ORACLE

ORACLE DOJO NR. 7

R. DURBEN, M. HOßFELD, S. SOLBACH

Oracle 12c

Multitenant Database Option

Oracle Dojo ist eine Serie von Heften, die Oracle Deutschland B.V. zu unterschiedlichsten Themen aus der Oracle-Welt herausgibt.

Der Begriff Dojo [‘do:dʒo] kommt aus dem japanischen Kampfsport und bedeutet Übungshalle oder Trainingsraum. Als „Trainingseinheiten“, die unseren Anwendern helfen, ihre Arbeit mit Oracle zu perfektionieren, sollen auch die Oracle Dojos verstanden werden. Ziel ist es, Oracle-Anwendern mit jedem Heft einen schnellen und fundierten Überblick zu einem abgeschlossenen Themengebiet zu bieten.

Im *Oracle Dojo Nr. 7* beschäftigen sich Ralf Durben, Manuel Hoßfeld und Sebastian Solbach aus der Business Unit Datenbank mit der neuen Oracle Datenbank Version **Oracle 12c** und führen in eine der wichtigsten Neuerungen ein, die 12c zu bieten hat: Multitenant Datenbankoption aka Pluggable Database.

ORACLE®

Inhalt

ORACLE

ORACLE DOJO NR. 7

RALF DURBEN, MANUEL HOßFELD, SEBASTIAN SOLBACH

Oracle 12c

Multitenant Database Option

VORWORT DES HERAUSGEBERS

Daten sind wieder „in“, sind wieder „sexy“. Daten sind wieder in den Mittelpunkt des Interesses gerückt. Dies ist auch der Grund, weshalb Datenbanksysteme eine großartige Renaissance erleben und die Datenbankspezialisten – DBAs, DB-Designer, Entwickler und Daten-Analytiker – zur gefragten Spezies machen. Als Datenbänker kann ich nur sagen: Gut so, denn Daten sind der wahre Schatz eines Unternehmens, und ein Unternehmen, das diesen Schatz ignoriert oder geringschätzig behandelt, handelt mehr als grob fahrlässig.

Die Fragen nach einer effizienten, skalierbaren und sicheren Datenbankinfrastruktur, die auch zukünftigen Anforderungen standhält, werden heute mehr denn je diskutiert. „Database as a Service“ (DBaaS) wird zum Maß der DB-Infrastruktur erhoben. Auch die vielen neuen DBMS, die in den letzten Jahren entstanden sind, zeigen die hohe Dynamik und Innovationskraft dieses Marktes und sind für uns als Marktführer Ansporn und Jungbrunnen zugleich.

Viele neue Anwendungen werden in den nächsten Jahren in immer schnellerer Folge in den Firmen ausgerollt und bestehende Systeme müssen renoviert und auf den neuesten Stand gebracht werden. Dabei werden neue Technologien und Entwicklungen wie zum Beispiel Big Data, die Einbeziehung

von völlig neuen Datenklassen und deren Auswertung (Analytic), eine immer stärkere Rolle spielen. Entscheidend für den Erfolg solcher Maßnahmen werden zwei Dinge sein: **Datenbanksysteme**, die die notwendigen Funktionalitäten liefern und **Menschen**, die die notwendigen Fertigkeiten besitzen, um die Anwendungssysteme zu bauen und zu betreiben.

Eine spannende Zeit und ein optimaler Zeitpunkt, um eine neue Oracle Datenbank Version auf den Markt zu bringen. **Oracle 12c** ist der neue Benchmark für Datenbanksysteme.

In den nächsten Monaten werden wir eine Reihe von Dojos veröffentlichen, die sich alle mit Themen rund um Oracle 12c beschäftigen. Dieses Dojo verantworten drei geschätzte Kollegen aus der Business Unit Datenbank: Ralf Durben, Manuel Hoßfeld und Sebastian Solbach. Sie führen in eine der wichtigsten Neuerungen ein, die 12c zu bieten hat: Multitenant Database Option aka Pluggable Database. Eine „coole" Technologie, die den Betrieb von Oracle Datenbanksystemen nicht weniger als revolutionieren wird.

Ich wünsche Ihnen viel Spaß beim Lesen und beim Testen.

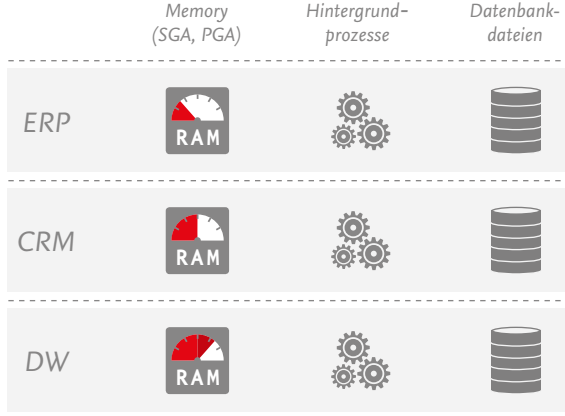
Ihr Günther Stürner
Vice President Sales Consulting

PS: Wir sind an Ihrer Meinung interessiert. Anregungen, Lob oder Kritik gerne an barbara.frank@oracle.com. Vielen Dank!

1 Einleitung

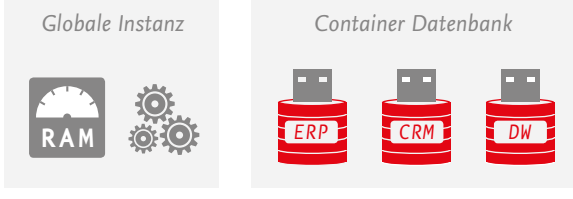
Die **Pluggable Database** in der Oracle Datenbank 12c ist nicht nur ein neues Feature, sondern eine für Oracle völlig neue Architektur zum Betrieb von Datenbanken, die auf vielen Gebieten Veränderungen und große Vorteile bewirkt. Dabei trennt Oracle die strikte Eins-zu-eins-Beziehung von Datenbank und Instanz und erlaubt den Betrieb einer globalen Instanz für mehrere Datenbanken.

Dieses Dojo zeigt die Neuerungen auf, die sich durch das Konzept der Pluggable Database ergeben. Basiswissen zur Administration von Oracle-Datenbanken wird vorausgesetzt.



Ein Oracle-Datenbanksystem bis zur Version 11 besteht grundsätzlich aus zwei Komponenten: der Datenbank selbst in Form von Dateien und einer Instanz (oder mehrerer Instanzen im Fall von RAC) in Form von Betriebssystemprozessen und Hauptspeicher. Werden mehrere Datenbanksysteme auf einem Server betrieben, sind entsprechend viele Instanzen aktiv.

Ab der Version 12 der Oracle-Datenbank gibt es die gravierende Neuerung, dass sich mehrere Datenbanken eine globale Instanz teilen können.



Die einzelne Datenbank wird also in eine Container-Datenbank „eingeklinkt“ und mit der globalen Instanz betrieben. Je nach Bedarf und Anwendung kann eine Datenbank von einer Container-Datenbank zu einer anderen umziehen.

Damit eröffnen sich sowohl in nahezu allen klassischen als auch in neuartigen Funktionalitätsbereichen völlig neue Möglichkeiten.

1.1 NEUE BEGRIFFE UND ABKÜRZUNGEN

Im Rahmen der Pluggable Databases werden neue Bezeichnungen für die einzelnen Datenbankformen eingeführt:

- Pluggable Database (PDB)

Eine für Anwendungen genutzte Datenbank, die unter Verwendung der neuen Architektur betrieben wird. Sie enthält alle Anwendungsdaten samt Data Dictionary, Datenbankbenutzer usw.

- Container Database (CDB)

Der Container, in den die PDBs eingeklinkt werden. Alle PDBs innerhalb einer CDB benutzen die gleiche Datenbankversion, die durch die CDB vorgegeben wird.

- Non-CDB

Eine Oracle-Datenbank, die nach herkömmlicher Architektur unter Oracle 12c betrieben wird.

- Seed-DB

Ein Datenbanktemplate, mit dem sehr schnell eine neue PDB erzeugt werden kann. Jede CDB enthält genau eine Seed-DB mit dem Namen PDB$SEED.

1.2 WAS ÄNDERT SICH GENAU?

Veränderungen gibt es sowohl bei den Bestandteilen des Datenbanksystems als auch beim Inhalt der Datenbank selbst:

Änderungen bei den Bestandteilen des Datenbanksystems

Eine PDB unterscheidet sich von einer Non-CDB dadurch, dass sie keine eigene(n)

- Instanz (Memory und Hintergrundprozesse)
- Kontrolldatei
- Redolog-Dateien
- Undo Tablespaces

besitzt. Diese Komponenten werden zentral von der CDB bereitgestellt, in der die PDB betrieben wird.

Änderungen beim Inhalt der Datenbanken

Im Rahmen der Pluggable Databases wird das Data Dictionary aufgeteilt in:

- Einen statischen Bereich, der die für die eingesetzte Software feststehenden Informationen beinhaltet. Dieser Bereich enthält Informationen, die während der Lebenszeit der Datenbank mit der jeweils gegebenen Version nicht mehr verändert werden.
- Einen anwendungsbezogenen Bereich, dessen Inhalte Veränderungen unterliegen können.

Der statische Bereich ist Bestandteil der CDB und wird über eine Verlinkung in den anwendungsbezogenen Bereich, der Bestandteil der PDB ist, einbezogen. Aus der Sicht einer PDB hat diese also ein komplettes Data Dictionary, auch wenn die nicht veränderlichen Teile davon aus der CDB stammen. Die interne Trennung hat keinerlei Auswirkung auf die Nutzung des Data Dictionarys!

1.3 WELCHE VORTEILE BIETET DIE NEUE ARCHITEKTUR?

Die neue Architektur bietet viele interessante Vorteile und Anwendungsmöglichkeiten:

Konsolidierung und Einsparung von Ressourcen

Im Rahmen von Konsolidierungsbestrebungen, zum Beispiel zur besseren Auslastung der vorhandenen Server, können mehrere Oracle-Datenbanken in einem gemeinsamen ORACLE_HOME betrieben werden, wenn diese die gleiche Datenbankversion nutzen sollen. Bei herkömmlichen Datenbanken (Non-CDBs) sind die Datenbankinstanzen aber oft nicht ständig ausgelastet. Das bedeutet praktisch, dass

- allokierter Hauptspeicher (zum Beispiel für die SGA) ungenutzt ist und auf dem Server nicht anderweitig verwendet werden kann.

- Hintergrundprozesse laufen, obwohl sie gerade nicht benötigt werden. Auch nicht ausgelastete Prozesse bedeuten durch die betriebssystemseitigen Prozesswechsel und den sonstigen Verwaltungsaufwand einen erhöhten Overhead.

Durch die von mehreren PDBs gemeinsam genutzten Instanzen können diese Ressourcen also viel effizienter genutzt werden. Bei Oracle-internen Tests konnte eine bis zu sechsfache Effizienzsteigerung beobachtet werden. Die Ressourcen, die zum Betrieb von 20 herkömmlichen Datenbanken benötigt wurden, reichten aus, um 120 PDBs zu betreiben. Diese Zahlen sind Einzelbeispiele und nicht unbedingt auf alle Umgebungen anwendbar.

Auch die Aufteilung des Data Dictionarys in einen zentralen statischen Bereich in der CDB und einen dezentralen Bereich in den PDBs bewirkt eine deutliche Absenkung des Storagebedarfs, da der zentrale Bereich nur noch einmal gespeichert wird.

Mandantenfähigkeit

Trotz Zusammenlegung einiger Bestandteile der einzelnen PDBs bleibt die volle Mandantenfähigkeit erhalten. Die Benutzer mit Zugriff auf eine PDB können nicht auf andere PDBs in der CDB zugreifen. Somit kann zum Beispiel für eine gegebene Anwendung für jeden Mandanten eine eigene PDB erstellt werden, ohne dass der volle Overhead einer eigenen herkömmlichen Datenbank erzeugt wird.

Erstellen einer Datenbank

Jede CDB beinhaltet eine sogenannte Seed-Datenbank, also eine Muster-Datenbank, die als Basis für eine PDB genutzt werden kann. Zum Erstellen einer neuen PDB wird einfach eine Kopie dieser Seed-Datenbank angefertigt. Damit reduziert sich die Wartezeit auf eine neue Datenbank auf den Kopiervorgang dieser Seed-Datenbank, der von Oracle entsprechend optimiert wurde. Somit steht eine neue PDB nach wenigen Minuten zur Verfügung.

Mobilität

Eine PDB kann jederzeit aus einer CDB entnommen („unplugged“) werden. Dabei werden die Datenbankdateien und die dazu passenden Metadaten zusammengestellt. Mit diesen Dateien kann die PDB nun in eine andere CDB eingeklinkt („plugged“) werden.

Cloudfähigkeit

Im Rahmen einer Cloud-Strategie ist eine Lösung für ein Database as a Service (DbaaS) also das einfache und schnelle Bereitstellen einer Datenbank als Service, von besonderer Wichtigkeit. Des Weiteren wird von einer Datenbank-Cloud erwartet, dass die Zuordnung der betriebenen Datenbanken zu den eingesetzten Servern möglichst flexibel ist, die Datenbanken also möglichst einfach „umziehen“ können. Beides wird von der Pluggable Database optimal umgesetzt.

Patchen

Das Patchen einer Oracle-Datenbank kann mithilfe der neuen Architektur extrem vereinfacht werden: Eine PDB kann gepatcht oder werden, indem sie zunächst aus ihrer ursprünglichen CDB entnommen („unplugged“) und anschließend in eine mit dem entsprechenden Patch versehenen neuen CDB eingeklinkt („plugged“) wird.

Datenbank-Cloning

Eine PDB kann sehr leicht und schnell dupliziert werden. Somit ist es mit wenig Aufwand möglich einen Datenbankklon zu erstellen, zum Beispiel um eine Testdatenbank aufzusetzen.

Resource Manager

Die Zuteilung von Ressourcen, wie zum Beispiel CPU oder Parallelität, ist seit vielen Jahren mit dem **Oracle Resource Manager** möglich, jedoch nur innerhalb einer gegebenen Instanz. Der Resource Manager konnte bislang zum Beispiel also nie ein Ressourcenmanagement für alle auf einem Server betriebenen Datenbanken mit ihren Instanzen vornehmen.

Mit der neuen Architektur steht der Resource Manager auch auf der Ebene der CDB zur Verfügung und kann so Ressourcen auch PDB-übergreifend zuteilen und begrenzen.

Data Guard

Der Betrieb von Physical-Standby-Datenbanken erfolgt auf der Ebene der CDB und somit für alle in ihr betriebenen PDBs. Das liegt daran, dass die Online-Redolog-Dateien auf der Ebene der CDB genutzt werden und somit Redo-Daten aller PDBs beinhalten.

Der große Vorteil liegt darin, dass jeweils mit einem Kommando das Erzeugen beziehungsweise der Betrieb von Standby-Datenbanken für alle in einer CDB betriebenen PDBs durchgeführt und somit der Aufwand für die Administration extrem reduziert wird.

Backup & Recovery

Das Erzeugen von Backups erfolgt auf der Ebene der CDB. Mit dem Erstellen eines Backups werden also alle in der CDB betriebenen PDBs gesichert. Auch hier zeigt sich eine große Aufwandseinsparung für die Administration.

Ein Restore, also das Wiedereinspielen eines Backups, kann sowohl auf CDB- als auch auf PDB-Ebene erfolgen, sogar für ein Point-in-Time Recovery einer einzelnen PDB. Die Erleichterung durch das Erzeugen eines gemeinsamen Backups für die gesamte CDB führt also zu keinerlei Einschränkungen im Bereich Recovery auf PDB-Ebene.

2 Abgrenzung und Einordnung gegenüber anderen Virtualisierungstechniken

Wie schon im ersten Kapitel erläutert, handelt es sich bei den Pluggable Databases im Grunde um eine Virtualisierung von Datenbanken mit den Mitteln der Datenbank selbst. Doch auch schon vor 12c konnte man Datenbanken in virtuellen Maschinen betreiben und mittels Schema-Konsolidierung oder dem Zusammenlegen verschiedener ORACLE_HOMEs auf einem Server ähnliche Effekte erreichen, wie sie PDBs versprechen.

Worin genau liegen nun also die Unterschiede dieser schon länger bekannten Methoden und wie lassen sich diese zu PDBs abgrenzen?

2.1 SCHEMA-KONSOLIDIERUNG

Die Idee der Schema-Konsolidierung geht davon aus, dass Datenbankanwendungen nicht zwingend tatsächlich eine „eigene" Datenbank benötigen und durch Zusammenlegen der DB-Schemas solcher Anwendungen in eine einzelne Datenbank eine Konsolidierung mit hohem Potenzial zur Ressourceneinsparung verwirklicht werden kann. Weitere

Vorteile liegen in den damit verbundenen Möglichkeiten für zentrale Verwaltung sowie zentralem Backup/Recovery.

Die Einschränkungen und Grenzen der Schema-Konsolidierung fallen schnell ins Auge: Zunächst funktioniert diese von vorneherein natürlich nur bei DB-Anwendungen, deren Schema-Namen sich auch tatsächlich unterscheiden oder zumindest nachträglich umbenennen lassen. Dies ist jedoch bei einigen Applikationen nicht der Fall, wenn diese „hartkodiert" nur einen einzigen Schema-Namen kennen beziehungsweise auf diesem beharren.

Eine weitere Einschränkung liegt darin, dass bei der Schema-Konsolidierung natürlich auch für alle auf diese Weise konsolidierten Anwendungen die gleichen Einstellungen der Datenbank gelten. Sollten zwei der Konsolidierungskandidaten komplett entgegengesetzte Vorstellungen davon haben, wie zum Beispiel Memory-Bereiche beschaffen sein müssen, können diese nicht in der gleichen Datenbank laufen. Auch die mangelnde Isolation der einzelnen Datenbankanwendungen untereinander kann ein Problem darstellen: Zwar gibt es mit dem Resource Manager in der Datenbank Mittel und Wege um zumindest zu verhindern, dass ein einzelnes Schema (respektive dessen Nutzer) alle Ressourcen der DB belegen kann. Sowohl aus Security-Sicht als auch bei der Notwendigkeit von verschiedenen Versionen/Patchleveln ist

es aber oft erforderlich, Anwendungen stärker voneinander abgrenzen zu können als nur durch Verwendung einzelner Schemas.

2.2 ORACLE_HOME-KONSOLIDIERUNG

Im Gegensatz zur Schema-Konsolidierung werden bei der sogenannten ORACLE_HOME-Konsolidierung nicht die DB-Schemas mehrerer Anwendungen in einer DB zusammengefasst, sondern mehrere Datenbanken in einem ORACLE_HOME.

Jede Datenbank ist eigenständig und verwendet eigene Ressourcen im Bereich Storage, Memory und CPU. Einzig die Software, zum Beispiel die Binaries, wird gemeinsam genutzt. Pro Datenbank ist es also möglich, unterschiedliche DB-Parametrisierungen zu verwenden und auch gleichlautende Schema-Namen stellen kein Problem dar. Im Gegenzug geht jedoch der Vorteil von zentralem Backup/Recovery verloren und auch die zentrale Administration ist nur noch bedingt gegeben. Erwähnenswert ist auch die Tatsache, dass sich alle so konsolidierten Datenbanken auf ein- und demselben Versionsstand befinden müssen, da nur ein gemeinsames ORACLE_HOME genutzt wird.

2.3 SERVER-VIRTUALISIERUNG MITTELS VIRTUELLER MASCHINEN (VMs)

Den höchsten Grad der Isolation im Sinne von Kapselung einzelner zu konsolidierender Datenbanken erreicht man mit der Server-Virtualisierung. Hier werden im Grunde „nur“ die zuvor auf einzelnen physischen Maschinen betriebenen Datenbanken komplett mit dem darunterliegenden Betriebssystem in eine virtuelle Maschine „verschoben“. Dies ermöglicht dann bei entsprechend ausgestatteten Servern, dass gegebenenfalls viele dieser ehemals einzelnen Datenbanken auf einem einzigen physischen Server betrieben werden können.

2.4 ZUSAMMENFASSUNG UND GEGENÜBERSTELLUNG MIT PLUGGABLE DATABASES

Die zuvor beschriebenen Eigenheiten der einzelnen Ansätze lassen sich wie folgt tabellarisch zusammenfassen (Bestnote sind 5 Sterne).

Wie man sieht, beinhaltet das Konzept der Pluggable Databases die Vorteile der Schema-Konsolidierung ohne deren Nachteile.

	Pluggable Databases	*Schema-Konsolidierung*	*ORACLE_HOME-Konsolidierung*	*Server-Virtualisierung*
Individuelle Einstellungen	***	-	*****	*****
Gleichlautende Schema-Namen möglich	*****	-	*****	*****
Namensfreiheit bei Non-Schema-Objekten	*****	-	*****	*****
Isolation der Daten	*****	*	***	*****
Individuelles Verschieben	*****	*	***	*****
Isolierte Verwaltung durch Fachabteilungen oder Entwickler	*****	*	***	*****
Zentrale Verwaltung/ Betriebsführung	*****	*****	***	*
Zentrales Desaster Recovery	*****	*****	-	-
Zentrales Backup & Recovery	*****	*****	-	-
Zentrales Ressourcenmanagement	*****	***	*	-

3 Deployment in Pluggable Databases

Das in den folgenden Abschnitten beschriebene Deployment in Pluggable Database-Umgebungen, also das Erstellen und Löschen von CDBs und PDBs, sowie das „Plug“ und „Unplug“ von PDBs, ist sowohl mit SQL-Kommandos als auch teilweise im **Database Configuration Assistant (DBCA)** möglich.

Im DBCA wählen Sie im Schritt 1 die *Operation Manage Pluggable Databases* aus. Sie erhalten im zweiten Schritt die Optionen:

- *Create a Pluggable Database*
- *Unplug a Pluggable Database*
- *Delete a Pluggable Database*
- *Configure a Pluggable Database*

Auch steht Ihnen **Oracle Enterprise Manager Cloud Control** als Verwaltungswerkzeug zur Verfügung. In diesem Dojo wird nur auf die SQL-Kommandos eingegangen.

3.1 ERSTELLEN VON CDBs

Das Erstellen einer CDB ist vor allem bei der Verwendung des Database Configuration Assistant (DBCA) sehr einfach,

denn dort aktivieren Sie nur die Checkbox *Create as Container Database*. Wenn Sie das Kommando CREATE DATABASE direkt verwenden, müssen Sie die Klausel ENABLE PLUGGABLE DATABASE zusammen mit Angaben zur Seed-Datenbank machen.

Das Kommando sieht dann zum Beispiel folgendermaßen aus:

```
CREATE DATABASE newcdb
   USER SYS IDENTIFIED BY sys_password
   USER SYSTEM IDENTIFIED BY system_password
   LOGFILE GROUP 1 ('/u01/logs/my/redo01a.log','/u02/
                   logs/my/redo01b.log')
             SIZE 100M BLOCKSIZE 512,
           GROUP 2 ('/u01/logs/my/redo02a.log','/u02/logs/
                    my/redo02b.log')
             SIZE 100M BLOCKSIZE 512,
           GROUP 3 ('/u01/logs/my/redo03a.log','/u02/logs/
                    my/redo03b.log')
             SIZE 100M BLOCKSIZE 512
   MAXLOGHISTORY 1
   MAXLOGFILES 16
   MAXLOGMEMBERS 3
   MAXDATAFILES 1024
   CHARACTER SET AL32UTF8
   NATIONAL CHARACTER SET AL16UTF16
   EXTENT MANAGEMENT LOCAL
```

```
DATAFILE '/u01/app/oracle/oradata/newcdb/system01.dbf'
    SIZE 700M REUSE AUTOEXTEND ON NEXT 10240K MAXSIZE
    UNLIMITED
SYSAUX DATAFILE '/u01/app/oracle/oradata/newcdb/sysaux01.dbf'
    SIZE 550M REUSE AUTOEXTEND ON NEXT 10240K MAXSIZE
    UNLIMITED
DEFAULT TABLESPACE deftbs
    DATAFILE '/u01/app/oracle/oradata/newcdb/deftbs01.dbf'
    SIZE 500M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED
DEFAULT TEMPORARY TABLESPACE temptbs1
    TEMPFILE '/u01/app/oracle/oradata/newcdb/temp01.dbf'
    SIZE 20M REUSE AUTOEXTEND ON NEXT 640K MAXSIZE UNLIMITED
UNDO TABLESPACE undotbs1
    DATAFILE '/u01/app/oracle/oradata/newcdb/undotbs01.dbf'
    SIZE 200M REUSE AUTOEXTEND ON NEXT 5120K MAXSIZE UNLIMITED
ENABLE PLUGGABLE DATABASE
    SEED
    FILE_NAME_CONVERT = ('/u01/app/oracle/oradata/newcdb/',
                         '/u01/app/oracle/oradata/pdbseed/')
    SYSTEM DATAFILES SIZE 125M AUTOEXTEND ON NEXT 10M MAXSIZE
    UNLIMITED
    SYSAUX DATAFILES SIZE 100M
USER_DATA TABLESPACE usertbs
    DATAFILE '/u01/app/oracle/oradata/newcdb/usertbs01.dbf'
    SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

Der untere Bereich betrifft die Erstellung als CDB: Die Klausel ENABLE PLUGGABLE DATABASE gibt an, dass die Datenbank als CDB erstellt werden soll. Dann geben Sie mit SEED an, wie die Seed-Datenbank aussieht. Die System-Datendateien werden von der CDB übernommen und die Namen mit FILE_NAME_CONVERT umgewandelt – einzig die Speichergrößen passen Sie noch an. Dann wird noch ein Daten-Tablespace mit Namen, Namen der Datendatei und Speichergröße definiert.

3.2 ERSTELLEN VON PDBs

Eine PDB wird grundsätzlich in einer Datenbanksitzung in der CDB erstellt. Der Datenbankbenutzer muss dabei das System-Privileg CREATE PLUGGABLE DATABASE haben. Die neue PDB darf keinen Namen einer in dieser CDB bereits bestehenden PDB bekommen. Sie haben nun verschiedene Optionen:

- Erstellen einer PDB mit der Seed-DB
- Erstellen einer PDB durch Klonen einer anderen PDB
- Migration einer Non-CDB zu einer PDB

Alle drei Varianten werden im Folgenden beschrieben.

Erstellen einer PDB mit der Seed-DB

Die Seed-DB ist ein Template, mit dem Sie sehr schnell eine neue PDB erstellen können. Jede CDB enthält genau eine Seed-DB. Mit der Seed-DB erstellen Sie eine Basis-PDB, die dann erweitert werden kann. Die Syntax zur Nutzung ist denkbar einfach:

Wenn die CDB mit **Oracle Managed Files** konfiguriert wird verwenden Sie

```
SQL> CREATE PLUGGABLE DATABASE pdb1
       ADMIN USER syspdb IDENTIFIED BY passwort
       DEFAULT TABLESPACE data;
```

Sie können auch ein separates Verzeichnis angeben, wenn die neue PDB in einem alternativen Verzeichnis gespeichert werden soll. Achten Sie dabei darauf, dass das angegebene Verzeichnis existieren muß:

```
SQL> CREATE PLUGGABLE DATABASE pdb1
       ADMIN USER syspdb IDENTIFIED BY passwort
       CREATE_FILE_DEST = '/oracle/oradata/pdb1/'
       DEFAULT TABLESPACE data;
```

Wenn die CDB nicht mit Oracle Managed Files konfiguriert ist, verwenden Sie

```
SQL> CREATE PLUGGABLE DATABASE pdb1
       ADMIN USER syspdb IDENTIFIED BY passwort
       FILE_NAME_CONVERT = ('/oracle/oradata/pdbseed/',
                            '/oracle/oradata/pdb1/')
          DEFAULT TABLESPACE pdb1_data
            DATAFILE '/oracle/oradata/pdb1/pdb_data01.dbf'
            SIZE 200M;
```

Sie geben also einen neuen Administrations-Benutzer an und in der STORAGE-Klausel des CREATE PLUGGABLE DATABASE-Kommandos können Sie die Limits für die neue PDB festlegen. In der neuen PDB wird auch der Default Tablespace PDB1_DATA mit der angegebenen Datendatei erstellt. Bei der Erstellung der neuen PDB werden die Datendateien der Seed-DB in ein neues Verzeichnis kopiert, welches Sie mit der Klausel FILE_NAME_CONVERT angeben.

Die neu erstellte PDB muss für eine weitere Nutzung mit ALTER PLUGGABLE DATABASE OPEN geöffnet werden.

Erstellen einer PDB durch Klonen einer anderen PDB

Sie können eine neue PDB auch durch Klonen einer bereits bestehenden PDB erstellen. Die bestehende Datenbank kann in der gleichen oder auch einer anderen CDB liegen.

Bevor eine bestehende PDB geklont werden kann, muss sie in einen transaktionskonsistenten Zustand gebracht und READ ONLY geöffnet sein.

```
SQL> SHUTDOWN IMMEDIATE
SQL> ALTER DATABASE OPEN READ ONLY;
```

Wenn die bestehende PDB lokal in der gleichen CDB liegt, benutzen Sie zum Beispiel das Kommando

```
SQL> CREATE PLUGGABLE DATABASE pdb2 FROM pdb1
   PATH_PREFIX = '/oracle/oradata/pdb2'
   FILE_NAME_CONVERT = ('/oracle/oradata/pdb1/', '/
   oracle/oradata/pdb2/')
   STORAGE (MAXSIZE 100G MAX_SHARED_TEMP_SIZE 900M);
```

wobei der FILE_NAME_CONVERT verwendet wird, um den Speicherort der Datendateien der neuen PDB anzugeben und die bestehenden Dateien der PDB1 in das passende Verzeichnis für die PDB2 zu kopieren.

In der STORAGE-Klausel des CREATE PLUGGABLE DATABASE-Kommandos können Sie die Limits für die neue PDB festlegen. Im Beispiel dürfen alle Tablespaces der neuen PDB zusammen 100GB (MAXSIZE) umfassen und die Sessions in der neuen PDB dürfen insgesamt maximal 900MB im Shared Temporary Tablespace der CDB in Anspruch nehmen.

- Wenn Sie eine PDB ohne Benutzerdaten klonen möchten, benutzen Sie die Klausel NO DATA. Dabei werden die Datenbankobjekte (z.B. Tabellen, Indizes,...) in der „Kopie“ zwar angelegt, jedoch ohne Daten (gilt nicht für

Objekte in den Tablespaces SYSTEM und SYSAUX). Dieses funktioniert aber nur, wenn die Quelldatenbank in den Tablespaces außer SYSTEM und SYSAUX folgende Datenbankobjekte nicht beinhaltet:Index-organized Tabellen

- Advanced Queue (AQ) Tabellen
- Cluster Tabellen
- Tabellen Cluster

Das Kommando sieht dann so aus:

```
SQL> CREATE PLUGGABLE DATABASE pdb2 FROM pdb1
  PATH_PREFIX = '/oracle/oradata/pdb2'
  FILE_NAME_CONVERT = ('/oracle/oradata/pdb1/', '/
  oracle/oradata/pdb2/')
  NO DATA;
```

Wenn Sie eine PDB einer anderen CDB klonen möchten, müssen Sie einen Datenbank-Link zu dieser PDB erstellen und im CREATE PLUGGABLE DATABASE-Kommando angeben:

```
CREATE PLUGGABLE DATABASE pdb2 FROM pdb1@db_link_to_
pdb1;
```

Migration einer Non-CDB zu einer PDB

Es gibt vier verschiedene Wege, eine Non-CDB zu einer PDB zu migrieren:

- Direktes Einklinken der Non-CDB als PDB (nur für Non-CDB ab DB-Version 12.1.0.1)

Für die Non-CDB-Datenbank werden mit dem PL/SQL-Package DBMS_PDB, welches Bestandteil einer 12c-Datenbank ist, spezielle XML-Metadateien erzeugt. Mit diesen Dateien und den Datenbankdateien kann in der CDB mit dem Kommando CREATE PLUGGABLE DATABASE die Datenbank als PDB erzeugt werden.

- Klonen einer Non-CDB als PDB über Datenbank-Link (nur für Non-CDB ab DB-Version 12.1.0.2)

Sie können über einen Datenbank-Link eine Non-CDB direkt zu einer PDB klonen. Auf der einen Seite sparen Sie dabei den manuellen Kopiervorgang der Datenbankdateien, andererseits muß ein funktionierender Datenbank-Link von der CDB zur Non-CDB erstellt werden.

- Full Database Import (auch für ältere Datenbankversionen)

Mit DATAPUMP wird ein Full Database Export der Non-CDB angefertigt. In der CDB wird dann eine neue (leere) PDB erzeugt und der Inhalt der Non-CDB mittels Full Database Import eingefügt.

- Replikation mit GoldenGate

In der CDB wird dann eine neue (leere) PDB erzeugt und der Inhalt der Non-CDB mittels Replikation über GoldenGate eingefügt.

Im Folgenden werden die ersten drei Methoden näher beschrieben.

Direkte Migration einer Non-CDB zur PDB

Die Non-CDB muss für diese Methode mindestens die Version 12c tragen, da ein spezielles PL/SQL-Package für Pluggable Databases verwendet wird. Wenn Sie also eine Datenbank einer früheren Version als PDB einbinden möchten, migrieren Sie diese zuerst nach 12c oder verwenden Sie eine der anderen Methoden. Die Schritte der direkten Umwandlung zu einer PDB sind wie folgt:

1. Prüfen Sie, ob der Characterset der CDB den Characterset der Non-CDB enthält. Der Characterset der zukünftigen PDB muss mit dem der CDB kompatibel sein.
2. Erstellen Sie ein Backup der Non-CDB.
3. Sorgen Sie in der Non-CDB für einen transaktionskonsistenten Zustand und öffnen Sie dann die Datenbank im READ ONLY-Modus:

```
SQL> SHUTDOWN IMMEDIATE
SQL> STARTUP MOUNT
SQL> ALTER DATABASE OPEN READ ONLY;
```

4. Starten Sie die Prozedur DBMS_PDB.DESCRIBE in der Non-CDB:

```
SQL> BEGIN
        DBMS_PDB.DESCRIBE(
           pdb_descr_file => '/home/oracle/ncdb.xml');
    END;
    /
```

   Die erzeugte XML-Datei enthält Informationen über Speicherort und Einstellungen aller Datendateien.

5. Fahren Sie die Non-CDB herunter.

6. Falls die Non-CDB auf einem anderen Server zur PDB werden soll, kopieren Sie alle Datenbankdateien und die oben erzeugte XML-Datei auf den neuen Server.

7. Erstellen Sie jetzt eine neue PDB unter Verwendung der Non-CDB. Verwenden Sie dazu das Kommando CREATE PLUGGABLE DATABASE, zum Beispiel:

```
SQL> CREATE PLUGGABLE DATABASE pdb1
USING '/home/oracle/ncdb.xml'
STORAGE UNLIMITED;
```

Im obigen Beispiel wurden die Datendateien auf dem Zielserver mit den Originalpfaden gespeichert, sodass keine Umbenennung der Datendateien notwendig ist. Die Datendateien werden also auch nicht kopiert. Dieses ist im Allgemeinen aber nicht der Fall und dann sieht die Syntax eher so aus:

```
SQL> CREATE PLUGGABLE DATABASE pdb1
USING '/home/oracle/ncdb.xml'
SOURCE_FILE_NAME_CONVERT=('/oradata/noncdb',
                          '/oradata/tmp')
FILE_NAME_CONVERT=('/oradata/tmp',
                   '/oradata/pdb1')
STORAGE (MAXSIZE 100G MAX_SHARED_TEMP_SIZE 900M);
```

Die Klausel SOURCE_FILE_NAME_CONVERT gibt den temporären Speicherort der Kopien der Datendateien auf dem Server an. Mit FILE_NAME_CONVERT geben Sie dann das Zielverzeichnis der PDB-Datendateien an.

Im vorliegenden Beispiel lagen die Datendateien der Non-CDB im Verzeichnis */oradata/noncdb*, wurden auf dem Zielserver in das Verzeichnis */oradata/tmp* kopiert und als PDB in */oradata/pdb1* installiert.

Sie können dieses zusätzliche Kopieren auf dem Zielserver auch auslassen und die Datendateien direkt an den Zielspeicherort der PDB kopieren. Statt der Klausel FILE_NAME_CONVERT benutzen Sie die Klausel NOCOPY:

```
SQL> CREATE PLUGGABLE DATABASE pdb1
USING '/home/oracle/ncdb.xml'
SOURCE_FILE_NAME_CONVERT=('/oradata/noncdb',
                          '/oradata/pdb1')
NOCOPY
STORAGE (MAXSIZE 100G MAX_SHARED_TEMP_SIZE 900M);
```

Wenn Sie die Fehlermeldung

```
ERROR at line 1:
ORA-27038: created file already exists
ORA-01119: error in creating database file
'/oradata/pdb1/temp01.dbf'
```

bekommen, benennen Sie die Temp-Datei der Non-CDB um und lassen Sie durch das CREATE-Kommando eine neue Temp-Datei anlegen.

In der STORAGE-Klausel des CREATE PLUGGABLE DATABASE-Kommandos können Sie die Limits für die neue PDB festlegen. Im Beispiel dürfen alle Tablespaces der neuen PDB zusammen 100GB (MAXSIZE) umfassen und die Sessions in der neuen PDB dürfen insgesamt maximal 900MB im Shared Temporary Tablespace der CDB in Anspruch nehmen.

8 Starten Sie das Skript *ORACLE_HOME/rdbms/admin/noncdb_to_pdb.sql*, bevor Sie die neue PDB öffnen:

```
$ sqlplus /nolog
SQL> connect sys/pwd@pdb_wdg as sysdba
SQL> sta ?/rdbms/admin/noncdb_to_pdb.sql
```

Öffnen Sie jetzt die PDB ganz normal (also READ WRITE-Modus):

```
SQL> SHUTDOWN IMMEDIATE
SQL> ALTER DATABASE OPEN;
```

9 Für die neue PDB steht noch kein Backup für ein späteres Recovery zur Verfügung. Es sollte also ein Backup angefertigt werden.

Klonen über Datenbank-Link

Führen Sie die folgenden Schritte durch:

1 Erstellen Sie einen Datenbank-Link von der CDB, in der die PDB erstellt werden soll, zur Non-CDB. Im folgenden wird von einem funktionierenden Datenbank-Link namens **‚db_link_to_noncdb'** ausgegangen. Beispiel:

```
SQL> CREATE DATABASE LINK db_link_to_noncdb
   CONNECT TO system IDENTIFIED BY welcome1
   USING '(DESCRIPTION =
           (ADDRESS =
              (PROTOCOL = TCP)
              (HOST = serv154)
```

```
      (PORT = 1521))
   (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = radul54)))'
```

2. Stellen Sie sicher, dass die Non-CDB geöffnet ist. Erstellen Sie die neue PDB durch Klonen (hier ist 'noncdb' der Name der zu klonenden Datenbank)

```
SQL> CREATE PLUGGABLE DATABASE pdb2 FROM noncdb@db_
link_to_noncdb
   PATH_PREFIX = '/oracle/oradata/pdb2'
   FILE_NAME_CONVERT = ('/oracle/oradata/pdb1/',
                        '/oracle/oradata/pdb2/');
```

3. Bevor Sie die neue PDB öffnen, melden Sie sich mit SYSDBA Privilegien an die neue PDB an und starten das Skript 'noncdb_to_pdb.sql'.

```
SQL> ALTER SESSION SET CONTAINER=pdb2;
SQL> @$ORACLE_HOME/rdbms/admin/noncdb_to_pdb.sql
```

Dieses Skript öffnet die neue PDB und führt einige Anpassungen durch. Anschließend wird die neue PDB wieder geschlossen.

4. Öffnen Sie die neue PDB

```
SQL> ALTER SESSION SET CONTAINER=cdb$root;
SQL> ALTER PLUGGABLE DATABASE pdb2 OPEN;
```

Full Database Import

Die Non-CDB wird inhaltlich in eine existierende PDB übertragen:

1. Dazu muss erst einmal eine neue PDB erstellt werden, wenn die CDB mit Oracle Managed Files konfiguriert ist, zum Beispiel mit

```
SQL> CREATE PLUGGABLE DATABASE pdb1
       ADMIN USER syspdb IDENTIFIED BY passwort
       DEFAULT TABLESPACE data;
```

oder sonst

```
SQL> CREATE PLUGGABLE DATABASE pdb1
      ADMIN USER syspdb IDENTIFIED BY passwort
      DEFAULT TABLESPACE data
         DATAFILE '/oracle/oradata/pdb1/data01.dbf'
         SIZE 200M;
```

2. Fügen Sie einen Eintrag in die Datei *tnsnames.ora* ein:

```
PDB1 =
   (DESCRIPTION =
```

```
      (ADDRESS = (PROTOCOL = TCP)(HOST = server.
      localdomain)
                 (PORT = 1521))
      (CONNECT_DATA =
         (SERVER = DEDICATED)
         (SERVICE_NAME = pdb1)
      )
   )
```

3. Öffnen Sie die Datenbank:

```
$ sqlplus sys@pdb1 as sysdba
Enter password:
SQL> alter database open;
```

4. Vergeben Sie Datenbankprivilegien an den neuen DBA-Benutzer (hier „syspdb“). Die Administration einer PDB sollte nicht mehr mit den Benutzern „SYS“ oder „SYSTEM“ durchgeführt werden! In diesem Beispiel wird der Einfachheit halber die bekannte Rolle DBA vergeben

```
SQL> grant dba to syspdb;
```

5. Exportieren Sie die Daten aus der alten Non-CDB als Full Database Export:

```
expdp admin@noncdb DIRECTORY=DPDIR DUMPFILE=NONCDB.dmp
FULL=Y
```

6 Erstellen Sie in der neuen PDB ein Directory für DATA PUMP:

```
SQL> create directory dpdir as '/home/oracle/dp';
```

7 Importieren Sie die Daten der Non-CDB mit DATA PUMP:

```
impdp syspdb@pdb1 DIRECTORY=DPDIR DUMPFILE=NONCDB.dmp
FULL=Y
```

3.3 LÖSCHEN VON PDBs

Sie löschen eine PDB mit dem Kommando DROP PLUGGABLE DATABASE. Dazu müssen Sie mit der CDB verbunden sein, also zum Beispiel:

```
$ sqlplus sys@cdb as sysdba
SQL> DROP PLUGGABLE DATABASE pdb1;
```

Dabei werden die Datendateien nicht gelöscht. Das Löschen der Datendateien erreichen Sie mit:

```
$ sqlplus sys@cdb as sysdba
SQL> DROP PLUGGABLE DATABASE pdb1 INCLUDING DATAFILES;
```

3.4 PLUG UND UNPLUG VON PDBs

Ausklinken („Unplug“) einer PDB

Sie können jederzeit eine PDB aus einer CDB ausklinken. Dabei wird eine XML-Datei mit Informationen über die PDB erstellt. Diese XML-Datei wird zusammen mit allen Datendateien der PDB benötigt, um diese PDB in eine andere CDB einzuklinken.

Ein „Unplug“, also das Ausklinken einer PDB aus einer CDB, können Sie mit dem Database Configuration Assistant (DBCA) durchführen. Dazu wählen Sie im Schritt 1 die Aktion *Manage Pluggable Databases* und dann im Schritt 2 *Unplug a Pluggable Database*.

Mittels SQL-Kommandos gehen Sie folgendermaßen vor:

1. Eine PDB kann nur dann ausgeklinkt werden, wenn sie mindestens einmal geöffnet war. Prüfen Sie dieses.

2. Fahren Sie die PDB herunter:

```
SQL> SHUTDOWN IMMEDIATE
```

3. Ein Ausklinken einer PDB wird in einer Datenbanksitzung in der CDB durchgeführt. Dazu müssen Sie mit dem Privileg SYSDBA oder SYSOPER angemeldet sein:

```
$ sqlplus sys@cdb as sysdba
```

4. Nutzen Sie das Kommando ALTER PLUGGABLE DATABASE wie folgt:

```
SQL> ALTER PLUGGABLE DATABASE pdb1
     UNPLUG INTO '/home/oracle/pdb1.xml';
```

Die XML-Datei beinhaltet die Informationen über die Datendateien. Diese Datendateien können Sie gemeinsam mit der XML-Datei nutzen, um diese PDB in einer anderen CDB einzuklinken.

5. Die ausgeklinkte PDB ist immer noch in der CDB in der View V$PDBS sichtbar, kann jedoch nicht mehr geöffnet werden. Löschen Sie die PDB also ohne ein Löschen der Datendateien:

```
SQL> DROP PLUGGABLE DATABASE pdb1;
```

Einklinken („Plug“) einer PDB

Das Einklinken einer PDB ist nicht mit dem Database Configuration Assistant (DBCA) möglich.

Das Einklinken mittels SQL-Kommando führen Sie mit dem Kommando CREATE PLUGGABLE DATABASE durch:

1. Kopieren Sie die Datendateien und die XML-Datei, die beim Unplug erstellt wurde, auf den Zielserver.

2. Melden Sie sich in der CDB an.

3. Prüfen Sie, ob die PDB kompatibel zur CDB ist mit:

```
SET SERVEROUTPUT ON
DECLARE
   compatible CONSTANT VARCHAR2(3) :=
      CASE DBMS_PDB.CHECK_PLUG_COMPATIBILITY(
         pdb_descr_file => '/home/oracle/pdb1.xml')
      WHEN TRUE THEN 'YES'
      ELSE 'NO'
END;
BEGIN
DBMS_OUTPUT.PUT_LINE(compatible);
END;
/
```

   Wenn die Ausgabe „*YES*“ ist, ist die PDB kompatibel zur CDB und kann eingeklinkt werden.

4. Klinken Sie die PDB mit dem Kommando CREATE PLUGGABLE DATABASE ein. Wenn die Speicherpfade der Datendateien gegenüber der Ursprungs-CDB gleich bleiben, reicht ein:

```
CREATE PLUGGABLE DATABASE pdb2 USING '/home/oracle/
pdb1.xml'
   NOCOPY
   TEMPFILE REUSE;
```

Wenn sich die Speicherpfade geändert haben, was in der Regel der Fall sein wird, benutzen Sie die Klausel SOURCE_FILE_NAME_CONVERT:

```
CREATE PLUGGABLE DATABASE pdb2 USING '/home/oracle/
pdb1.xml'
   NOCOPY
   SOURCE_FILE_NAME_CONVERT=('/u01/oradata/pdb1/',
                             '/u02/oradata/pdb1/)
   TEMPFILE REUSE;
```

Des Weiteren können Sie mit einer STORAGE-Klausel angeben, wie viel Storage die neue PDB verbrauchen darf, zum Beispiel:

```
CREATE PLUGGABLE DATABASE pdb2 USING '/home/oracle/
pdb1.xml'
   NOCOPY
   SOURCE_FILE_NAME_CONVERT=('/u01/oradata/pdb1/',
                             '/u02/oradata/pdb1/)
   STORAGE (MAXSIZE 100G MAX_SHARED_TEMP_SIZE 900M);
   TEMPFILE REUSE;
```

Im Beispiel dürfen alle Tablespaces der neuen PDB zusammen 100GB (MAXSIZE) umfassen und die Sessions in der neuen PDB dürfen insgesamt maximal 900MB im Shared Temporary Tablespace der CDB in Anspruch nehmen.

5 Prüfen Sie das Ergebnis in V$PDBS:

```
SQL> SELECT con_id,name,open_mode from V$PDBS;
    CON_ID NAME                  OPEN_MODE
---------- ------------------- ----------
         2 PDB$SEED            READ ONLY
         3 PDBRED              READ ONLY
         4 PDB2                MOUNTED
```

6 Die neue PDB ist im Status MOUNTED. Bereiten Sie die *tnsnames.ora*-Datei vor und öffnen Sie die Datenbank, indem Sie sich anmelden und ein ALTER DATABASE OPEN durchführen:

```
$ sqlplus sys@pdb2 as sysdba
SQL> ALTER DATABASE OPEN;
```

7 Führen Sie ein Backup für die neue PDB durch, da sie ohne neues Backup nicht „recovered“ werden kann.

4 Betriebszustände einer PDB

Eine PDB kann entweder geöffnet (OPEN) oder geschlossen (MOUNTED) sein. Die Zustände CLOSED und NOMOUNTED, wie sie von herkömmlichen Oracle-Datenbanken (Non-CDBs) bekannt sind, gibt es für PDBs nicht.

Nachdem eine PDB neu erstellt oder eingeklinkt wurde, ist sie im Status MOUNTED. Geöffnet werden kann sie in einer Datenbanksitzung in der PDB mit SYSDBA- oder SYSOPER-Privilegien mit:

```
SQL> STARTUP
```

oder

```
SQL> ALTER DATABASE OPEN [READONLY];
```

Ist eine PDB geöffnet, kann sie mit den bekannten SHUTDOWN-Kommandos heruntergefahren werden:

```
SQL> SHUTDOWN NORMAL
SQL> SHUTDOWN IMMEDIATE
SQL> SHUTDOWN TRANSACTIONAL
SQL> SHUTDOWN ABORT
```

Den aktuellen Zustand einer PDB können Sie wie folgt ermitteln:

4.1 PRÜFEN INNERHALB EINER PDB

Innerhalb einer PDB stehen Ihnen drei Views zur Verfügung:

View V$PDBS:

```
SQL> SELECT con_id,name,open_mode FROM V$PDBS;
```

```
   CON_ID    NAME                          OPEN_MODE
---------    ----------------------        ----------
       4     PDB4                          READ WRITE
```

Mögliche Werte sind MOUNTED, READ WRITE oder READ ONLY. Letztere Werte stehen für einen geöffneten Status.

View V$INSTANCE:

```
SQL> SELECT instance_name,status FROM V$INSTANCE;
INSTANCE_NAME   STATUS
-------------   ------------
cdb1            OPEN
```

Als Instance wird richtigerweise die CDB angegeben. Der Status bezieht sich aber auf die PDB. In V$INSTANCE können Sie im Status OPEN nicht sehen, ob die Datenbank READ ONLY oder READ WRITE geöffnet ist.

Nach einem

```
SQL> SHUTDOWN IMMEDIATE
```

auf die PDB sieht das Ergebnis so aus:

```
SQL> SELECT instance_name,status FROM V$INSTANCE;
INSTANCE_NAME   STATUS
-------------   ------------
cdb1            MOUNTED
```

View V$DATABASE:

Auch in V$DATABASE wird als Name die CDB angegeben, der OPEN_MODE bezieht sich aber auf die PDB:

```
SQL> SELECT name,open_mode FROM V$DATABASE;
NAME            OPEN_MODE
-------------   ------------
CDB1            READ WRITE
```

4.2 PRÜFEN INNERHALB DER CDB

View V$PDBS:

Die View V$PDBS zeigt in der CDB den Status aller PDBs an:

```
SQL> SELECT con_id,name,open_mode FROM V$PDBS;
    CON_ID    NAME                        OPEN_MODE
----------    ----------------------      ----------
         2    PDB$SEED                    READ ONLY
         3    PDBRED                      READ ONLY
         4    PDB4                        READ WRITE
```

View V$CONTAINERS:

Die View V$CONTAINERS zeigt zusätzlich den Status der CDB an:

```
SQL> SELECT con_id,name,open_mode FROM V$CONTAINERS;
```

```
CON_IDNAME                          OPEN_MODE
----------  ------------------------  ----------
         1  CDB$ROOT                  READ WRITE
         2  PDB$SEED                  READ ONLY
         3  PDBRED                    READ ONLY
         4  PDB4                      READ WRITE
```

4.3 ZUSTAND EINER PDB NACH NEUSTART EINER CDB

Nach einem Neustart einer CDB sind alle PDBs standardmäßig im Stadium MOUNT und müssen im Bedarfsfall geöffnet werden. Sie können für einzelne PDBs festlegen, dass diese nach einem Neustart einer CDB in den gleichen Betriebszustand, in dem sie vor dem Herunterfahren der CDB waren.

Dazu ändern Sie die eine einzelne Pluggable Datenbank mit

```
SQL> ALTER PLUGGABLE DATABASE pdb4 SAVE STATE;
```

Alle PDBs in einer Container Datenbank bearbeiten Sie entsprechend mit

```
SQL> ALTER PLUGGABLE DATABASE ALL SAVE STATE;
```

Sie können auch eine Liste von PDBs angeben:

```
SQL> ALTER PLUGGABLE DATABASE pdb1,pdb2 SAVE STATE;
```

Auch mit einem Ausschlußkriterium kann gearbeitet werden:

```
SQL> ALTER PLUGGABLE DATABASE ALL EXCEPT pdb1,pdb2
SAVE STATE;
```

Mit der Klausel DISCARD STATE stellen Sie den Default wieder her, also zum Beispiel mit

```
SQL> ALTER PLUGGABLE DATABASE apdb_a DISCARD STATE;
```

Im Falle einer Oracle RAC CDB können geben Sie mit der Klausel INSTANCES an, in welchen Instanzen die PDB betrieben werden soll:

```
SQL> ALTER PLUGGABLE DATABASE pdb4 SAVE STATE
INSTANCES=('inst1','inst3');
```

Eine Liste aller PDBs, die in einem Container mit dem Attribut SAVE STATE versehen sind, finden Sie in der Data Dictionary View DBA_PDB_SAVED_STATES:

```
SQL> SELECT con_id,con_name FROM dba_pdb_saved_states;
    CON_ID  CON_NAME
----------  --------
         4  PDB4
```

5 Zugriff auf eine Oracle Pluggable Datenbank

Schon seit den Zeiten von Oracle 8i gilt: Der Zugriff auf eine Oracle-Datenbank sollte immer über einen Service geschehen und nicht mehr über die Oracle Systems ID (SID) beziehungsweise den Instanznamen. Genau dies stimmt auch für die Pluggable Databases, denn es gibt ja für die einzelnen PDBs keine eigenen Instanzen mehr: Die CDB ist die Instanz. Deswegen kann man sich an die PDBs nicht mehr einfach durch Setzen der ORACLE_SID im Environment und anschließendes CONNECT / AS SYSDBA verbinden, sondern lässt sich immer über den Listener und den dort registrierten Service verbinden.

Dies kann zum Beispiel über die mit Oracle 9i eingeführten Easy-Connect-Methode geschehen (hierfür ist keine *tnsnames.ora* notwendig):

```
$ sqlplus <USER>/<PWD>@<HOST>:<LSN_PORT>/SERVICE
```

Hierbei ist der Port optional, wenn der Listener den Default Port verwendet:

```
$ sqlplus system/oracle@localhost:1521/PDB
```

Mit derselben Syntax und dem Servicenamen der CDB (= Instanznamen der Datenbank) verbindet man sich mit der CDB.

Alternativ kann natürlich auch die tnsnames.ora mit den entsprechenden Alias- und Serviceeinträgen gepflegt werden:

```
PDB1 =
   (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)
      (PORT = 1521))
      (CONNECT_DATA =
         (SERVER = DEDICATED)
         (SERVICE_NAME = pdb1)
      )
   )
```

Möchte oder muss man an sich an die PDB anmelden, obwohl kein Listener läuft und somit auf den Service nicht zugegriffen werden kann (da ein Connect über den Service somit nicht möglich ist), so gibt es noch die Alternative, sich nach der Verbindung mit der CDB (zum Beispiel über eine Bequeath-Verbindung durch das Setzen der ORACLE_SID) mit einem ALTER SESSION mit der PDB zu verbinden:

```
$ set ORACLE_SID=CDB
$ sqlplus system/oracle
SQL> ALTER SESSION SET CONTAINER = PDB1;
```

Die Namen der Container, mit denen Sie sich auf diese Weise verbinden können, ermitteln Sie in der CDB mit:

```
SQL> SELECT con_id,name,open_mode FROM V$CONTAINERS;
   CON_ID   NAME                          OPEN_MODE
---------   ------------------------      -----------
       1    CDB$ROOT                      READ WRITE
       2    PDB$SEED                      READ ONLY
       3    PDB1                          READ ONLY
       4    PDB2                          READ WRITE
```

Woran erkennt man nun, dass man mit der PDB oder der CDB verbunden ist? Denn eine Abfrage auf V$DATABASE oder V$INSTANCE gibt verständlicherweise nur die CDB aus. Dies kann man sich mit einem SHOW CON_NAME beziehungsweise SHOW CON_ID anzeigen lassen:

```
SQL> show con_name;
CON_NAME
------------------
PDB1
```

Selbstverständlich lassen sich in einer PDB noch zusätzliche Services anlegen. Dies ist zum Beispiel notwendig, wenn man zusätzliche Features wie Load Balancing, TAF oder das neue 12c Application Continuity verwenden möchte. Dies geschieht bei der Verwendung der Grid-Infrastruktur (egal ob für Standalone- oder Cluster-Systeme) über SRVCTL:

```
$ srvctl add service -s service -d database -p PDB1
...
```

SRVCTL wurde mit dem Parameter „–p“ erweitert, damit der Service weiß, mit welcher PDB der Client sich verbindet, wenn er den Service verwendet. Für Single-Instance-Datenbanken ohne Grid-Infrastruktur verwendet man DBMS_SERVICE. Hier definiert die PDB, in der der Service angelegt wird, an welche PDB der Client beim Verbinden auf den Service weitergeleitet wird.

```
SQL> begin
   dbms_service.create_service (
      SERVICE_NAME => 'PDBSERV'
     ,NETWORK_NAME => 'PDBSERV');
   dbms_service.start_service('PDBSERV')
end;
/
SQL> connect system/oracle@localhost/PDBSERV
SQL> show con_name
CON_NAME
------------------
PDB1
```

Zu beachten ist, dass Servicenamen innerhalb einer CDB eindeutig sein müssen, sodass ein Service mit einem bestimmten Namen nur in einer PDB existieren kann.

6 Instanzparameter

Bei der konventionellen Konsolidierung von Datenbankanwendungen (Schema-Konsolidierung) in eine einzige Datenbank gibt es für alle konsolidierten Datenbankanwendungen nur einen Satz an Instanzparametern. Dieses führt in vielen Fällen zu Einschränkungen.

Diese Einschränkungen werden bei dem Konzept von Pluggable Databases reduziert. Denn neben den global für die CDB einzustellenden Instanzparametern gibt es eigene Instanzparameter, die pro PDB eingestellt werden.

Sie ermitteln die globalen CDB-weiten Instanzparameter mit der folgenden Abfrage:

```
SELECT NAME FROM V$PARAMETER WHERE ISPDB_
MODIFIABLE='FALSE' ORDER BY NAME;
```

Zu den nach dieser Abfrage ausgegebenen Parametern zählen sämtliche Parameter, die aufgrund der Architektur von CDBs und PDBs prinzipbedingt nur „global“ eingestellt werden können. So ist es nicht verwunderlich, dass dort zum Beispiel Parameter für Memory-Einstellungen und Undo-Management auftauchen.

Die Parameter für die Einstellung der Hintergrund-Prozesse (zum Beispiel PROCESSES) werden auch global

auf der Ebene der CDB eingestellt. Eine Verteilung von CPU-Ressourcen auf die einzelnen PDBs kann mit dem Database Resource Manager vorgenommen werden, wie im weiteren Verlauf näher erläutert wird.

Sie ermitteln die lokalen PDB-Instanzparameter mit der folgenden Abfrage:

```
SELECT NAME FROM V$PARAMETER WHERE ISPDB_
MODIFIABLE='TRUE' ORDER BY NAME;
...
nls_sort
nls_territory
nls_time_format
nls_time_tz_format
nls_timestamp_format
nls_timestamp_tz_format
object_cache_max_size_percent
object_cache_optimal_size
olap_page_pool_size
open_cursors
optimizer_adaptive_reporting_only
optimizer_capture_sql_plan_baselines
optimizer_dynamic_sampling
optimizer_features_enable
optimizer_index_caching
optimizer_index_cost_adj
```

```
optimizer_mode
optimizer_secure_view_merging
optimizer_use_invisible_indexes
optimizer_use_pending_statistics
optimizer_use_sql_plan_baselines
... (Ausgabe gekürzt)
```

Beim Betrachten der Liste fallen einige Parameter besonders ins Auge: Auch wenn der Datenbankzeichensatz für die gesamte CDB festgelegt ist, lassen sich dennoch viele NLS_...-Parameter (zum Beispiel Sortierreihenfolge und Zeitzonen) individuell pro PDB setzen. Auch Parameter, die den Optimizer beeinflussen (wie zum Beispiel OPTIMIZER_INDEX_COST_ADJ), sowie für das Cursor-Handling (zum Beispiel CURSOR_SHARING) sind in der Liste zu finden.

Wenn ein Parameter nicht spezifisch für eine PDB gesetzt wird, gilt der Wert des Parameters aus der CDB in der sie läuft.

Sollten für eine einzelne PDB von diesem Default abweichende Parameter gesetzt worden sein, werden diese bei einem „Unplug“ auch in den Metadaten vermerkt und beim „Plug“ in eine andere CDB entsprechend wiederhergestellt.

Obwohl es Parameter sowohl auf CDB- als auch auf PDB-Ebene gibt, wird nur eine Parameterdatei verwendet. Parameter, die in einer PDB mittels ALTER SYSTEM ...

SCOPE=BOTH beziehungsweise SCOPE=SPFILE abweichend vom Wert der CDB gesetzt werden, sind Bestandteil der Metadaten einer PDB und werden demzufolge auch beim Einklinken in eine andere CDB berücksichtigt.

7 Data Dictionary

Das **Oracle Data Dictionary** wird inhaltlich um eine View-Ebene erweitert: die CDB-Ebene. Jede View, die als USER/ALL/DBA-View zur Verfügung steht, wird dabei um die Spalte CON_ID, also der Container-ID erweitert. In der CDB steht damit eine Sicht über alle PDBs hinweg zur Verfügung, zum Beispiel:

```
CDB> select username,con_id from cdb_users order by username;

USERNAME                              CON_ID
------------------------------    ----------
ADMIN                                      3
ANONYMOUS                                  2
ANONYMOUS                                  3
ANONYMOUS                                  1
APEX_040200                                3
APEX_040200                                2
APEX_040200                                1
:
```

Die Container_ID=1 steht immer für die CDB selbst, die Zuweisung der restlichen Container-IDs kann mit

```
SQL> select name,con_id from v$pdbs;
NAME                                     CON_ID
------------------------------     ----------
PDB$SEED                                    2
PDB1                                        3
```

ermittelt oder in einer JOIN-Abfrage eingebunden werden.

Die CDB-Views sind auch in jeder PDB nutzbar, zeigen jedoch nur die jeweils lokalen Daten an:

```
PDB> select username,con_id from cdb_users order by
username;

USERNAME                                 CON_ID
------------------------------     ----------
ADMIN                                       3
ANONYMOUS                                   3
APEX_040200                                 3
```

Grundsätzlich gibt das Data Dictionary innerhalb einer PDB nur die lokale Sicht wieder. Es werden innerhalb einer PDB im Data Dictionary also nur Daten angezeigt, die für die jeweilige PDB relevant sind.

Natürlich gibt es neue Views, die Informationen speziell für den Bereich Pluggable Databases anzeigen:

V$PDBS

In der CDB zeigt diese View alle PDBs an, die gerade in einer CDB betrieben werden.

In einer PDB zeigt diese View Informationen über die eigene PDB an.

```
SQL> desc v$pdbs
Name                  Null?    Type
--------------------  -------- ----------------------
CON_ID                         NUMBER
DBID                           NUMBER
CON_UID                        NUMBER
GUID                           RAW(16)
NAME                           VARCHAR2(30)
OPEN_MODE                      VARCHAR2(10)
RESTRICTED                     VARCHAR2(3)
OPEN_TIME                      TIMESTAMP(3)
CREATE_SCN                     NUMBER
TOTAL_SIZE                     NUMBER
```

CDB/DBA_PDBS

Diese View ist nur in der CDB verfügbar und zeigt eine Teilmenge von V$PDBS an.

```
SQL> desc dba_pdbs
Name                  Null?       Type
---------------       --------    -----------------
PDB_ID                NOT NULL    NUMBER
PDB_NAME              NOT NULL    VARCHAR2(128)
DBID                  NOT NULL    NUMBER
CON_UID               NOT NULL    NUMBER
GUID                              RAW(16)
STATUS                            VARCHAR2(13)
CREATION_SCN                      NUMBER
```

8 Benutzerverwaltung

In der neuen Architektur werden zwei Typen von Datenbankbenutzern unterschieden: Common User und Local User.

Local User sind PDB-lokale Datenbankbenutzer, die also nur in der jeweiligen PDB existieren und arbeiten dürfen. Diese Benutzer entsprechen den bislang bekannten Datenbankbenutzern.

Common User sind CDB-globale Datenbankbenutzer, die sowohl in der CDB als auch in allen PDBs definiert sind. Dabei können Common User in den verschiedenen PDBs unterschiedliche Privilegien bekommen.

8.1 LOKALE DATENBANKBENUTZER (LOCAL USER)

Lokale Datenbankbenutzer werden, wie von alten Versionen gewohnt, innerhalb der PDB mit dem CREATE USER-Kommando erstellt und mit dem GRANT-Kommando mit Privilegien versorgt. Die einzige Neuerung für neue lokale Datenbankbenutzer ist, dass der Name des Benutzers nicht mit den Zeichen „C##“ oder „c##“ beginnen darf.

Auch die Nutzung der Datenbankbenutzer „SYS“ und „SYSTEM“ als Administrationsbenutzer ändert sich, da diese in einer CDB grundsätzlich Common User sind und in allen PDBs das gleiche Passwort besitzen.

8.2 GLOBALE DATENBANKBENUTZER (COMMON USER)

Neue globale Datenbankbenutzer werden in einer Datenbanksitzung mit der CDB mit dem Kommando CREATE USER erstellt. Der Name eines benutzerdefinierten globalen Datenbankbenutzers beginnt mit „c##“ oder „C##“ und diese Benutzer werden sowohl in der CDB als auch in allen PDBs erstellt. Privilegien können getrennt sowohl für die CDB als auch für jede PDB mit dem bekannten GRANT-Kommando vergeben werden. Das Passwort eines globalen Datenbankbenutzers ist aber sowohl in der CDB als auch in allen PDBs gleich.

Auch wenn sich ein globaler Datenbankbenutzer an jede PDB (das Privileg CREATE SESSION wird hier vorausgesetzt) anmelden und die ihm dort gegebenen Privilegien nutzen kann, ist es möglich, die Sicht auf die PDBs in einer CDB-Sitzung einzuschränken. Dazu wird für den globalen Datenbankbenutzer auf CDB-Ebene eine Liste von „Default Access Container“ definiert.

Ein Beispiel zeigt dies deutlich: Ein Zugriff auf V$PDBS zeigt ja die PDBs, die in der CDB laufen. Als Benutzer „SYS“ sieht man alle PDBs:

```
SQL> CONNECT sys/passwort@cdb AS sysdba
SQL> SELECT con_id,name FROM v$pdbs;

CON_ID     NAME
--------   ------------------------------
       2   PDB$SEED
       3   RADU2_1
       4   RADU2_2
       5   PDB_WDG
```

Es wird ein neuer globaler Datenbankbenutzer erstellt mit:

```
SQL> CONNECT sys/passwort@cdb AS sysdba
SQL> CREATE USER c##wdgneu IDENTIFIED BY passwort;
SQL> GRANT create session, select_catalog_role TO
c##wdgneu;
```

Dieser neue globale Datenbankbenutzer sieht auf CDB-Ebene zunächst keine PDBs:

```
SQL> CONNECT c##wdgneu/passwort@cdb
SQL> SELECT con_id,name FROM v$pdbs;

no rows selected
```

Jetzt wird eine PDB als „Default Access Container" definiert:

```
SQL> CONNECT sys/passwort@cdb AS sysdba
SQL> ALTER USER c##wdgneu
     ADD CONTAINER_DATA = ( "RADU2_1" ) CONTAINER =
     CURRENT;
```

Als Ergebnis sieht der Benutzer „c##wdgneu" jetzt diese PDB im Data Dictionary der CDB:

```
SQL> CONNECT c##wdgneu/passwort@cdb
SQL> SELECT con_id,name FROM v$pdbs;

  CON_ID  NAME
--------  ------------------------------
       3  RADU2_1
```

Es gibt vordefinierte globale Datenbankbenutzer. Die wichtigsten sind „SYS" und „SYSTEM". Dabei ist zu beachten, dass das Passwort dieser beiden bekannten Benutzer über alle PDBs und der CDB hinweg gleich ist. Aus diesem

Grund ist die Nutzung von „SYS“ und „SYSTEM“ einem globalen Administrator für die gesamte CDB und aller PDBs vorbehalten. Sollen pro PDB separate Administratoren arbeiten, so müssen dafür separate lokale Datenbankbenutzer mit entsprechenden DBA-Privilegien angelegt werden.

8.3 INFORMATIONEN AUS DEM DATA DICTIONARY

Informationen über Datenbankbenutzer, Rollen und Privilegien finden Sie im Data Dictionary wie gewohnt unter:

- USER/ALL/DBA/CDB_USERS
- USER/ALL/DBA/CDB _ROLES
- USER/ALL/DBA/CDB _SYS_PRIVS
- USER/ALL/DBA/CDB _ROLE_PRIVS

9 Verwalten von Ressourcen

Der seit Oracle 8i eingeführte Database Resource Manager kann in einer klassischen Oracle-Datenbank, jetzt Non-CDB genannt, verschiedene Ressourcen (zum Beispiel CPU, Parallelität usw.) innerhalb der jeweiligen Datenbank an verschiedene Konsumentengruppen zusichern. Ein Resource Management über mehrere auf dem gleichen Server befindliche Oracle-Datenbanken hinweg ist damit aber nicht möglich.

Mit dem neuen Konzept der Pluggable Databases wird diese Herausforderung elegant gelöst, denn alle PDBs innerhalb einer CDB benutzen gemeinsam die zur Verfügung stehenden Ressourcen, sodass innerhalb der CDB eine Ressourcenverwaltung für alle PDBs durchgeführt werden kann.

Konsequenterweise wird daher der **Database Resource Manager** entsprechend erweitert.

In der CDB werden dazu sogenannte **CDB Resource Plans** erstellt. Diese legen mittels Direktiven für einzelne PDBs fest, welchen Anteil der insgesamt zur Verfügung stehenden Ressourcen diese überhaupt nutzen können. Konkret betrachtet werden auf dieser obersten Ebene folgende Elemente:

- „CPU“

Hier wird die nutzbare CPU-Leistung der Sessions einer PDB durch den Resource Manager gedrosselt.

- „Parallel Execution Servers“

Hier geht es darum, wie viele der insgesamt für die CDB vorhandenen „Parallelen Ausführungsprozesse“ anteilig durch jede PDB nutzbar sind.

Im ersten Schritt werden diese Elemente in Form ganzzahliger „Shares“ zugewiesen. Die Summe der Werte dieser Anteile repräsentiert dann die maximal zur Verfügung stehende Leistung der Datenbank.

Die durch Shares zugewiesenen Ressourcen stehen der entsprechenden PDB dann auf jeden Fall zur Verfügung – sie kann aber auch darüber hinaus Ressourcen verbrauchen, sofern die anderen in der CDB laufenden PDBs ihre jeweiligen Anteile nicht „ausreizen“ und somit noch Gesamtkapazität frei ist.

Im Unterschied dazu gibt es als weiteren, optionalen Schritt die Möglichkeit, feste Ressourcen-Limits als Prozentwerte festzulegen. Diese stellen dann eine absolute Obergrenze dar, die auch dann eingehalten wird, falls noch Ressourcen frei wären.

Ein Beispiel macht das Prinzip deutlich:

Wenn die PDBs A und B jeweils zwei Shares bekommen, die PDB C aber nur einen Share, so stehen den ersten beiden also

jeweils 2/5 und letzterer nur 1/5 der Leistung der CDB zur Verfügung. Bei einem zusätzlich festgelegten Ressourcen-Limit von 60% für PDB A und B könnten diese also zum Beispiel zwischen 40% (entsprechend 2/5) und 60% der CPU-Leistung verbrauchen.

Für die Anzahl der parallelen Ausführungsprozesse gilt sinngemäß das Gleiche: In diesem Beispiel würden demzufolge Parallel Queries von PDB A und B doppelt so oft „an die Reihe kommen" wie die von PDB C.

Tabellarisch dargestellt sieht dieses Beispiel so aus:

PDB	Shares	Prozentualer Anteil	Limit
PDB A	*2*	*40%*	*60%*
PDB B	*2*	*40%*	*60%*
PDB C	*1*	*20%*	
Summe Shares	*5*	*100%*	

Die Erstellung der Ressourcen-Pläne erfolgt mittels des PL/SQL-Packages DBMS_RESOURCE_MANAGER. Ein entsprechender PL/SQL-Block zur Erstellung der Plan-Direktiven des oben genannten Beispiels würde wie folgt aussehen:

```
BEGIN
   DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
      plan                    => 'mein_cdb_plan',
      pluggable_database      => 'pdb_a',
      shares                  => 2,
```

```
      utilization_limit        => 60,
      parallel_server_limit    => 60);
END;

BEGIN
   DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
      plan                     => 'mein_cdb_plan',
      pluggable_database       => 'pdb_b',
      shares                   => 2,
      utilization_limit        => 60,
      parallel_server_limit    => 60);
END;

BEGIN
   DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
      plan                     => 'mein_cdb_plan',
      pluggable_database       => 'pdb_c',
      shares                   => 1,
      utilization_limit        => NULL,
      parallel_server_limit    => NULL);
END;
```

Innerhalb einer PDB können die ihr auf CDB-Ebene zugewiesenen Ressourcen dann weiter an die Konsumentengruppen zugewiesen werden. Im obigen Beispiel bedeutet eine Zusicherung von 20% CPU an eine Konsumentengruppe der PDB A in Wahrheit 20% von 40% der Gesamtleistung des Servers, also 8%.

10 Backup & Recovery

Das Backup und Recovery einer kompletten CDB inklusive aller PDBs unterscheidet sich nicht von dem einer herkömmlichen Datenbank. Unterschiede gibt es, wenn man gezielt ein Backup beziehungsweise Recovery einer einzelnen PDB durchführen möchte.

Empfohlen wird die Sicherung einer gesamten CDB und dabei unterscheidet sich die Syntax nicht von der bisherigen RMAN-Syntax eines Backups einer Non-CDB.

Eine wichtige Ausnahme gibt es beim Einklinken („Plug-in“) einer neuen PDB, da diese ohne neues Backup der PDB nicht wiederhergestellt werden kann. Deswegen sollte nach einem „Plug-in“ einer PDB diese separat gesichert oder eine komplette Sicherung der CDB erstellt werden. Ein interessanter Aspekt hierbei ist, dass ein Backup der einzelnen PDBs und der ROOT-PDB dem Backup der gesamten CDB gleichwertig ist.

Der für ein Recovery notwendige ARCHIVELOG-Modus wird auf CDB-Ebene eingeschaltet. Das Setzen der ARCHIVE-Parameter (LOG_ARCHIVE_DEST_n, RECOVERY_FILE_DEST) ist also nur auf CDB-Ebene möglich.

10.1 BACKUP EINER PDB

Das Backup einer PDB kann mit dem Privileg SYSBACKUP (ist in SYSDBA enthalten) der PDB erstellt werden. In diesem Falle unterscheidet sich das BACKUP-Kommando nicht von einem normalen RMAN-Backup. Allerdings werden nur die Datendateien der PDB Tablespaces gesichert:

```
$ rman target sys/oracle@localhost/pdb
Recovery Manager: Release 12.1.0.1.0 - Production on
Tue Jan 22 05:59:10 2013
Copyright (c) 1982, 2012, Oracle and/or its affiliates.
All rights reserved.
connected to target database: CDB1 (DBID=771167504)
RMAN> backup as compressed backupset database;
Starting backup at 22-JAN-13
using target database control file instead of recovery
catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=255 device type=DISK
channel ORA_DISK_1: starting compressed full datafile
backup set
channel ORA_DISK_1: specifying datafile(s) in backup
set
input datafile file number=00009 name=/u01/oradata/
cdb1/pdb/sysaux01.dbf
input datafile file number=00008 name=/u01/oradata/
```

```
cdb1/pdb/system01.dbf
input datafile file number=00010 name=/u01/oradata/
cdb1/pdb/pdb_users01.dbf
channel ORA_DISK_1: starting piece 1 at 22-JAN-13
channel ORA_DISK_1: finished piece 1 at 22-JAN-13
piece handle=/u01/oradata/fra/CDB1/
backupset/2013_01_22/o1_mf_nnndf_
TAG20130122T055923_8hx6svb7_.bkp tag=TAG20130122T055923
comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time:
00:00:35
Finished backup at 22-JAN-13
```

Dasselbe Backup könnte auch der CDB-Administrator mit dem BACKUP PLUGGABLE DATABASE-Befehl durchführen.

```
$ rman target sys/oracle@localhost/cdb
RMAN> backup as compressed backupset pluggable
database pdb
```

Allerdings können nur auf CDB-Ebene die archivierten Redologs gesichert werden. Wird dieses auf PDB-Ebene versucht, quittiert RMAN dieses mit folgender Fehlermeldung:

```
$ rman target sys/oracle@localhost/pdb
RMAN> backup archivelog all;
Starting backup at 22-JAN-13
using channel ORA_DISK_1
specification does not match any archived log in the
```

```
repository
backup cancelled because there are no files to backup

Finished backup at 22-JAN-13
```

Eine Besonderheit ist noch das Backup der Container-Datenbank selbst (ROOT). Da dieses Backup keine Anwendungsdaten beinhaltet, gibt es die Empfehlung die ROOT-Datenbank täglich mit einem Full-Backup zu sichern. Dies geschieht mit dem Parameter *database root*:

```
$ rman target sys/oracle@localhost/cdb
RMAN> backup as compressed backupset database root
Starting backup at 22-JAN-13
using channel ORA_DISK_1
channel ORA_DISK_1: starting compressed full datafile
backup set
channel ORA_DISK_1: specifying datafile(s) in backup
set
input datafile file number=00003 name=/u01/oradata/
cdb1/sysaux01.dbf
input datafile file number=00001 name=/u01/oradata/
cdb1/system01.dbf
input datafile file number=00004 name=/u01/oradata/
cdb1/undotbs01.dbf
input datafile file number=00006 name=/u01/oradata/
cdb1/users01.dbf
channel ORA_DISK_1: starting piece 1 at 22-JAN-13
```

```
channel ORA_DISK_1: finished piece 1 at 22-JAN-13
piece handle=/u01/oradata/fra/CDB1/
backupset/2013_01_22/o1_mf_nnndf_
TAG20130122T061502_8hx7q76d_.bkp tag=TAG20130122T061502
comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time:
00:01:05

Finished backup at 22-JAN-13
```

10.2 RESTORE EINER PDB

Das Restore der kompletten Datenbank (CDB plus alle PDBs) unterscheidet sich nicht von dem Restore herkömmlicher Datenbanken. Auf CDB-Ebene kann die komplette Datenbank mit allen PDBs wiederhergestellt werden. Dies dürfte aber in den meisten Fällen selten notwendig sein, ein Restore einer einzelnen PDB ist wahrscheinlicher und oft sinnvoller.

Hier liegt dann auch wieder der Unterschied in der Handhabung, wenn nur einzelne PDBs wiederhergestellt werden sollen. Bevor der DBA eine PDB wiederherstellt, empfiehlt es sich, das Backup zu verifizieren:

```
$ rman target sys/oracle@localhost/pdb
RMAN> VALIDATE DATABASE;
RMAN> RESTORE DATABASE VALIDATE;
```

Auf CDB-Ebene sieht die Syntax folgendermaßen aus:

```
$ rman target sys/oracle@localhost/cdb
RMAN> VALIDATE PLUGGABLE DATABASE pdb;
RMAN> RESTORE PLUGGABLE DATABASE pdb VALIDATE;
```

Zum Wiederherstellen einer einzelnen PDB wird diese zuerst geschlossen (in den MOUNT-Status heruntergefahren) und dann zurückgesichert. Fehlende beziehungsweise korrupte Datendateien sollten auf *„offline“* gesetzt werden, da sich sonst die Datenbank nicht schließen lässt:

```
$ rman target sys/oracle@localhost/pdb
RMAN> ALTER PLUGGABLE DATABASE CLOSE;
RMAN> ALTER DATABASE DATAFILE 9 OFFLINE;
RMAN> RESTORE DATABASE;
RMAN> RECOVER DATABASE;
RMAN> ALTER DATBASE DATAFILE 9 ONLINE;
RMAN> ALTER PLUGGABLE DATABASE OPEN;
```

Hier das Prozedere auf CDB-Ebene:

```
$ rman target sys/oracle@localhost/cdb
RMAN> ALTER PLUGGABLE DATABASE pdb CLOSE;
RMAN> ALTER PLUGGABLE DATABASE DATAFILE 9 OFFLINE;
RMAN> RESTORE PLUGGABLE DATABASE pdb;
RMAN> RECOVER PLUGGABLE DATABASE pdb;
RMAN> ALTER DATBASE DATAFILE 9 ONLINE;
```

```
RMAN> ALTER PLUGGABLE DATABASE pdb OPEN;
```

Eine Besonderheit gibt es noch beim Recovery der ROOT-Datenbank, da dazu die gesamte CDB heruntergefahren werden muss:

```
$ rman target sys/oracle@localhost/cdb
RMAN> shutdown immediate;
RMAN> startup mount;
RMAN> RESTORE DATABASE root;
RMAN> RECOVER DATABASE root;
RMAN> ALTER DATBASE OPEN;
RMAN> ALTER DATABASE ALL OPEN;
```

Allerdings gibt es hier die Empfehlung nach dem Wiederherstellen der ROOT-Datenbank alle PDBs ebenfalls zu restoren und zu recovern. Da dies in dieser Reihenfolge aber langsamer ist als ein komplettes Restore und Recovery, ist man mit RESTORE DATABASE und RECOVER DATABASE schneller.

Sollte die ROOT-Datenbank korrupt sein, empfiehlt sich auch der Einsatz des Data Recovery Advisors zur Findung der besten Recovery-Strategie.

Ein Point-in-Time Recovery einzelner PDBs und einzelner Tablespaces wird ebenfalls für den PDB-Datenbankadministrator unterstützt und ähnelt in der Syntax den oben genannten Beispielen.

Durch ein Point-in-Time Recovery einzelner PDBs kann es zu unterschiedlichen Inkarnationen der PDBs in der CDB kommen.

Innerhalb einer PDB kann die aktuelle Inkarnation über folgendes Statement abgefragt werden:

```
$ sqlplus system/oracle@localhost/pdb
SQL> show con_id

CON_ID
------------------
3
SQL> select PDB_INCARNATION# from v$pdb_incarnation
where STATUS = 'CURRENT' and CON_ID = 3;

PDB_INCARNATION#
------------------
               0
```

Auf CDB-Ebene kann eine Übersicht über die Inkarnation aller PDBs ermittelt werden mit:

```
SQL> select PDB_INCARNATION#,con_id from v$pdb_
incarnation where status='CURRENT';

PDB_INCARNATION#         CON_ID
------------------    ----------
               0              1
               0              2
               0              3
               0              4
```

Selbstverständlich kann der PDB-Administrator auch ein Block Level Recovery durchführen. Die Syntax unterscheidet sich dabei nicht von der einer Non-CDB. Ein PDB-Administrator kann dieses Recovery natürlich nur für die eigene PDB durchführen, sonst gibt es den folgenden Fehler:

```
$ rman target sys/oracle@localhost/pdb
RMAN> recover datafile 3 block 13;
Starting recover at 22-JAN-13
using channel ORA_DISK_1
RMAN-00571:     ===========================================
RMAN-00569:     ====== ERROR MESSAGE STACK FOLLOWS ====
RMAN-00571:     ===========================================
RMAN-03002:failure of recover command at 01/22/2013
08:25:52
RMAN-20201:datafile not found in the recovery catalog
RMAN-06010:error while looking up datafile: 3
```

11 Oracle Data Guard

Oracle Data Guard, die Standby-Datenbank-Lösung von Oracle, basiert auf den Recovery-Mechanismen der Datenbank. Hierzu ist die Standby-Seite im ständigen Recovery und bietet somit den optimalen Desaster-Schutz für die Oracle-Datenbank. Für dieses Recovery verwendet auch die Standby-Datenbank die Informationen aus den Redolog-Dateien. Da bei dem neuen Konzept der Pluggable Database die Redolog-Dateien gemeinsam für alle PDBs auf CDB-Ebene geführt werden, ergibt sich daraus, dass auch die Standby-Datenbank immer auf der kompletten CDB basiert.

Das heißt, dass sich im herkömmlichen Betrieb eine Data Guard-Umgebung von einer CDB (inklusive aller PDBs) nicht unterscheidet von einer Data Guard-Umgebung einer Non-CDB. Der komplette Aufbau einer Standby basiert auf der CDB, genauso wie die Verwaltung, Wartung, Switch- und Failover-Mechanismen der Data Guard-Umgebung.

Damit hat eine Data Guard-Umgebung auf CDB-Ebene einen verringerten Verwaltungsaufwand im Vergleich zu mehreren Standby-Umgebungen ohne die Verwendung der Pluggable Database-Technologie. Gleichzeitig stehen alle erweiterten Funktionen einer Data Guard-Umgebung der CDB zur Verfügung, wie zum Beispiel das automatische Block Recovery von Active Data Guard oder das Umwandeln in eine Standby-Snapshot-Datenbank.

Allerdings sollte man darauf achten, dass bei der Verwendung von SQL*Plus zur Verwaltung einer Standby-Datenbank dieses auf CDB-Ebene passieren muss. Ein ALTER DATABASE SWITCHOVER TO ... funktioniert logischerweise nicht auf PDB-Ebene.

Der größte Unterschied in der Handhabung besteht aber sicherlich beim Anlegen, Einklinken und Ausklinken einer PDB in eine bestehende CDB, für die eine Standby-Datenbank existiert. Der Befehl des Anlegens, Einklinkens und Ausklinkens wird zwar auch auf der Standby-Seite ausgeführt, allerdings müssen die Datendateien hierfür ebenfalls auf der Standby-Seite schon zur Verfügung stehen:

- Beim Anlegen einer PDB auf Basis einer XML-Datei muss die XML-Datei ebenfalls der Standby-Seite zur Verfügung stehen.
- Im Falle des Anlegens einer PDB auf Basis einer bestehenden PDB müssen die zu klonenden PDB-Dateien vorher auf der Standby-Seite vorhanden sein. Diese dürfen aber durchaus an anderer Stelle stehen, sofern dieses im DB_FILE_NAME_CONVERT-Parameter hinterlegt ist.

- Das gleiche gilt für das Einklinken einer PDB: Auch hier müssen die Datendateien vorher auf die Standby-Seite transferiert worden sein. Auch diese unterliegen selbstverständlich den Änderungen durch den DB_FILE_NAME_CONVERT-Parameter.

Wenn sie eine neue PDB in der Primär-CDB erstellen, die Voraussetzungen auf der Standby-Seite aber nicht erfüllt sind, führt dieses dazu, dass dort die Redo-Log Informationen nicht verarbeitet werden können. Auf der Standby-Datenbank wird dadurch der Recovery Prozess abgebrochen und das Recovery kann erst wieder aufgenommen werden, wenn die fehlenden Datendateien auch auf der Standby Seite zur Verfügung stehen.

Da aber gerade die neue Technology der Pluggable Datenbanken es erlaubt schnell Datenbanken zu klonen z.B. für Testzwecke, gibt es in CDB Umgebungen nun vermehrt auch die Anforderungen bestimmte PDBs einer CDB nicht auf der Standby Seite ebenfalls zu benötigen. Daher wurde mit 12.1.0.2 die Möglichkeit geschaffen einzelne PDBs vom Recovery auf der Standby Seite auszuschließen und somit auch beim Anlegen einer PDB nicht auf die Datendateien auf der Standby Seite angewiesen zu sein.

Eine einzelne PDB kann vom Recovery auf der Standby Seite ausgenommen werden, indem man auf der Standby Seite mit dem Befehl

```
SQL> ALTER PLUGGABLE DATABASE <name> DISABLE RECOVERY;
```

das Recovery der entsprechenden PDBs ausschließt. Durch diesen Befehl wird die PDB auf der Standby Seite geschlossen und alle Datendateien, die zu der entsprechenden PDB gehören, auf OFFLINE gesetzt. Sollten auf der Primärseite weitere Datenfiles erstellt werden, so werden diese auf der Standby Seite nicht benötigt, sind aber im Katalog der CDB enthalten und werden auf der Standby Seite in der View v$datafile als „UNNAMED“ gelistet.

Dieses Verhalten lässt sich auch direkt beim Anlegen einer neuen PDB erzwingen. Dazu wird beim Erzeugen einer neuen PDB die Klausel „STANDBYS=NONE“ verwendet. Dies führt direkt dazu, dass das Recovery der entsprechenden PDBs auf allen Standby Datenbanken übersprungen wird und damit der Managed Recovery Prozess nicht abbricht.

Wenn Sie eine PDB auf der Standby-Seite vom Recovery ausschließen, diese aber zu einem späteren Zeitpunkt mit dem Befehl

```
SQL> ALTER PLUGGABLE DATABASE <name> ENABLE RECOVERY;
```

wieder als Standby-Datenbank nutzen möchten, müssen Sie ein manuelles Restore und Recovery der PDB durchführen um die PDB auf denselben Stand wie die übrigen PDBs innerhalb der Standby CDB zu bringen.

Da dies unter Umständen recht aufwändig sein kann, ist der Default beim Anlegen neuer PDBs auch, dass das Recovery auf allen Standby Datenbanken durchgeführt wird, d.h. neue PDBs werden automatisch mit „STANDBY=ALL“ angelegt.

Gerade wenn mehrere Standby Datenbanken existieren und die PDB nur in einigen davon aktualisiert werden soll, ist es einfacher die PDB zunächst mit „STANDBYS=ALL“ zu erstellen. Auf der Standby-Seite hält daraufhin der Managed Recovery Modus automatisch an. Dort wo die PDB nicht im Standby Modus betrieben werden soll, führen Sie den Befehl

```
SQL> ALTER PLUGGABLE DATABASE <name> DISABLE RECOVERY;
```

aus und starten wie gewohnt den Managed Recovery Prozeß.

Nichts desto trotz bietet sich damit die Möglichkeit gezielt Standbys mit weniger PDBs zu betreiben, als die Primärdatenbank enthält. Ein Umschalten solcher Konfigurationen sollte aber gut überlegt sein, da eben nicht alle Datenbanken auf der Standby Seite enthalten sind. Automatisches Umschalten, wie durch ein Fast-Start- Failover Konfiguration empfiehlt sich daher schon gar nicht. Ebenfalls sollte der

erhöhte administrative Aufwand nicht unterschätzt werden, mehrere Standbys zu betreiben, in denen nur ein Subset der PDBs enthalten sind.

Der Recovery-Status der einzelnen PDBs in solch gemischten Konfiguration, kann direkt über v$PDBS auf der Standby Seite abgefragt werden:

```
SQL> SELECT RECOVERY_STATUS FROM V$PDBS;
```

Mehr Informationen zum gezielten Einsatz von „`STANDBY=NONE/ALL`“ bzw. „`ALTER PLUGGABLE DATABASE DISABLE/ENABLE RECOVERY`“ findet sich in der My Oracle Support Note 1916648.1: „Making Use of the STANDBYS=NONE Feature with Oracle Multitenant“

Ein weiterer Unterschied in der Standby Verwaltung von PDBs liegt in der Möglichkeit, einzelne PDBs auf der Standby-Seite zu schließen (zum Beispiel bei Active Data Guard), obwohl diese auf der Primär-Seite geöffnet sind. Ein ALTER PLUGGABLE DATABASE OPEN/CLOSE ist also möglich. Das Schließen einer PDB auf Standby-Seite ist besonders wichtig, wenn eine PDB gelöscht (DROP) oder ausgeklinkt (UNPLUG) werden soll. Damit dies funktioniert, muss auf der Standby-Seite zuerst die Datenbank geschlossen worden sein. Wird dieses versäumt, wird die Standby-Seite den Recovery-Prozess für die komplette CDB (inklusive aller PDBs) anhalten, bis die PDB gestoppt

worden ist und der Managed-Recovery-Modus wieder aktiviert wurde.

Während ein DROP PLUGABBLE DATABASE selbstverständlich die Datendateien auch auf der Standby-Seite löscht (bei Verwendung von Oracle Managed Files), sollte man bei einem UNPLUG berücksichtigen, dass hiermit nur die Datendateien ausgeklinkt werden – genau wie auf der Primär-Seite bleiben diese auf Platte bestehen. Löscht man dann die Dateien auf der Primär-Seite manuell, muss man daran denken, dass die Dateien auf Standby-Seite nach wie vor existieren.

12 Vergleich von CDB und PDB

Zum Abschluss gibt die folgende Tabelle eine Übersicht über die typischen Verwaltungsoperationen sowohl in einer CDB als auch in einer PDB:

Verwaltung	*CDB-Ebene*	*PDB-Ebene*
Start/Stopp einer CDB	*Starten/Stoppen der CDB*	*Nein*
Start/Stopp einer PDB	*Schließen und Öffnen einzelner PDB*	*Schließen und Öffnen der betreffenden PDB*
Hinzufügen einer PDB	*Ja*	*Nein*
Löschen einer PDB	*Ja*	*Ja*
Instanzverwaltung	*Ja*	*Nein*
Parameter	*Globale Parameter (zum Beispiel Memory, Charakterset, Archive-Redolog-Lokationen, Threads)*	*Lokale Parameter (zum Beispiel Session-Parameter)*
Ressourcen	*Globale Verteilung der Ressourcen über alle PDBs*	*Lokale Verteilung der Ressourcen innerhalb der PDB*

Storage	*CDB-Ebene*	*PDB-Ebene*
System/Sysaux Tablespace	*Ja*	*Ja*
User Tablespace	*Nur sinnvoll für administratorische Zwecke*	*Ja*
Temp Tablespace	*Bedingt sinnvoll*	*Ja*
Undo Tablepace	*Ja*	*Nein*
Online Redologs	*Ja*	*Nein*
Benutzerobjekte und Daten	*Nur sinnvoll für administratorische Zwecke*	*Ja*

Security	CDB-Ebene	PDB-Ebene
Benutzer	SYS, SYSTEM, Common User	SYS, SYSTEM, Common User, Lokale PDB-Benutzer (DBAs und Anwendungsbenutzer)
Daten-Berechtigung	Common User können Rechte für die CDB bekommen	Common User können Rechte für die PDBs bekommen. Lokaler Nutzer haben nur Rechte auf eigener PDB

Verfügbarkeit	CDB-Ebene	PDB-Ebene
Data Guard	Standby-System für die komplette CDB inklusive aller PDBs	-
Backup	Komplettes Backup für gesamte CDB inklusive aller PDBs oder Backup/Restore einzelner PDBs, Backup der Archivelogs	Backup/Restore der einzelnen PDB
Patching	Patching der CDB durch Skripte, hat Auswirkung auf alle PDBs innerhalb der CDB	Patch durch schnelles Unplug/Plug der PDB von „alter" CDB in „neue" CDB. Ggf. noch Starten eines Skripts.

13 Weitere Informationen

Weitere Informationen stehen Ihnen unter *tinyurl.com/multitenantmoreinfos* zur Verfügung.

Copyright © 2014, Oracle. All rights reserved. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Herausgeber: Günther Stürner, Oracle Deutschland B.V.
Design: volkerstegmaier.de // Druck: Stober GmbH, Eggenstein

ORACLE®

Zugriff auf die komplette
Oracle Dojo-Bibliothek unter
http:// tinyurl.com/dojoonline

ORACLE®