Kapitel 2

Die Oracle-Datenbankarchitektur

Die Kenntnis der Oracle-Datenbankarchitektur ist für den Administrator sehr wichtig. Nur wer den Aufbau, die Zusammenhänge sowie die internen Prozessabläufe kennt, ist in der Lage, Architekturen zu planen und zu implementieren, die täglichen Supportanforderungen zu meistern und Troubleshooting zu betreiben.

Die Architektur der Oracle-Datenbank ist sehr komplex, und es ist eine längere praktische Erfahrung erforderlich, um sie in ihrer Gesamtheit kennenzulernen und zu beherrschen. Je länger und intensiver Sie sich mit dem Thema beschäftigen, desto umfangreicher wird Ihr Wissen und desto plausibler werden die Zusammenhänge.

2.1 Übersicht über die Architektur

Die wohl am häufigsten verwendeten Begriffe sind *Datenbank* und *Instanz*. Die Datenbank ist die Zusammenfassung aller Dateien wie Datafiles, Kontrolldateien, Log-Dateien, `SPFILE` usw., also alle Objekte, die sich auf der Disk befinden. Dagegen beschreibt die Instanz alle Strukturen, die sich im Hauptspeicher befinden, wie z.B. Buffer Cache, Shared Pool oder die Hintergrundprozesse. Zu jeder Datenbank gehört mindestens eine Instanz. In einer Real-Application-Clusters-Umgebung benutzen mehrere Instanzen dieselbe Datenbank.

2.1.1 Die Struktur der Datenbank

Eine Oracle-Datenbank wird nach logischen und physischen Strukturen unterschieden. Logische Strukturen sind z.B. Schemata oder Tablespaces. Physische Strukturen sind Dateien und Einheiten, die auf Betriebssystemebene sichtbar und greifbar sind. Dazu gehören Datafiles, Kontrolldateien oder Datenblöcke.

Die Dateien der Datenbank können im Dateisystem des Datenbankservers, in einem Cluster-Dateisystem, im Oracle-Automatic-Storage-Management (ASM) oder als Raw Devices gespeichert werden.

Zu den grundlegenden Strukturen gehört Folgendes:

- *Tablespaces* sind Container für Objekte wie Tabellen oder Indexe. Diese Objekte können nach verschiedenen Kriterien wie Applikationslogik oder Performance gezielt in verschiedene Tablespaces gelegt werden. In der `SYSTEM`-Tablespace befindet sich unter anderem der Datenbankkatalog. Die `SYSAUX`-Tablespace ist für Repositories oder Tabellen von Werkzeugen vorgesehen. Die `TEMPORARY`-Tablespace dient der Aufnahme von temporären Segmenten. Eine Tablespace kann aus einem oder mehreren Datafiles bestehen.
- *Datenblöcke* sind die kleinste Speichereinheit einer Oracle-Datenbank. Zulässig in Oracle 19c sind Größen von 2, 4, 8, 16 und 32 KB.
- *Extents* sind Gruppen von Datenblöcken, die zusammenhängend gespeichert werden.

- *Segmente* sind Gruppen von Extents, die eine logische Struktur bilden. Alle Extents einer Tabelle, eines Index oder eines Clusters bilden ein Segment.

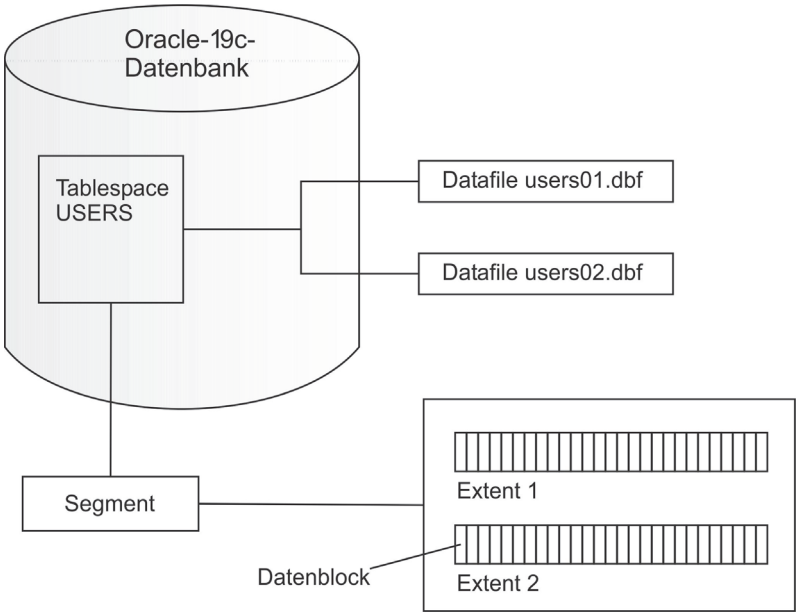

Abb. 2.1: Die Struktur der Oracle-Datenbank

Zu einer Datenbank gehören folgende Dateitypen:

- Datafiles
- Tempfiles
- Kontrolldateien
- Online-Redo-Log-Dateien
- Server-Parameter-File (`SPFILE`)
- Passwordfile
- Archived-Redo-Log-Dateien (optional)
- Flashback-Log-Dateien (optional)
- Block-Change-Tracking-File (optional)
- Alertlog- und Incident-Dateien

Datafiles sind die physische Umsetzung von Tablespaces. Eine Tablespace besteht aus einem oder mehreren Datafiles. Ein Datafile ist genau einer Tablespace zugeordnet.

Tempfiles sind Dateien, die temporäre Tablespaces verkörpern. Sie enthalten bei geschlossener Datenbank keine relevanten Daten und werden beim Öffnen der Datenbank automatisch neu angelegt, falls sie nicht existieren.

Die *Kontrolldatei* speichert Informationen über die physische Datenbankstruktur. Sie enthält unter anderem den Datenbanknamen, die Datenbankidentifizierungsnummer (DBID) und

Namen, Speicherort sowie Status der Datafiles, Tempfiles und Online-Redo-Log-Dateien. Aus Sicherheitsgründen sollte mindestens ein Spiegel der Kontrolldatei angelegt werden.

Online-Redo-Log-Dateien sind das Transaktionslog der Datenbank. Sie garantieren die Transaktionssicherheit und werden für Recovery-Zwecke benötigt.

Im *Server-Parameter-File* befinden sich die Initialisierungsparameter der Datenbank.

Das *Passwordfile* enthält die Passwörter der privilegierten Benutzer. Es wird benötigt, wenn die Datenbank nicht geöffnet ist.

Archived-Redo-Log-Dateien sind archivierte Online-Redo-Log-Dateien. Sie gewährleisten die Wiederherstellung der Datenbank zu einem beliebigen Zeitpunkt, da Online-Redo-Log-Dateien zyklisch überschrieben werden. Archived-Redo-Log-Dateien werden geschrieben, wenn die Datenbank im `ARCHIVELOG`-Modus betrieben wird.

Flashback-Log-Dateien sind optional und werden nur erzeugt, wenn das Flashback-Database-Feature aktiviert ist. Mit ihrer Hilfe ist es möglich, die Datenbank auf einen beliebigen Zeitpunkt in der Vergangenheit zurückzusetzen, ohne dass eine Sicherung eingespielt werden muss.

Das *Block-Change-Tracking-File* speichert Informationen über die Datenblöcke, die seit der letzten Vollsicherung geändert wurden, und dient als Index für inkrementelle Sicherungen. Es muss explizit aktiviert werden.

In Oracle 19c gibt es elf Segmenttypen. Der größte Anteil fällt auf Daten- und Indexsegmente sowie `UNDO`- und `TEMP`-Segmente. Weiterhin gibt es spezielle Segmente für Large Objects, Cluster und Partitionen. Die Segmenttypen sind:

- `CLUSTER`
- `INDEX`
- `INDEX PARTITION`
- `LOB PARTITION`
- `LOB INDEX`
- `LOB SEGMENT`
- `NESTED TABLE`
- `ROLLBACK SEGMENT`
- `TABLE`
- `TABLE PARTITION`
- `TYPE2 UNDO`

Oracle-Datenblöcke

Der Oracle-Datenblock ist die kleinste Einheit in der Hierarchie der Speicherstrukturen. Er enthält die Sätze von Tabellen, Indexeinträge oder Large Objects (LOB). Die Standardgröße der Datenbank wird beim Erstellen festgelegt und kann anschließend nicht mehr geändert werden. Es besteht jedoch die Möglichkeit, Tablespaces mit unterschiedlichen Blockgrößen anzulegen. Viele Systeme benutzen eine Blockgröße von 4 KB oder 8 KB. Für große oder Data-Warehouse-Datenbanken ist eine Blockgröße von 16 KB empfohlen, um einen Performance-Gewinn insbesondere beim Lesen der Daten zu erzielen. Die Blockgröße der Datenbank sollte stets ein ganzzahliges Vielfaches der Blockgröße des Betriebssystems sein. Bei

der Wahl der Blockgröße ist außerdem zu beachten, dass es bei einer zu hoch gewählten Größe zu Konkurrenzsituationen und damit zu Wartezeiten kommen kann, da sich die meisten internen Operationen auf Blockebene abspielen.

Ein normaler, unkomprimierter Datenblock besteht aus einem Kopf (Blockheader) und einem Rumpf.

Abb. 2.2: Aufbau eines Datenblocks

Der Header enthält allgemeine Information wie die Speicheradresse auf der Disk und den Blocktyp. Blocktypen, die der Transaktionsverwaltung unterliegen, enthalten zusätzlich historische Informationen zu den durchgeführten Transaktionen. Im Tabellenverzeichnis befinden sich Informationen über die Tabellen der gespeicherten Sätze. Das Zeilenverzeichnis beschreibt die Positionen der Sätze (Zeilen), die im Block gespeichert sind.

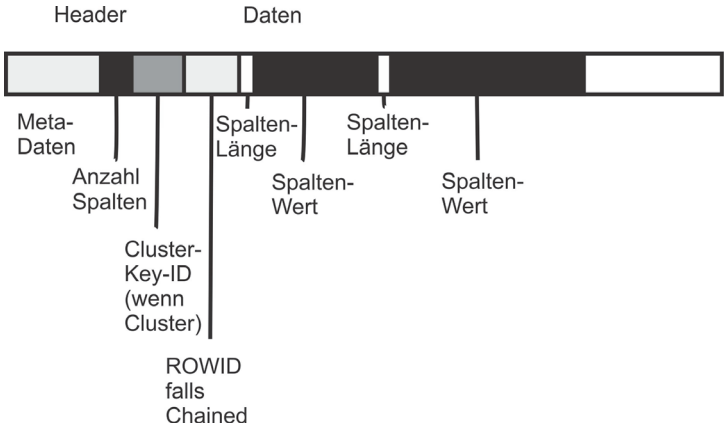

Abb. 2.3: Struktur eines Datensatzes im Block

Passt ein Datensatz nicht komplett in einen Block, dann wird er auf zwei Blöcke aufgeteilt. Man spricht dann von *Chained Rows*. Die `ROWID` des Blocks, in dem der Datensatz fortgesetzt wird, verweist dann auf den nächsten Block. Chained Rows wirken sich negativ auf die Performance aus und sollten im großen Umfang vermieden werden.

Wenn Sie sich einen Datenblock etwas genauer anschauen möchten, besteht die Möglichkeit, einen Dump zu erzeugen. Im Beispiel in Listing 2.1 wird eine kleine Tabelle mit nur einem Satz verwendet. Die SQL-Abfrage liefert die Blocknummer 52608 zurück. Dort befindet sich der Segment-Header. Der folgende Block ist der erste Datenblock. Die Ausgabe des Dumpfiles erfolgt im Diagnostic-Verzeichnis der Datenbank, da, wo die Alert-Log-Datei liegt.

```
SQL> SELECT file_id, block_id, blocks
  2  FROM dba_extents
  3  WHERE segment_name = 'TEST';
   FILE_ID   BLOCK_ID     BLOCKS
---------- ---------- ----------
         1      52608          8
SQL> ALTER SYSTEM DUMP DATAFILE 1 BLOCK 52609;
System wurde geändert.
```

Listing 2.1: Dump für einen Datenblock erstellen

Neben vielen interessanten Informationen aus dem Block-Header liefert die Dump-Datei den kompletten Inhalt des Blocks.

```
Dump of memory from 0x00007FA35294EE00 to 0x00007FA352950E00
7FA35294EE00 0000A206 0040CD81 000DB189 06010000
[......@.........]
7FA35294EE10 0000C4A5 00000001 00005B01 000DB17C
[.........[..|...]
7FA35294EE20 00000000 00030002 00000000 00030004
[................]
7FA35294EE30 000002F8 01416DA1 00210089 00002001
[.....mA...!.. ..]
7FA35294EE40 000DB189 00000000 00000000 00000000
[................]
7FA35294EE50 00000000 00000000 00000000 00010100
[................]
7FA35294EE60 0014FFFF 1F5C1F70 00001F5C 1F700001
[....p.\.\.....p.]
7FA35294EE70 00000000 00000000 00000000 00000000
[................]
        Repeat 500 times
7FA352950DC0 00000000 00000000 00000000 0202012C
[............,...]
7FA352950DD0 412902C1 41414141 41414141 41414141
[..)AAAAAAAAAAAA]
7FA352950DE0 41414141 41414141 41414141 41414141
```

```
[AAAAAAAAAAAAAAAA]
7FA352950DF0 41414141 41414141 41414141 B1890601
[AAAAAAAAAAAA....]
```

Listing 2.2: Dump eines Datenblocks

Über die `ROWID` kann jeder Satz in der Datenbank eindeutig identifiziert werden. Sie enthält alle Informationen, die Oracle benötigt, um schnell auf den Satz zugreifen zu können. Sie besitzt das folgende Format:

```
OOOOOO FFF BBBBBB RRR
```

Dabei ist:

- `OOOOOO`: Die Objekt-ID der Tabelle oder des Objekts
- `FFF`: Die Nummer des Datafiles
- `BBBBBB`: Die Blocknummer im Datafile
- `RRR`: Die Zeilennummer im Block

Die `ROWID` ist eine implizite Spalte und kann mit SQL-Mitteln abgefragt werden:

```
SQL> SELECT rowid,dummy FROM dual;
ROWID              D
------------------ -
AAAACMAABAAAAV5AAA X
```

Oracle beginnt mit dem Füllen eines Datenblocks am Ende. Dabei reduziert sich der freie Platz zwischen Daten und Header (siehe Abbildung 2.2). Um Row Chaining und Row Migration zu vermeiden, behält jeder Block einen freien Platz für Updates. Dieser freie Platz wird durch den Storage-Parameter `PCTFREE` festgesetzt. Hier gilt es, einen Kompromiss zwischen Platzverschwendung und genügend Freiraum für Updates zu finden. Der Standardwert ist 10 Prozent.

Extents

Ein Extent besteht aus einem zusammenhängenden Bereich von Datenblöcken. Neue Extents werden automatisch angelegt, wenn das Segment wächst, so lange, bis eine Grenze erreicht wird. Extents, die von einem Segment belegt wurden, werden nicht automatisch wieder zurückgegeben, auch wenn sie leer sein sollten. Sie werden erst mit dem Löschen des Objekts wieder freigegeben.

Segments

Ein Segment ist eine Zusammenfassung von Extents und repräsentiert ein Objekt in der Datenbank, das mit Daten gefüllt werden kann. So entspricht zum Beispiel eine Tabelle einem Segment.

Ein Segment wird nicht zwangsläufig beim Erstellen eines Objekts angelegt. Für Tabellen, Indexe sowie Partitionen gilt, dass zunächst nur die Metadaten im Datenbankkatalog erstellt werden. Sobald der erste Datensatz eingefügt wird, erfolgt das Anlegen des Segments. Dieses Feature wird als *Deferred Segment Creation* bezeichnet.

Oracle verwaltet den Platz innerhalb eines Segments mithilfe des High Water Mark (HWM). Die Blöcke, die sich oberhalb des HWM befinden, sind unformatiert und wurden noch nicht benutzt. Blöcke unterhalb des HWM können durchaus leer sein, zum Beispiel weil Daten gelöscht wurden.

Tablespaces

Eine Tablespace ist ein logischer Container für Segmente, also Tabellen, Indexe, Cluster, LOBs. Die SYSTEM-Tablespace enthält als wichtigste Komponente den Datenbankkatalog. Eigentümer des Katalogs ist der Benutzer SYS. Die SYSTEM-Tablespace wird mit dem Erstellen der Datenbank automatisch angelegt.

Die Tablespace SYSAUX enthält Schemata von Oracle-Komponenten und kann für weitere Werkzeuge oder Komponenten verwendet werden, die ein Repository benötigen.

In der UNDO-Tablespace werden UNDO-Segmente gespeichert. Diese werden für das Zurückrollen von Transaktionen, die Lesekonsistenz und Flashback-Operationen benötigt.

Eine temporäre Tablespace enthält temporäre Segmente, die für die Gültigkeitsdauer einer Session benötigt werden. Dies ist der Fall bei größeren Sortieroperationen oder beim Anlegen von temporären Tabellen.

```
SQL> SELECT tablespace_name,file_name
  2  FROM dba_data_files
  3  ORDER BY 1,2;
TABLESPACE_NAME  FILE_NAME
---------------- ----------------------------------------
SYSAUX           +DATA/mitp/datafile/sysaux.264.818620179
SYSTEM           +DATA/mitp/datafile/system.258.818620149
UNDOTBS1         +DATA/mitp/datafile/undotbs1.261.818620197
USERS            +DATA/mitp/datafile/users.267.818620231
```

Listing 2.3: Permanente Tablespaces und Datafiles anzeigen

Es gibt zwei grundlegende Arten von Tablespaces:

- Locally Managed Tablespaces
- Dictionary Managed Tablespaces

Der Standardtyp ist die Locally Managed Tablespace. Die Verwaltung des freien Platzes erfolgt über ein Bitmap im Header des Datafiles. Segmente innerhalb der Tablespace können automatisch oder manuell verwaltet werden. Das Automatic Segment Space Management (ASSM) ist der Standard, mit Ausnahme der Tablespaces SYSTEM, UNDO und TEMP. ASSM bietet neben einer besseren Performance den Vorteil, dass die manuelle Verwaltung der Storage-Parameter entfällt, was die Administration vereinfacht.

Die Dictionary Managed Tablespace verwendet den Datenbankkatalog zur Verwaltung der Extents. Er war in früheren Versionen der Standardtyp und wurde durch die Locally Managed Tablespace abgelöst, um eine bessere Performance zu erzielen und Konflikte im Katalog zu vermeiden. Dictionary Managed Tablespaces können aus Kompatibilitätsgründen noch verwendet werden.

Mit der Version 10g wurde ein weiterer Tablespace-Typ eingeführt: die Bigfile Tablespace. Dies war vor allem der zunehmenden Größe von Datenbanken geschuldet. Das Datafile einer Smallfile Tablespace (Default) kann aus maximal 4 Millionen Datenblöcken bestehen. Bei einer Blockgröße von 8 KB bedeutet das, dass die maximale Größe eines Datafiles nicht größer als 32 GB werden kann. Das Datafile einer Bigfile Tablespace kann bei einer Blockgröße von 8 KB immerhin 32 TB groß werden. Allerdings darf es nicht mehr als ein Datafile in der Bigfile Tablespace geben.

Hinweis

Weitere Informationen zur Administration und Verwaltung von Speicherstrukturen sowie zum Thema »Fragmentierung« finden Sie in Kapitel 3, »Interne Strukturen und Objekte«.

Redo-Log-Dateien

Die Redo-Log-Dateien bilden das Transaktionslog der Datenbank. Änderungen in der Datenbank landen nach Abschluss einer Transaktion (`COMMIT`) in der Regel nicht auf der Disk, sondern bleiben bis zum nächsten Checkpoint im Buffer Cache der Datenbank. Sie werden jedoch in die Redo-Log-Dateien geschrieben, sodass bei einem Datenbank-Crash alle abgeschlossenen Transaktionen wiederhergestellt werden können. Online-Redo-Log-Dateien sind also kritisch für die Wiederherstellbarkeit der Datenbank.

Die Online-Redo-Log-Dateien bestehen aus mehreren Gruppen. Jede Gruppe kann mehrere Member (Dateien) enthalten. Die Member einer Gruppe sind identisch gespiegelte Dateien und dienen der Ausfallsicherheit. Ist ein Member einer Gruppe korrupt oder versehentlich gelöscht worden, kann immer noch auf die übrigen Member zurückgegriffen werden. In der Regel verwendet man zwei Member pro Gruppe.

Der Log Writer schreibt immer in genau eine Gruppe, diese besitzt den Status `CURRENT`. Ist die Gruppe voll, erfolgt ein Log Switch. Dabei wechselt der Log Writer zur nächsten Gruppe und markiert diese als `CURRENT`. Ist der Log Writer bei der letzten Gruppe angekommen, beginnt er wieder mit der ersten und überschreibt diese. Läuft die Datenbank im `Archivelog`-Modus, wird die Gruppe vor dem Überschreiben archiviert. Es wird eine Kopie in Form einer Archived-Redo-Log-Datei erstellt. Damit garantiert Oracle durch die vorangegangene Sicherung eine Wiederherstellung zu einem beliebigen Zeitpunkt. Mit jedem Log Switch wird eine neue Sequence-Nummer erstellt.

```
SQL> SELECT group#,sequence#,status,first_time
  2  FROM v$log;
    GROUP#  SEQUENCE# STATUS           FIRST_TIME
---------- ---------- ---------------- -------------------
         1         73 CURRENT          23.06.2019 20:00:51
```

```
         2          71 INACTIVE          20.06.2019 21:00:16
         3          72 INACTIVE          23.06.2019 18:49:32
```

Listing 2.4: Informationen über die Online-Redo-Log-Gruppen abfragen

Tipp

Standardmäßig werden vom DBCA drei Log-Gruppen mit einer Größe von 50 MB angelegt. Die Größe ist für viele Datenbanken zu klein und die Anzahl zu gering. Legen Sie Redo-Log-Dateien mit mindestens fünf Gruppen und einer Größe von 250 MB an. Aufgrund ihrer Kritikalität für die Wiederherstellbarkeit der Datenbank sollten die Gruppen mit mindestens zwei Membern (ein Spiegel) angelegt werden.

Kontrolldateien

In der Kontrolldatei befinden sich u.a. die Informationen über alle Dateien, die unmittelbar zur Datenbank gehören. Das sind Datafiles, Tempfiles und Online-Redo-Log-Dateien. Die Kontrolldatei ist sozusagen die Klammer, die die Datenbank zusammenhält. Damit ist sie sehr kritisch und sollte ebenfalls gespiegelt werden. Der Speicherort der Kontrolldateien wird durch den Datenbankparameter `control_files` festgelegt.

Server-Parameter-File (SPFILE)

Das `SPFILE` enthält die Werte aller Parameter der Datenbank, die sogenannten *Init-Parameter*. Es ist eine Binär-Datei und sie sollte nicht mit einem Editor bearbeitet werden, da sie beschädigt werden könnte. Ändern Sie Werte im `SPFILE` nur mit dem `ALTER SYSTEM`-Befehl oder erstellen Sie eine Datei im Textformat (`PFILE`) aus dem `SPFILE`.

Passwordfile

Auch Benutzer mit `SYSDBA`- oder `SYSOPER`-Privilegien identifizieren sich mithilfe des verschlüsselten Passworts im Datenbankkatalog. Ist die Datenbank nicht geöffnet, zum Beispiel vor dem Start der Instanz, kann der Katalog zur Prüfung des Passworts nicht herangezogen werden. In diesem Fall und wenn keine Identifizierung über das Betriebssystem erfolgt, wird das Passwordfile herangezogen.

Es befindet sich im Verzeichnis `$ORACLE_HOME/dbs` (bzw. `%ORACLE_HOME%\database` unter Windows) und hat den Namen `orapw<SID>`. Erstellung und Bearbeitung erfolgt mit dem Werkzeug `orapwd`. Eine Änderung des Passworts mit dem `ALTER USER`-Kommando bewirkt, dass das neue Passwort sowohl im Datenbankkatalog als auch im Passwordfile gespeichert wird. Die SQL-Abfrage in Listing 2.5 liefert alle Benutzer, die im Passwordfile hinterlegt sind.

```
SQL> SELECT * FROM v$pwfile_users;
USERNAME   SYSDB SYSOP SYSAS SYSBA SYSDG SYSKM     CON_ID
---------- ----- ----- ----- ----- ----- ----- ----------
SYS        TRUE  TRUE  FALSE FALSE FALSE FALSE          0
SYSDG      FALSE FALSE FALSE FALSE TRUE  FALSE          0
```

```
SYSBACKUP  FALSE FALSE FALSE TRUE  FALSE FALSE          0
SYSKM      FALSE FALSE FALSE FALSE FALSE TRUE           0
```

Listing 2.5: Benutzer im Passwordfile abfragen

Archived-Redo-Log-Dateien

Im Archivelog-Modus werden die Online-Redo-Log-Dateien beim Log-Switch in Archived-Redo-Log-Dateien kopiert. Das Zielverzeichnis für die Dateien kann im Parameter `log_archive_dest_n` hinterlegt werden. Der Parameter kann zum Beispiel so aussehen:

```
log_archive_dest_1='LOCATION=/u01/oracle/archive'
```

Ist kein Verzeichnis als Ziel definiert, werden die Dateien in die Fast Recovery Area geschrieben, falls diese definiert ist. Sie werden dann im OMF-Format hinterlegt. Alle Informationen über die Archived-Redo-Log-Dateien finden Sie in der View `V$ARCHIVED_LOG` .

```
SQL> SELECT name,sequence# FROM v$archived_log;
NAME                                                SEQUENCE#
-------------------------------------------------- ----------
/u01/oracle/fast_recovery_area/MITP/archivelog/201         33
4_01_06/o1_mf_1_33_9do1d2bn_.arc
```

Listing 2.6: Informationen über Archived-Redo-Log-Dateien im Katalog abfragen

Vorsicht

Wird im Verzeichnis für die Archived-Redo-Log-Dateien noch eine Fast Recovery Area definiert, werden die Dateien in das Verzeichnis `$ORACLE_HOME/dbs` (unter Windows `%ORACLE_HOME\database`) geschrieben, ohne dass ein Fehler gemeldet wird. Die Größe des Dateisystems ist in der Regel nicht dafür ausgelegt. Ist das Dateisystem zu 100 % gefüllt, ist ein regulärer Betrieb der Datenbank nicht mehr möglich.

Flashback-Log-Dateien

Flashback-Log-Dateien dienen dem Zurücksetzen der Datenbank auf einen früheren Zustand. Sie werden mit dem Befehl `FLASHBACK DATABASE` angewandt und funktionieren ähnlich wie die Redo-Log-Dateien, nur in zeitlich umgekehrter Richtung. Sie werden standardmäßig nicht geschrieben und müssen durch den DBA aktiviert werden. Dafür gibt es zwei Optionen:

- Permanente Aktivierung der Flashback-Log-Dateien
- Erstellen eines garantierten Restore Points (GRP). Das Schreiben der Flashback-Log-Dateien wird dynamisch eingeschaltet, und nach dem Löschen des GRP wird abgestellt.

Voraussetzung in beiden Fällen ist, dass die Datenbank im `ARCHIVELOG`-Modus läuft.

Eine permanente Aktivierung erfolgt durch einen `ALTER DATABASE`-Befehl.

```
SQL> ALTER DATABASE FLASHBACK ON;
Datenbank wurde geändert.
```

Listing 2.7: Das Schreiben von Flashback-Log-Dateien aktivieren

Auf dieselbe Art kann das Schreiben der Log-Dateien deaktiviert werden. Das Ein- und Ausschalten kann dynamisch bei geöffneter Datenbank durchgeführt werden. Flashback-Log-Dateien werden in die Fast Recovery Area geschrieben. In diesem Zusammenhang ist der Parameter `DB_FLASHBACK_RETENTION_TARGET` zu beachten. Er garantiert, dass ältere Log-Dateien nicht vor Erreichen des Zeitraums gelöscht werden. Die Angabe erfolgt in Minuten.

Die zweite Option ist das Setzen eines garantierten Restore Point. In diesem Fall hat der Wert des Parameters `DB_FLASHBACK_RETENTION_TARGET` keinen Einfluss auf das Löschen der Flashback-Log-Dateien. Diese bleiben so lange erhalten, wie der Restore Point aktiv ist. Das Setzen und Löschen eines Restore Point kann ebenfalls dynamisch bei geöffneter Datenbank erfolgen. Mit dem Löschen des Restore Point werden alle zugehörigen Flashback-Log-Dateien gelöscht. Ein Beispiel für das Setzen und das Löschen finden Sie in Listing 2.8.

```
SQL> CREATE RESTORE POINT before_upgrade GUARANTEE FLASHBACK DATABASE;
Restore-Punkt erstellt.
SQL> SELECT name,time FROM v$restore_point;
NAME             TIME
---------------- --------------------------------
BEFORE_UPGRADE   05-JAN-19 12.23.40.000000000 PM
SQL> DROP RESTORE POINT before_upgrade;
Restore-Punkt gelöscht.
```

Listing 2.8: Einen garantierten Restore Point setzen und löschen

Block-Change-Tracking-File

Mit einem Block-Change-Tracking-File (BCT-File) können Zeit- und Ressourcenverbrauch für eine inkrementelle Sicherung mit dem Recovery Manager deutlich reduziert werden. Normalerweise liest RMAN bei einer inkrementellen Sicherung jeden Datenblock und prüft, ob er seit der letzten Sicherung geändert wurde. Diese Methode führt zu einem hohen I/O-Aufkommen und einer Laufzeit, die sich nur unwesentlich von einer Vollsicherung unterscheidet.

Das Block-Change-Tracking-File ist ein Index der geänderten Blöcke. Ist es aktiviert, dann benutzt RMAN diesen Index und liest nur die geänderten Datenblöcke. Damit reduzieren sich die Sicherungszeiten auf ca. 10 % bis 20 % und das I/O-Aufkommen wird deutlich verringert. Das BCT-File ist standardmäßig deaktiviert. Die Aktivierung erfolgt durch ein `ALTER DATABASE`-Kommando.

```
SQL> ALTER DATABASE ENABLE BLOCK CHANGE TRACKING
  2  USING FILE '/u01/oracle/bct/MITP_bct.bin';
```

```
Datenbank wurde geändert.
SQL> ALTER DATABASE DISABLE BLOCK CHANGE TRACKING;
Datenbank wurde geändert.
```

Listing 2.9: Das Schreiben eines Block-Change-Tracking-Files ein- und ausschalten

Das BCT-File ist eine Binärdatei und bleibt relativ klein. Seine Größe bewegt sich auch für größere Datenbanken im MB-Bereich. Es sollte vor einer Vollsicherung angelegt werden, damit es zur darauf folgenden inkrementellen Sicherung wirksam wird.

Diagnostic-Dateien

Die Funktionalität für die Diagnostik der Datenbank wurde mit der Version 11g wesentlich erweitert. Gleichzeitig wurde eine neue Struktur der zugehörigen Log-, Trace- und Diagnostic-Dateien eingeführt. Das Feature wird auch als *Advanced Diagnostic Repository* (ADR) bezeichnet.

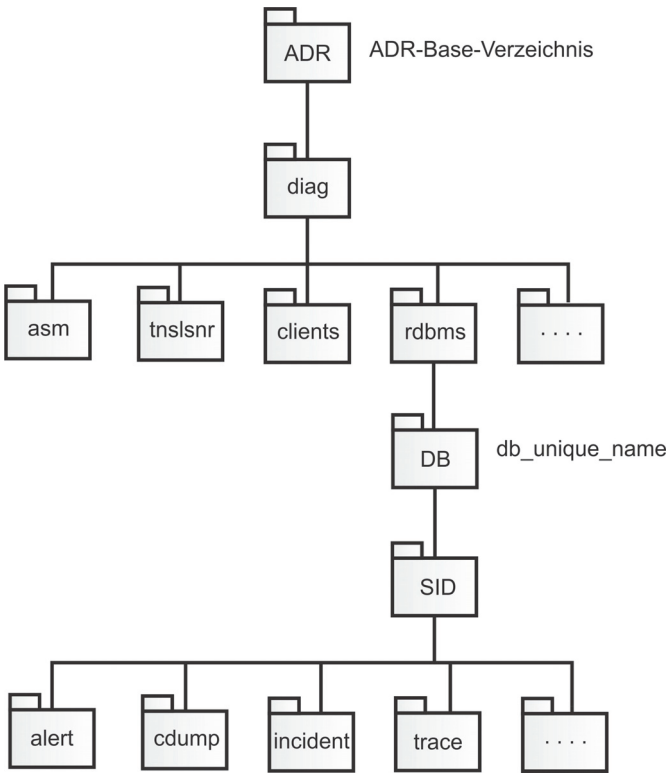

Abb. 2.4: Die ADR-Verzeichnisstruktur

Das ADR-Base-Verzeichnis wird durch den Datenbankparameter `DIAGNOSTIC_DEST` definiert. Die Struktur für die Diagnostic-Verzeichnisse der Datenbank kann mit der SQL-Anweisung in Listing 2.10 abgefragt werden.

```
SQL> SELECT name,value FROM v$diag_info;
NAME                 VALUE
-------------------- ----------------------------------------
Diag Enabled         TRUE
ADR Base             /u01/oracle
ADR Home             /u01/oracle/diag/rdbms/doag/DOAG
Diag Trace           /u01/oracle/diag/rdbms/doag/DOAG/trace
Diag Alert           /u01/oracle/diag/rdbms/doag/DOAG/alert
Diag Incident        /u01/oracle/diag/rdbms/doag/DOAG/incident
Diag Cdump           /u01/oracle/diag/rdbms/doag/DOAG/cdump
Health Monitor       /u01/oracle/diag/rdbms/doag/DOAG/hm
Default Trace File   /u01/oracle/diag/rdbms/doag/DOAG/trace/DO
                     AG_ora_13775.trc
Active Problem Count 0
Active Incident Count 0
```

Listing 2.10: ADR-Informationen abfragen

Die Navigation durch die Verzeichnisse ist zeitaufwendiger gegenüber der früheren Struktur. Eine Vereinfachung liefert das Kommandozeilenwerkzeug `adrci`. Das Beispiel in Listing 2.11 zeigt, wie ein schneller Blick in das Alert-Log der Datenbank möglich ist. Für eine laufende Tail-Anzeige können Sie das Kommando `adrci> SHOW ALERT -TAIL -F` verwenden.

```
$ adrci
ADRCI: Release 19.0.0.0.0 - Production on Mi Nov 27 14:32:37 2019
Copyright (c) 1982, 2019, Oracle and/or its affiliates.
All rights reserved.
ADR base = "/u01/oracle"
adrci> SHOW ALERT
Choose the home from which to view the alert log:
1: diag/rdbms/mitp/MITP
2: diag/rdbms/test/TEST
3: diag/tnslsnr/serv7/listener
4: diag/tnslsnr/serv7/listener0
Q: to quit
Please select option: 1
. . .
```

Listing 2.11: Zugriff auf die Alert-Log-Datei mit `adrci`

2.1.2 Die Struktur der Instanz

Das Hochfahren der Instanz ist der erste Schritt beim Starten einer Datenbank. Wenn Sie im `STARTUP`-Befehl die `NOMOUNT`-Option verwenden, wird nur die Instanz gestartet, ohne die Kontrolldatei und die Datafiles zu öffnen. Oracle liest die Initialisierungsparameter aus dem `SPFILE`, initialisiert die Hauptspeicherstrukturen des Memory (SGA und PGA) und startet die Hintergrundprozesse.

Hinweis

Hintergrundprozesse sind unter Unix und Windows unterschiedlich implementiert, bedingt durch die unterschiedliche Architektur der Betriebssysteme. Während unter Unix jeder Hintergrundprozess ein einzelner unabhängiger Prozess ist, wird unter Windows ein Thread unter dem Hauptprogramm `oracle.exe` gebildet.

Der Hauptbestandteil der Instanz ist die *System Global Area* (SGA), die sich im Shared Memory befindet. Die SGA enthält Daten und Buffer, die datenbankweit von allen Sessions (Benutzern) gemeinsam benutzt werden. Ein weiterer Bestandteil des Memory ist die *Program Global Area* (PGA). In ihr befinden sich sitzungsspezifische Informationen, die nicht mit anderen Sessions geteilt werden. Deshalb wird die PGA auch als *Private Memory* bezeichnet.

Hinweis

Traditionell wurde im Betriebssystem zwischen Shared und Private Memory unterschieden und die Bereichsgrößen sind fix definiert. Von diesem Prinzip ist man abgekommen. Es wird nur noch ein Hauptspeicherbereich erstellt, der sowohl vom Private Memory als auch vom Shared Memory der Datenbank benutzt wird. Aus Sicht der Datenbankarchitektur ist die Unterscheidung nach wie vor gerechtfertigt.

Die System Global Area besteht im Wesentlichen aus folgenden Komponenten:

- Database Buffer Cache
- Redo Log Buffer
- Shared Pool (u.a. Library Cache, Dictionary Cache)
- Large Pool
- Java Pool
- Streams Pool

Im *Buffer Cache* werden Kopien der Datenblöcke gespeichert, um einen schnellen Zugriff auf deren Inhalt zu ermöglichen. Im Buffer Cache gibt es drei verschiedene Pools:

- Der *Default Pool* ist der Bereich, in dem alle Datenblöcke standardmäßig gespeichert werden.
- Sie können dem *Keep Pool* Segmente zuweisen, die regelmäßig frequentiert werden. Der Mechanismus im Default Pool würde diese Datenblöcke regelmäßig auslagern.
- Der *Recycle Pool* ist für große Segmente gedacht, die selten angefordert werden. Diese würden im Default Pool häufiger benötigte Segmente verdrängen.

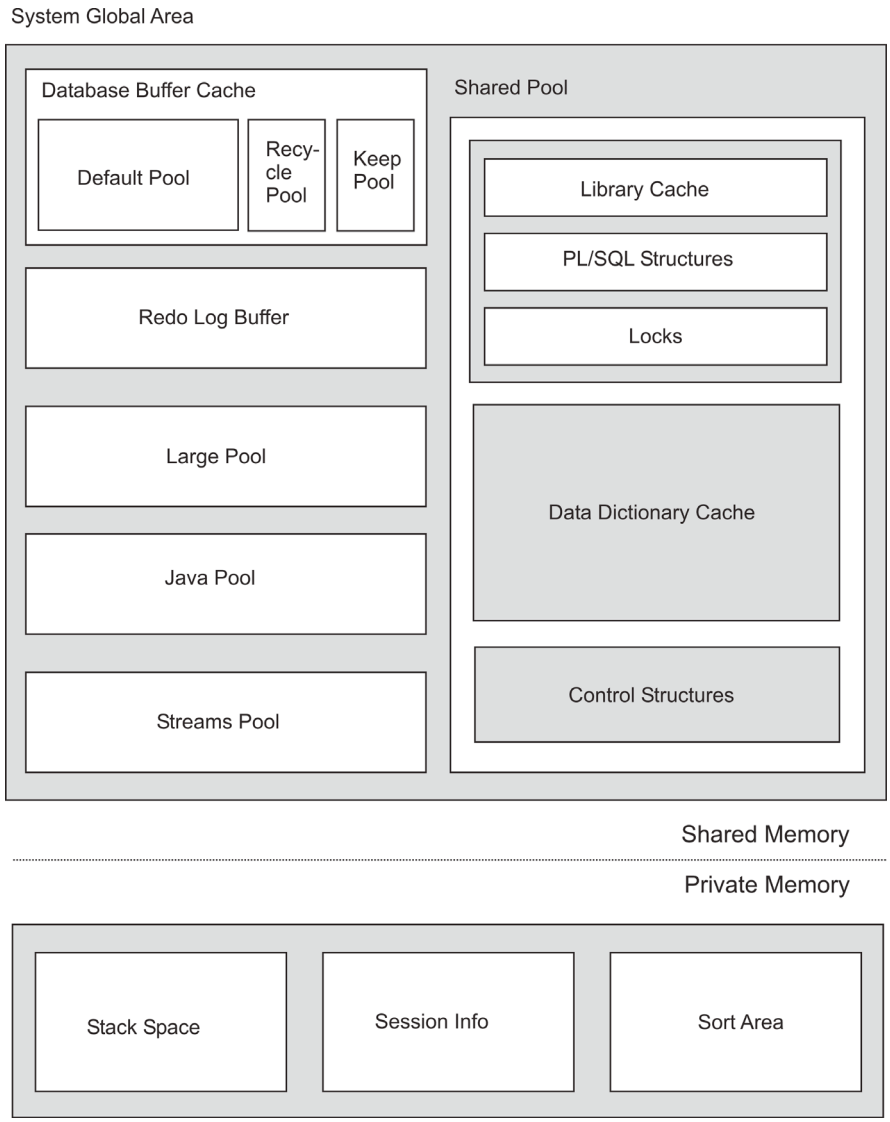

Abb. 2.5: Die Architektur der Oracle-Instanz

Datenblöcke im Buffer Cache können die folgenden drei verschiedenen Charakteristiken annehmen:

- *Dirty Buffer* sind Datenblöcke, die in die Datafiles geschrieben werden müssen, da sie gegenüber dem Original in der Datei verändert wurden.
- *Free Buffer* enthalten keine Daten oder stehen zum Überschreiben zur Verfügung. Sie werden mit Blöcken gefüllt, die von der Disk gelesen werden.
- *Pinned Buffer* sind Blöcke, die gerade benutzt werden oder für zukünftige Benutzung reserviert sind.

Oracle verwendet zwei Listen, um den Buffer Cache zu verwalten. Die *Write List*, auch *Dirty Buffer List* genannt, registriert alle Blöcke, die verändert wurden und auf Disk geschrieben werden müssen. Die *LRU-Liste* (Least Recently Used List) enthält *Free Buffer*, *Pinned Buffer* und *Dirty Buffer*, die noch nicht auf der Write List stehen. In der LRU-Liste stehen die zuletzt am häufigsten benutzten Blöcke vorn.

Wenn ein Prozess auf einen Datenblock zugreift, wird er an den Anfang der LRU-Liste gesetzt. Gleichzeitig wird ein Block am Ende der Liste hinausgeschoben. Mit diesem Mechanismus wird garantiert, dass sich die am häufigsten benutzten Datenblöcke im Buffer Cache befinden.

Hinweis

Es gibt eine Ausnahme von dieser Vorgehensweise. Wird ein Full Table Scan durchgeführt, werden die gelesenen Blöcke nicht an den Anfang, sondern an das Ende der LRU-Liste geschrieben. Damit wird verhindert, dass andere häufig verwendete Blöcke durch Full Table Scans von der LRU-Liste verdrängt werden.

Wenn ein Datenblock von einem Oracle-Prozess angefordert wird, wird zuerst geprüft, ob sich der Block im Buffer Cache befindet. Ist der Block im Cache vorhanden, dann nennt man diese Situation einen *Cache Hit*. Andernfalls muss er von der Disk gelesen werden. Dies nennt man einen *Cache Miss*.

Bevor Oracle einen Datenblock von der Disk in den Buffer Cache laden kann, muss ein freier Buffer gefunden werden. Der Prozess sucht, bis er einen freien Buffer gefunden hat oder bis ein Schwellenwert erreicht ist. Findet der Prozess während des Suchlaufs einen Dirty Buffer, entfernt er diesen von der LRU-Liste und trägt ihn in die Write-Liste ein. Anschließend sucht er weiter. Wird ein freier Buffer gefunden, lädt der Prozess den Datenblock von der Disk und trägt ihn in die LRU-Liste ein.

Hat der Prozess bei Erreichen des Schwellenwerts keinen freien Buffer gefunden, dann hört er auf zu suchen und signalisiert die Situation dem *Database-Writer-Prozess*. Der Database Writer fängt an, Dirty Buffer auf Disk zu schreiben, womit wieder freie Buffer entstehen.

Hinweis

Seit der Version 10g ist es möglich, verschiedene Blockgrößen innerhalb einer Datenbank zu verwalten. Für jede Blockgröße muss ein eigener Buffer Cache bereitgestellt werden. Die Blockgröße der `SYSTEM`-Tablespace bestimmt die Standardblockgröße der Datenbank.

Im *Redo Log Buffer* befinden sich die Daten, die durch den *Log-Writer-Prozess* (LGWR) in die Online-Redo-Log-Dateien geschrieben werden. Redo-Log-Daten werden zur Wiederherstellung von Transaktionen verwendet.

Der *Library Cache* enthält SQL-Anweisungen, PL/SQL-Prozeduren und Kontrollstrukturen wie Locking-Informationen. Befindet sich eine SQL-Anweisung in der Shared SQL Area, wurde sie bereits geparst und kann von anderen Sessions direkt ausgeführt werden. Oracle behandelt PL/SQL-Strukturen ähnlich wie SQL-Anweisungen. Sie werden von der PL/SQL Engine ausgeführt.

Der *Large Pool* ist ein optionaler Bereich der SGA und erfüllt eine ähnliche Funktion wie der Shared Pool mit der Einschränkung, dass nur bestimmte Typen und Größen von Shared Memory zugewiesen werden können. Der Hauptspeicher für den Large Pool wird direkt aus der SGA zugewiesen. Der Large Pool wird unter anderem für folgende Operationen verwendet:

- Session-Informationen des Shared Server
- I/O-Server-Prozesse
- Backup- und Restore-Operationen
- Message Buffer für parallele Abfragen

Der *Streams Pool* wurde in Oracle 10g eingeführt. Er dient der Speicherung von *Buffered Queues*. Buffered Queues werden vorwiegend von Oracle Streams benutzt und besitzen signifikante Performance-Vorteile gegenüber herkömmlichen Queues.

Die *Program Global Area* (PGA) und die *User Global Area* (UGA) befinden sich im Private Memory. Andere Sitzungen haben keinen Zugriff darauf. Während die PGA im Wesentlichen die Sort Area und den Stack Space enthält, werden in der UGA die Statusinformationen der Session gespeichert.

2.1.3 Automatic Memory Management (AMM)

Das Automatic Memory Management wurde in Oracle 10g eingeführt. Es war da noch auf den Shared Memory beschränkt. Seit Oracle 11g ist es möglich, auch den Private Memory einzubinden. Damit ist Oracle nicht nur in der Lage, sowohl den Shared als auch den Private Memory dynamisch zu verwalten, sondern auch Speicher zwischen beiden Bereichen dynamisch auszutauschen.

AMM wird von den Plattformen AIX, Solaris, HP-UX, Linux und Windows unterstützt. Für das AMM wurden zwei neue Init-Parameter eingeführt: `MEMORY_TARGET` und `MEMORY_MAX_TARGET`. Der Wert von `MEMORY_TARGET` kann bis zur Grenze `MEMORY_MAX_TARGET` dynamisch verändert werden. Um die Datenbank auf AMM einzustellen, ist es ausreichend, den Parameter `MEMORY_TARGET` auf den gewünschten Wert und die übrigen AMM-Parameter auf »null« zu setzen. Für die Parameter untereinander existieren die folgenden Abhängigkeiten:

- `MEMORY_TARGET` ist auf einen Wert ungleich »null« gesetzt (AMM eingeschaltet):
  - Wenn gleichzeitig `SGA_TARGET` und `PGA_AGGREGATE_TARGET` gesetzt sind, werden diese als die Minimalwerte für diese Bereiche angesehen.
  - Falls `SGA_TARGET` gesetzt und `PGA_AGGREGATE_TARGET` nicht gesetzt ist, werden beide Parameter durch AMM gesetzt. Initial wird `PGA_AGGREGATE_TARGET` auf die Differenz zwischen `MEMORY_TARGET` und `SGA_TARGET` gesetzt.
  - Ist `PGA_AGGREGATE_TARGET` gesetzt und `SGA_TARGET` nicht gesetzt, dann werden beide Parameter durch AMM getunt. Die initiale Größe für `SGA_TARGET` ist die Differenz zwischen `MEMORY_TARGET` und `PGA_AGGREGATE_TARGET`.
  - Sind weder `SGA_TARGET` noch `PGA_AGGREGATE_TARGET` gesetzt, dann werden beide Bereiche durch AMM ohne Minimalwert verwaltet. Beim Start der Instanz werden 60 % an die SGA und 40 % an die PGA vergeben.

- MEMORY_TARGET ist nicht oder auf »null« gesetzt.
  - Wenn SGA_TARGET gesetzt ist, werden die Pools der SGA durch AMM verwaltet, so wie das aus Oracle 10g bekannt ist. Die PGA wird durch AMM verwaltet, unabhängig davon, ob der Parameter PGA_AGGREGATE_TARGET gesetzt ist oder nicht.
  - Sind weder SGA_TARGET noch PGA_AGGREGATE_TARGET gesetzt, dann wird die PGA durch AMM verwaltet, die SGA jedoch nicht.
  - Ist nur MEMORY_MAX_TARGET gesetzt, wird MEMORY_TARGET auf »null« gesetzt, und das automatische Tuning für SGA und PGA ist ausgeschaltet.
  - Wenn SGA_MAX_SIZE nicht gesetzt ist, wird der Parameter intern auf MEMORY_MAX_TARGET gestellt.

Hinweis

Wenn Sie in einer Init-Parameterdatei die Zeile für MEMORY_MAX_TARGET weglassen und MEMORY_TARGET auf einen Wert größer »null« setzen, wird MEMORY_MAX_TARGET automatisch auf den Wert von MEMORY_TARGET gesetzt.

Mit den folgenden Schritten schalten Sie AMM für eine Oracle-19c-Datenbank ein:

1. Ermitteln Sie die aktuellen Werte für die Initialisierungsparameter SGA_TARGET und PGA_AGGREGATE_TARGET.

```
SQL> SHOW PARAMETER sga_target
NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------
sga_target                           big integer 268435456
SQL> SHOW PARAMETER pga_aggregate_target
NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------
pga_aggregate_target                 big integer 536870912
```

2. Fragen Sie mit der folgenden SQL-Anweisung den Maximalwert ab, den die PGA seit dem Start der Datenbank benötigt hat.

```
SQL> SELECT value FROM v$pgastat
  2  WHERE name = 'maximum PGA allocated';
     VALUE
----------
  98786304
```

3. Legen Sie den Wert für MEMORY_TARGET auf Basis der ermittelten Größen fest.
4. Setzen Sie die Parameter für das AMM.

```
SQL> ALTER SYSTEM SET memory_target=1328M SCOPE=spfile;
System altered.
```

```
SQL> ALTER SYSTEM SET pga_aggregate_target=0 SCOPE=spfile;
System altered.
SQL> ALTER SYSTEM SET sga_target=0 SCOPE=spfile;
System altered.
```

5. Führen Sie einen Neustart der Datenbank durch.

Nach der Aktivierung von AMM passt Oracle die Größen der Hauptspeicherbereiche jeweils dem aktuellen Workload auf der Datenbank an. Für den Datenbankadministrator stehen folgende Views zur Verfügung:

- `V$MEMORY_DYNAMIC_COMPONENTS`: Enthält zusammengefasste Informationen über die Änderungen von Speichergrößen seit dem Start der Instanz.
- `V$MEMORY_RESIZE:OPS`: Liefert eine Historie alle Operationen zur Änderung von Speichergrößen seit dem Start der Instanz.
- `V$MEMORY_TARGET_ADVICE`: Erstellt eine Schätzung der Verbesserung der Datenbank-Performance in Abhängigkeit von der Größe des Gesamtspeichers für Oracle.

```
SQL> SELECT component, current_size, min_size, max_size
  2  FROM v$memory_dynamic_components;
COMPONENT                      CURRENT_SIZE   MIN_SIZE   MAX_SIZE
------------------------------ ------------ ---------- ----------
shared pool                       234881024  234881024  234881024
large pool                         16777216   16777216   16777216
java pool                          16777216   16777216   16777216
streams pool                              0          0          0
SGA Target                       1056964608 1056964608 1056964608
DEFAULT buffer cache              771751936  771751936  771751936
KEEP buffer cache                         0          0          0
RECYCLE buffer cache                      0          0          0
DEFAULT 2K buffer cache                   0          0          0
DEFAULT 4K buffer cache                   0          0          0
DEFAULT 8K buffer cache                   0          0          0
DEFAULT 16K buffer cache                  0          0          0
DEFAULT 32K buffer cache                  0          0          0
Shared IO Pool                            0          0          0
PGA Target                        335544320  335544320  335544320
ASM Buffer Cache                          0          0          0
```

Listing 2.12: Zusammenfassung der Werte des AMM

```
SQL> SELECT parameter, initial_size, target_size, status,
  2  start_time, end_time
  3  FROM v$memory_resize_ops;
```

```
PARAMETER              INITIAL_ TARGET_SIZE STATUS    START_TI END_TIME
---------------------- -------- ----------- --------- -------- --------
shared_pool_size              0   234881024 COMPLETE  12:13:01 12:13:01
db_cache_size             51936   771751936 COMPLETE  12:13:01 12:13:02
java_pool_size                0    16777216 COMPLETE  12:13:01 12:13:01
streams_pool_size             0           0 COMPLETE  12:13:01 12:13:01
sga_target                    0  1056964608 COMPLETE  12:13:01 12:13:01
db_cache_size                 0   771751936 COMPLETE  12:13:01 12:13:01
db_keep_cache_size            0           0 COMPLETE  12:13:01 12:13:01
db_recycle_cache_size         0           0 COMPLETE  12:13:01 12:13:01
db_2k_cache_size              0           0 COMPLETE  12:13:01 12:13:01
db_4k_cache_size              0           0 COMPLETE  12:13:01 12:13:01
db_8k_cache_size              0           0 COMPLETE  12:13:01 12:13:01
db_16k_cache_size             0           0 COMPLETE  12:13:01 12:13:01
db_32k_cache_size             0           0 COMPLETE  12:13:01 12:13:01
pga_aggregate_target          0   335544320 COMPLETE  12:13:01 12:13:01
db_cache_size                 0           0 COMPLETE  12:13:01 12:13:01
large_pool_size               0    16777216 COMPLETE  12:13:01 12:13:01
```

Listing 2.13: Historie der durch den AMM vorgenommenen Veränderungen

```
SQL> SELECT * FROM v$memory_target_advice
  2  ORDER BY memory_size;
MEMORY_SIZE MEMORY_SIZE_FACTOR ESTD_DB_TIME ESTD_DB_TIME_FA    VERSION
----------- ------------------ ------------ --------------- ----------
        664                 .5         1007               1          0
        996                .75         1007               1          0
       1328                  1         1007               1          0
       1660               1.25         1007               1          0
       1992                1.5         1007               1          0
       2324               1.75         1007               1          0
       2656                  2         1007               1          0
```

Listing 2.14: Die Werte des AMM Advisor

Es stellt sich die Frage, wie Oracle den Hauptspeicher verwaltet und Speicherbereiche zwischen PGA und SGA austauscht. Wie ist der Hauptspeicher, hier am Beispiel eines Linux-Betriebssystems, bei gestarteter Instanz mit eingeschaltetem AMM konfiguriert? Schauen wir uns die Shared-Memory-Segmente an:

```
$ ipcs -m
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0xa5d6936c 229378     oracle     660        4096       0
```

Die Shared-Memory-Segmente weisen nur eine Größe von 4 KB auf. Wie schafft es Oracle dann, ein zeitnahes Resizing der Bereiche vorzunehmen? Wie sieht das *Mapped Memory* des Database Writer aus?

```
$ pmap 'pgrep -f dbw'
8021:   ora_dbw0_MITP
. . .
20001000  16380K rwxs-  /dev/shm/ora_MITP_229378_0
21000000  16384K rwxs-  /dev/shm/ora_MITP_229378_1
22000000  16384K rwxs-  /dev/shm/ora_MITP_229378_2
23000000  16384K rwxs-  /dev/shm/ora_MITP_229378_3
24000000  16384K rwxs-  /dev/shm/ora_MITP_229378_4
. . .
```

Listing 2.15: Der Mapped Memory des Database Writer

Hier wird offensichtlich, dass Oracle `/dev/shm` für die Implementierung von Shared Memory verwendet und dafür Segmente in der Größe von 16 MB verwendet. Oracle benutzt eine Segmentgröße von 4 MB, wenn `MEMORY_MAX_TARGET` kleiner als 1024 MB ist, sonst 16 MB.

Die Konfiguration der Instanz sieht wie folgt aus:

```
SQL> show parameter target
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------
. . .
memory_max_target                    big integer 1328M
memory_target                        big integer 1328M
pga_aggregate_target                 big integer 0
sga_target                           big integer 0
SQL> SELECT component, current_size
  2  FROM v$memory_dynamic_components;
COMPONENT               CURRENT_SIZE
----------------------- ------------
shared pool                234881024
large pool                  16777216
java pool                   16777216
streams pool                       0
SGA Target                 822083584
DEFAULT buffer cache       536870912
PGA Target                 335544320
. . .
```

AMM hat also ein SGA Target von 784 MB und ein PGA Target von 320 MB gesetzt. Der Wert für das SGA Target wird bestätigt durch den von Oracle aktuell benutzten Shared Memory:

```
$ df -k /dev/shm
Filesystem           1K-blocks      Used Available Use% Mounted on
none                   1683404    802100    881304  48% /dev/shm
```

Jetzt wird der Wert für die PGA auf 900 MB erhöht. Offensichtlich gibt Oracle den Speicher aus dem Shared-Memory-Bereich und verwendet ihn für die PGA als *Private Memory*.

```
SQL> ALTER SYSTEM SET PGA_AGGREGATE_TARGET=900M;
System altered.
$ df -k /dev/shm
Filesystem           1K-blocks      Used Available Use% Mounted on
none                   1683404    195152   1488252  12% /dev/shm
```

Listing 2.16: Vergrößerung der PGA auf Kosten von Shared Memory

Damit ist das Prinzip klar, wie Oracle die Speicherbereiche dynamisch zuweist und auch Hauptspeicher zwischen Shared Memory und Private Memory verschiebt.

Hinweis

Stellen Sie sicher, dass ein hinreichend großes temporäres Dateisystem auf `/dev/shm` gemountet ist. Andernfalls können Sie AMM nicht einsetzen und erhalten beim Start der Instanz die Fehlermeldung ORA-00845.

```
# umount /dev/shm
SQL> startup
ORA-00845: MEMORY_TARGET not supported on this system
```

Listing 2.17: Starten der Instanz ohne `tmpfs`

Sobald genügend `tmpfs` auf `/dev/shm` gemountet ist, lässt sich die Instanz mit eingeschaltetem AMM wieder normal starten.

```
# mount -t tmpfs shmfs -o size=1600m /dev/shm
SQL> startup
ORACLE instance started.
Total System Global Area 1389391872 bytes
. . .
```

Listing 2.18: `tmpfs` auf `/dev/shm` zuweisen

2.2 Prozesse und Abläufe

In den vorhergehenden Abschnitten haben Sie die Datenbankarchitektur unter den Blickwinkeln Datenbank und Instanz kennengelernt. Dahinter verbergen sich natürlich eine

ganze Reihe von komplexen Prozessen und Abläufen. Mit den wichtigsten wollen wir uns in diesem Abschnitt beschäftigen.

2.2.1 Die Oracle-Hintergrundprozesse

Die Hintergrundprozesse stellen die Verbindung zwischen den Dateien der Datenbank und den Hauptspeicherstrukturen der Instanz her. Sie sind das Gehirn des Datenbanksystems und steuern alle Abläufe. Auch wenn sie voneinander unabhängig laufen, findet eine Kommunikation zwischen den Prozessen statt. Mit dem Hochfahren der Instanz werden die Kernprozesse gestartet. Bestimmte Prozesse werden nur dann gestartet, wenn das zugehörige Feature aktiviert ist.

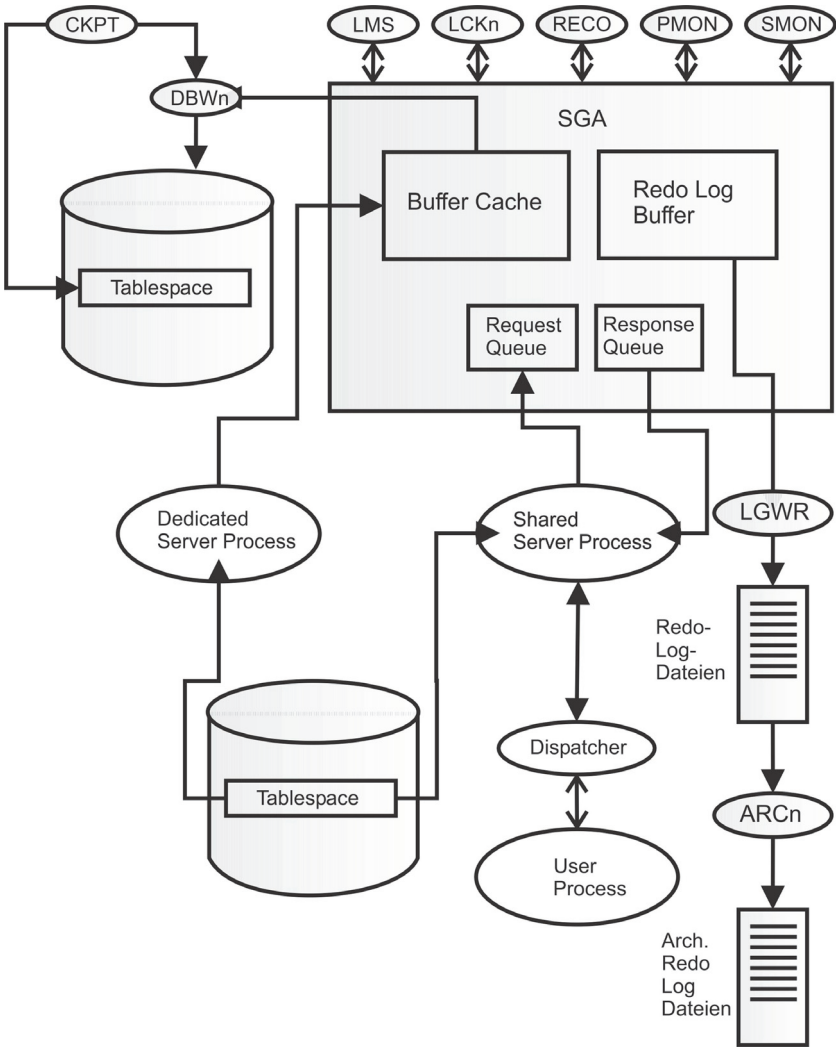

Abb. 2.6: Hintergrundprozesse der Oracle-Datenbank

In einer Client-Server-Umgebung wird für jeden Client, der eine Verbindung zur Datenbank herstellt, ein Client-Server-Prozess auf dem Datenbankserver gestartet. Diese Prozesse sind keine Hintergrundprozesse der Datenbank.

Jeder Hintergrundprozess übernimmt bestimmte Aufgaben im gesamten Prozessablauf der Datenbank. Nur wenn alle notwendigen Prozesse fehlerfrei laufen, ist die Datenbank voll funktionsfähig. Die wichtigsten Prozesse sind in der folgenden Liste beschrieben:

- Der *System Monitor* (SMON) führt beim Start der Instanz, falls erforderlich, ein Instance oder Crash Recovery durch. Weiterhin ist der SMON für das Löschen von temporären Segmenten, die nicht mehr benutzt werden, verantwortlich. Der SMON wacht regelmäßig auf und kontrolliert, ob er gebraucht wird. Andere Prozesse rufen ihn auf, wenn sie feststellen, dass er eine Aufgabe ausführen muss.
- Der *Process Monitor* (PMON) entfernt Benutzerprozesse, die nicht normal beendet wurden. Gleichzeitig gibt er alle von den Prozessen benutzten Ressourcen frei. So wie der SMON wacht der PMON regelmäßig auf und überprüft, ob Aufgaben zu erledigen sind.
- Der *Database Writer* (DBWn) ist verantwortlich für die Verwaltung des Buffer Cache. Seine Hauptaufgabe ist es, geänderte Datenblöcke aus dem Buffer Cache auf Disk zu schreiben. Standardmäßig schreibt er die Blöcke zuerst, die am Ende der LRU-Liste stehen. Zur Verbesserung der Performance können mehrere Database-Writer-Prozesse gestartet werden. Neben einer periodischen Überprüfung wird der Database Writer aus folgenden Situationen heraus aktiviert:
  - Ein Client-Server-Prozess kann keinen freien Buffer finden und erreicht den Schwellenwert für das Timeout. Daraufhin benachrichtigt er den Database Writer.
  - Wenn ein Checkpoint ausgeführt wird.
- Checkpoints zwingen den Database-Writer-Prozess, alle Dirty Buffer auf Disk zu schreiben. Der *Checkpoint Process* (CKPT) ändert die Header aller Datafiles und der Kontrolldateien und trägt die Checkpoint SCN ein. Ein Checkpoint wird durch folgende Situationen ausgelöst:
  - Jeder *Log Switch* der Online-Redo-Log-Dateien löst einen Checkpoint aus.
  - Die Initialisierungsparameter `LOG_CHECKPOINT_INTERVAL`, `LOG_CHECKPOINT_TIMEOUT` und `FAST_START_MTTR_TARGET` haben Einfluss auf die Häufigkeit von Checkpoints.
- Der *Log Writer* (LGWR) ist verantwortlich für die Verwaltung des Inhalts des Redo Log Buffer. Er schreibt die Redo Log Buffer in die Online-Redo-Log-Dateien auf Disk. Der Log Writer wird in den folgenden Situationen aktiviert:
  - Durch eine `COMMIT`-Anweisung in einer Session
  - Alle drei Sekunden
  - Wenn der Redo Log Buffer zu einem Drittel gefüllt ist
  - Wenn der Database Writer Datenblöcke auf die Disk schreibt
- Der *Recovery Process* (RECO) löst Fehler auf, die im Zusammenhang mit verteilten Transaktionen stehen. Er verbindet sich zu den Datenbanken, die in die Transaktionen eingebunden sind.
- Der *Archiver Process* (ARCn) erstellt Kopien der Online-Redo-Log-Dateien in Form von Archived-Redo-Log-Dateien. Dieser Prozess ist nur gestartet, wenn die Datenbank im `ARCHIVELOG`-Modus läuft. In einer Instanz können bis zu zehn Archivelog-Prozesse

(ARC0 bis ARC9) gestartet werden. Der Log Writer startet weitere Prozesse, wenn die aktuelle Anzahl nicht ausreicht.

- Der *Recovery Writer Process* (RVWR) schreibt die Flashback-Log-Dateien in der Flash Recovery Area. Er wird nur gestartet, wenn das Flashback-Database-Feature aktiviert ist.
- Der *Queue Monitor Process* (QMNn) ist beim Einsatz von Advanced Queuing ein optionaler Prozess. Er überwacht die Nachrichtenwarteschlangen. Es können bis zu zehn QMN-Prozesse gestartet werden.
- Der *Dispatcher Process* (Dnnn) ist Teil der Shared-Server-Architektur. Er ist für die Kommunikation mit dem Client verantwortlich.
- Der *Shared Server Process* (Snnn) bedient in einer Shared-Server-Umgebung mehrere Clients.

Wenn Sie die Parallel-Query-Option aktiviert haben, wird die Abfrage auf mehrere Prozesse verteilt. Neben dem Masterprozess werden *Parallel-Server-Prozesse* (Pnnn) gestartet.

2.2.2 Lesekonsistenz

An den Anfang des Themas wollen wir eine Frage stellen. In einer Datenbank wird eine lang laufende `SELECT`-Anweisung auf einer großen Tabelle ausgeführt. Die Anweisung wird um 10:00 Uhr gestartet und führt einen Full Table Scan aus, liest also Block für Block. In der Zwischenzeit ändert eine andere Session einen Datensatz in dieser Tabelle und führt ein `COMMIT` aus, sodass die Änderungen datenbankweit sichtbar werden. Dies passiert um 10:05 Uhr. Um 10:07 liest die `SELECT`-Anweisung diesen Datenblock. Was wird im Ergebnis der `SELECT`-Anweisung erscheinen: der ursprüngliche oder der geänderte Datensatz? Was denken Sie?

Bevor wir die Frage beantworten, werden noch einige Begriffe und Abläufe geklärt.

Die System Change Number (SCN)

Die *System Change Number* (SCN) charakterisiert den eindeutigen Zustand einer Datenbank zu einem bestimmten Zeitpunkt. Sie wird in jedem Datenblock-, Segment- und Datafile-Header gespeichert. Damit ist es möglich, den Zustand eines jeden Datenblocks zuzuordnen. Sie kann als logischer, interner Zeitstempel betrachtet werden.

Die SCN wird kontinuierlich bei bestimmten Ereignissen erhöht. So besitzt zum Beispiel jede Transaktion in der Datenbank eine SCN. Wird eine `COMMIT`-Anweisung ausgeführt, dann wird gleichzeitig eine SCN an diese Transaktion gebunden. `COMMIT`-Anweisungen werden als `COMMIT`-Record in die Redo-Log-Dateien geschrieben. In diesem `COMMIT`-Record befindet sich die SCN, die die Transaktion eindeutig identifiziert.

Transaktionen

Die Gewährleistung der Transaktionssicherheit ist eine der wichtigsten Aufgaben einer relationalen Datenbank. Eine bestimmte Anzahl von SQL-Anweisungen wird als Transaktion zusammengefasst und gemeinsam auf die Datenbank angewandt oder gemeinsam zurückgerollt. Es ist nicht wie in anderen Datenbanksystemen erforderlich, den Anfang einer Transaktion zu propagieren. Dies erfolgt implizit. Oracle hält die folgenden drei Anweisungen zur Transaktionssteuerung bereit:

- `COMMIT`: Trägt alle Änderungen der Transaktion als permanent in der Datenbank ein und macht sie für alle Sessions sichtbar. Die Transaktion wird abgeschlossen, und implizit wird eine neue geöffnet.
- `ROLLBACK`: Rollt alle durch die Transaktion gemachten Änderungen zurück. Aus dem Blickwinkel anderer Sessions haben diese Änderungen niemals stattgefunden. Die Transaktion wird abgeschlossen, und implizit wird eine neue geöffnet.
- `SAVEPOINT`: Definiert einen Punkt innerhalb einer Transaktion, auf den separat zurückgerollt werden kann.

Hinweis

Für die DDL-Anweisung erfolgt ein impliziter Abschluss der Transaktion. `COMMIT`- oder `ROLLBACK`-Anweisung sind nicht erforderlich und unwirksam.

Oracle verwendet ein sogenanntes *optimistisches Locking*. Es wird davon ausgegangen, dass die überwiegende Mehrheit der Transaktionen mit `COMMIT` abgeschlossen wird. Aus diesem Grund werden `COMMIT`-Anweisungen sehr schnell abgeschlossen, wogegen `ROLLBACK`-Anweisungen lange laufen können.

Eine Transaktion ist aktiv, solange noch keine `COMMIT`- oder `ROLLBACK`-Anweisung abgesetzt wurde. Alle in einer nicht abgeschlossenen Transaktion gemachten Änderungen sind als temporär zu betrachten. Dabei werden die folgenden Prozesse ausgelöst:

- Es werden `UNDO`-Daten generiert, die ein Zurückrollen der Transaktion ermöglichen. Die `UNDO`-Daten enthalten die originalen, unveränderten Werte.
- Die Änderungen werden über den Redo Log Buffer in die Online-Redo-Log-Dateien in Form von sogenannten *Redo Records* geschrieben. Es werden sowohl die Änderungen der Daten- als auch der `UNDO`-Blöcke gespeichert.
- Die Änderungen werden im Buffer Cache der Datenbank gespeichert. Eine `COMMIT`-Anweisung löst nicht das Speichern der geänderten Datenblöcke auf die Disk aus. Dies geschieht in der Regel mit dem nächsten Checkpoint oder eines anderweitig geforderten Speicherns der Datenblöcke.
- Zeilen einer Tabelle, die von den Änderungen betroffen sind, werden bis zum Abschluss der Transaktion gesperrt. Andere Sessions können Änderungen durch nicht abgeschlossene Transaktionen nicht sehen.

Hinweis

Obwohl der interne Buffer-Cache-Mechanismus auf Blockebene arbeitet, erfolgt das Sperren von Datenstrukturen auf Zeilenebene. Dies ist nicht selbstverständlich und wird in anderen Datenbanksystemen anders gehandhabt. Der Vorteil liegt in einer deutlichen Verringerung der Konkurrenz durch Sperren auf Daten.

Eine `COMMIT`-Anweisung beendet die Transaktion und weist Oracle an, alle darin vorgenommenen Änderungen als persistent zu betrachten und datenbankweit zur Verfügung zu stellen. Dabei werden folgende Prozesse ausgelöst:

- Es wird eine SCN für die Transaktion generiert.
- Der Log Writer schreibt alle Redo Records der Transaktion in die Online-Redo-Log-Dateien.
- Alle Sperren auf den Datensätzen werden aufgehoben.
- Alle Savepoints werden gelöscht.
- Die Transaktion wird beendet.

Nach Abschluss der Transaktion werden die Änderungen für alle anderen Sessions der Datenbank sichtbar.

Eine `ROLLBACK`-Anweisung macht alle Änderungen, die innerhalb der Transaktion ausgeführt wurden, rückgängig. Dabei werden folgende Aktionen ausgeführt:

- Alle in den Datenblöcken vorgenommenen Änderungen werden zurückgerollt. Grundlage sind die gespeicherten `UNDO`-Daten.
- Alle Sperren auf den Datensätzen werden aufgehoben.
- Die Transaktion wird nicht gespeichert, so als hätte sie nie stattgefunden.
- Eine neue Transaktion wird implizit gestartet.

Oracle bietet die Möglichkeit, mit autonomen Transaktionen zu arbeiten. Sie werden von einer anderen Transaktion gestartet und sind nicht davon abhängig, ob die Basistransaktion mit `COMMIT` oder mit `ROLLBACK` abgeschlossen wird. Eine autonome Transaktion besitzt folgende Eigenschaften:

- Nicht abgeschlossene Änderungen der Haupttransaktion sind in der autonomen Transaktion nicht sichtbar.
- Sie können weitere autonome Transaktionen starten.

Alle aktiven Transaktionen finden Sie in der View `V$TRANSACTION`. Die SQL-Abfrage in Listing 2.19 gibt die Transaktionen aus. Über die Startzeit können die zugehörigen Sessions und SQL-Anweisungen identifiziert werden.

```
SQL> SELECT b.sid, b.serial#,b.username,b.status,
  2  c.sql_text,e.object_name,a.start_time
  3  FROM v$transaction a,v$session b,v$sql c,
     v$locked_object d,all_objects e
  4  WHERE a.ses_addr = b.SADDR
  5  AND b.prev_sql_addr=c.address(+) AND
     b.prev_hash_value=c.hash_value(+)
  6  AND b.prev_child_number = c.child_number(+) AND
     b.prev_sql_id=c.sql_id
  7  AND d.object_id=e.object_id AND d.session_id=b.sid(+);
  SID USER STATUS   SQL_TEXT             OBJECT_N START_TIME
----- ---- -------- -------------------- -------- --------------------
  237 SYS  INACTIVE UPDATE kb SET text=' KB       01/06/19 16:35:33
```

```
                    New Headline' WHERE
                    id = 1
```

Listing 2.19: Offene Transaktionen abfragen

Checkpoint

Ein Checkpoint schreibt alle geänderten Datenblöcke aus dem Buffer Cache in die Tablespaces auf die Disk. Mit einem Checkpoint soll Folgendes erreicht werden:

- Regelmäßiges Schreiben von Änderungen aus dem Buffer Cache in die Tablespaces
- Die Zeit für den Recovery-Prozess im Fehlerfall reduzieren
- Wird für bestimmte Operationen benötigt (z.B. *shutdown immediate*)

Ein Checkpoint wird in folgenden Situationen angefordert:

- Beim Herunterfahren der Instanz mit den Optionen `normal`, `transactional` oder `immediate`
- Vorbereitung einer Online-Sicherung mit dem Befehl `ALTER {DATABASE|TABLESPACE} BEGIN BACKUP`
- Setzen einer Tablespace in den Status `OFFLINE`
- Automatischer Wechsel einer Online-Redo-Log-Gruppe
- Manuelle Auslösung mit dem Befehl `ALTER SYSTEM CHECKPOINT`

Folgende Typen eines Checkpoints können ausgelöst werden:

- *Thread Checkpoint:* Es werden alle geänderten Buffer der Instanz auf die Disk geschrieben.
- *Tablespace Checkpoint:* Es werden die geänderten Buffer einer Tablespace auf die Disk geschrieben.
- *Datafile Checkpoint:* Es werden die geänderten Buffer eines Datafiles gespeichert.
- *Incremental Checkpoint:* Hat das Ziel, einen Teil der geänderten Buffer auf die Disk zu schreiben, um große Checkpoint-Operationen zu vermeiden oder freie Buffer zu schaffen.

Tipp

Vermeiden Sie zu kleine Intervalle zwischen den Checkpoints durch eine hinreichende Größe der Online-Redo-Log-Dateien. Bei jedem Log Switch wird ein Checkpoint angefordert.

Die Häufigkeit von Checkpoints lässt sich auch mit den folgenden Parametern steuern:

- `log_checkpoint_timeout`: Maximale Zeit in Sekunden zwischen zwei Checkpoints. Ist sinnvoll für Systeme mit geringem Transaktionsaufkommen.
- `log_checkpoint_interval`: Anzahl von 512-KB-Blöcken, nachdem ein Checkpoint ausgelöst wird.
- `fast_start_mttr_target`: Anzahl von Sekunden, die die Datenbank für das Crash Recovery benötigt.

Je häufiger ein Checkpoint ausgeführt wird, desto geringer ist die Zeit für das Crash Recovery. Andererseits belastet jeder Checkpoint die Ressourcen des Systems. Es gilt also, einen Kompromiss zwischen Recovery-Zeit und Checkpoint-Häufigkeit zu finden.

Crash Recovery (Instance Recovery)

Beim Crash Recovery werden die Daten aus den Online-Redo-Log-Dateien angewandt, um die Konsistenz der Datenbank wiederherzustellen. In der Regel ist eine Fehlersituation oder Situation vorausgegangen, die zu einem inkonsistenten Zustand der Datenbank geführt hat. Dies kann zum Beispiel ein *shutdown abort* sein. Mit dem Crash Recovery wird die Datenbank wieder in einen konsistenten Zustand versetzt.

Hinweis

Für ein Crash Recovery sind keine Backup-Dateien erforderlich. Es kann komplett mit den Online-Redo-Log-Dateien durchgeführt werden. Archived-Redo-Log-Dateien werden ebenfalls nicht benötigt.

Oracle erkennt automatisch, ob ein Instance Recovery erforderlich ist. Wird ein Crash Recovery durchgeführt, dann erfolgt ein Eintrag in der Alert-Log-Datei.

```
ALTER DATABASE OPEN
Mon Jan 06 18:37:18 2019
Beginning crash recovery of 1 threads
 parallel recovery started with 2 processes
. . .
Completed crash recovery at
 Thread 1: logseq 1160, block 47258, scn 5200809
 38 data blocks read, 38 data blocks written, 44 redo k-bytes read
```

Listing 2.20: Eintrag des Crash Recovery in der Alert-Log-Datei

Ob ein Crash Recovery erforderlich ist, wird am Status der Online-Redo-Log-Dateien festgemacht. Verantwortlich dafür ist der SMON-Prozess. Das Recovery findet in zwei Phasen statt:

1. *Roll Forward:* Es werden alle Änderungen zwischen dem letzten Checkpoint und dem Ende der aktuellen Online-Redo-Log-Datei angewandt. Die Anwendung erfolgt dabei auch auf die `UNDO`-Segmente. Am Ende enthalten die Datenblöcke auch alle Änderungen von Transaktionen, die nicht mit `COMMIT` abgeschlossen wurden. Um die Datenbank auf einen konsistenten Stand zu bringen, müssen diese Änderungen zurückgerollt werden.
2. *Rollback:* Das Rollback der nicht abgeschlossenen Transaktionen erfolgt mit den Informationen aus den `UNDO`-Segmenten. Es wird so lange zurückgerollt, bis die SCN der Checkpoint-Position erreicht wird. Die Checkpoint-Position garantiert, dass alle Änderungen mit niedrigerer SCN in die Datafiles geschrieben wurden.

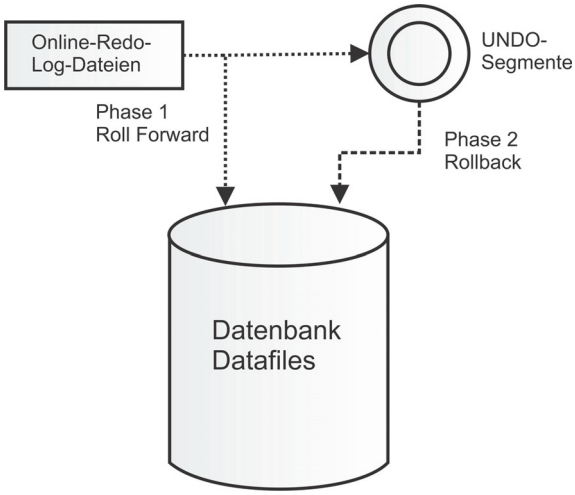

Abb. 2.7: Phasen im Crash Recovery

Nach diesem Prinzip sind nach dem Recovery alle zum Crash-Zeitpunkt abgeschlossenen Transaktionen in der Datenbank gespeichert. Alle Änderungen nicht abgeschlossener Transaktionen wurden zurückgerollt.

Isolation Level

Die Oracle-Datenbank kennt die folgenden drei Isolation-Level:

- *Read Committed (Standard):* Jede SQL-Abfrage sieht nur die Daten, die vor Beginn der SQL-Anweisung (nicht der Transaktion!) durch abgeschlossene Transaktionen (`COMMIT`) anderer Sessions eingegangen sind. Das Prinzip wird »Lesekonsistenz« genannt. Ein Schreibkonflikt entsteht, wenn eine andere Session eine Änderung auf dieselbe Ressource durchführen will. Die Ressource bleibt so lange gesperrt, bis die Transaktion abgeschlossen ist.
- *Serializable:* Jede SQL-Abfrage sieht nur die Daten, die vor Beginn der Transaktion (nicht der SQL-Anweisung!) durch abgeschlossene Transaktionen (`COMMIT`) anderer Sessions eingegangen sind. Das Isolation Level funktioniert in einer Art und Weise, als ob keine anderen Benutzer auf der Datenbank wären.
- *Read Only:* Das Isolation-Level ist vergleichbar mit »Serializable«. Zusätzlich erlauben Read-only-Transaktionen nicht, dass die Daten durch andere Sessions verändert werden.

In der Praxis sollte der Standard beibehalten werden. Andere Isolation Level machen nur in wenigen Situationen Sinn.

Lesekonsistenz

Erinnern Sie sich noch an die Frage am Anfang des Abschnitts im Zusammenhang mit der lang laufenden SQL-Abfrage? Sie sollte spätestens jetzt beantwortet sein. Oracle garantiert im Isolation-Level *Read Committed* Lesekonsistenz. Alle Änderungen, die nach Beginn der SQL-Anweisung eingegangen sind, werden für das Ergebnis nicht berücksichtigt.

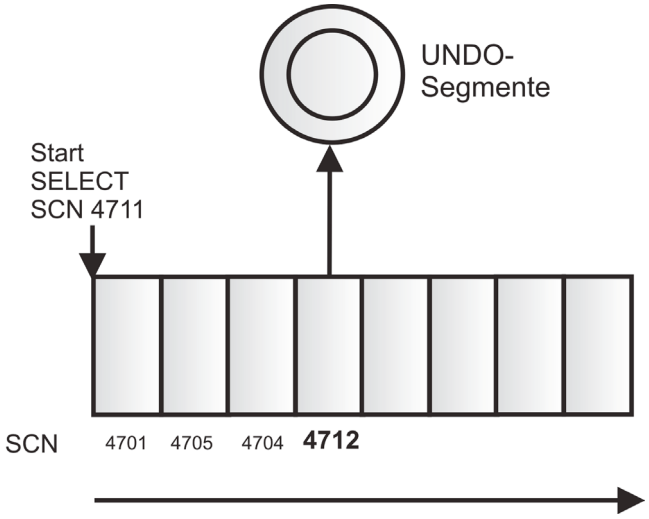

Abb. 2.8: Umsetzung der Lesekonsistenz über UNDO-Segmente

In diesem Zusammenhang tritt auch der Fehler »ORA-01555 Snapshot too old« auf. UNDO-Segmente werden regelmäßig freigegeben, um Platz für neue Transaktionen zu schaffen. Der Fehler tritt auf, wenn ein UNDO-Segment, das für die Lesekonsistenz benötigt wird, überschrieben wurde. Die SELECT-Anweisung bricht ab. Zwar gibt es einen Datenbankparameter UNDO_RETENTION, allerdings ist die Retention nicht garantiert. Werden dringend freie UNDO-Segmente benötigt, werden sie überschrieben, auch wenn die Aufbewahrungszeit noch nicht erreicht ist.

Der Fehler tritt naturgemäß dann auf, wenn lang laufende SELECT-Anweisungen auf ein hohes Transaktionsvolumen treffen.

Tipp

Um den Fehler »Snapshot too old« zu vermeiden, sollte als erste Maßnahme der UNDO-Tablespace vergrößert werden. Damit sinkt die Wahrscheinlichkeit, dass UNDO-Segmente zu schnell überschrieben werden. Prüfen Sie auch, ob SQL-Optimierung möglich ist und damit die Laufzeit der SQL-Anweisung verkürzt werden kann.