ORACLE

ORACLE DOJO NR. 7

RALF DURBEN

Oracle Multitenant Database

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

Der Begriff Dojo [ˈdoːdʒo] 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* beschäftigt sich Ralf Durben mit der modernen Container-Datenbank Architektur, auch Multitenant genannt, die Oracle seit der Datenbank Version 12c anbietet. Dabei werden die wichtigsten Features bis zur Datenbank Version 18c berücksichtigt. Ein Dankeschön geht an Sebastian Solbach und Manuel Hoßfeld, die an der ersten Auflage dieses Dojos mitgewirkt haben und an Ulrike Schwinn für das Lektorat.

ORACLE®

Inhalt

ORACLE®

ORACLE

ORACLE DOJO NR. 7

RALF DURBEN

Oracle Multitenant Database

VORWORT

Auch in den Zeiten von Cloud Computing und agiler Entwicklung ist die Speicherung der Daten von hoher Bedeutung. Nicht umsonst wird über die Daten oftmals vom „modernen Gold“ gesprochen. Jedoch kommen neue Herausforderungen auf die Oracle Datenbank zu.

Die Anwendungsentwicklung erfolgt mit neuen agilen Methoden, wie zum Beispiel DevOps, und produziert in immer schnellerer Abfolge neue Anwendungen und deren Versionen. Neben der Konsolidierung von Ressourcen müssen Datenbanken heute also viel schneller und möglichst automatisiert bereitgestellt werden. Sowohl neue, als auch Kopien von bestehenden Datenbanken für Entwicklung oder Tests, werden teilweise nur kurze Zeit und gleichermaßen mit wenig Vorlaufzeit benötigt. Auch die Mobilität von Datenbanken in einer hybriden Cloud-Umgebung ist von hoher Wichtigkeit.

Um diesen Anforderungen Rechnung zu tragen, hat Oracle mit der Datenbank Version 12c die containerbasierte Architektur „Multitenant“ eingeführt und in der aktuellen Version 18c stark erweitert.

Dieses Dojo gibt einen schnellen Einstieg in diese Datenbankarchitektur und zeigt auf, wie heutige Anforderungen mit neuen innovativen Wegen gelöst werden. Die vorliegende Auflage basiert auf der Datenbank Version 18c. Da aufgrund des Formats eine Auswahl der wichtigsten Features und Möglichkeiten erfolgen musste, wird dieses Dojo ergänzt durch eine Themenseite in unserem Cloud- und Technologieblog, die Sie unter https://blogs.oracle.com/coretec/Dojo7 finden.

Ihr Ralf Durben
Master Principal Sales Consultant

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

Zugriff auf alle Dojos: http://tinyurl.com/dojos-online

1 Einleitung

Oracle Multitenant in der Oracle Datenbank ist seit der Version 12c nicht nur ein neues Feature, sondern eine für Oracle Datenbanken seit der Version 12 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. Der Betrieb von Oracle Datenbanken mit der „alten" Architektur ist in der Version 18c weiterhin möglich.

Dieses Dojo gibt einen Einstieg in Oracle Multitenant, wobei Basiswissen zur Administration von Oracle-Datenbanken vorausgesetzt wird. Des Weiteren kann dieses Dojo nicht das Handbuch ersetzen. Daher werden bei Syntaxhinweisen nicht alle möglichen Optionen genannt.

Ein Oracle-Datenbanksystem besteht bis zur Version 11 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, auch „Pluggable Datenbank“ oder kurz PDB genannt, eine globale Instanz teilen können.

Die PDB wird in eine sogenannte Container-Datenbank (oder kurz CDB) „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 von Oracle Multitenant 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.

Eine CDB kann bis zu 4096 PDBs betreiben, wenn sie in einer Exadata Maschine oder in einem Oracle Cloud Service läuft. Bei sonstigen Installationen der Oracle 18c Enterprise Edition Datenbank beträgt die maximale Anzahl von PDBs pro CDB 252. In einer CDB der 18c Express Edition dürfen bis zu 3 PDBs genutzt werden.

- ***Non-CDB***
  Eine Oracle-Datenbank, die nach herkömmlicher Architektur 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
- Redo-Log-Dateien

besitzt. Diese Komponenten werden zentral von der CDB bereitgestellt, in der die PDB betrieben wird. Des Weiteren kann die CDB mit zentralen Undo Tablespaces für die PDBs betrieben werden, wobei auch auf PDB Ebene ein lokales Undo Management genutzt werden kann. Dieses muß aber auf der Ebene der CDB festgelegt werden und eine CDB läuft entweder im „Shared Undo Mode“ mit zentralem Undo Management, oder im „Local Undo Mode“.

Änderungen beim Inhalt der Datenbanken (Data Dictionary)

Im Rahmen von Oracle Multitenant wird das Data Dictionary aufgeteilt in:

- Einen statischen Bereich, der die für die eingesetzte Datenbank-Software feststehenden Informationen beinhaltet. Dieser Bereich enthält Informationen, die während der Lebenszeit der Datenbank mit der jeweils gegebenen Version nicht mehr verändert werden und für alle Oracle Datenbanken mit dieser Version gültig sind.
- Einen individuellen Bereich, dessen Inhalte durch die Nutzung der Datenbank entstehen.

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 und damit unabhängig von der Anzahl der PDBs nur einmal gespeichert sind. Die interne Trennung hat keinerlei Auswirkung auf die Nutzung des Data Dictionaries!

Änderungen beim Inhalt der Datenbanken (Datenmodell für Anwendungen)

Auch auf der Ebene der Anwendung ist eine Trennung von Metadaten des Datenmodells und der Daten selbst sinnvoll, wenn die Anwendung mehrfach betrieben wird. Anwendungsfälle hierfür sind z.B. verschiedene Stufen zur Produktion (wie Test-, Integration- und Produktionsdatenbank) oder eine für mehrere Mandanten parallel betriebene Applikation. Dabei würde jeder Mandant eine eigene PDB bekommen.

In solch einem Fall müssten alle Definitionen von Tabellen, Indizes, PL/SQL Prozeduren und weiterer Datenbankobjekte in jeder einzelnen PDB gespeichert werden.

Durch die Verwendung von Applikationscontainern kann hier eine Zentralisierung vorgenommen werden, wobei die

PDBs, die in einem Applikationscontainer laufen, zentral gespeicherte Definitionen per Referenzierung einbinden. So kann eine Tabelle in dem Applikationscontainer erstellt werden und damit allen PDBs zur Verfügung gestellt werden. Die PDBs speichern die Daten dieser Tabellen aber lokal ab, so dass hier eine physische Trennung vorgenommen wird.

Eine Ausnahme stellen dabei übergreifende Stammdaten dar, denn ein Applikationscontainer kann auch vorgegebene Daten für die Anwendungs-PDBs zur Verfügung stellen. Ein einfaches Beispiel wäre ein Katalog aller Postleitzahlen Deutschlands, der auch nur im Applikationscontainer gepflegt werden muß und dann immer automatisch allen PDBs zur Verfügung steht.

1.3 WELCHE VORTEILE BIETET DIE MULTITENANT 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, bzw. Applikationscontainer und einen dezentralen Bereich in den PDBs bewirkt eine deutliche Absenkung des Storagebedarfs, da der zentrale Bereich nur noch einmal gespeichert wird.

Gleiches gilt für umfangreiche Definitionen von Datenmodellen, die in einem Applikationscontainer einmalig

gespeichert werden und dann in allen Anwendungs-PDBs genutzt werden können.

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. Dieses ist sowohl mit oder ohne einer Downtime der PDB möglich.

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 mit Pluggable Datenbanken optimal umgesetzt.

Patchen

Das Patchen einer Oracle-Datenbank kann mithilfe der neuen Architektur extrem vereinfacht werden: Eine PDB kann gepatcht werden, indem sie zunächst aus ihrer ursprünglichen CDB entnommen („unplugged“) und anschließend in eine mit dem entsprechenden Patch versehenen neuen CDB eingeklinkt („plugged“) wird. Es ist dann noch das Einspielen eines Skripts erforderlich, welches aber weniger Zeit in Anspruch nimmt, als bei einer Non-CDB.

Datenbank-Cloning

Eine PDB kann sehr leicht und schnell dupliziert werden, sowohl innerhalb einer CDB, als auch zwischen mehreren CDBs. Somit ist es mit wenig Aufwand möglich einen Datenbankklon zu erstellen, zum Beispiel um eine Test-

datenbank aufzusetzen. Dieses Klonen ist wieder mit oder ohne Downtime für die PDB möglich. Des Weiteren können Sie mittels Refresh-Mechanismen einen Datenbankklon mit dem Original synchronisieren.

Resource Manager

Die Zuteilung von Ressourcen, wie zum Beispiel CPU oder Parallelität, ist seit vielen Jahren mit dem Oracle Database 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-Redo-Log-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 Oracle Multitenant im Grunde um eine Virtualisierung von Datenbanken mit den Mitteln der Datenbank selbst. Dieses stellt eine interessante Alternative zum Betrieb von Non-CDB Datenbanken in virtuellen Maschinen dar. Auch der althergebrachten Schema-Konsolidierung oder dem

Zusammenlegen verschiedener ORACLE_HOMEs auf einem Server ist Oracle Multitenant überlegen.

Worin genau liegen nun also die Unterschiede dieser schon länger bekannten Methoden und wie lassen sich diese zu Oracle Multitenant 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 vornherein 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.

Bezüglich der Nutzung von Ressourcen weist dieser Ansatz aber wieder den Nachteil auf, dass bei Datenbanken, die nicht unter Volllast stehen, die allokierten SGA-Ressourcen brach liegen.

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):

	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	*****	*	***	***

Isolierte Verwaltung durch Fachabteilungen oder Entwickler	*****	*****	***	*
Zentrales Desaster Recovery	*****	*****	–	–
Zentrales Backup & Recovery	*****	*****	–	–
Zentrales Ressourcenmanagement	*****	***	*	–

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

3 Deployment von Pluggable Databases

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

Im DBCA können Sie beim Erstellen einer Oracle Datenbank angeben, ob diese als Container Datenbank erstellt werden soll. Wenn Sie dieses auswählen, können Sie angeben, wieviele PDBs angelegt werden sollen.

Falls Sie die Modifikation einer CDB auswählen, 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')
 MAXLOGHISTORY 1
 MAXLOGFILES 16
 MAXLOGMEMBERS 3
 MAXDATAFILES 1024
 CHARACTER SET AL32UTF8
 NATIONAL CHARACTER SET AL16UTF16
 EXTENT MANAGEMENT LOCAL
    DATAFILE '/u01/app/oracle/oradata/newcdb/system01.dbf'
    SIZE 700M REUSE AUTOEXTEND ON NEXT 10240K MAXSIZE UNLIMITED
 SYSAUX
    DATAFILE '/u01/app/oracle/oradata/newcdb/sysaux01.dbf'
    SIZE 550M REUSE AUTOEXTEND ON NEXT 10240K MAXSIZE UNLIMITED
 DEFAULT TABLESPACE deftbs
    DATAFILE '/u01/app/oracle/oradata/newcdb/deftbs01.dbf'
    SIZE 500M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED
 DEFAULT TEMPORARY TABLESPACE tempts1
    TEMPFILE '/u01/app/oracle/oradata/newcdb/temp01.dbf'
    SIZE 20M REUSE AUTOEXTEND ON NEXT 640K MAXSIZE UNLIMITED
 UNDO TABLESPACE undotbs1
    DATAFILE '/u01/app/oracle/oradata/newcdb/undotbs01.dbf'
    SIZE 200M REUSE AUTOEXTEND ON NEXT 5120K MAXSIZE UNLIMITED
ENABLE PLUGGABLE DATABASE
```

```
    SEED
    FILE_NAME_CONVERT = ('/u01/app/oracle/oradata/newcdb/',
                         '/u01/app/oracle/oradata/pdbseed/')
    SYSTEM DATAFILES SIZE 125M AUTOEXTEND ON NEXT 10M MAXSIZE
    UNLIMITED
    SYSAUX DATAFILES SIZE 100M
LOCAL UNDO ON
USER_DATA TABLESPACE usertbs
DATAFILE '/u01/app/oracle/oradata/newcdb/usertbs01.dbf'
SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

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.

Mit der Klausel `LOCAL UNDO ON` wird der „Local Undo Mode“ aktiviert, bei dem die einzelnen PDBs einen eigenen Undo Tablespace bekommen. Dieses ermöglicht zum Beispiel ein einfach durchzuführendes FLASHBACK DATABASE auf PDB Ebene. Der „Local Undo Mode“ wird auch automatisch vom Tool DBCA (Database Configuration Assistant) verwendet.

Die Verwendung des Zeichensatzes AL32UTF8 ist sehr empfehlenswert, da in diesem Fall die einzelnen PDBs, die später in dieser CDB betrieben werden, möglichst frei in der Wahl ihres Zeichensatzes sind. Es gilt nämlich die Regel, dass PDBs einen anderen Zeichensatz verwenden können als die CDB, solange dieser in der PDB genutzte Zeichensatz eine Teilmenge des in der CDB genutzten Zeichensatzes ist. Beispielsweise kann im obigen Fall eine PDB den Zeichensatz „we8iso8859p1“ verwenden (für West-Europa) und eine andere PDB den Zeichensatz „Kanji“ (für Japan).

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 pdb_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 `PDB_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. Dabei gibt es drei Varianten:

- das Klonen einer PDB, deren Inhalt sich während des Klonens nicht ändert

- Hot Cloning
- Kloning mit der Option REFRESHABLE, bei der die Klon-PDB bzgl. aller Änderungen in der Quell-PDB aktualisiert wird.

Das Klonen von PDBs kann mit den nachfolgend beschriebenen SQL-Kommandos oder unter Verwendung des Database Configuration Assistants (DBCA) durchgeführt werden.

ACHTUNG: *Beachten Sie beim Klonen von PDBs, dass zwar die PDB mit ihren Inhalten dupliziert wird, jedoch nicht die Einstellung hinsichtlich PDB Lockdowns, die in einem späteren Kapitel beschrieben werden!*

Variante 1: Klonen einer READ ONLY PDB

Dieses ist die Variante, die in der Version 12.1 der Oracle Datenbank zur Verfügung steht. Damit die zu klonende PDB geklont werden kann, muss sie in einen transaktionskonsistenten Zustand gebracht werden 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 zum Klonen einer PDB 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:

```
SQL> CREATE PLUGGABLE DATABASE pdb2 FROM pdb1@db_link_to_pdb1;
```

Variante 2: Hot Cloning

Wenn Ihre CDB im Archivelog Modus läuft können Sie auch einen sogenannten „Hot Clone“ erstellen. Dabei verwenden Sie einfach das oben verwendete Kommando, ohne die Quell-PDB in den READ ONLY Status versetzt zu haben. Während des Klon-Vorgangs kann die Quell-PDB ganz normal genutzt werden.

Das „Hot Cloning“ funktioniert auch zwischen zwei unterschiedlichen CDBs.

Variante 3: REFRESHABLE Clone

Sie können einen PDB-Klon auch so erstellen, dass alle Datenänderungen, die in der Quell-PDB vorgenommen werden, auch im Klon angewendet werden. Dieses ist zum Beispiel sinnvoll, wenn die Klone als Datenfarm für Lesezugriffe verwendet werden sollen und die Aktualisierung der Daten nicht sofort durchgeführt werden muß.

Dabei können Sie die Aktualisierung manuell vornehmen, oder ein Zeitintervall für automatische Aktualisierungen definieren. Die Klon-PDB muß dabei im READ ONLY Modus und die Quell-PDB im ARCHIVELOGMODE betrieben werden.

Beachten Sie, dass Quell- und Ziel-PDB in verschiedenen Container-Datenbanken (CDB) liegen müssen. Refreshable Klone innerhalb der CDB, in der die Quell PDB liegt, können nicht erstellt werden. In diesem Fall bekommen Sie einen Syntax-Fehler. Sie können aber in einem Szenario von zwei CDBs die Quell-PDB in der CDB1 und mehrere Refreshable Klone in der CDB2 betreiben.

Weiterhin ist zu beachten, dass in der Ziel-CDB ein Datenbank-Link für den Zugriff auf die Quell-CDB benötigt wird. Dieser Datenbank-Link muß eine Verbindung zur Quell-CDB mit einem Benutzer aufbauen, der in beiden CDBs existieren muß. Der Benutzer SYSTEM ist dazu geeignet. Möchten Sie einen eigenen Benutzer verwenden,

der als Common User in der CDB erstellt wird, müssen Sie folgende Rechte vergeben:

```
SQL> GRANT create session, resource TO dblink_user CONTAINER=all;
SQL> GRANT create pluggable database TO dblink_user CONTAINER=all;
SQL> GRANT sysoper TO dblink_user CONTAINER=all;
```

Außerdem muß der User auf alle Container zugreifen können. Also muß von der CDB$ROOT folgendes ausgeführt werden:

```
SQL> ALTER USER dblink_user SET CONTAINER_DATA=all
     CONTAINER=current;
SQL> GRANT select ON cdb_pdbs TO dblink_user;
```

Sie erstellen einen Refreshable Klon zum Beispiel mit

```
SQL> CREATE PLUGGABLE DATABASE pdb2 FROM pdb1@db_link_to_cdb1
    PATH_PREFIX = '/oracle/oradata/pdb2'
    FILE_NAME_CONVERT = ('/oracle/oradata/pdb1/',
                         '/oracle/oradata/pdb2/')
    STORAGE (MAXSIZE 100G MAX_SHARED_TEMP_SIZE 900M)
    REFRESH MODE MANUAL;
```

für manuelle Refreshs. Alternativ geben Sie an:

- REFRESH MODE EVERY n MINUTES
  für automatische Aktualisierungen

- REFRESH MODE NONE
  um die Aktualisierung abzuschalten und die PDB zum Beispiel in den READ WRITE Modus zu bringen.

Ein manuelles Refresh erzeugen Sie, indem Sie sich mit der PDB verbinden

```
SQL> ALTER SESSION SET CONTAINER=pdb2;
```

Dann müssen Sie die PDB kurz schließen mit

```
SQL> ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;
```

und führen dann ein REFRESH aus.

```
SQL> ALTER PLUGGABLE DATABASE REFRESH;
```

Anschließend öffnen Sie die PDB wieder in den READ ONLY Modus:

```
SQL> ALTER PLUGGABLE DATABASE OPEN READ ONLY;
```

Die Daten im Klon sind jetzt auf dem aktuellen Stand. Auf diese Weise können sehr leicht Kataloge zur Verfügung gestellt werden, die zum Beispiel einmal pro Tag aktualisiert werden müssen.

Wenn Ihre Container Datenbank (CDB) als Datenbankservice in der Oracle Cloud betrieben wird, können Sie Refreshable Clone PDBs aber auch nutzen, um ein Standby-Konzept für einzelne PDBs zu nutzen. Das bekannte Standby-Konzept Data Guard wirkt immer auf die gesamte CDB, also auch auf alle PDBs

gleichermaßen, so dass ein Switchover oder Failover immer alle PDBs betrifft. Refreshable Clone PDBs können ihren Status (Master oder Standby) tauschen, es ist hier also auch ein Switchover oder Failover möglich, jedoch auf der Ebene der einzelnen PDB.

Der Statuswechsel, also ein Switchover wird auf der Seite der Master PDB durchgeführt. Dazu muß in der entsprechenden CDB wieder ein Datenbanklink auf die CDB mit dem bisherigen Refreshable Klon existieren. Sie verbinden sich also mit der Master PDB und erzielen den Statuswechsel dann mit

```
SQL> ALTER SESSION SET CONTAINER = pdb1;
SQL> ALTER PLUGGABLE DATABASE REFRESH MODE MANUAL
     FROM pdb2@db_link_to_cdb2
     SWITCHOVER;
```

Jetzt ist die Pluggable Datenbank pdb1 ein Refreshable Klon, der nur READ ONLY geöffnet wird.

3.3 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 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 Oracle Data Pump 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 von der Version 12c oder höher sein, 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 höher) 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. In den folgenden Abschnitten wird von einem funktionierenden Datenbank-Link 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'*, welches Sie im Verzeichnis *$ORACLE_HOME/rdbms/admin* finden.

```
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.

3. 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
```

3.4 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;
```

3.5 VERSCHIEBEN VON PDBS ZWISCHEN VERSCHIEDENEN CDBS

Pluggable Datenbanken können von einer CDB zu einer anderen CDB verschoben werden. Dabei stehen zwei Verfahren zur Verfügung:

- Aushängen (Unplug) aus der alten CDB, Kopieren der Datendateien und Einhängen (Plug) der PDB in die neue CDB.
- Relocate-Operation, wobei der Umzug komplett automatisch und mit minimaler Downtime erfolgt.

Plug und Unplug von PDBs

Aushängen („Unplug“) einer PDB

Sie können jederzeit eine PDB aus einer CDB aushängen. 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.

Mit dem Aushängen der PDB ist diese nicht mehr erreichbar, der Transport ist also mit Downtime verbunden.

Ein „Unplug“, also das Aushängen einer PDB aus einer CDB, können Sie mit dem Database Configuration Assistant (DBCA), Oracle Enterprise Manager und auch SQL Developer durchführen. Dazu wählen Sie im Falle des DBCA 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 ausgehängt werden, wenn sie mindestens einmal geöffnet war. Prüfen Sie dieses.
2. Fahren Sie die PDB herunter:

```
SQL> SHUTDOWN IMMEDIATE
```

3. Ein Aushängen 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 einzuhängen.

5. Die ausgehängte 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;
```

Einhängen („Plug“) einer PDB

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

Das Einhängen 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 eingehängt werden.

4. Hängen 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. Erstellen Sie jetzt ein Backup der PDB, damit diese in Zukunft per Recovery wiederhergestellt werden kann.

PDB Relocate

Mit PDB Relocate können Sie eine PDB fast online von einer CDB in eine andere CDB transportieren. Fast online bedeutet, dass es einen kurzen Umschaltzeitraum gibt, der für die Anwendungen eine Downtime bedeutet. Dazu muß es einen

Datenbanklink von der Ziel-CDB zu Quell-CDB geben, wie es auch schon oben zu den Refreshable Clones beschrieben wurde, wobei der im Datenbanklink verwendete Benutzer neben dem Recht CREATE PLUGGABLE DATABASE auch das Recht SYSOPER oder SYSDBA haben muß.

Grundsätzlich kann durch PDB Relocate eine PDB auch zwischen CDBs verschoben werden, die auf unterschiedlichen Plattformen laufen, jedoch müssen diese die gleiche Filesystem Endianness besitzen. Des Weiteren muß die Ziel-CDB natürlich auch alle Datenbankoptionen installiert haben, wie die Quell-CDB.

Die Quell-PDB muß dabei entweder im ARCHIVELOG betrieben werden (vorgegeben durch die Quell-CDB), oder im READ ONLY Modus sein.

Für das PDB Relocate gehen Sie wie folgt vor:

1. Erstellen Sie einen Benutzer in der Quell-CDB:

```
SQL> CREATE USER c##admin IDENTIFIED BY <passwort>;
SQL> GRANT connect,resource,create pluggable database,sysoper
    TO c##admin CONTAINER=ALL;
```

2. Erstellen Sie den Datenbanklink mit

```
SQL> CREATE PUBLIC DATABASE LINK db_link_to_cdb1
    CONNECT TO c##admin IDENTIFIED BY <passwort> USING 'CDB1';
```

3. Führen Sie in der Ziel-CDB das `CREATE PLUGGABLE DATABASE` Kommando aus:

```
SQL> CREATE PLUGGABLE DATABASE pdb1 FROM pdb1@db_link_to_cdb1
       FILE_NAME_CONVERT = ('/oracle/oradata/pdb1/',
                            '/oracle/oradata/pdb1/')
       RELOCATE AVAILABILITY MAX;
```

4. Die PDB ist in der Ziel-CDB im Modus MOUNTED und im Status RELOCATING. Dieses können Sie sehen mit

```
SQL> SELECT name,open_mode FROM v$pdbs WHERE name='PDB1';
 NAME       OPEN_MODE
 --------   ----------
 PDB1       MOUNTED

SQL> SELECT pdb_name,status FROM cdb_pdbs WHERE pdb_name='PDB1';
 PDB_NAME   STATUS
 --------   ----------
 PDB1       RELOCATING
```

5. Sie können immer noch mit der Quell-PDB arbeiten und auch Daten ändern.

6. Sie öffnen jetzt die PDB in der Ziel-CDB mit

```
SQL> ALTER PLUGGABLE DATABASE pdb1 OPEN;
```

7. Jetzt wird die transportierte PDB auf den neuesten Stand gebracht. Während dieses Schrittes ist die PDB weder in der Quell-CDB noch in der Ziel-CDB verwendbar. Jetzt ist also eine Downtime zu verzeichnen. Die PDB in der Ziel-CDB wird auch automatisch als Service eingetragen.

   Je nachdem, ob die beiden CDBs mit dem gleichen Listener arbeiten, oder nicht, verwenden Sie die Option AVAILABILITY. Falls es unterschiedliche Listener sind, verwenden Sie `AVAILABILITY MAX` und der Quell-Listener wird so konfiguriert, dass er die Verbindungsaufrufe korrekt an die neue Adresse weiterleitet.

8. Der Status der PDB ist jetzt NORMAL

```
SQL> SELECT pdb_name,status FROM cdb_pdbs WHERE pdb_name='PDB1';
 PDB_NAME   STATUS
 --------   ----------
 PDB1       NORMAL
```

9. Erstellen Sie jetzt ein Backup der PDB, damit diese in Zukunft per Recovery wiederhergestellt werden kann.

   PDB Relocate kann auch eingesetzt werden, um PDBs zu patchen. Dazu muß nach dem Kopiervorgang der PDB in der Ziel-CDB das Tool „datapatch“ ausgeführt werden. Damit ist ein Patching mit minimaler Downtime möglich.

Des Weiteren ist diese Methode ideal, um einzelne Datenbanken in die Cloud zu verschieben, denn während des Kopiervorgangs, so lange er auch dauern mag, ist die Datenbank (PDB) weiterhin verfügbar.

4 Die Nutzung von Applikations-containern

Oftmals werden für eine Applikation mehrere Datenbanken benötigt, sei es um neben der Produktions-Datenbank auch Test- und Integrations-Datenbanken zu betreiben, oder auch wenn eine Anwendung mehrfach, zum Beispiel getrennt für verschiedene Filialen, Regionen oder Mandanten, betrieben wird. In allen diesen Fällen benötigen diese Datenbanken das gleiche Datenmodell und der traditionelle Ansatz zur Erstellung einer solchen Datenbank besteht im Ausführen vorgegebener Skripte in jeder einzelnen Datenbank.

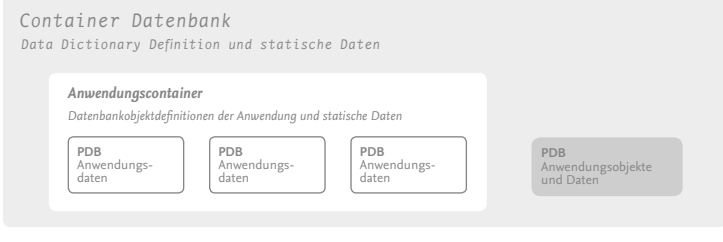
Die Multitenant Architektur beinhaltet, wie in Kapitel 1 beschrieben, eine Trennung der Speicherung von Data Dictionary Definitionen und auch Inhalten, wobei ein statischer Teil auf CDB Ebene und der dynamische Teil auf PDB Ebene gespeichert wird.

Das Data Dictionary ist das Datenmodell der Oracle Datenbank und man kann es mit einem Datenmodell von Anwendungen

gut vergleichen. Also kann man auch auf der Ebene der Anwendung diese Trennung durchführen. Es gibt für jede Anwendung

- Das Datenmodell, also die Definition von Datenbankobjekten
- Statische Daten, wie zum Beispiel ein Katalog von Postleitzahlen
- Dynamische Daten, wie zum Beispiel Buchungsdaten

Wenn für eine Anwendung mehrere Datenbanken betrieben werden, stellt sich schnell die Frage, ob man nicht, wie beim Data Dictionary auch, eine einmalige, zentrale Speicherung all dessen durchführen kann, was in allen PDBs gleich genutzt wird. Genau für dieses Ziel gibt es den Applikationscontainer (auch Anwendungscontainer genannt). Das folgende Schaubild zeigt diesen Zusammenhang.



Da eine CDB (je nach Art der Nutzung) für bis zu 4096 PDBs genutzt werden kann, werden in aller Regel auch mehrere verschiedene Anwendungen in mehreren PDBs betrieben. Sie können dazu mehrere Applikationscontainer in einer CDB betreiben. Jeder Applikationscontainer ist sowohl eine PDB (in der Haupt-CDB), als auch CDB (für die Anwendungs-PDBs). Auch für den Applikationscontainer kann eine Seed-Datenbank vorbereitet werden, um die Erstellung neuer PDBs zu beschleunigen.

4.1 ERSTELLEN EINES APPLIKATIONSCONTAINERS

Um einen Applikationscontainer zu erstellen, melden Sie sich an die CDB an. Sie benötigen das Privileg `CREATE PLUGGABLE DATABASE`. Dann verwenden Sie das Kommando `CREATE PLUGGABLE DATABASE` ergänzt um die Klausel `AS APPLICATION CONTAINER`. Hier ein Beispiel:

```
SQL> CREATE PLUGGABLE DATABASE myapp_ac AS APPLICATION CONTAINER
       ADMIN USER myapp_ac_adm IDENTIFIED BY manager
       FILE_NAME_CONVERT=('pdbseed','myapp');
```

Sie öffnen dann den neuen Applikationscontainer mit

```
SQL> ALTER PLUGGABLE DATABASE myapp_ac OPEN;
```

Sie können sich nun an den neuen Applikationscontainer anmelden, um weitere Aktionen durchzuführen, die in den nächsten Abschnitten beschrieben werden.

4.2 ERSTELLEN EINER SEED-DATENBANK IN EINEM APPLIKATIONSCONTAINER

Mit einer Seed-Datenbank erstellen Sie durch schnelles Kloning die später angeforderten Anwendungs-PDBs. Um eine Seed-Datenbank zu erstellen, melden Sie sich an den Applikationscontainer an. Dann führen Sie das Kommando `CREATE PLUGGABLE DATABASE AS SEED` aus. Beachten Sie dabei, dass in einem Applikationscontainer nur eine Seed-Datenbank erstellt werden kann.

```
SQL> CREATE PLUGGABLE DATABASE AS SEED
       ADMIN USER pdbadmin IDENTIFIED BY manager
       FILE_NAME_CONVERT=('pdbseed','myapp_ac_seed');
```

Auch diese PDB müssen Sie zunächst öffnen mit

```
SQL> ALTER PLUGGABLE DATABASE myapp_ac$SEED OPEN;
```

In der SEED-Datenbank können Sie Datenbankobjekte erstellen, die später beim Kloning auch in der PDB lokal gespeichert sind. Dabei wird aber ein Kopieren vorgenommen. Spätere Änderungen in der Seed-Datenbank haben keinen Effekt auf bereits existierende Anwendungs-PDBs.

Des Weiteren stellen Sie die PDB-spezifischen Instanzparameter ein. An dieser Stelle verweise ich auf eine weitere Funktionalität, die in einem späteren Kapitel beschrieben wird: PDB Lockdown Profiles. Damit können Sie zum

Beispiel Einfluß nehmen auf die Datenbankoptionen und -features, die in der PDB nutzbar sind.

4.3 ERSTELLEN VON ANWENDUNGS-PDBS

Wenn Sie den Applikationscontainer inklusive Seed-Datenbank vorbereitet haben, können Sie nun die Anwendungs-PDBs erstellen. Dazu melden Sie sich an den Applikationscontainer an und erstellen die PDB zum Beispiel mit

```
SQL> CREATE PLUGGABLE DATABASE pdb_app2
       ADMIN USER pdbadmin IDENTIFIED BY manager
       DEFAULT TABLESPACE data
           DATAFILE '/oracle/oradata/pdb1/data01.dbf' SIZE 200m
       FILE_NAME_CONVERT=('myapp_ac_seed','pdb_app2');
```

Dabei wird die Seed Datenbank geklont. Beachten Sie, dass nachträgliche Änderungen an der Seed Datenbank nicht für die bereits erstellten Anwendungs-PDBs gelten.

Die Verflechtungen im Bereich Applikationscontainer, also ob eine PDB in einem Applikationscontainer liegt, und wenn ja in welchem, können Sie der View V$CONTAINERS entnehmen mit

```
SQL> col name format a25
col app_root format a8
col app_pdb format a7
col app_seed format a8
```

```
col app_r_id format 99999999
set pagesize 100
SELECT con_id,name,
       application_root as app_root,
       application_pdb as app_pdb,
       application_seed as app_seed,
       application_root_con_id as app_r_id
FROM v$containers;
CON_ID  NAME                 APP_ROOT  APP_PDB  APP_SEED  APP_R_ID
------  -------------------  --------  -------  --------  ---------
     1  CDB$ROOT             NO        NO       NO
     2  PDB$SEED             NO        NO       NO
     3  PDB1                 NO        NO       NO
     4  DEMOS                NO        NO       NO
     5  myapp_ac$SEED        NO        YES      YES       7
     6  PDB_APP1             NO        YES      NO        7
     7  myapp_ac             YES       NO       NO
     8  PDB_APP2             NO        YES      NO        7
```

4.4 ERSTELLUNG DES ZENTRAL GESPEICHERTEN DATENMODELLS

Sie können im Applikationscontainer Datenbankobjekte speichern, die inhaltlich oder auf der Ebene der Metadaten von den Anwendungs-PDBs referenziert werden. Aus der Sicht der PDB sind dieses lokale Datenbankobjekte, die aber nur einmalig gespeichert sind.

Applikationscontainer unterstützen dabei auch eine Versionierung von Anwendungen. Dazu müssen spezielle Rahmenbedingungen geschaffen werden, die am Ende des Kapitels gezeigt werden. Ein einfaches Erstellen einer Tabelle in einem Applikationscontainer reicht also nicht, um diese allen Anwendungs-PDB zugänglich zu machen.

Vielmehr muß im Applikationscontainer in einer speziellen Syntax eine Anwendung mit einer angegebenen Version installiert werden. Entsprechend können Sie später diese Anwendung patchen und upgraden.

Am folgenden kleinen Beispiel erkennen Sie das Prinzip:

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION mywdg_app BEGIN INSTALL '1.0';
SQL> CREATE USER wdg IDENTIFIED BY wdg CONTAINER=ALL;
SQL> GRANT CREATE SESSION, DBA TO wdg;
SQL> CONNECT wdg/wdg@MYAPP_AC
SQL> CREATE TABLE wdg.plz_verzeichnis SHARING=DATA
                (PLZ number,ort varchar2(100));
SQL> INSERT INTO wdg.plz_verzeichnis VALUES (59556 , 'Lippstadt');
SQL> COMMIT;
SQL> ALTER PLUGGABLE DATABASE APPLICATION mywdg_app END INSTALL '1.0';
```

Sie beginnen also eine Installation einer Anwendung und erstellen einen oder mehrere Datenbankbenutzer mit der Option `CONTAINER=ALL`. Mit diesem Benutzer (oder Benutzern, falls mehrere erstellt wurden) erstellen Sie dann die Datenbankobjekte.

Es gibt dabei drei Varianten der gemeinsamen Nutzung von Objekten:

- ***METADATA***
  Die PDBs sehen nur die Definition, also die Metadaten von Datenbankobjekten, also zum Beispiel die Definition einer Tabelle mit Spalten und Datentypen. Die Daten werden allein in der Anwendungs-PDB gespeichert.
- ***DATA***
  Die Daten werden im Applikationscontainer gespeichert und stehen in der Anwendungs-PDB nur lesend zur Verfügung, wie zum Beispiel eine Tabelle mit den Postleitzahlen von Deutschland.
- ***EXTENDED DATA***
  Vorgegebene Daten sind im Applikationscontainer nur lesend gespeichert, aber die Anwendungs-PDB kann weitere Daten lokal speichern. Das könnte eine Tabelle mit global gültigen Daten sein, die in der Anwendungs-PDB mit lokalen Daten erweitert wird.

Mit dem Initialisierungsparameter „`DEFAULT_SHARING`“ können Sie auf der Ebene des Applikationscontainers einen Default Modus setzen. Standardmäßig ist dieser Parameter auf `METADATA` gesetzt.

Sie können aber auch auf Objektebene eine individuelle Einstellung vornehmen, wie im obigen Beispiel. Die Tabelle PLZ_VERZEICHNIS wird inklusive Daten im Applikationscontainer gespeichert und soll in den Applikations-PDBs nur lesend zur Verfügung stehen.

Nachdem die Anwendung im Applikationscontainer installiert ist, müssen Sie in den Anwendungscontainern einen SYNC durchführen mit

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION mywdg_app SYNC;
```

Erst dann sind die neuesten Definitionen verfügbar. Damit können also auch neue Versionen von Anwendungen im Applikationscontainer vorbereitet werden, ohne dass die Anwendungs-PDBs davon betroffen sind. Erst nachdem die neue Version getestet und freigegeben ist, schalten Sie die neue Version mit dem SYNC Kommando „scharf“. Im folgenden Beispiel wird eine neue Version der Anwendung mit einer neuen PL/SQL Funktion erstellt:

Im Applikationscontainer:

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION mywdg_app BEGIN UPGRADE
'1.0' to '2.0';
SQL> CREATE OR REPLACE FUNCTION wdg.plz_check (p_plz number,p_ort
varchar2)
RETURN number
```

```
AS
 valid number:=0;
 n     number;
BEGIN
 select count(*) into n
  from  wdg.plz_verzeichnis
 where  plz=p_plz and ort=p_ort;
 if n>0 then valid:=1; end if;
 return valid;
END;
/
SQL> ALTER PLUGGABLE DATABASE APPLICATION mywdg_app END UPGRADE to
'2.0';
```

Das Update wird in der Anwendungs-PDB einfach aktiviert mit

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION mywdg_app SYNC;
```

Mit den neuen Applikationscontainern ist es also möglich, Anwendungsobjekte (Tabellen, Indizes,...) und Anwendungslogik, die mit Datenbankobjekten wie zum Beispiel PL/SQL Stored Procedures implementiert ist, einmalig und zentral zu speichern. Damit verringert sich der Speicher- und Wartungsaufwand, wenn die Anwendung in mehreren Datenbanken betrieben wird. Gerade für den mandantenorientierten Einsatz von Oracle Multitenant ist dieses

Feature interessant, wenn das gleiche Datenmodell für jeden Mandanten in einer eigenen Datenbank verwendet wird.

5 Proxy-PDBs

In einer Umgebung mit verschiedenen Container-Datenbanken kann es notwendig sein, dass über CDB-Grenzen hinweg auf andere PDBs zugegriffen werden soll. Dieses kann direkt über Datenbanklinks geschehen, die auf lokal gültigen Aliasnamen für die Verbindungsinformationen basieren (TNS-Alias). Es gibt aber auch die Möglichkeit eine Brücke in Form einer Proxy-PDB zu erstellen.

Eine Proxy-PDB besteht dabei nur aus den System- und Sysaux-Tablespaces, die bei der Erstellung von der Quell-PDB kopiert werden. Sobald die Proxy-PDB erstellt und geöffnet ist, kann man sich an ihr ganz normal anmelden und Datenbankkommandos, wie zum Beispiel Datenabfragen und Datenänderungen durchführen. Dabei finden diese Aktionen immer in der Quell-PDB statt. Im Gegensatz zu einer mit der REFRESH-Option erstellten Kopie einer PDB (siehe Kapitel zur Erstellung von PDBs), die nur in eine Richtung aktualisiert wird, können Proxy-PDBs also nicht nur zum Lesen von Daten verwendet werden, sondern auch zum Ändern der Daten.

Dabei finden die von der Oracle Datenbank gewohnten Regeln zur Transaktionskonsistenz Anwendung. Eine Datenbank-Sitzung in der Proxy-PDB ist also genauso zu betrachten, wie eine Datenbanksitzung in der Quell-PDB.

5.1 ERSTELLEN VON PROXY-PDBS

Proxy-PDBs werden nur über CDB-Grenzen hinweg verwendet. Um eine Proxy-PDB zu erstellen, benötigen Sie also zunächst einen Datenbanklink von CDB zu CDB. Die folgenden Kommandos benutzen Sie auf der Ebene der CDB, in der die Proxy-PDB erstellt werden soll.

Sie erstellen zunächst den Datenbanklink mit

```
CREATE PUBLIC DATABASE LINK li_cdb1
       CONNECT TO system IDENTIFIED BY welcome1 USING 'CDB1';
```

Dann erstellen Sie die Proxy-PDB mit

```
CREATE PLUGGABLE DATABASE pdb_prox1 AS PROXY FROM pdb1@li_cdb1
    FILE_NAME_CONVERT = ('/u01/app/oracle/oradata/CDB1/pdb1/',
                         '/u01/app/oracle/oradata/CDB2/pdb_prox1/');
```

Anschließend öffnen Sie die neue Proxy-PDB

```
ALTER PLUGGABLE DATABASE pdb_prox1 OPEN;
```

Jetzt können Sie sich ganz normal bei der Proxy-PDB anmelden, zum Beispiel mit einem TNS-Alias, wenn dieser in der Datei *tnsnames.ora* erstellt wurde:

```
connect hr/welcome1@pdb_prox1
```

und können jetzt so mit den Daten arbeiten, als hätten Sie sich direkt an der PDB pdb1 angemeldet.

6 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
```

Die folgenden Abschnitte behandeln typische Abfragen.

6.1 PRÜFEN DES ZUSTANDS EINER PDB 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
```

6.2 PRÜFEN DES ZUSTANDS EINER PDB 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$CONTAINERS;
    CON_ID  NAME                           OPEN_MODE
----------  ------------------------------  ----------
         1  CDB$ROOT                       READ WRITE
         2  PDB$SEED                       READ ONLY
         3  PDBRED                         READ ONLY
         4  PDB4                           READ WRITE
```

6.3 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 sich nach einem Neustart einer CDB im gleichen Betriebszustand befinden, 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 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
```

7 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 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 man an sich an die PDB anmelden, obwohl kein Listener läuft, und somit auf den Service nicht zugegriffen werden kann, kann man sich auch direkt an die CDB anmelden und dann in die PDB wechseln. Das Anmelden an die CDB

erfolgt dann direkt auf dem Server durch eine Bequeath-Verbindung durch das Setzen der ORACLE_SID. Mit dem Kommando ALTER SESSION kann dann die Verbindung mit der PDB aufgenommen werden:

```
$ 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? 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 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.

8 Initialisierungsparameter

Instanzparameter steuern in vielerlei Hinsicht das Verhalten des Oracle Datenbanksystems. Dabei gibt es Instanzparameter, die global für die gesamte Container Datenbank (CDB) gelten und andere, die für jede einzelne Pluggable Datenbank (PDB) eingestellt werden können.

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 hinsichtlich der Gesamtgröße der SGA auftauchen.

Sie können aber durchaus auf PDB-Ebene Mindestwerte setzen, damit die Anforderungen, die durch die Anwendung gestellt werden, auch garantiert erfüllt werden. Dabei sind aber Grenzen zu beachten, wie zum Beispiel beim Parameter `DB_CACHE_SIZE`, der maximal halb so groß eingestellt werden darf, wie es auf CDB-Ebene der Fall ist. Die genauen Regeln können Sie dem Handbuch entnehmen und würden den Rahmen dieses Dojos sprengen. Folgende Initialisierungsparameter können Sie im Bereich Memory auf PDB-Ebene setzen:

```
DB_CACHE_SIZE
SHARED_POOL_SIZE
PGA_AGGREGATE_LIMIT
PGA_AGGREGATE_TARGET
SGA_MIN_SIZE
SGA_TARGET
```

Diese Parameter können auf PDB-Ebene nur dann erfolgreich gesetzt werden, wenn auf CDB-Ebene gilt:

- Der Parameter NONCDB_COMPATIBLE=false
- Der Parameter MEMORY_TARGET=0 oder ist gar nicht gesetzt

Die Parameter für die Einstellung der Hintergrund-Prozesse (zum Beispiel PROCESSES) werden 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 in einem späteren Kapitel 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.

9 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:

```
SQL_in_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_in_CDB> 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:

```
SQL_in_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(128)
OPEN_MODE                              VARCHAR2(10)
RESTRICTED                             VARCHAR2(3)
OPEN_TIME                              TIMESTAMP(3) WITH TIME ZONE
CREATE_SCN                             NUMBER
TOTAL_SIZE                             NUMBER
BLOCK_SIZE                             NUMBER
RECOVERY_STATUS                        VARCHAR2(8)
SNAPSHOT_PARENT_CON_ID                 NUMBER
```

```
APPLICATION_ROOT                VARCHAR2(3)
APPLICATION_PDB                 VARCHAR2(3)
APPLICATION_SEED                VARCHAR2(3)
APPLICATION_ROOT_CON_ID         NUMBER
APPLICATION_ROOT_CLONE          VARCHAR2(3)
PROXY_PDB                       VARCHAR2(3)
LOCAL_UNDO                      NUMBER
UNDO_SCN                        NUMBER
UNDO_TIMESTAMP                  DATE
CREATION_TIME                   DATE
DIAGNOSTICS_SIZE                NUMBER
 PDB_COUNT                      NUMBER
 AUDIT_FILES_SIZE               NUMBER
 MAX_SIZE                       NUMBER
 MAX_DIAGNOSTICS_SIZE           NUMBER
 MAX_AUDIT_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(10)
CREATION_SCN                              NUMBER
VSN                                       NUMBER
LOGGING                                   VARCHAR2(9)
FORCE_LOGGING                             VARCHAR2(3)
FORCE_NOLOGGING                           VARCHAR2(3)
APPLICATION_ROOT                          VARCHAR2(3)
APPLICATION_PDB                           VARCHAR2(3)
 APPLICATION_SEED                         VARCHAR2(3)
 APPLICATION_ROOT_CON_ID                  NUMBER
 IS_PROXY_PDB                             VARCHAR2(3)
 CON_ID                          NOT NULL  NUMBER
 UPGRADE_PRIORITY                         NUMBER
 APPLICATION_CLONE                        VARCHAR2(3)
 FOREIGN_CDB_DBID                         NUMBER
 UNPLUG_SCN                               NUMBER
 FOREIGN_PDB_ID                           NUMBER
 CREATION_TIME                   NOT NULL  DATE
 REFRESH_MODE                             VARCHAR2(6)
 REFRESH_INTERVAL                         NUMBER
```

10 Benutzerverwaltung

In der Multitenant 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.

10.1 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.

10.2 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 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.

10.3 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

11 Sicherheit im Betrieb mit Lockdown Profilen

Durch die Konsolidierung mehrerer Pluggable Datenbanken in Container Datenbanken stellen sich verschiedene Fragen hinsichtlich des Umgangs mit Sicherheitsanforderungen im Betrieb dieser Datenbanken. Wie schon im vorherigen Kapitel beschrieben, gibt es zwei grundlegende Arten von Benutzern: globale und lokale Benutzer, die jeweils Privilegien in den PDBs zugewiesen bekommen.

Zusätzlich stellt sich aber die Frage, inwieweit ein Administrator einer Pluggable Datenbank privilegierte Kommandos ausführen kann, oder welche Datenbankoptionen durch eine PDB verwendet werden dürfen. Gerade Letzteres kann in einer konsolidierten Umgebung für ein Lizenzmanagement von großem Interesse sein.

Auch der Zugriff auf das Netzwerk mittels der Packages UTL_TCP, UTL_HTTP und andere, sowie der Zugriff auf das Betriebssystem mittels zum Beispiel UTL_FILE sollte bei Sicherheitsüberlegungen mit einbezogen werden.

Zu diesem Zweck können Sie sogenannte Lockdown Profile definieren. Diese erlauben oder verbieten die Nutzung von

- ***Datenbank-Optionen***
  Zur Zeit werden dabei die Optionen ‚Partitioning' und ‚Database Queing' unterstützt
- ***Datenbank-Features***
  Es werden eine Reihe von Features unterstützt, die im Handbuch beschrieben sind. Hier ein Auszug: `AWR_ACCESS, COMMON_SCHEMA_ACCESS, NETWORK_ACCESS, OS_ACCESS`
- ***SQL-Statements***
  Sie können die Nutzung der folgenden SQL-Statements steuern: `ALTER DATABASE, ALTER PLUGGABLE DATABASE, ALTER SESSION, ALTER SYSTEM`

Die Lockdown Profiles werden auf der Ebene der CDB oder Applikationscontainer erstellt, verbunden mit einem Namen. Auf der Ebene der PDB kann maximal ein Lockdown Profile, welches in der CDB existieren muß, per Instanzparameter aktiviert werden.

Sie können Lockdown Profiles auch in der Seed-Datenbank eines Applikationscontainers erstellen. Diese Einstellung wird dann automatisch bei Erstellung einer Anwendungs-PDB berücksichtigt.

Es gibt drei vorgefertigte Lockdown Profile, die jedoch keine Definition beinhalten und noch mit Leben gefüllt werden müssen, um genutzt zu werden.

Sie können in einer CDB die vorhandenen Lockdown Profile abfragen, mit

```
SQL> set linesize 200
col profile_name format a20
col status format a7
col rule_type format a9
col rule format a15
col clause_option format a20
col clause format a20
col option_value format a20
select * from dba_lockdown_profiles;

PROFILE_NAME RULE_TYPE RULE CLAUSE CLAUSE_OPTION OPTION_VALUE STATUS
------------ --------- ---- ------ ------------- ------------ ------
PRIVATE_DBAAS                                                  EMPTY
PUBLIC_DBAAS                                                   EMPTY
SAAS                                                           EMPTY
```

Die oben gezeigten Lockdown Profile `SAAS`, `PRIVATE_DBAAS` und `PUBLIC_SAAS` werden Sie als Standardprofile vorfinden.

11.1 ERSTELLEN VON LOCKDOWN PROFILEN

Sie können in einer CDB ein Lockdown Profile recht einfach anlegen mit

```
SQL> CREATE LOCKDOWN PROFILE neues_profil;
```

Dazu brauchen Sie das Privileg `CREATE LOCKDOWN PROFILE`. Sie erhalten ein leeres Profile.

Ein bestehendes Lockdown Profile kann kopiert werden mit

```
SQL> CREATE LOCKDOWN PROFILE neues_profil FROM altes_profil;
```

Dabei wird das bestehende Profil einmalig kopiert. Zukünftige Änderungen am alten Profil werden nicht im neuen Profil angewendet.

Sie können ein neues Profil aber auch so anlegen, daß die Einstellungen eines anderen Profils immer aktuell eingebunden werden, also Änderungen am eingebundenen Profil dynamisch auch im neuen Profil gelten. Dazu benutzen Sie

```
SQL> CREATE LOCKDOWN PROFILE neues_profil INCLUDING anderes_profil;
```

Damit können umfangreiche Sicherheitskonzepte realisiert werden.

11.2 LOCKDOWN PROFILE MIT INHALT FÜLLEN

Sie füllen ein leeres Lockdown Profil mit `ALTER LOCKDOWN PROFILE` mit Inhalt. Dabei können Sie Features, Optionen oder Statements ein- bzw. abschalten. Hier einige Beispiele:

Abschalten der Datenbankoption „Partitioning“

```
ALTER LOCKDOWN PROFILE wdg DISABLE OPTION = ('Partitioning');
```

Abschalten des Datenbankfeatures XDB

```
ALTER LOCKDOWN PROFILE wdg DISABLE FEATURE = ('XDB_PROTOCOLS');
```

Abschalten des Statements ALTER SYSTEM

```
ALTER LOCKDOWN PROFILE wdg DISABLE STATEMENT = ('ALTER SYSTEM');
```

Eine Auflistung der Features, Optionen und Statements, die über Lockdown Profiles ein- bzw. abgeschaltet werden können, finden Sie im SQL Handbuch der Datenbank.

Sie sollten jedoch vorsichtig mit diesem Feature umgehen. Beispielsweise ist es möglich, das SQL-Kommando `SELECT` in einer PDB abzuschalten. Sobald dieses geschehen ist, ist ein Arbeiten mit der PDB praktisch nicht mehr möglich, da jede neue Datenbankverbindung zunächst Lesevorgänge durchführt.

11.3 AKTIVIEREN VON LOCKDOWN PROFILEN

Pro PDB können Sie maximal ein Lockdown Profile aktivieren, indem Sie dort in einer Datenbanksitzung den Instanzparameter `PDB_LOCKDOWN` setzen, der aber nur für die betreffende PDB wirkt:

```
ALTER SYSTEM SET PDB_LOCKDOWN = wdg;
```

Dazu benötigen Sie das `ALTER SYSTEM` Privileg in der PDB.

11.4 ANWENDUNG EINES LOCKDOWN PROFILES

In einem einfachen Beispiel soll die Verwendung von Lockdown Profilen gezeigt werden. Dabei soll die Verwendung von Optionen in der Datenbank unterbunden werden, sowie die Nutzung von `ALTER SYSTEM` Kommandos. Damit können innerhalb der PDB dann auch keine Initialisierungsparameter mehr geändert werden:

DROP LOCKDOWN PROFILE wdg;

CREATE LOCKDOWN PROFILE wdg;

ALTER LOCKDOWN PROFILE wdg DISABLE OPTION ALL;

ALTER LOCKDOWN PROFILE wdg DISABLE STATEMENT = ('ALTER SYSTEM');

select profile_name,rule_type,rule,status from dba_lockdown_profiles;

```
PROFILE_NAME   RULE_TYPE    RULE            STATUS
------------   ---------    -------------   -------
PRIVATE_DBAAS                               EMPTY
PUBLIC_DBAAS                                EMPTY
```

```
SAAS                                    EMPTY
WDG             OPTION      ALL         DISABLE
WDG             STATEMENT   ALTER       SYSTEM      DISABLE
```

Beachten Sie, dass auch das Privileg `ALTER SYSTEM` innerhalb der PDB deaktiviert wird. Sie schalten also das Lockdown Profile in der PDB ein mit

```
SQL> alter system set pdb_lockdown=wdg;
System altered.
```

Ab sofort können keine `ALTER SYSTEM` Kommandos mehr innerhalb der PDB ausgeführt werden. Beachten Sie, dass damit auch die Möglichkeit, das Lockdown Profile innerhalb der PDB zu wechseln nicht mehr möglich ist:

```
SQL> alter system set pdb_lockdown=SAAS;
alter system set pdb_lockdown=SAAS
*
ERROR at line 1:
ORA-01031: insufficient privileges
```

12 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 Ressource 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 werden auf dieser obersten Ebene folgende Elemente betrachtet:

- ***„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%.

13 Undo Management

Undo-Daten, die zum Beispiel für ein Rollback einer nicht beendeten Transaktion verwendet werden, werden in einem Undo Tablespace einer Oracle Datenbank gespeichert.

Für Multitenant bedeutet das mit Oracle 12c Release 1, dass es auf der Ebene der CDB diesen Undo Tablespace gibt. Dieses wird „Shared Undo Mode“ genannt. Diese

Daten werden bei einem Transport einer PDB von einer CDB zu einer anderen nicht mitgenommen. Da die Undo Daten auch für die Funktionalität des Flashbacks benötigt werden, ist ein Flashback nach einem Transport also nicht möglich.

Ab Oracle 12c Release 2 gibt es die Alternative des „Local Undo Mode". Dabei bekommt jede PDB einen eigenen Undo Tablespace. Dadurch werden auch die Undo Daten bei einem Transport mitgenommen und alle Funktionalitäten, die auf diesen Undo Daten beruhen, können weiterhin verwendet werden. Auch in 12.2 ist der „Shared Undo Mode" der Standardmodus. Die Einstellung des „Local Undo Mode" erfolgt auf CDB-Ebene, entweder beim Anlegen einer CDB mit

```
SQL> CREATE DATABASE newcdb
...
LOCAL UNDO ON
```

oder der Modus wird nachträglich eingeschaltet mit

```
SQL> SHUTDOWN
SQL> STARTUP UPGRADE
SQL> ALTER DATABASE LOCAL UNDO ON;
SQL> SHUTDOWN
SQL> STARTUP
```

Danach können Sie noch den Undo Tablespace in der Seed-Datenbank manuell erstellen, um die Größe dieses Tablespaces für die PDBs vorzugeben, die über die Seed-Datenbank erstellt werden. Dazu gehen Sie folgendermaßen vor:

1. Verbinden Sie sich mit der CDB und führen Sie folgendes aus:

```
SQL> ALTER PLUGGABLE DATABASE PDB$SEED OPEN READ WRITE FORCE;
SQL> ALTER SESSION SET CONTAINER=PDB$SEED;
```

2. Erstellen Sie einen Undo Tablespace in der Seed Datenbank

```
SQL> CREATE UNDO TABLESPACE undo4seed
       DATAFILE 'undo4seed_1.dbf'
       SIZE 20M AUTOEXTEND ON
       RETENTION GUARANTEE;
```

3. Verbinden Sie sich wieder mit der CDB und führen Sie folgendes aus:

```
SQL> ALTER SESSION SET CONTAINER=CDB$ROOT;
SQL> ALTER PLUGGABLE DATABASE PDB$SEED OPEN READ ONLY FORCE;
```

Jetzt ist die Seed Datenbank wieder im MOUNTED Stadium und kann für neue PDBs verwendet werden.

14 Umgang mit dem Automatic Workload Repository

Das Automatic Workload Repository (AWR) wurde mit der Version 10g der Oracle Datenbank eingeführt und muß über das Diagnostics Pack lizenziert sein. Dieses ist nur für die Enterprise Edition möglich und in den entsprechenden Oracle Cloud Service enthalten. Technisch werden dabei in der Datenbank regelmäßige Zugriffe direkt auf die SGA (System Global Area) der Datenbank durchgeführt, um sehr performant auf Nutzungsstatistiken der Datenbanksitzungen zuzugreifen. Die Daten werden in Tabellen im Tablespace SYSAUX gespeichert.

Das AWR bietet umfangreiche Daten für eine Analyse von Performance-Engpässen und sonstigen Problemen, die von den Anwendungen verursacht werden können und ist mittlerweile für viele eine unverzichtbare Basis für die Verwaltung von Oracle Datenbanken geworden.

Oracle Multitenant wurde mit der Version 12c Release 1 der Oracle Datenbank eingeführt und bei dieser Version wird das AWR ausschließlich auf der Ebene der CDB gespeichert, was auch weiterhin der Default ist. Auch die Erfassungsintervalle werden dabei ausschließlich in der CDB festgelegt. Da diese Daten nur in der CDB gespeichert sind, werde diese bei einem Transport einer PDB von einer CDB zur anderen CDB (Unplug und Plug) nicht mitgenommen.

Um Oracle Multitenant als Konsolidierungsplattform noch weiter zu verbessern, wurde mit der Version 12c Release 2 der Oracle Datenbank eine wichtige Veränderung beim AWR durchgeführt: die für eine PDB relevanten Daten des AWR können sowohl in der CDB, als auch in der PDB gespeichert werden. Dieses ist wichtig, zum Beispiel dann, wenn eine PDB von einer CDB zu einer anderen CDB verschoben werden soll und die AWR Daten dabei in der PDB erhalten bleiben sollen.

In einer Multitenant Umgebung speichert das AWR generelle, instanzspezifische, als auch PDB spezifische Daten.

- Die CDB speichert grundsätzlich die Daten aller dieser drei Datenklassen im Tablespace SYSAUX der CDB. Das automatische Erstellen der Snapshots ist automatisch eingerichtet mit einem Erfassungsintervall von 60 Minuten.
- Die PDB kann zusätzlich die PDB spezifischen AWR Daten im eigenen Tablespace SYSAUX speichern. Bei einem Transport einer PDB über Unplug und Plug werden die AWR Daten also mitgenommen.

14.1 AUTOMATISCHE PDB SEITIGE AWR-SNAPSHOTS AKTIVIEREN

Per Default ist die automatische Erstellung von Snapshots für PDBs deaktiviert, die Aktivierung erfolgt in der jeweiligen PDB über

```
ALTER SYSTEM SET AWR_PDB_AUTOFLUSH_ENABLED=TRUE;  --(der Default is
FALSE).
```

Danach setzen Sie mit `DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS` eine Zeit für das Erfassungsintervall (in Minuten), also zum Beispiel:

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(
  interval  =>  60,
  retention =>  20160);
```

Sie können diese Einstellungen auch in Oracle Enterprise Manager Cloud Control vornehmen, sobald Sie das Plug-in der Version 13.2.2 für Oracle Datenbanken installiert haben.

AWR Snapshots können auch manuell (wie gewohnt mit dem PL/SQL Package `DBMS_WORKLOAD_REPOSITORY`) vorgenommen werden, wobei die Vorgehensweise sich nicht unterscheidet, ob auf CDB- oder PDB-Ebene. Es entscheidet einzig, an welcher Datenbank Sie angemeldet sind.

Jede PDB führt dabei eigene sogenannte Snapshots aus, entweder manuell, oder über ein regelmäßiges Intervall. Die

Kommandos dazu unterscheiden sich nicht vom bislang gewohnten Umgang mit dem AWR, nur dass die in einer Datenbanksitzung in einer PDB ausgeführt werden. Bei einem Transport einer PDB über Unplug und Plug, bzw. PDB Relocate werden die AWR-Daten also mitgenommen.

Die in einer PDB erstellten Snapshots sind unabhängig von denen, die in der CDB erstellt werden und haben auch eigene Snapshot IDs.

Über die in einem vorherigen Kapitel beschriebenen PDB Lockdown Profiles kann die Erstellung von AWR Snapshots auf PDB-Ebene verhindert werden mit

```
ALTER LOCKDOWN PROFILE profile_name DISABLE FEATURE=('AWR_ACCESS');
```

Dieses sollte aber nur verwendet werden, wenn die Nutzung des AWR für einzelne PDBs oder auf PDB Ebene generell verhindert werden soll. Sollte das Diagnostics Pack für die CDB nicht lizenziert sein, empfiehlt es sich die Funktionalität des AWR generell abzuschalten mit

```
ALTER SYSTEM SET control_management_pack_access=none;
```

Mit diesem Kommando wird die Nutzung des Diagnostics Pack wie gewohnt sowohl für die CDB, als auch für alle dort betriebenen PDBs ausgeschaltet.

14.2 PDB SEITIGE AWR-SNAPSHOTS MANUELL ERSTELLEN

Sie erstellen einen manuellen AWR Snapshot mit

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT('TYPICAL');
```

wobei der Parameter den Umfang an Daten angibt, der dadurch ermittelt wird. Der Wert ‚TYPICAL' entspricht auch dem Default auf CDB Ebene.

In Cloud Control können Sie einen Snapshot auf PDB-Ebene auch graphisch erstellen, indem Sie den Button "*Create*" nutzen (Navigation zur Seite: Performance -> AWR -> AWR Administration -> Snapshots (auf die Anzahl klicken)).

14.3 AWR REPORTS

AWR Reports können Sie wie gewohnt mit den vorbereiteten Skripten erstellen. Wenn Sie den AWR Report für die Datenbank erstellen möchten bei der Sie angemeldet sind (CDB oder PDB), verwenden Sie

```
@$ORACLE_HOME/rdbms/admin/awrrpt.sql
```

Wenn Sie einen PDB spezifischen AWR Report erstellen möchten, während Sie an der CDB angemeldet sind, verwenden Sie

```
@$ORACLE_HOME/rdbms/admin/awrrpti.sql
```

und geben im Verlauf des Skripts den Namen der PDB ein.

In Cloud Control können Sie einen AWR Report auf PDB-Ebene auch graphisch erstellen, indem Sie das Kommando „View Report“ in der Select Liste nutzen (Navigation zur Seite: Performance -> AWR -> AWR Administration -> Snapshots (auf die Anzahl klicken)).

14.4 DATA DICTIONARY

Beim Zugriff auf die Daten im AWR gilt, dass wenn innerhalb einer PDB darauf zugegriffen wird, automatisch nur die PDB spezifischen Daten angezeigt werden. Sowohl SQL-Skripte zum Erstellen der AWR-Reports, also auch die bekannten Data Dictionary Views bzgl. AWR (z.B. `DBA_HIST_SYSSTAT`) können auf beiden Ebenen verwendet werden.

- ***DBA_HIST* (also zum Beispiel DBA_HIST_SYSSTAT)***
  zeigt die AWR Daten an, die in der CDB gespeichert sind. Wird diese View von einer PDB aufgerufen, werden nur die Daten angezeigt, die die aufrufende PDB betreffen (aber in der CDB gespeichert sind).
- ***DBA_HIST_CON* (also zum Beispiel DBA_HIST_CON_SYSSTAT)***
  zeigt mehr Details an als DBA_HIST*, aber auf dem gleichen Level, also die Daten, die in der CDB gespeichert sind.

- ***CDB_HIST* (also zum Beispiel CDB_HIST_ SYSSTAT oder CDB_HIST_CON_SYSSTAT)***
  zeigt alle Daten an, die in den PDBs gespeichert sind. Von der CDB aufgerufen, werden alle Daten aller PDBs angezeigt. Aus einer PDB heraus sieht man natürlich nur die PDB-eigenen Daten.
- ***AWR_ROOT* (also zum Beispiel AWR_ROOT_SYSSTAT) – Neu seit Oracle 12c Release 2***
  ist ein Äquivalent zu DBA_HIST*.
- ***AWR_PDB* (also zum Beispiel AWR_ROOT_SYSSTAT) – Neu seit Oracle 12c Release 2***
  zeigt die lokalen AWR Daten an, je nachdem, wie man verbunden ist. Bei einer Verbindung mit einer PDB werden die lokalen AWR Daten für die PDB angezeigt, bei einer Verbindung mit der CDB die in der CDB gespeicherten Daten.

15 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.

15.1 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 Redo-Logs 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
```

15.2 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
```

15.3 FLASHBACK EINER PDB

Sie können eine einzelne PDB auch mit Flashback zurücksetzen. Voraussetzung für eine einfache Durchführung ist, dass die CDB im „Local Undo Mode“ betrieben wird, die PDBs also eigene Undo Tablespaces haben (vgl. Kapitel „Undo Management“). Auch der ARCHIVELOG Modus muß auf der Ebene der CDB eingeschaltet sein.

Ob Ihre CDB mit dem „Local Undo Mode“ betrieben wird, können Sie abfragen mit

```
select name , local_undo from v$pdbs;
```

Der Wert 1 in der Spalte „local_undo“ zeigt den „Local Undo Mode“ an, während der Wert 0 für den „Shared Undo Mode“ steht.

Sind beide Voraussetzungen erfüllt, führen Sie ein Flashback durch, indem Sie sich an die geöffnete CDB mit dem Privileg `SYSBACKUP` oder `SYSDBA` anmelden. Schließen Sie die PDB, die zurückgesetzt werden soll:

```
ALTER PLUGGABLE DATABASE pdb1 CLOSE;
```

Jetzt können Sie die PDB ganz einfach zurücksetzen mit einem der folgenden Kommandos:

```
FLASHBACK PLUGGABLE DATABASE pdb1 TO SCN 24368;
FLASHBACK PLUGGABLE DATABASE pdb1 TO RESTORE POINT testbeginn;
```

```
FLASHBACK PLUGGABLE DATABASE pdb1 TO CLEAN RESTORE POINT
testbeginn;
```

Die PDB wird dann noch mit der Option `RESETLOGS` gestartet:

```
ALTER PLUGGABLE DATABASE pdb1 OPEN RESETLOGS;
```

Falls sich Ihre PDB in einer CDB befindet, die nicht im „Local Undo Mode“ betrieben wird, muß eine Hilfsdatenbank erstellt werden. Das Prozedere wird in diesem Fall also aufwendiger und ist im Handbuch (siehe Kapitel „Weitere Informationen“) beschrieben.

16 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 Redo-Log-Dateien. Da bei dem neuen Konzept der Pluggable Database die Redo-Log-Dateien gemeinsam für alle PDBs auf CDB-Ebene geführt werden, ergibt sich daraus, dass auch die Standby-Datenbank immer auf der kompletten CDB basiert.

Wenn eine Standby-Lösung für einzelne PDBs gewünscht wird, müssen statt Data Guard sogenannte Refreshable Clone PDBs eingesetzt werden, die im dazugehörigen Kapitel beschrieben werden.

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 Informa-

tionen 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 „STANDBYS=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 „STANDBYS=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 PLUGGABLE 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.

17 Patching

Das Einspielen von Patches für Oracle Datenbanken erfolgt mit dem Oracle Patch Tool OPatch. Dabei wird die Software im ORACLE_HOME teilweise durch neue Dateien ersetzt. Um diese wirksam werden zu lassen, muß die Oracle Datenbank einmal durchgestartet werden. Damit ist der Vorgang des Patchens aber noch nicht abgeschlossen, denn in aller Regel müssen auch noch Anpassungen in der Datenbank vorgenommen werden. Dazu gibt es das Tool „datapatch“, welches diese Anpassungen vornimmt.

Um Downtime zu vermeiden gibt es die Vorgehensweise des „out-of-place“-Patchings. Dabei wird eine Kopie des aktuellen ORACLE_HOMEs erstellt und dann diese Kopie gepatcht. Im Anschluß daran wird die Datenbank heruntergefahren, mit dem neuen ORACLE_HOME hochgefahren und dann wiederum der Inhalt der Datenbank mit „datapatch“ aktualisiert.

Auf Multitenant Datenbanken übertragen kann natürlich die gesamte CDB mit allen PDBs in einem Durchgang „in-place“ gepatcht werden, wobei während des Patchings eine Downtime zu verzeichnen ist. Eine höhere Verfügbarkeit ist aber gegeben, wenn eine Kopie des aktuellen ORACLE_HOMEs gepatcht und damit eine neue CDB als neue Betriebsplattform erstellt wird. Dann können alle PDBs von der

alten CDB nacheinander zur neuen CDB mittels Unplug/Plug oder PDB Relocate verschoben werden. Natürlich muß auch hier mit „datapatch“ der Inhalt der PDBs angepasst werden. Dabei prüft „datapatch“ immer , welche PDBs wirklich eine Anpassung benötigen, sodass der „Umzug“ der PDBs nacheinander erfolgen kann und „datapatch“ nur die PDBs bearbeitet, für die „datapatch“ noch nicht gelaufen ist.

17.1 PATCHING MITTELS UNPLUG/PLUG

Das folgende Beispiel zeigt, wie eine PDB per Unplug/Plug von einer ungepatchten CDB in eine gepatchte CDB verschoben und dann gepatcht wird.

Zunächst wird die PDB aus der ungepatchten CDB ausgehängt:

```
SQL> ALTER PLUGGABLE DATABASE mypdb CLOSE;

Pluggable database altered.

SQL> ALTER PLUGGABLE DATABASE mypdb UNPLUG INTO
     '/opt/oracle/oradata/MYCDB/mypdb.xml';

Pluggable database altered.

SQL> DROP PLUGGABLE DATABASE mypdb;

Pluggable database dropped.
```

Nachdem die Datenbankdateien und die XML-Datei, die das Kommando UNPLUG erzeugt hat, so kopiert oder verschoben wurden, dass die gepatchte CDB darauf zugreifen kann, prüfen Sie die Kompatibilität in der gepatchten CDB:

```
SQL> declare
  2  compatible constant varchar2(3) :=
  3  case dbms_pdb.check_plug_compatibility(
  4  pdb_descr_file => '/opt/oracle/oradata/PCDB/mypdb.xml')
  5  when true then 'YES'
  6  else 'NO'
  7  end;
  8  begin
  9  dbms_output.put('New PDB is compatible: ');
 10  dbms_output.put_line(compatible);
 11  end;
 12  /
New PDB is compatible: NO

PL/SQL procedure successfully completed.
```

```
SQL> SELECT type||' '||message||' '||action as Info FROM pdb_plug_in_violations
WHERE name = 'MYPDB';

INFO

-----------------------------------------------

ERROR DBRU bundle patch 180116 (DATABASE RELEASE UPDATE 12.2.0.1.180116):
Installed in the CDB but not in the PDB. Call datapatch to install in the PDB or
the CDB
```

Die Nicht-Kompatibilität ergibt sich also nur aus der Tatsache, dass die PDB noch nicht inhaltlich gepatcht wurde. Sie kann also eingeklinkt werden:

```
SQL> CREATE PLUGGABLE DATABASE mypdb USING
     '/opt/oracle/oradata/PCDB/mypdb.xml' NOCOPY
     SOURCE_FILE_NAME_CONVERT=('/opt/oracle/oradata/MYCDB',
                               '/opt/oracle/oradata/PCDB')
     TEMPFILE REUSE;

Pluggable database created.

SQL> SELECT name,open_mode FROM v$pdbs;

NAME         OPEN_MODE
------------ ----------
PDB$SEED     READ ONLY
MYPDB        MOUNTED
```

```
SQL> ALTER PLUGGABLE DATABASE mypdb OPEN;

Warning: PDB altered with errors.

SQL> SELECT name,open_mode FROM v$pdbs;
NAME        OPEN_MODE
---------- ----------
PDB$SEED    READ ONLY
MYPDB       READ WRITE
SQL> exit
```

Jetzt starten Sie das Tool „datapatch“ mit

```
cd $ORACLE_HOME/OPatch ; ./datapatch -verbose
SQL Patching tool version …
Connecting to database...OK
Note:  Datapatch will only apply or rollback SQL fixes for PDBs
       that are in an open state, no patches will be applied to
       closed PDBs.
       Please refer to Note: Datapatch: Database 12c Post Patch
       SQL Automation (Doc ID 1585822.1)
Bootstrapping registry and package to current versions...done
Determining current state...done
Current state of SQL patches:
Bundle series DBRU:
  ID 180116 in the binary registry and ID 180116 in PDB CDB$ROOT,
ID 180116 in PDB PDB$SEED, ID 180116 in PDB WDGPDB
```

```
Adding patches to installation queue and performing prereq checks...
Installation queue:
  For the following PDBs: CDB$ROOT PDB$SEED WDGPDB
    Nothing to roll back
    Nothing to apply
  For the following PDBs: MYPDB
    Nothing to roll back
    The following patches will be applied:
      27105253 (DATABASE RELEASE UPDATE 12.2.0.1.180116)
Installing patches...
Patch installation complete.  Total patches installed: 1
Validating logfiles...
Patch 27105253 apply (pdb MYPDB): SUCCESS
  logfile: ... (no errors)
SQL Patching tool complete
```

Das Patchen ist damit abgeschlossen.

17.2 PATCHING MITTELS PDB RELOCATE

Das folgende Beispiel zeigt, wie eine PDB per PDB Relocate von einer ungepatchten CDB in eine gepatchte CDB verschoben und dann gepatcht wird. Der Vorteil gegenüber eines Unplug/Plug liegt darin, dass die PDB während der Verschiebung von der ungepatchten CDB in die gepatchte CDB weiterhin online bleibt, auch für schreibende Datenbanksitzungen. Damit ist die Downtime also stark verringert. Streng genommen

darf man nicht von einem „Online Patchen“ sprechen, aber es kommt diesem schon sehr nahe.

Zunächst wird ein Database Link benötigt, um die beiden CDBs miteinander agieren zu lassen. Falls noch nicht geschehen, wird in der ungepatchten CDB ein Datenbankbenutzer erstellt und mit den notwendigen Rechten ausgestattet:

```
SQL> CREATE USER c##admin IDENTIFIED BY skeruicvbsjfvhskjsjkwiou;
User created.
SQL> GRANT connect,resource,create pluggable database,sysoper TO
c##admin CONTAINER=ALL;
Grant succeeded.
```

Dann wird der Database Link in der gepatchten CDB erstellt.

```
SQL> CREATE PUBLIC DATABASE LINK db_link_to_mycdb
CONNECT TO c##admin IDENTIFIED BY skeruicvbsjfvhskjsjkwiou
USING '(DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=MYCDB))
(ADDRESS=(PROTOCOL=TCP)(HOST=172.17.0.2)(PORT=1521)))';
Database link created.
```

Jetzt kann die PDB per Relocate von der ungepatchten CDB in die gepatchte CDB übertragen werden.

```
SQL> CREATE PLUGGABLE DATABASE mypdb FROM mypdb@db_link_to_mycdb
FILE_NAME_CONVERT = ('/opt/oracle/oradata/MYCDB',
                     '/opt/oracle/oradata/PCDB')
RELOCATE AVAILABILITY MAX;
Pluggable database created.

SQL> SELECT pdb_name,status FROM cdb_pdbs;

PDB_NAME    STATUS
---------   ----------
PDB$SEED    NORMAL
MYPDB       RELOCATING

SQL> ALTER PLUGGABLE DATABASE mypdb OPEN;

Warning: PDB altered with errors.

SQL> SELECT pdb_name,status FROM cdb_pdbs;

PDB_NAME    STATUS
---------   ----------
PDB$SEED    NORMAL
MYPDB       NORMAL

SQL> SELECT name,open_mode FROM v$pdbs;

NAME        OPEN_MODE
------      ----------
PDB$SEED    READ ONLY
MYPDB       READ WRITE

SQL> exit
```

Abschließend wird das Tool „datapatch“ gestartet.

```
cd $ORACLE_HOME/OPatch ; ./datapatch -verbose
```

Die Ausgabe entspricht der im vorherigen Abschnitt beschriebenen Ausführung.

18 Vergleich von CDB und PDB

Die folgende Tabelle gibt 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, Ausnahme Application-Container*
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*
Automatic Workload Repository	*Ja*	*Ja (Nein bis 12.1.0.2)*

Storage	CDB-Ebene	PDB-Ebene
System/Sysaux Tablespace	Ja	Ja
User Tablespace	Nur sinnvoll für administrative Zwecke	Ja
Temp Tablespace	Bedingt sinnvoll	Ja
Undo Tablepace	Ja	Möglich mit local Undo
Online Redo-Logs	Ja	Nein
Benutzerobjekte und Daten	Nur sinnvoll für administrative 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. Lokale 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
Flashback	Ja, inklusive aller PDBs	Ja
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.

19 Lizenzierung

Die Nutzung von Oracle Multitenant erfordert die separate Lizenzierung der gleichnamigen Datenbankoption „Oracle Multitenant“. Dieses betrifft alle CDBs, die mehr als eine PDB betreiben, mit einer Ausnahme: In einer CDB der 18c Express Edition dürfen bis zu 3 PDBs genutzt werden.

Wenn eine CDB nur exakt eine PDB betreibt, ist die separate Lizenzierung der Datenbankoption nicht erforderlich. In diesem Fall spricht man auch von „Single Tenant“.

Die Datenbankoption „Oracle Multitenant“ ist nur für die Enterprise Edition der Oracle Datenbank lizenzierbar. Im Falle der Standard Edition kann also nur ein Betrieb nach dem „Single Tenant“ Modell vorgenommen werden.

Derzeit ist der Betrieb von Oracle Datenbanken auch mit der Non-CDB-Architektur (also die Architektur, die bis Oracle 11 genutzt wurde) möglich. Diese Non-CDB-Architektur ist jedoch im Status „Deprecated“ und wird in Zukunft auslaufen. Ein genaues Datum dazu wurde von Oracle aber noch nicht genannt.

Die Cloud-Angebote von Oracle setzen im Bereich Datenbankservices immer per Default auf die x-Tenant Architektur. Die folgende Auflistung zeigt, welche Architektur im jeweiligen PaaS-Angebot enthalten ist (Stand Dezember 2018):

- Standard Edition: Single Tenant
- Enterprise Edition: Single Tenant
- Enterprise Edition High Performance: Multitenant
- Enterprise Edition Extreme Performance: Multitenant

20 Weitere Informationen

Weitere Informationen stehen Ihnen unter https://blogs.oracle.com/coretec/Dojo7 zur Verfügung.

Zugriff auf die komplette
Oracle Dojo-Bibliothek unter
https://tinyurl.com/dojos-online

Herausgeber: Roland Aussermeier, Oracle Deutschland B.V.

Design: volkerstegmaier.de // Druck: Stober GmbH, Eggenstein

ORACLE®

ORACLE®