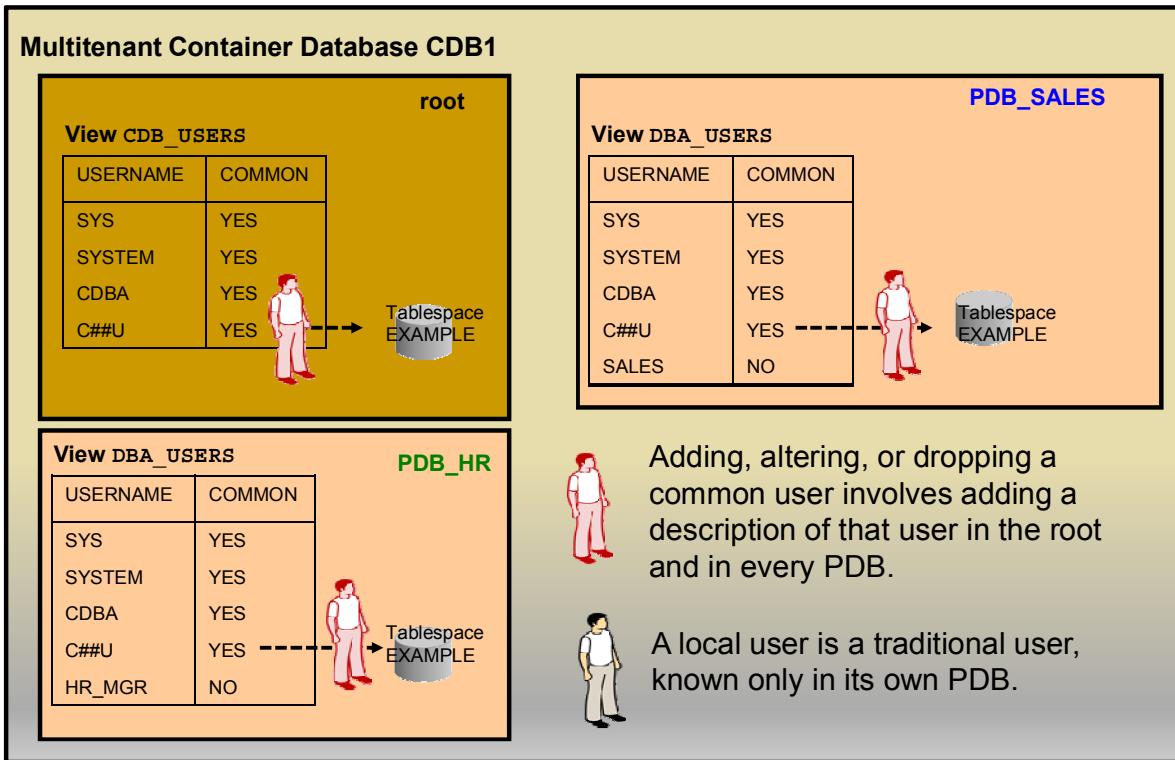
- The first command shown in the slide creates a common user when issued in the root container. To create a common user, you must be connected to the root container. The name of a common user must begin with `C##` characters.
- The second command creates a local user when the common user is connected to a specific PDB.

If the container clause is omitted, the default depends on the context. If the common user is connected to the root container the default is `CONTAINER=ALL`. If the common user is connected to a PDB, the default is `CONTAINER=CURRENT`.

When creating a common user, any tablespace, tablespace group, or profile specified in the `CREATE` command must exist in every PDB. If none of these are specified, the default `TABLESPACE`, `TEMPORARY TABLESPACE`, and `PROFILE` for the PDB will be used.

Note: The `DROP USER` and `ALTER USER` commands are the same as in a non-CDB.

Common and Local Schemas



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Adding, altering, or dropping a common user involves adding, changing, or dropping a description of that user in the root and in every PDB. After changing the container, common users can or cannot perform operations depending on the type of privileges granted in the container (PDB).

A PDB that was not OPEN READ WRITE or RESTRICTED when a common user was created attempts to add the description of that common user as a part of the synchronization operation. If during the process, a new common user name conflicts with a local username defined in a given PDB, or a tablespace group or a profile that was specified in CREATE USER statement does not exist in that PDB, an error is reported.

Altering or dropping a common user involves modifying or removing a description of that user in the root container and in every PDB that is OPEN READ WRITE or RESTRICTED, effectively replaying the statement in each container.

Common and Local Privileges

- A privilege granted across all containers is a common privilege.
- A privilege granted in the context of a single PDB is a local privilege.
- Local users can only exercise privileges locally in the context of the PDB.
- Common users can only exercise privileges in the context of the PDB to which they are connected.
- Common users connected to the root container can exercise cross-container privileges, such as creating a common user.



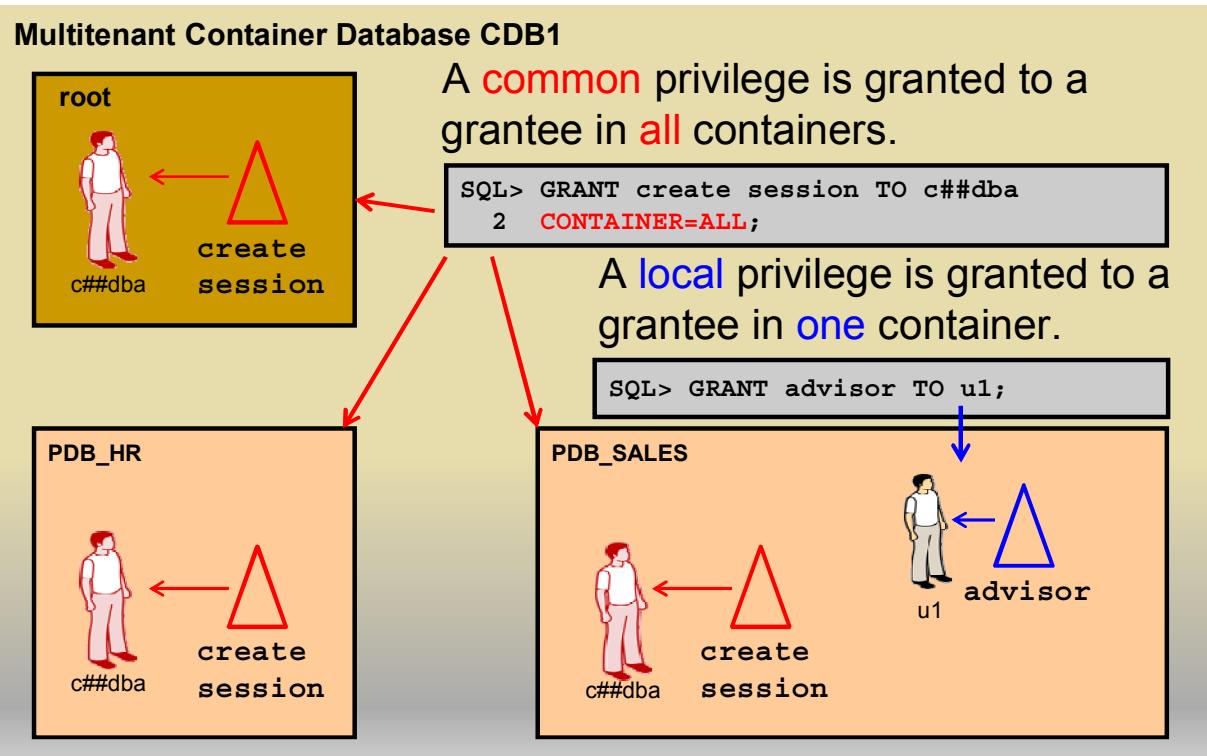
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A privilege becomes common or local based on the way it is granted. A privilege granted across all containers is a common privilege. A privilege granted in a specific PDB is local.

Common and local users can exercise common and local privileges that have been granted in the context of the PDB to which they are connected. A common user can be granted both common and local privileges. That means the privileges granted to a common user can be different in every PDB.

Only common users can connect to the root container. Even in the root container, privileges can be granted as local or common. Cross-container operations such as creating a common user can only be performed while connected to the root container and by common users with common privileges for those operations.

Granting Common and Local Privileges



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A privilege granted across all containers is a **common** privilege. In the example in the slide, the `CREATE SESSION` privilege is granted commonly to the `c##dba` user using the clause `CONTAINER=ALL`. The grant operation is replicated in all containers. Consequently, the same user `c##dba` is granted the same privilege in each container.

A privilege granted in a specific PDB is **local**. In the example in the slide, the `ADVISOR` privilege is granted locally to the `u1` user. The grant operation is not replicated in all containers. Even if the user were a common user, the grant operation would not have been replicated. Consequently, the user `u1` is granted the privilege in the `PDB_SALES` container only.

Granting and Revoking Privileges

- Grant a privilege commonly: Becomes a common privilege

```
SQL> GRANT drop user TO c##_user CONTAINER=ALL;
```

- Grant a privilege locally: Becomes a local privilege

```
SQL> GRANT alter user TO local_user CONTAINER=CURRENT;
```

- Grant a local privilege.

```
SQL> GRANT select any table TO local_user;
```

- Revoke a common privilege.

```
SQL> REVOKE drop user FROM c##_user CONTAINER=ALL;
```

- Revoke a local privilege locally.

```
SQL> REVOKE alter user FROM local_user;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The syntax and behavior of the GRANT and REVOKE are mostly unchanged. The syntax has been extended with the CONTAINER clause so that users with the proper privileges can grant privileges commonly or locally.

When a user grants a privilege with the CONTAINER=ALL clause, the privilege becomes a common privilege. The user must have the SET CONTAINER privilege for all the PDBs because he or she grants the privilege to the same common user in each container.

When a user grants a privilege with the CONTAINER=CURRENT clause, the privilege becomes a local privilege.

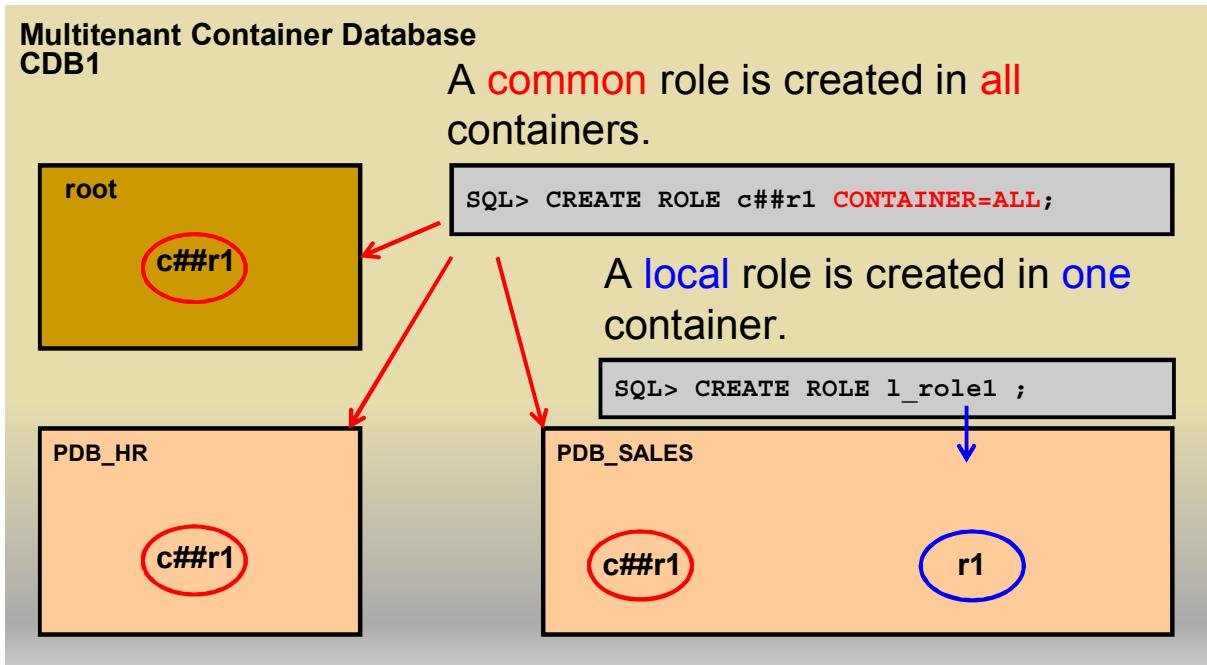
To grant a system privilege, the grantor must have the GRANT ANY PRIVILEGE system privilege or have been granted the privilege WITH ADMIN OPTION.

To grant an object privilege, the grantor must be the owner of the object, have the GRANT ANY OBJECT PRIVILEGE, or have the object privilege WITH GRANT OPTION.

The command to revoke privileges follow the same rules as grants. A common role cannot be revoked from a common user without the CONTAINER=ALL clause.

Object privilege grants track the grantor. If the grantor's privilege has the WITH GRANT OPTION, and is revoked, so are the grants that the grantor made.

Creating Common and Local Roles



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A role created across all containers is a **common** role. In the example of the slide, the `c##r1` role is created commonly using the clause `CONTAINER=ALL`. The `CREATE` operation is replicated in all containers. Consequently, the same role `c##r1` is created in each container.

A role created in a specific PDB is **local**. In the example in the slide, the `l_role1` role is created locally. The `CREATE` operation is not replicated in all containers. Consequently, the role `l_role1` is created in the `PDB_SALES` container only.

Creating Common and Local Roles

- A local user can create local roles.

```
SQL> CREATE ROLE L_HR CONTAINER=CURRENT;
```

- A common user can create common roles or local roles.

```
SQL> CREATE ROLE C##_R1 CONTAINER=ALL;
```

- Local roles can be granted to local or common users.
- Common roles can be granted to local or common users.
- Local roles can be granted to common roles.
- Common roles can be granted to local roles.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

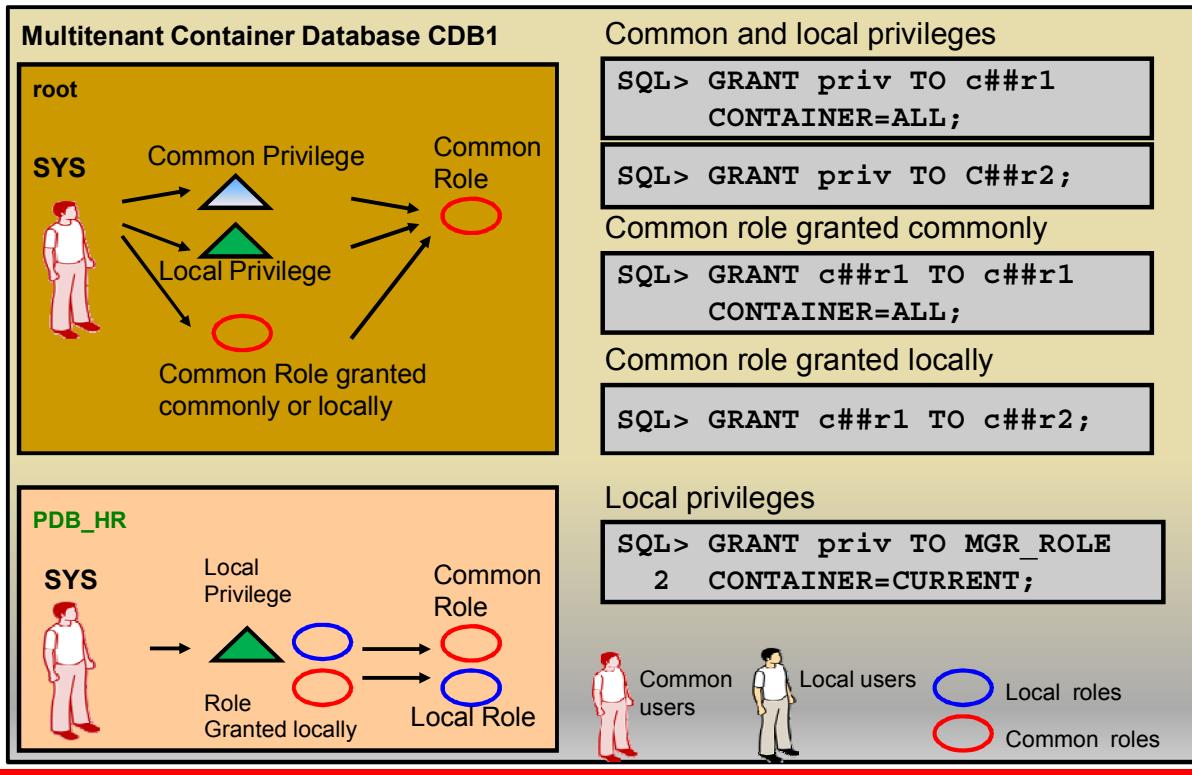
The syntax and privileges are the same as in a non-CDB for a local user to create a local role.

A user can create common roles if the user has the CREATE ROLE privilege, and SET CONTAINER for all PDBs. A common user can create a local role if the user has the CREATE ROLE and SET CONTAINER privileges for that PDB. The CONTAINER clause determines whether the role is common or local. A common role must begin with C## characters.

Any role can be granted to any role or user. It does not matter whether the user or role is defined to be local or common. Consider that a role is a container for privileges. When a role is granted to a user, the privileges in the role are limited to the context of the PDB.

For example, the common role C##_R1 is created with a common privilege CREATE SESSION. When the role C##_R1 is granted to a common user, C##_TEST, that user can connect to any PDB. But when C##_R1 is granted to a local user, lu_PDB1, defined in PDB_HR, that local user can only connect to PDB_HR.

Granting Common or Local Privileges / Roles to Roles



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

To grant a role, the grantor must have the GRANT ANY ROLE privilege or have been granted the role WITH ADMIN OPTION.

Privileges assigned to a common role are common privileges as long as CONTAINER=ALL is specified and, therefore, are applied in all containers. If CONTAINER=ALL is not specified, the privilege assigned to a common role is local. The first command shown in the slide shows a privilege granted commonly to a common role and, therefore, usable in each of the PDBs. The second command shown in the slide shows a privilege granted locally to a common role and, therefore, usable only in the PDB where the command is executed. Common roles can be created in the root container only.

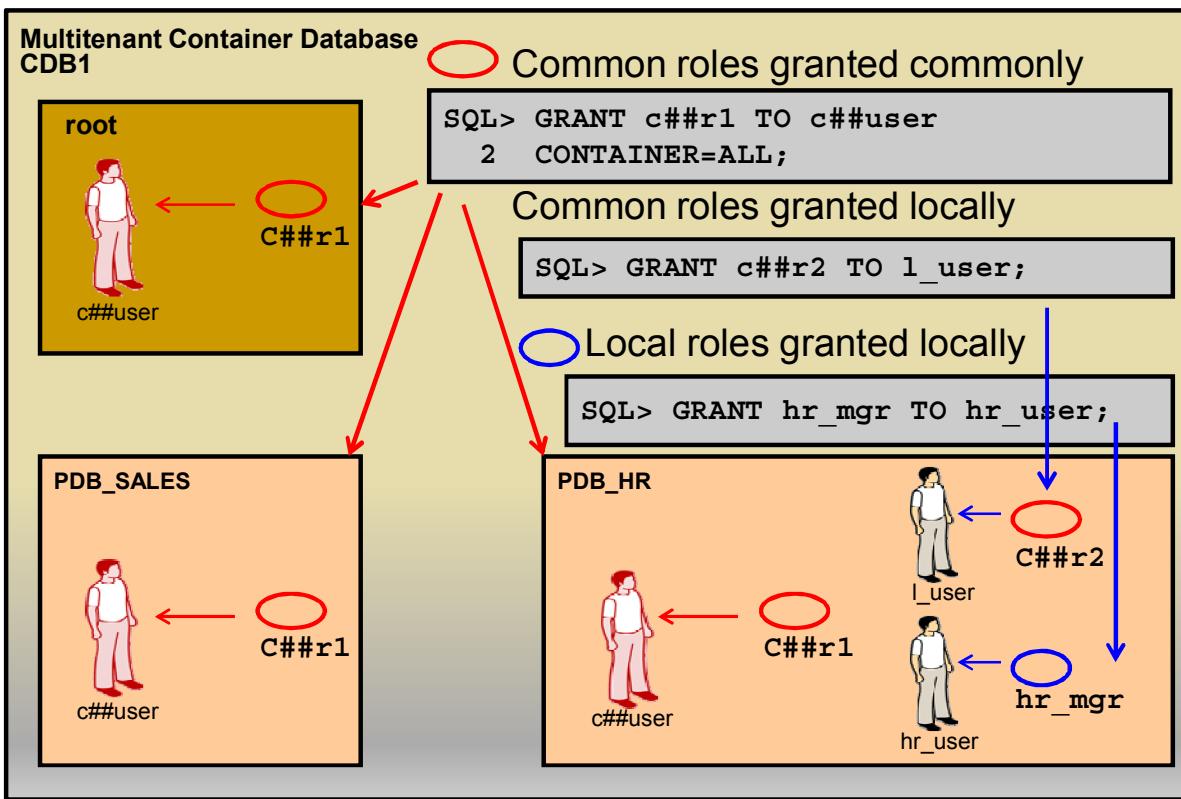
Example: A common role, C##_VIEWER, granted commonly the SELECT ANY TABLE privilege is granted to a common user C##_HR_CLERK. The SELECT ANY TABLE privilege can be used by C##_HR_CLERK while connected to any PDB.

Privileges assigned to local roles can only be local privileges. A local role can be created in a specific PDB and cannot be created in the root container. The fifth command executed in the PDB_HR container shows a privilege being granted locally to the MGR_ROLE role.

Common roles may be granted to any user or role commonly or locally in the root. Local and common roles can be granted to any user or role locally only in any PDB.

A local user can be assigned a common role, and the privileges assigned by that role can only be exercised in the PDB where the local user is defined.

Granting Common and Local Roles to Users



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Common roles may be granted to any user or role in any PDB. Local roles may be granted to any user or role that exists in the same PDB. In the first example, the common role , C##R1, is granted to the common user C##USER. This will grant the role in all the containers.

A local role can be granted to a common user and a common role to a local user. A privilege granted directly or through a role is exercised in the context of a single PDB; for example, a common role C##_VIEWER, containing SELECT ANY TABLE privilege is granted to a common user C##_HR_CLERK. The SELECT ANY TABLE privilege can be used by C##_HR_CLERK while connected to any PDB. A local role LR_HR in the PDB_HR PDB has the INSERT INTO HR.EMPLOYEES privilege and is granted to C##_HR_CLERK. C##_HR_CLERK can use the INSERT INTO HR.EMPLOYEES only while connected to PDB_HR.

A local user can be assigned a common role, and as before the privileges assigned by that role can only be exercised in the PDB where the local user is defined. If the local role LR_HR is granted to the C##USER, the C##USER would be able to exercise the INSERT INTO HR.EMPLOYEES privilege only while connected to PDB_HR.

Granting and Revoking Roles

- Grant a common role commonly.

```
SQL> GRANT c##_role TO c##_user CONTAINER=ALL;
```

- Grant a local role to a local user locally.

```
SQL> GRANT local_role TO local_user CONTAINER=CURRENT;
```

- Grant a local role to a common user locally.

```
SQL> GRANT l_role TO c##_user CONTAINER=CURRENT;
```

- Grant a common role to a common user locally.

```
SQL> GRANT c##_role TO c##_user CONTAINER=CURRENT;
```

- Revoke a common role commonly.

```
SQL> REVOKE c##_role FROM c##_user CONTAINER=ALL;
```

- Revoke a local or common role locally.

```
SQL> REVOKE local_role FROM local_user;
```

```
SQL> REVOKE c##_role   FROM c##_user  
CONTAINER=CURRENT;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The syntax and behavior of the GRANT and REVOKE are mostly unchanged. The syntax has been extended with the CONTAINER clause so that common and local roles can be granted or revoked commonly or locally.

To grant a role, the grantor must have the GRANT ANY ROLE privilege or have been granted the role WITH ADMIN OPTION.

A common role can be revoked locally from a user or role with the CONTAINER=CURRENT clause or commonly with the CONTAINER=ALL clause.

Creating Shared and Nonshared Objects

- Nonshared tables can be created by common and local users.
- Shared tables cannot be created by common users created by the customer.
- Shared tables can be created by Oracle-supplied users.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Nonshared tables and other nonshared database objects may be created in a PDB by either common or local users. Shared tables can be created by Oracle-supplied common users. Common users created by a user being granted the `CREATE USER` privilege cannot create shared tables or objects.

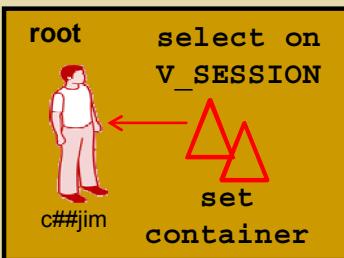
There are two types of shared objects, Object Link, and Metadata Link.

- Object Links point to objects in the root and are used to share data across containers. An example of these Object Link objects is AWR objects.
- Metadata Links share metadata with objects in the root but have private copies of data. An example of these Metadata Link objects is DBA_xxx views.

You can find the type of sharing in the `SHARING` column of `DBA_OBJECTS`.

Enabling Common Users to Access Data in Specific PDBs

CDB1



- Session1 SYS in root switching to PDB

```
SQL> CONNECT / AS SYSDBA
SQL> ALTER SESSION SET CONTAINER = pdb_hr;
```

- Session2 SYS , but session2 C##JIM

USERNAME	CON_ID	V\$SESSION	USERNAME	CON_ID
SYS	1		SYS	1
SYS	6		DBSNMP	1
DBSNMP	1			

- Enable common users to access data about specific PDBs

```
SQL> ALTER USER c##jim
2 SET CONTAINER_DATA = (CDB$ROOT, PDB_HR,
3 PDB2_2) FOR V$SESSION CONTAINER=CURRENT;
```

- Session2 C##JIM views all rows

USERNAME	CON_ID
SYS	1
SYS	6
DBSNMP	1

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Common users can view information about root and about the entire CDB by querying a set of Oracle-supplied tables or views that are created as container data objects. A common user connected to root can see metadata pertaining to PDBs by way of the container data objects (for example, CDB views and V\$ views) in root, provided that the common user has been granted privileges required to access these views and his or her `CONTAINER_DATA` attribute has been set to allow seeing data about various PDBs.

By default, common users cannot view information about specific PDBs. You can control common users' ability to see data pertaining to specific PDBs. For example, you can restrict the information that a common user can see by querying V\$, GV\$, CDB_, and Automatic Workload Repository DBA_HIST* views for information that pertains to one or more specific containers. This is useful in cases where you do not want to expose sensitive information about other PDBs.

In the example in the slide, the common user `C##jim` although granted the `SELECT` privilege on `V$SESSION` view cannot view all sessions of all PDBs when selecting from `V$SESSION` view. For this purpose, set the user attribute `CONTAINER_DATA` to the list of the containers for which he is allowed to view data from `V$SESSION` view.

Finding Information About CONTAINER_DATA Attributes

Find information about the default (user-level) and object-specific CONTAINER_DATA attributes that are explicitly set to a value other than DEFAULT.

USERNAME	DEFAULT	OBJECT_NAME	ALL	CONTAINER_	CON_ID
C##JIM	N	V\$SESSION	N	PDB_HR	1
C##JIM	N	V\$SESSION	N	CDB\$ROOT	1
C##JIM	N	V\$SESSION	N	PDB2_2	1
SYSTEM	Y		Y		1
DBSNMP	Y		Y		1
SYSBACKUP	Y		Y		1
SYS	Y		Y		1

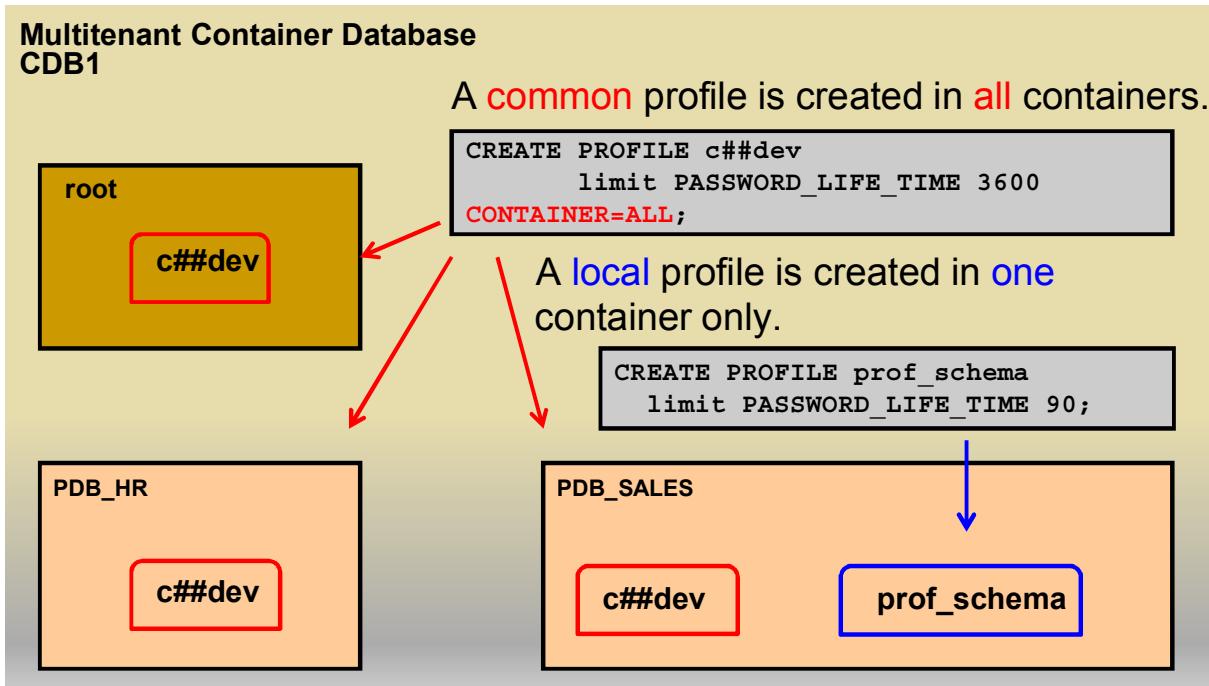


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To find information about the default (user-level) and object-specific CONTAINER_DATA attributes that are explicitly set to a value other than DEFAULT, you can query the DBA_CONTAINER_DATA data dictionary view where:

- USERNAME is the name of the user whose attribute is described by this row
- DEFAULT_ATTR is an indicator of whether the attribute is a default attribute
- OWNER is the name of the object owner if the attribute is object specific
- OBJECT_NAME is the name of the object if the attribute is object specific
- ALL_CONTAINERS is an indicator of whether this attribute applies to all containers
- CONTAINER_NAME is the name of a container included in this attribute if it does not apply to all containers

Creating Common and Local Profiles



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A profile created across all containers is a common profile. In the example in the slide, the `c##dev` profile is created commonly using the clause `CONTAINER=ALL`. The `CREATE` operation is replicated in all containers. Consequently, the same profile `c##dev` is created in each container.

A profile created in a specific PDB is local. In the example in the slide, the `prof_schema` profile is created locally. The `CREATE` operation is not replicated in all containers. Consequently, the profile `prof_schema` is created in the `PDB_SALES` container only.

Creating Common and Local Profiles

- A local user can create local profiles.

```
SQL> CREATE PROFILE PROF_SCHEMA  
  2      limit PASSWORD_LIFE_TIME 900  
  3  CONTAINER=CURRENT;
```

- A common user can create common or local profiles.

```
SQL> CREATE PROFILE C##_DEVELOPER  
  2      limit PASSWORD_LIFE_TIME 3600  
  3  CONTAINER=ALL;
```

- Local profiles can be assigned to local or common users.
- Common profiles can be assigned to local or common users.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

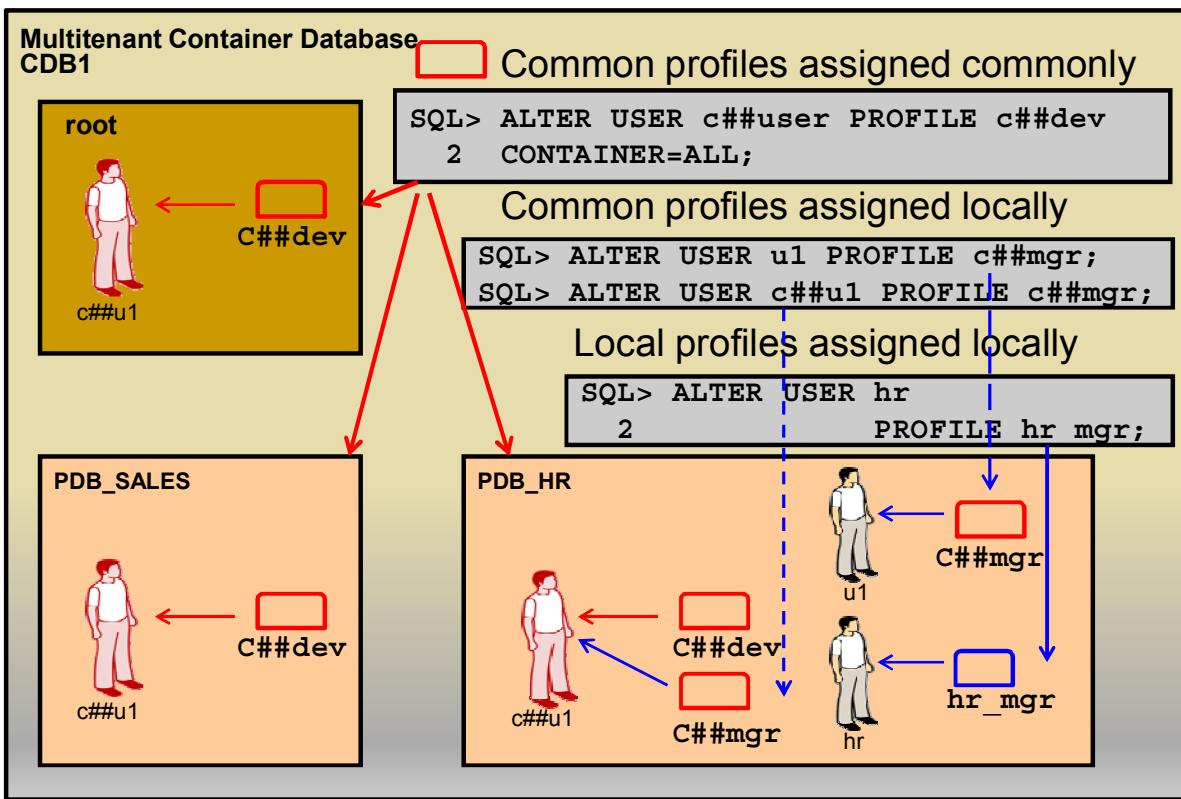
The syntax and the DEFAULT profile are the same as in a non-CDB for a local user to create a local profile.

A user can create common profile if the user has the CREATE PROFILE privilege, and SET CONTAINER for all PDBs. A common user can create a local profile if the user has the CREATE PROFILE and SET CONTAINER privileges for that PDB. The CONTAINER clause determines whether the profile is common or local. A common profile must begin with C## characters.

Any profile can be assigned to any user. It does not matter whether the user is defined to be local or common. Consider that a profile is a container for limits. When a profile is assigned to a user, the limits in the profile are limited to the context of the PDB.

For example the common profile C##_DEVELOPER is created with a limit PASSWORD_FILE_TIME. When the profile C##_DEVELOPER is assigned to a common user, C##_TEST, that user's password will have a life time to 3600 days in any PDB. But when C##_DEVELOPER is assigned to a local user, lu_PDB1, defined in PDB_HR that local user can only connect to PDB_HR during 900 days without changing his password.

Assigning Common and Local Profiles to Users



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Common profiles may be assigned to any user in any PDB. Local profiles may be assigned to any user that exists in the same PDB. In the first example, the common profile, `C##dev`, is assigned to the common user `C##USER`. This will assign the profile in all the containers.

A local profile can be assigned to a common user or to a local user. Example: A local profile `MGR`, owning a `PASSWORD_LIFE_TIME` limit of 90 days is assigned to a common user `C##_HR_USER` in a specific PDB. The `PASSWORD_LIFE_TIME` limit is set to 90 days for `C##_HR_USER` while connected to this PDB.

A local user can be assigned a common profile, and as before, the limits of the profile assigned can only be exercised in the PDB where the local user is defined.

Assigning Profiles

- Assign a common profile commonly.

```
SQL> ALTER USER c##user PROFILE c##prof CONTAINER=ALL;
```

- Assign a local profile to a local user locally.

```
SQL> ALTER USER jim PROFILE developer;
```

- Assign a local profile to a local user locally.

```
SQL> ALTER USER c##user PROFILE developer;
```

- Assign the local profile DEFAULT commonly.

```
SQL> ALTER USER c##user PROFILE DEFAULT CONTAINER=ALL;
```

- Assign the local profile DEFAULT locally.

```
SQL> ALTER USER jim PROFILE DEFAULT  
2 CONTAINER=CURRENT;
```

- Modify the limits of the profile DEFAULT locally only.

```
SQL> ALTER PROFILE DEFAULT LIMIT password_file_time 1;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The syntax and behavior of the `ALTER USER` to assign a profile are mostly unchanged. The syntax has been extended with the `CONTAINER` clause so that profiles can be assigned commonly or locally.

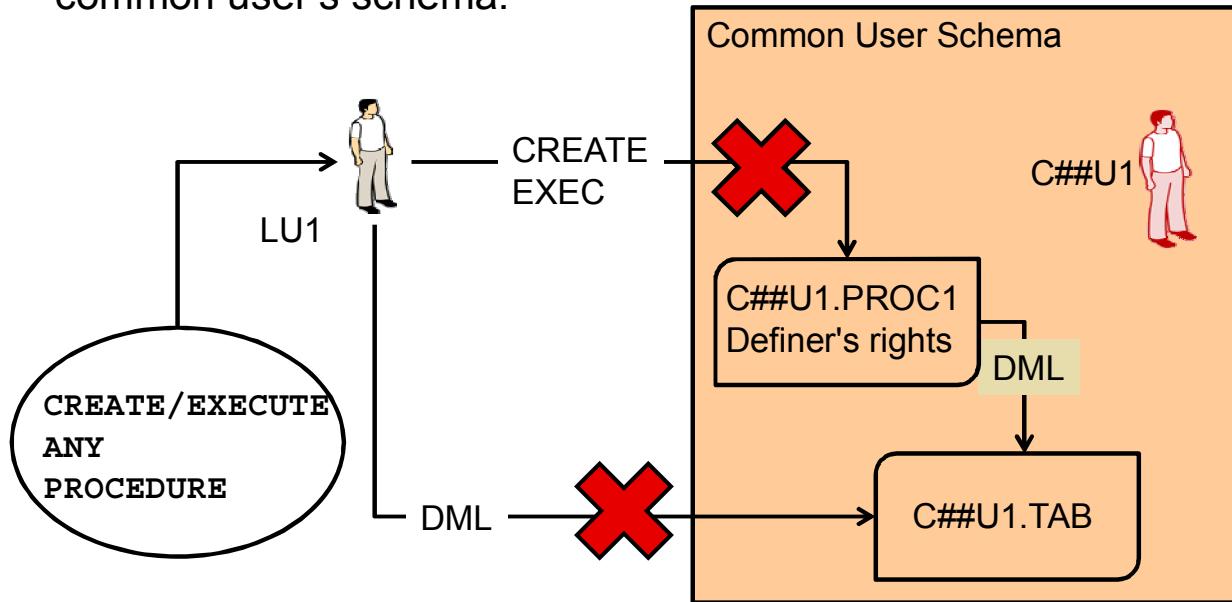
To assign a profile, the user performing the operation must have the `ALTER USER` privilege.

The `DEFAULT` profile is a local profile. The limits of the `DEFAULT` local profile can be modified only in a PDB, not from the root.

```
SQL> alter profile default limit password_reuse_max 100  
container=ALL;  
alter profile default limit password_reuse_max 100 container=ALL  
*  
ERROR at line 1:  
ORA-65142: A local profile can be altered only within the current  
container
```

Restriction on Definer's Rights

A local user **cannot** exercise local system privileges on a common user's schema.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A user cannot exercise local system privileges on a common user's schema; for example, suppose a local user does not have privileges to perform DML on a table in a common users schema, but does have CREATE ANY PROCEDURE and EXECUTE ANY PROCEDURE privileges. The local user is not allowed to create a definer's rights PL/SQL procedure that can issue statements as the common user against the common user's schema and then proceed to execute that procedure to get around restrictions imposed on local users.

Quiz

You can create roles in PDBs. Which statement is true about common and local roles?

- a. You can create local roles in the root.
- b. You can create common roles only when connected in the root.
- c. You can create common roles in PDBs.
- d. You can create common and local roles in seed.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

Quiz

Choose all that apply to make a true statement. In a PDB, a common user and a local user can:

- a. Have the same name
- b. Be granted the same privileges
- c. Be granted both local and common roles
- d. Open and close the PDB if granted the appropriate privilege



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b, c, d

Summary

In this lesson, you should have learned how to:

- Manage common and local users
- Manage common and local roles
- Manage common and local privileges
- Manage the CONTAINER_DATA attributes of common users
- Manage common and local profiles



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 6 Overview: Managing Users, Roles, and Privileges in CDB and PDBs

These practices cover the following topics:

- Creating a common user
- Creating a local user
- Granting common and local privileges
- Creating common and local roles
- Creating common and local roles commonly and locally
- Managing the CONTAINER_DATA attributes of common users



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

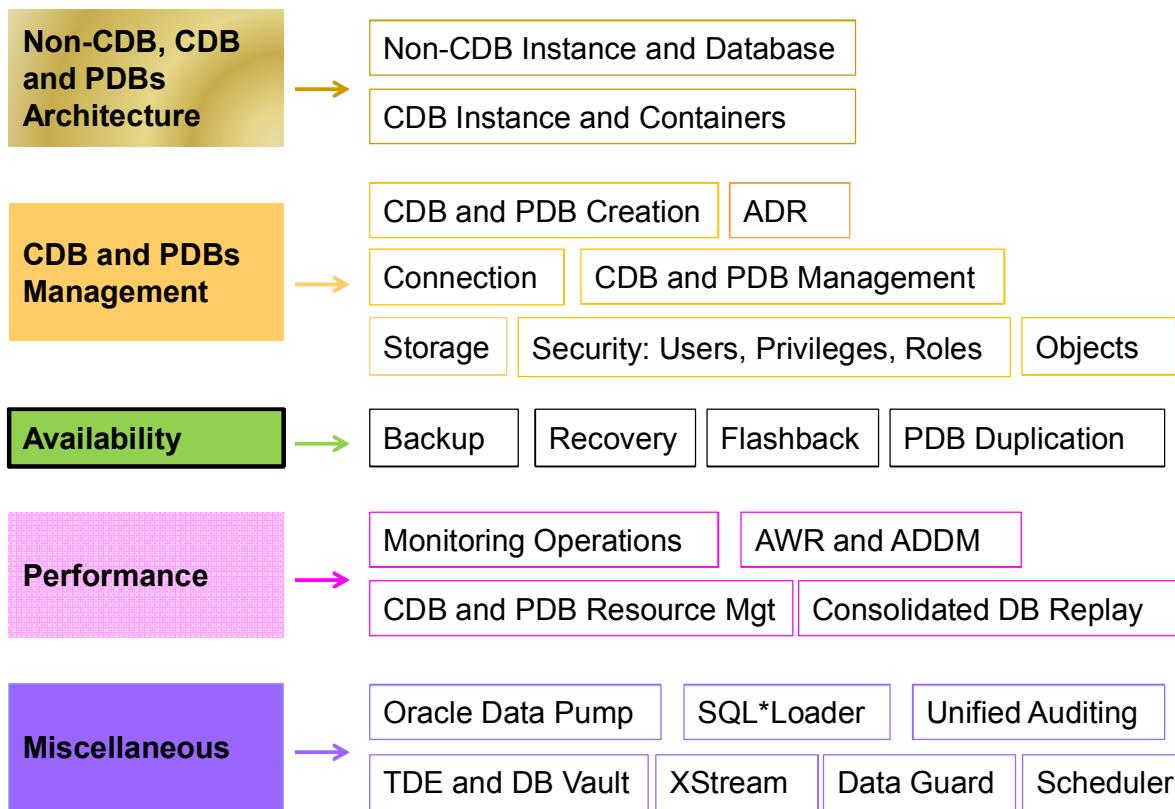
Backup, Recovery, Flashback CDB and PDBs



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Course Structure



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This lesson explains how to:

- Back up a CDB and PDBs
- Recover a CDB and PDBs
- Flash back a CDB
- Duplicate PDBs

Objectives

After completing this lesson, you should be able to:

- Perform CDB and PDB backups
- Use RMAN backups to plug unplugged PDBs
- Recover CDBs from essential files loss
- Recover PDBs from PDB datafiles loss
- Perform flashback database
- Duplicate PDBs
- Validate CDBs and PDBs



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Note: For a complete understanding of Oracle PDB new feature and usage, refer to the following guides in the Oracle documentation:

- *Oracle Database Administrator's Guide 12c Release 1 (12.1)*
- *Oracle Database Backup and Recovery User's Guide 12c Release 1 (12.1)*

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

- *Oracle Database 12c New Features Demo Series* demonstrations:
 - *How to Backup and Recover CDB*
- *Oracle By Example (OBE)*:
 - *Performing a Point-In-Time Recovery for a Pluggable Database*

Any database, non-CDB or CDB, needs to be backed up in case a potential recovery is required.

The ARCHIVELOG mode can be set only at CDB level and not at PDB level due to redo log files sharing.

With a non-CDB, you can perform whole and partial non-CDB backup and on another level, hot tablespace or datafile backups.

With CDBs, you can still perform the same types of backups. The new level for backing up data is the PDB level.

The granularity of media recovery is very flexible and can be done for the entire CDB, for a PDB, for a tablespace, for a data file, or even for a block.

You can perform different types of recovery when a failure occurs:

- What to do if an instance failure occurs?
- What to do if a PDB tempfile is missing?
- How to proceed if a controlfile is missing or corrupted?
- What to do if a root datafile is missing or corrupted?
- How to proceed if a PDB datafile is missing or corrupted?
- Do you use flashback database if a local schema is dropped or is there another type of flashback available at the PDB level?

New Syntax and Clauses in RMAN

```
$ export ORACLE_SID=cdb1
$ rman TARGET / ← → $ rman TARGET jim@pdb1
```

- DATABASE keyword operates on all PDBs and root.

```
RMAN> BACKUP DATABASE;
RMAN> RECOVER DATABASE;
```

- PDB operates on individual PDBs.

```
RMAN> BACKUP PLUGGABLE DATABASE hr_pdb, sales_pdb;
RMAN> RECOVER PLUGGABLE DATABASE hr_pdb;
```

- Back up, restore, recover the root using CDB\$ROOT keyword.

```
RMAN> BACKUP PLUGGABLE DATABASE "CDB$ROOT";
```

- Qualify tablespace of PDB with PDB name.

```
RMAN> BACKUP TABLESPACE sales_pdb:tbs2;
RMAN> RESTORE TABLESPACE system;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use Recovery Manager (RMAN) or Enterprise Manager to back up and recover entire CDBs, partial CDBs, individual whole PDBs, or partial PDBs such as tablespace/datafile of specific PDBs.

The CDB or PDBs are the possible targets for RMAN; an individual PDB is a valid RMAN TARGET database. Connect to the root or a PDB as a user with SYSDBA or SYSBACKUP privilege .

The traditional syntax, such as BACKUP DATABASE, RESTORE DATABASE, and RECOVER DATABASE, operates on the root and all its PDBs and, therefore, on the whole CDB, or on a single PDB depending on the connection.

A new syntax, namely PDB, is introduced to allow backup, restore, recover single or several PDBs.

To back up, restore, or recover, the root, CDB\$ROOT, must be used.

The RMAN TABLESPACE syntax can be qualified with the PDB name so that a user can specify the name of the tablespace as it is known to PDB during backup, restore, and recover commands. If no PDB qualifier is used, then root is used by default. To list the tablespaces and their associated PDB, use the REPORT SCHEMA syntax.

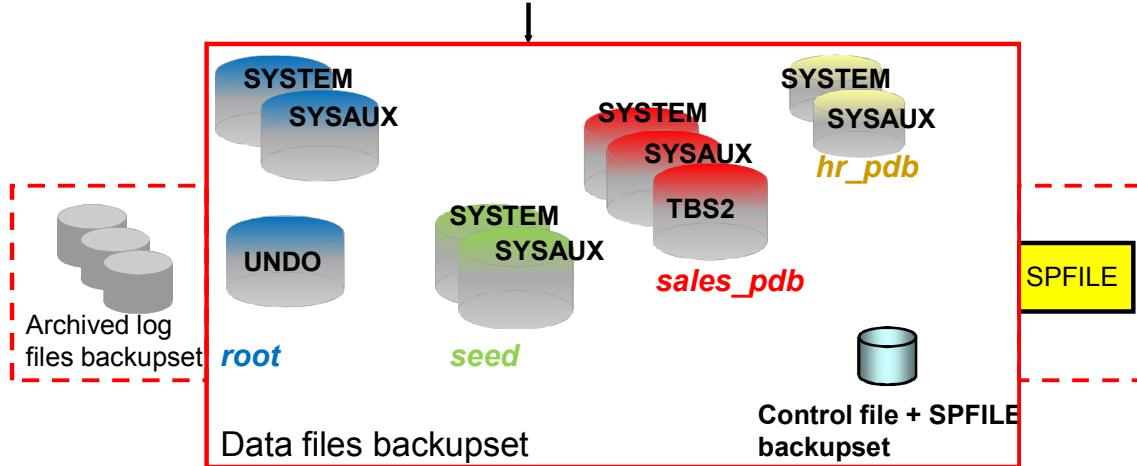
The same new clauses can be applied to DUPLICATE, SWITCH, REPORT, CONVERT, CHANGE, LIST, DELETE.

The root needs to be available for recovery, backup, and RMAN backups.

CDB Backup: Whole CDB Backup

Back up all PDBs data files and root files.

```
RMAN> CONNECT TARGET /
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;
RMAN> BACKUP DATABASE PLUS ARCHIVELOG;
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

RMAN can back up the entire CDB and contained or individual PDBs. In addition, individual tablespaces or data files can be backed up from a specific PDB.

A whole database backup of a CDB can be, in a similar way to non-CDBs, either backup sets or image copies of the entire set of data files, namely the root data files and all PDBs data files, and the control file. You can optionally include the server parameter file (SPFILE) and archived redo log files.

Using RMAN to make an image copy of all the CDB files simply requires mounting or opening the CDB, starting RMAN, connecting to the root with SYSDBA or SYSBACKUP privilege, and entering the BACKUP command shown in the slide. Optionally, you can supply the DELETE INPUT option when backing up archivelog files.

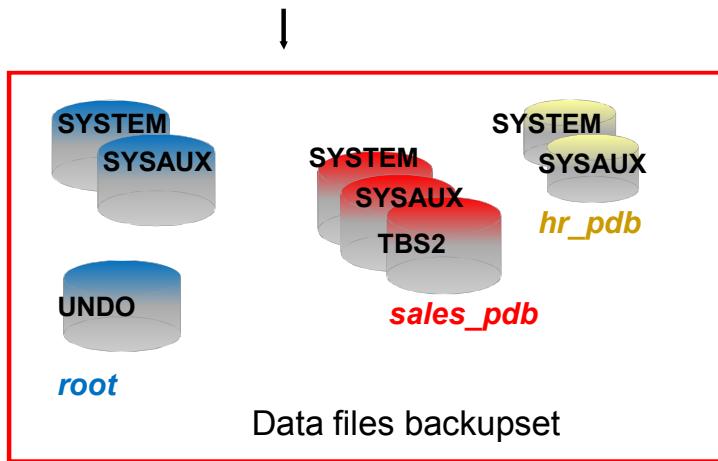
You can also create a backup (either a backup set or image copies) of previous image copies of all data files and control files in the CDB by using the following command:

```
RMAN> BACKUP COPY OF DATABASE;
```

CDB Backup: User-Managed Hot CDB Backup

Perform a user-managed hot CDB backup.

```
SQL> CONNECT / AS SYSDBA
SQL> ALTER DATABASE BEGIN BACKUP;
SQL> !cp datafiles /backup_dir
SQL> ALTER DATABASE END BACKUP;
```



ORACLE

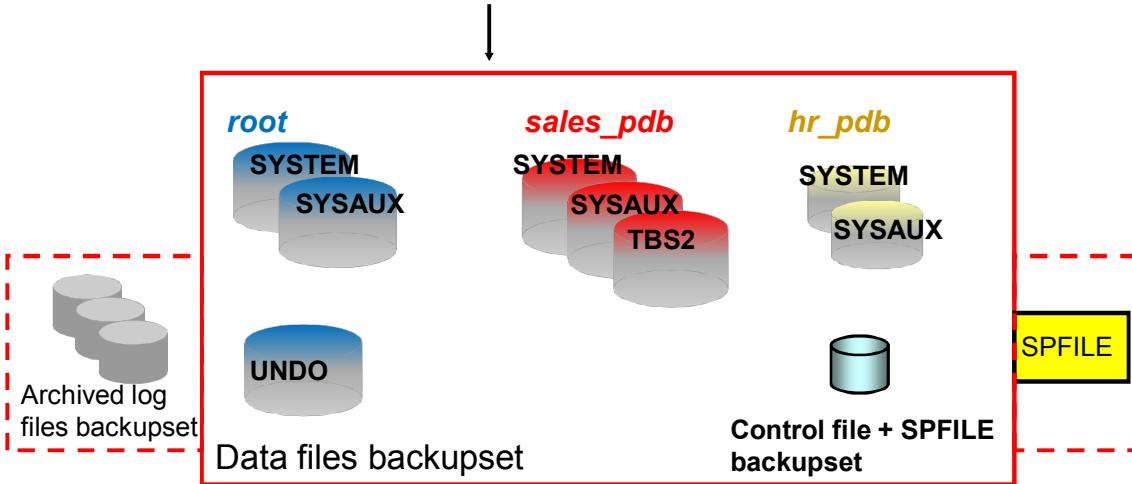
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

After setting the whole CDB in backup mode, all data files except those of the seed PDB are placed in ACTIVE mode. This mode allows you to back up the data files with an O/S command. The data files of the read-only seed container can be backed up any time because they are not subject to any write operation.

CDB Backup: Partial CDB Backup

Back up the root and/or individual PDBs.

```
RMAN> CONNECT TARGET /
RMAN> BACKUP PLUGGABLE DATABASE "CDB$ROOT", sales_pdb;
RMAN> BACKUP PLUGGABLE DATABASE hr_pdb PLUS ARCHIVELOG;
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A partial CDB backup backs up the entire set of data files of the root, all datafiles of defined PDBs, and the control file and SPFILE because it has been configured to be backed up automatically after each backup.

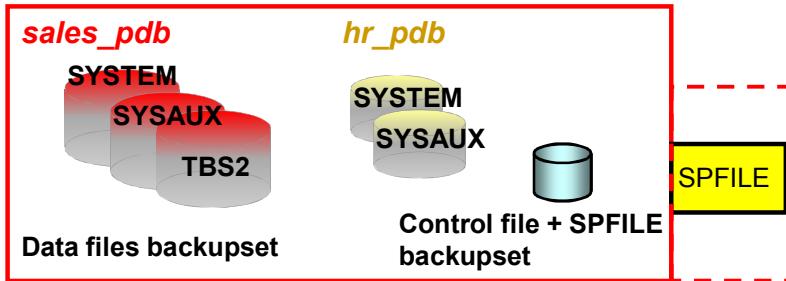
The command `BACKUP PDB "CDB$ROOT" , sales_pdb` backs up all datafiles of the root container, namely the SYSTEM, SYSAUX, and UNDO datafiles, and then all datafiles of the `sales_pdb` PDB, namely the SYSTEM, SYSAUX, and TBS2 datafiles.

PDB Backup: Whole PDB Backup

- Back up whole PDBs with RMAN:

```
RMAN> CONNECT TARGET /
RMAN> BACKUP PLUGGABLE DATABASE sales_pdb;
```

```
RMAN> BACKUP PLUGGABLE DATABASE sales_pdb, hr_pdb;
```



- Perform a user-managed hot PDB backup:

```
SQL> ALTER PLUGGABLE DATABASE sales_pdb BEGIN BACKUP;
SQL> !cp pdb_datafiles /backup_dir
SQL> ALTER PLUGGABLE DATABASE sales_pdb END BACKUP;
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A whole PDB backup backs up the entire set of data files of an individual PDB, and the control file and SPFILE because it has been configured to be backed up automatically after each backup.

In the first example, the command `BACKUP PDB` backs up all datafiles of the `sales_pdb` PDB, namely the `SYSTEM`, `SYSAUX`, and `TBS2` datafiles. You could also connect to the PDB target as follows:

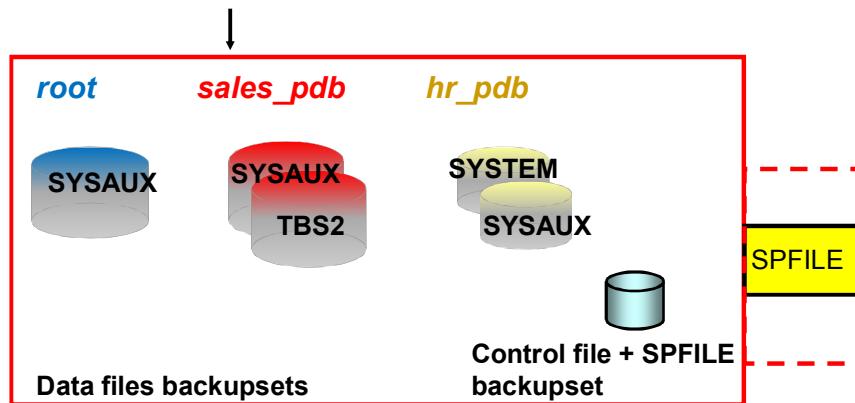
```
$ rman TARGET sys@sales_pdb
```

In the second example, the same command backs up all datafiles of two named PDBs.

You can also perform user-managed hot backups using OS commands. You have to set the PDB in hot backup mode first. To set the PDB in hot backup mode, use SQL*Plus and the new command `ALTER PLUGGABLE DATABASE` with the clauses `BEGIN BACKUP` and `END BACKUP` as shown in the third example.

PDB Backup: Partial PDB Backup

```
RMAN> CONNECT TARGET /
RMAN> REPORT SCHEMA;
RMAN> BACKUP TABLESPACE sales_pdb:tbs2;
RMAN> BACKUP TABLESPACE hr_pdb:system,
          sales_pdb:sysaux;
RMAN> BACKUP TABLESPACE sysaux, hr_pdb:sysaux;
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

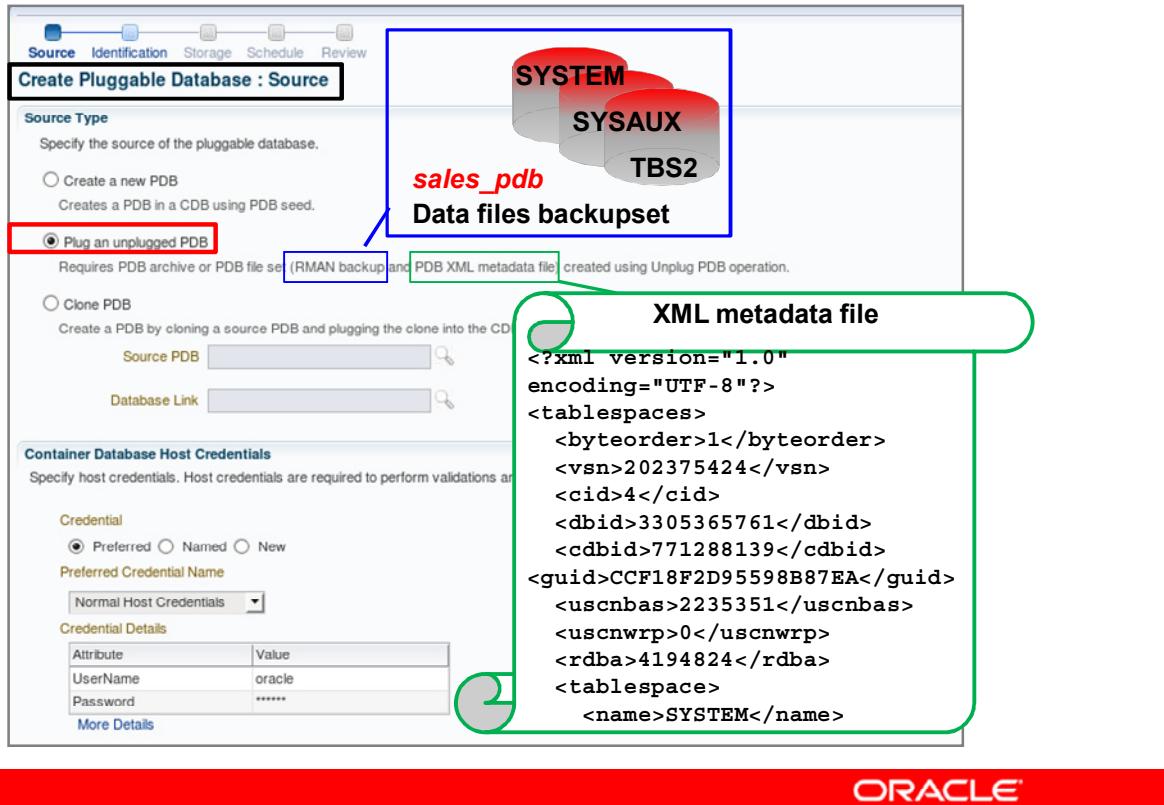
A partial PDB backup backs up the data files named tablespaces of individual PDBs, and the control file and SPFILE because it has been configured to be backed up automatically at each backup performed.

The example uses the command `BACKUP TABLESPACE` to back up all datafiles of tablespace TBS2 of sales_pdb PDB. To find the names of tablespaces within PDBs, use the `REPORT SCHEMA` command.

The second backup uses the same command to back up all datafiles of tablespace SYSTEM of hr_pdb PDB and all datafiles of tablespace SYSAUX of sales_pdb PDB.

The third backup uses the same command to back up all datafiles of tablespace SYSAUX of the root and all datafiles of tablespace SYSAUX of hr_pdb PDB.

Using RMAN Backup to Plug an Unplugged PDB



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Using an RMAN PDB backup, you can create a new pluggable database in a CDB after having unplugged a PDB.

To plug in an unplugged PDB, follow these steps:

- From the Administration menu of the Database page, choose Pluggable Database, then choose Create Pluggable Database.
- On the Source page of the Create Pluggable Database Wizard, select “Plug an unplugged PDB”. This requires either:
 - A PDB archive, which is a compressed TAR file consisting of PDB XML metadata file and all datafiles that belong to PDB. This method is used for transporting the PDB when both the source and target CDB are using a file system for storage. It is not supported for a PDB using Oracle ASM for storage.
 - Or ,
 - A PDB file set that consists of PDB XML metadata file and RMAN backup of PDB. This method is recommended for transporting the PDB when the source or target CDB is using Oracle ASM for storage.
- Enter the Host Credentials for the host. You can choose Preferred, Named, or New credentials, and then click Next. Enterprise Manager displays the Destination page of the wizard.

Recovery

- Instance recovery: **CDB level only**
- Automatic missing tempfile re-creation at **CDB open**
- Complete media recovery after file loss or corruption
 - **CDB mounted:** Same as for non-CDB
 - Redo log files, control files
 - **SYSTEM** root and **PDB** datafiles, UNDO datafiles
 - **PDB opened:** Any PDB datafile other than SYSTEM
 - **Tablespace OFFLINE:** Any other PDB or CDB datafile
- Incomplete media recovery after file loss or corruption
 - CDB mounted: The whole CDB back in time
 - PDB closed: A whole PDB back in time
 - TSPITR for **any tablespace** except SYSTEM, UNDO, SYSAUX
- Block recovery: No change
- **Flashback database:** **CDB mounted**



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In a CDB, the granularity of media recovery is very flexible and can be done for the entire CDB, for a PDB, for a tablespace, for a data file, or even for a block.

Crash and instance recovery is supported for a CDB as a whole, because there is one single instance for the root and all its PDBs. When the instance crashes, the only possible instance recovery is a CDB instance recovery.

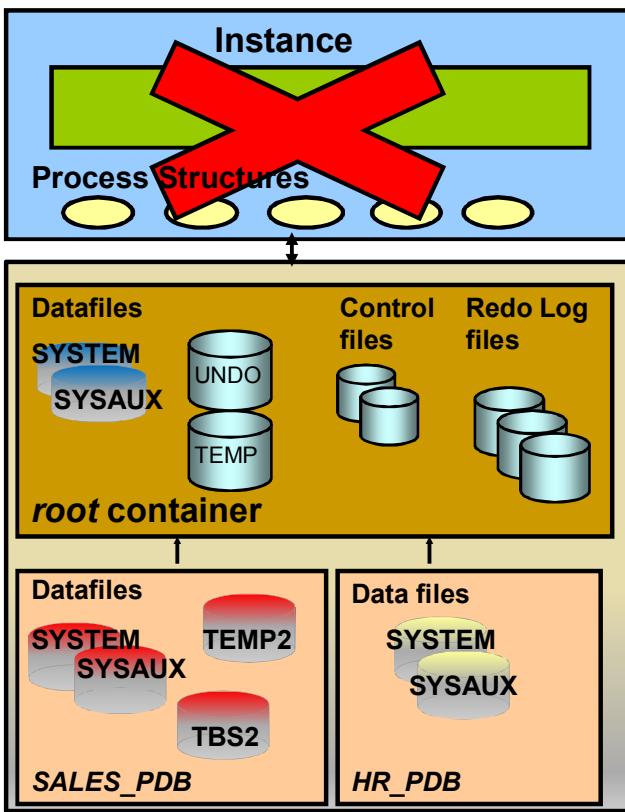
If redo log files or control files, datafiles from the root **SYSTEM** or **UNDO** tablespace are lost, because these files belong to the root; the only possible media recovery is at CDB level.

There exists a new level of media recovery supported, individual PDB media recovery after datafile loss or corruption of a PDB.

There exists a new level of incomplete or PITR recovery, PDB PITR that will look like a tablespace recovery.

Correcting “human error” applying to the root and PDBs requires Flashback at the CDB level only. For example, if you dropped a common or local schema, you must use flashback database applying to all containers, the root and the PDBs.

Instance Failure



PDB instance recovery is **impossible**.

After instance failure:

- Connect to the root
- Open the root
- Open all PDBs with:
 - Triggers

`SQL> STARTUP`

– SQL statement

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Crash-instance recovery is supported for a CDB as a whole, because there is one single instance for the root and all its PDBs.

Redo log files required to perform the instance recovery are stored in the unique set of redo log files of the CDB. There is a single shared redo stream for root and all the PDBs used to perform instance recovery. The instance recovery is performed while opening the root.

There is no way to perform a PDB instance recovery.

For an instance to open a data file, the system change number (SCN) contained in the data file's header must match the current SCN that is stored in the control files.

If the numbers do not match, the instance applies redo data from the online redo logs, sequentially “redoing” transactions until the data files are up-to-date. After all data files have been synchronized with the control files, the root is opened and by default all PDBs are still in mount state.

When redo logs are applied, all transactions are applied to bring the CDB up to the state as of the time of failure. This usually includes transactions that are in progress but have not yet been committed. After the root has been opened, the uncommitted transactions are rolled back on root datafiles. After the PDBs are opened, the uncommitted transactions are rolled back on PDBs' datafiles. When opening a CDB, by default all PDBs remain in mounted state. Either triggers are automatically executed to open the PDBs or you execute the

`ALTER PLUGGABLE DATABASE ALL OPEN` command to open them all.

NOARCHIVELOG Mode

If the database is in NOARCHIVELOG mode, and a data file is lost, perform the following tasks:

- Shut down the instance if it is not already down.
- Restore the entire CDB including all data files and control files.
- Start up the instance and open the CDB and all PDBs.

Users must reenter all changes made since the last backup.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The loss of any data file from a CDB in NOARCHIVELOG mode requires complete restoration of the CDB, including control files and all data files of the root and all PDBs. If you have incremental backups, then you need to perform the restore and recover operations on the CDB.

With the database in NOARCHIVELOG mode, recovery is possible only up to the time of the last backup. So users must reenter all changes made since that backup.

For this type of recovery, proceed with the following steps:

- Shut down the instance if it is not already down.
- Restore the entire CDB including all data and controlfiles from the last backup.
- Start up the instance and open the root and PDBs.

In the following slides, you consider that the CDB is in ARCHIVELOG mode.

Media Failure: CDB or PDB Tempfile Recovery

SQL statements that require temporary space to execute may fail if one of the tempfiles is missing: similar to non-CDB

```
SQL> CONNECT / AS SYSDBA
SQL> select * from DBA_OBJECTS
      2          order by 1,2,3,4,5,6,7,8,9,10,11,12,13;
select * from DBA_OBJECTS order by
1,2,3,4,5,6,7,8,9,10,11,12,13
*
ERROR at line 1:
ORA-01565: error in identifying file
'/u01/app/oracle/oradata/CDB1/temp01.dbf'
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
```

- Automatic re-creation of temporary files at **CDB opening**
- Manual re-creation also possible



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

If a tempfile belonging to the CDB temporary tablespace is lost or damaged, and the user issuing the statement uses it, an error during the execution of SQL statements that require this temporary space occurs.

The SQL statement shown in the slide has a long list of columns to order by, which results in the need for temporary space from the root temporary tablespace. The missing file error is encountered when this statement requiring a sort is executed.

The CDB instance can start up with a missing temporary file. If any of the temporary files do not exist when the CDB instance is started, they are created automatically and the CDB opens normally. When this happens, a message like the following appears in the alert log during startup:

- Re-creating the tempfile /u01/app/oracle/oradata/CDB1/temp01.dbf

You can decide a manual re-creation instead, while connected to root:

```
SQL> ALTER TABLESPACE temp ADD TEMPFILE
      2  '/u01/app/oracle/oradata/CDB1/temp02.dbf' SIZE 20M;
SQL> ALTER TABLESPACE temp DROP TEMPFILE
      3  '/u01/app/oracle/oradata/CDB1/temp01.dbf';
```

Media Failure: PDB Tempfile Recovery

SQL statements that require temporary space to execute may fail if one of the tempfiles is missing.

```
SQL> CONNECT local_user@HR_PDB
SQL> select * from my_table order by
1,2,3,4,5,6,7,8,9,10,11,12,13;
select * from my_table order by
1,2,3,4,5,6,7,8,9,10,11,12,13
*
ERROR at line 1:
ORA-01565: error in identifying file
'/u01/app/oracle/oradata/CDB1/HR_PDB/temp2_01.dbf'
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
```

- Automatic re-creation of temporary files at **CDB opening only**
- Manual re-creation also possible



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If a tempfile belonging to a PDB temporary tablespace is lost or damaged, and the user issuing the statement uses it, an error during the execution of SQL statements that require that temporary space for sorting occurs.

The SQL statement shown in the slide has a long list of columns to order by, which results in the need for temporary space from the PDB temporary tablespace. The missing file error is encountered when this statement requiring a sort is executed.

The PDB can open with a missing temporary file. If any of the temporary files do not exist when the PDB is opened, they are not created automatically. They are automatically re-created at CDB startup.

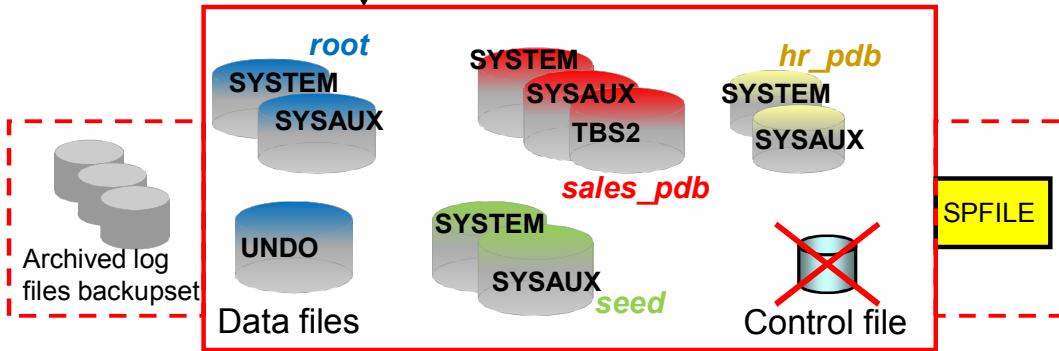
You can perform a manual re-creation instead, while connected to the PDB:

```
SQL> ALTER TABLESPACE temp ADD TEMPFILE
  2  '/u01/app/oracle/oradata/CDB1/HR_PDB/temp2_02.dbf'
  3  SIZE 20M;
SQL> ALTER TABLESPACE temp DROP TEMPFILE
  2  '/u01/app/oracle/oradata/CDB1/HR_PDB/temp2_01.dbf';
```

Media Failure: Control File Loss

Similar to non-CDBs: CDB mounted

```
RMAN> CONNECT TARGET /
RMAN> STARTUP NOMOUNT;
RMAN> RESTORE CONTROLFILE FROM AUTOBACKUP;
RMAN> ALTER DATABASE MOUNT;
RMAN> RECOVER DATABASE;
RMAN> ALTER DATABASE OPEN RESETLOGS;
RMAN> ALTER PLUGGABLE DATABASE ALL OPEN;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If a controlfile is missing or corrupted, because controlfiles belong to the CDB, the instance soon crashes and a whole CDB media recovery is required.

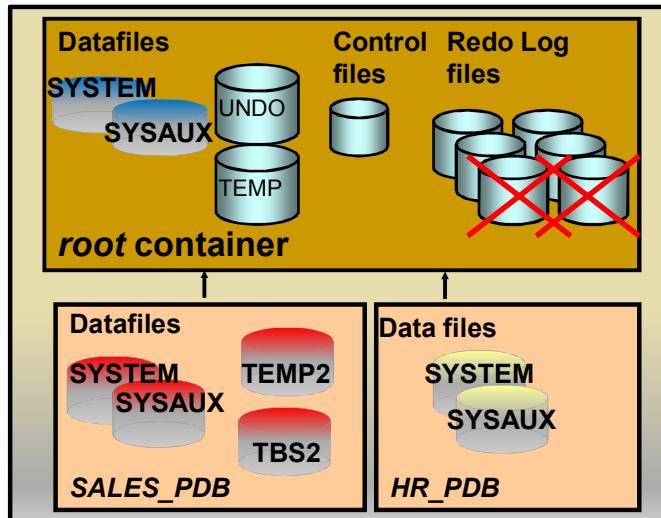
1. First start the CDB instance.
2. Then restore the controlfile from a backup.
3. Mount the CDB.
4. Then recover and open the CDB in resetlogs.

Two possibilities:

```
RMAN> ALTER DATABASE OPEN RESETLOGS;
RMAN> ALTER PLUGGABLE DATABASE ALL OPEN;
Or ,
SQL> ALTER DATABASE OPEN RESETLOGS;
```

Note: With SQL*Plus, the trigger AFTER STARTUP ON DATABASE to open the PDBs is fired. This is not the case with RMAN.

Media Failure: Redo Log File Loss



Similar to non-CDB procedures:

1. Connect to the root container.
2. Examine the STATUS of the lost file: ACTIVE, INACTIVE, CURRENT
3. Proceed as with non-CDBs.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

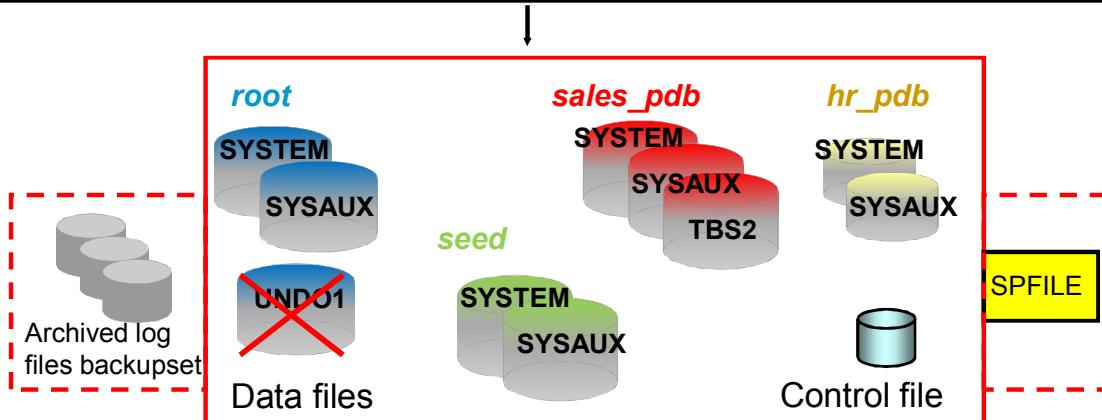
Missing or corrupted redo log files—because there is only one redo stream for the CDB instance (or one redo stream for each instance of a RAC CDB)—require a whole CDB media recovery as explained in the slide.

Depending on whether a whole redo log group or only a redo log member is missing, follow the same procedures as those for non-CDBs.

Media Failure: root SYSTEM or UNDO datafile

Similar to non-CDBs: CDB mounted

```
RMAN> STARTUP MOUNT;
RMAN> RESTORE TABLESPACE und01;
RMAN> RECOVER TABLESPACE und01;
RMAN> ALTER DATABASE OPEN;
RMAN> ALTER PLUGGABLE DATABASE ALL OPEN;
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If the missing or corrupted data file belongs to the root container SYSTEM or UNDO tablespace, then the CDB instance will require shutdown, and a media recovery is required. In a RAC environment, you would shut down all instances of the CDB.

This means that all PDBs will be closed.

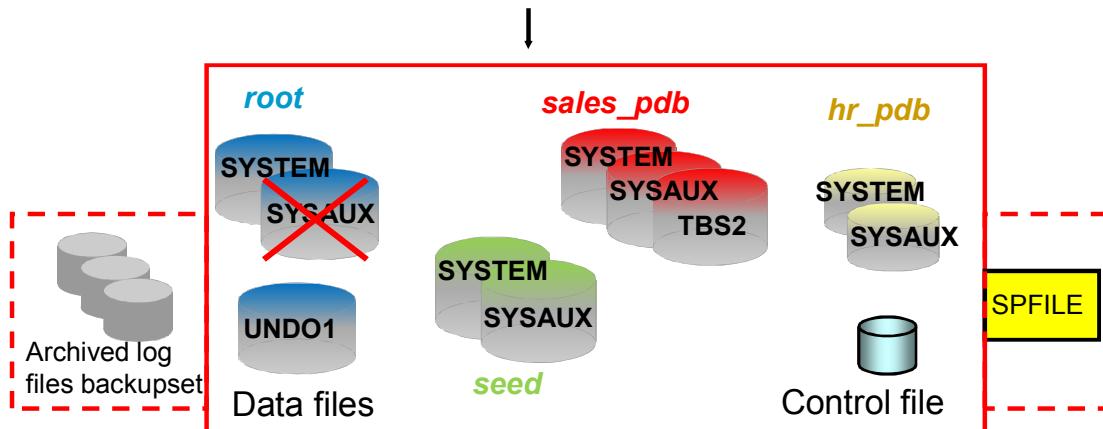
The CDB must be mounted before restoring and recovering the missing root datafile.

After the root datafile is recovered, open the CDB and all PDBs.

Media Failure: root SYSAUX Datafile

Similar to non-CDBs: tablespace OFFLINE

```
RMAN> ALTER TABLESPACE sysaux OFFLINE IMMEDIATE;
RMAN> RESTORE TABLESPACE sysaux;
RMAN> RECOVER TABLESPACE sysaux;
RMAN> ALTER TABLESPACE sysaux ONLINE;
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If the data file that is missing or corrupted is a data file of the root tablespaces other than *SYSTEM* or *UND0*, you offline the tablespace, and then perform a tablespace media recovery.

This means that no PDB is closed.

The CDB remains open while restoring and recovering the missing root datafile.

After the root datafile is recovered, online the tablespace.

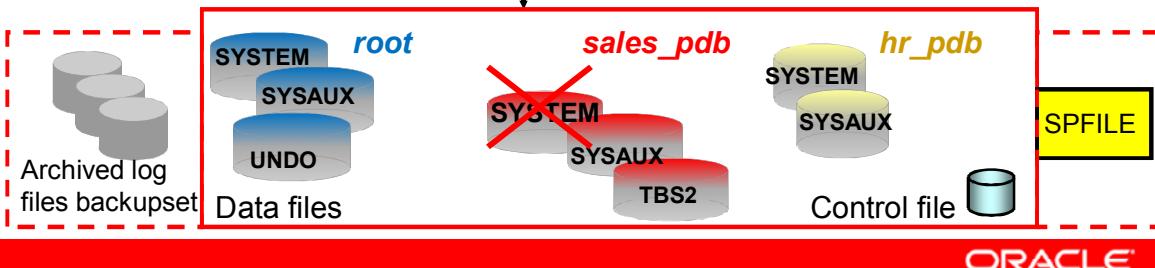
Media Failure: PDB SYSTEM Datafile

- If the PDB is opened, the CDB can only be mounted.
- If the PDB is closed, the CDB can be left opened.
 - Restore and recover the PDB:

```
RMAN> STARTUP MOUNT;           — useless on closed PDB —
RMAN> RESTORE PLUGGABLE DATABASE sales_pdb;
RMAN> RECOVER PLUGGABLE DATABASE sales_pdb;
RMAN> ALTER DATABASE OPEN;    — useless on closed PDB —
RMAN> ALTER PLUGGABLE DATABASE sales_pdb OPEN;
```

- Restore and recover the missing tablespace or data file:

```
RMAN> RESTORE TABLESPACE sales_pdb:system;
RMAN> RECOVER TABLESPACE sales_pdb:system;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If the data file that is missing or corrupted belongs to a PDB and more specifically to the **SYSTEM** tablespace, the CDB must be closed unless the PDB is already closed.

A pluggable database or tablespace or data file media recovery is required before the PDB can be reopened.

If the PDB was closed at the time issue, the users can still work in other PDBs during the PDB recovery.

If the PDB was still open at the time issue, users cannot work at all in any other PDB because the CDB needs to be shut down and mounted.

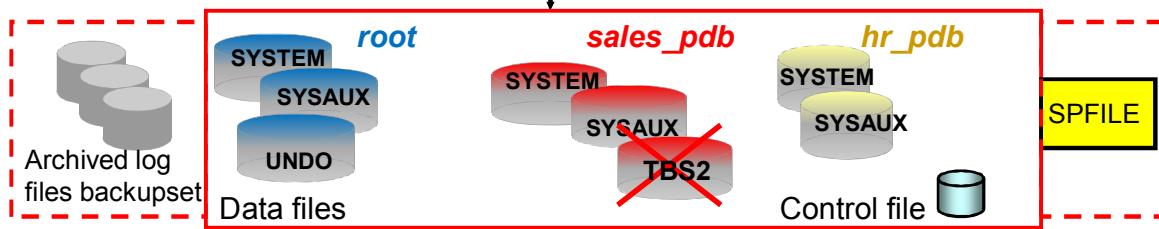
The recovery must be issued from root.

Media Failure: PDB Nonsystem Datafile

Similar to non-CDBs: Perform the recovery within the PDB

- Connect to the PDB.
- Put the tablespace OFFLINE.
- Other PDBs are not impacted.

```
SQL> CONNECT system@sales_pdb
SQL> ALTER TABLESPACE tbs2 OFFLINE IMMEDIATE;
RMAN> CONNECT TARGET /
RMAN> RESTORE TABLESPACE sales_pdb:tbs2;
RMAN> RECOVER TABLESPACE sales_pdb:tbs2;
SQL> ALTER TABLESPACE tbs2 ONLINE;
```



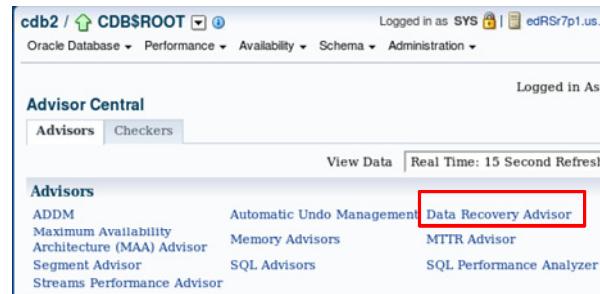
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If the data file that is missing or corrupted belongs to a PDB and more specifically to any tablespace other than SYSTEM tablespace, the PDB need not be closed. The data file with the error is taken OFFLINE IMMEDIATE. Media recovery for that data file is required and performed in a similar way to media recovery of a set of tablespaces. During that recovery time, users can work with other PDB tablespaces and within other PDBs.

The recovery must be issued from the root.

Using Data Recovery Advisor

- Fast detection, analysis, and repair of failures
- Minimizing disruptions for users
- Downtime and runtime failures
- User interfaces:
 - Enterprise Manager (several paths)
 - RMAN command line
- Supported database configurations:
 - Single-instance, CDB, non-CDB
 - Not RAC
 - Supporting failover to standby, but not analysis and repair of standby databases



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The Data Recovery Advisor (DRA) automatically gathers data failure information when an error is encountered. In addition, it can proactively check for failures. In this mode, it can potentially detect and analyze data failures before a database process discovers the corruption and signals an error. (Note that repairs are always under human control.)

Data failures can be very serious. For example, if your current log files are missing, you cannot start your database. Some data failures (such as block corruptions in data files) are not catastrophic, in that they do not take the database down or prevent you from starting the Oracle instance. The Data Recovery Advisor handles both cases: the one when you cannot start up the database (because some required database files are missing, inconsistent, or corrupted) and the one when file corruptions are discovered during run time.

Supported Database Configurations

In the current release, the Data Recovery Advisor supports single-instance CDB and non-CDBs. Oracle Real Application Clusters (RAC) databases are not supported.

Data Failures

Select	Failure Description	Impact	Priority	Status	Time Detected
<input type="checkbox"/>	▼ Data Failures				
<input checked="" type="checkbox"/>	▼ One or more non-system datafiles are missing	See impact for individual child failures	HIGH	OPEN	2012-11-05 09:07:01
<input checked="" type="checkbox"/>	Datafile 34: '/u01/app/oracle/oradata/cdb2/cdata_01.dbf' is missing	Some objects in tablespace CDATA might be unavailable	HIGH	OPEN	2012-11-05 09:06:56
<input checked="" type="checkbox"/>	Datafile 13: '/u01/app/oracle/oradata/cdb2/pdb2_2/CDB2/CDDDD0A5BF67AB8E0436B23B98B87D/datafile/01_mf_sysaux_88w8vyg8_.dbf' is missing	Some objects in tablespace SYSAUX might be unavailable	HIGH	OPEN	2012-11-05 09:08:22

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The Data Recovery Advisor is available from Enterprise Manager Cloud Control. When failures exist, there are several ways to access the Data Recovery Advisor. The following examples all begin on the Database home page:

- Availability > Backup & Recovery > Perform Recovery ... > Advise and Recover > Advise > Continue with Advise... > Continue > Submit Recovery Job
- Performance > Advisors Home > Data Recovery Advisor > Advise > Continue with Advise... > Continue > Submit Recovery Job

Data failures are detected by checks, which are diagnostic procedures that assess the health of the database or its components. Each check can diagnose one or more failures, which are mapped to a repair.

Checks can be reactive or proactive. When an error occurs in the database, “reactive checks” are automatically executed. You can also initiate “proactive checks”, for example, by executing the VALIDATE DATABASE, VALIDATE DATABASE ROOT, VALIDATE PLUGGABLE DATABASE commands.

The example of the slide shows two detected failures about two missing datafiles: one datafile missing in the root container and another one missing in a PDB.

Data Recovery Advisor

RMAN Command-Line Interface

1. List the failures.

```
RMAN> LIST FAILURE DETAIL;
```

2. Get advices for recovery.

```
RMAN> ADVISE FAILURE;
```

3. Preview the recovery script.

```
RMAN> REPAIR FAILURE PREVIEW;
```

4. Apply the recovery script.

```
RMAN> REPAIR FAILURE;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If you suspect or know that a database failure has occurred, then use the `LIST FAILURE` command to obtain information about these failures.

```
RMAN> list failure;
using target database control file instead of recovery
catalog
Database Role: PRIMARY
List of Database Failures
=====
Failure ID Priority Status      Time Detected Summary
-----  -----  -----
8          HIGH    OPEN       05-NOV-12   One or more
non-system datafiles are missing
```

The `ADVISE FAILURE` command displays recommended repair options.

The `REPAIR FAILURE PREVIEW` command is used after an `ADVISE FAILURE` command **within the same RMAN session**. By default, the command uses the single, recommended repair option of the last `ADVISE FAILURE` execution in the current session. The `PREVIEW` clause allows you to view the recovery commands. If you agree with the suggested script, execute the `REPAIR FAILURE` command. After completing the repair, the command closes the failure.

Media Failure: PITR

- PDB PITR

```
RMAN> ALTER PLUGGABLE DATABASE PDB1 CLOSE;
RMAN> RUN {
      SET UNTIL SCN = 1851648 ;
      RESTORE pluggable DATABASE pdb2_1;
      RECOVER pluggable DATABASE pdb2_1
          AUXILIARY DESTINATION='/u01/app/oracle/oradata';
      ALTER PLUGGABLE DATABASE pdb2_1 OPEN RESETLOGS;
    }
```

- PDB Tablespace PITR

```
RMAN> RECOVER TABLESPACE PDB1:TEST_TBS
2>           UNTIL SCN 832972
3>           AUXILIARY DESTINATION '/tmp/CDB1/reco';
RMAN> ALTER TABLESPACE PDB1:TEST_TBS ONLINE;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If you need to recover a PDB database to a point in time in the past beyond flashback retention, then in this case, flashback is not possible, therefore a point-in-time recovery is necessary.

Recovering a PDB to a point in time does not affect all parts of the CDB—the whole CDB is still opened and, therefore, all other PDBs are opened. After recovering a PDB to a specified point in time, when you open the PDB using the `RESETLOGS` option, a new incarnation of the PDB is created. The PDB `RESETLOGS` does not perform a `RESETLOGS` for the CDB.

- A PDB record in the controlfile is updated.
- Each redo log record carries PDB ID in the redo header. This is how recovery knows which redo applies to which PDB. Redo logs are shared by all PDBs—redo from each PDB is written to a single set of redo logs.

Conceptually, a PDB resetlogs is similar to a database resetlogs.

After recovery, the old backup of the PDB remains valid and can be used if a media failure occurs. After restoring/recoverring a PDB to a past point in time, one cannot open the PDB read only. PDB read-write open through resetlogs is required.

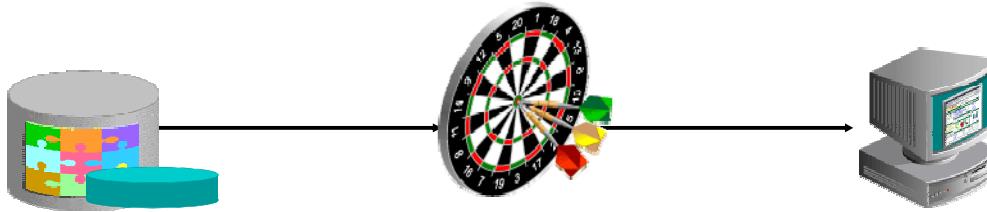
A PDB incarnation is a subincarnation of the CDB. For example, if the CDB is incarnation 5, and a PDB is incarnation 3, then the fully specified incarnation number of the PDB is (5, 3). The initial incarnation of a PDB is 0. To view the incarnation of a PDB, query the `V$PDB_INCARNTION` view.

If you do not use a fast recovery area, you must specify the temporary location of the auxiliary set files by using the `AUXILIARY DESTINATION` clause (example in the slide).

Each PDB has its own set of tablespaces. TSPITR can be used to recover a tablespace to an earlier point in time. Perform the TSPITR as described in the second example in the slide, specifying the full tablespace name including the PDB name.

Recovering a tablespace of a PDB to a point in time does not affect all parts of the CDB—the whole CDB is still opened and, therefore, all PDBs are opened. After recovering a tablespace back in time, put the tablespace back online.

Flashback CDB



1. Configure the FRA.
2. Set the retention target.
3. Enable Flashback Database.

```
SQL> SHUTDOWN IMMEDIATE
SQL> STARTUP MOUNT
SQL> ALTER DATABASE ARCHIVELOG;
SQL> ALTER SYSTEM SET
  2  DB_FLASHBACK_RETENTION_TARGET=2880 SCOPE=BOTH;
SQL> ALTER DATABASE FLASHBACK ON;
SQL> ALTER DATABASE OPEN;
```

If the CDB is in ARCHIVELOG mode, there is no need to restart it.

- No flashback of root without flashing back the whole CDB
- No flashback to a point earlier than the point at which PDBPITR was performed

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can configure Flashback Database for the CDB as you would do for any non-CDB:

1. Configure the Fast Recovery Area.
2. Set the retention target with the `DB_FLASHBACK_RETENTION_TARGET` initialization parameter.
3. Enable Flashback Database with the following command:

```
SQL> ALTER DATABASE FLASHBACK ON;
```

Before you can issue the command to enable Flashback Database, the database must be configured for archiving.

You can disable Flashback Database with the `ALTER DATABASE FLASHBACK OFF` command. As a result, all existing Flashback Database logs are deleted automatically.

Restrictions:

- You cannot flash back the root alone without flashing back the entire CDB.
- Flashback Database operations on a CDB may not be permitted if point-in-time recovery has been performed on any of its PDBs. When point-in-time recovery is performed on a PDB, you cannot directly rewind the CDB to a point that is earlier than the point at which DBPITR for the PDB was performed.

Flashback CDB

A common or local user is dropped.

1. Flashback CDB: CDB mounted in exclusive mode

```
SQL> STARTUP MOUNT  
SQL> FLASHBACK DATABASE TO SCN 53943;
```

2. To review changes: Open CDB and PDBs in READ ONLY.

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

3. To finalize: Flashback again if necessary and open CDB with RESETLOGS.

```
RMAN> SHUTDOWN IMMEDIATE  
RMAN> STARTUP MOUNT  
RMAN> FLASHBACK DATABASE TO SCN 10;  
RMAN> ALTER DATABASE OPEN RESETLOGS;  
RMAN> ALTER PLUGGABLE DATABASE ALL OPEN;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A common schema has been accidentally dropped in the root. You have to flash back the CDB to the time before the common user was dropped impacting all PDBs.

You can use the RMAN FLASHBACK DATABASE command to execute the Flashback Database operation. You can use SEQUENCE and THREAD to specify a redo log sequence number and thread as a lower limit.

Alternatively, you can use the SQL FLASHBACK DATABASE command to return the database to a past time or SCN. If you use the TO SCN clause, you must provide a number. If you specify TO TIMESTAMP, you must provide a time stamp value. You can also specify a restore point name.

Note

- The CDB must be mounted in exclusive mode to issue the FLASHBACK DATABASE command and opened read-only to review changes. The CDB must be opened read/write with the RESETLOGS option when finished.
- When the CDB is opened in READ ONLY mode, the PDBs are still mounted. Open PDBs in READ ONLY mode too to review changes.

Duplicating Pluggable Databases

- A single pluggable database

```
RMAN> DUPLICATE DATABASE TO cdb1  
2>                      PLUGGABLE DATABASE pdb1;
```

- Several pluggable databases

```
RMAN> DUPLICATE DATABASE TO cdb1  
          PLUGGABLE DATABASE pdb1, pdb3;
```

- All pluggable databases except one

```
RMAN> DUPLICATE DATABASE TO cdb1  
          SKIP PLUGGABLE DATABASE pdb3;
```

- A PDB and tablespaces of other PDBs

```
RMAN> DUPLICATE DATABASE TO cdb1  
          PLUGGABLE DATABASE pdb1  
          TABLESPACE pdb2:users;
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

RMAN enables you to duplicate PDBs by using the DUPLICATE command.

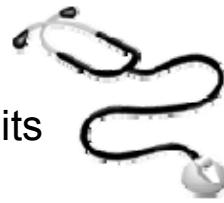
To duplicate PDBs, you must create the auxiliary instance as a CDB. To do so, start the instance with the declaration `enable_pluggable_database=TRUE` in the initialization parameter file. When you duplicate one or more PDBs, RMAN also duplicates the root (`CDB$ROOT`) and the seed database (`PDB$SEED`). The resulting duplicate database is a fully new, functional CDB that contains the root, the seed database, and the duplicated PDBs.

The first example shows how to duplicate a single pluggable database, the second one how to duplicate a set of pluggable databases, the third one how to duplicate all the databases in the multitenant container database, except a pluggable database, and the last one how to duplicate a set of tablespaces within a pluggable database.

You must be logged in to the root as a user who is granted the `SYSDBA` or `SYSBACKUP` role.

Find the whole procedure in *Oracle Database Backup and Recovery User's Guide 12c Release 1 Chapter Duplicating a Database*

Checking for Block Corruption



Invoking proactive health check of the database and its components using RMAN VALIDATE command:

- Scans the specified files and verifies their contents
 - CDB: All datafiles of the root and PDBs

```
RMAN> VALIDATE DATABASE;
```

- root: All datafiles of the root only

```
RMAN> VALIDATE DATABASE ROOT;
```

- PDB: All datafiles of the listed PDBs

```
RMAN> VALIDATE PLUGGABLE DATABASE pdb1, pdb2;
```

- Confirms that the datafiles exist and are in the correct location
- Checks for corrupt data blocks

The Oracle logo, which consists of the word "ORACLE" in a red rectangle with a white outline.

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For very important databases, you may want to execute proactive checks (possibly daily during low peak interval periods). You can schedule periodic health checks by using the RMAN VALIDATE command to validate any datafile of a whole CDB, of the root only, of a PDB, and of individual backup sets and data blocks.

- When connected to the root, validate:
 - The whole CDB using the VALIDATE DATABASE command
 - The root only using the VALIDATE DATABASE ROOT command
 - A PDB or several PDBs using VALIDATE PLUGGABLE DATABASE command.
- When connected to the PDB, validate the PDB with the VALIDATE DATABASE command.

Any problem detected during validation is displayed to you. If a failure is detected, it is logged into ADR as a finding.

Similarly, you can validate the restore of the CDB, the root only, or PDBs using the RESTORE DATABASE VALIDATE, RESTORE DATABASE ROOT VALIDATE, and RESTORE PLUGGABLE DATABASE *pdb1* VALIDATE commands, respectively.

For more detailed information about logical and physical corruption, refer to the Oracle University course *Oracle Database 12c: Backup and Recovery Workshop Ed 1*.

Special Situations

- Creating a controlfile backup script:

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

```
CREATE CONTROLFILE ...
  datafile sales_pdb file1
  datafile sales_pdb file2
  ...
  datafile hr_pdb file1 ...;
```

- Fast incremental backups using block change tracking:
 - At CDB level: Change tracking uses absolute data file #.
- PDB close **incompatible** with CDB hot backup:

```
SQL> ALTER DATABASE BEGIN BACKUP;
SQL> ALTER PLUGGABLE DATABASE hr_pdb CLOSE;
ALTER PLUGGABLE DATABASE pdb1_1 close
*
ERROR at line 1:
ORA-01149: cannot shutdown - file 10 has online backup set
ORA-01110: data file 10: '/D1/oradata/cdb1/pdb1_1/users01.dbf'
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- Besides the controlfile backup with RMAN autobackup configuration, you can also use the SQL statement that generates a script including the CREATE CONTROLFILE statement with all datafiles of the CDB, including the root and PDBs. The statement is still the same as for non-CDBs:

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

To perform this operation, connect to the root. Otherwise, you encounter an error message:

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
ALTER DATABASE BACKUP CONTROLFILE TO TRACE
*
ERROR at line 1:
ORA-65040: operation not allowed from within a pluggable database
```

- Fast incremental backups can use the block change tracking implemented at CDB level because the block change tracking uses absolute data file number:

```
SQL> ALTER DATABASE ENABLE BLOCK CHANGE TRACKING;
```

- While performing a hot CDB backup, you cannot close any PDB.

New Data Dictionary View and Column

New views

- RC_PDBS

New column PLUGGABLE_DBID in V\$ views and RC_xxx views

- V\$DATAFILE_COPY
- V\$BACKUP_DATAFILE
- V\$PROXY_DATAFILE
- RC_DATAFILE_COPY
- RC_BACKUP_DATAFILE
- RC_PROXY_DATAFILE



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

There is a new recovery catalog view RC_PDBS:

- PDB_KEY is the primary key for this PDB in the recovery catalog.
- DB_KEY is the primary key for the CDB in the recovery catalog. Use this column to join with almost any other catalog view.
- NAME is the name of the PDB.
- CON_ID is the unique identifier of the CDB.
- DBID is the unique identifier of the PDB.
- DROP_CHANGE# is the SCN at which the PDB was unplugged from the CDB.
- DROP_TIME is the time at which the PDB was unplugged from the CDB.

The principal V\$ and recovery catalog views contain a new column PLUGGABLE_DBID referencing the unique ID of a PDB, information stored in controlfile and datafile headers.

Quiz

Which statements are true about PDB and CDB backups?

- a. You can connect to a specific target PDB in RMAN.
- b. You can back up a whole PDB only if connected to the target PDB in RMAN.
- c. You may back up specific PDB tablespaces with a tablespace backup.
- d. You may back up the root container with a container-specific backup.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a, c, d

Quiz

You can recover a single PDB, provided the CDB is in the MOUNTED state.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

Summary

In this lesson, you should have learned how to:

- Perform CDB and PDB backups
- Use RMAN backups to plug unplugged PDBs
- Recover CDBs from essential files loss
- Recover PDBs from PDB datafiles loss
- Perform flashback database
- Duplicate PDBs
- Validate CDBs and PDBs



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 7 Overview: Managing CDB and PDBs Backup and Recovery

These practices cover the following topics:

- Cold backup of a CDB
- Whole CDB backup with RMAN
- PDB backup with RMAN
- Recovery from SYSTEM PDB datafile loss
- Recovery from nonessential PDB datafile loss
- SQL PDB Hot backup
- SQL controlfile backup
- Recovery from all controlfiles loss
- Recovery from redo log member loss
- Recovery from SYSTEM root datafile loss
- Recovery from a nonessential root datafile loss using DRA
- PDB PITR and PDB tablespaces PITR
- CDB flashback from common user DROP
- Plug an unplugged PDB using EM Cloud Control (using RMAN backup)



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

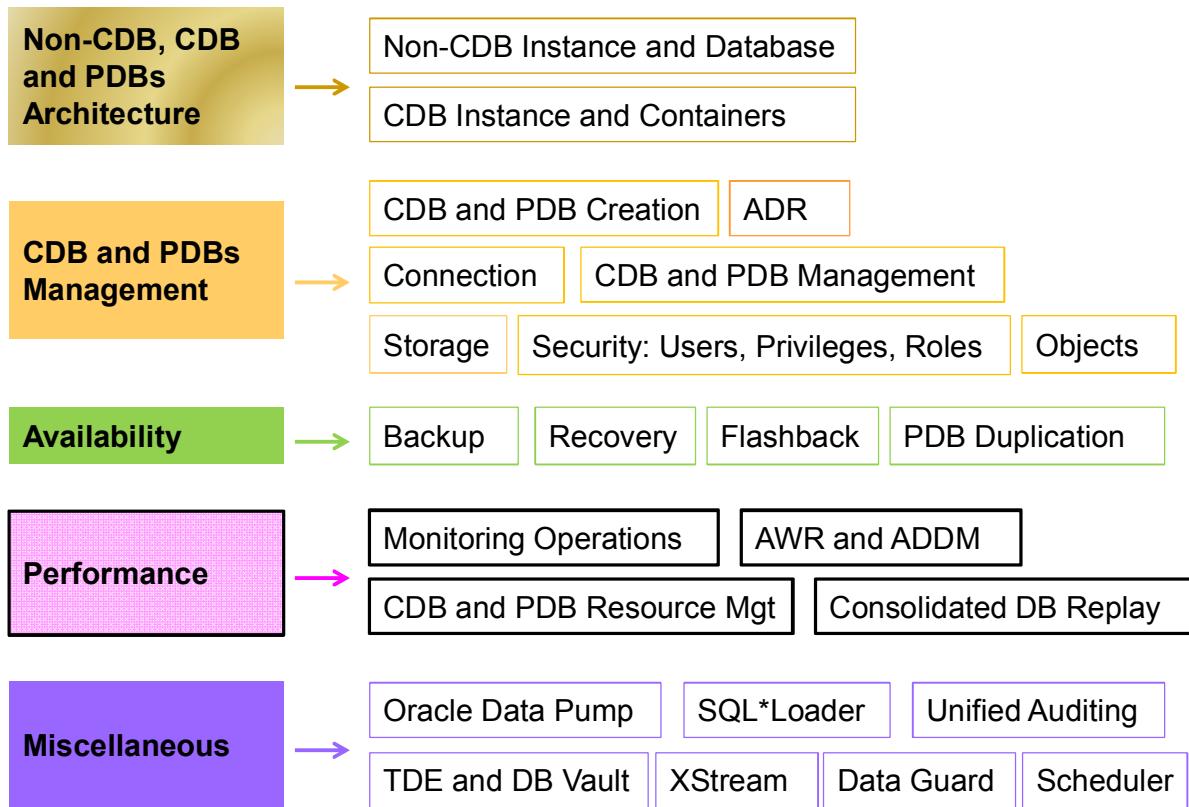
8

Performance

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Course Structure



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This lesson explains how to:

- Monitor SQL statements in a CDB and PDBs
- Manage the AWR and ADDM tasks in a CDB and PDBs
- Allocate resources at CDB and PDB levels using Resource Manager
- Benefit from the Consolidated Database Replay in a CDB

Objectives

After completing this lesson, you should be able to:

- Monitor operations in a CDB and PDBs
- Monitor performance in a CDB and PDBs
- Manage resource allocation between PDBs and within a PDB
- Describe use cases for Consolidated Database Replay



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

- *Oracle Database Administrator's Guide 12c Release 1 (12.1)* – Chapter *Administering a CDB with SQL*Plus*
- *Oracle Database Administrator's Guide 12c Release 1 (12.1)* – Chapter *Using Oracle resource Manager for PDBs with SQL*Plus*
- *Oracle PL/SQL Packages and Types Reference 12c Release 1 (12.1)*
- *Oracle Database Real Application Testing User's Guide 12c Release 1 (12.1)*

Refer to other sources of information:

- *Oracle Database 12c: SQL Tuning New Features* self study
- *Oracle Database 12c: New Features for Administrators* course – lesson *SQL Tuning Enhancements*.
- *Oracle Database 12c New Features Demo Series* demonstrations under Oracle Learning Library:
 - *Real-Time ADDM*

Tuning a Multitenant Container Database

Basic rules:

- The PDB appears to applications exactly the same as a non-CDB.
- Some Initialization parameters can be set at the PDB level.
- SQL statements are tuned on a per PDB basis.
 - Objects statistics are gathered in the PDB of the object.
- AWR tools run at the instance CDB level.
 - ASH
 - ADDM
- The cursors in the shared pool are identified by PDB.
- Instance-wide information is kept in the root container.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The rules relating to the behavior of the PDB are fairly simple, but have some complex implications.

The first rule is that a PDB servicing an application will behave in exactly the same way a non-CDB with the same data would with regard to the SQL and PL/SQL issued by the application. The implications of this statement with regard to tuning include that some initialization parameters can be set at the PDB level, and some of these parameters affect SQL performance.

SQL statements are tuned at the PDB level. The SQL Tuning Advisor runs in a specific PDB. SQL Profiles are applied at the PDB level. These require that cursors are identified by PDB in the shared pool. The objects statistics are collected with the DBMS_STATS package in the PDB where the object resides.

The Automatic Workload Repository (AWR) snapshots and Active Session History (ASH) data are kept in the root container. This information is used to provide instance-wide tuning advice using the Automatic Database Diagnostic Monitor (ADDM), AWR, and ASH tools.

Taking these rules into consideration, the tuning methodology for a CDB follows the same steps as for a non-CDB.

Tuning Methodology

Tuning steps:

- Identify the scope of the problem (OS, database, and so on).
- Tune the following from the top down:
 - Tune the design before tuning the application code.
 - Tune the code (SQL) before tuning the instance.
- Tune the area with the greatest potential benefit:
 - Identify the performance problem (AWR).
 - Analyze and look for skewed and tunable components.
 - Use appropriate tools to tune the components implicated.
- Stop tuning when the goal is met.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The tuning methodology for a multitenant container database (CDB) with many pluggable databases is basically the same as for a single instance non-CDB. Oracle Corporation has developed a tuning methodology based on years of experience. The methodology presented in this course is also presented in the *Oracle Database Performance Tuning Guide*. This methodology is applied independent of the tools that you use. The ADDM tool follows this methodology automatically. The basic steps are the following:

- Check the OS statistics and the general machine health before tuning the instance to be sure that the problem is in the database instance.
- Tune from the top down. Start with the design, then the application, and then the instance. As an example, try to eliminate the full table scans causing the I/O contention before tuning the tablespace layout on disk. The design should use appropriate data structures for the application and load characteristics. For example, reverse key indexes may work well in a RAC environment to reduce hot blocks due to a sequential primary key, but it may also lead to a large number of blocks being shipped across the interconnects if every instance is inserting into the same table. The applications should avoid processes that require serialization through a single resource. A simple example is a single check number generator used by multiple processes. Tuning at the instance level is often limited by design and application choices. With existing applications, this step is often not available, because the design and code are not modifiable.

- Tune the area with the greatest potential benefit. The tuning methodology presented in this course is simple. Identify the biggest bottleneck and tune it. Repeat. The Oracle tuning tools use DB Time to identify problem areas. All tools have some way to identify the SQL statements, resource contention, or services that are taking the most time. Oracle Database 12c provides a time model and metrics to automate the process of identifying bottlenecks.
- Stop tuning when you meet your goal. This step implies that you set tuning goals.

This is a general approach to tuning the database instance and may require multiple passes. Ideally someone with database tuning experience will be involved in the design and development from the beginning. This individual, for example, could suggest indexes to limit full table scans on frequently accessed tables.

From a practical perspective, tuning during the design and development phases of a project tends to be more top down. The tuning efforts during testing and production phases are often reactive and bottom up. In all phases, tuning depends on actual test cases because theoretical tuning does not know all the variables that can be present. After a problem area is suspected, or discovered, a test case is created and the area tuned as in all the examples given in this course. Tune the area that has the greatest potential benefit. Tune to reduce the longest waits and the largest service times.

As you notice the techniques are very much the same, no matter what life cycle phase. A test case or actual application is run, the available diagnostic tools are applied, a solution is proposed and tested.

General Tuning Session

Tuning sessions have the same procedure:

1. Define the problem and state the goal.
2. Collect current performance statistics.
3. Consider some common performance errors.
4. Build a trial solution.
5. Implement and measure the change.
6. Decide: “Did the solution meet the goal?”
 - No? Then go to step 3 and repeat.
 - Yes? Then create a new baseline.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When tuning, you focus on specific areas that offer the greatest return for your tuning effort. The steps are generic and apply to any performance monitoring tool. The recommended tuning methodology is as follows:

1. **Define the problem and state the goal:** This is the analysis step. The Oracle performance and diagnostic tools use a time model that can be used to quickly identify the problem areas. The information source could be users, database statistics, metrics, or database diagnostic reports. Be sure to collect accurate and factual data that corresponds to the problem. State the problem in terms that are measurable and directly related to the database operations. As an example, if the run time on the “XYZ” report is two times the baseline, the goal becomes: Make the run time on the “XYZ” report equal to or less than the baseline.
2. **Collect current statistics:** Examine the host system and the database statistics. Collect a full set of operating system and database statistics, and compare these with your baseline statistics. The baseline statistics are a set of statistics that are taken when the instance is running acceptably. Examine the differences to determine what has changed on the system. Did the “XYZ” report change? Did the data change? Is the session producing the report waiting on something?

3. **Consider common performance errors:** From your list of differences in the collected statistics, make a comparison with common performance errors. Determine whether one of these errors has occurred on your system.
4. **Build a trial solution:** Include a conceptual model in your solution. The purpose of this model is to assist you with the overall picture of the database. You are looking for answers to the following questions:
 - Why is the performance degraded?
 - How can you resolve the problem to meet your goal?
5. **Implement and measure the change:** After you have developed the trial solution, make the appropriate change. Make only one change at a time. If you make multiple changes at the same time, you will not know which change is effective. If the changes do not solve the problem, you would not know whether some changes helped and others hindered. Collect statistics to measure the change.
6. **“Did the solution meet the goal?”** Compare the current and the baseline sets. If you determine that more tuning is required, return to step 3 and repeat the process. If your solution meets the goal, make the current set of statistics the new baseline set. You met your goal! Stop tuning!

Sizing the CDB

Areas of concern:

- Memory (SGA and PGA)
 - Buffer Cache
 - Shared Pool
 - Program Global Area
- CPU over allocation
- SQL Tuning



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When considering non-CDBs' consolidation into a single CDB, several questions arise:

- How much memory will I need?
- How much CPU will I need for peak loads?
- Can I tune the applications independently?

An estimate of memory and CPU requirements can be obtained by gathering statistics as in an AWR report over a period of time for each non-CDB instance to estimate, and then take a sum of non-CDBs' buffer cache, shared pool, and PGA in the current instances. These estimates should also include period of peak usage, and the times of peak usage for each non-CDB to be consolidated.

One of the main reasons for consolidation is to reduce CPU over allocation. But a strategy for handling peak usage should be determined. Do multiple non-CDBs experience peak usage at the same time? Can the loads be time shifted? Can the applications tolerate being throttled by Resource Manager?

Application tuning is often the most effective tuning. If the application is well tuned on a non-CDB, it behaves well on a PDB. Then how to determine which PDB is handling the poorly behaving SQL? SQL statements even flagged at the CDB level have a tag that identifies the PDB of origin.

Testing the Estimates

Consolidated Database Replay tests:

- Consolidation of servers
- Scale-up
- Peak load capacity



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

After performing the initial sizing estimates for the multitenant container database, you can use Consolidated Database Replay to test the estimates by combining captured workloads from each of the individual non-CDB databases into a scheduled replay in a single CDB. This replay schedule can be modified to adjust peak loads, and test scenarios where additional workloads are expected to be added. Details on Consolidated Database Replay is covered in the last part of the lesson.

Allocating Resources in the CDB

Choose a strategy:

- Allow all PDBs to use all the resources
 - Gives maximum flexibility for each PDB
 - Allows any PDB to consume all available resources
- Assign a minimum allocation to each PDB
 - Ensures all PDBs get a specific share of the resources
 - Allows any PDB to consume any unused resources
- Assign a maximum allocation to each PDB
 - Prevents a PDB from taking more than the maximum value assigned
 - May result in unused capacity



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A valid concern in a CDB is that one or more PDBs will consume so much resource that other PDBs will suffer performance degradation. The consolidated resource plan allows you to choose a strategy for allocating resources among the various PDBs in a CDB. The resource plan sets limits on the CPU, IO, and parallel processes that may be used by a PDB. The three main strategies are:

- **No resource plan:** All PDBs have equal access to all the resources. There is no arbitration. If one PDB starts a large parallel query it could consume all the resources in the CDB instance. This is the default configuration.
- **Minimum allocation:** Set a resource plan to give each PDB a minimum resource share, for example, one share each. In this case, each PDB is guaranteed a share of the resource. That share is equal to $1/(\text{the number of shares allocated})$. When all the CDB resources are being used, the resource manager will force PDB processes using more than their minimum share to wait, allowing all PDBs to get a minimum share.
- **Limited allocation:** Using a resource plan, each PDB is assigned shares to set its minimum allocation, and a percent limit to set the maximum amount of a resource such as CPU used by the PDB. When a PDB attempts to exceed the maximum value, the resource manager forces the processes to wait.

Details on consolidated resource plan are covered in the next part of the lesson.

Initialization Parameters in a CDB

One SPFILE for a CDB instance

- Each PDB can change some initialization parameters.
- PDB parameters are stored in the root container.
- PDB parameter changes persist across PDB restarts.
- PDB parameter changes travel with unplug/plug and clone operations.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When placing multiple non-CDBs (becoming PDBs) into a single CDB, consider any special initialization parameters the non-CDB may have required. By default, all the PDBs will inherit the initialization parameters of the CDB, because there is only one SPFILE per CDB instance. There are some, approximately 150 parameters that can be modified on a per PDB basis. These parameters are those whose value in column `ISPDB_MODIFIABLE` in `V$PARAMETER` is `TRUE`. Because these SPFILE changes for specific PDBs are stored in the root container, these modifications will persist across a PDB reopen, that is an `ALTER PLUGGABLE DATABASE ... CLOSE`, and `ALTER PLUGGABLE DATABASE ... OPEN`. The PDB-specific initialization parameters will travel with the PDB when it is unplugged, and plugged, or when it is cloned.

Enterprise Manager Cloud Control 12c: Setting Initialization Parameters

The parameter values listed here are currently used by the running instance.

Name	Basic	Modified	Dynamic	Category
ddl	All	All	All	All

Filter on a name or partial name

Apply changes in current running instance(s) mode to SPFile. For static parameters, you must restart the database.

Name	Help	Revisions	Value	Comments	Type	Basic	Modified	Dynamic	Category
ddl_lock_timeout			50		Integer		✓	✓	Miscellaneous
enable_ddl_logging			FALSE		Boolean		✓	✓	Miscellaneous

Execute On Multiple Databases | Show SQL | Revert | Apply

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Enterprise Manager Cloud Control 12c allows you to set the initialization parameter at the CDB (instance) level and at the PDB level.

The different values set in PDBs for a parameter can be displayed in the `PDB_SPFILE$` table:

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

DB_UNIQ_NAME	PDB_UID	NAME	VALUE\$
<hr/>			
cdb2	3072231663	ddl_lock_timeout	10
cdb2	4030283986	ddl_lock_timeout	20
cdb2	3485283967	ddl_lock_timeout	50

Tuning CDB Memory

Memory areas to be tuned:

- Buffer Cache
- Shared Pool
- Program Global Area (PGA)

Memory advisors provide guidance at CDB level.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The buffer cache in the CDB behaves much like the buffer cache in a non-CDB. The database buffer blocks use the same least recently used algorithms that are used in the non-CDB. However, some additional information is kept in the data block header.

The PDB container id (`con_id`) is kept with each block. So you can find the number and status of blocks being used by each PDB at any given time with:

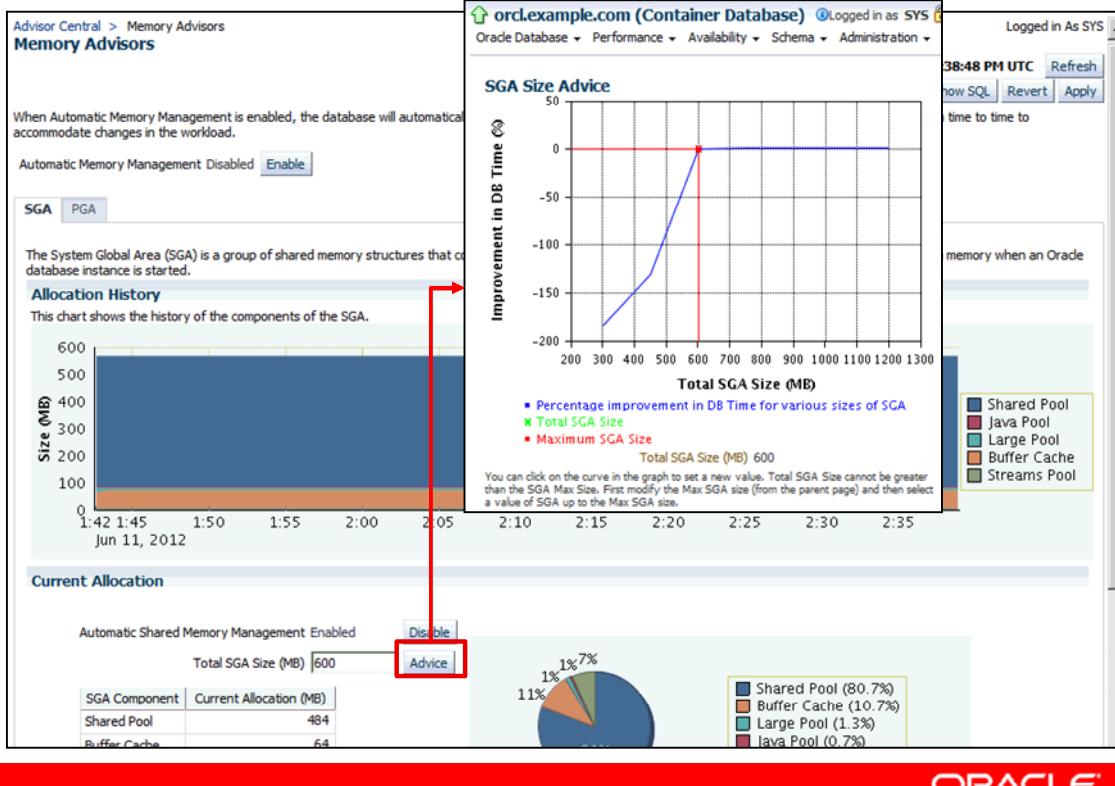
```
SQL> SELECT status, con_id, count(*)  
      2  FROM v_$bh GROUP BY status, con_id ORDER BY 2;
```

The number of blocks used by any PDB will vary depending on the activity.

Several memory areas are created in the shared pool for each open PDB. As in non-CDB, each session has a PGA, for handling private sorts, joins, parallel queries and other temporary data. By default, there is one temporary tablespace for the CDB, but additional temporary tablespaces may be added in a PDB. These tablespaces may be set to be the default temporary tablespace for the PDB, or individual users can be assigned to specific temporary tablespaces.

The memory advisors for the buffer cache, shared pool, streams pool, and PGA continue to provide guidance for adjusting the instance memory parameters.

Enterprise Manager Cloud Control 12c: Memory Advisors



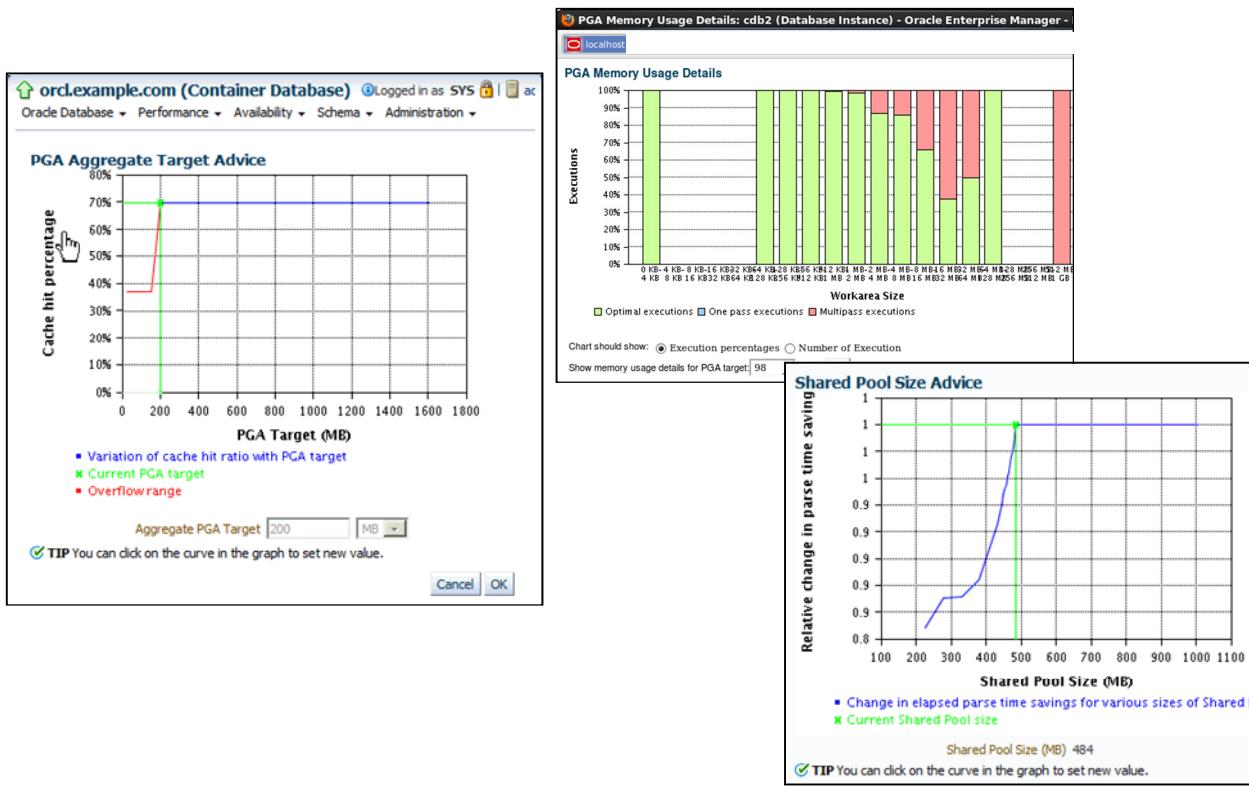
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Enterprise Manager Cloud Control provides memory advisors appropriate for the database settings. As shown in the slide, the memory advisor displays the maximum SGA size set, when the automatic shared memory management is enabled. Using the Advice button, you can get an advice on sizing the SGA according to the statistics collected in the AWR.

The Memory Advisors are available in Enterprise Manager Cloud Control, at the CDB level.

Memory Advisors



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

EM CC has other memory advisors (already seen in previous versions of Enterprise Manager). Screenshots of PGA advisor and Shared Pool Advisor are shown.

The buffer cache and shared pool are tuned instance-wide. This is a very good application for automatic shared memory management, so that if the application load changes, when the load on some PDBs increases and on others decreases, the allocations to buffer cache and shared pool can be automatically adjusted.

The target PGA used by the entire instance can also be set (exactly as it is in previous versions of the database).

Limiting PGA Usage

`PGA_AGGREGATE_TARGET` sets a soft limit.

- It applies to tuneable memory only.
- The target value can be exceeded by sessions using untuneable memory.
- A large number of sessions could still exhaust server memory.

`PGA_AGGREGATE_LIMIT` sets a hard limit.

- For allocated memory
- Aborts calls/queries to reduce memory use
- Terminates sessions if required to reduce memory use



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When there are many sessions, each one having a PGA memory area allocated, it is possible that the total PGA will exceed the amount of memory available on the server.

`PGA_AGGREGATE_TARGET` allows you to set a soft limit on the amount of PGA used by all the sessions in the instance. This limit works on the tuneable portion of session PGA memory. In OLTP applications, often the majority of PGA is untuneable. With a large number of sessions each having a portion of untuneable PGA, they could consume all the server memory.

`PGA_AGGREGATE_LIMIT` sets a hard limit on the amount of PGA memory used. If the PGA limit parameter is explicitly set, the sessions using the most memory will have their calls aborted assuming that this will free about 75% of their PGA memory. As many sessions as needed to get under the limit will be affected. If more than 10% of the sessions will be affected, sessions will be terminated with the assumption that this will free 100% of their allocated memory.

If the PGA limit parameter was not explicitly set, the single session using the most memory will either abort its query/call if 75% of the allocated memory would be sufficient to go below the limit or is killed.

Action will be taken every three seconds as long as the limit is being exceeded.

Whenever possible, processes will kill their own calls or sessions themselves to avoid additional congestion in PMON process cleanup.

Tuning SQL

SQL Tuning is at the PDB level.

SQL statements are affected by PDB:

- Schema
- Data volume
- Optimizer parameters

SQL tuning recommendations are implemented by PDB.

Other SQL Tuning enhancements to:

- Adaptive SQL Plan Management
- Automatic SQL plan baseline evolution
- The SQL management base
- SQL plan directives
- Statistics-gathering performance improvements



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

As in the non-CDB application tuning, tuning the SQL and PL/SQL commonly provides the largest benefit. Each SQL cursor, and the corresponding history stored in AWR has a tag identifying the source PDB. The SQL Tuning Advisor and SQL Access Advisor execute in the context on a PDB. This is reasonable because each SQL statement executes in one and only one PDB. Even though SQL statements having identical text may be executed in multiple PDBs, the statistics, schema, indexes, data volume, and optimizer parameters could be different in each PDB.

SQL Profiles and recommendations generated by the advisors are implemented in a single PDB.

Oracle Database 12c has added several enhancements that make SQL tuning more automatic such as enhancements to:

- Adaptive SQL plan management
- Automatic SQL plan baseline evolution
- SQL management base
- SQL plan directives
- Statistics-gathering performance improvement

For more details, see the *Oracle Database 12c: SQL Tuning New Features* self study and *Oracle Database 12c: New Features for Administrators* course Lesson *SQL Tuning Enhancements*.

AWR SQL Reports

SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- %Total - CPU Time as a percentage of Total DB CPU
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User IO Time as a percentage of Elapsed Time
- Captured SQL account for 57.0% of Total CPU Time (s): 112
- Captured PL/SQL account for 82.4% of Total CPU Time (s): 112

CPU Time (s)	Executions	CPU per Exec (s)	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	PDB Name	SQL Text
28.33	219	0.13	25.27	54.25	52.22	3.52	4xy1ps2v9m96z	SQL*Plus	PERFTEST	DECLARE max_orders NUMBER(6) ...
16.04	2,163	0.01	14.31	149.56	10.72	68.71	0w2apuc6u2zsp	Swingbench User Thread	SOEPDB	BEGIN :1 := orderentry.neworde...
7.20	4	1.80	6.42	27.64	26.04	61.73	0v30sznmzmnby			insert into wrh\$\$_mvparameter (...
6.81	33,787	0.00	6.07	13.23	51.46	5.53	b83t16nqg7vt9	SQL*Plus	PERFTEST	INSERT INTO ORDER_ITEMS(ORDER...
6.46	2,538	0.00	5.76	43.32	14.91	52.21	147a57cxq3w5y	Swingbench User Thread	SOEPDB	BEGIN :1 := orderentry.browsep...
5.50	6,660	0.00	4.91	9.98	55.11	0.61	658qfxar410kx	SQL*Plus	PERFTEST	SELECT ORDER_ID FROM (SELECT O...
5.29	221	0.02	4.72	10.07	52.49	4.83	f20zqr4dp2wht	SQL*Plus	PERFTEST	DECLARE new_order_id NUMBER(12...
4.65	112	0.04	4.15	15.20	30.56	10.42	2xvct3vhuxs6f	SQL*Plus	PERFTEST	DECLARE max_records NUMBER(6) ...
3.86	35,746	0.00	3.44	6.78	56.98	0.60	24p3kfnpxdrx5	SQL*Plus	PERFTEST	SELECT PRODUCT_ID, NVL(LIST_PR...
3.43	8	0.43	3.06	6.26	54.73	4.79	g57kbmvd1gqfk	Realtime Connection		
3.14	6,515	0.00	2.80	40.37	7.78	71.18	c13sma6rkr27c	New Order	SOEPDB	SELECT PRODUCTS.PRODUCT_ID, PR...
2.19	1	2.19	1.96	8.18	26.81	55.43	5ik5fx9w818p1	Admin Connection		SELECT OBJECT_ID, SUBPROGRAM_I...

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The AWR report includes the Top SQL ordered by various measures. In a CDB, each SQL is tagged with the PDB name where the SQL originated. Two similar statements can have been executed from different PDBs.

ADDM Tasks

The screenshot shows the ADDM interface for a Container Database (cdb2). A red box highlights the top navigation bar where 'cdb2 (Container Database)' is selected. An oval highlights the 'Processing: Run ADDM Now' button. A callout box points to the message 'A snapshot is being taken which will automatically result in an ADDM run'. Below this, another red box highlights the 'SQL Tuning' section under 'Recommendations'. A callout box points to the 'Action' and 'Rationale' for a specific recommendation: 'Investigate the ALTER PLUGGABLE DATABASE statement with SQL ID "411k08d873avg" for possible performance improvements.' The Oracle logo is visible at the bottom right.

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The ADDM tasks are also executed at the CDB level and provide recommendations on statements executed in any container, root, and PDBs.

SQL Advisors

SQL Advisors provide guidance at PDB level.

The screenshot shows the Oracle Enterprise Manager Cloud Control interface. At the top, there's a navigation bar with 'CDB name' set to 'orcl.example.com / **PERF2**' and 'PDB name' set to 'PERF2'. Below the navigation bar, the main content area is titled 'SQL Advisors'. It contains three sections: 'SQL Access Advisor', 'SQL Tuning Advisor', and 'SQL Repair Advisor'. Each section provides a brief description and links to further details or tools. The 'SQL Tuning Advisor' section includes a link to 'Support Workbench' and another to 'SQL Worksheet'. The 'SQL Repair Advisor' section includes a link to 'SQL Worksheet'. The bottom right corner of the interface features the 'ORACLE' logo.

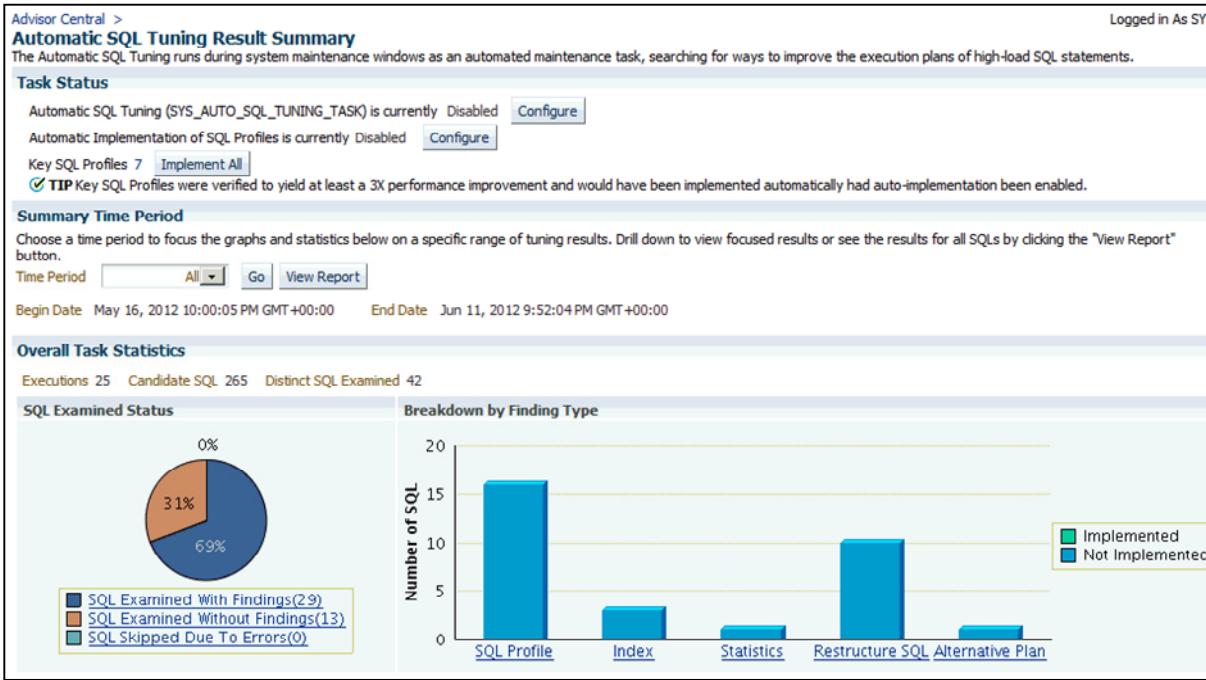
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The SQL Advisors are available in Enterprise Manager Cloud Control, at the PDB level. Notice at the upper left of the screenshot that a picker is available to choose which PDB is being examined.

To run SQL Tuning Advisor or SQL Performance Analyzer on 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

SQL Tuning Advisor



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The SQL Tuning Advisor may be run manually for selected SQL statements or a set of statements, but the Automatic SQL Tuning job is run daily against the top SQL all PDBs statements taken together. The result of the automatic job is shown in the slide. The results may be examined in detail, or implemented individually or all at once.

Quiz

Regarding initialization parameters, which of the following statements are true?

- a. By default, initialization parameters are set at CDB instance level.
- b. Some parameters can be set in the PDB.
- c. The parameters set in a PDB revert to instance defaults when PDB is restarted.
- d. All initialization parameters for optimizer are set at the CDB instance level.



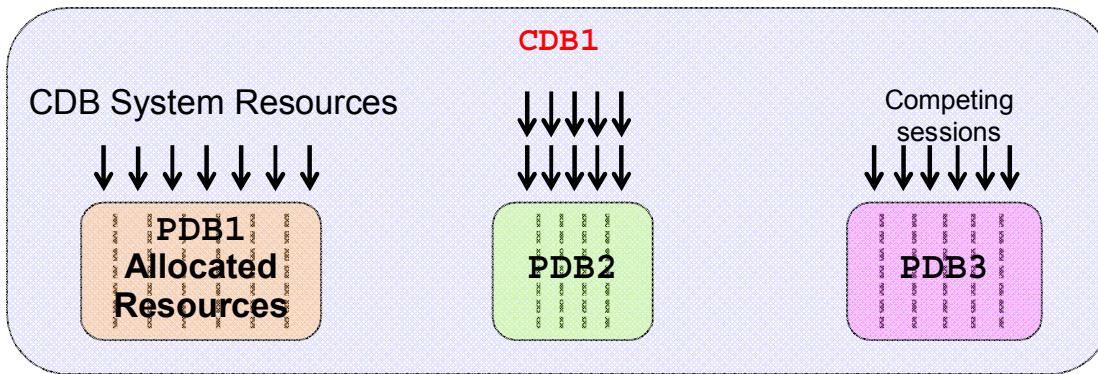
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a, b

Resource Manager and Pluggable Databases

In a CDB, Resource Manager manages resources:

- Between PDBs
- Within each PDB



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

In a CDB, Resource Manager can manage resources on two basic levels:

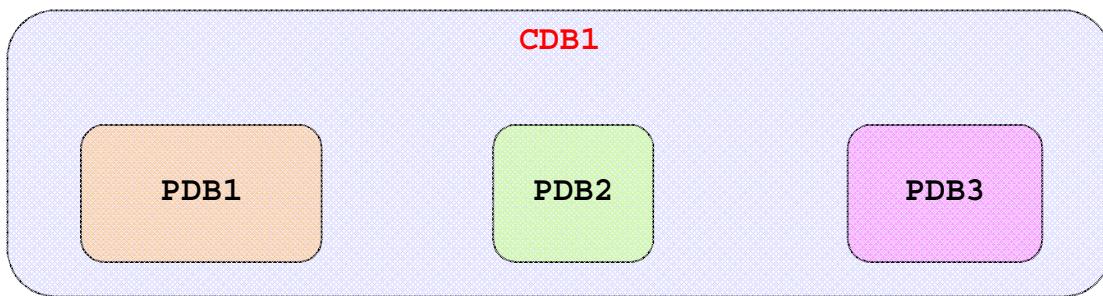
- **CDB level:** Resource Manager can manage the workloads for multiple PDBs that are contending for system and CDB resources. You can specify how resources are allocated to PDBs, and you can limit the resource utilization of specific PDBs.
- **PDB level:** Resource Manager can manage the workloads within each PDB.

Resource Manager allocates the resources in two steps:

1. It allocates a portion of the system's resources to each PDB.
2. In a specific PDB, it allocates a portion of system resources obtained in Step 1 to each session connected to the PDB.

Managing Resources Between PDBs

- PDBs compete for resources: CPU, Exadata I/O, parallel servers
 - System shares are used to allocate resources for each PDB.
 - Limits are used to cap resource utilization of each PDB.
- When a new PDB is plugged in, the CDB DBA can specify a default or explicit allocation.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

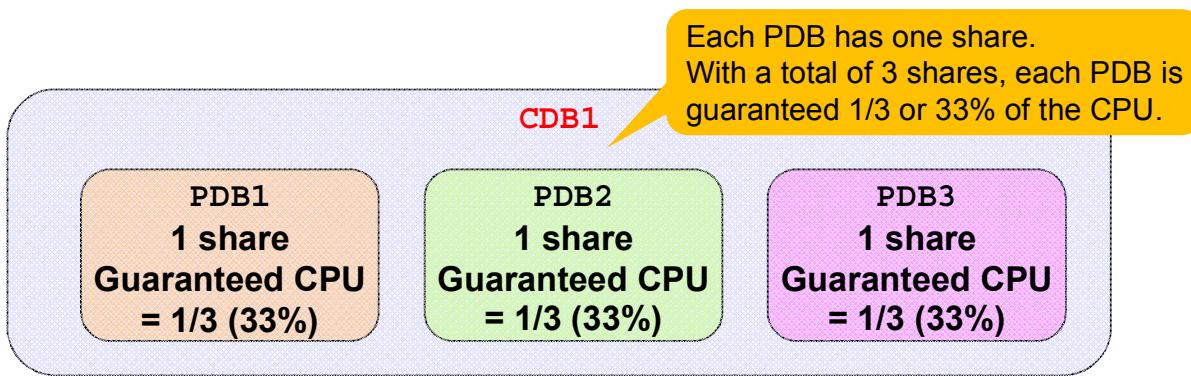
In a CDB with multiple PDBs, some PDBs typically are more important than others. The Resource Manager enables you to prioritize the resource (CPU and I/O, as well as allocation of parallel execution slaves in the context of parallel statement queuing) usage of specific PDBs. This is done by granting different PDBs different shares of the system resources so that more resources are allocated to the more important PDBs.

In addition, limits can be used to restrain the system resource usage of specific PDBs.

When a PDB is plugged in to a CDB and no directive is defined for it, the PDB uses the default directive for PDBs. As the CDB DBA, you can control this default and you can also create a specific directive for each new PDB.

Note: No consumer groups nor shares can be defined for the root container.

CDB Resource Plan Basics: Share



- Specify resources using “shares”.
 - Recomputation is not required when PDBs are added or removed.
 - PDB1 is guaranteed 33% of the CPU.
 - PDB1 is limited to 100% of the CPU.
 - PDB1 is actually using 20% of the CPU.

ORACLE

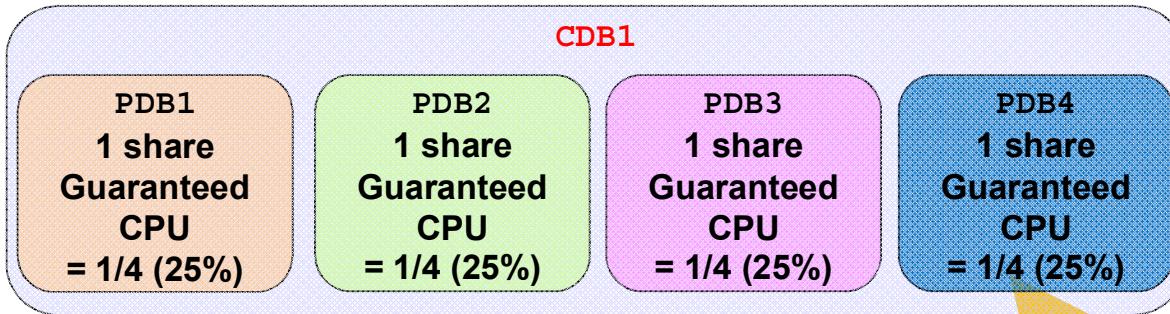
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To allocate resources among PDBs, you assign a share value to each PDB. A higher share value results in more resources for a PDB.

In this example, each existing PDB is assigned one share. With a total of three shares, each PDB is guaranteed to get at least 33% of the system resources.

Depending on the workload, each PDB can get up to 100% of the system resources but may actually use only 20% of these resources.

CDB Resource Plan Basics: Share



- Default allocation: one share

New PDB gets the default allocation:
1 share.
With a total of 4 shares, each PDB is
guaranteed 1/4 or 25% of the CPU.



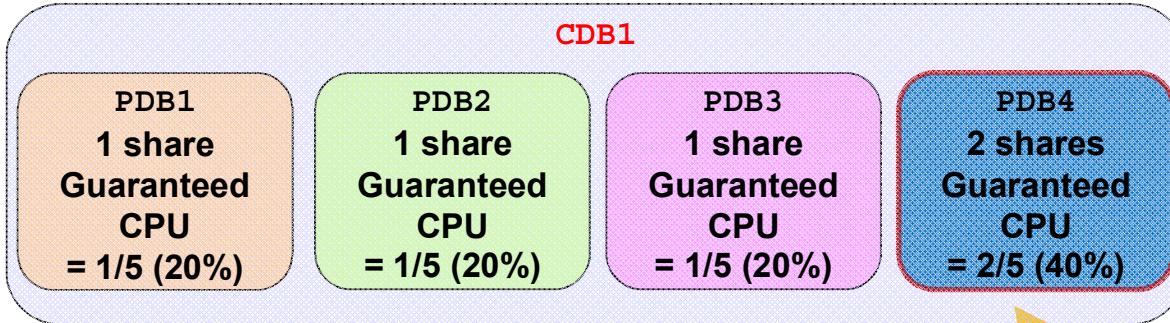
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If a new PDB is added, it automatically gets the default share value of one.

In this case, with a total of four shares, each PDB is guaranteed to get 25% of the system resources.

Note: Each CDB resource plan gets a default directive added to it. You can change this default directive if its default value is not suitable for your plan.

CDB Resource Plan Basics: Share



If this PDB is more important, you can explicitly allocate it more shares. It then gets more resources than the other PDBs.

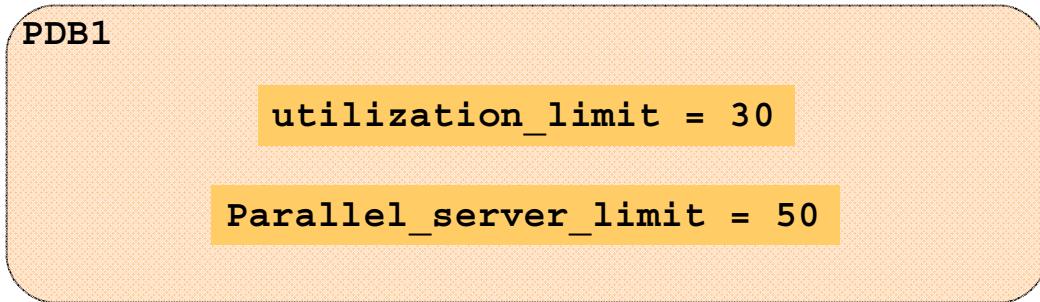
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If you consider PDB4 to be of higher priority than the others, you can explicitly assign it more shares. In this example, PDB4 gets two shares while the other PDBs get one share each. With a total of five shares, PDB1, PDB2, and PDB3 are guaranteed to get 1/5 (20%) of the system resources because they were assigned one share each. Because PDB4 was assigned two shares, it is guaranteed to get 2/5 (40%) of the system resources.

CDB Resource Plan Basics: Limits

- Two limits can be defined for each PDB:
 - Utilization limit for CPU, Exadata I/Os, and parallel servers
 - Parallel server limit to override the utilization limit
- Default values: 100%
- You can change default values.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A limit restrains the system resource usage of a specific PDB. You can specify limits for CPU, Exadata I/Os, and parallel execution servers.

To limit the CPU, Exadata I/Os, and parallel servers (PARALLEL_SERVER_TARGET initialization parameter) usage of each PDB, you can create a plan directive using the UTILIZATION_LIMIT parameter expressed as a percentage of the system resources the PDB can use. Resource Manager throttles the PDB sessions so that the CPU, Exadata I/Os, and parallel servers utilization for the PDB does not exceed the utilization limit. In the slide example, PDB1 gets a maximum of 30% of the system CPU, Exadata I/Os bandwidth, and available parallel servers for the CDB instance.

For parallel execution servers, you can override the value defined by the UTILIZATION_LIMIT by creating a plan directive that uses the PARALLEL_SERVER_LIMIT parameter. The PARALLEL_SERVER_LIMIT value corresponds to a percentage of PARALLEL_SERVERS_TARGET. For example, if the PARALLEL_SERVERS_TARGET initialization parameter is set to 200 and the PARALLEL_SERVER_LIMIT is set to 50% for a PDB, then the maximum number of parallel servers the PDB can use is 100 (200 X .50).

When you do not explicitly define limits for a PDB, the PDB uses the default limits for PDBs that are set to 100%. These are the values corresponding to the default directive that is automatically added to your plan.

Note: You can also change the default directive attribute values for PDBs by using the UPDATE_CDB_DEFAULT_DIRECTIVE procedure in the DBMS_RESOURCE_MANAGER package.

CDB Resource Plan: Full Example

PDB/Directive Name	Shares	Utilization Limit	Parallel Server Limit
(Default Allocation)	(1)	(100%)	(100%)
(Autotask Allocation)	(-1)	(90%)	(100%)
PDB1	1	30%	50%
PDB2	1	30%	80%
PDB3	1	30%	30%
PDB4	2	100%	100%

PDB1 is:

- Guaranteed 1/5 (20%) of CPU and Exadata disk bandwidth
- Limited to 30% of CPU and Exadata disk bandwidth
- Limited to 50% of the parallel servers

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This example shows you a possible CDB resource plan created in the root container.

The default allocation directive applies to PDBs for which specific directives have not been defined.

Default allocation is one share and both corresponding limits are set to their default of 100%. These are the default values for the default allocation directive.

The autotask allocation directive applies to automatic maintenance tasks that are run in the root or in PDBs.

The autotask allocation is -1 share, which means that the automated maintenance tasks get an allocation of 20% of the system resources. You should always change this default value. By default, the autotask allocation gets a utilization limit of 90% and 100% for its parallel server limit.

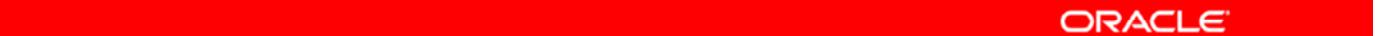
PDB1 is allocated 1 share which represents 1/5 or 20% of the CPU, Exadata disk bandwidth, and queued parallel queries will be selected 1/5 of the time compared to the other PDBs. In addition, a utilization limit of 30% of the resources is set as well as a 50% limit of the available parallel servers.

Root

Creating a CDB Resource Plan

1. Create a pending area.
2. Create a CDB resource plan.
3. Create directives for the PDBs.
4. (*Optional*) Update the default PDB directives.
5. (*Optional*) Update the default autotask directives.
6. Validate the pending area.
7. Submit the pending area.

DBMS_RESOURCE_MANAGER

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

CDB resource plan management is done directly from the root container.

The slide shows you the different steps you can use to create a CDB resource plan.

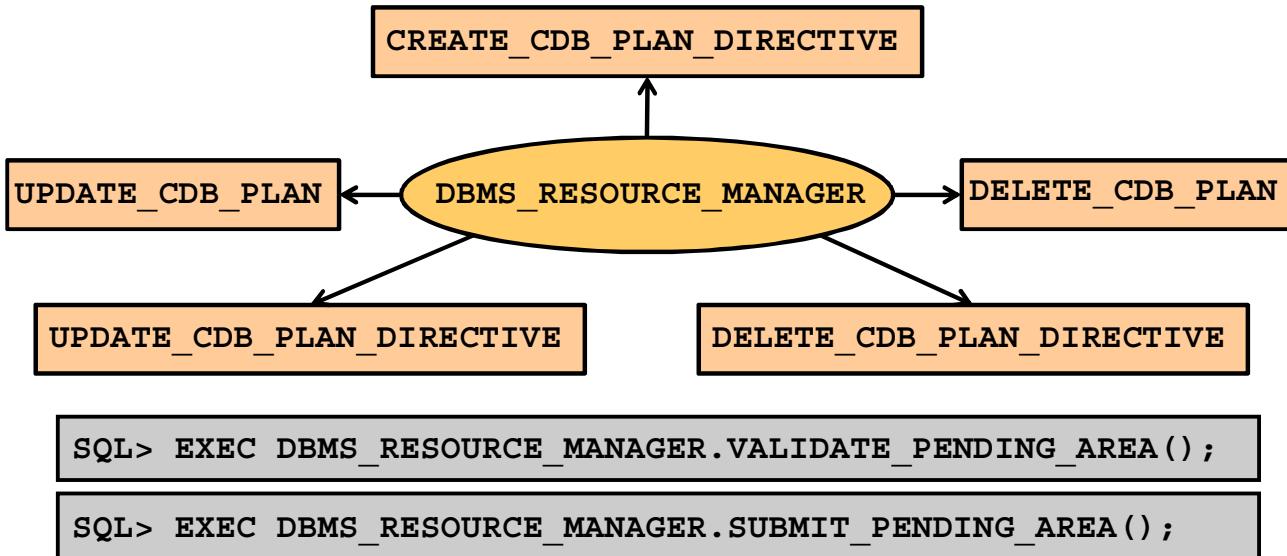
You use the DBMS_RESOURCE_MANAGER package to create a CDB resource plan and define the directives for the plan.

Note: If you try to define a CDB resource plan in a PDB, you get an error.

Maintaining a CDB Resource Plan

```
SQL> CONNECT sys AS SYSDBA
```

```
SQL> EXEC DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```



```
SQL> EXEC DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

```
SQL> EXEC DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

1. You can update a CDB resource plan to change its comment using the UPDATE_CDB_PLAN procedure.
2. When you create a PDB in a CDB, you can create a CDB resource plan directive for the PDB using the CREATE_CDB_PLAN_DIRECTIVE procedure. The directive specifies how resources are allocated to the new PDB.
3. You can delete the CDB resource plan directive for a PDB using the DELETE_CDB_PLAN_DIRECTIVE procedure. You might delete the directive for a PDB if you unplug or drop the PDB. However, you can retain the directive, and if the PDB is plugged into the CDB in the future, the existing directive applies to the PDB.
4. You can update the CDB resource plan directive for a PDB using the UPDATE_CDB_PLAN_DIRECTIVE procedure. The directive specifies how resources are allocated to the PDB.
5. You can delete a CDB resource plan using the DELETE_CDB_PLAN procedure. You might delete a CDB resource plan if the plan is no longer needed. You can enable a different CDB resource plan, or you can disable Resource Manager for the CDB. If you delete an active CDB resource plan, then some directives in PDB resource plans become disabled.

Creating CDB Resource Plan: SQL Example

```
SQL> CONNECT sys AS SYSDBA
```

```
SQL> EXEC DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

```
SQL> BEGIN
  2   DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN(
  3     plan      => 'daytime_plan',
  4     comment  => 'CDB resource plan for mycdb');
  5 END;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

From the root container of your CDB, connect as the `SYS` user.

Then, create a pending area using the `CREATE_PENDING_AREA` procedure.

You can now create a CDB resource plan named `daytime_plan` using the `CREATE_CDB_PLAN` procedure.

Creating CDB Resource Plan: SQL Example

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
    plan                  => 'daytime_plan',
    pluggable_database    => 'salespdb',
    shares                => 3,
    utilization_limit     => NULL,
    parallel_server_limit => NULL);
END;
```

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
    plan                  => 'daytime_plan',
    pluggable_database    => 'hrpdb',
    shares                => 1,
    utilization_limit     => 70,
    parallel_server_limit => 70);
END;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Create the CDB resource plan directives for the PDBs using the CREATE_CDB_PLAN_DIRECTIVE procedure. Each directive specifies how resources are allocated to a specific PDB.

Here, the salespdb PDB gets three shares and default limits, hrpdb gets 1 share and 70% for both limits.

Creating CDB Resource Plan: SQL Example

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_CDB_DEFAULT_DIRECTIVE(
    plan                      => 'daytime_plan',
    new_shares                => 1,
    new_utilization_limit     => 50,
    new_parallel_server_limit => 50);
END;
```

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_CDB_AUTOTASK_DIRECTIVE(
    plan                      => 'daytime_plan',
    new_shares                => 1,
    new_utilization_limit     => 75,
    new_parallel_server_limit => 75);
END;
```

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

All other PDBs in this CDB use the default PDB directive that you can optionally update using the UPDATE_CDB_DEFAULT_DIRECTIVE. In the example in the slide, you only change the limits' defaults to 50%.

Similarly, if the current autotask CDB resource plan directive does not meet your requirements, then update the directive using the UPDATE_CDB_AUTOTASK_DIRECTIVE procedure. Here, all three parameters are changed to 1 and 75% respectively.

When you are done, simply validate and submit your pending area.

Viewing CDB Resource Plan Directives

```
SQL> SELECT plan, pluggable_database, shares,
  2 utilization_limit, parallel_server_limit
  3 FROM dba_cdb_rsrc_plan_directives
  4 ORDER BY plan;
```

Plan	Pluggable Database	Shares	Utilization Limit	Parallel Server Limit
DAYTIME_PLAN	ORA\$DEFAULT_PDB_DIRECTIVE	1	50	50
DAYTIME_PLAN	ORA\$AUTOTASK	1	75	75
DAYTIME_PLAN	SALESPDB	3		
DAYTIME_PLAN	HRPDB	1	70	70
DEFAULT_CDB_PLAN	ORA\$DEFAULT_PDB_DIRECTIVE	1	100	100
DEFAULT_CDB_PLAN	ORA\$AUTOTASK		90	100
DEFAULT_MAINTENANCE_PLAN	ORA\$DEFAULT_PDB_DIRECTIVE	1	100	100
DEFAULT_MAINTENANCE_PLAN	ORA\$AUTOTASK		90	100
ORA\$INTERNAL_CDB_PLAN	ORA\$DEFAULT_PDB_DIRECTIVE			
ORA\$INTERNAL_CDB_PLAN	ORA\$AUTOTASK			



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This example uses the DBA_CDB_RSRC_PLAN_DIRECTIVES view to display all of the directives defined in all of the CDB resource plans in the root container.

The ORA\$INTERNAL_CDB_PLAN is the default CDB resource plan. It is a very simple plan where no resources are managed.

Note: From the root container, you can use the DBA_CDB_RSRC_PLANS view to display all of the CDB resource plans defined in the root container.

Enabling a CDB Resource Plan

```
SQL> ALTER SYSTEM SET resource_manager_plan='daytime_plan';
```

```
BEGIN  
  DBMS_SCHEDULER.CREATE_WINDOW(  
    window_name      => 'daytime',  
    resource_plan   => 'daytime_plan',  
    start_date      => '12-Feb-01 8:00:00AM',  
    repeat_interval=> 'freq=daily',  
    duration        => interval '10' hour);  
END;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You enable the Resource Manager for a CDB by setting the RESOURCE_MANAGER_PLAN initialization parameter in the root. This parameter specifies a CDB resource plan. If no plan is specified with this parameter, then the Resource Manager is not enabled.

The example in the slide uses the ALTER SYSTEM command to set the CDB resource plan to the daytime_plan.

Optionally, you can associate a CDB plan to scheduler window so that the plan is automatically set when the window opens.

Note: You can disable CDB resource management by using the following SQL statement from the root container: ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = ''

If you disable a CDB resource plan, then some directives in PDB resource plans become disabled.

Managing Resources Within a PDB

- In a non-CDB database, workloads within a database are managed with resource plans.
- In a PDB, workloads are also managed with resource plans, also called PDB resource plans.
- The functionality is similar except for the following differences:

Non-CDB Database	PDB Database
Multilevel resource plans	Single-level resource plans only
Up to 32 consumer groups	Up to 8 consumer groups
Subplans	No subplans



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A CDB resource plan determines the amount of resources allocated to each PDB. A PDB resource plan determines how the resources allocated to a specific PDB are allocated to consumer groups within that PDB. A PDB resource plan is similar to a resource plan for a non-CDB. Specifically, a PDB resource plan allocates resource among the consumer groups within a PDB.

In a CDB, the following restrictions apply to PDB resource plans:

- PDB resource plans cannot have a multiple-level scheduling policy.
- PDB resource plans can have a maximum of eight consumer groups.
- PDB resource plans cannot have subplans.

Note: If you try to create a PDB plan in the root, you get an error.

Managing PDB Resource Plans

- Connect to the PDB to manage the PDB plan.
- Use exactly the same procedures as for managing a resource plan in a non-CDB environment.
- For CREATE_PLAN_DIRECTIVE procedure:
 - New SHARE argument
 - Replace MAX_UTILIZATION_LIMIT and PARALLEL_TARGET_PERCENTAGE arguments with UTILIZATION_LIMIT and PARALLEL_SERVER_LIMIT
- You can view CDB and PDB resource plans using V\$RSRC_PLAN.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Note

V\$RSRC_PLAN exists in previous releases of the Oracle Database. The resource plan with con_id=1 is the active CDB resource plan.

Putting It Together

- How do CDB and PDB resource plans work together?
 1. Resources allocated to a PDB, based on CDB resource plan
 2. Resource allocated to a consumer group based on the PDB resource plan

CDB Plan

PDB	Shares	Utilization Limit
PDB1	1	50%
PDB2	1	50%
PDB3	2	100%

PDB3 Plan

Consumer Group	Shares	Utilization Limit
OLTP	3	100%
REPORTS	1	50%
OTHER	1	50%

- What does this mean for PDB3 Reports?
 - Guaranteed $50\% (2/4) \times 20\% (1/5) = 10\%$ of the resources
 - Limited to $100\% \times 50\% = 50\%$ of the resources

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Resources are allocated to each PDB based on the CDB resource plan in action. Each PDB receives a certain percentage of the system resources, and Resource Manager distributes that lot to each consumer group based on that PDB's resource plan in action.

An example is shown in the slide. The example just use SHARES and UTILIZATION LIMIT for simplicity.

As you can see, the REPORTS consumer group is guaranteed to receive 10% of the total system resources available. This is because PDB3 is allocated two shares out of four in the CDB resource plan, which represents 50% of the resources, and out of this 50%, the REPORTS consumer group receives one share out of five, which represents 20% of the given 50%. Similarly, the REPORTS consumer group in PDB3 is limited to 50% of total system resources.

Considerations

- Setting a PDB resource plan is optional. If not specified, all sessions within the PDB are treated equally.
- A non-CDB is plugged into a CDB with a plan:
 - The plan runs exactly the same on the PDB if:
 - Consumer groups <= 8
 - There are no subplans
 - All allocations are on level 1
 - The plan is converted.
 - The original plan is stored in the dictionary with the LEGACY status.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Note: V\$RSRC_PLAN exists in previous releases of the Oracle Database. The resource plan with con_id=1 is the active CDB resource plan.

If a non-CDB with a plan is plugged into a CDB as a new PDB, the plan acts exactly the same way in the PDB if it does not violate any of the PDBs restrictions:

- The number of consumer groups does not exceed 8.
- There are no subplans.
- All allocations are on level 1.

However, if it violates any of these restrictions, the plan is converted to something equivalent. If the plan is enabled, the converted version is used. If the converted plan does not meet your expectations, you can still use the original plan, stored in the dictionary with the LEGACY status.

Consolidated Database Replay Use Cases

Perform realistic load testing for scenarios including:

- Consolidation of servers
- Workload scale-up
- Test peak load capacity



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In many enterprises, mission-critical business functions rely on databases. The changing technology environment forces businesses to adopt the latest technology to stay competitive. The challenge is keep the transition from disrupting business.

Several questions need to be answered to be assured of a smooth transition:

- Can the new server handle all the projected consolidate databases?
- How much additional workload can the new server handle?
- What if the current workload increases?

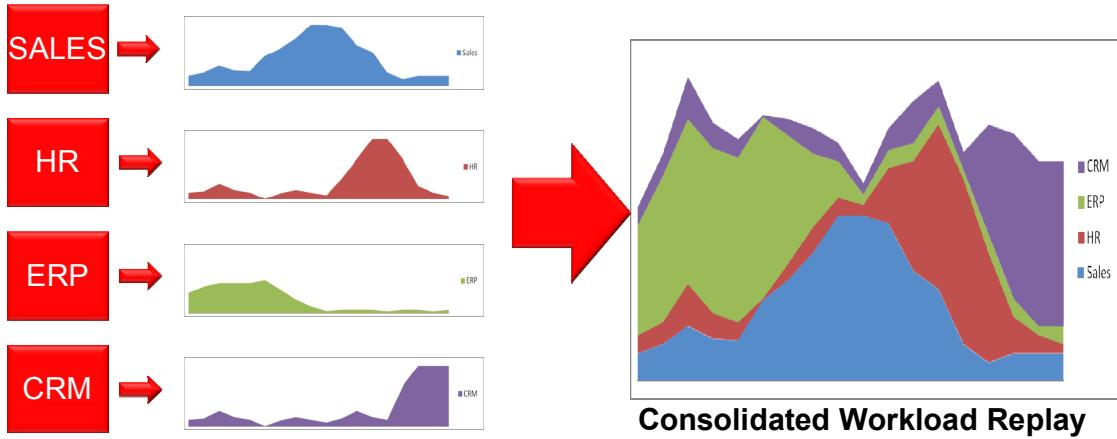
Consolidated Database Replay allows you to test with real production workloads to answer these questions. With Consolidated Database Replay, multiple captured workloads can be replayed concurrently. A captured workload can be divided into subsets covering particular time periods, and multiple captured workloads can be scheduled to run at specified times.

Because a subset of a captured workload is a self-contained captured workload, a subset can be replicated, time-shifted, and replayed with other subsets.

These features allow you test scenarios such as consolidation by capturing multiple workloads on different servers and replaying on one server, scale-up by replaying multiple subsets concurrently, and test peak load conditions by replaying captures together with varying schedules.

Use Cases: Source Workloads

Workload: Each application has distinct peaks at different times of the workday.



- Each workload captured on different databases and being consolidated is independent of the other.
- Allows multiple workloads captured from different non-CDBs or PDBs to be replayed concurrently in a single CDB

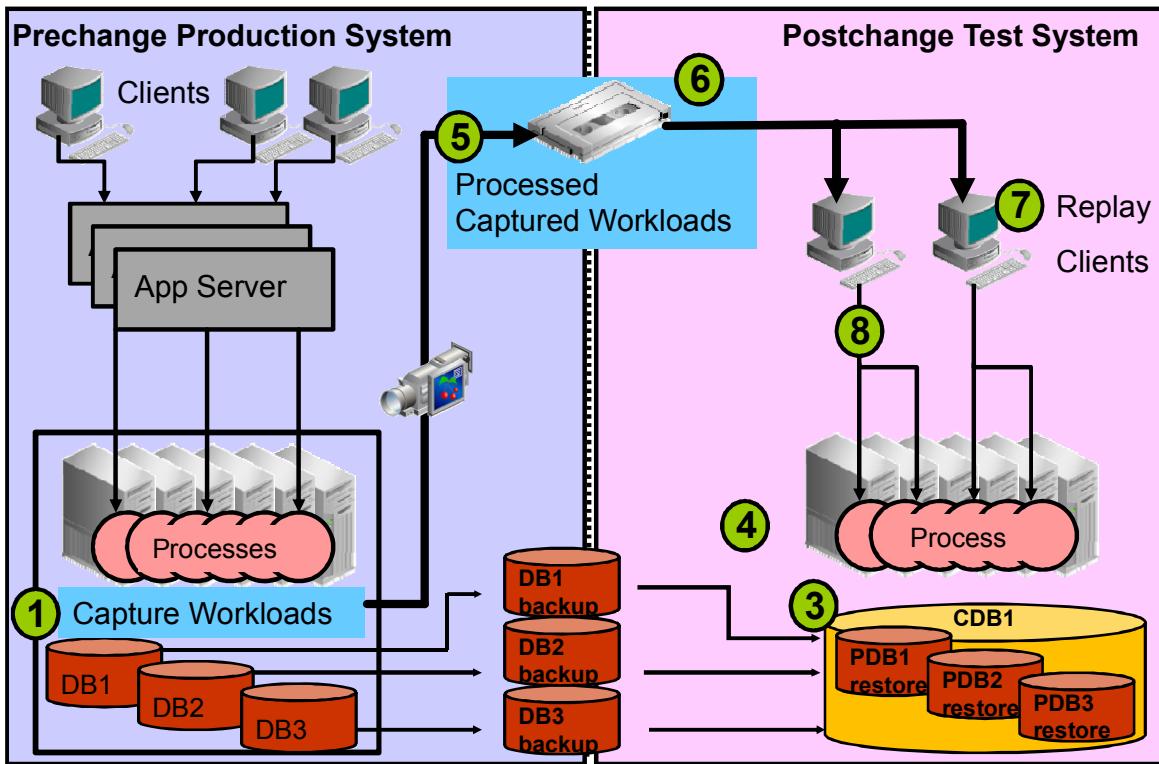
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The ability to capture workloads from more than one non-CDB, and then replay all of the workloads as a single replay against a single CDB is a valuable tool for those wishing to do database consolidation. This assumes that a PDB exists in the CDB for each of the non-CDB capture workloads. It is a simple matter to then remap the connections to the PDB service names in the CDB.

The tool also offers the ability to capture workloads from more than one PDB of different CDBs, and then replay all of the workloads as a single replay against a single CDB. This is a valuable tool for those wishing to merge dispersed PDBs into a single CDB.

The Big Picture



ORACLE

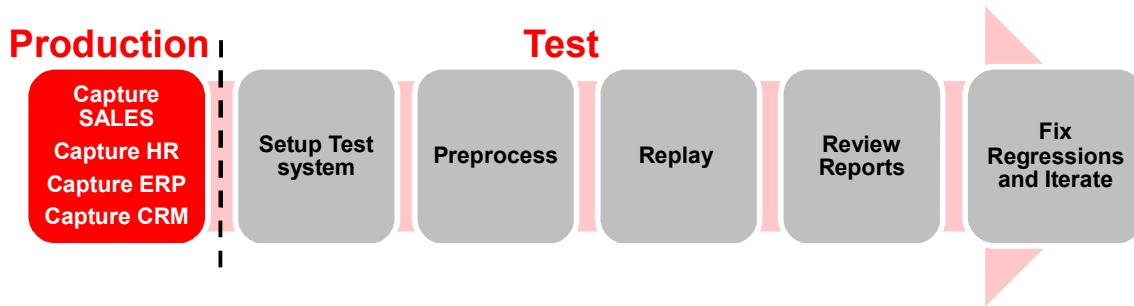
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The following are the typical steps to perform Database Replay. Note not all steps are shown in the slide.

1. Capture the workload on a database.
2. Optionally, export the AWR data.
3. Restore the replay database on a test system to match the capture database at the start of the workload capture. The DBA determines the details of this step. It depends on the backup strategy used to return the test system to the starting SCN of the capture recording. With Consolidated Database Replay in Oracle Database 12c, this may mean restoring multiple non-CDBs as PDBs (export / import) or PDBs (cloning) in a single CDB to the start conditions for each capture being replayed.
4. Make changes (such as performing an upgrade or simply test performance when all non-CDBs are consolidated within a single CDB) to the test system as required.
5. Copy the generated workload files to the replay system.
6. Preprocess the captured workload on the test system or on a system with the same version of database that exists on the test system.
7. Configure the test system for the replay. (Calibrate the `wrc` processes.)
8. Replay the workload on the restored PDBs in the single CDB.

Step 1

Consolidate multiple non-CDBs or PDBs on a single CDB.



- Capture a typical 8-hour workday in each non-DB or PDB.
- Export performance data from each DB (AWR, SQL Tuning Sets).



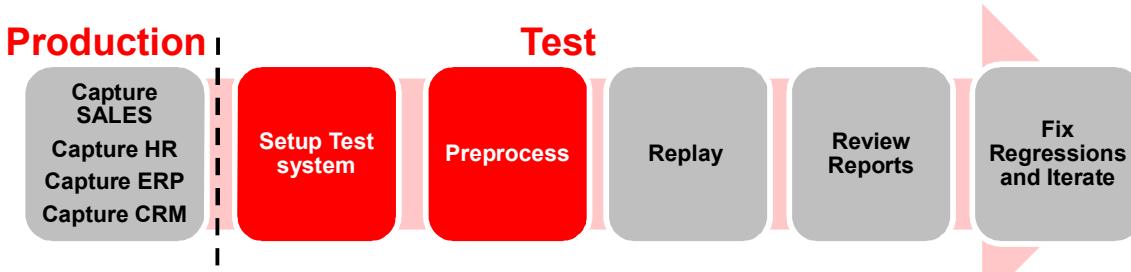
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The first step consists of capturing the workloads from the multiple non-CDBs that you consider as good candidates for consolidation into a single CDB, and/or from PDBs of distinct CDBs that you also consider as good candidates for consolidation into another CDB.

The captured files are generated in separate directories.

You may export the AWR and SQL Tuning Sets to import into the CDB before replay.

Step 2



- Move all capture files from production to staging system.
- Restore non-CDBs as PDBs or PDBs to the start conditions for each capture being replayed on the single CDB.
- Process each workload (needed once).
- Replay each workload in isolation on new hardware to obtain baseline and verify suitability.
- Flash back/restore.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

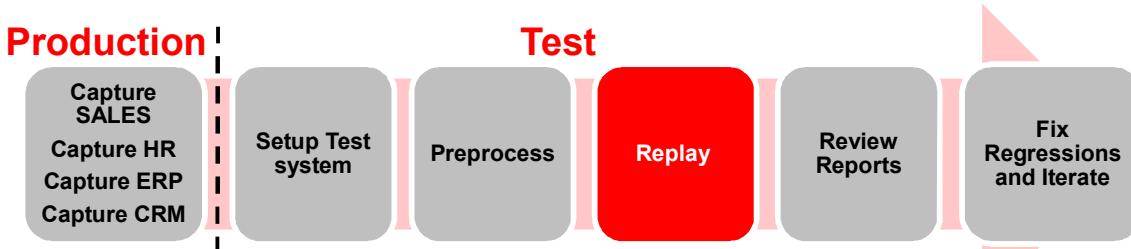
Set a replay directory on the test system to place all capture workloads in the same directory.

Then into the CDB on the test system, import each non-CDB captured (if any) into its dedicated precreated PDB so that the data correspond to what they were at the capture start. Clone each PDB captured (if any) into the test CDB so that the data correspond to what they were at the capture start. Then process all capture workloads once. This preprocessing must be executed once. The same preprocessed files can then be reused for repetitive replays.

Think about replaying each capture separately on the new CDB to verify if on this new server, within a CDB, you already get new performance results.

Then flash back the whole CDB to the time of the capture start.

Step 3



- Configure the test system for the replay.
 - Add each workload using `ADD_CAPTURE()`.
 - Initialize consolidated replay and remap all connections.
 - Optionally, remap schema users.
 - Synchronize on “Object_ID”.
 - Calibrate the `wrc` processes.
- Replay the workload on the restored database.
 - Two objects with the same name in different PDBs will be different Object IDs and will not collide during replay.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Before starting the replay itself, first create a replay schedule that sets a directory containing multiple workload captures as the current replay directory. Add all the capture workloads by capture directory name. The schedule is now saved and associated with the replay directory and can be used for a replay.

Then initialize the replay. This step reads the information from the workloads in the replay directory and populates the `DBA_WORKLOAD_CONNECTION_MAP` table. Query this table to find the connections that need to be remapped.

Each capture workload in a schedule can remap each connection in the workload to a different connect for replay. This allows each capture to be mapped to a different PDB in a CDB.

Then specify the parameters of synchronization during the replay process of multiple capture workloads. `Object_id` synchronization offers more finer grain synchronization and, therefore, more replay concurrency. The capture process tracks “`object_ids`” used by each user call, and, therefore, the replay process can minimize object collision. Two objects with the same name in different PDBs will be different Object IDs and will not collide during the replay.

Now the replay in the CDB is ready to accept workload replay client (WRC) connections to replay the multiple captures at the same time.

Step 4

Production

Capture
SALES
Capture HR
Capture ERP
Capture CRM

Test

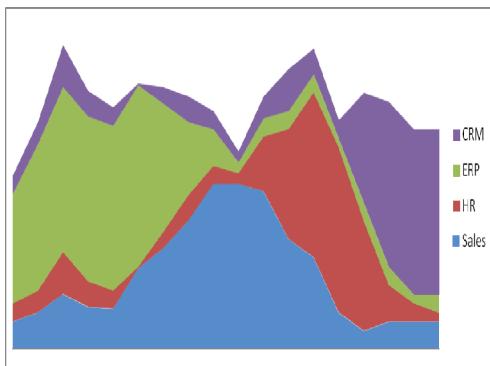
Setup Test system

Preprocess

Replay

Review Reports

Fix Regressions and Iterate



- Verify:
 - Quality of service metrics as perceived by application user per workload
 - Overall capacity
- Only replay can provide definitive answers.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Enterprise Manager Cloud Control can provide UI to perform all steps and generate the final report. The report will provide a comparison of the performance before (after the capture) and after (after the replay).

Consolidated Replay Steps

1. Create captures in separate directories.
2. Place all capture workloads in the same directory.
3. Process capture workloads for target.
4. Set replay directory.
5. Create Replay schedule.
 - a. Add capture workloads.
 - b. Specify replay order of capture workloads.
6. The replay CDB is restored to start of capture state.
7. Initialize Consolidated Replay.
8. Remap Connections.
9. Prepare Consolidated Replay.
10. Calibrate and start Workload Replay Clients(WRC)
11. Start Consolidated Replay.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Shown here is a more detailed set of steps for using Consolidated Replay. In this procedure, it is assumed that you have already captured the workloads from different non-CDBs or PDBs of different CDBs. The slides in the rest of this lesson are ordered to follow this procedure.

The steps above are a general procedure for preparing to perform a consolidated replay.

In step 5, you must start with calling the `BEGIN_REPLAY_SCHEDULE` and end with calling `END_REPLAY_SCHEDULE`.

In step 6, the replay CDB is restored to the start of capture state. A variety of methods may be used to restore the replay CDB to the proper state including full transportable export non-CDB / import PDB, cloning PDB, Data Guard snapshot standby, flashback database, and full database recovery.

In step 10, one or more `wrc` processes are started and optionally calibrated

In step 11, the `START_CONsolidated_REPLAY` procedure is executed. This procedure assumes sufficient number of `wrc` processes have been started.

To get information about the detailed procedures used for completing a Consolidated Database Replay in a CDB environment, refer to Appendix A.

Quiz

Oracle Resource Manager can be used to allocate resources to PDBs and within PDBs.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a

Summary

In this lesson, you should have learned how to:

- Monitor operations in a CDB and PDBs
- Monitor performance in a CDB and PDBs
- Manage resource allocation between PDBs and within a PDB
- Describe use cases for Consolidated Database Replay



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 8 Overview: Performance

These practices cover the following topics:

- Creating and setting CDB Resource Manager plan and directives to share resources between PDBs
- Using ADDM to get recommendations using Enterprise Manager Cloud Control



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

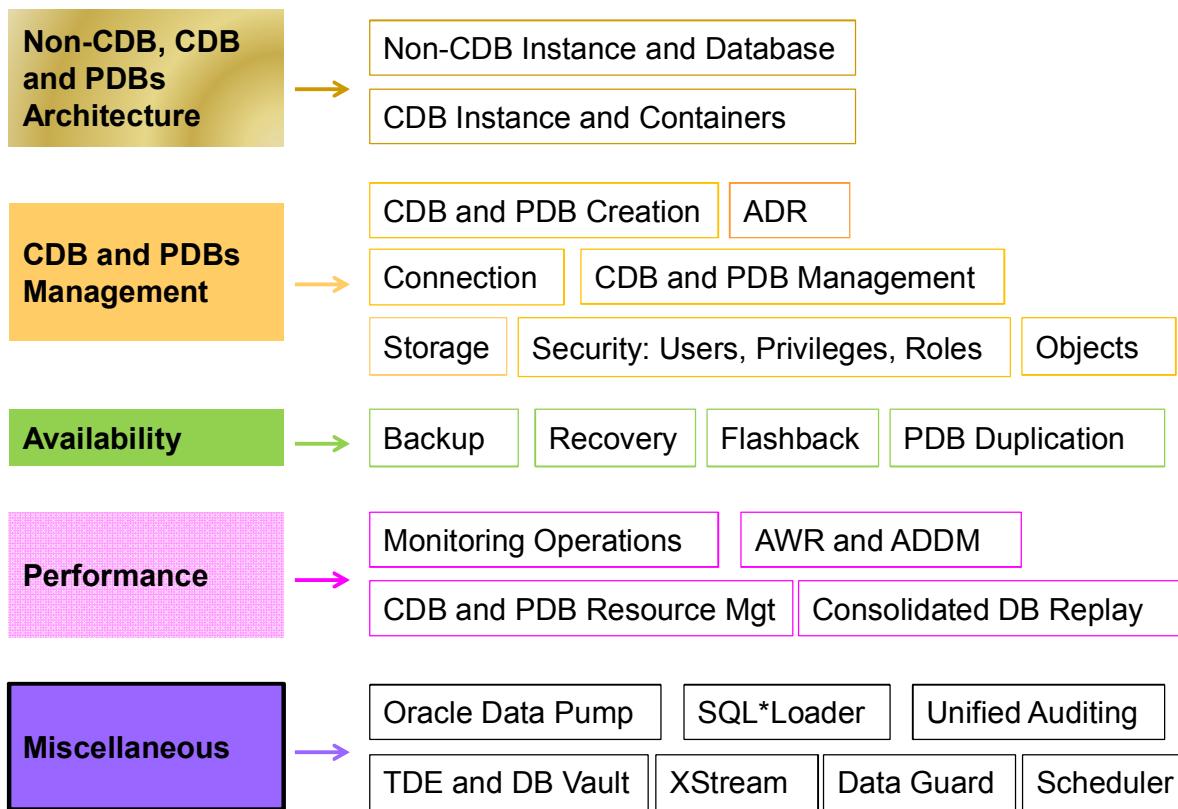
9

Miscellaneous

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Course Structure



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This lesson explains how to:

- Export and import data from and into a CDB or PDB
- Load data into a PDB
- Audit data in a CDB and PDB
- Secure data with encryption and Database Vault in a CDB and PDB
- Replicate data with capture / apply changes from / to a PDB or the entire CDB with XStream
- Use Data Guard with a CDB and PDB
- Schedule operations in a CDB and PDB

Objectives

After completing this lesson, you should be able to:

- Use Oracle Data Pump Export / Import with a non-CDB or CDB and PDB
- Use SQL*Loader to load data into a PDB
- Audit data of a CDB and PDB using Unified Auditing
- Secure data in a CDB and PDB using Transparent Data Encryption and Database Vault
- Describe the limits of data replication
- Describe XStreams usage with a PDB or the entire CDB
- Describe Data Guard with a CDB and PDB
- Schedule operations in a PDB using Oracle Scheduler
- Mine PDB statements using LogMiner



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

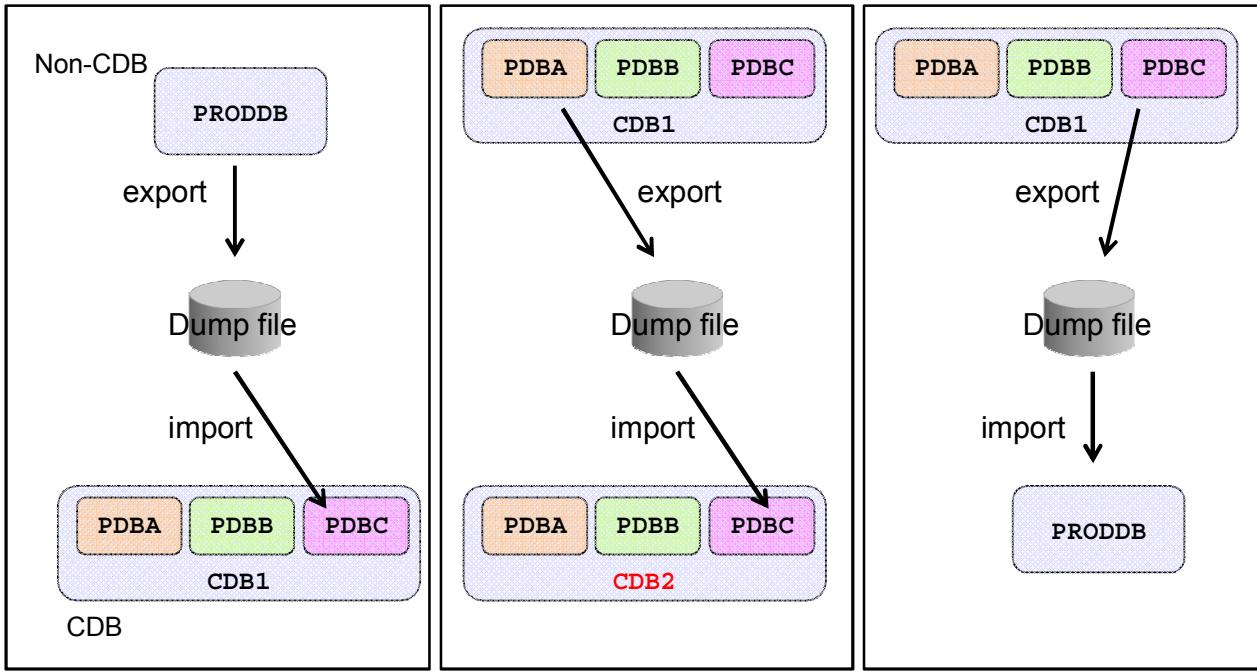
Note: To get detailed information on how to perform any of the operations explained in this lesson, refer to the following guides in the Oracle documentation:

- *Oracle Database Administrator's Guide 12c Release 1 (12.1) - Transporting Data*
- *Oracle Database Utilities 12c Release 1 (12.1)*
- *Oracle Database Security Guide 12c Release 1 (12.1)*
- *Oracle Database Advanced Security Guide 12c Release 1 (12.1)*
- *Oracle Database XStream Guide 12c Release 1 (12.1)*
- *Oracle Data Guard Concepts and Administration 12c Release 1 (12.1) - Creating a Physical Standby Database*
- *Oracle Data Guard Concepts and Administration 12c Release 1 (12.1) - Creating a Logical Standby Database*

Refer to other sources of information:

- *Oracle Database 12c New Features Demo Series* demonstrations under Oracle Learning Library:
 - *Unified Auditing*

Using Oracle Data Pump with PDBs



Use the **PDB service name** to export from or import into a PDB.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

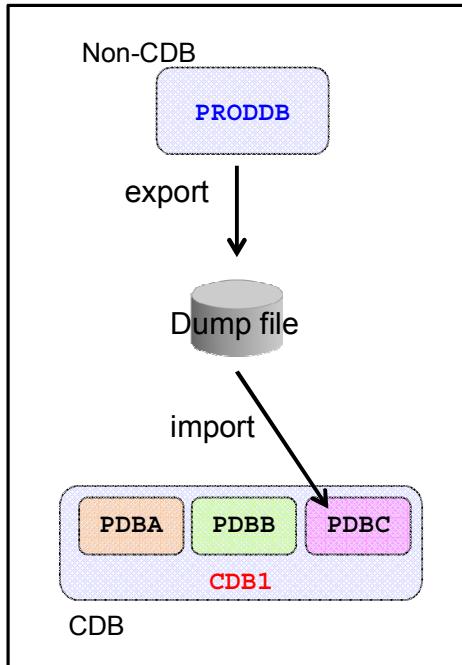
All types of export and import operations are possible between non-CDBs and PDBs. There are no supported CDB-wide Data Pump export or import operations, but only per-PDB Data Pump export or import operations, specifying the service name in the `USERID` clause.

- Export data from a non-CDB to import it into a PDB of a CDB.
- Export data from a PDB to import it into another PDB within the same CDB.
- Export data from a PDB to import it into a PDB of another CDB.
- Export data from a PDB to import it into a non-CDB.

Different types of Data Pump export and import are possible:

- Conventional export and import to export a full database (non-CDB or PDB) to import it into another database (non-CDB or PDB)
- Full transportable export and import to transport a full database (non-CDB or PDB) to import it into another database (non-CDB or PDB)
- Conventional or transportable tablespace export and import to export a full tablespace of a non-CDB or PDB to import it into another tablespace of a non-CDB or PDB
- Schema export and import to export a full schema of a non-CDB or PDB to import it into another tablespace of a non-CDB or PDB
- Table export and import to export a table of a non-CDB or PDB to import it into a non-CDB or PDB

Exporting from non-CDB and Importing into PDB



1. Export **PRODDDB** with **FULL** clause:

```
$ expdp system@PRODDDB FULL=Y
DUMPFILE=proddb.dmp
```

2. If **PDBC** does not exist in **CDB1**, create **PDBC** in **CDB1**:

```
SQL> CONNECT sys@CDB1
SQL> CREATE PLUGGABLE DATABASE
2 PDBC USING pdb$seed;
```

3. Open **PDBC**.
4. Create a Data Pump directory in **PDBC**.
5. Copy the dumpfile to the directory.
6. Create same **PRODDDB** tablespaces in **PDBC** for new local users' objects.
7. Import into **PDBC** with **FULL** and **REMAP** clauses:

```
$ impdp system@PDBC FULL=Y
DUMPFILE=proddb.dmp
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To export data from a non-CDB and import it into a PDB of a CDB, use the steps as described in the slide.

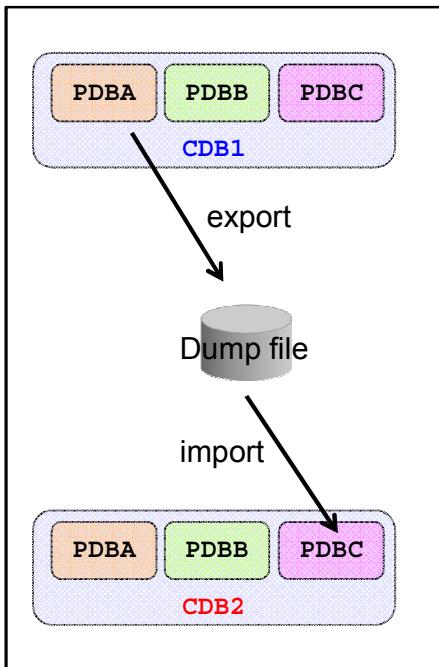
The choice made in the slide is to perform a conventional full database export from the non-CDB and a conventional full database import into the PDB. You can also perform a full transportable or a tablespace or schema or table-level export and import.

The tablespace export and import can be of either types: conventional or transportable.

The users exported from the non-CDB are re-created as local users in the PDB.

The tablespaces for the new local users and objects need to be created in the PDB before the import.

Exporting and Importing Between PDBs



1. Export **PDBA** of **CDB1** with FULL clause:

```
$ expdp system@PDBA FULL=Y ...
```
2. If **PDBC** does not exist in **CDB2**, create **PDBC** in **CDB2**:

```
SQL> CONNECT sys@CDB2
SQL> CREATE PLUGGABLE DATABASE
2  PDBC USING pdb$seed;
```
3. Open **PDBC**.
4. Create a Data Pump directory in **PDBC**.
5. Copy the dumpfile to the directory.
6. Create same **PDBA** tablespaces in **PDBC** for new local users objects.
7. Import into **PDBC** of **CDB2** with FULL and REMAP clauses:

```
$ impdp system@PDBC FULL=Y
REMAP SCHEMA=c##u:lu...
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To export data from a PDB and import it into a PDB of the same or another CDB, use the steps as described in the slide.

The choice made in the slide is to perform a full database export from the PDB and a full database import into a PDB of another CDB. You can also perform a full transportable or a tablespace or schema or table-level export and import.

The tablespace export and import can be of either types: conventional or transportable.

The local users exported from the PDB are re-created as local users in the target PDB.

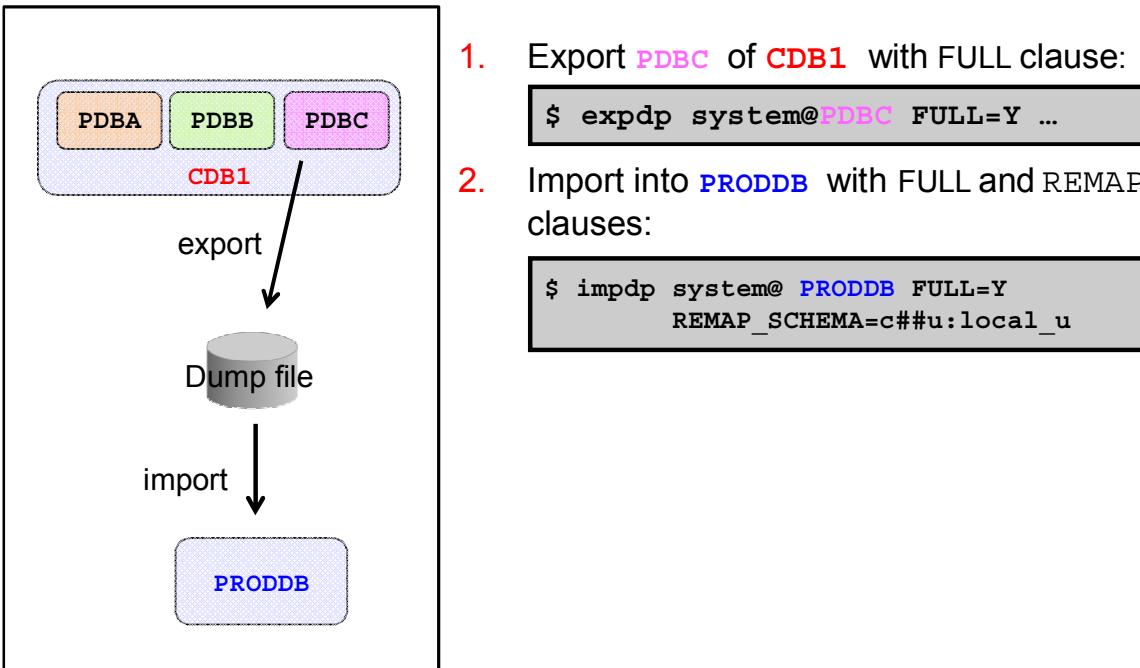
The common users are not re-created because their names prefixed with C## imply that a common user should be created. The statement fails with the following error message:

```
ORA-65094:invalid local user or role name
```

The only way to have common users re-created as local users is to use the clause REMAP_SCHEMA=C##xxx:local_user_name.

You can also create a common user in the root.

Exporting from PDB and Importing into non-CDB



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To export data from a PDB and import it into a non-CDB, use the steps as described in the slide.

The choice made in the slide is to perform a full database export from the PDB and a full database import into a non-CDB. You can also perform a full transportable or tablespace or schema or table-level export and import.

The tablespace export and import can be of either types: conventional or transportable.

The local users exported from the PDB are re-created as conventional users in the target non-CDB.

The common users are not re-created because their names prefixed with C## imply that a common user should be created. Type of users are not recognized in a non-CDB. The statement fails with the following error message:

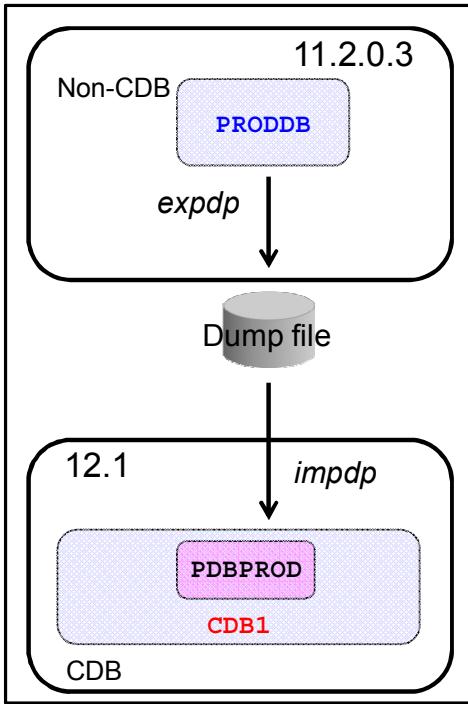
ORA-65094:invalid local user or role name

The only way to have common users re-created as conventional users is to use the clause REMAP_SCHEMA=C##xxx:user_name.

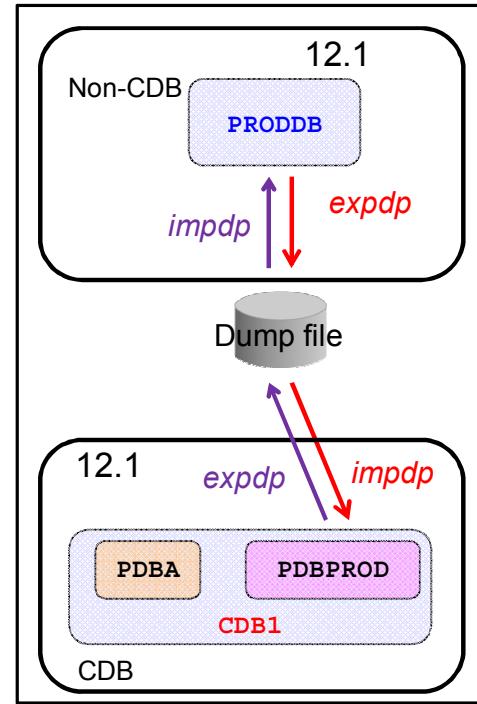
Full Transportable Export/Import:

Overview

UPGRADE



TRANSPORT



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The full transportable export/import feature is useful in several scenarios:

- Upgrading to a new release of Oracle Database: You can use full transportable export/import to upgrade a database from 11.2.0.3 or higher to Oracle Database 12c. To do so:
 - Install Oracle Database 12c and create an empty database, non-CDB or CDB.
 - Use full transportable export/import to transport the 11.2.0.3 non-CDB into the Oracle Database 12c non-CDB or PDB.
- Transporting a non-CDB into a non-CDB or CDB: Transported into a CDB, the transported database becomes a PDB associated with the CDB. Full transportable export/import can move an 11.2.0.3 or higher database into an Oracle Database 12c database efficiently.

Full Transportable Export/Import: Usage

A full transportable export exports all objects and data necessary to create a complete copy of the database.

- TRANSPORTABLE=ALWAYS parameter
- FULL parameter

```
$ expdp user_name@pdb FULL=y DUMPFILE=expdat.dmp  
      DIRECTORY=data_pump_dir TRANSPORTABLE=always  
      VERSION=12.0 LOGFILE=export.log
```

A full transportable import imports a dump file only if it has been created using the transportable option during export.

- TRANSPORT_DATAFILES
- If NETWORK_LINK used, requires:
 - TRANSPORTABLE=ALWAYS parameter



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Full Transportable Export

A full transportable export exports all objects and data necessary to create a complete copy of the database. A mix of data movement methods is used:

- Objects residing in transportable tablespaces have only their metadata unloaded into the dump file set. The data itself is moved when you copy the data files to the target database. The data files that must be copied are listed at the end of the log file for the export operation.
- Objects residing in nontransportable tablespaces (for example, SYSTEM and SYSAUX) have both their metadata and data unloaded into the dump file set, using direct path unload and external tables.

The example shows a full transportable export of a PDB using the VERSION parameter. The VERSION parameter is required if the source database is an 11.2.0.3 or a higher Oracle Database 11g release.

Performing a full transportable export has the following restrictions:

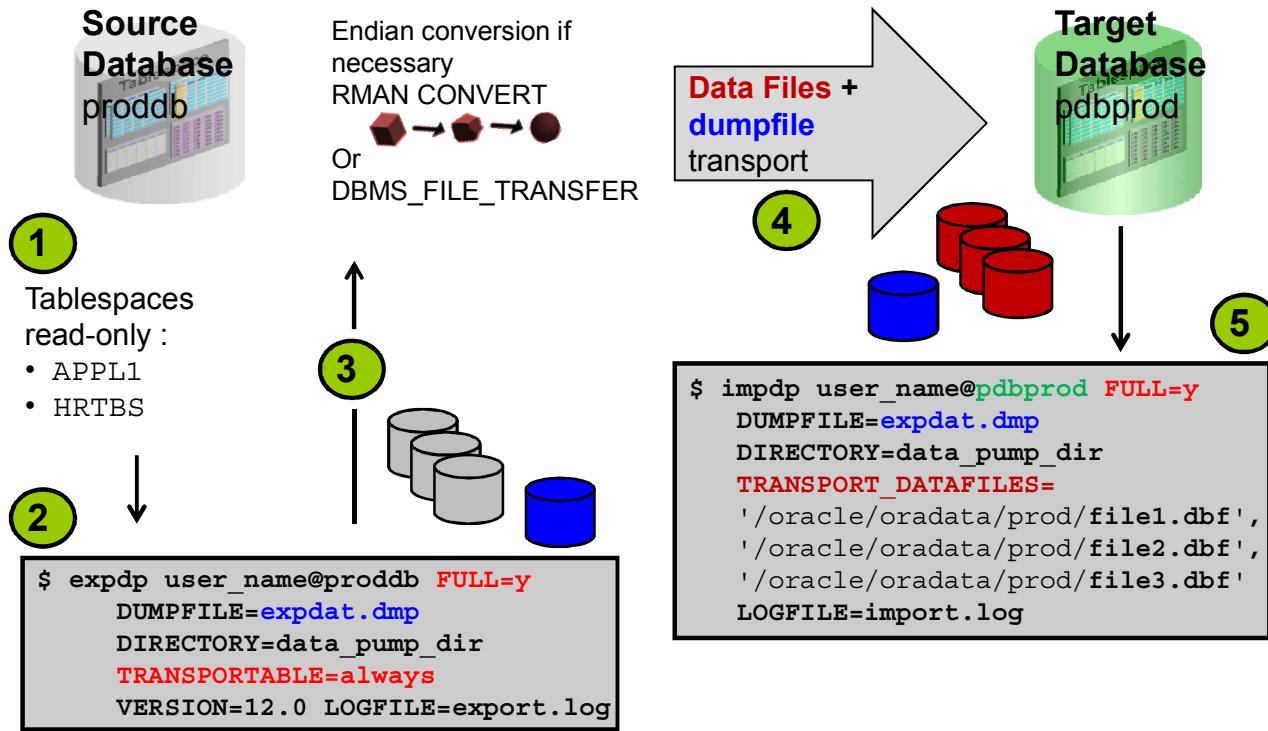
- If the database being exported contains either encrypted tablespaces or tables with encrypted columns (either Transparent Data Encryption [TDE] columns or SecureFile LOB columns), then the `ENCRYPTION_PASSWORD` parameter must also be supplied.
- The source and target databases must be on platforms with the same endianness if there are encrypted tablespaces in the source database.
- If the source platform and the target platform are of different endianness, you must convert the data being transported so that it is in the format of the target platform. Use either the `DBMS_FILE_TRANSFER` package or the `RMAN CONVERT` command.
- A full transportable export is not restartable.
- All objects with storage that are selected for export must have all of their storage segments either entirely within administrative, non-transportable tablespaces (`SYSTEM / SYSAUX`) or entirely within user-defined, transportable tablespaces. Storage for a single object cannot straddle the two kinds of tablespaces.
- When transporting a database over the network using full transportable export, tables with `LONG` or `LONG RAW` columns that reside in administrative tablespaces (such as `SYSTEM` or `SYSAUX`) are not supported.
- When transporting a database over the network using full transportable export, auditing cannot be enabled for tables stored in an administrative tablespace (such as `SYSTEM` and `SYSAUX`) if the audit trail information itself is stored in a user-defined tablespace.
- If both the source and target databases are running Oracle Database 12c Release 1 (12.1), then to perform a full transportable export, either the Oracle Data Pump `VERSION` parameter must be set to at least 12.0. or the `COMPATIBLE` database initialization parameter must be set to at least 12.0 or later.
- Full transportable exports are supported from an 11.2.0.3 source database.

Full Transportable Import

Performing a full transportable import has the following requirements:

- If you are using a network link, then the database specified on the `NETWORK_LINK` parameter must be Oracle Database 11g release 2 (11.2.0.3) or later, and the Oracle Data Pump `VERSION` parameter must be set to at least 12. (In a non-network import, `VERSION=12` is implicitly determined from the dump file.)
- If the source platform and the target platform are of different endianness, then you must convert the data being transported so that it is in the format of the target platform. You can use the `DBMS_FILE_TRANSFER` package or the `RMAN CONVERT` command to convert the data.
- A full transportable import of encrypted tablespaces is not supported in network mode or dump file mode if the source and target platforms do not have the same endianess.
- When transporting a database over the network using full transportable import, tables with `LONG` or `LONG RAW` columns that reside in administrative tablespaces (such as `SYSTEM` or `SYSAUX`) are not supported.
- When transporting a database over the network using full transportable import, auditing cannot be enabled for tables stored in an administrative tablespace (such as `SYSTEM` and `SYSAUX`) if the audit trail information itself is stored in a user-defined tablespace.

Full Transportable Export/Import: Example



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To perform a full transportable operation, follow the steps as below:

1. Before the export, make all of the user-defined tablespaces in the database read-only.
2. Invoke the Oracle Data Pump export utility as a user with the DATAPUMP_EXP_FULL_DATABASE role and specify the full transportable export options: FULL=Y, TRANSPORTABLE=ALWAYS. The LOGFILE parameter is important because it will contain the list of data files that need to be transported for the import operation. To perform the operation on an 11.2.0.3 non-CDBs, use the VERSION parameter. Full transportable import is supported only for Oracle Database 12c non-CDBs and PDBs.
3. Before the import, transport the dump file.
4. Transport the datafiles that you may have converted. If you are transporting the database to a platform different from the source platform, then determine if cross-platform database transport is supported for both the source and target platforms. If both platforms have the same endianness, no conversion is necessary. Otherwise you must do a conversion of each tablespace in the database either at the source or target database using either DBMS_FILE_TRANSFER or RMAN CONVERT command.
5. Invoke the Oracle Data Pump import utility as a user with the DATAPUMP_IMP_FULL_DATABASE role and specify the full transportable import options: FULL=Y, TRANSPORT_DATAFILES.
6. Make the source tablespaces read-write. You can perform this operation before step 5.

Transporting a Database Over the Network: Example

Transport a database over the network: perform an import using the NETWORK_LINK parameter.

1. Create a database link in the target to the source database
2. Make the user-defined tablespaces in the source database read-only
3. Transport the datafiles for all of the user-defined tablespaces from the source to the target location
4. Perform conversion of the datafiles if necessary
5. Import in the target database

```
$ impdp username@dbname full=Y network_link=sourcedb  
      transportable=always  
      transport_datafiles='/oracle/oradata/prod/sales01.dbf',  
                           '/oracle/oradata/prod/cust01.dbf'  
      version=12 logfile=import.log
```

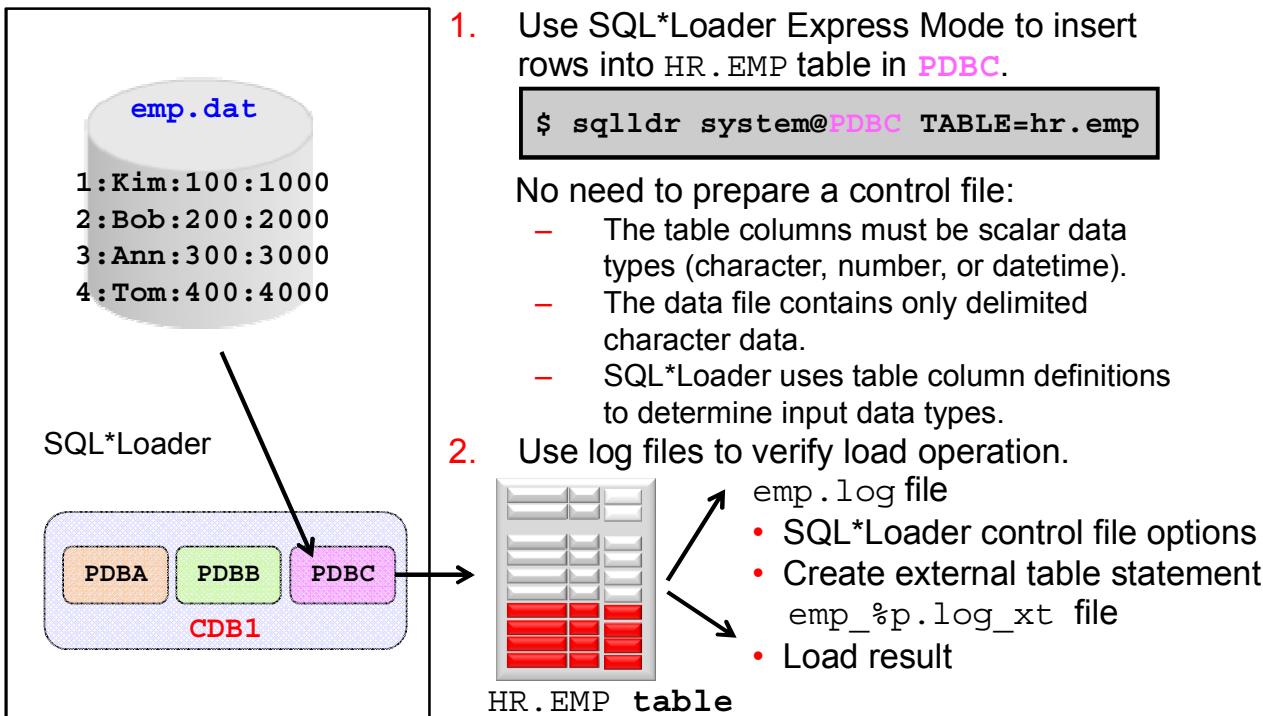


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To transport a database over the network, use import with the NETWORK_LINK parameter. The import is performed using a database link, and there is no dump file involved.

- If the source database is an 11.2.0.3 database or a higher Oracle Database 11g release database, then the VERSION parameter is required and must be set to 12. If the source database is an Oracle Database 12c non-CDB or PDB, then the VERSION parameter is not required.
- If the source or target database is a PDB, use the PDB service name in the USERID clause.
- The Oracle Data Pump network import copies the metadata for objects contained within the user-defined tablespaces and both the metadata and data for user-defined objects contained within the administrative tablespaces, such as SYSTEM and SYSAUX.
- When the import is complete, the user-defined tablespaces are in read-write mode.
- Make the user-defined tablespaces read-write again at the source database.

Using SQL*Loader with PDBs



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To load records from a text file into a PDB, if you activate SQL*Loader Express Mode, specifying only the username and the `TABLE` parameter, it uses default settings for a number of other parameters. You can override most of the defaults by specifying additional parameters on the command line.

SQL*Loader express mode generates two files. The names of the log files come from the name of the table (by default).

- A log file that includes:
 - The control file output
 - A SQL script for creating the external table and performing the load using a SQL `INSERT AS SELECT` statement

Neither the control file nor the SQL script are used by SQL*Loader express mode. They are made available to you in case you want to use them as a starting point to perform operations using regular SQL*Loader or stand-alone external tables.

- A log file is similar to a SQL*Loader log file that describes the result of the operation. The "%p" represents the process ID of the SQL*Loader process.

Quiz

Which of the following statements are true?

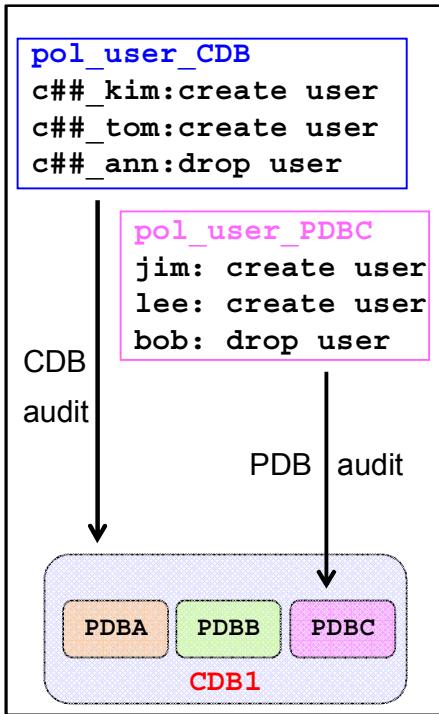
- a. You cannot export a PDB and import into a non-CDB.
- b. You can export a PDB and import into another PDB within the same CDB.
- c. You cannot export a PDB and import into a PDB of another CDB.
- d. You can only export a non-CDB and import into a non-CDB.
- e. You can upgrade a 11.2.0.3 non-CDB to a 12c PDB using Data Pump export and import.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b, e

Auditing Actions in a CDB and PDBs



Choose CDB or PDB level:

1. Connect to the root or a PDB.
2. Create audit policies:
 - Audit options: Systemwide or object-specific or role
 - Optional **WHEN** condition
EVALUATE PER STATEMENT
 - **CONTAINER** = CURRENT | ALL
3. Enable/disable audit policies:
AUDIT and **NOAUDIT**
 - Define users audited: all by default

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Auditing actions based on system or object privileges or roles are performed, provided that you create and enable audit policies. The policies can be created either at the CDB level for the whole CDB or at a PDB level for a single PDB. What are audit policies?

- A named group of audit options that enable you to audit a particular aspect of user behavior in the database by using system, object, or role privileges
- Created with the `CREATE AUDIT POLICY` command requiring the `AUDIT_ADMIN` role
- Enforced conditionally and evaluated during the statement execution or once during the session or at the instance level
- Applied either in the root or a PDB: This provides the ability to create audit policies used by all PDBs and audit policies used exclusively for each PDB. An audit configuration that is not enforced across all PDBs means it applies only within a PDB and is not visible outside it.
- In the example in the slide, the `pol_user_CDB` created from the root will audit any `CREATE` or `DROP USER` performed by specific users (`C##_KIM`, `C##_TOM`, and `C##_ANN`) and the `pol_user_PDBC` created from the PDB will audit any `CREATE` or `DROP USER` performed by specific users (`JIM`, `LEE`, and `BOB`)
- Not active until explicitly enabled with the `AUDIT` command
- Applied with the exception of certain user lists

Creating an Audit Policy at CDB/PDB Level

Create audit policies for the whole CDB or a specific PDB.

- Connect to the root or the specific PDB.

`SQL> CONNECT / AS SYSDBA` or `SQL> CONNECT system@PDBC`

- Create audit policies based on `system` privilege, `actions`, and or `roles`.

```
SQL> CREATE AUDIT POLICY audit_mixed_pol1_CDB
  2      PRIVILEGES DROP ANY TABLE
  3      ACTIONS      CREATE TABLE, DROP TABLE, TRUNCATE TABLE
  4      ROLES        emp_role;
```

- Create audit policies based on `object-specific` options.

```
SQL> CREATE AUDIT POLICY audit_objpriv_pol2_PDB
  2      ACTIONS EXECUTE, GRANT ON hr.raise_salary_proc;
```

- Condition and evaluation `PER SESSION`

```
SQL> CREATE AUDIT POLICY audit_mixed_pol3_PDB ROLES hr_role
  2 WHEN 'SYS_CONTEXT (''USERENV'', ''SESSION_USER'')='JIM''
  3 EVALUATE PER SESSION;
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To create an audit policy for the whole CDB, you must first connect to the root. To create an audit policy for a specific PDB, you must first connect to the PDB. The `CONTAINER` clause of the `CREATE AUDIT POLICY` is automatically set to `ALL` in the first case (meaning all PDBs) and `CURRENT` in the second case. Then define systemwide or object-specific audit options.

The systemwide options can be of three types:

- The **privilege audit option** audits all events, which exercise the specified system privilege, as in the first example in the slide with the `PRIVILEGES` clause.
- The **action audit option** indicates which RDBMS action should be audited, such as the first example in the slide with the `ACTIONS` clause.
- The **role audit option** audits the use of all system or object privileges granted directly to the role `EMP_ROLE`, as in the first example in the slide with the `ROLES` clause.

You can configure privilege, action, and role audit options in the same audit policy, as shown in the first example at the CDB level.

Object-specific options are the second type of audit option. These options are actions that are specific to objects in the database. The second example creates an audit policy to audit any execute action on any procedural object, and any grant on the `HR. RAISE_SALARY_PROC` procedure at the CDB level.

The audit policies can evaluate the condition per statement, once per session (as in the last example in the slide) or once per instance.

Enabling / Disabling the Audit Policy

Enable audit policies while connected to the root or to a PDB:

- Apply to all users.

```
SQL> AUDIT POLICY audit_mixed_pol1_CDB;
```

- Apply only to some users.

```
SQL> CONNECT system@PDBC
SQL> AUDIT POLICY audit_objpriv_pol2_PDB BY scott, oe;
```

```
SQL> AUDIT POLICY audit_mixed_pol1_CDB;
audit policy AUDIT_MIXED_POL1_CDB
*
ERROR at line 1:
ORA-46357: Audit policy AUDIT_MIXED_POL1_CDB not found.
```

- Audit the recording based on failed or succeeded actions.

```
SQL> AUDIT POLICY audit_syspriv_pol1 WHENEVER SUCCESSFUL;
```

- Disable audit policies by using the NOAUDIT command.

```
SQL> NOAUDIT POLICY auditpol;
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Audit Policy Enabling

The next step is to enable the audit policy by using the AUDIT statement. The statement AUDIT POLICY has to be executed in the root if the policy is a CDB policy or in the PDB if the policy is a PDB policy.

The second example shows how to enable an audit policy of a specific PDB once connected to the PDB. The third example shows that you cannot enable a CDB audit policy while connected to a PDB. The PDB does not know about the CDB policy. The same is true if you attempt to enable a PDB audit policy while connected to the root. The root does not know about the PDB policy.

All users are audited by default, except if you define a list of those audited.

- Apply the audit policy to one or more users by using the BY clause.
- Exclude some users by using the EXCEPT clause.

Audit records are generated whether the user's actions failed or succeeded. If you want to audit actions only when the user's actions succeeded or failed, use the WHENEVER SUCCESSFUL or WHENEVER NOT SUCCESSFUL clause. When you omit the WHENEVER clause, the statement is audited whether the action is successful or not, and the RETURN_CODE column displays whether the action succeeded or not.

Audit Policy Disabling

To disable an audit policy, use the NOAUDIT command.

Viewing the Audit Policy

- View all CDB or PDB audit policies.

```
SQL> CONNECT / AS SYSDBA
```

```
SQL> SELECT POLICY_NAME,
  2        AUDIT_OPTION
  3   FROM AUDIT_UNIFIED_POLICIES;

POLICY_NAME AUDIT_OPTION
-----
POL1_CDB      DELETE
POL2_CDB      TRUNCATE TABLE
```

```
SQL> CONNECT system@PDBC
```

```
SQL> SELECT POLICY_NAME,
  2        AUDIT_OPTION
  3   FROM AUDIT_UNIFIED_POLICIES;

POLICY_NAME AUDIT_OPTION
-----
POL1_PDBC    CREATE VIEW
POL2_PDBC    CREATE SEQUENCE
```

Or

- View the CDB- or PDB-enabled audit policies.

```
SQL> SELECT POLICY_NAME
  FROM AUDIT_UNIFIED_ENABLED_POLICIES;

POLICY_NAME
-----
POL2_CDB
```

```
SQL> SELECT POLICY_NAME
  FROM AUDIT_UNIFIED_ENABLED_POLICIES;

POLICY_NAME
-----
POL2_PDBC
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Verify the list of audit policies created in the AUDIT_UNIFIED_POLICIES view.

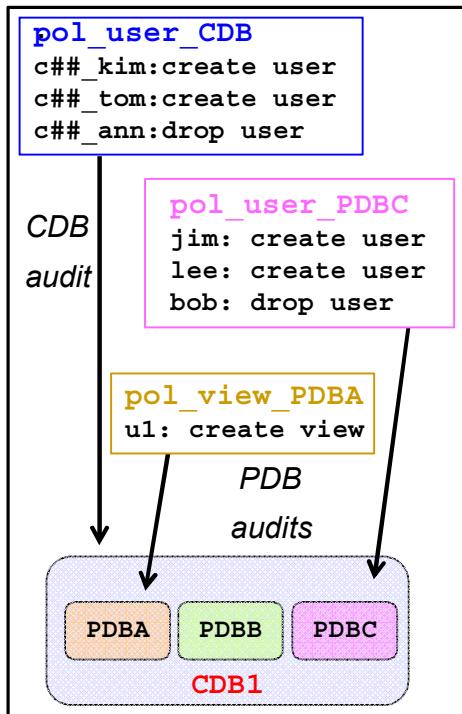
- The CONDITION_EVAL_OPT column defines when the condition, if any, is evaluated.

Verify the list of enabled audit policies in the AUDIT_UNIFIED_ENABLED_POLICIES view.

- The ENABLED_OPT column defines if a list of audited users is defined with the BY value or an exception list of excluded users is defined with the EXCEPT value. The audited or excluded users are listed in the USER_NAME column.
- The SUCCESS and FAILURE columns define if the policy generates audit records only when the user's actions succeed or fail.

Viewing the Audit Records

CDB_UNIFIED_AUDIT_TRAIL



Audit records are generated in each container with a distinct DBID.

- When connected to **PDBA**, only **PDBA** rows in **UNIFIED_AUDIT_TRAIL**
- When connected to **PDBC**, only **PDBC** rows in **UNIFIED_AUDIT_TRAIL**
- When connected to the **root**, only **root** rows in **UNIFIED_AUDIT_TRAIL**
- Query **CDB_UNIFIED_AUDIT_TRAIL** to get a consolidated view of all containers audited rows.

```

SQL> SELECT con_id, ACTION_NAME
  2  FROM CDB_UNIFIED_AUDIT_TRAIL;

  CON_ID ACTION_NAME
  -----
  1  ALTER USER
  3  CREATE VIEW
  4  CREATE TABLE
  
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In a non-CDB, the **UNIFIED_AUDIT_TRAIL** view displays all audit rows. In a CDB, the **UNIFIED_AUDIT_TRAIL** view displays all audit rows of the container you are connected to.

For example, if an audit policy has been created and enabled in **PDBA**, and an audit policy has been created and enabled in **PDBC**, audit records are generated in both PDBs with a distinct DBID.

- When connected to **PDB1**, the **UNIFIED_AUDIT_TRAIL** view displays only **PDB1** rows.
- When connected to **PDB2**, the **UNIFIED_AUDIT_TRAIL** view displays only **PDB2** rows.
- When connected to the **root**, the **UNIFIED_AUDIT_TRAIL** view displays only **root** rows.

You can query the **CDB_UNIFIED_AUDIT_TRAIL** view. This is a consolidated view of all containers, to retrieve all audit rows of the CDB.

Dropping the Audit Policy

1. Disable the audit policy.

or

```
SQL> CONNECT / AS SYSDBA
```

```
SQL> CONNECT system@PDBC
```

```
SQL> NOAUDIT POLICY CDB_pol1;
```

```
SQL> NOAUDIT POLICY PDBC_pol1;
```

2. Drop the audit policy.

```
SQL> DROP AUDIT POLICY  
2          CDB_pol1;
```

```
SQL> DROP AUDIT POLICY  
2          PDBC_pol1;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To drop a unified audit policy, first disable it, and then run the `DROP AUDIT POLICY` statement to remove it. If a unified system-related audit policy is already enabled for a session, the effect of dropping a system-related audit policy is not seen by this existing session. Until that time, the unified system-related audit policy's settings remain in effect. However, the effect is immediate for object-related unified audit policies.

In a CDB, you disable and drop a CDB audit policy only from `root` and a local audit policy only from the PDB to which it applies.

Audit Cleanup

Clean up records in SYS.UNIFIED_AUDIT_TRAIL:

- Schedule an automatic purge job to purge a PDB's records.

```
DBMS_AUDIT_MGMT.CREATE_PURGE_JOB  
(AUDIT_TRAIL_TYPE=> DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,  
AUDIT_TRAIL_PURGE_INTERVAL => 12,  
AUDIT_TRAIL_PURGE_NAME => 'Audit_Trail_PDB',  
USE_LAST_ARCH_TIMESTAMP => TRUE,  
CONTAINER => DBMS_AUDIT_MGMT.CONTAINER_CURRENT);
```

- Manually purge the ALL audit records of the CDB.

```
DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL(  
AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,  
CONTAINER => DBMS_AUDIT_MGMT.CONTAINER_ALL)
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To perform the audit trail purge tasks, you use the DBMS_AUDIT_MGMT package. You must have the AUDIT_ADMIN role to use the package.

The DBMS_AUDIT_MGMT.CREATE_PURGE_JOB procedure owns a new attribute CONTAINER. Setting CONTAINER to CONTAINER_CURRENT deletes audit trail records for the current PDB. Setting CONTAINER to CONTAINER_ALL deletes audit trail records for all PDBs creating a job in the root, and the invocation of this job will invoke cleanup in all the PDBs.

Specifies whether the last archived timestamp should be used for deciding on the records that should be deleted.

Setting USE_LAST_ARCH_TIMESTAMP to TRUE indicates that only audit records created before the last archive timestamp should be deleted. A value of FALSE indicates that all audit records should be deleted. The default value is TRUE. Oracle recommends using this value, because this helps guard against inadvertent deletion of records.

You can automate the cleanup process by creating and scheduling a cleanup purge job, or you can manually run a cleanup purge job. If you manually run cleanup purge jobs, use the DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL procedure with a new type value of DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED.

Quiz

Select the view that you use to view all audit records of a CDB.

- a. DBA_UNIFIED_AUDIT_POLICIES
- b. UNIFIED_AUDIT_ENABLED_POLICIES
- c. CDB_UNIFIED_AUDIT_TRAIL
- d. CDB_UNIFIED_AUDIT_POLICIES
- e. UNIFIED_AUDIT_TRAIL



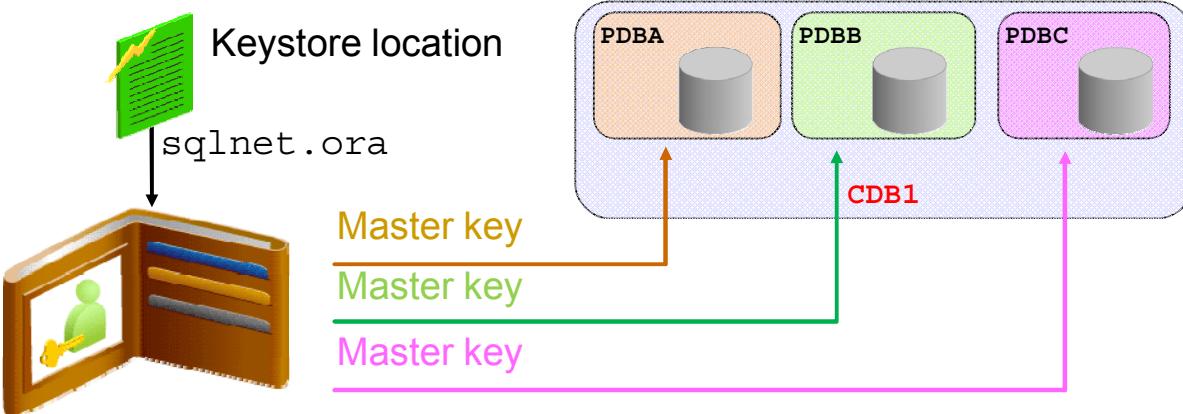
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: c

Securing Data with Transparent Data Encryption

One master key per PDB to encrypt PDB data

- Each PDB has its own master key used to encrypt data in the PDB.
- The master key must be transported from the source database keystore to the target database keystore when a PDB is moved from one host to another.



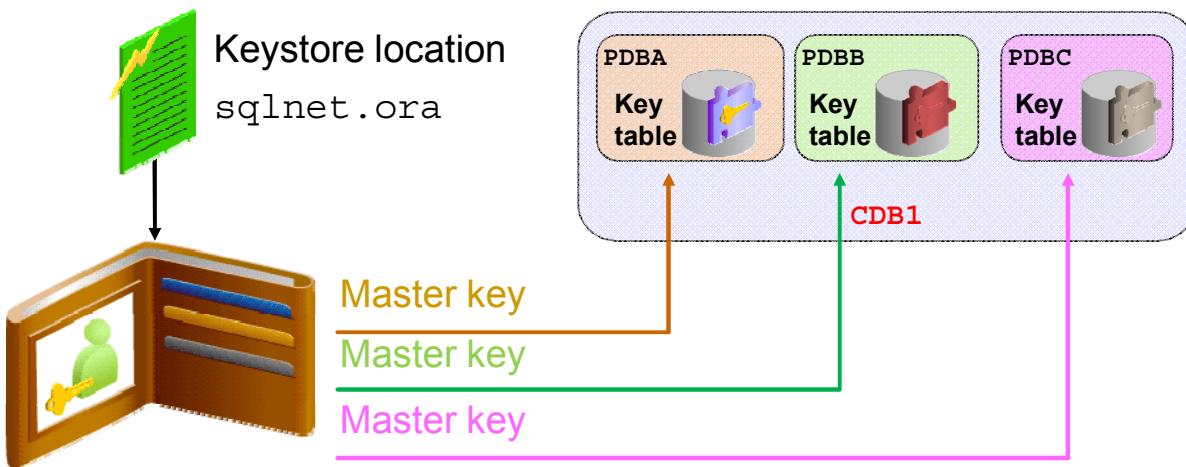
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

On a CDB, each PDB has its own master key used to encrypt data in the PDB. The master key must be transported from the source database keystore to the target database keystore when a PDB is moved from one host to another.

The Keystore and Master Keys

```
ENCRYPTION_WALLET_LOCATION =
(SOURCE =
(METHOD = FILE)
(METHOD_DATA =
(DIRECTORY =
/u01/app/oracle/product/12.1.0/dbhome_1/wallet)))
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For column encryption, each PDB maintains its own ENC\$ table, which is not a metadata-linked object. TDE creates a key for each table that uses encrypted columns and each encrypted tablespace. The table key is stored in the PDB data dictionary and the tablespace keys are stored in the tablespace data files in. Both tablespace and table keys are encrypted with the PDB master key. There is one master key for each PDB. The master keys are stored in a PKCS12 wallet or a PKCS11-based HSM, outside the CDB. For the CDB to use TDE, a keystore must exist.

Use the following procedure to create a wallet and a master key.

1. Create a directory to hold the wallet, which is accessible to the Oracle software owner.
2. Specify the location of the wallet file used to store the encryption master keys by adding an entry in the \$ORACLE_HOME/network/admin/sqlnet.ora file as shown in the following example:

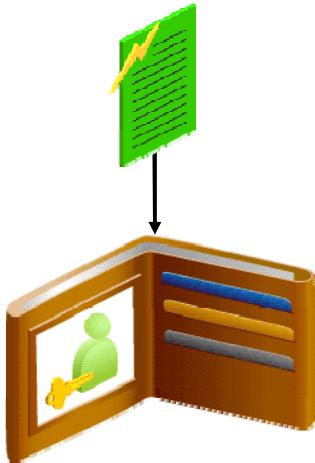
```
ENCRYPTION_WALLET_LOCATION=
(SOURCE=(METHOD=FILE) (METHOD_DATA=
(DIRECTORY=/u01/app/oracle/product/12.1.0/db_1/wallet)))
```

3. Connect to the root as a user with the SYSKM privilege.

Creating the Keystore

1. Create the keystore.

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE  
2   'u01/app/oracle/product/12.1.0/dbhome_1/wallet'  
3 IDENTIFIED BY software_keystore_password;
```



ORACLE

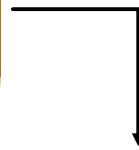
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

1. To create the keystore, use the following command:

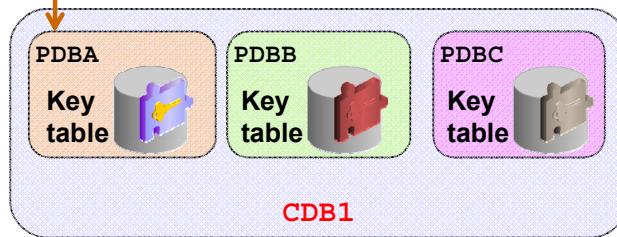
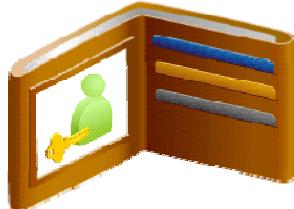
```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE  
2   'u01/app/oracle/product/12.1.0/dbhome_1/wallet'  
3 IDENTIFIED BY software_keystore_password;
```

Opening the Keystore

- Open the keystore for all PDBs or only for a specific PDB.



```
SQL> CONNECT sys@PDBA AS SYSKM
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE
2 OPEN IDENTIFIED BY password CONTAINER = CURRENT;
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- Open the keystore:

- In root:

```
SQL> CONNECT / AS SYSKM
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
2 IDENTIFIED BY software_keystore_password
3 CONTAINER=ALL;
```

Enter ALL to open the keystore in all PDBs in this CDB.

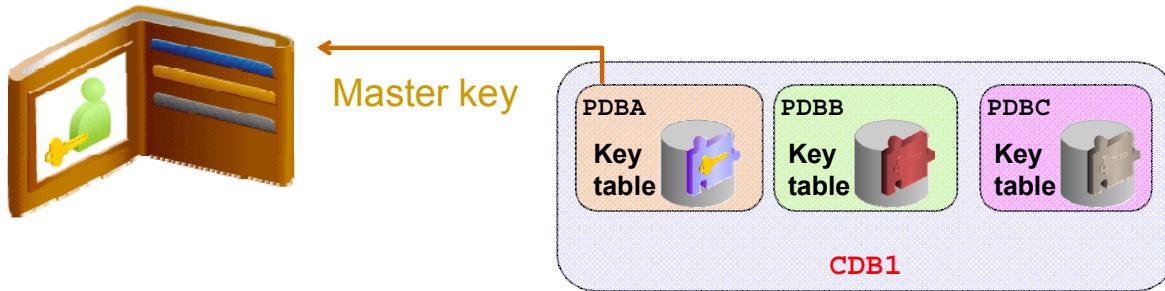
- In a PDB only:

```
SQL> CONNECT sys@pdb1 AS SYSKM
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
2 IDENTIFIED BY software_keystore_password
3 CONTAINER=CURRENT;
```

Setting Master Encryption Keys

- Set the master encryption key for a PDB.

```
SQL> CONNECT sys@PDBA AS SYSKM  
SQL> ADMINISTER KEY MANAGEMENT SET KEY  
2      IDENTIFIED BY password WITH BACKUP  
3      CONTAINER = CURRENT;
```



You can now encrypt data in tablespaces and tables.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- Set the master key:

- In root: Use `CONTAINER=ALL` to open the keystore in all PDBs in this CDB.
- In a PDB only: Use `CONTAINER=CURRENT` to open the keystore for the PDB.

Unplug and Plug PDB with Encrypted Data

1. Export the master encryption key of the PDB to unplug.

```
SQL> ADMINISTER KEY MANAGEMENT
  2  EXPORT ENCRYPTION KEYS WITH SECRET my_secret
  3  TO '/tmp/export.p12' IDENTIFIED BY password
  4  CONTAINER = 'HR_PDB1';
```

2. Unplug and plug the PDB.
3. Import the master encryption key into the new PDB.

```
SQL> ADMINISTER KEY MANAGEMENT
  2  IMPORT ENCRYPTION KEYS WITH SECRET my_secret
  3  FROM '/tmp/export.p12' CONTAINER = 'HR_PDB2'
  4  IDENTIFIED BY password WITH BACKUP;
```

4. Open the keystore.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE
  2  OPEN IDENTIFIED BY password CONTAINER = CURRENT;
```

ORACLE

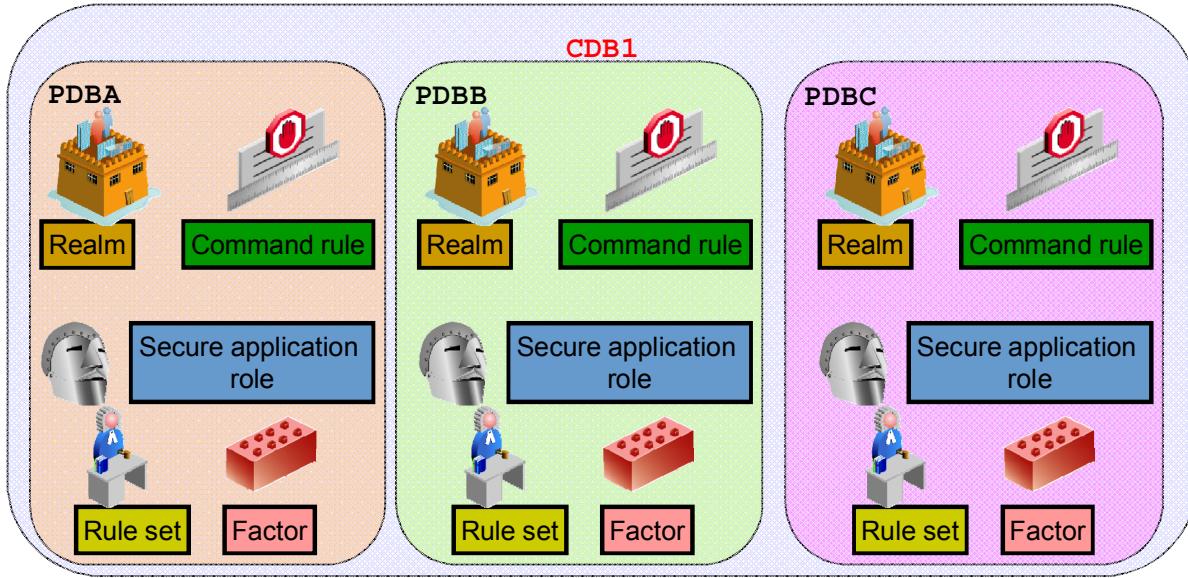
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The steps use the following sequence:

1. Export the Master Encryption Key from the PDB:
 - a. Log into root as a user who has been granted the SYSKM system privilege.
 - b. Query the STATUS column of the V\$ENCRYPTION_WALLET view to find if the keystore is open.
 - c. Export the Master Encryption Key from the PDB using the ADMINISTER KEY MANAGEMENT EXPORT command.
2. Unplug the PDB and plug the PDB.
3. Import the Master Encryption Key into the PDB using the ADMINISTER KEY MANAGEMENT IMPORT command.
4. Open the keystore using the ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN command.

Securing Data with Oracle Database Vault

In Oracle Database Vault, each PDB has its own Database Vault metadata. Database Vault constructs like realms are isolated within a PDB.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In Oracle Database Vault, each PDB has its own Database Vault metadata. Database Vault constructs like realms, command rules, secure application roles are isolated within a PDB.

Quiz

Which of the following statements are true?

- a. Using Oracle Database Vault to secure application data allows realms configuration at CDB level only.
- b. Encrypting data using TDE allows encryption in root only.
- c. Encryption uses a single master key for the whole CDB.
- d. None of the above.



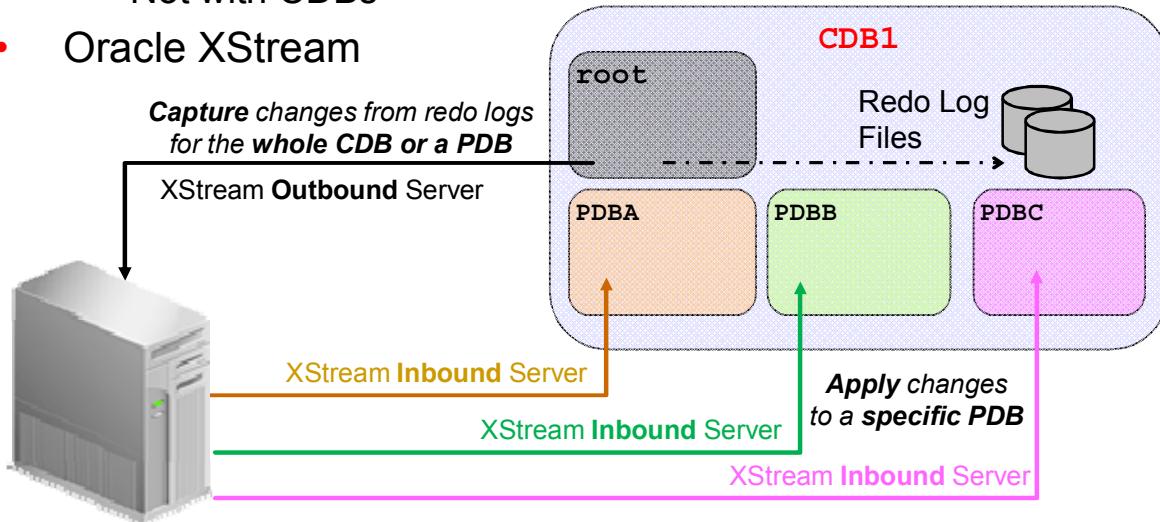
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: d

Using Xstreams with a CDB and PDB

Replicate data:

- Oracle Streams supported in Oracle Database 12c
 - With non-CDBs
 - Not with CDBs
- Oracle XStream



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

Oracle Streams continues to be supported in the Oracle Database 12c non-CDBs but is no longer enhanced for new database features or datatypes. Oracle Streams cannot be used with CDBs.

XStream is a programmatic interface to allow a client application access to the changes in the database, known as XStream Outbound Server. XStream Inbound Server allows a client application to feed changes into the database and takes advantage of the apply process available in the database. Oracle GoldenGate is the logical replication and XStream is licensed via the Oracle GoldenGate license. Capturing changes from the database must always be from the root due to the single unified redo log for the CDB. The XStream outbound can be configured to capture changes from a PDB or the entire CDB. Applying changes via Oracle GoldenGate is done per PDB. An XStream Inbound Server is configured to apply changes into a specific PDB and performs all of its work within the context of the PDB.

Creating a Standby of a CDB

Oracle Data Guard at CDB level

- **Create a standby of a CDB** as you create a standby of a primary non-CDB.
- **Create a PDB on a primary CDB.**
 - From an XML file: Copy the data files specified in the XML file to the standby database.
 - As a clone from another PDB: Copy the data files belonging to the source PDB to the standby database.
- **Remove or rename PDBs in a primary CDB.**
 - UNPLUG and DROP operations on the PDB: The PDB must be closed on the primary as well as all standby databases.
 - RENAME operation on the PDB: The PDB must be in open restricted mode on the primary, and closed on all standby databases.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Creating a Physical Standby of a CDB

The following are some of the behavioral differences to be aware of when you create and use a physical standby of a CDB:

- The database role is defined at the CDB level, not at the individual container level.
- If you execute a switchover or failover operation, the entire CDB undergoes the role change.
- Any DDL related to role changes must be executed in the root container because a role is associated with an entire CDB. Individual PDBs do not have their own roles.
- In a physical standby of a CDB, the syntax of SQL statements is generally the same as for non-CDBs. However, the effect of some statements, including the following, may be different:
 - ALTER DATABASE RECOVER MANAGED STANDBY functions only in the root container. It is not allowed in a PDB.
 - A role is associated with an entire CDB. Individual PDBs do not have their own roles. Therefore, the following role change DDL associated with physical standbys affects the entire CDB:

```
SQL> ALTER DATABASE SWITCHOVER TO target_db_name  
SQL> ALTER DATABASE ACTIVATE PHYSICAL STANDBY
```

- The `ALTER PLUGGABLE DATABASE [OPEN|CLOSE]` SQL statement is supported on the standby, provided you have already opened the root container.
- The `ALTER PLUGGABLE DATABASE RECOVER` statement is not supported on the standby. (Standby recovery is always at the CDB level.)
- To administer a CDB environment, you must have the `CDB_DBA` role.
- Oracle recommends that the standby database have its own keystore.

Creating a Logical Standby of a CDB

Refer to the *Oracle Data Guard Concepts and Administration 12c Release 1 (12.1) - Creating a Logical Standby Database chapter* to get a full description of the topic.

Creating PDBs on a Primary CDB

You must first copy your data files to the standby. Do one of the following, as appropriate:

- If you plan to create a PDB from an XML file, then copy the data files specified in the XML file to the standby database.
- If you plan to create a PDB as a clone from a different PDB, then copy the data files belonging to the source PDB to the standby database.

The path name of the data files on the standby database must be the same as the path name that will result when you create the PDB on the primary in the next step, unless the `DB_FILE_NAME_CONVERT` database initialization parameter has been configured on the standby. In that case, the path name of the data files on the standby database should be the path name on the primary with `DB_FILE_NAME_CONVERT` applied.

Removing PDBs from the Primary CDB

The following restrictions apply when you are removing or renaming a PDB at the primary:

- If the primary database is a CDB, then to perform `DDL UNPLUG` and `DROP` operations on a PDB, the PDB must first be closed on the primary as well as all standby databases.
- If the primary database is a CDB, then to perform a `DDL RENAME` operation on a PDB, the PDB must first be put in open restricted mode on the primary, and closed on all standby databases.
- If you do not close the PDB at the standby before removing it or renaming it at the primary database, then the standby stops the recovery process for all PDBs. You must close the dropped PDB at the standby and then restart recovery using the following SQL statement:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE;
```

Quiz

Oracle Streams can be used with CDBs.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

Scheduling Operations in a PDB

- A job defined in a PDB runs only if a PDB is open.
- User-created scheduler objects can be exported/imported into the PDB using Data Pump.
- Predefined scheduler objects are NOT exported.
 - Any changes made to these objects have to be made once again after the database has been imported into the PDB.

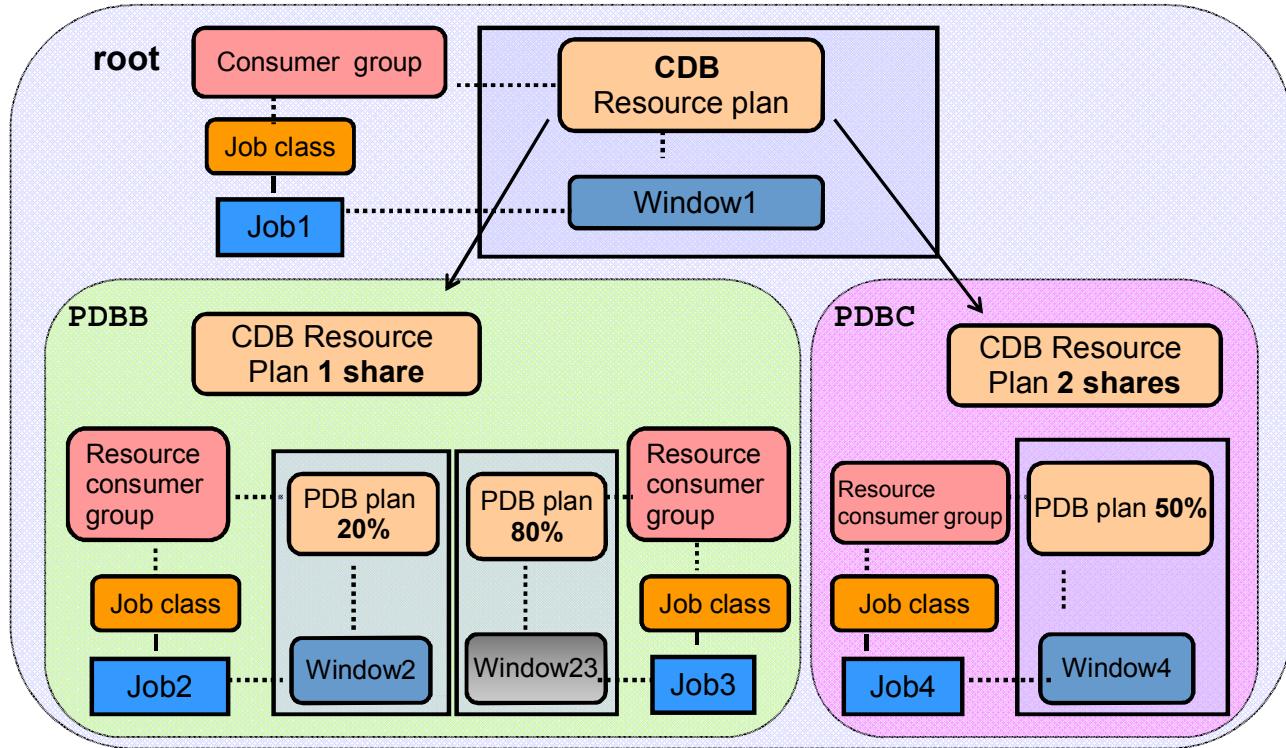


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Scheduler objects created by the user can be exported/imported into the PDB using Oracle Data Pump. Predefined scheduler objects are not exported and that means that any changes made to these objects by the user have to be made once again after the database has been imported into the PDB. A job defined in a PDB only runs if a PDB is open.

Jobs Coordinator and Resources

CDB1



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Scheduler Attribute Settings

Scheduler attribute settings is performed at the PDB level only. For example, if you set the `EMAIL_SENDER` attribute in the root, it applies to the jobs that run in the root, not the jobs running in a specific PDB. If you want to pick a new `EMAIL_SENDER` for a PDB, you must set the global attribute in that PDB.

Job Coordinator Process

The coordinator looks at the root database and all the PDBs to select jobs. The availability of resources to run a job depends on the consumer group of the job and the resource plan currently in effect in the container. The coordinator makes no attempt to be fair to every PDB. The only way to ensure that jobs from a PDB are not starved is to allocate enough resources to it.

Windows are open in the PDB and root database levels. In a non-CDB, only one window can be open at any given time. In a CDB, there are two levels of windows:

- At the PDB level, windows can be used to set resource plans that allocate resources among consumer groups belonging to that PDB.
- At the root level, windows can be used to allocate resources to various different PDBs.

Therefore, at any time, there can be a window open in the root and one in each PDB.

When a job slave executes a job, it switches to the PDB that the job belongs to.

Mining Statements of a PDB Using LogMiner

- Mining the CDB redo log files
- V\$LOGMNR_CONTENTS view
 - SRC_CON_NAME contains the pluggable database (PDB) name.
 - SRC_CON_ID contains the PDB ID.
 - SRC_CON_DBID contains the PDB identifier.
 - SRC_CON_GUID contains the GUID associated with the PDB.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The redo log files are common for the whole CDB. The information contained in redo log files is annotated with the identity of the PDB where a change occurs.

The V\$LOGMNR_CONTENTS view containing the log history information has new columns, the CON_ID, SRC_CON_ID, SRC_CON_NAME, SRC_CON_DBID, and SRC_CON_GUID.

The possible values include for CON_ID:

- **0**: This value is used for rows containing data that pertain to the whole CDB. (This value is also used for rows in non-CDBs).
- **1**: This value is used for rows containing data that pertain to only the root.
- **n**: It is the applicable container ID for the rows containing data.

The SRC_CON_NAME contains the PDB name. This information will only be available when mining with a current LogMiner dictionary.

The SRC_CON_ID contains the PDB ID (the ID column from the DBA_PDBS view). This information will be available with or without a current LogMiner dictionary.

The SRC_CON_DBID contains the PDB identifier (the DBID column from the DBA_PDBS view). This information will only be available when mining with a current LogMiner dictionary.

The SRC_CON_GUID contains the GUID associated with the PDB (the GUID column from the DBA_PDBS view). This information will only be available when mining with a current LogMiner dictionary.

Summary

In this lesson, you should have learned how to:

- Use Oracle Data Pump Export / Import with a CDB and PDB
- Use SQL*Loader to load data into a PDB
- Audit data of a CDB and PDB using Unified Auditing
- Secure data in a CDB and PDB using Transparent Data Encryption and Database Vault
- Describe the limits of data replication
- Describe XStreams usage with a PDB or the entire CDB
- Describe Data Guard with a CDB and PDB
- Schedule operations in a PDB using Oracle Scheduler
- Mine PDB statements using LogMiner



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 9 Overview: Miscellaneous

These practices cover the following topics:

- Auditing operations in a PDB using Unified Auditing
- Performing a full transportable export of a non-CDB and import into a PDB
- Exporting data from a PDB to import it into another PDB



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Consolidated Database Replay Procedures and Tables



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Consolidated Replay Steps

1. Create captures in separate directories.
2. Place all capture workloads in the same directory.
3. Process Capture workloads for target.
4. Set replay directory.
5. Create Replay schedule.
 - a. Add capture workloads
 - b. Specify replay order of capture workloads.
6. The replay CDB is restored to start of capture state.
7. Initialize Consolidated Replay.
8. Remap Connections.
9. Prepare Consolidated Replay.
10. Calibrate and Start Workload Replay Clients (WRC).
11. Start Consolidated Replay.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Shown here is a more detailed set of steps for using Consolidated Replay. In this procedure, it is assumed that you have already captured the workloads from different non-CDBs or PDBs of different CDBs. The slides in the rest of this lesson are ordered to follow this procedure.

The steps above are a general procedure for preparing to perform a consolidated replay.

In step 5, you must start with calling the `BEGIN_REPLAY_SCHEDULE` and end with calling `END_REPLAY_SCHEDULE`.

In step 6, the replay CDB is restored to the start of capture state. A variety of methods may be used to restore the replay CDB to the proper state including full transportable export non-CDB / import PDB, cloning PDB, Data Guard snapshot standby, flashback database, and full database recovery.

In step 10, one or more `wrc` processes are started and optionally calibrated.

In step 11, the `START_CONsolidated_REPLAY` procedure is executed. This procedure assumes sufficient number of `wrc` processes have been started.

Procedures for Steps 4 and 5

4. Set replay directory.
5. Create Replay schedule.
 - a. Add capture workloads.
 - b. Specify replay order of capture workloads.

```

DECLARE
    capture1      NUMBER;
    capture2      NUMBER;
BEGIN
    DBMS_WORKLOAD_REPLAY.SET_REPLAY_DIRECTORY('cap_root');
    DBMS_WORKLOAD_REPLAY.BEGIN_REPLAY_SCHEDULE('CONS_SCHEDULE');
    select DBMS_WORKLOAD_REPLAY.ADD_CAPTURE('CRM')
        into capture1 from dual;
    select DBMS_WORKLOAD_REPLAY.ADD_CAPTURE('SALES')
        into capture2 from dual;
    select DBMS_WORKLOAD_REPLAY.ADD_SCHEDULE_ORDERING(
        schedule_capture_id => capture2,
        waitfor_capture_id => capture1) from dual;
    DBMS_WORKLOAD_REPLAY.END_REPLAY_SCHEDULE;
END;
  
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To create a replay schedule, first start the process with a call to `SET_REPLAY_DIRECTORY`. This procedure sets a directory that contains multiple workload captures as the current replay directory. `cap_root` is a directory that contains multiple workload captures as the current replay directory.

Then continue the process with a call to `BEGIN_REPLAY_SCHEDULE`. Only one replay schedule can be in creation mode at a time. Calling this again before `END_REPLAY_SCHEDULE` will cause an error.

Add all the capture workloads by capture directory name. Each `ADD_CAPTURE` function call returns a `capture_id`. With the `ADD_CAPTURE` function, you can also specify the capture replay attributes.

The `capture_id` is used to specify the workload ordering with the `ADD_SCHEDULE_ORDERING` function.

The last procedure `END_REPLAY_SCHEDULE` wraps up the creation of the current schedule. The schedule is now saved and associated with the replay directory and can be used for a replay.

Procedure to Initialize the Replay

6. Restore Replay database.
7. Initialize the replay.

```
DBMS_WORKLOAD_REPLAY.INITIALIZE_CONSOLIDATED_REPLAY(
    REPLAY_NAME => 'CONS_REPLAY',
    SCHEDULE_NAME => 'CONS_SCHEDULE') ;
```

- This procedure loads the connection information in the capture subsets into the DBA_WORKLOAD_CONNECTION_MAP view.
- Use the connection identifiers (conn_id) from this table to remap connections.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Restore the replay database to the start state of the capture using the method of your choice. Initialize the replay with the `INITIALIZE_CONSOLIDATED_REPLAY` procedure. This procedure reads the information from the workloads in the replay directory and populates the `DBA_WORKLOAD_CONNECTION_MAP` table. Query this table to find the connections that need to be remapped.

```
SQL> DESC dba_workload_connection_map
      Name          Null?    Type
----- 
REPLAY_ID           NOT NULL NUMBER
CONN_ID            NOT NULL NUMBER
SCHEDULE_CAP_ID        NUMBER
CAPTURE_CONN        NOT NULL VARCHAR2(4000)
REPLAY_CONN          VARCHAR2(4000)
```

Remap Connections with PDBs

8. Remap connections.

- For each capture in a schedule, map the service names for the connection.
- Each capture can have different connection even if they are identical captures.

```
DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION(
    schedule_cap_id => 1
    connection_id     => 2,
    replay_connection => "oe/oe@pdb_oe.example.com") ;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Each capture workload in a schedule can remap each connection in the workload to a different connect for replay. This allows each capture to be mapped to a different pluggable database (PDB) in a multitenant container database (CDB).

This allows multiple copies of a capture to each map to a different PDB with each PDB matched to the start of the capture.

Procedure to Prepare the Replay

9. Prepare Consolidated Replay.
 - Specific type of synchronization

```
DBMS_WORKLOAD_REPLAY.PREPARE_CONSOLIDATED_REPLAY(  
    synchronization => 'OBJECT_ID');
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The next step is run the `PREPARE_CONSOLIDATED_REPLAY` procedure. The procedure `PREPARE_CONSOLIDATED_REPLAY` does similar work for consolidated replay as `PREPARE_REPLAY` does for the single replay. It sets up the replay options such as synchronization mode, connection time scale, think time scale, and then puts the DB state in the REPLAY mode. One or more external replay clients (WRC) can be started once the `PREPARE_CONSOLIDATED_REPLAY` procedure has been executed.

Note: `PREPARE_CONSOLIDATED_REPLAY` requires that preceding steps have been followed.

Modes of Synchronization

For single replays

- SCN synchronization is the default
- Recorded SCNs
 - Determine object dependencies
 - Replay call ordering

For Consolidated replay:

- Object ID synchronization:
 - It is recommended (SCN synchronization not supported).
 - It enables fine grain synchronization, providing more replay concurrency.
 - OBJECT_IDs are tracked by user call to minimize object collision on replay.
 - If collision happens, then replay orders calls. Otherwise, replay lets it run to get more concurrency.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The synchronization parameter controls whether the COMMIT order in the captured workload will be preserved during replay.

If this parameter is set to SCN, the COMMIT order in the captured workload will be preserved during replay and all replay actions will be executed only after all dependent COMMIT actions have completed. The SCN-based synchronization is not supported for consolidated replay.

Setting this parameter to OBJECT_ID allows for more concurrency during workload replays for COMMIT actions that do not reference the same database objects during workload capture. Independent transactions on the same database are likely to yield faster replay using Object ID synchronization.

With OBJECT_ID each workload captured on different database and being consolidated is independent of the other. Two objects with same name in different PDBs will be different “object_ids” and will not collide during replay.

You can disable this option by setting the parameter to OFF. This option can yield a faster replay, but the replay will likely yield significant replay divergence. However, this may be desirable if the workload consists primarily of independent transactions, and divergence during unsynchronized replay is acceptable.

Note that consolidated replay only supports non-sync mode (OFF) and the OBJECT_ID synchronization.

Procedure to Start Replay

10. Calibrate and start the workload replay clients.

```
$ wrc REPLAYDIR=/home/oracle/solutions/dbreplay \
  MODE=calibrate
```

```
$ wrc REPLAYDIR=/home/oracle/solutions/dbreplay \
  MODE=replay USERID=system PASSWORD=oracle
Workload Replay Client ...
Wait for the replay to start (21:47:01)
```

11. Execute the START_CONSOLIDATED_REPLAY procedure.

```
SQL> EXEC DBMS_WORKLOAD_REPLAY.START_CONSOLIDATED_REPLAY
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The replay database is ready to accept workload replay client (WRC) connections once the PREPARE_CONSOLIDATED_REPLAY procedure is executed.

As shown, `wrc` is an OS process and can be run from a client machine in the network. These machines must have the replay client software and a copy of the replay directory that contains the processed capture workloads.

Calibrate the workload clients with the calibrate mode. Because the client is a multithreaded process, it can open multiple sessions to the replay database. So one `wrc` process can simulate many user processes on the capture system. The output of the calibrate mode is an estimate of the number of `wrc` processes that will be needed.

The next step is to start the number of `wrc` clients recommended by the calibrate mode. These processes start, contact the replay database, and wait for the `START_CONSOLIDATED_REPLAY` command to be issued. After a workload replay is started, new replay clients will not be able to connect to the database. Only replay clients that were started before the `START_CONSOLIDATED_REPLAY` procedure is executed will be used to replay the multiple-capture capture.

Tables

Tables:

- DBA_WORKLOAD_REPLAY_SCHEDULES
- DBA_WORKLOAD_SCHEDULE_ORDERING



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The tables allow the workload schedules to be saved and reused. These tables exist in the root container of a CDB and in the PDBs.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

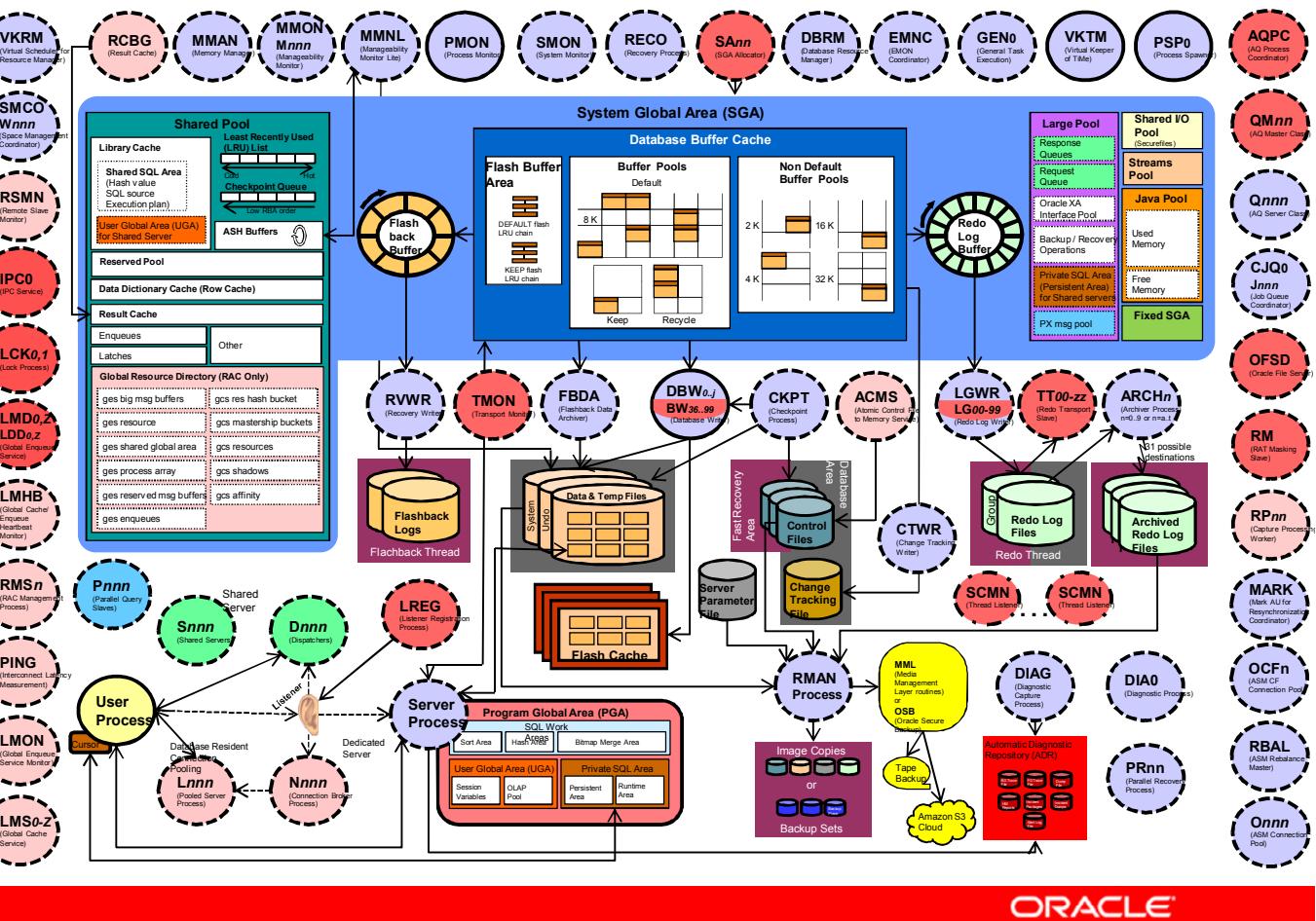
New Processes, Views, Parameters, Packages and Privileges

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Note: For a complete list of views, parameters, packages, and privileges related to multitenant container databases and pluggable databases, refer to the following guides in the Oracle documentation:

- *Oracle Database Reference 12c Release 1 (12.1)*
- *Oracle Database PL/SQL Packages and Types Reference 12c Release 1 (12.1)*



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The graphic shows the background processes, SGA and PGA components of the Oracle Database 12c instance.

1. Circle elements represent Oracle processes. If they are surrounded by dotted lines (equal dotted elements), it means they can run either as threads or OS processes. If they are surrounded by a solid line, it means they can only run as OS processes. The SCMN case is an exception. When using the Multi-Process Multi-Threaded architecture, each OS process, running more than one Oracle Process, also runs a special thread called SCMN that is basically an internal listener thread.
2. Darker circle elements are new elements for DB 12c.
3. The two main different color nuances for circle elements are to make the distinction between RAC and non-RAC processes.
4. Files are represented by cylinders.
5. Storage location for those files are divided into three main areas: Fast Recover Area, Database Area, and Automatic Diagnostic Repository. Areas are designated by background rectangles using three different colors. Exception is the server parameter file. If you find a file type part of two areas it means that some corresponding files can be in both areas.
6. Inside one circle element you may see two names. This is to indicate the second (smaller letters) is a slave of the first.

New Views

View

- CDB_xxx: All of the objects in the CDB across all PDBs
- DBA_xxx: All of the objects in a container or PDB
- CDB_pdbs: All PDBS within CDB
- CDB_tablespaces: All tablespaces within CDB
- CDB_users: All users within CDB (common and local)
- V\$PDBS: Displays information about PDBs associated with the current instance
- V\$CONTAINERS : Displays information about PDBs and the root associated with the current instance
- PDB_PLUG_IN_VIOLATIONS : Displays information about PDB violations after compatibility check with CDB
- RC_PDBS: Recovery catalog view about PDB backups



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

New Parameters and Packages

Parameter

- `ENABLE_PLUGGABLE_DATABASE=true`
Required to create a CDB at CDB instance startup
- `PDB_FILE_NAME_CONVERT`
Maps names of existing files to new file names when processing a `CREATE PLUGGABLE DATABASE` statement
- `CDB_COMPATIBLE=true`
Enables you to get behavior similar to a non-CDB

Package

- `DBMS_PDB.DESCRIBE`
- `DBMS_PDB.CHECK_PLUG_COMPATIBILITY`



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.