Making Everything Easier!™

Oracle® Multitenant FOR DUMMIES®

A Wiley Brand

Learn to:

- Consolidate using Oracle Multitenant
- Upgrade and provision databases faster
- Manage many databases as one

Brought to you by

ORACLE®

Lawrence C. Miller, CISSP

Oracle® Multitenant For Dummies®

A Wiley Brand

by Lawrence C. Miller, CISSP

For Dummies®

A Wiley Brand

Oracle® Multitenant For Dummies®

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN 978-1-118-72870-3 (pbk); ISBN 978-1-118-73150-5 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

We're proud of this book and of the people who worked on it. For details on how to create a custom *For Dummies* book for your business or organization, contact info@dummies.biz or visit www.wiley.com/go/custompub. For details on licensing the *For Dummies* brand for products or services, contact BrandedRights&Licenses@Wiley.com. Some of the people who helped bring this book to market include the following:

Acquisitions, Editorial, and Vertical Websites

Senior Project Editor: Zoë Wykes

Editorial Manager: Rev Mengle

Acquisitions Editor: Katie Mohr

Senior Business Development Representative: Karen L. Hattan

Custom Publishing Project Specialist: Michael Sullivan

Composition Services

Senior Project Coordinator: Kristie Rees

Layout and Graphics: Jennifer Creasey

Proofreader: Dwight Ramsey

Special Help from Oracle: Bryn Llewellyn, Patrick Wheeler, Markus Weber, William Hardie, Kimberly Billings, Michael Timpanaro-Perrotta, Patti Lawrence

Publishing and Editorial for Technology Dummies

Richard Swadley, Vice President and Executive Group Publisher

Andy Cummings, Vice President and Publisher

Mary Bednarek, Executive Director, Acquisitions

Mary C. Corder, Editorial Director

Publishing and Editorial for Consumer Dummies

Kathleen Nebenhaus, Vice President and Executive Publisher

Composition Services

Debbie Stailey, Director of Composition Services

Business Development

Lisa Coleman, Director, New Market and Brand Development

Contents at a Glance

Introduction

Oracle Multitenant — an Oracle Database 12*c* option — introduces a new architecture that enables organizations to easily consolidate multiple Oracle databases. In this new architecture, a multitenant container database can hold many pluggable databases. An administrator deals with the multitenant container database, but application code connects to one pluggable database. Existing Oracle databases can be easily adopted as pluggable databases — with no changes to any associated applications required.

This new multitenant architecture delivers all the benefits of managing many databases as one, while retaining the isolation and resource prioritization of separate databases. In addition, Oracle Multitenant enables rapid provisioning and upgrades and is fully compatible with other Oracle Database options, including Oracle Real Application Clusters and Active Data Guard.

Oracle Multitenant For Dummies explains the *who*, *what*, *when*, *where*, *why,* and *how* of multitenant container and pluggable databases, but not in that order! You find out what pluggable databases are, how they work, who benefits from them, where to deploy them, and why they are an ideal solution for organizations that need to reduce their IT and database management costs, while addressing their database challenges.

About This Book

This book contains volumes of information that rival the U.S. Congressional Record or the complete *Encyclopedia Britannica,* conveniently distilled into five short chapters chock-full of just the information you need! Here's a brief look at what awaits you in the pages ahead.

Chapter 1: Why Oracle Multitenant? I start by explaining just *what* exactly a multitenant container and pluggable databases are, *why* multitenant container and pluggable databases are such an innovative solution to many common database challenges, and *who* in your organization can benefit from implementing Oracle Multitenant.

Chapter 2: Database Consolidation with Oracle Multitenant. Here, you take a look under the hood and learn *how* Oracle Multitenant helps you consolidate your databases.

Chapter 3: Managing Oracle Multitenant. In this chapter, you get an overview of some of the common operations on multitenant container databases and pluggable databases, and *how* to perform these operations.

Chapter 4: Deploying Oracle Multitenant. Here, you learn about different deployment options and scenarios — *when* and *where* to use various configurations of Oracle Multitenant.

Chapter 5: Ten Benefits of Oracle Multitenant. Finally, in that classic *For Dummies* style, I tell you about ten great benefits of Oracle Multitenant for your organization.

Note: In this book, I use “Oracle Multitenant” to refer to the new multitenant database architecture as a whole. When referring to a specific component such as a multitenant container database or a pluggable database, I refer to it as a CDB or PDB, respectively.

Icons Used in This Book

Throughout this book, you occasionally see icons that call attention to important information that is particularly worth noting. You won’t find any winking smiley faces or other cute little emoticons, but you’ll definitely want to take note! Here’s what to expect.

REMEMBER This icon points out information that may well be worth committing to your nonvolatile memory, your gray matter, or your noggin — along with anniversaries and birthdays!

TECHNICAL STUFF If you’re an insufferable insomniac or vying to be the life of a World of Warcraft party, take note. This icon explains the jargon beneath the jargon and is the stuff legends — well, at least nerds — are made of.

TIP Thank you for reading, hope you enjoy the book, please take care of your writers! Seriously, this icon points out helpful suggestions and useful nuggets of information.

WARNING! Proceed at your own risk . . . *well, okay* — it’s actually nothing *that* hazardous. These helpful alerts offer practical advice to help you avoid making potentially costly mistakes.

Where to Go from Here

With our apologies to Lewis Carroll, Alice, and the Cheshire Cat:

"Would you tell me, please, which way I ought to go from here?"

"That depends a good deal on where you want to get to," said the Cat — er, the Dummies Man.

"I don't much care where . . . ," said Alice.

"Then it doesn't matter which way you go!"

That's certainly true of *Oracle Multitenant For Dummies* which, like *Alice in Wonderland*, is destined to become a timeless classic!

If you don't know where you're going, any chapter will get you there — but Chapter 1 might be a good place to start! However, if you see a particular topic that piques your interest, feel free to jump ahead to that chapter. Read this book in any order that suits you (though I don't recommend upside down or backwards).

I promise that you won't get lost falling down the rabbit hole!

Chapter 1

Why Oracle Multitenant?

In This Chapter

- Recognizing some common database issues
- Describing Oracle Multitenant
- Moving your databases to the cloud
- Looking at the benefits of Oracle Multitenant

Traditional database deployment models have led to database sprawl in many enterprise datacenters, contributing to rising costs and management complexity.

In this chapter, you find out what Oracle Multitenant is and how it can help you address many database challenges. You also find out how this new multitenant architecture can facilitate and accelerate your organization's journey to the database cloud.

Challenges Addressed by Oracle Multitenant

Enterprise IT organizations have long been deploying stand-alone databases and their associated applications onto dedicated server infrastructures to support different departments or lines of business (LOBs), allowing IT organizations to sidestep complex issues such as:

- Interoperability between multiple applications that coexist on the same server
- Resource planning, prioritization, allocation, and contention
- Segmentation for data security and regulatory compliance
- Lack of vendor support for applications on shared systems — the old "that isn't supported" excuse

As a result, it's fairly common to find enterprise datacenters with hundreds — even thousands — of databases scattered over almost as many servers. This database sprawl has led to a number of challenging and costly issues in itself, including

- **Underutilization of technology infrastructure.** Deploying individual, purpose-built servers to run a single business application and its associated database tier typically results in severe underutilization of powerful servers, massive storage systems, and high-speed networks.
- **Inefficient use of administrative resources.** Database administrators must devote far too much time to the day-to-day tasks necessary to

maintain these rapidly growing, widely dispersed, and disparate databases. These tasks include

- **Provisioning.** Creating new databases, moving existing databases between different servers, and cloning databases for various purposes — such as development, testing, and troubleshooting — has become a major task for database administrators.
- **Patching and upgrading.** Applying patches, managing multiple database versions, and upgrading existing databases across hundreds or thousands of databases is time-consuming.
- **Security and compliance.** Attempting to manage security and regulatory compliance at an individual database level rather than implementing a comprehensive security strategy can create complexity and increase risk.
- **Management.** Siloed database deployments prevent IT organizations from getting a complete view of the database environment, which impedes efforts to proactively monitor performance and inhibits their ability to respond quickly to emerging business needs.

What Is Oracle Multitenant?

Oracle Database 12*c* supports a new multitenant architecture that lets you have many "sub databases" inside a single "super database." The super database is the *multitenant container database* or CDB, and the sub databases are the *pluggable databases* or PDBs. In other words, this new architecture, referred to as the *multitenant architecture*, lets you have many PDBs inside a single CDB.

The term *non-CDB* is used to describe Oracle databases through Oracle Database 11*g*, as well as Oracle Database 12*c* databases that do not take advantage of the new multitenant architecture. However, as you find out in this book, there's absolutely no reason you shouldn't implement the multitenant architecture because Oracle Multitenant offers tangible benefits for the business, as well as for IT!

From the point of view of a client connecting to the Oracle Database, the PDB is the database. The application, and any application code, that ran against a non-CDB before will run — *with no change* — in a PDB and will produce the very same result. Ideally, one PDB will be used to hold a *single* application backend and all of the data needed to support that particular application. Unlike schema-based consolidation (described in Chapter 2) used within a non-CDB, the PDB provides a clear, declarative means to contain an application backend so that the Oracle system knows explicitly which parts of the data and the system belong to which application backend.

From the point of view of the operating system, the CDB is the database. Oracle Multitenant is fully compatible with Oracle Real Application Clusters (RAC). It's also possible for a PDB to be open on only a subset of the nodes of a RAC, providing greater flexibility and scalability in a RAC configuration.

From the client's point of view, the PDB is the database. From the server's point of view, the CDB is the database.

Oracle Database 12*c* supports both multitenant and non-CDB architectures. Therefore, you can upgrade a pre-12.1 database to a 12.1 non-CDB and continue to operate it as such. Or, you can upgrade a pre-12.1 database to a 12.1 non-CDB and adopt it as a PDB in a CDB — which is PDC (pretty darn cool!).

A PDB is fully compatible with a non-CDB, which is known as the *PDB/non-CDB compatibility guarantee*.

Journey to the Database Cloud

Increasingly, enterprises are pursuing database consolidation initiatives and/or cloud computing strategies in order to lower costs and improve agility.

Oracle Multitenant can simplify and accelerate your organization's journey to the database cloud by providing an excellent platform for database standardization and consolidation.

In Jules Verne's *Journey to the Center of the Earth,* the story's adventurers descend into an extinct volcano and encounter prehistoric animals and various other natural hazards on their excursion. Although the journey to the cloud is unlikely to entail volcanos and dinosaurs, it can be fraught with disaster and no less risky for many organizations.

Most organizations begin the journey with complex, disparate IT silos. Moving to the cloud requires identifying discrete steps and defining the specific value that each step delivers along the way. When properly implemented, each step is an enabler for subsequent steps. To help guide your journey, Oracle has developed a "Journey to the Database Cloud" maturity model (see Figure 1-1) based on real-world experience.

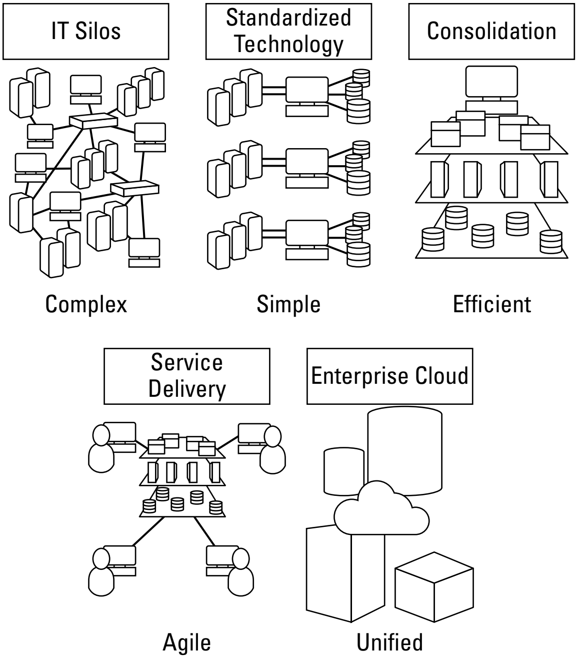

Figure 1-1: Oracle's "Journey to the Database Cloud" maturity model.

Oracle's "Journey to the Database Cloud" consists of four phases:

- **Standardized Technology.** Simplify to reduce operational costs and business risk.

- **Consolidation.** Efficiently reduce datacenter footprint — both hardware and software.
- **Service Delivery.** Automate to increase business agility.
- **Enterprise Cloud.** Unify services for location independence and unlimited capacity.

In the following sections, you learn how Oracle Multitenant supports the standardization and consolidation steps in the "Journey to the Database Cloud." Details of the Service Delivery and Enterprise Cloud phases are beyond the scope of this book.

Standardizing for simplicity

The high cost of running many databases is exacerbated by variations between these databases. The most obvious variables are the type of hardware, the operating system, and the choice of Oracle Database edition, version, and options. Other variables include patch levels, security administration, and management processes.

PDBs are standardized and self-contained databases that address these database challenges. CDBs promote standardization at the database level and thus enable standardization at all technology layers below it, such as operating system and server hardware platform.

As a best practice, standardization should be pursued aggressively and implemented at every level of your technology infrastructure. This promotes simplification with easier automation and management of modular components.

Consolidating for efficiency

In many IT environments, servers are severely underutilized. Standardization allows organizations to achieve greater efficiency in their IT operations through consolidation, which enables higher utilization of technology resources. This can reduce future capital expenditures (CapEx) since fewer resources are required to deliver the same business functionality and guarantee service-level agreements (SLAs).

Operating expenses (OpEx) can also be lowered since fewer divergent technology components need to be procured, deployed, monitored, and managed, and many of these lifecycle management operations can be automated.

In the consolidation phase of the journey to the database cloud, the goal is to reduce CapEx and OpEx while maintaining your business SLAs.

Benefits of Oracle Multitenant

The Oracle Multitenant deployment model promotes standardization of servers, storage, and operating systems, and consolidation onto a standardized database version deployed on shared infrastructure. This model yields the highest savings in CapEx and OpEx for your private database cloud deployment, because the number of divergent technology components and software versions is significantly reduced.

Using Oracle Multitenant, consolidating multiple databases as PDBs in a single CDB on shared public or private cloud infrastructure delivers the following key benefits.

- **Cost-effective consolidation.** Consolidating multiple PDBs into a CDB reduces operating system and memory overheads and hardware server footprint (lowers CapEx). This improves the sharing of resources such as background processes, shared memory, and Oracle metadata, and thus increases resource utilization (yields higher return on investment, or ROI).
- **Manage many as one.** A CDB containing many PDBs can be managed and monitored as a single database using the powerful and intuitive Oracle Enterprise Manager and other database management tools. For example:
  - Starting a CDB starts all PDBs contained in that CDB.
  - Upgrading or patching the dictionary of a CDB results in propagating the resulting change to all PDBs contained in that CDB.
  - Backing up and replicating the CDB as a whole assures the backup and replication of all PDBs in that CDB at once.
- **Increased agility.** PDBs can be plugged or unplugged extremely rapidly for patching, upgrade, or migration purposes, thereby tremendously decreasing OpEx.
- **Rapid provisioning.** PDBs can be provisioned extremely rapidly by cloning existing pluggable databases. Provisioning can be done even faster on file systems that support the copy-on-write feature, such as Oracle ACFS (Automatic Storage Management Cluster File System) or Oracle ZFS.

- **Ease of adoption.** If your application has been certified against Oracle Database 12*c*, it will run unchanged when its database is deployed as a PDB in a CDB.
- **Security in the cloud.** Oracle Multitenant is implemented in a fully isolated architecture that ensures security privileges and safeguards are maintained without compromise in extremely high-density consolidation and multitenancy cloud environments.
- **Shared resource management.** In a consolidated environment, resource contention is an issue that must be addressed. Figure 1-2 shows an example of consolidating disparate systems (DW, CRM, and ERP) as different PDBs, contained within the same CDB. Oracle Resource Manager provides resource isolation and prioritization at a per-PDB level and allows you to assign different priorities to the applications supported by these various PDBs.

TIP

Oracle Resource Manager has been enhanced in Oracle Database 12*c* to support the new multitenant architecture. You can define resource plans that operate between PDBs within the same CDB to meet application performance and availability SLAs. Policies can be defined in terms of *shares* and *caps* (see Chapter 4) to dictate how resources are used at a per-PDB level.

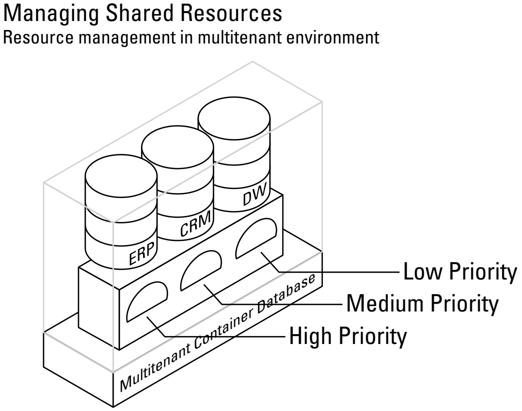
Managing Shared Resources
Resource management in multitenant environment
ERP
CRM
DW
Low Priority
Medium Priority
High Priority
Multitenant Container Database

Figure 1-2: Consolidating with Oracle Multitenant.

Chapter 2

Database Consolidation with Oracle Multitenant

In This Chapter

- Understanding different consolidation models
- Recognizing schema-based consolidation issues
- Introducing consolidation in the multitenant architecture

Consolidating databases onto shared resources enables IT departments to improve service levels — as measured in terms of database performance, availability, and data security — and reduce capital and operating costs.

Prior to Oracle Database 12*c*, schema-based consolidation was the best way to achieve the highest consolidation density in stand-alone Oracle Databases. However, the schema-based consolidation model has drawbacks.

In this chapter, you learn about different database consolidation models and how Oracle Multitenant delivers all the benefits of the schema-based model — with none of the shortcomings.

Differentiating between Consolidation Models

The established starting point for consolidation of Oracle Databases is standardization of the server and storage infrastructure, the operating system, and a limited set of Oracle Database versions.

The next logical step is to consolidate Oracle Databases. Organizations have traditionally consolidated their Oracle Databases using one, or a combination, of the following three strategies (see Figure 2-1):

- **Multiple virtual machines (VMs) on a single physical server.** Each VM runs a separate Oracle Database. This strategy is simple to implement but provides the lowest consolidation density of the three database consolidation strategies. Separate copies of the operating system and database server software must be maintained in each VM, which therefore might help reduce physical sprawl but not virtual sprawl.
- **Dedicated database model.** Several separate databases are installed onto the same physical server. Each application backend is installed in its own dedicated database. Each database has its own Oracle Home, system tables, memory, and processes. Consolidation density in the dedicated database model is limited by the server hardware's ability to operate the individual installed databases and application backends within their respective SLAs (service-level agreements).
- **Within-database consolidation model.** In this model, each application backend is contained in

its own schema (pre-12*c*) or pluggable database (in 12*c* and beyond). Multiple application backends are supported within a single host database. In Oracle Database 12*c*, the host database is referred to as the multitenant container database, or CDB.

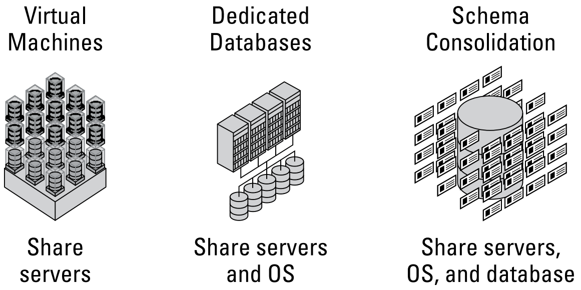

Figure 2-1: Three different database consolidation models.

Schema-based Consolidation

Prior to Oracle Database version 12*c,* the within-database consolidation model was implemented using *schema-based consolidation*. In schema-based consolidation, the consolidated database essentially consists of one or more application backends (or schemas) running across one or more servers. Multiple distinct application backends can be installed into the same database by running the installation program for each application in turn.

Schema-based consolidation supports a significantly higher consolidation density — measured in application backends per platform — than is possible when each application backend is in its own database.

However, schema-based consolidation is inherently difficult and has several disadvantages, including

- **Name collision might prevent schema-based consolidation.** Some application backends are implemented entirely within a single schema and can usually be installed in the same database as other application backends. Other application backends are implemented in several schemas, which implies cross-schema references and therefore a dependency on the specific schema names that the design has adopted. It is possible that the same schema names will be used in different application backends.

  The name of a schema must be unique within the database as a whole, so application backends with colliding schema names cannot be installed in the same database. The names of other phenomena, such as roles, directories, editions, public synonyms, public database links, and tablespaces, must also be unique within the database as a whole.

- **Security can be challenging.** Potential security issues to be considered and mitigated when designing a schema-based consolidation model include

  - The person who installs a particular application backend needs to be authenticated as a database user with powerful privileges like Create User, Alter User, and Drop User. This person could then make changes to any other application backend in the same database and could view or change confidential data.

  - A poorly designed application backend whose implementation uses more than one schema might rely on system privileges like Select Any Table, Delete Any Table, and so on. In such a case, a SQL injection vulnerability, for example, might allow a run-time user of one application backend to read or change another application backend's data.

- **Per application backend point-in-time recovery is difficult.** Point-in-time recovery is usually needed when an error in application code, introduced by an application patch, causes data corruption. In such a case, the single application backend — not the whole database — needs to be recovered to the time just before the patch was applied. Tablespace point-in-time recovery might be sufficient, but this can be difficult when the corrupt data spans several tablespaces or if an administrator drops one or more mutually referencing tables by mistake.
- **Resource management between application backends is difficult.** Resource management relies on one or more specific services being used to start the sessions that access a particular application backend, and no service can be used for more than one application backend.
- **Cloning a single application backend is difficult.** Data Pump is the only option.

Data Pump enables high-speed movement of data and metadata from one Oracle Database to another. It's ideal for large databases and data warehousing environments, where high-performance data movement offers significant time savings to database administrators.

Consolidation in the Multitenant Architecture

Oracle Database 12*c* supports a new model for within-database consolidation, known as Oracle Multitenant.

At the heart of this new multitenant architecture is the horizontal partitioning of the Oracle data dictionary. In essence, the root database holds the definition of the Oracle-supplied data dictionary along with all the necessary system details of all the pluggable databases. Each pluggable database holds all the necessary information of the application and user data associated with the pluggable database. This separation allows you to achieve the same high density benefits of schema-based consolidation, while addressing the inherent problems of schema-based consolidation.

It is this separation that gives you the "pluggability" of Oracle Multitenant (see Figure 2-2).

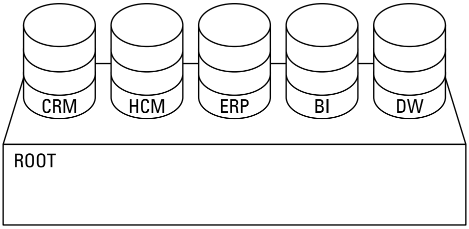

Figure 2-2: High-level view of consolidating pluggable databases into a multitenant container database.

In contrast to those disadvantages of schema-based consolidation (explained in the preceding section), consolidation with Oracle Multitenant provides the following advantages:

- **Global namespaces at the PDB-level.** A PDB defines a global namespace and contains the effects of privileges, just as a non-CDB does.
- **Enhanced security.** With Oracle Multitenant, you can migrate each individual schema into a PDB of its own. Each of these PDBs acts identically from a user's point of view to a non-CDB. This is known as *tenant isolation* — the different PDBs in a CDB are unaware of other PDBs in the same CDB.
- **Point-in-time recovery is supported at the PDB-level.** Oracle Active Data Guard and scheduled RMAN backups are conducted at the CDB-level.
- **The CDB architecture extends Resource Manager.** Database Resource Manager (DBRM) plans that are defined for the CDB can be applied to all PDBs contained in that CDB.
- **A PDB can be unplugged from one CDB and plugged into another CDB running an upgraded Oracle Database version.** The PDB will be upgraded automatically to the new version when it is plugged in to the new CDB.
- **Creating a new PDB as a clone of an existing PDB is fast and easy.** When the underlying file system supports thin provisioning, many terabytes can be cloned almost instantaneously.

Chapter 3

Managing Oracle Multitenant

In This Chapter

- Moving and upgrading PDBs
- Copying production PDBs
- Creating a brand-new PDB
- Cleaning up PDBs

Typical operations on PDBs (pluggable databases) include unplug and plug, create, clone, and drop. In this chapter, you see how these operations, implemented in the context of the multitenant architecture, bring new paradigms for managing operations, such as provisioning and patching the Oracle Database.

Unplug and Plug a PDB

The PDB's data dictionary includes information such as definitions of its application schema, the database users, and their privileges. To unplug a PDB, you must first close it. Then, you can unplug the PDB with a single SQL statement, using the following syntax:

```
alter pluggable database [PDB name]
unplug into '[path to external manifest]'
```

In the preceding example and the examples that follow, don't include the brackets when entering the commands.

Correspondingly, you plug in with a single SQL statement, using the following syntax:

```
create pluggable database [PDB name]
using '[path to external manifest]'
```

You can use unplug and plug operations for a number of management tasks, such as:

- Moving a PDB for a business-critical application when its performance no longer meets its defined SLA (service-level agreement) or to upgrade server hardware
- Patching or upgrading an Oracle Database

Moving a PDB

By design, a PDB holds exactly one application backend. To move an application backend — for example, from one CDB to another on the same or on a different server, you must close and then unplug the associated PDB. On unplug, the PDB's metadata is written to an external manifest that accompanies the PDB's datafiles during the move. The PDB is then plugged in to the new CDB.

Patching and upgrading

The multitenant architecture provides two fast-and-easy methods for patching and upgrading pluggable databases.

You can perform an in-place upgrade by simply applying a patch or upgrade at the multitenant container database (CDB) level; all PDBs in the CDB are then automatically patched or upgraded. This is an example of what Oracle refers to as the "manage many as one" benefit of Oracle Multitenant: all PDBs benefit from the upgrade of their common CDB.

The second method of patching and upgrading PDBs takes advantage of the unplug/plug capability of PDBs to provide a more granular level of control over the individual PDBs in a CDB. In this method, you simply create a second CDB at the newer Oracle version or patch level. You then unplug the PDBs from the older version CDB and plug them into the newer version CDB. As each of the PDBs is plugged into the new CDB, it is automatically patched or upgraded. This method gives you the flexibility to upgrade individual PDBs using a phased approach.

When an application backend is hosted in a PDB, the outage time for the application, caused by patching or upgrading the Oracle version, will always be less than the outage time required for patching or upgrading a non-CDB. The outage time for a patch or upgrade using the unplug/plug approach on a PDB can sometimes be measured in seconds.

Clone a PDB

The multitenant architecture supports creating an identical copy of a PDB. This copy, called a *clone,* may be created either in the same CDB as the source PDB (a local clone) or in a different CDB (a remote clone).

If the underlying file system supports cloning, you can create a within-CDB clone by using snapshot copy in the SQL statement.

The syntax of the SQL statement for a local clone is

```
create pluggable database [PDB clone name]
from [PDB original name]
```

The syntax of the SQL statement for a remote clone is

```
create pluggable database [PDB clone name]
from [PDB original name]@[path to root of
remote CDB]
```

The create PDB, plug in PDB, and clone PDB operations are all variants of the create pluggable database SQL statement.

Create a PDB

The create PDB operation is implemented as a special case of the clone PDB operation. The syntax for the create PDB SQL statement is

```
create pluggable database [PDB name]
path_prefix = '[path for created directory
objects]'
admin user [admin username] identified by
[admin password] roles = ([admin roles to be
granted])
```

The preceding SQL statement creates a new PDB, names it, specifies the path to the subtree in the file system for any directory objects that are created, and creates a local admin user that is granted a list of roles.

Drop a PDB

The drop PDB operation has two variations: the variation you use will depend on your particular scenario.

In the first scenario, the PDB is simply no longer needed. This would be common in a non-production environment where, for example, a PDB that represents a starting point for a specific development task is repeatedly cloned and dropped. This type of drop is referred to as a *destructive drop*.

The syntax for the destructive drop SQL statement is

```
drop pluggable database [PDB name] including
datafiles
```

WARNING! As its name implies, a destructive drop removes a PDB and all of its datafiles, leaving no trace. This command should be used with extreme caution.

The second scenario applies after unplugging a PDB, for example, when you are about to move it from one CDB to another. In this case, following the unplug operation, you want to remove the metadata definitions of the PDB from the source CDB without dropping the datafiles — which contain the application backend you're going to move to the other CDB! What is needed here is known as a *non-destructive* drop.

The syntax for the non-destructive drop SQL statement is

```
drop pluggable database [PDB name] keep
datafiles
```

Chapter 4

Deploying Oracle Multitenant

In This Chapter

- Differentiating CDB and PDB choices
- Using different adoption approaches
- Going it alone with a single PDB
- Dealing with resource contention

The new multitenant architecture in Oracle Database 12*c* has several deployment models. In this chapter, you learn what configuration decisions need to be made for the multitenant container database (CDB) as a whole, which options can be set for individual pluggable databases (PDBs), how to adopt a non-CDB as a PDB, and how Oracle Multitenant handles resource contention between application backends.

Choosing CDB and PDB Options

With consolidation, you can manage many databases as a single database. Rather than managing separate application backends independently at a per-PDB level, you can manage many PDBs at the CDB level. When deploying Oracle Multitenant, organizations should standardize several combinations of CDB parameters, based on a small number of SLAs (service-level agreements) defined for their various application backends — for example, "Gold," "Silver," and "Bronze." A Gold CDB, with the highest SLA requirements for your most critical application backends, might be configured with Oracle RAC (Real Application Clusters), Oracle RMAN (Recovery Manager), and Active Data Guard. For less critical application backends, a Bronze CDB might only require RMAN. At a minimum, you should create separate CDBs for each SLA level. You can then configure other options differently for each PDB in a CDB.

REMEMBER

Choices that need to be made for the CDB as a whole include

- **Oracle Database software version and platform specifics.** Every PDB within the same CDB must run the same Oracle Database version. Both within-database consolidation models — schema-based and PDB-based (see Chapter 2) — require a single choice of platform and operating system for all the consolidated application backends.
- **Control files, (s)pfile, and password file.** These files are common for the CDB as a whole.
- **Data Guard, RMAN backup, redo and undo.** Data Guard and RMAN backup are each managed as a single task for all PDBs within the CDB, rather than as separate tasks for each individual application backend. Both Data Guard and RMAN backup rely on the redo and undo logs for the database, and therefore redo and undo are managed for the CDB as a whole.

- **Oracle Real Application Clusters (RAC).** In a RAC environment, the CDB is opened across the entire cluster. However, the services for individual PDBs can be selectively opened in specific nodes of the cluster if desired, thus allowing for partitioning of the SGA across the cluster and increasing scalability.
- **Character set.** The character set applies to the CDB as a whole.

  Note: Some legacy applications may have an application backend that uses a specific character set that cannot be easily re-engineered. These legacy applications are an excellent use case for having multiple CDBs on the same platform, each created with its own character set.
- **CDB-wide initialization parameters and database properties.** Approximately 200 initialization parameters and database properties are set at the CDB-level, rather than at the level of the individual PDBs.
- **AWR (Automatic Workload Repository) Reports.** You can create AWR Reports for the CDB as a whole. To support ad hoc performance investigations within the scope of a single application backend, you can also create AWR Reports for a particular PDB.

TIP

In addition to separate CDBs for different SLAs, your organization may need to create other CDBs to address further operational requirements, such as standby databases and upgrades.

Because each PDB is functionally isolated from other PDBs in the same CDB, you can make several different choices for each PDB in a CDB, such as:

- **PDB point-in-time recovery.** One of the most notable features in the multitenant architecture is the ability to perform PDB point-in-time recovery without interfering with the availability of other PDBs in the same CDB.
- **Ad hoc RMAN backup for a PDB.** Performing scheduled RMAN backups of the CDB enables you to manage many databases as one database. However, occasions may arise when it is useful to make an ad hoc backup for a particular PDB.
- **PDB-settable initialization parameters and database properties.** A small number of initialization parameters and database properties can be set at the PDB-level to provide granular control over individual PDBs.
- **Temporary tablespace.** By default, the CDB has a single TEMP tablespace, but each PDB may create its own temporary tablespace. Each PDB also has its own set of SYSTEM and SYSAUX tablespaces, among others (see Figure 4-1).

There is an inheritance model for the PDB-level parameters. These parameters are inherited from the CDB, but some can be overridden at the PDB level.

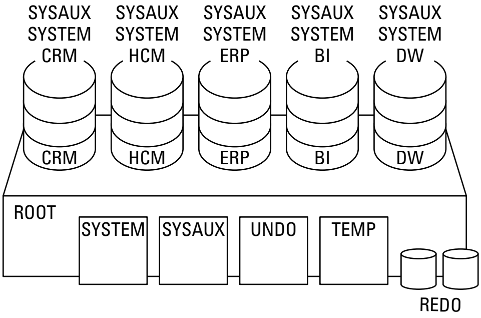

Figure 4-1: Namespaces and files in the CDB and PDBs.

Adopting a Non-CDB as a PDB

Two approaches are available for adopting a non-CDB as a PDB: direct adoption of a non-CDB as a PDB (known as the *upgrade-and-plug-in adoption approach*) and adoption of the content of a non-CDB into an empty PDB (known as the *Data Pump adoption approach*).

Upgrade-and-plug-in adoption

The upgrade-and-plug-in adoption approach is very simple — both conceptually and practically. The non-CDB is simply placed into read-only mode, a description file (or manifest) is generated, and the non-CDB is shut down. The non-CDB's datafiles can now be plugged into the target CDB as if it were an unplugged PDB and then opened as a PDB. Finally, a post-plug script is run to remove any redundant metadata.

You should run an ad hoc RMAN backup after shutting down the to-be-adopted non-CDB.

Oracle Database 12*c* supports both the new multitenant architecture and the old non-CDB architecture. Only an Oracle Database version 12.1 (or later) non-CDB can be directly adopted as a PDB. Thus, an earlier non-CDB version must first be upgraded to Oracle Database version 12.1 (or later).

Data Pump adoption

When the size of the non-CDB to be adopted is relatively small, using the Data Pump adoption approach rather than the upgrade-and-plug-in adoption approach might be quicker.

Oracle Data Pump, introduced in Oracle Database 11*g*, provides fast data and metadata movement between Oracle databases.

In Oracle Database 12*c*, Data Pump supports full database export and import, making maximum possible use of transportable tablespaces using single commands. This capability is called Full Transportable Export/Import, and you can use it to migrate most of the customer-created data from a non-CDB to a PDB.

You cannot move certain kinds of database artifacts (for example, XML schemas) using Data Pump. To migrate other artifacts, such as directories and grants made to public, you need to use ad hoc methods.

Upgrading Consolidated Non-CDBs to PDBs

Upgrading Oracle databases that were previously consolidated using the dedicated database or within-database models (see Chapter 2) to PDBs is a relatively straightforward process. Assuming only one application backend per database, you simply upgrade each individual database on the physical server (in the dedicated database model) or on each virtual machine to a PDB.

To upgrade databases that have been consolidated using the schema-based model (also discussed in Chapter 2), you must first determine whether there are any inter-schema dependencies in the databases you plan to upgrade.

If each application backend is fully isolated within a single schema, you can create a separate PDB for each schema. For example, if there are three independent schemas in a single non-CDB, upgrade to Oracle Database 12*c* and adopt the database as a PDB. Next, you would create additional clones of the PDB (see Chapter 3) for each independent schema — giving you a total of three PDBs in this example — and then simply drop the other schemas from each PDB so that only one of each schema exists per PDB.

However, if there are inter-schema dependencies, the schemas can't be easily separated into different PDBs without breaking the dependencies. Instead, the inter-dependencies must be carefully analyzed. Interdependent schemas should be treated as single application databases to be upgraded to a single PDB.

Deploying a Single PDB in a CDB

Suppose that an application backend has such a high throughput requirement that a particular platform must be dedicated exclusively to it. You could install the application backend either in a non-CDB or in a PDB that is the only PDB in its CDB. So why choose the single PDB in a CDB deployment option?

Even if you never exceed one PDB in a CDB, the multitenant architecture brings significant benefits. In terms of functionality, you get advanced “third generation” Data Pump capabilities that provide a relatively easy and extremely fast way to move data from one database to another by simply unplugging the PDB from one CDB and plugging it back into another CDB.

Along the same lines, this deployment model also creates a new paradigm for patching the Oracle version or upgrading to newer server hardware. For example, to upgrade a lone PDB, simply create a new CDB at the upgraded version level or on the new server, then unplug the PDB from the older CDB and plug it into the upgraded CDB.

Managing Resources between PDBs

When multiple sessions using different application backends are hosted on the same platform, they must compete for computing resources that include

- CPU
- I/O
- Number of concurrent sessions
- Parallel execution servers

Oracle Multitenant provides a powerful declarative mechanism that enables administrators to tell the Oracle Database where the resource boundaries are: Each application backend is installed in its own PDB, and no PDB houses more than one application backend.

Oracle Resource Manager is enhanced in Oracle Database 12*c* by the ability to define policies that govern the resource contention between the PDBs contained in the CDB. Policies may be defined to govern CPU, Exadata I/O, sessions, and parallel execution servers.

Resource Manager uses an industry-standard model to manage resource contention: *share* (the number of shares allocated to each PDB) and *cap* (the maximum utilization limit applied to each PDB). A given resource, such as CPU cycles, is divided into even shares and allocated to an application backend or PDB, for example. As the number of application backends competing for resources gradually increases, a cap ensures that headroom remains for the total planned capacity of the resource. This cap helps manage user expectations with regard to application performance.

For example, when an application is moved to a newer, more powerful server and it is the only application that is initially hosted on that server, performance gains are easily recognizable. However, as more applications are consolidated onto the server, performance decreases. It is better, therefore, to cap the performance for an application when it is first moved to a consolidation server to a level that can be expected when consolidation is complete.

Chapter 5

Ten Benefits of Oracle Multitenant

In This Chapter

- Exploring ten great reasons to use Oracle Multitenant

Oracle Multitenant enables organizations to easily consolidate multiple databases and accelerate their journey to the database cloud — all without changing their applications. This chapter explains some of the most compelling benefits of this new multitenant architecture.

Maximum Consolidation Density

Achieving maximum database consolidation density using schema-based consolidation (see Chapter 2) typically requires major application rewrites. Oracle Multitenant simplifies the consolidation process by plugging multiple pluggable databases (PDBs) into a single multitenant container database (CDB) without changing applications. In the multitenant architecture, memory and background processes are shared by all PDBs in a CDB.

Rapid Provisioning

Provisioning databases for various purposes including testing, development, and problem diagnosis can be very time consuming. Oracle Multitenant enables rapid provisioning of new databases as PDBs within an existing multitenant container database (CDB).

Easy Cloning

Database administrators can easily copy production databases and plug them into development and test containers. Additionally, if the underlying file system, such as Oracle ZFS or Oracle ACFS (Automatic Storage Management Cluster File System), supports copy on writes, cloning of PDBs can occur within containers almost instantaneously.

Faster Upgrades and Patches

Database administrators have to apply upgrades and patches to keep their databases current with software releases and fixes. In traditional database architectures, such updates must be applied to each individual database, including production, test, and development databases. With Oracle Multitenant, upgrades and patches are applied to the CDB only — not to each PDB — thereby simplifying and speeding up the entire process.

Flexible Upgrades

In some situations, you may need to perform selective updates and not concurrently update all PDBs in a container. In these situations, administrators simply create a new updated CDB and selectively unplug PDBs from existing CDBs, then plug them into the new CDB with the latest updates.

Simplified Oracle Recovery Manager (RMAN) Backups

Instead of running separate database backups, with Oracle Multitenant, databases need to be backed up at the multitenant container level only. In other words, all PDBs consolidated into a CDB will be backed up as one, and administrators retain the flexibility to perform recovery operations at the individual PDB level, if required.

Granular Point-in-Time Recovery

In the multitenant architecture, point-in-time recovery (PITR) is supported at the pluggable database level, giving you granular control of recovery operations at a per-PDB level, when required.

Comprehensive Data Protection

With Oracle Multitenant, administrators can easily manage standby systems in another datacenter using Active Data Guard. You only need to set up a standby configuration at the multitenant container level in order to replicate all PDBs in that CDB.

Efficient Resource Management

The multitenant architecture provides efficient control of intra-PDB resource contention using Oracle Resource Manager.

Scale-out Capability

Databases can scale out quickly and easily in the multitenant architecture by opening only certain PDBs on certain nodes in an Oracle Real Application Clusters (RAC) configuration.

TIP

Oracle Multitenant is fully compatible with existing database options and features including Oracle Real Application Clusters (RAC), Active Data Guard, Real Application Testing, and Recovery Manager.

Accelerate your journey to the database cloud

Oracle Database 12*c* introduces a new architecture — Oracle Multitenant — that greatly simplifies database consolidation and common database operations such as provisioning, patching, upgrading, and cloning. In this book, you find out how!

- ***Understand the new multitenant architecture* — *and its benefits***
- ***Provision, patch, and upgrade databases* — *safely and efficiently with minimal downtime***
- ***Standardize and consolidate your databases* — *and manage many as one***

Oracle engineers hardware and software to work together in the cloud and in your data center. For more information about Oracle (NYSE:ORCL), visit oracle.com.

Open the book and find:

- **How multitenant container databases and pluggable databases simplify database management**
- **How to increase database consolidation density**
- **Where to deploy Oracle Multitenant**
- **How to reduce the cost of managing your databases**

Go to Dummies.com®
for videos, step-by-step examples, how-to articles, or to shop!

FOR DUMMIES®
A Wiley Brand

ISBN: 978-1-118-72870-3
Not for resale