akadia Information Technology AG

Oracle Survival Guide

«When a Crisis is in Porgress, it
is easy to make a simple mistake»

Ein Rezepthandbuch für nicht Alltägliches

Der vorliegende «Oracle Survival Guide» soll helfen, nicht alltägliche (Not)-Situationen im Zusammenhang mit Oracle zu meistern. Es ist kein Lehrbuch, sondern eher ein Leitfaden für System-Administratoren. Datenverlust und deren Wiedergewinnung bilden den Schwerpunkt des Inhalts. Um diese Aufgabe zu erfüllen, muß ein Verständnis der Oracle Architektur vorhanden sein. Aus diesem Grunde werden die wesentlichen Merkmale von Oracle kurz erläutert.

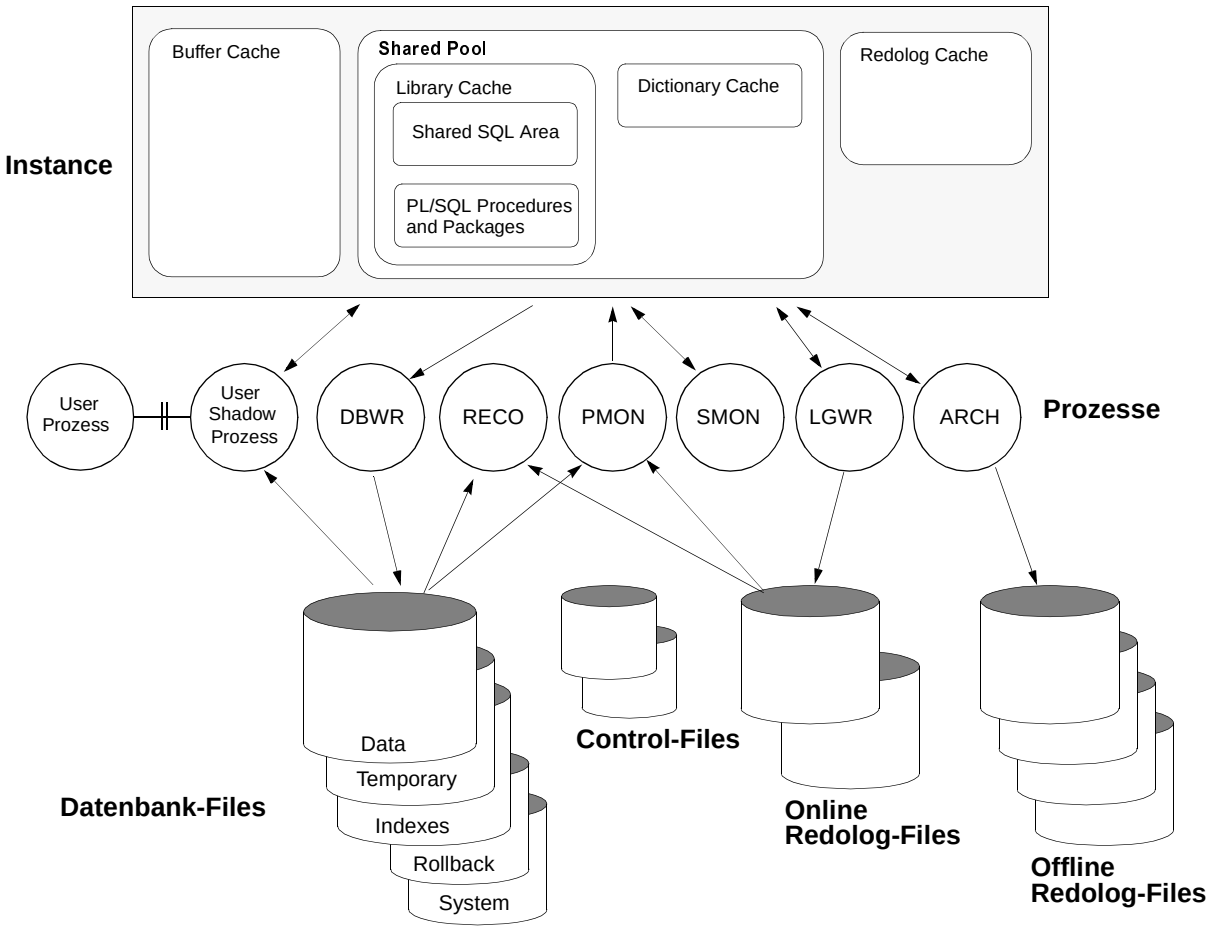
Akadia AG
Zieglerstrasse 34
CH-3007 Bern
Tel: 031 385 30 15 / Fax: 033 345 02 42
EMail: Martin.Zahn@akadia.ch

Inhalt

1. Oracle Architektur

Dedicated Server

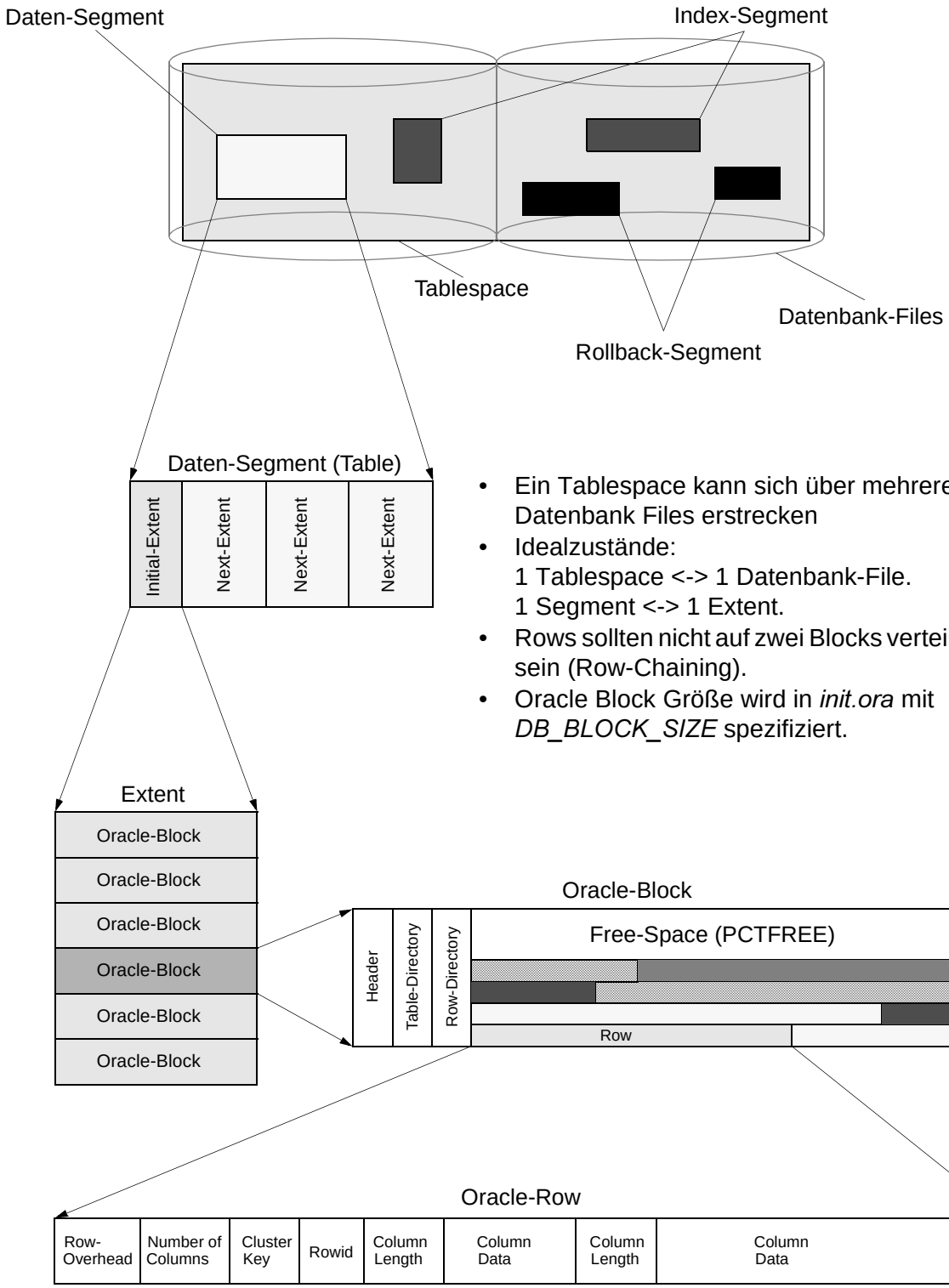
Eine Oracle Datenbank besteht physikalisch aus einem fest allozierten Shared-Memory, der sogenannten SGA (System Global Area) und den Datenbankfiles. Im Umgang mit Oracle befasst man sich aber mehrheitlich mit logischen Größen wie Tablespaces, Segmenten usw.



Kurzbeschrieb

Wird eine Oracle Datenbank mittels ***svrmgrl startup*** gestartet, so wird als erstes im Memory des Rechners Speicherplatz alloziert. Damit ist die Instance (Aktive SGA) mit den Bereichen Database Buffer Pool, Shared Pool und Redolog Buffer Cache aktiv. Eine Kontrolle der SGA kann mit dem Kommando ***ipcs -m*** oder aus SVRMGRL mittels ***show sga*** vorgenommen werden. Zusammen mit dem Kreieren der SGA erfolgt auch das Aufstarten der Oracle Background-Prozesse DBWR, PMON, SMON und LGWR. Der Online-Backup Archiver ARCH wird nur gestartet wenn die Datenbank im ARCHIVELOG Modus betrieben wird. Als zweiten Schritt beim Datenbank-Startup erfolgt das Öffnen der Controlfiles, (mounten der Datenbank). Die Controlfiles enthalten statische Informationen der Datenbank wie Datenbank-Name, Datenbank-Files, Sequenz-Nummern der Offline Redolog Files usw. Als letzten Schritt erfolgt das Öffnen der Datenbank- und Redolog-Files (öffnen der Datenbank). Damit ist die Datenbank betriebsbereit und Userprozesse können gestartet werden. Auf einer Workstation können gleichzeitig mehrere Datenbanken gleichzeitig aktiv existieren. Um eine bestimmte Datenbank zu nutzen, muß die Umgebungsvariable ORACLE_SID auf den gewünschten Datenbanknamen eingestellt werden.

Datenbank Struktur



- Ein Tablespace kann sich über mehrere Datenbank Files erstrecken
- Idealzustände:
  1 Tablespace <-> 1 Datenbank-File.
  1 Segment <-> 1 Extent.
- Rows sollten nicht auf zwei Blocks verteilt sein (Row-Chaining).
- Oracle Block Größe wird in *init.ora* mit *DB_BLOCK_SIZE* spezifiziert.

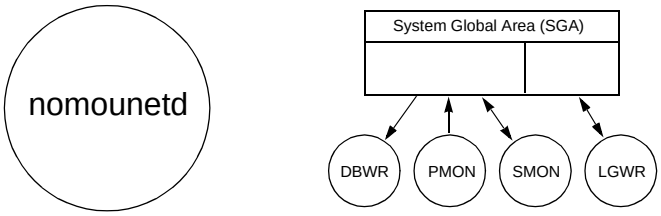
Oracle unterscheidet zwischen logischen Größen (Tablespace, Segmente) und physikalischen Größen (Files, Extents, Blocks). Die physikalischen Größen werden beim Spezifizieren der logischen Größen angegeben (CREATE ...). Segmente sind Tables, Indexe, Rollback- und temporäre Segmente sowie Cluster. Oft wird dazu auch der Begriff Datenbank-Objekt benutzt.

2. Startup und Shutdown der Datenbank

Das Starten und Stoppen einer Datenbank gehört zu den privilegiertesten Kommandos und kann deshalb nur von System privilegierten DBA's durchgeführt werden (gehören der UNIX Group **dba** an).

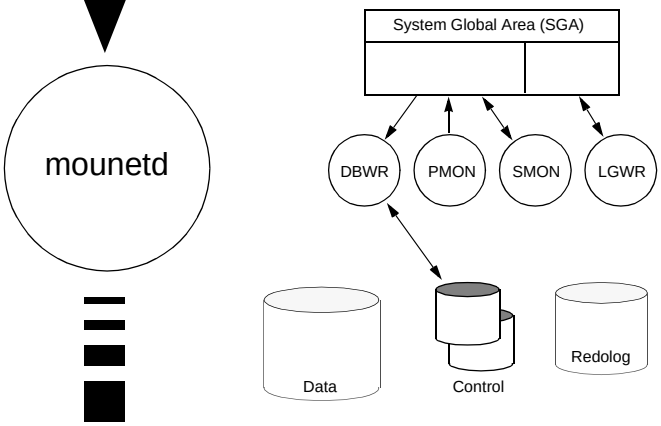
Zustände

Man kennt grundsätzlich 3 Zustände, nomount, mounted und open.



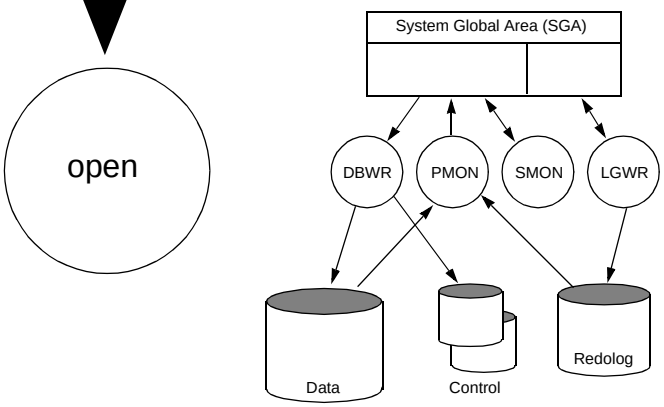
SVRMGRL> **startup nomount**

- Kreieren einer Datenbank



SVRMGRL> **startup mount**

- Add, Drop, Rename von Datenbankfiles.
- Wechseln zwischen ARCHIVELOG und NONARCHIVELOG



SVRMGRL> **startup**

- Normaler Betrieb

3. Manipulationen mit Datenbankfiles

3.1 Datafiles

Oracle Datafiles sind normale UNIX-Files, welche vollständig unter der Kontrolle des Oracle Kernels stehen. Es wird nie direkt aus einem Userprozess in ein Datenbankfile geschrieben oder gelesen.

Umbenennen, Verschieben von Datenbankfiles

Damit können bestehende Datenbankfiles umbenennt oder in ein anderes Directory verschoben werden.

```
svrmgrl
connect internal
startup mount                    (Instance und Controlfiles aktiv)
cd $ORACLE_HOME/kurs
mv usrB.dbf usersB.dbf
alter database rename file 'old' to 'new';
alter database open;             (Datenfiles aktiv)
```

Offline stellen von Datenbankfiles

Damit kann ein defektes oder verloren gegangenes Datenbankfile aus der Datenbank entfernt werden (... Verlust verkraftbar). Dies gilt nicht für Datenfiles, die den System Tablespace oder Rollbacksegmente enthalten.

```
svrmgrl
connect internal
startup mount
alter database archivelog;
alter database datafile 'XXX.dbf' offline;
alter database noarchivelog;
alter database open;
```

Ein Zugriff auf Tablespacedaten in diesem File ist nicht mehr möglich. Die Datenbank kann nun ohne das Datenfile betrieben werden.

Verlust von Datenbankfiles

Wird die Datenbank im NOARCHIVELOG Mode betrieben, so verliert man die Transaktionen seit dem letzten «Cold-Backup». Es gibt nun zwei Möglichkeiten, den alten Zustand wieder herzustellen:

Variante 1 Backup aller Files incl. Control und Redologs auf einen älteren Zustand.

Variante 2 Controlfiles neu erstellen, dabei wird das verlorene Datenbankfile nicht mehr erwähnt, man hat einen neueren Zustand aber ohne das verlorene Datenbankfile.

```
svrmgrl
startup nomount
create controlfile database B
logfile   group 1 ('/oracle/kurs/log_group1_1B.dbf',
                   '/oracle/kurs/log_group1_2B.dbf')size 50k,
          group 2 ('/oracle/kurs/log_group2_1B.dbf',
                   '/oracle/kurs/log_group2_2B.dbf') size 50k
                   noresetlogs
datafile           '/oracle/kurs/systB.dbf' size 7M,
                   '/oracle/kurs/rbsB.dbf' size 2M,
                   '/oracle/kurs/tempB.dbf' size 2M;
alter database open;
```

Die Datenbank kann ohne die Datenbankfiles *tools.dbf* und *users.dbf* betrieben werden. Die Tablespaces sind noch aktiv und müssen nun noch gelöscht werden. Mit NORESETLOGS wird der Inhalt der Online Redolog Files beachtet. Die Tablespaces werden nun im Controlfile und im Datadictionary als "MISSING" deklariert und müssen noch gelöscht werden.

Zustand der Datenbankfiles vor dem Löschen der Tablespaces

```
Datafiles in Data Dictionary (View: dba_data_files)

Tablespace Datafile                                File-ID        [KB] Status
---------- ------------------------------------ ------- ----------- ---------
SYSTEM     /oracle7/kurs/systB.dbf                    1       5,243 AVAILABLE
RBS        /oracle7/kurs/rbsB.dbf                     2       2,097 AVAILABLE
TEMP       /oracle7/kurs/tempB.dbf                    3       2,097 AVAILABLE
TOOLS      MISSING0004                                4       2,097 AVAILABLE
USERS      MISSING0005                                5       2,097 AVAILABLE

Datafiles in Controlfile (Table: v$dbfile)

Datafile                                     File-ID
---------------------------------------- -------
/oracle7/kurs/tempB.dbf                        3
/oracle7/kurs/rbsB.dbf                         2
/oracle7/kurs/systB.dbf                        1
MISSING0004                                    4
MISSING0005                                    5
```

Entfernen der Tablespaces aus dem Controlfile und DataDictionary.

```
svrmgrl
connect internal
drop tablespace tools including contents;
drop tablespace users including contents;
```

Zustand der Datenbankfiles nach dem Löschen der Tablespaces

```
Datafiles in Data Dictionary (View: dba_data_files)

Tablespace Datafile                                File-ID        [KB] Status
---------- ------------------------------------ ------- ----------- ---------
SYSTEM     /oracle7/kurs/systB.dbf                    1       5,243 AVAILABLE
RBS        /oracle7/kurs/rbsB.dbf                     2       2,097 AVAILABLE
TEMP       /oracle7/kurs/tempB.dbf                    3       2,097 AVAILABLE

Datafiles in Controlfile (Table: v$dbfile)

Datafile                                     File-ID
---------------------------------------- -------
/oracle7/kurs/tempB.dbf                        3
/oracle7/kurs/rbsB.dbf                         2
/oracle7/kurs/systB.dbf                        1
```

Damit sind die korrupten Datenbankfiles endgültig aus der Datenbank eliminiert und der Inhalt der Controlfiles stimmt mit dem Datendictionary wieder überein.

Beachte:

> Um Controlfiles neu anlegen zu können muss die Struktur der Datenbank exakt bekannt sein. Es empfiehlt sich also dringend diese für Notfälle zu dokumentieren !

4.2 Controlfiles

Controlfiles beinhalten statische Informationen der Datenbank. Mittels ***startup mount*** werden die Controlfiles geöffnet. Es sind immer mindestens zwei Controlfiles auf verschiedenen Disks anzulegen. Der aktuelle Inhalt der Controlfiles sollte unbedingt dokumentiert werden um im Falle eines
Verlustes die Datenbank neu initialisieren zu können.

Inhalt der Controlfiles

```
select substr(name,1,30), file# from v$dbfile

SUBSTR(NAME,1,30)                   FILE#
------------------------------ ----------
/oracle7/kurs/systB.dbf                 1
/oracle7/kurs/rbsB.dbf                  2
/oracle7/kurs/tempB.dbf                 3
/oracle7/kurs/toolB.dbf                 4
/oracle7/kurs/usrB.dbf                  5

select  substr(member,1,30),group#,status from v$logfile;

SUBSTR(MEMBER,1,30)                GROUP# STATUS
------------------------------ ---------- -------
/oracle7/kurs/log_group1_1B.db          1
/oracle7/kurs/log_group1_2B.db          1
/oracle7/kurs/log_group2_1B.db          2
/oracle7/kurs/log_group2_2B.db          2
```

Im Datendictionary erfolgt das Mapping zwischen den im Controlfile vermerkten Datenbankfiles und den zugeordneten Tablespaces:

Tablespace Mapping auf Datenbankfiles

```
select substr(tablespace_name,1,10),
       substr(file_name,1,30),
       file_id,
       bytes/1000 bytes,
       status
  from dba_data_files
  order by file_id;

SUBSTR(TAB SUBSTR(FILE_NAME,1,30)      FILE_ID  BYTES    STATUS
---------- --------------------------- -------- -------- ---------
SYSTEM     /oracle7/kurs/systB.dbf     1       7340.032 AVAILABLE
RBS        /oracle7/kurs/rbsB.dbf      2       3145.728 AVAILABLE
TEMP       /oracle7/kurs/tempB.dbf     3       1048.576 AVAILABLE
TOOLS      /oracle7/kurs/toolB.dbf     4       1048.576 AVAILABLE
USERS      /oracle7/kurs/usrB.dbf      5       1048.576 AVAILABLE
```

Verlust eines oder mehrerer Controlfiles im NOARCHIVELOG Mode

Da Controlfiles keine dynamischen Daten enthalten können gesicherte Controlfiles unter UNIX zurückkopiert werden. Dies jedoch nur wenn die DB im NOARCHIVELOG Mode betrieben wird.

```
mv ctrl1B.ctl /tmp
svrmgrl
startup

ORA-00205: error in identifying control file '/oracle7/kurs/ctrl1B.ctl'
ORA-07360: sfifi: stat error, unable to obtain information about file.
HP-UX Error: 2: No such file or directory
Attempting to shutdown instance.........ORACLE instance shut down.

mv /tmp/ctrl1B.ctl .
svrmgrl
startup
```

Datenbank startet wieder normal.

Verlust aller Controlfiles im NOARCHIVELOG Mode

Verliert man alle Controlfiles und es existiert kein Backup eines Controlfiles, so können diese nach eigenen Angaben neu erstellt werden. Eine weitere wichtige Anwendung wäre die Rettung einer Datenbank, falls man einen Tablespace oder Datafile verloren hat und über kein Backup-Konzept mit ARCHIVER verfügt, die Datenbank aber ohne diesen verlorenen Tablespace noch retten möchte. Man muß jedoch über die Datenbank genaue Angaben der Files und deren Größen haben.

Verlust aller Controlfiles, Offline Backupkonzept, alle Datenbankfiles und Redologs sind vorhanden und intakt:

```
sqldba lmode=true
connect internal
startup nomount
create controlfile database B
    logfile group 1 ('/oracle7/kurs/log_group1_1B.dbf',
                     '/oracle7/kurs/log_group1_2B.dbf') size 50k,
            group 2 ('/oracle7/kurs/log_group2_1B.dbf',
                     '/oracle7/kurs/log_group2_2B.dbf') size 50k
                     resetlogs
    datafile         '/oracle7/kurs/systB.dbf'          size 7M,
                     '/oracle7/kurs/rbsB.dbf'           size 3M,
                     '/oracle7/kurs/tempB.dbf'          size 1M,
                     '/oracle7/kurs/toolB.dbf'          size 1M,
                     '/oracle7/kurs/usrB.dbf'           size 1M;
alter database archivelog;
alter database open resetlogs;
disconnect
shutdown
```

Die Controlfiles welche im Datenbank-Konfigurationsfile *init<ORASID>.ora* vermerkt sind werden automatisch wieder erstellt (control_files = ...). Das File *init<ORASID>.ora* muß somit nicht editiert werden.

Mit der Option **resetlogs** wird der **Inhalt des aktuellen Redolog-Files ignoriert**. Das erstellte Controlfile enthält ein Flag, damit die Datenbank mittels *alter database open resetlogs* geöffnet werden kann.

Voraussetzung dazu ist jedoch, dass die Datenbank im ARCHIVE Mode betrieben wird. Deshalb muß **unmittelbar nach** dem Kreieren der Datenbank das Kommando *alter database archivelog*; ausgeführt werden. Wird die Option *noresetlogs* gewählt, so erfolgt ein Recovery mit allen archivierten Redolog Files.

Weitere Controlfiles zur Datenbank hinzufügen

Es sollten immer mindestens 2 Controlfiles auf verschiedenen Disks vorhanden sein. Diese werden von Oracle simultan verwaltet. Vorgehen um ein weiteres Controlfile hinzuzufügen:

```
File init<ORASID>.ora editieren, weiteres Controlfile einfügen
svrmgrl shutdown
Umkopieren eines Controlfiles mittels UNIX Kommando
svrmgrl startup
```

Datenbank Dokumentation

Um ein Controlfile mittels *CREATE CONTROLFILE ...* zu erstellen muss die Datenbankstruktur bekannt sein. Man kann auf einfache Weise das gesamte *CREATE CONTROLFILE ...* Statement generieren. Damit wird im DUMP_DEST Directory ein Script mit dem aktuellen *CREATE CONTROLFILE ...* Statement generiert.

```
svrmgrl
connect internal
alter database backup controlfile to trace;
```

Wichtigste Punkte zu Controlfiles

Nachfolgend die wichtigsten Punkte betreffend Controlfiles

- Inhalt der Controlfile :- Namen der Datenbankfiles
  - Checkpoint Information aus Redologfile
  - On/Offline Status der Tablespaces
  - Log Sequence Nummer (ARCHIVELOG)
- Mindestens ein Controlfile muß immer online sein
- Ohne Controlfile kann die Datenbank nicht gestartet werden
- Im ARCHIVELOG Mode muss ein Controlfile mittels ***alter database backup controlfile to ...*** gesichert werden, damit dieses konsistent ist. Das so erstellte Controlfile kann dann mit normalen UNIX Kommandos gesichert werden.
- Mittels *create controlfile ...* kann ein neues Controlfile erstellt werden (Katastrophenfall). Dazu muss allerdings die exakte Struktur der Datenbank bekannt sein.
- Optimal sind zwei Controlfiles auf verschiedenen Disks.

4.3 Redologfiles

Redologfiles

Redologfiles sichern die Datenbank vor «Media failures», sie beinhalten alle Datenbank Änderungen. Sobald eine Transaktion abgeschlossen ist («commited»), schreibt der *lgwr* Prozess die Redolog Buffer ins Redologfile. Während einem Recovery werden die Redologfiles zurückappliziert bis alle «commited» Transaktionen wieder hergestellt sind (Rollforward).

Rollbacksegmente

Rollbacksegmente beinhalten die Änderungen einer Transaktion, solange diese noch nicht abgeschlossen ist («uncommited»). Rollbacksegmente haben zwei Aufgaben zu erfüllen

Gewährleistung der Lesekonsistenz

Jeder Benutzer sieht die Daten so wie zu Beginn der Transaktion, eventuelle Änderungen während der Transaktion durch einen anderen Benutzer sieht er nicht. Dies hat den großen Vorteil, dass Oracle dadurch nicht ganze Tables locken muß.

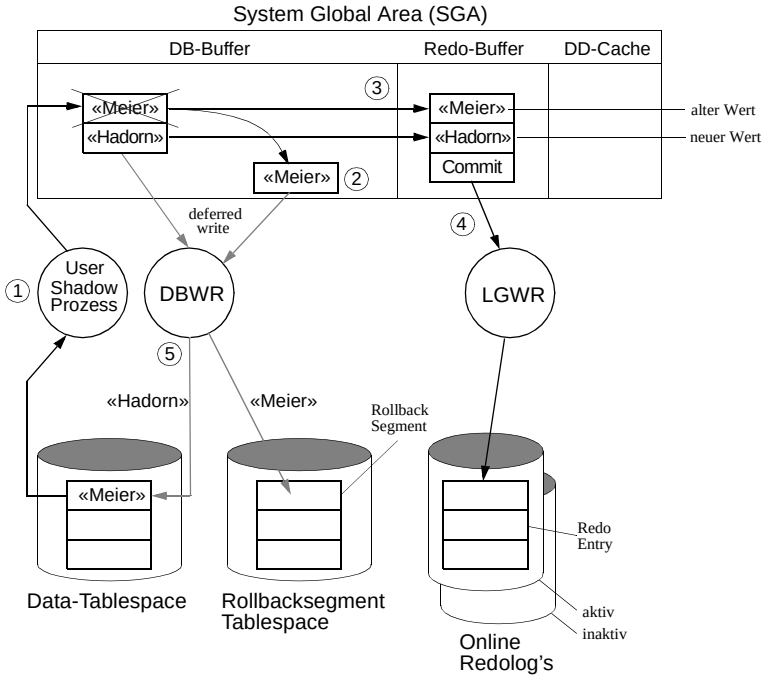
Rollback von uncommited Transaktionen

Bei abgebrochenen Transaktionen werden die Rollbacksegmente benutzt um die Transaktion rückgängig zu machen. Nach jedem Recovery mit einem Rollforward der Redologfiles folgt ein Rollback der Rollbacksegmente. Das heißt «uncommited» Transaktionen werden rückgängig gemacht.

Ablauf eines UPDATE mit einem Commit

Nachfolgend der Ablauf eines UPDATE mit nachfolgendem COMMIT

```
❒ update emp
  set name = 'Hadorn'
  where ename = 'Meier'
❒ commit
```



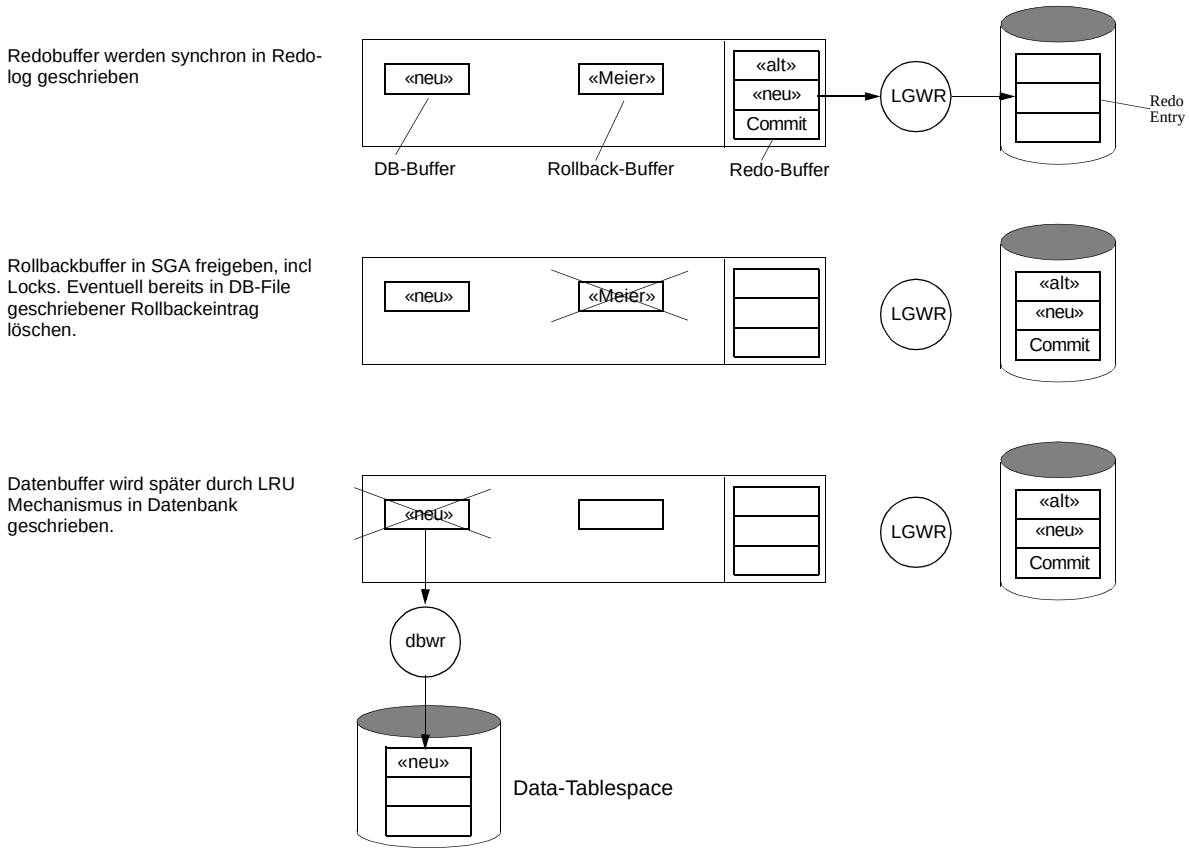
1. User Shadowprozess liest den benötigten Block aus dem Datenbankfile in die SGA, sofern sich dieser nicht schon dort befindet.
2. Alter Wert = «Meier» wird in Rollbacksegment Buffer in der SGA für eventuelle Rollbacks gesichert.
3. Alter und neuer Wert wird im Redobuffer eingetragen, gleichzeitig wird der Datenbuffer mit dem neuen Wert überschrieben.
4. Erfolgt das COMMIT so wird der Redobuffer vom *lgwr* in das freie Redo-Entry im aktiven Redologfile geschrieben.
5. Irgendwann gemäß LRU Algorithmus schreibt der *dbwr* den Daten- und Rollbackbuffer in die Table bzw. in das Rollbacksegment auf der Disk.

Ablauf eines Commit

Irrtümlich wird oft angenommen, dass bei einem COMMIT die geänderten Datenbuffer vom *dbwr* in die Datenbankfiles übertragen werden.

Dies ist nicht der Fall !

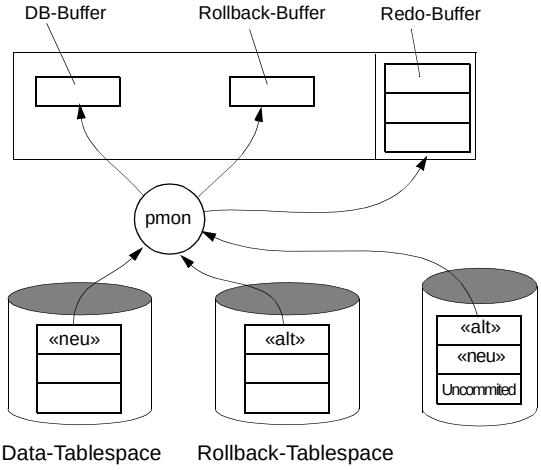
Bei einem COMMIT werden die Redobuffer vom *lgwr* in das aktive Redologfile übertragen. Bei vielen Transaktionen kann es also sein, dass der *lgwr* der aktivste Prozess ist und dass die Redologfiles sehr oft geschrieben werden. Deshalb sollten die Redologfiles auf einer sehr schnellen Disk plaziert werden um eine optimale Performance zu erreichen. Damit jedoch auch der *dbwr* Prozess dazu veranlaßt wird die geänderten Datenbuffer auf die Disk zu schreiben verwendet Oracle den Checkpoint Mechanismus (siehe auch Prozesse *dbwr* und *lgwr*).



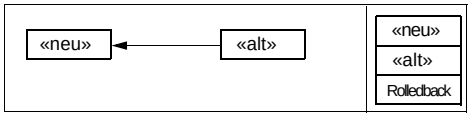
Ablauf eines Rollback

Durch das SQL Kommando ROLLBACK werden alle Transaktionen welche nicht commited sind rückgängig gemacht. Dazu werden die Rollbackbuffer aus den Rollbacksegmenten in der SGA wieder erstellt und der Rollback wird durchgeführt.

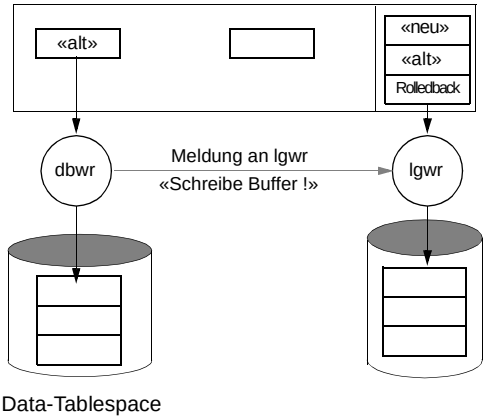
Laden der bereits geschriebenen Buffer in die SGA, Rollbackbuffer aus Rollbacksegmenten, Redobuffer aus Redologfile und Datenbuffer aus Tabelle.



Rollback in der SGA durchführen, Redobuffer werden vertauscht und mit einem «rolled back» Record versehen. Der alte Wert im Rollbackbuffer wird in den Datenbuffer übertragen, er überschreibt den Wert des Datenbuffers.



Der *dbwr* Prozess schreibt den alten Wert in die Datenbank zurück, gleichzeitig schreibt auch der *lgwr* seinen Redobuffer ins aktive Redofile.



Beachte:

> Das Redolog File gewährt in jedem Fall die Konsitenz, indem jeder Redobuffer gekennzeichnet ist mit «commited», «uncommited» oder «rolled back»

Online Redolog Files

Es müssen mindestens zwei Redologfiles vorhanden sein, das aktive (Online) und das inaktive (Offline). Redologfiles sollten aus Performance Gründen alle die gleiche Größe aufweisen.

Größe der Redologfiles:

Große Redologfiles mit entsprechend großem LOG_CHECKPOINT_INTERVAL reduzieren die Anzahl Checkpoints. Die Datenbuffer bleiben also länger in der SGA, was eine entsprechend große SGA voraussetzt. Eine Instance Recovery dauert dementsprechend länger, da alle Transaktionen seit dem letzten Checkpoint zurückappliziert werden müssen. Die Größe der Redologfiles ist also abhängig von der gewünschten Performance und der zumutbaren Recoveryzeit beim Start einer abgestürzten Datenbank. Als Mittelwert kann eine Größe von 500K - 800K angenommen werden. Die Größe sollte 5M nicht überschreiten.

Checkpoints

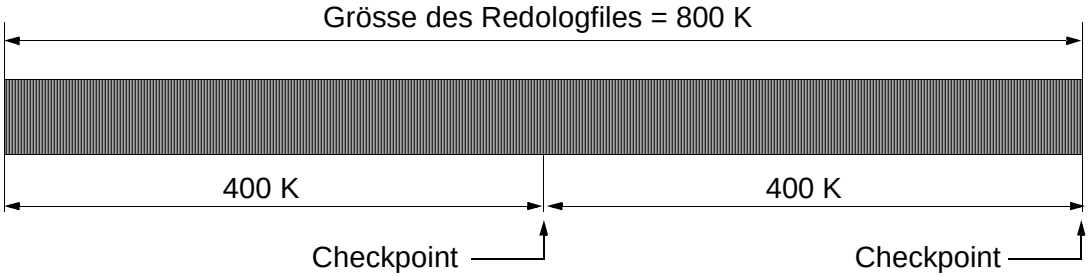
Nachdem eine bestimmte Anzahl Redobuffer vom *lgwr* in das Online Redologfile geschrieben wurden, erfolgt ein sogenannter Checkpoint. Der *dbwr* Prozess wird veranlaßt alle modifizierten Daten- und Rollbackbuffer in die Files zu schreiben, unabhängig ob die Buffer «commited» oder «uncommited» sind. Checkpoints gewährleisten also, dass auch oft benutzte SGA Buffer regelmäßig auf die Disk geschrieben werden, da diese bedingt durch den LRU Algorithmus möglicherweise gar nie geschrieben würden! Die Folge der Checkpoints kann durch den *init.ora* Parameter LOG_CHECKPOINT_INTERVAL eingestellt werden.

Berechnungsbeispiel für LOG_CHECKPOINT_INTERVAL

- Größe Redologfile = 800 KByte
- UNIX Diskblock Size = 512 Byte
- Checkpoint soll erfolgen wenn Redolog zur Hälfte gefüllt ist.

$$\text{LOG\_CHECKPOINT\_INTERVAL} = \frac{\text{Grösse der Redologfiles}}{\text{Interval} * \text{Diskblock Size}}$$

LOG_CHECKPOINT_INTERVAL = 800000 / (2 * 512) = 780



In Oracle7 existiert neu der Parameter LOG_CHECKPOINT_TIMEOUT, mit dem der Checkpoint zeitgesteuert veranlaßt werden kann. Er wird vorallem bei Parallel Server Implementationen benutzt.

Wann erfolgen Checkpoints ?

Checkpoints garantieren, dass in bestimmten Abständen alle DB-Buffer vom LGWR auf die Datenbankfiles geschrieben werden. Der Zeitpunkt des Checkpoints ist auch für das Recovery sehr wichtig, indem alle Redolog-Entries, die älter sind als der Checkpoint für ein Instance Recovery nicht mehr benötigt werden. Das heisst, häufigere Checkpoints ermöglichen eine kürzere Instance Recovery Zeit. Checkpoints führen jedoch zu einem gewissen Overhead und können die Performance verschlechtern. Wann erfolgen aber Checkpoints ?

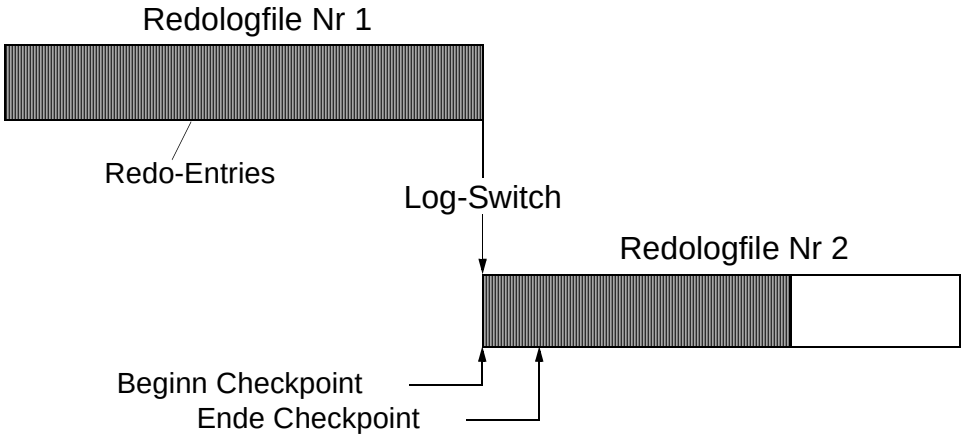
- Bei einem Log-Switch des Redolog-Files. Ist ein Checkpoint «in progress», so überschreibt ein Checkpoint ausgelöst durch einen Log-Switch diesen «running» Checkpoint.
- Der init.ora Parameter LOG_CHECKPOINT_INTERVAL kann so eingestellt werden, dass nach einer gewissen Anzahl Redolog-Buffer ein Checkpoint ausgelöst wird. Sinnvoll bei sehr grossen Redolog-Files.
- Ein BEGIN BACKUP löst immer einen Checkpoint auf den durch den Backup betroffenen Tablespace bzw. dessen File(s) aus.
- Ein ALTER TABLESPACE ts-name OFFLINE NORMAL / TEMPORARY löst immer einen Checkpoint auf den durch ALTER betroffenen Tablespace bzw. dessen File(s) aus.
- Bei einem SHUTDOWN (NORMAL oder IMMEDIATE).
- Bei einem manuellen CHECKPOINT, *alter system switch logfile*;

Logswitch

Beim Wechsel des Redologfiles = Logswitch erfolgt immer ein Checkpoint. Soll also nur am Ende des Redologfiles ein Checkpoint erfolgen, so kann der LOG_CHECKPOINT_INTERVAL Parameter entsprechend groß gesetzt werden. Bei einer Größe des Redologfiles von 800KByte also mindestens 800000 / (1 * 512) = 1562. Bei kleinen Redologfiles reicht ein Checkpoint bei jedem Logswitch, bei grossen Redologfiles sollte man sich über die zulässige Recoveryzeit Gedanken machen und eventuell einen tieferen LOG_CHECKPOINT_INTERVAL verwenden.

Wird der Parameter also ca. auf die doppelte Größe des Redologfiles gesetzt, so erfolgt nur am Ende des Redologfiles ein Checkpoint

Ein Logswitch bezeichnet den Wechsel des aktiven Redologfiles.



Unmittelbar nach einem Logswitch beginnt ein Checkpoint. Während dieser Zeit werden beide Redologfiles für ein eventuelles Instance-Recovery benötigt. Erst zum Zeitpunkt «Ende Checkpoint» wird das inaktive Redologfile durch den Archiver archiviert.

Manueller Logswitch

Ein Logswitch kann auch manuell durchgeführt werden

```
alter system switch logfile;
```

Protokoll im *alert.log*

```
Thread 1 advanced to log sequence 186
  Current log# 2 seq# 186 mem# 0: /oracle7/kurs/log_group2_1B.dbf
  Current log# 2 seq# 186 mem# 1: /oracle7/kurs/log_group2_2B.dbf
```

Manueller Checkpoint

```
alter system checkpoint;
```

Vom manuellen Checkpoint erfolgt kein Eintrag im *alert.log*.

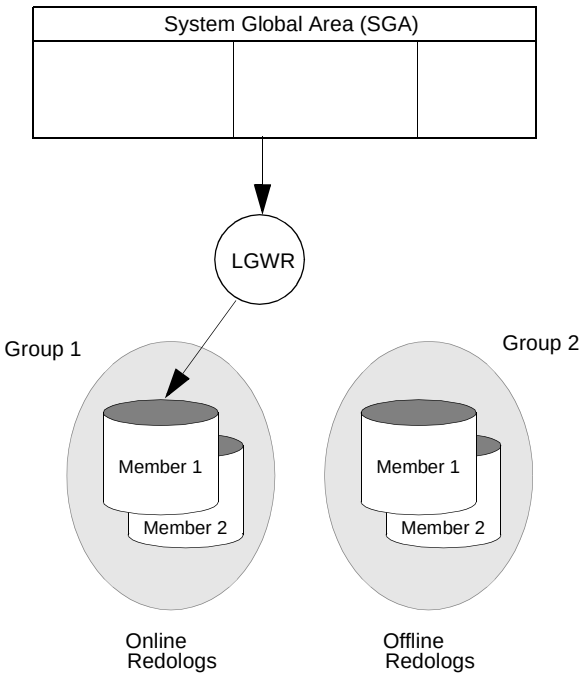
Gespiegelte Redologfiles

Unter Oracle7 wurde mit der Einführung von multiplexed Redologs eine der größten Schwachstellen von Version 6 eliminiert. Verlor man unter Version 6 das aktive Redologfile, so verlor man zwingend sämtliche Transaktionen (commited, uncommited, rolled back) des aktiven Redologfiles, das trotz Onlinebackupkonzept. Besonders schlimm war es, dass auch Transaktionen die commited wurden verloren gingen, **da immer das Redologfile die Konsistenz gewährleistet !**

Dies war für viele kritische Applikationen unzulässig und so wurde Oracle lange Zeit für solche Aufgaben nicht gewählt. Viele grosse Anwender umgingen diesen Mangel, indem sie die Redologfiles auf Betriebssystemebene spiegelten (zB VMS):

Konzept

Mehrere gleichzeitig beschriebene Redologs werden «Group» genannt. Solange ein «Member» einer Group verfügbar ist, arbeitet das System normal weiter. Meldungen werden ins *alert.log* geschrieben. Groups werden fortlaufend durchnumeriert, die Groups erhalten eine eindeutige Nummer als Identifikation. Alle Groups sollten die gleiche Anzahl Members aufweisen, die Members einer Group sollten wenn möglich auf verschiedenen physikalischen Disks liegen. Alle Members einer Group müssen die gleiche Grösse haben.



Kreieren von Redolog Groups beim CREATE DATABASE

Redolog Gruppen werden grundsätzlich beim CREATE Database angelegt, sie können aber auch zu einem späteren Zeitpunkt erweitert werden.

```
create database "B"
maxinstances 1
maxlogfiles  6
character set "WE8DEC"
logfile   group 1('/oracle7/kurs/log_group1_1B.dbf',
                  '/oracle7/kurs/log_group1_2B.dbf') size 800k,
          group 2('/oracle7/kurs/log_group2_1B.dbf',
                  '/oracle7/kurs/log_group2_2B.dbf') size 800k
datafile          '/oracle7/kurs/systB.dbf'          size 7M;
```

Neue Group später hinzufügen

Eine neue Group kann bei laufendem System hinzugefügt werden:

```
alter database B
add logfile group 3
('/oracle7/kurs/log_group3_1B.dbf',
 '/oracle7/kurs/log_group3_2B.dbf') size 50K;
```

Verlust eines Group Members

Das System arbeitet normal weiter, im *alert.log* und *lgwr.trc* erfolgt ein Eintrag.

```
rm /oracle7/kurs/log_group1_2B.dbf
svrmgrl
connect internal
alter system switch logfile(Logswitch manuell auslösen)
```

Eintrag im alert.log

```
Fri Oct  6 08:55:01 1995
Errors in file /oracle7/rdbms/log/lgwr_1089.trc:
ORA-00321: log 1 of thread 1, cannot update log file header
ORA-00312: online log 1 thread 1: '/oracle7/kurs/
log_group1_2B.dbf'
Fri Oct  6 08:55:01 1995
Errors in file /oracle7/rdbms/log/lgwr_1089.trc:
ORA-00313: open failed for members of log group 1 of thread 1
Thread 1 advanced to log sequence 187
```

Neben den Trace- und Alertfiles kann auch die Tabelle *v$logfile* abgefragt werden:

```
select group#, status, substr(member,1,30) from v$logfile;

    GROUP# STATUS  SUBSTR(MEMBER,1,30)
---------- ------- ------------------------------
         1         /oracle7/kurs/log_group1_1B.dfb
         1 INVALID /oracle7/kurs/log_group1_2B.dbf
         2         /oracle7/kurs/log_group2_1B.dbf
         2         /oracle7/kurs/log_group2_2B.dbf
```

Group Member ersetzen

Zuerst muss der invalid Member gedropped werden bevor dieser wieder hinzugefügt werden kann. Eventuell muss ein manueller Logswitch durchgeführt werden wenn die Group gerade aktiv ist.

```
alter database B
drop logfile member '/oracle7/kurs/log_group1_2B.dbf';

ORA-01609: log 1 is the current log for thread 1 - cannot drop
ORA-01517: log member: '/oracle7/kurs/log_group1_2B.dbf'

alter system switch logfile;
alter database B
  add logfile member '/oracle7/kurs/log_group1_2B.dbf' to group 1;
```

CLEAR LOGFILE

Als Alternative kann man kann dazu auch das Kommanda ALTER DATABASE CLEAR LOGFILE benutzen.

```
-rw-r-----   1 oracle dba  5243392 Mar 18 07:16 TOL2_log1A.rdo
-rw-r-----   1 oracle dba  5243392 Mar 18 07:16 TOL2_log1B.rdo
-rw-r-----   1 oracle dba  5243392 Mar 18 07:16 TOL2_log2A.rdo
-rw-r-----   1 oracle dba  5243392 Mar 18 07:16 TOL2_log2B.rdo

rm TOL2_log2B.rdo

svrmgrl
connect internal
select * from v$logfile;

GROUP#     STATUS  MEMBER
---------- ------- ----------------------------------------
         1         /opt/oracle/oradata/TOL2/rdo/TOL2_log1A.rdo
         1         /opt/oracle/oradata/TOL2/rdo/TOL2_log1B.rdo
         2         /opt/oracle/oradata/TOL2/rdo/TOL2_log2A.rdo
         2 INVALID /opt/oracle/oradata/TOL2/rdo/TOL2_log2B.rdo

alter system switch logfile;
alter database clear logfile group 2;
select * from v$logfile;

GROUP#     STATUS  MEMBER
---------- ------- ----------------------------------------
         1         /opt/oracle/oradata/TOL2/rdo/TOL2_log1A.rdo
         1         /opt/oracle/oradata/TOL2/rdo/TOL2_log1B.rdo
         2         /opt/oracle/oradata/TOL2/rdo/TOL2_log2A.rdo
         2         /opt/oracle/oradata/TOL2/rdo/TOL2_log2B.rdo
```

Das Logfile TOL2_log2B.rdo wurde komplett neu angelegt. CLEAR LOGFILE GROUP entspicht einem Dropping und Adding einer Gruppe. Das Kommando kann auch benutzt werden wenn nur zwei Gruppen vorhanden sind. CLEAR LOGFILE kann nicht benutzt werden, wenn die Gruppe ein Media Recovery benötigt. Dies kann mit der View **V$RECOVER_FILE** kontrolliert werden.

Group löschen

Eine gesamte Gruppe kann ebenfalls gelöscht werden, dies sit nur möglich für inaktive Gruppen, dazu muss mit alter system switch logfile auf eine andere Gruppe wechseln.

```
alter database
drop logfile group 3;
```

Beachte: Oracle merkt wenn nur noch zwei Gruppen vorhanden sind, und verhindert ein Löschen. Es werden immer mindestens 2 Gruppen benötigt. Will man trotzdem beispielsweise die Gruppe 1 löschen, so muss zuerst eine dritte Gruppe angelegt werden.

Backup im NOARCHIVELOG Mode

Wird die DB im NOARCHIVE Mode betrieben, so müssen bei geschlossener Datenbank ALLE Files, inklusive den Online-Redolog Files gesichert werden. Damit hat man eine konsistente Datenbank, ein offline Backup bei offener DB ist wertlos.

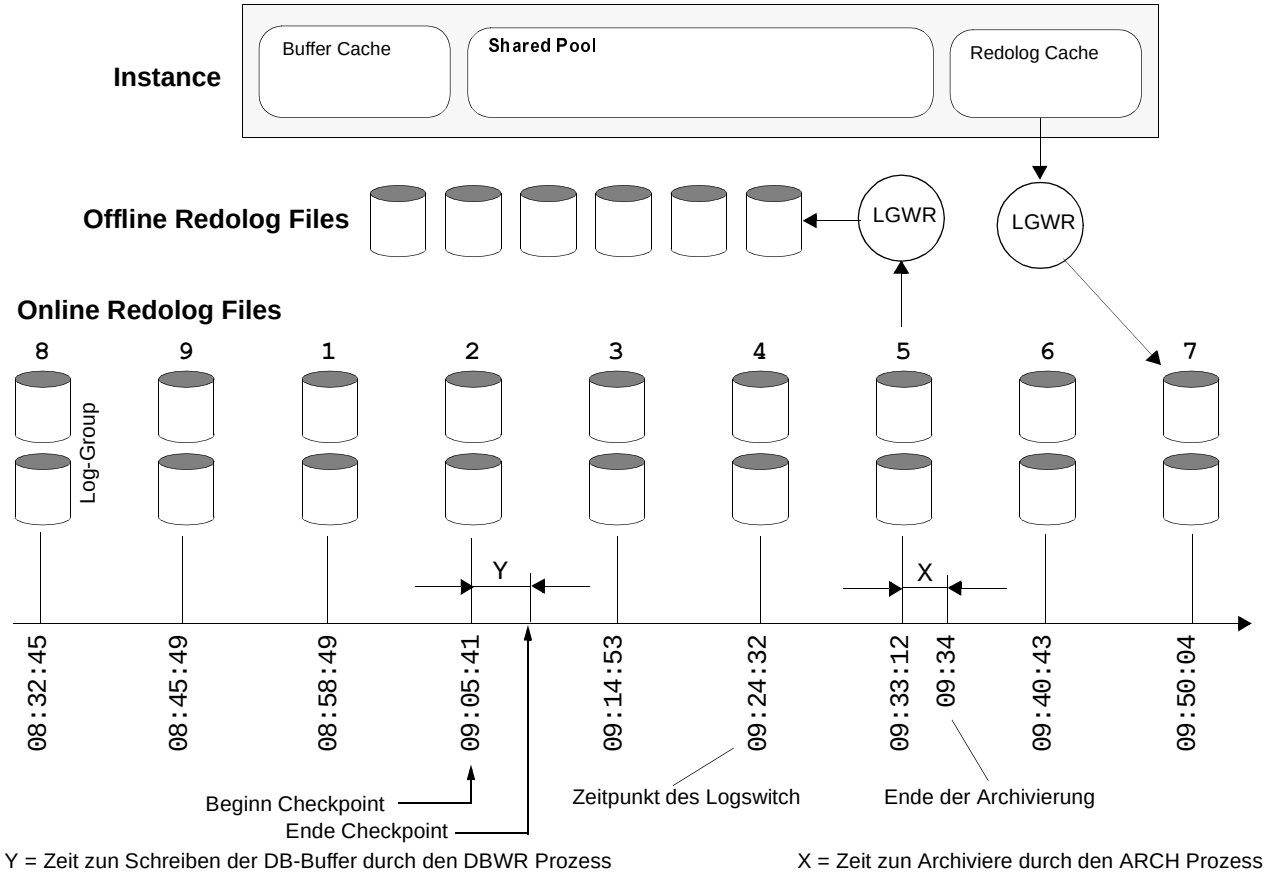
Verlust der Online Redologs

Verliert man die gesamte Redolog Group im NOARCHIVE Mode, so verliert man alle Transaktionen des aktiven Redologs. Die Datenbank stürzt sofort ab und kann nicht mehr gestartet werd. Ein Wiederherstellen ist nicht möglich, auch mittels CLEAR LOGFILE können die Redolog-Files nicht mehr erstellt werden. Wird die DB im ARCHIVELOG Mode betrieben, so muss ein CANCEL based Recovery bis zum letzten Offline Redologfile gefahren werden und die DB kann nun mit RESETLOGS geöffnet werden.

Redolog Timing

Verschiedene zeitliche Abängigkeiten spielen im Zusammenhang mit den Redolog Files ein wichtige Rolle für die Performance. Aus der View V$LOG und den Timestamps der offline Redologfiles können folgende Angaben berechnet werden:

- Zeitpunkt des Log-Switch
- Wie lange schreibt der LGWR Prozess in eine bestimme Log-Group ?
- Wie lange braucht der ARCH Prozess um ein Offline Redologfile zu archivieren ?



Zeitpunkt des Logswitch

Aus V$LOG und dem ALERT.LOG kann man den Beginn des Logswitches erkennen.

```
Set linesize 132
Col grp_member for a50 heading member

Select group#, sequence#,
       Members, archived, status, first_time
  From v$log
/

GROUP# SEQUENCE#   MEMBERS   ARC  STATUS    FIRST_TIME
------ ---------   -------   ---  ------    ----------------
     1      1697   2         YES  INACTIVE  12/08/98 08:58:49
     2      1698   2         YES  INACTIVE  12/08/98 09:05:41
     3      1699   2         YES  INACTIVE  12/08/98 09:14:53
     4      1700   2         YES  INACTIVE  12/08/98 09:24:32
     5      1701   2         YES  INACTIVE  12/08/98 09:33:12
     6      1702   2         YES  INACTIVE  12/08/98 09:40:43
     7      1703   2          NO   CURRENT  12/08/98 09:50:04
     8      1695   2         YES  INACTIVE  12/08/98 08:32:45
     9      1696   2         YES  INACTIVE  12/08/98 08:45:49
```

Wie lange schreibt LGWR ?

Dies ist die Differenz zwischen den Log-Switches:

```
  FIRST_TIME (Group 1): 08:58:49
- FIRST_TIME (Group 9): 08:45:49
--------------------------------
Differenz                  7:00
--------------------------------
```

Wie lange braucht ARCH zu Archivieren ?

Dies kann man aus der Differenz zwischen dem physischem Modification Date und dem Beginn des Logswitches errechnen.

```
-rw-r--- 1 oracle dba 20972032 Dec  8 09:34 prodarch0000001700.arc
-rw-r--- 1 oracle dba 20972032 Dec  8 09:42 prodarch0000001701.arc
-rw-r--- 1 oracle dba 20972032 Dec  8 09:51 prodarch0000001702.arc

  End prodarch0000001700.arc 09:34
- FIRST_TIME (Group 5)       09:33:12
-------------------------------------
Differenz                   00:00:48
-------------------------------------
```

Anzahl Log-Switches Pro Stunde ?

Aus der View V$LOG_HISTORY kann die Anzahl Log-Switches selektiert werden welche pro Stunde über den ganzen Tag verteilt stattfinden. Im Attribut TIME steht wann das erste Log-Entry (lowest SCN) eingetragen wurde, also den Beginn des Log-Switches markiert.

```
select substr(time,1,5) day,
       to_char(sum(decode(substr(time,10,2),'00',1,0)),'99') "00",
       to_char(sum(decode(substr(time,10,2),'01',1,0)),'99') "01",
       to_char(sum(decode(substr(time,10,2),'02',1,0)),'99') "02",
       to_char(sum(decode(substr(time,10,2),'03',1,0)),'99') "03",
       to_char(sum(decode(substr(time,10,2),'04',1,0)),'99') "04",
       to_char(sum(decode(substr(time,10,2),'05',1,0)),'99') "05",
       to_char(sum(decode(substr(time,10,2),'06',1,0)),'99') "06",
       to_char(sum(decode(substr(time,10,2),'07',1,0)),'99') "07",
       to_char(sum(decode(substr(time,10,2),'08',1,0)),'99') "08",
       to_char(sum(decode(substr(time,10,2),'09',1,0)),'99') "09",
       to_char(sum(decode(substr(time,10,2),'10',1,0)),'99') "10",
       to_char(sum(decode(substr(time,10,2),'11',1,0)),'99') "11",
       to_char(sum(decode(substr(time,10,2),'12',1,0)),'99') "12",
       to_char(sum(decode(substr(time,10,2),'13',1,0)),'99') "13",
       to_char(sum(decode(substr(time,10,2),'14',1,0)),'99') "14",
       to_char(sum(decode(substr(time,10,2),'15',1,0)),'99') "15",
       to_char(sum(decode(substr(time,10,2),'16',1,0)),'99') "16",
       to_char(sum(decode(substr(time,10,2),'17',1,0)),'99') "17",
       to_char(sum(decode(substr(time,10,2),'18',1,0)),'99') "18",
       to_char(sum(decode(substr(time,10,2),'19',1,0)),'99') "19",
       to_char(sum(decode(substr(time,10,2),'20',1,0)),'99') "20",
       to_char(sum(decode(substr(time,10,2),'21',1,0)),'99') "21",
       to_char(sum(decode(substr(time,10,2),'22',1,0)),'99') "22",
       to_char(sum(decode(substr(time,10,2),'23',1,0)),'99') "23"
  from v$log_history
 group by substr(time,1,5)
/
```

```
DAY   00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23
---   -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
12/01  2  2  4  7  8  7  7  8  4  3  4  3  3  3  2  3  3  2  3  2  3  2  2  2
12/02  2  1  3  6  4  5  3  2  3  3  4  2  3  3  3  2  3  2  3  3  2  3  2  2
12/03  2  1  4  7  6  3  3  4  4  4  6  3  1  3  5  1  0  0  4  3  3  4  3  2
12/04  2  2  2  7  9  7  5  4  5  4  5  4  4  3  5  4  5  4  3  3 10  3  3  2
12/05  2  2  2  4  5  5  6  7  6  3  5  3  3  3  5  3  5  3  3  3  8  2  2  2
12/06  2  1  3  3  4  4  5  7  5  3  3  2  2  2  2  2  4  2  2  3  8  3  2  2
12/07  1  1  3  3  4  5  3  5  3  2  3  4  3  3  4  4  5  3  3  4  7  3  2  3
12/08  2  1  3  4  4  5  5  3  4  3  3  3  3  3  2  3  2  3  2  3  6  2  2  2
12/09  2  1  2  4  4  4  5  3  3  3  3  3  2  3  2  3  3  2  3  3  2  3  2  2
```

Daraus kann man die Belastung des Systems erkennen, je mehr Log-Swichtes pro Stunde stattfinden, desto mehr Transaktionen finden in der Datenbank statt.

Dauer des Checkpoints

Um die Dauer eines Checkpoints zu bestimmen muss der init<SID>.ora Parameter LOG_CHECKPOINTS_TO_ALERT = TRUE gesetzt werden. Damit werden Checkpoints im ALERT.LOG geloggt.

```
Thu Mar 18 21:23:36 1999
Beginning log switch checkpoint up to RBA [0x10.2.10], SCN: 0x0000.00003b5d
Thread 1 advanced to log sequence 16
  Current log# 2 seq# 16 mem# 0: /opt/oracle/oradata/TOL2/rdo/TOL2_log2A.rdo
  Current log# 2 seq# 16 mem# 1: /opt/oracle/oradata/TOL2/rdo/TOL2_log2B.rdo
Thu Mar 18 21:23:43 1999
Completed checkpoint up to RBA [0x10.2.10], SCN: 0x0000.00003b5d
```

Man erkennt den Beginn des Checkpoints um 23:36 und das Ende um 23:43. Der Logwitch Checkpoint löst den Checkpoint aus wie ersichtlich ist.

Wichtigste Punkte Redologfiles

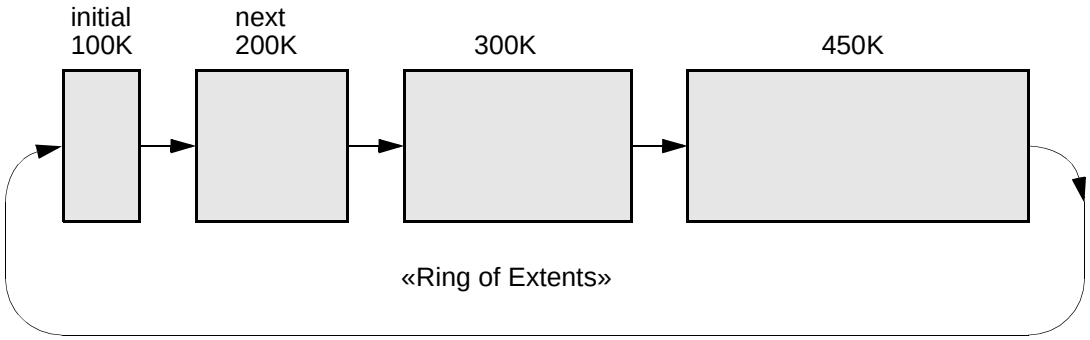
- Der Verlust einer Online Redolog Group bedeutet **immer einen Datenverlust**, mindestens bis zum letzten Checkpoint. Deshalb nicht allzu große Redologfiles wählen oder den Paratmeter LOG_CHECKPOINT_INTERVAL so setzen, dass nicht nur bei einem Log-Switch ein Checkpoint erfolgt. **Redologfiles auf einer separaten, schnellen Disk plazieren !**
- Rollbacksegmente beinhalten Daten, welche noch nicht «commited» sind. Bei Rollbacks werden diese Daten zurückappliziert. **Der Inhalt der Rollbacksegmente ist durch die Redologfiles geschützt.**

4.4 Rollbacksegmente

Sie dienen dazu uncommited Transaktionen bei Bedarf rückgängig zu machen (Rollback). Im weiteren gewährleisten sie eine lesekonsistente Sicht der Daten. Beim Kreieren der Datenbank wird automatisch das Rollbacksegment SYSTEM im System Tablespace erstellt. Um weitere Tablespaces zur Datenbank hinzufügen zu können, muß mindestens ein weiteres Rollbacksegment erstellt werden

Aufbau der Rollbacksegmente

Rollbacksegmente sind als «Ring of Extents» aufgebaut. Sie beinhalten relativ temporäre Daten. Blocks werden zyklisch immer wieder neu benutzt:



Storage Parameter für gemischte Transaktionen:

```
initial = 100K
next = 200K
minextents = 2
```

Große Transaktionen «blasen» die Rollbacksegmente auf, was zu vielen Extents führt (Fragmentierung). Solche Rollbacksegmente sollten regelmäßig gedropped und neu erstellt werden. Initial muß ein Rollbacksegment jedoch aus mindestens 2 Extents bestehen (minextents = 2).

Transaktionen pro Rollbacksegment

Jede Transaktion ist genau einem Rollbacksegment zugeteilt, sie kann nicht über mehrere Rollbacksegmente verteilt sein. Ein Rollbacksegment kann jedoch mehrere Transaktionen simultan verarbeiten. Zur Berechnung der Anzahl Rollbacksegmente kann die folgende Faustregel herangezogen werden: Pro Oracle User sollten ca. 4 Transaktionen pro Rollbacksegment eingeplant werden. Dazu wird der *init.ora* Parameter TRANSACTIONS_PER_ROLLBACK_SEGMENT auf 4 gesetzt.

$$\text{Anzahl Rollbacksegmente} = \frac{\text{TRANSACTIONS}}{\text{TRANSACTIONS\_PER\_ROLLBACK\_SEGMENT}}$$

TRANSACTIONS = 1,1 * Prozesses
Prozesses = Anzahl concurrent Users + Anzahl Background-Prozesse +1

Beispiel:

15 User-Connects
4 Transaktionen pro Rollbacksegment
Anzahl Rollbacksegmente = ?

Der *init.ora* Parameter ProzessES stellt die Anzahl «CONNECTS» auf die Datenbank dar. Dabei dürfen die Background Prozesse nicht vergessen werden.

Prozesses = 15 + 7 + 1 = 23
TRANSACTIONS = 23 * 1,1 = 25
Anzahl Rollbacksegmente = 25 / 4 = 6

Zuordnung Transaktion <--> Rollbacksegment

Eine bestimmte Transaktion kann mit dem Kommando SET TRANSACTION USE ROLLBACKSEGMENT *rs_name* einem Rollbacksegment zugeordnet werden.

Größe von Rollbacksegmenten

Bei gemischten Transaktionen sind folgende Werte als Anhaltswerte zu verwenden. Besser mehrere kleinere Rollbacksegmente als wenig große. MAXEXTENTS muß auf das Betriebssystem Maximum gesetzt werden, um auch lange Transaktionen verarbeiten zu können.

Maximum number of extents exceeded !

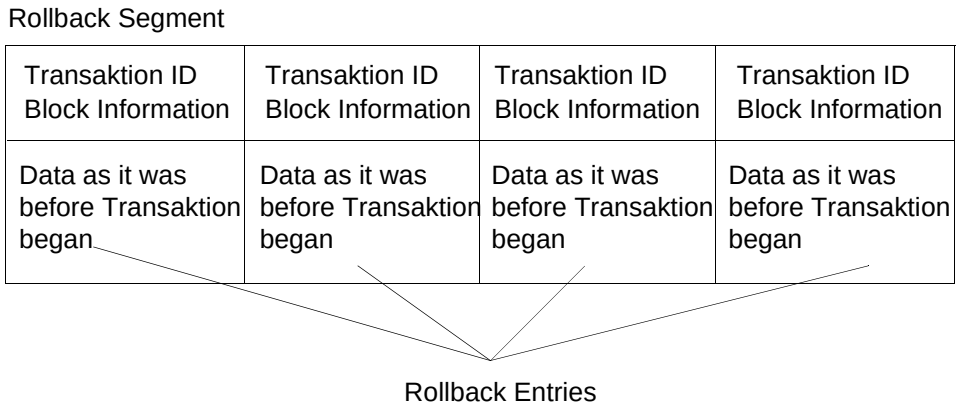
Bei sehr, sehr grossen Transaktionen kann es passieren, dass die Rollbacksegmente zu klein sind. Man versucht in diesem Fall ein Rollbacksegment mit MAXEXTENTS = mögliches Maximum anzulegen und / oder die Transaktion aufzusplitten.

OPTIMAL Size

Wächst ein Rollbacksegment infolge einer sehr grossen Transaktion einmal an, so werden viele Extents kreiert, die entsprechend auch viel Diskplatz benötigen. Wird beim Erstellen der Rollbacksegmente die Option OPTIMAL angegeben, so verkleinert sich das Rollbacksegment dynamisch wieder wenn die nächste Transaktion das Rollbacksegment berührt.

Inhalt der Rollbacksegmente

Rollbacksegmente enthalten die Information der «uncommited» Transaktionen, das heißt der Zustand der Daten vor Transaktionsbeginn. Oracle verteilt die aktiven Transaktionen auf die zur Verfügung stehenden Rollbacksegmente. Sind zuwenig Rollbacksegmente vorhanden so entstehen Waits. Zur Überprüfung kann die dynamische Table v$waitstat herangezogen werden. Der SQL *DBA Monitor kann zur Auswertung benutzt werden.



Neuerstellen von Rollbacksegmenten

Nach einem Datenbank Export/Import sollten die Rollbacksegmente gedropped und wieder erstellt werden, da diese beim Export/Import nicht optimiert werden. Rollbacksegmente mit mehr als 4-5 Extents gelten als fragmentiert. Rollbacksegment können dynamisch während dem normalen Betrieb (neu in Oracle7) gedropped und erstellt werden:

```
sqlplus system/...
alter rollback segment r01 offline;
Rollback segment altered
drop rollback segment r01;
Rollback segment dropped.
create rollback segment r01 tablespace rbs
  storage (
  initial 100k
  next 150k
  minextents 5
  maxextents 121
  optimal 800k);
Rollback segment created.

alter rollback segment r01 online;
```

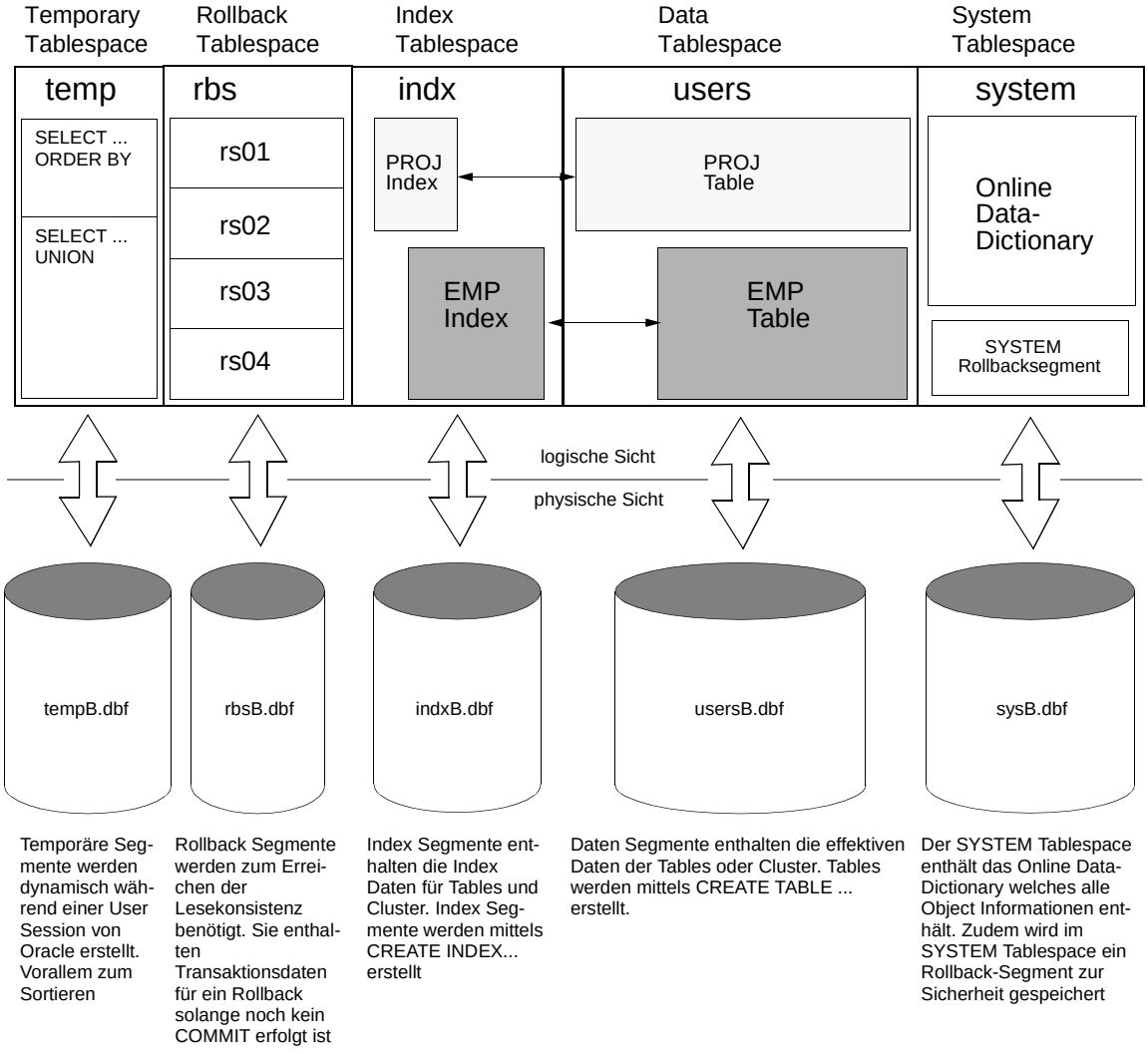
Wichtigste Punkte Rollbacksegmente

- Der Verlust einer Online Redolog Group bedeutet **immer einen Datenverlust**, mindestens bis zum letzten Checkpoint. Deshalb nicht allzu große Redologfiles wählen oder den Paratmeter LOG_CHECKPOINT_INTERVAL so setzen, dass nicht nur bei einem Log-Switch ein Checkpoint erfolgt. **Redologfiles auf einer separaten, schnellen Disk plazieren !**
- MAXEXTENTS bei Rollbacksegmenten auf das mögliche Maximum setzen (UNIX = 121) damit auch große Transaktionen verarbeitet werden können.
- Rollbacksegmente ab und zu droppen/neu kreieren, um Fragmentierung nicht zu groß werden zu lassen
- Bei großen Reports mit der Option SET TRANSACTION READ ONLY kann es passieren, dass zuwenig Platz in den Rollbacksegmenten zur Verfügung steht: *ORA 1555 snapshot too old (rollback segment too small).* In diesem Fall müssen die Rollbacksegmente vergrößert werden oder die Transaktion muss gesplittet werden (Mit MAXEXTENTS = UNLIMITED wird das kaum eintreten !).
- Rollbacksegmente beinhalten Daten, welche noch nicht «commited» sind. Bei Rollbacks werden diese Daten zurückappliziert. **Der Inhalt der Rollbacksegmente ist durch die Redologfiles geschützt.**
- Eine Transaktion ist genau einem Rollbacksegment zugeordnet. Ein Rollbacksegment kann mehrere Transaktionen gleichzeitig verarbeiten.
- Der große Vorteil der Rollbacksegmente ist, dass Oracle Tables nicht als ganzes locken muß. Dadurch können mehrere Benutzer simultan mit den gleichen Daten arbeiten ohne wesentliche Performance Verschlechterung.
- Ob genügend Rollbacksegmente vorhanden sind, kann nur wirklich während des produktiven Betriebes festgestellt werden. Dazu eignet sich der SVRMGRL Monitor. Man überprüft die Spalte *Header Waits/Sec*. Ist dieser Wert groß, so müssen weitere Rollbacksegmente hinzugefügt werden. Zur Auswertung wird die dynamische Table *v$waitstat* benutzt.

5. Tablespaces

Logische und physische Sicht der Tablespaces

Tablespaces sind logische Unterteilungen der Datenbank, damit Daten entsprechend ihrem Verwendungszweck gespeichert werden können. So werden Index-, Daten-, Rollback-, und temporäre Segmente immer in verschiedenen Tablespaces gespeichert. Im Idealfall entspricht ein Tablespace genau einem physikalischen Datenbankfile.



Grundsätzliches zu Tablespaces

- Der SYSTEM Tablespace wird beim Erstellen der Datenbank kreiert. Er muß zwingend vorhanden sein.
- Der SYSTEM Tablespace sowie der ROLLBACK Tablespace dürfen auf keinen Fall verloren gehen. Es gibt keine Möglichkeit die Datenbank ohne diese Tablespaces zu starten.
- Optimal besteht ein Tablespace aus einem Datenbank-File. Wenn nötig, können zu einem Tablespace weitere Datenbank-Files hinzugefügt werden.
- Tablespaces können offline gestellt werden. (Tablespace-Backup, Reorganisation, Hinzufügen von Datenbank-Files etc).
- Jedem Oracle Benutzer wird ein Default- und Temporary-Tablespace zugeteilt. **User-Daten nicht im SYSTEM Tablespace erstellen.**

Tablespace Organisation

User Daten

Dies ist der Default Tablespace für Benutzer-Tables. Jedem Benutzer wird dieser Tablespace zur Speicherung seiner Daten zugewiesen. Um eine optimale Performance zu erreichen sollten in diesem Tablespace keine Index-Segmente kreiert werden.

Indexe

Der Index Tablespace enthält die Indexe der User-Tables. Um eine optimale Performance zu erreichen sollten in diesem Tablespace keine Tables kreiert werden.

SYSTEM

Der System Tablespace muß immer verfügbar sein, da er das Online Data Dictionary mit den Locations sämtlicher Datenbank Objekte enthält. Geht der SYSTEM Tablespace verloren, so kann die Datenbank nicht mehr gestartet werden.

Rollback

Der Rollbacksegment Tablespace beinhaltet die privaten Rollback-Segmente. Rollbacksegmente gewährleisten Lesekonsistenz auf Transaktions- und Befehlsebene.

Temporäre Daten

Im Temporary Tablespace werden von Oracle intern Sortier Operationen vorgenommen (SELECT .... ORDER BY). Temporäre Segmente werden dynamisch kreiert und wieder gelöscht. Sie unterstehen nicht der Kontrolle durch die Userprozesse.

Erstellen von Tablespaces

Der SYSTEM Tablespace wird beim Erstellen der Datenbank erstellt. Ohne diesen Tablespace kann eine Datenbank nicht gestartet werden. Er darf auf keinen Fall verloren gehen !

```
create database "B"
    maxinstances 1
    maxlogfiles  6
    character set "WE8DEC"
    ...
    datafile  '/oracle7/kurs/systB.dbf' size 7M;
```

Zusätzliche Tablespaces können jederzeit angelegt werden.

```
create tablespace rbs datafile
        '/oracle7/kurs/rbsB.dbf' size 3M
default storage (
        initial          128k
        next             128k
        pctincrease        0
        minextents         2
);
```

Löschen von Tablespaces (incl Inhalt)

Der SYSTEM Tablespace darf nicht gelöscht werden. Die zusätzlichen Tablespaces inclusive Inhalt werden wie folgt gelöscht

```
sqlplus system/...
alter tablespace users offline;
drop tablespace data_ts including contents;
```

Datenbankfile mittels UNIX Kommando löschen

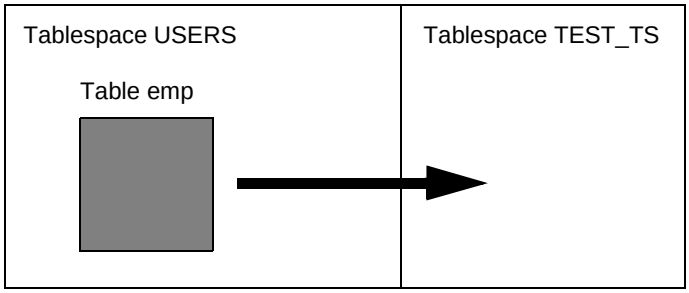
Offline/Online stellen der Tablespaces

Tablespaces können zeitweise für den normalen Zugriff gesperrt werden. Sind Daten in einem Tablespace defekt, so muß nicht die gesamte Datenbank außer Betrieb genommen werden. Ein Online Recovery des Tablespaces ist so möglich. Ein einmal offline gestellter Tablespace bleibt offline, auch wenn die Datenbank gestoppt und wieder gestartet wird. Der SYSTEM Tablespace muß immer online sein !

```
sqlplus system/...
alter tablespace users offline;
```

Verschieben einer Table in anderen Tablespace

Am einfachsten mittels CREATE TABLE AS SELECT * ..




```
create table emp1 tablespace test_ts as select * from emp;
drop table emp;
rename table emp1 to emp;
```

Beachte, dass bei dieser Methode die Constraints nicht mitgenommen werden (ausser den NOT NULL Constraints).

Starten der Datenbank ohne Daten-Tablespace

Ist ein Datenfile eines Tablespaces korrupt oder irrtümlich gelöscht worden, so kann die Datenbank ohne diesen Tablespace gestartet werden. **Das Starten der Datenbank ohne SYSTEM oder ROLLBACK Tablespace ist jedoch nicht möglich**. Diese beiden Tablespaces dürfen nicht verloren gehen !

Voraussetzungen :

- SYSTEM- und ROLLBACK-Tablespace sind in Ordnung
- Redolog-Files und Control-Files sind in Ordnung

```
sqlplus system/...
alter tablespace users offline;
Tablespace altered.
```

Datenbank stoppen und wieder starten

Die Datenbank ist nun gestartet, ohne den Tablespace *Users*. Selbstverständlich können auf keine Daten im Tablespace mehr zugegriffen werden:

```
sqlplus scott/tiger
select * from emp;
ERROR at line 1:
ORA-00942: table or view does not exist
```

6. Segmente

Segmente sind nach den Tablespaces die nächst **kleinere logische** Einheit, welche durch den Oracle Benutzer direkt manipuliert werden kann. Segmente sind Oracle Objekte, denen Daten zugrunde liegen. Zum Unterschied zu den Tablespaces können Segmente nicht offline/online gestellt werden. Daten-Segmente (Tables) können im Gegensatz zu Tablespaces exportiert/importiert werden. Segmente bestehen aus den physikalischen Größen Extents und Blocks.

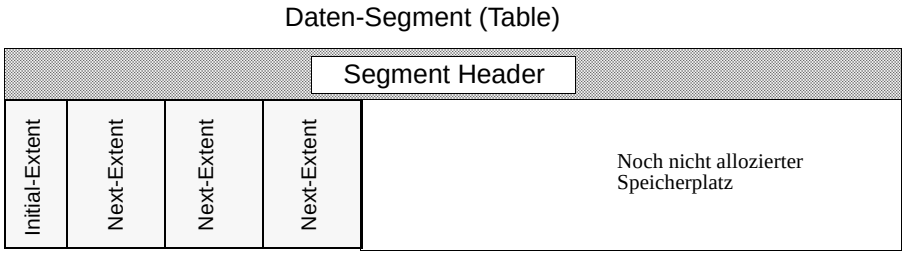
Segment Arten

Es gibt verschiedene Segmente, deren Aufbau sehr unterschiedlich ist:

- Daten-Segment = Tables
- Index-Segment = Indexe
- Rollback-Segmente (Ring of Extents)
- Temporäre-Segmente (Sortier Operationen)
- Bootstrap-Segment (Oracle intern)

Aufbau der Segment

Der Aufbau der Segmente ist für alle gleich. Sie bestehen aus «Extents», welche wiederum in Blocks unterteilt sind. Oracle alloziert bei Bedarf immer ein zusätzliches Extent zu einer bestimmten Anzahl Oracle Blocks. Die Größe und Organisation der Extents wird beim Kreieren des Segments mittels «Storage Parameter» definiert.



Storage Parameter welche die Extents direkt betreffen:

INITIAL: Größe des ersten Segments in Bytes. Default = 5 Oracle Block

NEXT: Größe des nächsten Extents in Bytes. Default = 5 Oracle Blocks

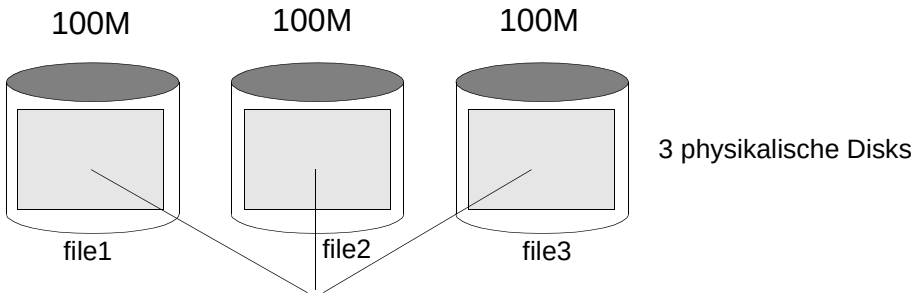
MAXEXTENTS: Maximale Anzahl Extents die für das Segment möglich sind. Default ist abhängig von OS und OS-Blocksize.

MINEXTENTS: Anzahl Extents wenn das Segment kreiert wird. Default = 1. Bei Rollbacksegmenten muß MINEXTENTS = 2 gesetzt werden. Die beste Performance erreicht man, wenn das gesamte Segment in ein Extent paßt. Besteht ein Segment nach vielen INSERTS, UPDATES aus vielen Extents so muß reorganisiert werden. Dazu werden die Daten mit der Option COMPRESS=y exportiert, die Datenbank neu erstellt und wieder importiert. Beim Importieren werden die Segmente jeweils in ein Extent gepackt. Bei sehr großen Tables welche auf mehrere Disks verteilt werden müssen, kann MINEXTENTS erhöht werden (Table Stripping).

PCTINCREASE: Prozent um wieviel das neue Extent gegenüber dem letzten wachsen soll. PCTINCREASE = 50 heißt also 1,5 mal größer als das letzte Extent. Bei PCTINCREASE = 0 sind alle Extents gleich groß.

Table-Stripping

Eine sehr große Table hat nicht Platz auf einer Disk, muß also auf mehrere Disks verteilt werden. Pro Disk wird genau ein Extent angelegt. Dies erreicht man durch das Setzen von MINEXTENTS.

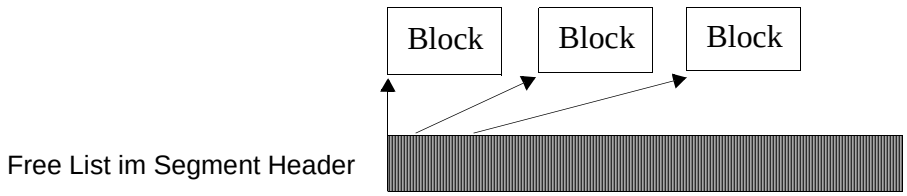


3 Extents auf 1 Table (300M) verteilt

```
create tablespace verybig datafile 'file1' size 100M;
alter tablespace verybig add datafile 'file2' size 100M;
alter tablespace verybig add datafile 'file3' size 100M;

create table bigtab (
....
....
)
tablespace verybig
storage (
          initial 100M      (1. Extent auf erster Disk)
          next 100M         (2. Extent auf zweiter Disk)
          minextents 3      (mindestens 3 Extents erstellen beim kreieren)
          pctincrease 0     (auch 3. Extent soll 100M groß sein)
)
```

Segment Header

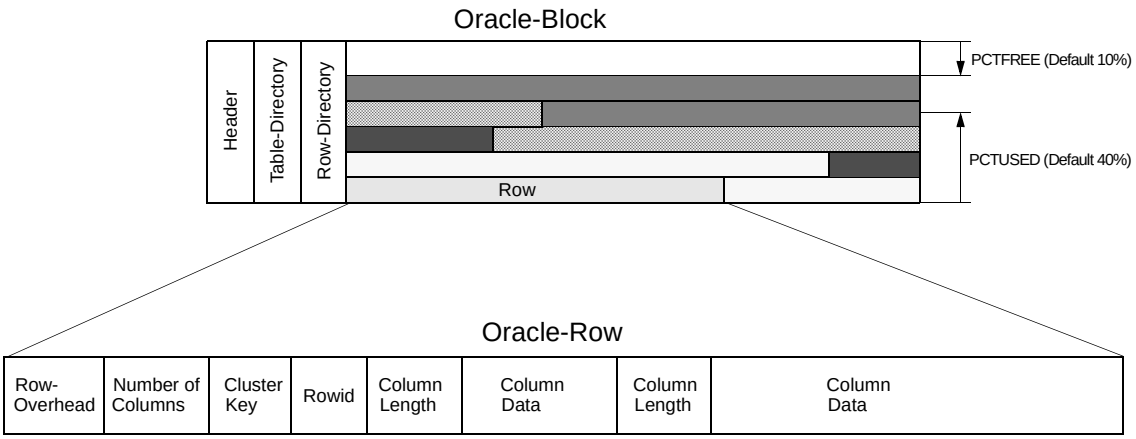
Jedes Segment unterhält im Segment Header eine Liste der freien Blöcke. Beim Hinzufügen eines Extents werden alle Blocks dieser Liste zugefügt. Die Liste wird von Oracle verwaltet und ist nicht einsehbar. Bei INSERT's von Records muß diese Liste gelesen und modifiziert werden. Bei Tables mit sehr vielen Insert's kann dadurch ein Engpaß entstehen, was durch die dynamische Table *V$WAITSTAT* mit einem hohen Wert für *buffer busy waits* angezeigt wird. Man kann in diesem Fall den *init.ora* Wert FREE_LIST_PROC erhöhen. (Anzahl Free Lists per Instance).



6.1 Datensegmente

Datensegment (Table)

Der Aufbau der Tables, Indexe und Cluster ist praktisch identisch. Am Beispiel Table soll der dies erklärt werden. Im Zusammenhang mit diesen Objekten ist darauf zu achten, dass ein Record (Row) einer Table in einen Block paßt. Ist dies nicht der Fall, so spricht man von «Block Chaining» was die Performance verschlechtert.



Aufbau Block

Eine Table besteht aus der logischen Größe «Segment», welche der logischen Einheit «Tablespace» zugeordnet ist. Logische Größen können durch den Oracle Benutzer direkt bearbeitet werden (create, alter drop). Die physikalischen Größen Extents, Blocks und Rows werden von Oracle intern verwaltet. Eine Table (Daten Segment) besteht aus einer Anzahl Extents (Optimal = 1 Extent). Oracle alloziert bei Bedarf immer ein weiteres Extent Speicherplatz. Extents sind in gleich große Blocks zu 2 oder 4K eingeteilt. Der Aufbau eines Oracle Blocks ist für Tables, Indexe und Cluster gleich. Ein Table-Record ist schließlich in einer Row gespeichert.

Header:

Der Header enthält generelle Informationen des Blocks wie Block Adresse, Segment Typ (Data, Index, Cluster oder Rollback).

Table Directory:

Informationen zu «clustered tables». Nur Cluster enthält ein Table Directory.

Row-Directory:

Informationen zu den aktuellen Rows im Block wie beispielsweise die Adresse jedes einzelnen Row Teils.

Free-Space:

Freier Platz für UPDATES und INSERTS. Mittels PCTFREE kann definiert werden wieviel ein Block gefüllt wird bevor ein neuer Block alloziert wird. Der Default Wert von PCTFREE ist 10, das heißt ein Block wird zu maximal 90 % gefüllt.

Wahl des geeigneten PCTFREE

Ändern sich die Daten im normalen Umfang durch INSERT's und UPDATE's so kann der Defaultwert von 10 für Tables als optimal angenommen werden. Bei Indexes kann ein tieferer Wert gewählt werden, da hier kaum Platz für spätere UPDATE's benötigt wird.
Ein PCTFREE von 0 bei Index-Segmenten ist angebracht.

Folgen eines kleinen PCTFREE:

- Wenig Platz für spätere Updates
- Blocks werden von Oracle gut ausgenutzt.
- Platzsparend
- Performance wird schlechter, wenn «Block Chaining» eintritt

Wahl des geeigneten PCTUSED

Wird der Wert von PCTFREE in einem Block einmal überschritten, so fügt Oracle keine neuen Rows in diesen Block ein, bis der Füllungsgrad wieder unterhalb PCTUSED fällt.. Oracle versucht jedoch den Block mindestens auf der Schwelle von PCTUSED gefüllt zu halten. Defaultwert von PCTUSED = 40%

Regeln für PCTFREE und PCTUSED

- PCTUSED + PCTFREE <= 100%
- Ist PCTUSED + PCTFREE = 100 versucht Oracle den Block möglichst auf dem Maximum gefüllt zu halten.
- Je kleiner die Differenz zwischen PCTFREE und PCTUSED ist, desto besser wird der Block ausgenutzt auf Kosten der Performance.

Angabe der Storage Parameter

Storage Parameter können auf zwei Stufen definiert werden. Tiefer liegende Angaben überschreiben höher gelegene. Nicht alle Parameter können überall definiert werden.

```
create database ...
Keine Angaben möglich !
```

Stufe Tablespace («Schablone»)

```
create tablespace
default storage (
        inital
        next
        minextents
        maxextents
        pctincrease
)
```

Stufe Segment

```
create cluster
pctused
pctfree
initrans
maxtrans
storage (
        inital
        next
        minextents
        maxextents
        pctincrease
)
```

```
create index
initrans
maxtrans
storage (
        inital
        next
        minextents
        maxextents
        pctincrease
)
pctfree
```

```
create table
pctused
pctfree
initrans
maxtrans
storage (
        inital
        next
        minextents
        maxextents
        pctincrease
)
```

```
create  rollback segment
storage (
        inital
        next
        minextents
        maxextents
)
```

Beachte:

> Die Angabe der Storage Parameter sollte auf der Stufe «Segment» unbedingt angegeben werden. Man sollte sich nicht einfach mit den Default Werten zufrieden geben, sondern überlegen welche Charakteristik die Daten haben. Insbesondere bei Tables ist das sehr wichtig.

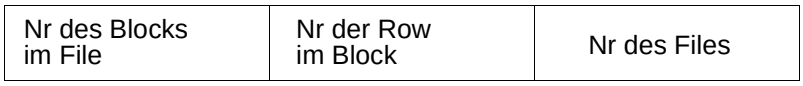
Aufbau Oracle Row

Eine Oracle Row ist die kleinste physikalische Speichereinheit. Sie besteht im optimalen Fall aus «einem Stück», ist also nicht auf mehrere Blocks verteilt. Oracle versucht wenn immer möglich Rows als Ganzes zu speichern und nicht zu splitten. Jede Row beginnt mit dem *Row Header*. Von jeder Column wird auch deren Länge in der Row festgehalten. Primary Keys sollten beim create Statement zuerst angegeben werden. LONG-Daten ganz am Schluß.

«Block-Chaining» und dessen Nachweis

Ist eine Row auf mehrere Blocks verteilt, so spricht man von «Block chaining». Dies ist ein unerwünschter Zustand, da die Performance dadurch beeinträchtigt wird. Tritt dies oft auf, so muß PCTFREE verändert werden. Nach einem Export/Import verschwindet das unerwünschte «Block chaining». Zum Nachweis dient der Pseudo Datentyp ROWID. Jede Row besitzt eine eindeutige physikalische Adresse, welche durch ROWID repräsentiert wird. ROWID ist aus drei Teilen aufgebaut:

Aufbau der **Oracle-7** ROWID




```
select rowid, ename from emp;

  ROWID              ENAME
         ------------------ ----------
         000005DF.0000.0004 SMITH
         000005DF.0001.0004 ALLEN
         000005DF.0002.0004 WARD
         000005DF.0003.0004 JONES
         000005DF.0004.0004 MARTIN
         000005DF.0005.0004 BLAKE
         000005DF.0006.0004 CLARK
         000005DF.0007.0004 SCOTT
         000005DF.0008.0004 KING
         000005DF.0009.0004 TURNER
         000005DF.000A.0004 ADAMS
```

Alle Row sind im gleichen File
Nummer der Row im Block
Alle Rows sind im gleichen Block, kein «Block Chaining»

Block Statistik mit Column ROWID

Um dies zu demonstrieren wird die sehr grosse SQL*Plus *help* Tabelle benutzt. Sie beinhaltet ein HELP zu SQL Kommandos in SQL*Plus.

Anzahl Blocks einer Table die Daten enthalten:

```
select count(distinct(substr(rowid,1,8))) Anz_Blocks from help;

ANZ_BLOCKS
----------
       260
```

Anzahl Rows pro Block:

```
select substr(rowid,1,8)Block_Nr, count(*) Anzahl_Rows
from help
group by substr(rowid,1,8);

BLOCK_NR ANZAHL_ROWS
-------- -----------
000005DF          14
```

Die Verteilung der Anzahl Rows pro Block sollte möglichst gleichmäßig sein, dann ist keine Reorganisation nötig:

```
select avg(count(*)) Avg, max(count(*)) Max, min(count(*)) Min
from help
group by substr(rowid,1,8);

 AVG        MAX        MIN
---- ---------- ----------
31.4         48          5
```

Der Block mit der maximalem Anzahl Rows (48) steht im Gegensatz zum Block mit der kleinsten Anzahl Rows (5), im Mittel besitzen die Blocks 31 Rows.

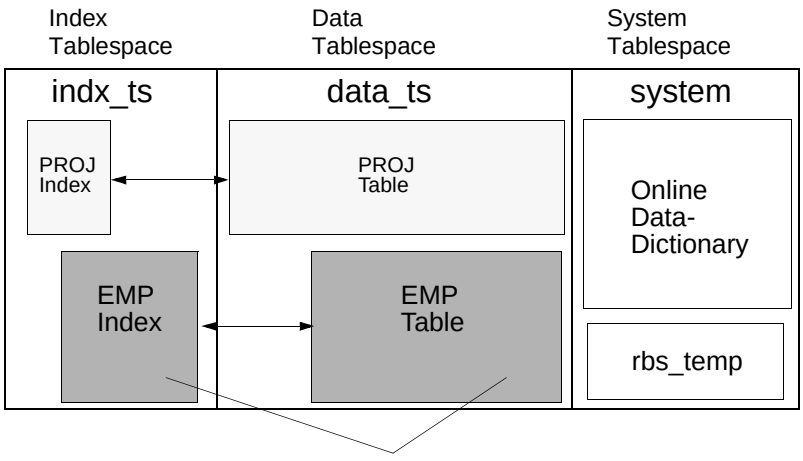
6.2 Indexsegmente

Indexsegment (Index)

Index-Segmente sind optionale, zusätzliche Strukturen zu Tables und Cluster. Sie verbessern die Ausführungszeit eines Querys erheblich, da keine Fulltable Scans mehr durchgeführt werden müssen. Zudem garantieren sie mit der Option UNIQUE, dass garantiert jede Row in einer Table eindeutig ist, also nur einmal vorkommt. Der Primary Key einer Table ist immer indexiert. Indexe können aus einer oder auch mehreren Columns (Concatenated Index) gebildet werden. Die Verwendung von Indexen ist nicht zwingend nötig, aber dringend empfohlen, vorallem bei Joins. Indexe sind physikalisch unabhängig von der zugrunde liegenden Tabelle. Sie können jederzeit gedropped und wieder neu erstellt werden, ohne dass die Daten dabei verändert werden. Wird ein Index gedropped, so läuft die Applikation wie bisher, nur langsamer. Indexe unterstehen der Kontrolle von Oracle, das heißt: Ändern sich die Daten der Table durch INSERT's, UPDATE's, DELETE's so werden die zugehörigen Index-Strukturen vom RDBMS automatisch mitgeführt. Sind auf einer Table sehr viele Indexe vorhanden, so kann durch die Änderungen an den Daten eine gewisse Performance Verschlechterung eintreten, da die Index-Strukturen simultan mitkorrigiert werden müssen. Bei Tables mit sehr vielen Veränderungen, müssen die Indexe gezielt auf die wichtigsten Columns gelegt werden. Jeder Index belegt ein eigenes Segment, kann also über Storage Parameter verfügen. Indexe sollten wenn möglich immer auf bereits bestehende Daten angelegt werden, häufig modifizierte Indexe sollten periodisch gedropped und neu erstellt werden. Dazu eignet sich die Möglichkeit des Exports/Imports mit der Option INDEXFILE=y.

Anlegen der Index-Strukturen

Indexe sollten aus Performance Überlegungen nicht im gleichen Tablespace wie die Daten selbst angelegt werden. Dies hat zwar den Nachteil, dass beim Offline stellen des Index Tablespace der Zugriff auf die Daten langsamer wird.



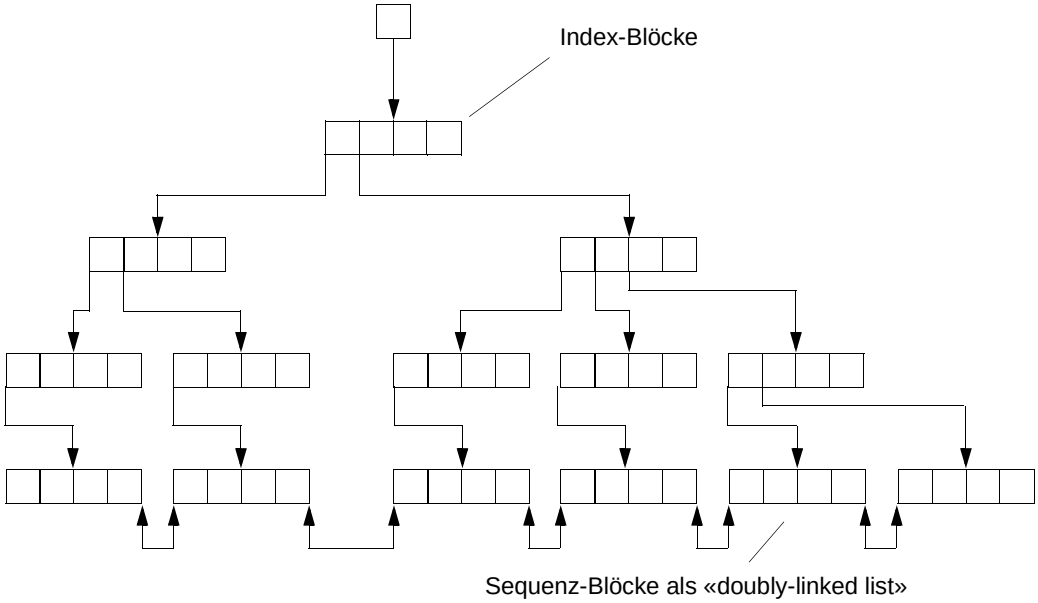
Daten und Indexe immer in unterschiedlichen Tablespaces erstellen

Indexe als B-Trees

Index Strukturen werden in Oracle aus sogenannten B-Trees aufgebaut. B-Trees können im Gegensatz zu normalen Binär-Bäumen optimal auf einem sekundären Medium (zB Harddisk) gespeichert werden und von dort in den Hauptspeicher geladen werden. Binär-Bäume weisen gleich viele Knoten wie Keys auf, im Gegensatz zu den B-Trees, wo eigentliche Informationsblöcke mit mehreren Keys zu Knoten zusammengefaßt werden.

Struktur von B-Trees

B-Trees bestehen aus Index- und Sequenzblöcken. In den Sequenzblökken werden die ROWID's der Records gespeichert:



Index-Blöcke: Baumstrukturierter Index-Satz zu den Sequenzblöcken
Sequenz-Blöcke: «Leafs», enthalten die ROWID's

Unique und Non-unique Indexes

Wird eine Column mit einem Unique Index versehen, dann muß jeder Wert der Column eindeutig sein. Dies trifft bei Primary Keys zu, auf diese wird immer ein Unique Index angelegt, was unter Oracle7 automatisch erfolgt. Doch auch nicht eindeutige Columns können indexiert werden, indem beim Kreieren des Index das Schlüsselwort «unique» angegeben wird. Das Anlegen von Indexes auf non-unique Columns ist jedoch nur dann sinnvoll, wenn der Wertebereich groß ist. Eine Column, die z.B. nur die Werte «männlich», «weiblich» beinhalten kann, ist denkbar ungeeignet zum indexieren !

Ideale Columns zum Indexieren:

- Jeder Wert ist eindeutig (unique)
- Großer Wertebereich
- Columns, die viele Nulls haben und wenig Werte, dabei oft Abfragen auf diese Werte vorkommen. Das Statement *select ... where x > 0* ist dem Statement s*elect ... where x is not null* vorzuziehen, da dieses den Index auf *x* benutzt.

Concatenated Index

Aus mehreren Columns bestehende Indexe werden «Concatenated Index» genannt. Sie sind sehr ideal, wenn Eindeutigkeit über mehrere Columns gefordert ist.

Vorwahl	Telefon	PLZ	Ort	Straße
033	45 29 75	3628	Uttigen	Auweg 31
031	61 67 76	3000	Bern	Ralligweg

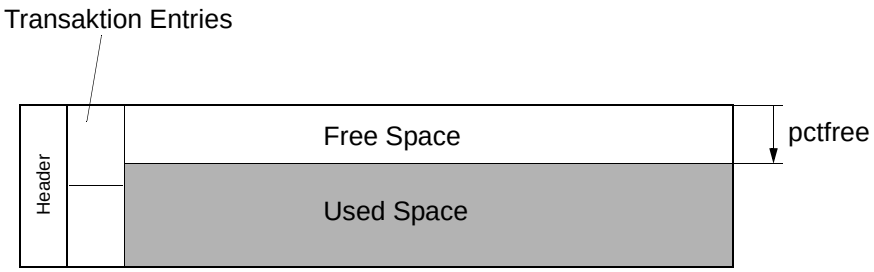
Die Columns «Tel.-Vorwahl» und «Telefon» sind zusammen unique und deshalb für einen concatenated Index geeignet.

Storage Parameter von Indexes

Ist der Index unique, kann PCTFREE = 0 gewählt werden, da Updates an Unique Keys nicht vorkommen.

```
create unique index i_emp on emp (ename)
initrans 4
maxtrans 255
tablespace indx_ts
storage (
   initial 100K
   next 100K
   pctincrease 0
)
pctfree 0;
```

Der Parameter *initrans* definiert die Anzahl «Transaktion Entries», die in jedem Block reserviert werden. Das «Transaktion Entry» definiert wieviele Transaktionen gleichzeitig den Block verändern können. Oracle reserviert für jedes Entry 23 Bytes im Free Space des Blocks. Bei Indexes ist *initrans* per Default = 2.



7. Objekte

Im allgemeinen zählt man auch die Datenbank Segmente zu den Datenbank Objekten, diesen liegen im Gegensatz zu den im folgenden Abschnitt erläuterten Strukturen jedoch Daten zu Grunde.

Arten von Objekten

Oracle kennt folgende Objekte, denen keine Daten zu Grunde liegen:

- Sequences
- Views
- Synoyme
- Constraints

Sequences

Jede Tabelle muss über einen eindeutigen Key (UNIQUE KEY) verfügen um einen Datensatz zu referenzieren. Eignet sich kein Feld aus der realen Welt für einen Key, so verwendet man meist eine Sequence um einen solchen «versteckten» Key zu generieren. Insbesondere im Multiuser Betrieb, wenn viele Benutzer in einer Tabelle Datensätze unabhängig voneinander einfügen, löschen oder ändern bilden Sequences ideale Datenstrukturen, da dadurch keine Wartezeiten entstehen.

Wichtigste Eigenschaften von Sequences:

- Unabhängig von einer Tabelle, eigenständiges Objekt
- Auf Sequences können nur SELECT Statements angewendet werden.
- Sequences können im Memory gehalten werden (cached) um sehr rasch einen Key zu generieren.
- Sequences können «Löcher» (Gaps) aufweisen, insbesondere bei sehr vielen Transaktionen.
- Sequences können von mehreren Tabellen benutzt werden.
- Sequences können nicht benutzt werden in Subqueries, Query in einer View, SELECT DISTINCT, SELECT mit GROUP BY und in der WHERE Clause.
- Sequences werden beim Export / Import automatisch mit der Tabelle mitgenommen.

Kreieren einer Sequence

```
CREATE SEQUENCE my_seq_adresse
 INCREMENT BY 1
 START WITH 1
 MINVALUE 1
 NOMAXVALUE
 NOCYCLE
 NOCACHE
 ORDER;
```

Generieren einer Sequence Nummer

```
INSERT INTO adresse (adr_nr, adresse)
VALUES(my_seq_adresse.NEXTVAL,'Hans Müller');
```

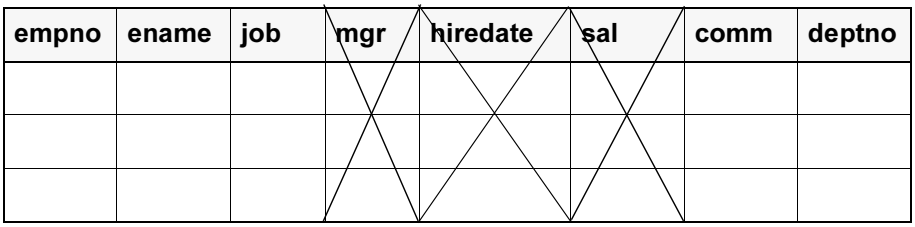
Abfragen der aktuellen Sequence Nummer

```
SELECT my_seq_adresse.CURRVAL from dual;
```

Views

Views bilden **äußerst hilfsreiche Datenstrukturen** bei der Applikationsentwicklung. Views dienen vorallem dazu, Komplexizität zu verbergen, das heißt Joins über viele Tabellen werden unsichtbar. Ein weiterer sehr großer Vorteil von Views ist, dass sie auf dem Server installiert werden und damit komplexe Abfragen über das Netz beschleunigen.

Ausblenden von Columns

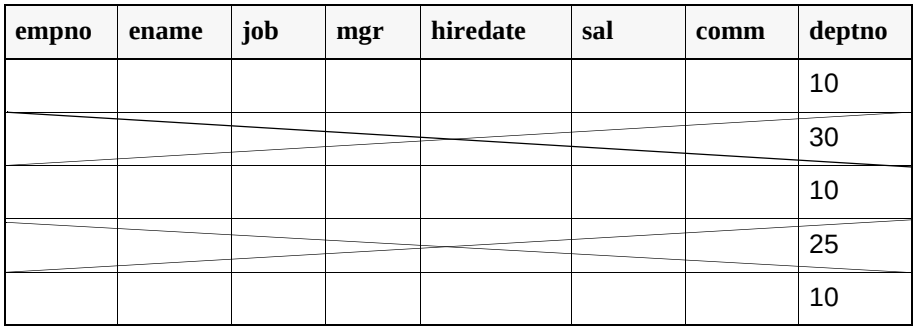



```
create view emp_view
as select empno, ename, job, comm, deptno from emp;

select * from emp_view;

EMPNO     ENAME        JOB          COMM       DEPTNO
----------------     --------     -------    ----------
7369      SMITH        CLERK                   20
7499      ALLEN        SALESMAN     300        30
7521      WARD         SALESMAN     500        30
7566      JONES        MANAGER                 20
```

Ausblenden von Rows

Mittels Views und einer WHERE Bedingung können Rows ausgeblendet werden:

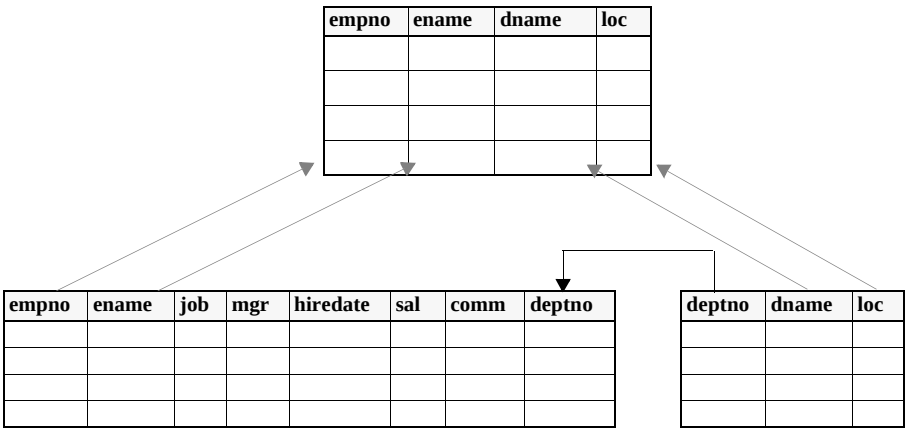
:




```
create view emp_view
as select empno, ename, job, comm, deptno from emp
where deptno = 20;

select * from emp_view;


EMPNO     ENAME        JOB          COMM       DEPTNO
---------- ---------- ---------- ---------- ------------
7369      SMITH        CLERK                   20
7566      JONES        MANAGER                 20
```

Verbergen von Komplexizität

Damit können Joins «verborgen» werden, die Anwendungsentwicklung wird wesentlich einfacher.



Ohne die Verwendung einer View ....

```
select a.empno, a.ename, b.dname, b.loc
from emp a, dept b
where a.deptno = b.deptno
```

.... verwendet man mit einer View

```
create view my_view as
select a.empno, a.ename, b.dname, b.loc
from emp a, dept b
where a.deptno = b.deptno

select * from my_view;
```

Ausblenden von Records, Columns in einer View

Projektionen und Selektionen können nun auch auf Views angewendet werden, das komplexe vollumfängliche SELECT Statement wird durch Oracle intern generiert.

```
select ename,dname
from my_view
where dname = 'SALES';
```

Einschränkungen auf Views

- Wenn das Query einer Views aus Joins, SET, DISTINCT oder GROUP_BY besteht, so können keine Rows mittels INSERT, UPDATE oder DELETE manipuliert werden.
- Enthält das Query die mächtige Expression DECODE, so können ebenfalls keine Rows mittels INSERT, UPDATE oder DELETE manipuliert werden.

Synonyme

Jedes Objekt in einer Oracle Datenbank hat einen Besitzer. Auf diese Weise können sehr einfach gesamte Projekte von Datenbank zu Datenbank gezügelt werden, indem der Applikationsowner exportiert und wieder importiert werden. Um ein Objekt Datenbankweit eindeutig zu kennzeichnen und allen anderen Benutzern zugänglich zu machen werden Synonyme verwendet. Public Synonyme sind in der Datenbank eindeutig durch das RDBMS kontrolliert vorhanden, dies im Gegensatz zu einem normalen Objekt, das durchaus unter dem gleichen Namen mehrfach vorkommen kann:

Tabelle *projekt* des Benutzer scott:	scott.project
Tabelle *projekt* des Benutzers watson:	watson.project

Kreieren eines Synonyms

Um nun die Tabelle projekt allen Benutzern eindeutig zugänglich zu machen verwendet man ein Public Synonym.

```
create public synonym emp for scott.emp;
```

Damit können alle Benutzer die Tabelle *emp* referenzieren.

8. Oracle Prozesstruktur

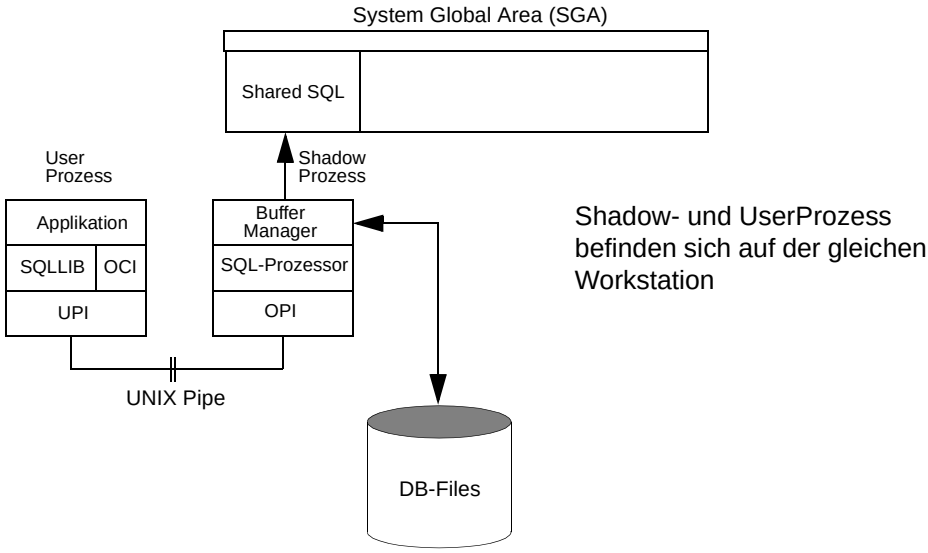
8.1 Architektur

Prozess-Typen

Beim Datenbankstartup kreiert Oracle automatisch mehrere Prozesse. Aus Sicherheitsgründen sind diese eingeteilt in Userprozesse und Oracle Prozesse. Ein Userprozess (z.B. sqlplus) hat nie direkten Zugriff auf die Datenbank, sondern kommuniziert mit einem «Shadowprozess», welcher den Zugriff auf die Datenbank vornimmt. Bei UNIX Systemen sind Userprozess und Shadowprozess immer zwei reelle Prozesse (Two-Task Variante), bei anderen Betriebssystemen kann der User- und Shadowprozess auch in einem reellen Prozess vereint sein (Single-Task Variante). Den Hintergrundprozessen sind klar definierte Aufgaben zugeteilt. In einem Client-Server Umfeld kommt zusätzlich noch ein Netzwerkserverprozess hinzu der das Dispatching der Client-Requests übernimmt. Oracle7 unterstützt zusammen mit SQL*Net 2 auch das des Multithreaded Server Konzepts zur Unterstützung von sehr vielen Requests über das Netzwerk.

Two-Task Architektur

Ein Userprozess (zB SQL*Plus) hat nie direkten Zugriff auf die Datenbank. Er kommuniziert über eine definierte Schnittstelle mit seinem entsprechenden Shadowprozess. Sind Userprozess und Shadowprozess physikalisch auf der gleichen UNIX Workstation, so kommunizieren die beiden Prozesse via Pipe.



Shadowprozess

Der Shadowprozess ist in drei Teile eingeteilt:

- Buffer-Manager: Komponente, welche von der Datenbank liest und in die SGA schreibt.
- SQL-Prozessor: Dies ist der Teil, der die SQL Kommandos auflöst (Parser, Optimizer) sowie die Kernel-Zugriffs Mechanismen und Kernel Routinen beinhaltet.
- OPI: Oracle Prozess Interface, Schnittstelle zu UNIX Betriebssystem um via Pipe mit dem Userprozess kommunizieren zu können.

Userprozess

Der Userprozess besteht im wesentlichen aus vier Komponenten:

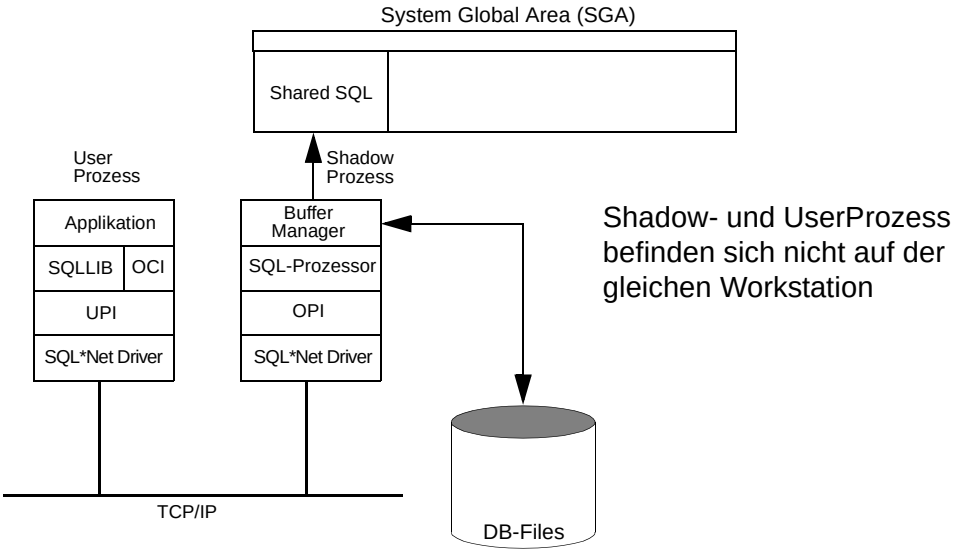
- Applikation: Eigentliche Anwendung wie SQLPLUS, Forms usw.
- SQLLIB: Precompiler Schnittstelle (z.B. Pro*C). Library mit vordefinierten Calls um auf Datenbank zugreifen zu können. (High-Level Interface).
- OCI: Oracle Call Interface, Low-Level Interface. Wird vom «normalen User» kaum verwendet.
- UPI: Zu OPI entsprechende Schnittstelle

Single-Task Architektur

Mit der Einführung von Oracle7 auf DOS/Windows und MAC Systemen, hat diese Implementation an Bedeutung gewonnen. Da diese Betriebssysteme keine Prozesskommunikation unterstützen sind User- und Shadowprozess in einem Task vereint.

SQL*Net Implementation

SQL*Net ist eine Erweiterung zur lokalen Two Task Implementation. Die Interprozesskommunikation zwischen dem Userprozess und dem Shadowprozess ist dadurch erweitert, dass OPI und UPI auf verschiedenen Rechnern sind. SQL*Net verwendet zur Interprozess Kommunikation bei UNIX das Berkeley Socket TCP/IP Interface.

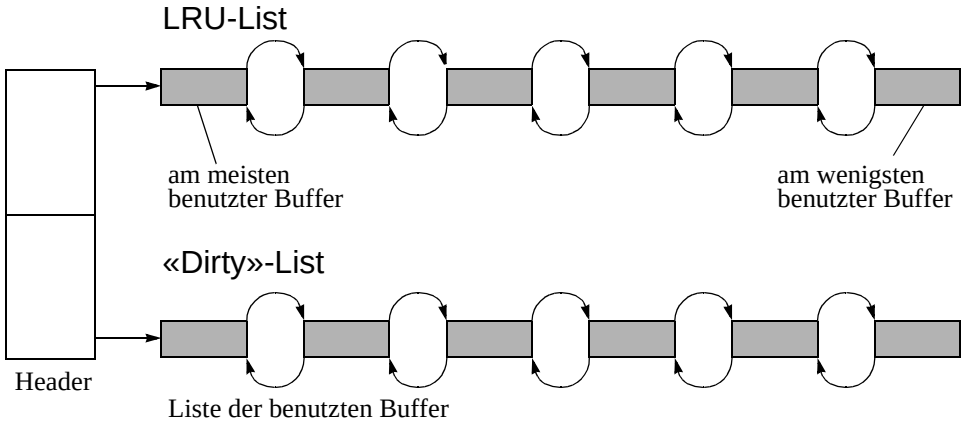


8.2 Hintergrundprozesse

Die nötigen Hintergrundprozesse werden beim Startup der Datenbank automatisch gestartet (startup nomount). Sie sind keinem «Controlling Terminal» mehr zugeordnet, das heißt Fehlermeldungen dieser Prozesse erscheinen nicht auf der Console. Sie protokollieren in die Oracle Log-Files: `?/rdbms/log/xxx.trc` und `?/rdbms/log/alert.log`.
Diese Files sind bei Datenbank-Problemen zu konsultieren. Von Zeit zu Zeit sollten die Logfiles im Directory `?/rdbms/log` gelöscht werden.

dbwr

Der Database Writer *dbwr* schreibt alle modifizierten Daten-, Index- und Rollbacksegment Buffer aus der SGA in die Datenbankfiles. Modifizierte Buffer werden nach dem LRU («Least recently used») Algorithmus verzögert (Deferred Write) auf die Disk geschrieben. Diejenigen Buffer welche sehr oft in der SGA benötigt werden, behält der RDBMS-Kernel in der SGA um einen schnellen Zugriff zu ermöglichen. Die Buffer, auf welche kein Zugriff mehr erfolgt, werden geschrieben. Wären also nicht auch andere Mechanismen vorhanden, so würden die viel benutzten Buffer gar nie geschrieben. Dies darf aber nicht der Fall sein, deshalb ist der LRU Algorithmus nicht das einzige Kriterium zum Schreiben der Buffer.



Oracle trägt die geänderten Buffer («dirty Buffer») in der «Dirty-List» ein. Diese Buffer müssen vom *dbwr* in die Datenbankfiles geschrieben werden, wenn die Bedingungen zum Schreiben erreicht werden. In der «LRU-List» sind die Buffer entsprechend ihrer Anzahl Zugriffe geordnet. Wird ein Buffer benutzt, so wird er aus der «LRU-List» entfernt und in die «Dirty-List» eingetragen. Beide Listen sind doppelt verkettet. Die Anzahl Database Writer kann mit dem *init.ora* Parameter DB_WRITERS definiert werden. Sind die Tablespaces auf mehrere Disks verteilt, sollte man gleich viele *dbwr's* definieren wie aktive Tablespace Disk's vorhanden sind.

lgwr

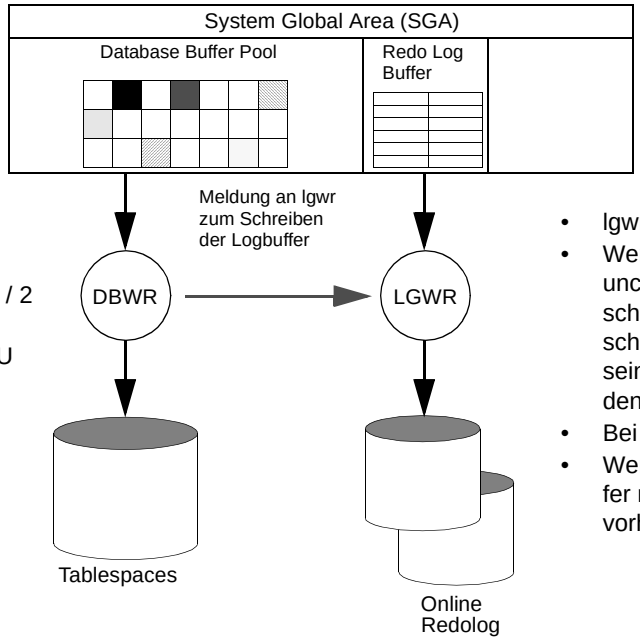
Der Log Writer *lgwr* schreibt die Logbuffer in das aktive Redolog File. Ein «gespiegeltes» Schreiben auf zwei aktive Redolog Files ist in der Version 7 möglich geworden. Verliert man alle aktiven Redolog Files, so verliert man in jedem Fall getätigte Transaktionen. Wie auch der Database Writer schreibt der Log Writer die Redo-Entries mehrerer Transaktionen gleichzeitig in das Redolog File («Piggybacking»). Bei sehr vielen Transaktionen welche «commited» werden, kann der Log Writer rasch zum aktivsten Prozess im System werden. Bei gespiegelten Redologfiles schreibt der *lgwr* parallel auf alle aktiven Redologfiles gleichzeitig.

Deferred Write

Um eine gute Performance zu erreichen, werden die Datenbuffer so lange wie möglich in der SGA behalten. Um das Risiko der Daten Inskonsitenz zu vermindern müssen die modifizierten Buffer von Zeit zu Zeit in die Datenbankfiles geschrieben werden.

dbwr schreibt wenn:

- Bei jedem Checkpoint
- wenn DB_BLOCK_WRITE_BATCH / 2 in der Dirty-List erreicht wird
- Wenn keine leeren Buffer in der LRU mehr vorhanden sind



- lgwr schreibt wenn:
- Wenn der *dbwr* einen uncomitted Buffer schreiben muß, schreibt auch der *lgwr* seinen entsprechenden Redobuffer.
- Bei jedem Commit
- Wenn keine freien Buffer mehr in der SGA vorhanden sind.

Beachte:

> Schreibt der Datenbase Writer *dbwr* einen «uncommited» Buffer in das Datenbankfile, so unterrichtet er den Log Writer *lgwr* auch seinen entsprechenden Redobuffer zu schreiben, selbst wenn noch kein Commit erfolgt ist. **Es darf also nie ein Datenbuffer in der Datenbank sein, wenn nicht auch der entsprechende Redobuffer im Redolog File vorhanden ist !** Das Redolog File gewährt in jedem Fall die Konsistenz der Daten-, Index- und Rollbacksegment Buffer

pmon

Der Prozess Monitor *pmon* führt die Rollbacks von «abgestürzten» und abgebrochenenTransaktionen durch. Er liest also die Rollback-Segmente. Auch das Freigeben von Locks nach einem Prozess Recovery ist die Aufgabe des *pmon.*

smon

Der System Monitor ist verantwortlich für das Instance Recovery. Eine Instance Recovery erfolgt nach einem SHUTDOWN ABORT. Der System Monitor *smon* übernimmt im weiteren das Löschen von temporären Segmenten

arch

Archiviert die vollen Online Redolog Files. Ist nur im ARCHIVELOG Mode aktiv.

reco

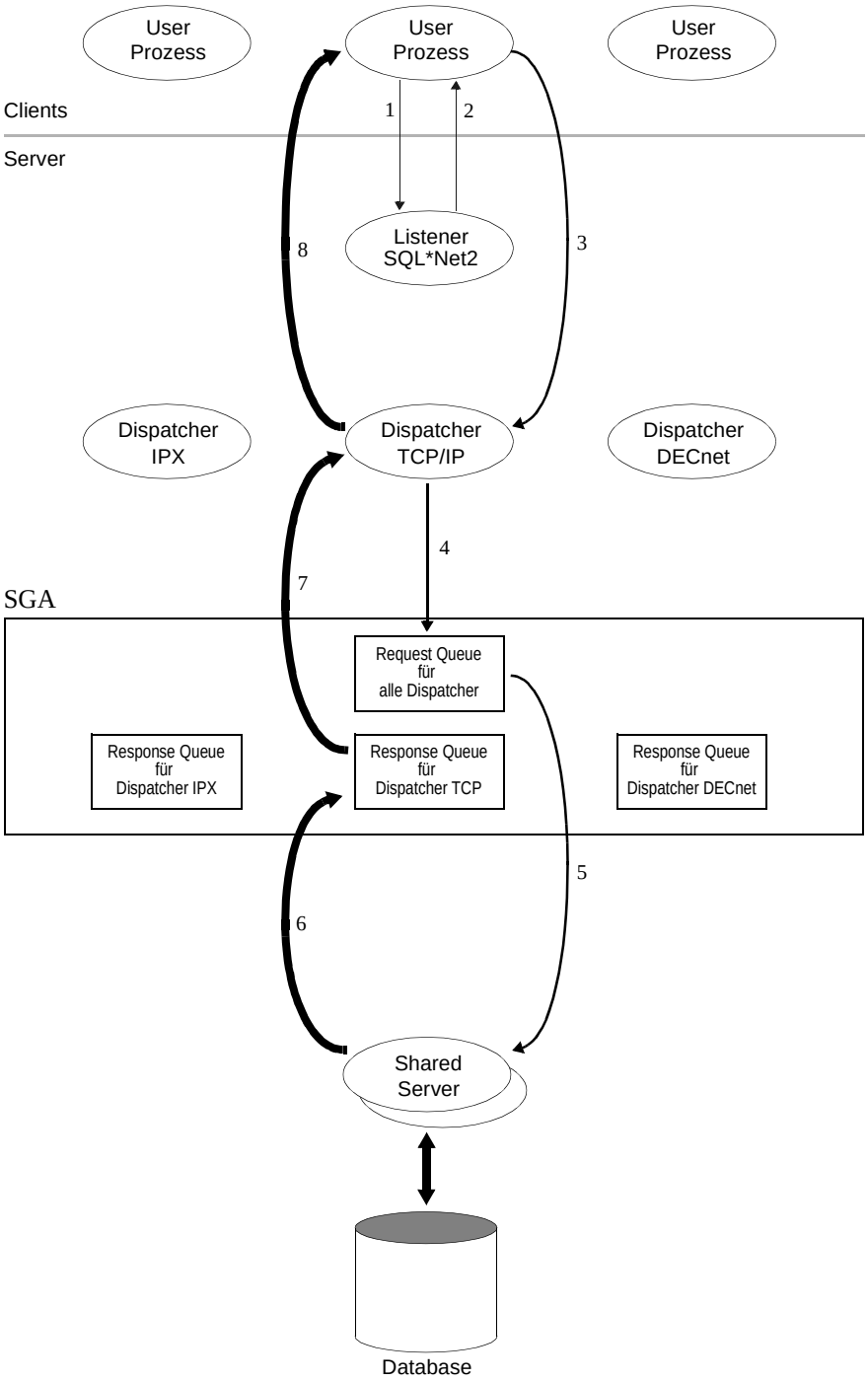
Der Recoverer Prozess *reco* versucht nicht erfolgreich durchgeführte 2 Phasen Commits bei verteilten Datenbanken zu wiederholen. Gelingt dies nicht, so werden entsprechende Einträge in die «In-Doubt» Tabellen im Datendictionary geschrieben. Solche Transaktionen müssen manuell korrigiert werden.

9. Multi-Threaded Server und SQL*Net 2

9.1 Multi-Threaded Server

Wird Oracle7 als «Dedicated Server» betrieben, so erzeugt jeder Benutzerprozess einen ihm zugewiesenen Shadowprozess. Dies kann bei vielen gleichzeitigen Benutzern zu Betriebssystem Engpässen führen.

Durch das Konzept des Multi-Threaded Server Konzepts können die Shadowprozess reduziert werden.



Ablauf

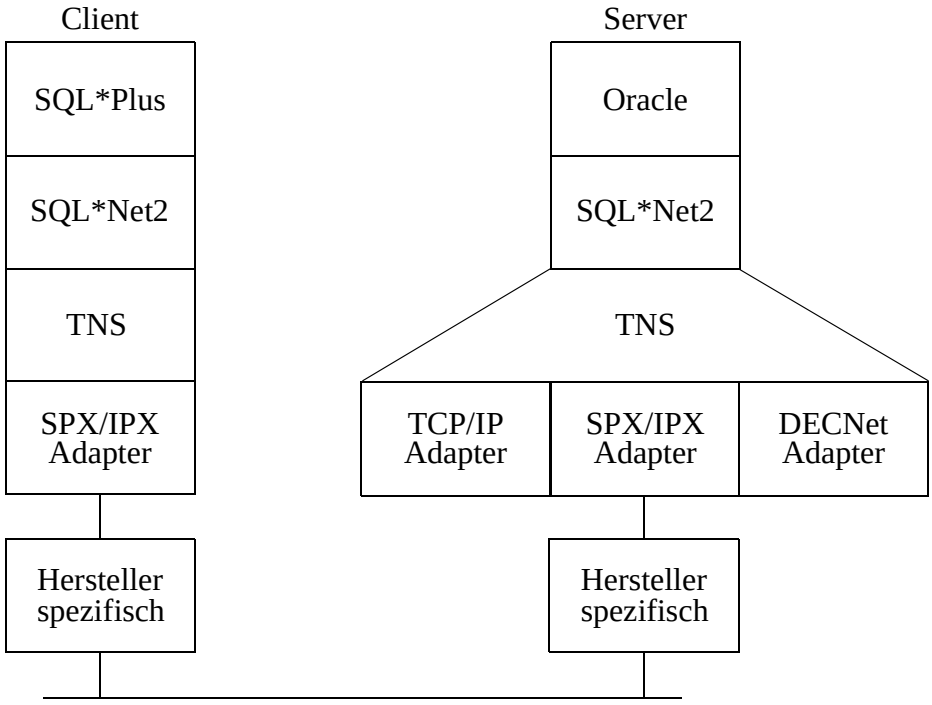
1. Client stellt Request an den Listener. Dazu wird ein SQL*Net 2 Protokoll verwendet wie beispielsweise TCP/IP, SPX/IPX, DECnet oder LU6.2. Um die protokollspezifischen Teile in ein einheiliches Format zu wandeln benutzt Oracle TNS (Transparent Network Substrate). Für jedes einzelne Protokoll liefert Oracle einen separaten Adapter.
2. Listener weist dem Client den protokollspezifischen Dispatcher Prozess zu. Pro SQL*Net2 Protokoll exitiert ein Dispatcher.
3. Der Client stellt das SQL Kommando an den zugewiesenen Dispatcher
4. Der Dispatcher stellt das SQL Kommando in die Request Queue in der SGA. Es gibt genau eine einzige Request Queue.
5. Der erste freie Shared Server holt das SQL Kommando ab und führt es via ein normaler Dedicated Server aus.
6. Der Shared Server stellt das Ergebnis in die Response Queue des entsprechenden Dispatchers. Jeder Dispatcher hat seine eigene Response Queue.
7. Der Dispatcher liefert schlussendlich das Ergebnis an den Client Prozess zurück.

TNS und SQL*Net Adapters

Wenn SQL*Net2 installiert wird, so werden die beiden Adapter BEQ und IPC automatisch installiert und konfiguriert.

BEQ: Prozesskommunikation über Unix Pipes. Dieser Adapter kann nicht mit dem MTS kommunizieren, er benutzt immer einen Dedicated Server Prozess.

IPC: Prozesskommunikation über UNIX Domain Name Sockets



Adapters Kommando

Anzeigen welche Adapter installiert sind

$ adapters

```
Installed SQL*Net V2 Protocol Adapters are:
   V2 BEQ Protocol Adapter
   V2 IPC Protocol Adapter
   V2 TCP/IP Protocol Adapter
```

Anzeige welche Adapter in SQL*Plus gelinkt sind

$ adapters sqlplus

```
SQL*Net V2 Protocol Adapters linked with sqlplus are:
   V2 BEQ Protocol Adapter
   V2 IPC Protocol Adapter
   V2 TCP/IP Protocol Adapter
```

Konfiguration Multi-Threaded Server

1. MTS Parameter in INIT.ORA definieren

- MTS_LISTENER_ADDRESS

  Mögliche Netzwerkprotokolle festlegen mit denen Datenbank via Listener kommunizieren kann. Diese Adresse muss auch in der Datei *listener.ora* definiert werden. Für UNIX Ports braucht es keinen Eintrag mehr in */etc/services*, die Portnummer kann seleber gewählt werden.

```
MTS_LISTENER_ADDRESS = "(ADDRESS=(PROTOCOL) (PORT=5555) (HOST=wizard)"
MTS_LISTENER_ADDRESS = "(ADDRESS=(decnet) (OBJECT=OUTA) (HOST=wizard)"
```

- MTS_DISPATCHERS

  Hier wird definiert, welche Dispatcher beim aufstarten der Instance gestartet werden müssen. Pro Protokoll können ein oder mehrere Dispatcher gestartet werden.

```
MTS_DISPATCHERS = "ipc, 1"# Run exactly one IPC Dispatcher
MTS_DISPATCHERS = "tcp, 2"# Run two TCP Dispatchers
```

  Damit werden die Prozess *ora_d000_MYSID*, *ora_d001_MYSID*, *ora_d002_MYSID*, *ora_d003_MYSID*, *ora_d004_MYSID* und *ora_d005_MYSID* gestartet.

- MTS_SERVICE

  Dies definiert den sogenannten *Service Name*, mit dem ein User beim CONNECT auf eine Instance einen Dispatcher benutzen kann. Oracle kontrolliert bei jedem Datenbank Connect ob ein Dispatcher für diesen Service vorhanden ist. Gemäss Oracle Manual (Admin Guide Version 7.1.4 Seite A-32) sollte *Service Name* dem ORACLE_SID entsprechen. Wenn ein Dispatcher aus irgend einem Grund kein Connect durchführen kann, so connected Oracle via Dedicated Server. *Der Service Name* muss mit (SID = xxx) im File *tnsnames.ora* übereinstimmen.

```
MTS_SERVICE = MYSID
```

  analog in tnsnames.ora:

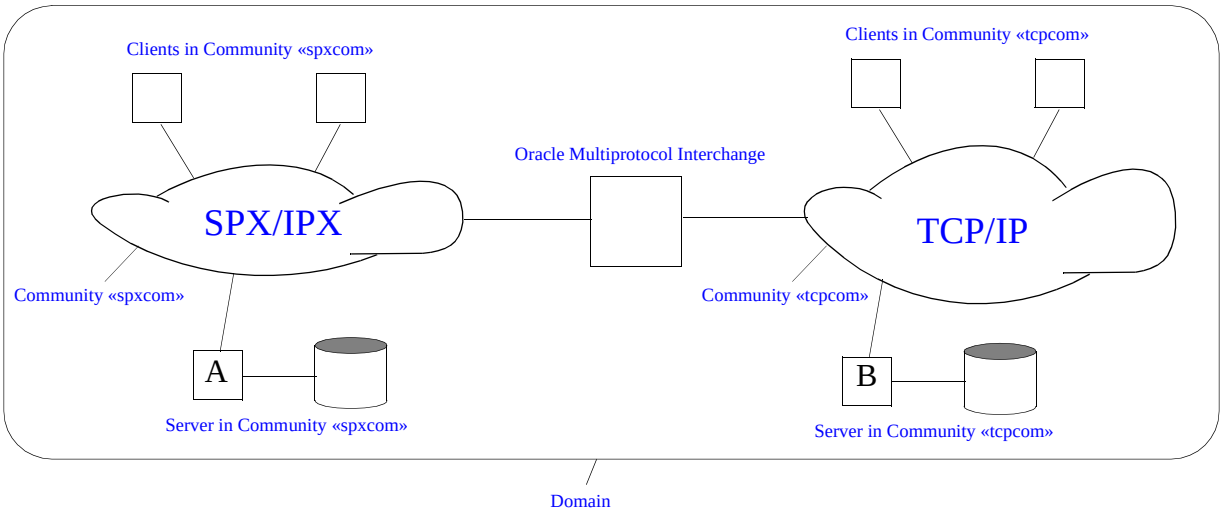
```
(CONNECT_DATA =  (SID = MYSID))
```

  Die anderen MTS Parameter sind selbsterklärend.

9.2 SQL*Net Version2

Übersicht

Mit SQL*Net2 können Datenbanken in unterschiedlichen Netzen transparent angesprochen werden.

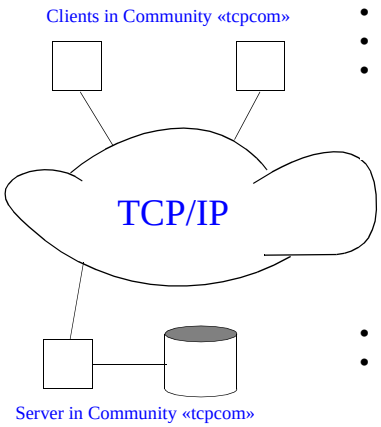


Domain: Hierarchischer Aufbau von Rechnersystemen analog dem Domain Name System im Internet wie beispielsweise *A.glue.ch.* Für flache Hierarchien verwendet Oracle den Default Domainnamen «*world*».

Community: Bezeichnung eines Netzes mit gleichem Netzwerkprotokoll wie beispielsweise «tcpcom»

Konfigurationsfiles

Sowohl Client wie auch Server benötigen verschiedene Konfigurationsfiles damit die Zuordnungen möglich werden.



- Client Konfigurationsfiles:
- tnsnames.ora
- sqlnet.ora

- Server Konfigurationsfiles:
- listener.ora

Da auf einem Server in der Regel auch Client Applikationen benutzt werden, verfügt der Server auch über das Client Konfigurationsfile «tnsnames.ora».

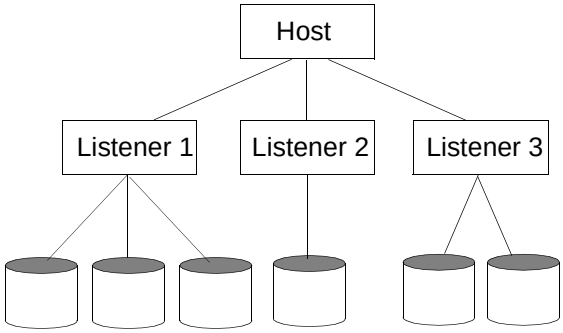
Listener

Für den Betrieb des Multi-Threaded Servers ist SQL*Net 2 erforderlich. SQL*Net 2 setzt einen Listener Prozess ein, der in der Lage ist Client Requests von verschiedenen Netzwerkprotokollen zu bearbeiten. Er verbindet den Client Prozess mit dem protokollspezifischen Dispatcher der gewünschten Datenbank. Der Listener Prozess läuft nur auf dem Datenbank Server, die Clients benötigen keinen Listener.

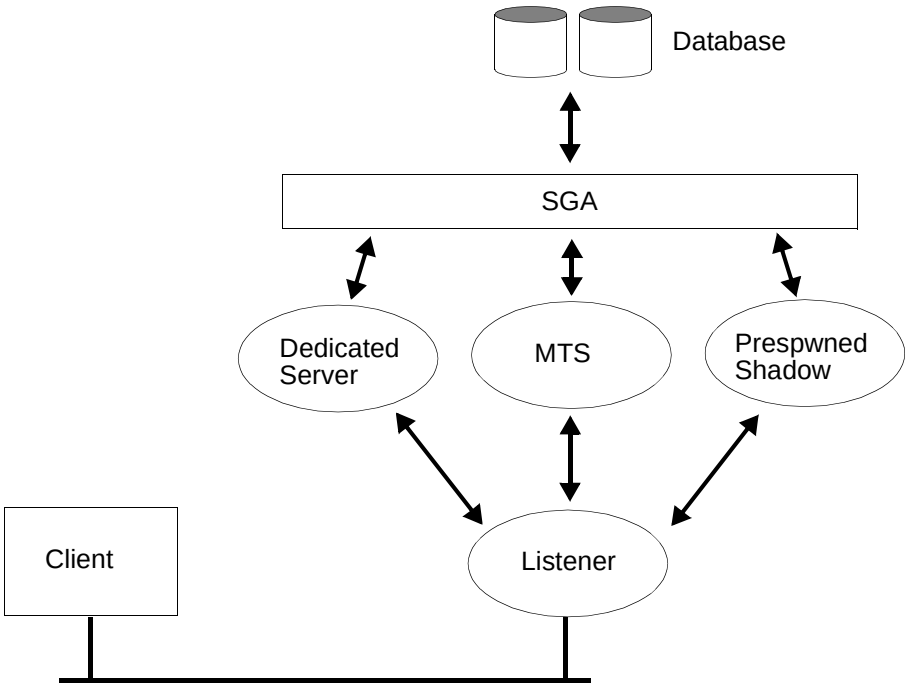
- Der Default Name des Listeners ist **LISTENER.** Da der Listener Name pro Host eindeutig sein muss hängt der Oracle Network Manager dem Listener Name den Host- und Domainnamen an:

  LISTENER ==> LISTENER_hostname.domain

- Pro Host können mehrere Listener konfiguriert werden, jeder Listener kann mehrere Datenbanken verwalten:



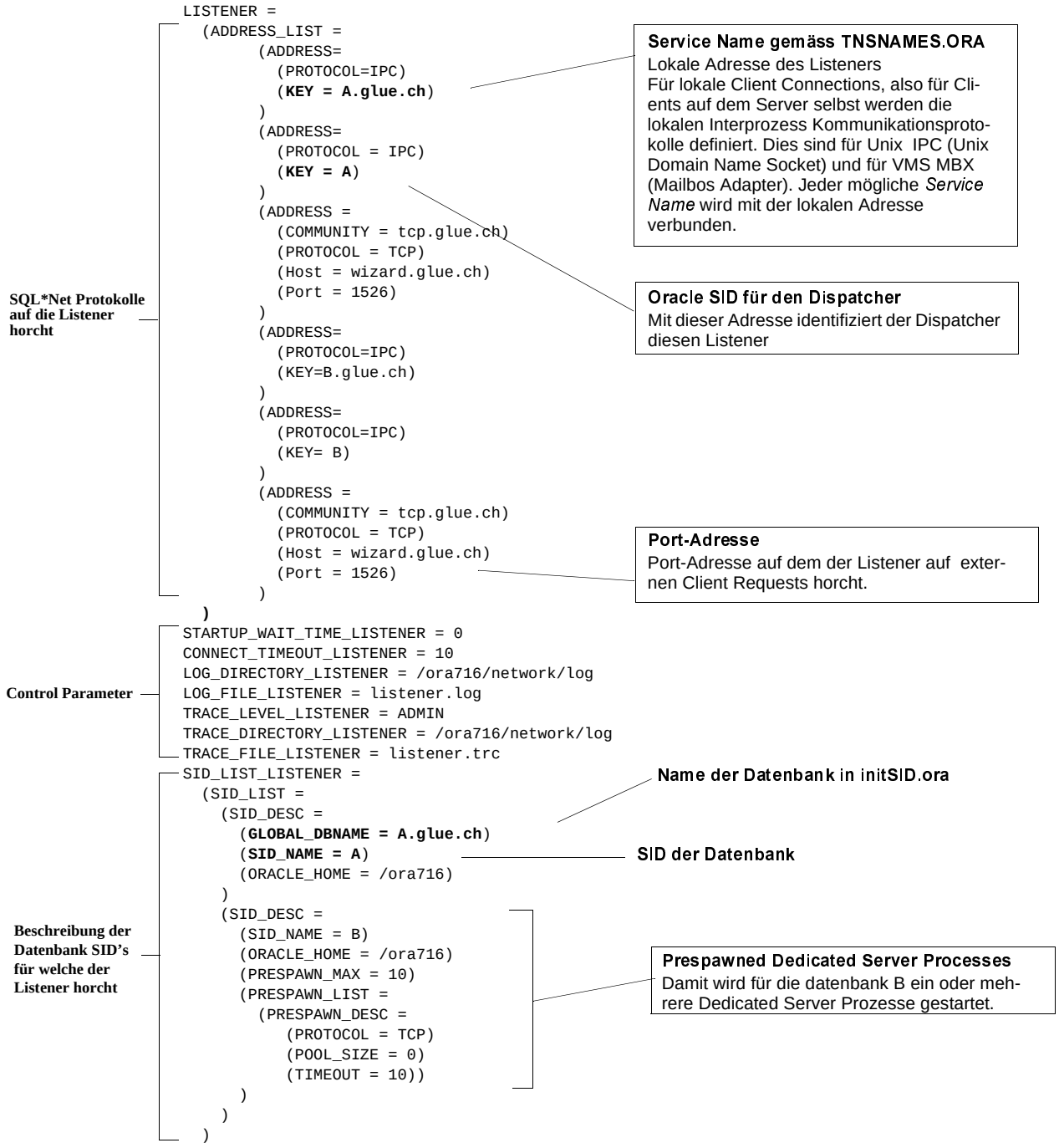
- Listener kann mit einem Dedicated Server, einem Multithreaded Server oder einem Prespawned Shadow Prozess kommunizieren



Listener.ora

Der Aufbau des Files listener.ora (ein File pro Host) besteht aus:

- dem oder den Namen des Listeners.
- den Adressen (IPC,TCP,DECNET etc) auf die der Listener horcht.
- den Datenbanken (SID's) die der Listener verwaltet.
- diversen Parametern



Beachte

Der Service Name in ADRESS_LIST muss mit SERVICE_NAME im File TNSNAMES.ORA übereinstimmen.

GLOBAL_DBNAME muss mit Eintrag in INIT.ORA übereinstimmen.

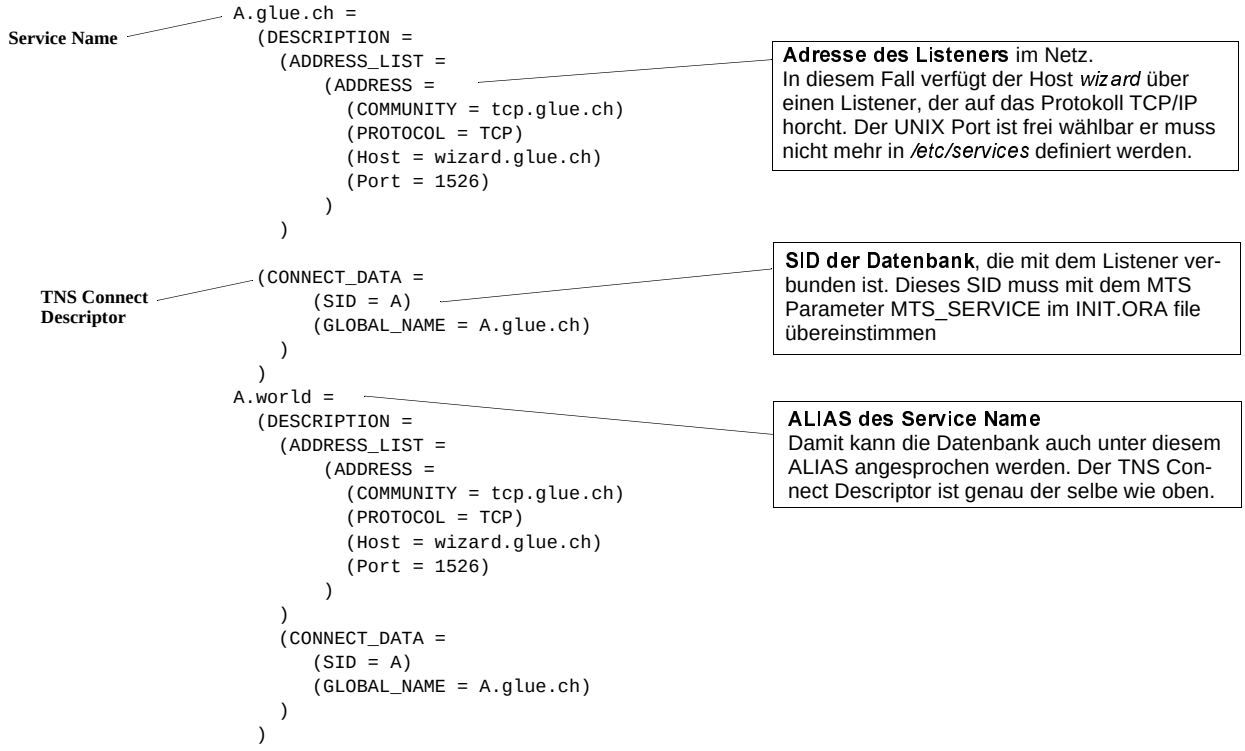
SID_NAME muss mit Oracle SID übereinstimmen.

Die Adressen ADRESS_LIST müssen auvh im INIT.ORA für den MTS Server definiert werden.

tnsnames.ora

Das Client Konfigurationsfiles tnsnames.ora weist den Weg zu vorhandenen Listener Prozessen, damit eine Verbindung zu einer Datenbank hergestellt werden kann. Der Userprozess wird mit einem sogenannten ***Service Name*** aufgerufen, der dann im Konfigurationsfile in einen *Connect Descriptor* umgewandelt wird. Dieser Connect Descriptor wird dann über das Netz an den definierten Listener geschickt.

Beispiel: `sqlplus scott/tiger@A`



- Der zweite eingetragene **Service Name** *A.world* ist ein **Alias** für *A.glue.ch*, er definiert den genau gleichen Connect Descriptor. Beim Aufruf eines Clients muss der Default Domain Name *world* nicht angegeben werden (*sqlplus scott/tiger@A* oder *sqlplus scott/tiger@A.glue.ch*). Das Konfigurationsfile kann mit dem Oracle Tools *netman* erstellt werden.

- Man beachte nochmals, dass TNS den **Connect Descriptor** dem Listeners am definierten Server überreicht. Dieser weist aufgrund des (SID = ***mysid***) die Datenbank dem Client zu. Im Parameterfile INIT.ORA muss der Multi-Threaded Server mittels mts_service = ***mysid*** zugewiesen werden.

- Service Name muss mit DB_NAME im INIT.ora File des betreffenden Servers übereinstimmen. Jeder Service muss einen Connect Descriptor aufweisen. Der Connect Descriptor beschreibt den Listener und dessen Datenbank SID die benutzt werden soll.

sqlnet.ora

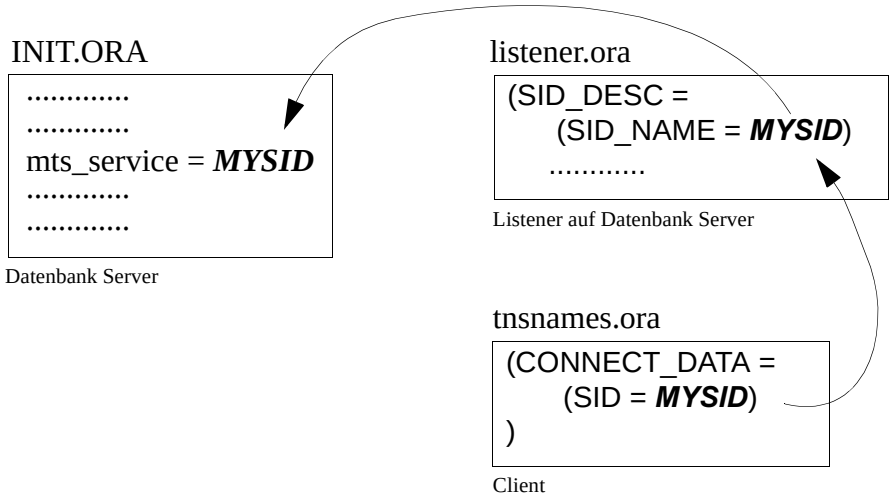
Das Konfigurationsfile sqlnet.ora wird von allen Clients im Netzwerk benötigt. Es beinhaltet die folgenden Angaben:

- Dead Connection Dedection. In zeitlichen Abständen wird die Verbindung zum Server kontrolliert.
  SQLNET.EXPIRE_TIME
- Default Domains.
  DEFAULT_DOMAIN
- Verwendung eines Dedicated Servers anstelle des MTS (Default). Man beachte, dass ein Shutdown / Startup nur über einen Dedicated Server gemacht werden kann. Oracle OEM benötigt diesen Eintrag.
  USE_DEDICATED_SERVER=YES
- Automatische Verwendung von IPC's
  Für den Oracle OEM wird die automatische Verwendung von IPC's deaktiviert:
  AUTOMATIC_IPC=OFF

```
########################################################################
# Filename......: sqlnet.ora
# Purpose.......: Configuration File for SQL*Net2
# Node..........: ASUS (OEM-Console)
# Date..........: 15.2.1997 / Martin Zahn
########################################################################
#
AUTOMATIC_IPC = OFF
TRACE_LEVEL_CLIENT = OFF
SQLNET.EXPIRE_TIME = 0
NAMES.DIRECTORY_PATH = (TNSNAMES)
NAMES.DEFAULT_DOMAIN = world
NAME.DEFAULT_ZONE = world
#
# OEM Konsole, to allow Startup/Shutdown
# of the Remote Database
#
USE_DEDICATED_SERVER = on
```

Auffinden der Datenbank

Damit der Client die gewünschte Datenbank auf dem Server findet und dort der entsprechende MTS Dispatcher gestartet werden kann müssen die Datenbank Identifiers in den Files *INIT.ORA*, *tnsnames.ora* und *listener.ora* übereinstimmen.



10. Oracle Datendictionary

Das Datendictionary (DD) besteht aus einer Reihe von readonly Tabellen, die alle beim Kreieren der Datenbank durch die Scripts *sql.bsq* und *catalog.sql* erstellt werden. Nicht nur für Oracle selbst ist das Datendictionary von Bedeutung, sondern für jeden Datenbankbenutzer enthält es viele Informationen die für ihn wichtig sein können. Änderumgen an den Datendictionary Tabellen dürfen auf keinen Fall durchgeführt werden (etwas als User SYS). Das Datendictionary ist strukturiert in Tabellen und Views wie normale Datenbankdaten.

Informationen im DD

- Namen aller Datenbankbenutzer mit ihren Rechten
- Privilegien und Rollen jedes Benutzers
- Namen der Schema Objekte (Tables, Views, Snapshots, Indexes, Clusters, Synonyms, Sequences, Procedures, Functions. Packages, Triggers, etc).
- Informationen zu Integrity Constraints
- Beanspruchter Platz durch die Datenbankobjekte

Viele Tabellen des Datendictionary werden in die SGA geladen um einen sehr raschen Zugriff zu ermöglichen.

Zugriff auf das DD

Der Zugriff erfolgt «dosiert», je nach Benutzerart. So hat beispielsweise ein DBA mehr Einsicht als ein normaler Benutzer. Es existieren folgende DD-Views

Spezielle	User	All	DBA
DICTIONARY Zeigt alle Tables, Views etc die für Benutzer zugänglich sind	USER_xxx Zugriff auf eigene Objekte	ALL_xxx Zugriff auf Objekte die der Gruppe PUBLIC zugeordnet wurden. Ebenso Zugriff auf Objekte die explizit gegranted wurden.	DBA_xxx Zugriff nur für DBA's, zeigen alle Informationen
DUAL Pseudotabelle die immer ein Resultat liefert.			

Beispiel

Dynamisch ein SQL Script generieren, das die Inhalte in den eigenen Tabellen löscht.

```
select 'delete from '||table_name||';'
from user_tables
order by table_name;
```

```
delete from BONUS;
delete from CUSTOMER;
delete from DEPT;
.....
```

11. Datenintegrität (Integrity Constraints)

In einem relationalen Datenbankmodell werden auf logischer Ebene Regeln definiert, welche die zugrunde liegenden Daten erfüllen müssen. Zur Implementierung dieser Regeln bietet Oracle zwei grunsätzlich verschiene Metoden an, die deklarative und programmtechnische Integrität.

Deklartive Integrität

Ein Grund für den Erfolg von Oracle ist bestimmt dem äusserst mächtigen Konzept der deklartiven Integritätskontrolle zuzuschreiben. Gerade im Client / Server Konzept spielt dieses Konzept heute eine grosse Rolle. Beim deklartiven Konzept werden die Regeln bereits beim Kreieren der Objekte vergeben, dies bedeutet später bei der Applikationsprogrammierung eine wesentliche Zeitersparnis. Zudem sind diese Regeln immer auf dem Server angesiedelt was in einem Client / Server Konzept grosse Vorteile auf die Performance hat. Zudem gelten diese so definierten Regeln unabhängig von der Applikation.

Folgende Regeln können deklartiv vergeben werden:

- NOT NULL constraint
- UNIQUE constraint
- PRIMARY KEY constraint
- FOREIGN KEY constraint
- CHECK constraint

Beispiele

```
CREATE TABLE kanton (
 kanton_nr    NUMBER(2)     PRIMARY KEY,
 kanton_name  VARCHAR2(5)   NOT NULL,
 datum_mut    DATE);

CREATE TABLE bezirk (
 bezirk_name  VARCHAR2(5)   PRIMARY KEY,
 kanton_nr    NUMBER(2)     REFERENCES
                            kanton(kanton_nr)
                            ON DELETE CASCADE
                            NOT NULL,
 datum_mut    DATE);

CREATE TABLE etappe (
 etappe_nr        NUMBER(10   PRIMARY KEY,
 projekt_nr       NUMBER(10)  REFERENCES
                              projekt(projekt_nr)
                              ON DELETE CASCADE
                              NOT NULL,
 subv_nr_projekt  NUMBER(5),
 subv_nr_etappe   NUMBER(2)   NOT NULL,
                              UNIQUE(
                              projekt_nr,
                              subv_nr_projekt,
                              subv_nr_etappe));
```

Programmtechnische Integrität

Komplexere Integritätsregeln, die deklarativ nicht realisiert werden können müssen programmtechnisch realisiert werden. Oracle7 bietet auch hier sehr effiziente Methoden an wie

- Datenbank Trigger
- Stored Procedures
- Applikatorische Überprüfung

Datenbank Trigger

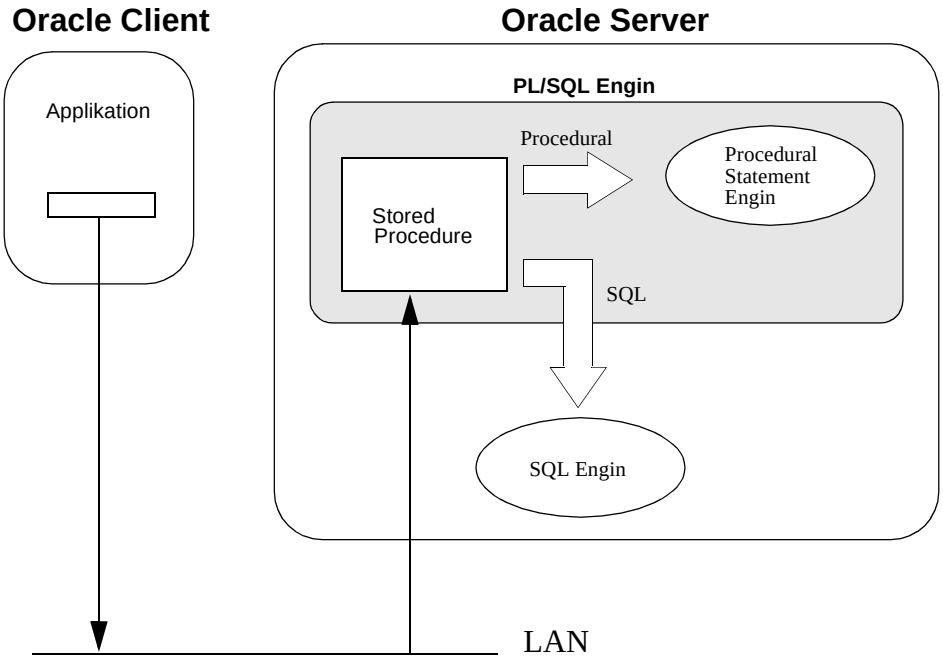
Datenbank Trigger «feuern» unabhängig von der Applikation auf dem Server. Dadurch können komplexe Integritätsregeln implementiert werden beispielsweise für ein Auditing.

Beispiel:

```
PROMPT Auditing of Table KANTON
CREATE TRIGGER audit_kanton_ch
  BEFORE INSERT OR UPDATE ON kanton
  FOR EACH ROW
  BEGIN
  :NEW.kurz_mut := USER;
  :NEW.datum_mut := SYSDATE;
  END;
```

Stored Procedures

Proceduren und Functionen können ausserhalb der Client Applikation direkt auf dem Server implementiert werden und dort von dieser aufgerufen werden. Dadurch wird ein grosser Performancegewinn erzielt.



12. Distributed Databases

12.1 Verteilungsarten

Verteilte Datenbanken gewinnen zunehmend an Bedeutung, man holt sich Informationen dort wo sie vorhanden sind. Dies setzt jedoch ein wesentlich anspruchsvolleres Konzept voraus als bisher behandelt. Moderne Datenbanksysteme wie Oracle 7.2 verfügen über mächtige Mechanismen zur Unterstützung von verteilten Datenbanken. Grundsätzlich befinden sich die Daten eines grossen Systems (Data Warehouse) nicht mehr allein lokal auf dem zentralen Server, sondern können (theoretisch) weltweit verteilt sein.

Man unterscheidet dazu folgende Verteilungsarten:

Dispersion

Tabellen sind **redundanzfrei auf den Netzknoten verteilt**, jede Tabelle kommt nur einmal auf genau einem Knoten im System vor. Dieser (optimale) Zustand wird zur Zeit kaum angestrebt, da die Netzzugriffe noch zulange dauern. Doch in Zukunft (ATM) wird dieser Zustand wohl mehr und mehr Realität, Anwendungen dazu sind zuhauf vorhanden (Reservationssysteme, Informationssysteme etc).

Replikation

Bei der Replikation werden gewisse Tabellen auf weitere Netzknoten kopiert (repliziert). Dies selbstverständlich nicht durch ein simples Kopieren, sondern unter strenger Kontrolle. Eine Variante der Replikation stellen **Snapshots** dar, die von Oracle7 unterstützt werden.

Fragmentierung

Fragmente sind Teile einer Tabelle (bestimmte Anzahl Datensätze) auf die Netzknoten verteilt.

12.2 Transparenz

Unter Transparenz versteht man das gezielte «Verstecken von Komplexizität». Dem Anwender soll der Eindruck von Einfachheit vermittelt werden. Erstaunlicherweise gelingt dies bereits in vielen Fällen.

Lokale Transparenz

Der Benutzer kann auf einer Tabelle auf einem entfernten Netzknoten genau gleich arbeiten wie wenn diese lokal vorhanden wäre. Dies ist in Oracle7 vollständig möglich durch **Database Links** und **Synonyme.**

Netzwerk Transparenz

Teilnetze des Gesamtsystems können aus unterschiedlichen Netzprotokollen aufgebaut sein (TCP/IP, IPX, DECnet etc). Dies ist in Oracle7 praktisch vollständig implementiert mittels **SQL*Net 2** und dem Oracle **Transparent Network Substrate** (TNS) sowie dem **Multiprotocol Interchange** Zusatz.

DBMS Transparenz

Datenbanken unterschiedlicher Hersteller sollen in einem Informationssystem benutzt werden können, das heißt auf den Netzknoten können verschiedene DBMS residieren. Oracle7 unterstützt dies bereits auch für sehr viele Datenbanken durch das Konzept der Oracle **Open Transparent Gateways.**

12.3 Manipulationen mit verteilten Daten

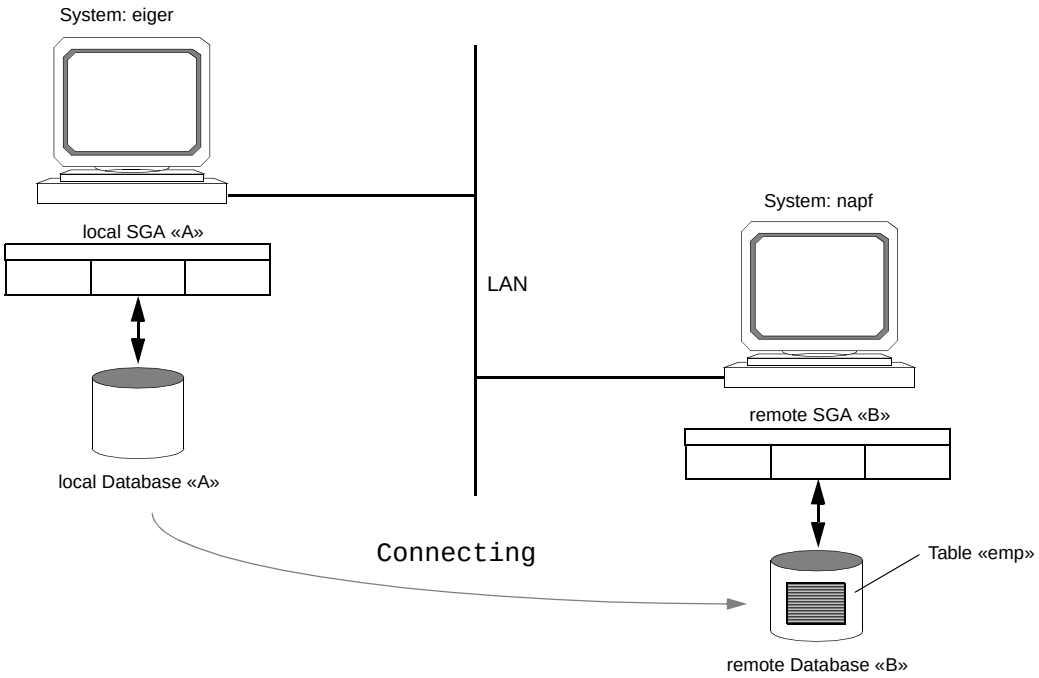
Mittels Database Links können von der lokalen Datenbank aus auf die Tables auf einer Remote Datenbank zugegriffen werden.

Folgende Funktionen sind möglich:

- Connecting auf Remote Datenbank
- Kopieren von Tables von / zu Remote Datenbank
- Queries von Tables auf der Remote Datenbank

Connecting auf Remote Datenbank

Dazu «logged» man sich auf der entfernten Datenbank ein (analog TELNET) und kann dann mit der Remote Datenbank arbeiten.




```
sqlplus scott/tiger@CONNECT_STRING
select * from emp;
```

Erstellen eines Datenbank Links

Ein explizites Connecting auf die Remote Datenbank ist nicht mehr erforderlich. Die Verbindung wird durch einen Datenbank Link erstellt. Damit kann permanent auf die Remote Tabelle zugegriffen werden. Die Tabelle existiert nur auf der Remote Datenbank. Unter Oracle6 konnten nur SELECTS durchgeführt werden, in Oracle7 sind auch INSERTS, UPDATES und DELETES möglich, selbstverständlich werden dabei die Zugriffsrechte überprüft.

```
DROP PUBLIC DATABASE LINK B.WORLD;          (Remote DB-Name = «B»)
CREATE PUBLIC DATABASE LINK B.WORLD
  CONNECT TO ota IDENTIFIED
  BY ota USING 'OTA';     (Connecct-Sting in tnsnames.ora = OTA)

CREATE PUBLIC SYNONYM SSCORDER FOR SSCORDER@B.WORLD;
```

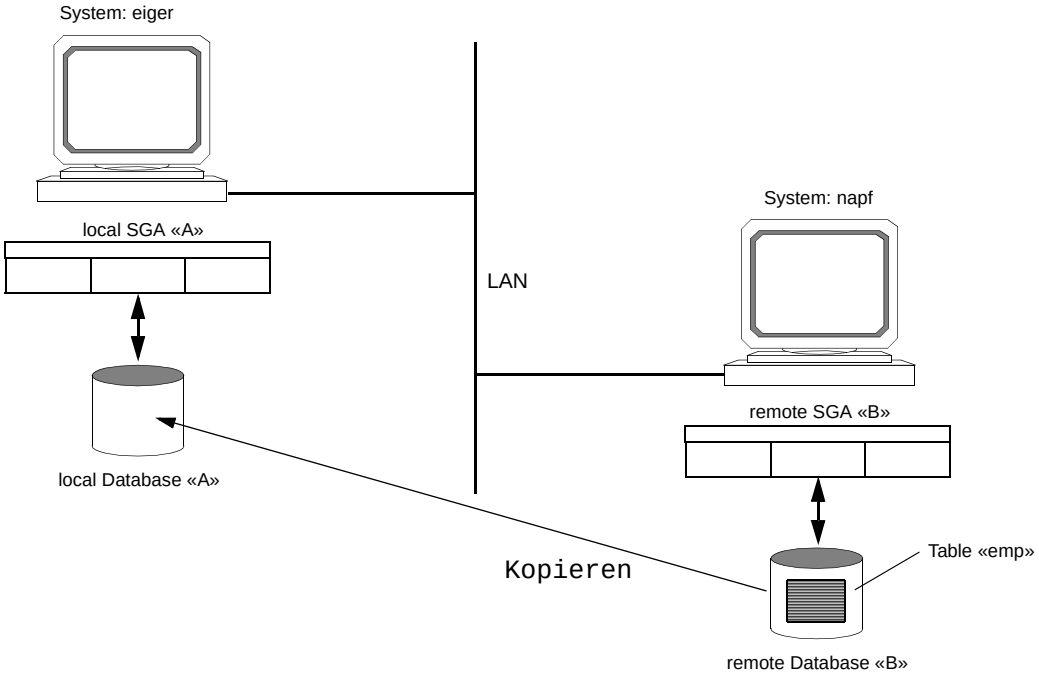
Global Naming

Der INIT.ORA Parameter GLOBAL_NAMES = TRUE erzwingt den gleichen DB-Link Namen wie die DB auf welche der DB-Link zeigt. Enforce that a dblink has same name as the db it connects to).

```
global_names  = TRUE
```

Table von Remote Database kopieren

Mit dem SQL*Plus Kommando COPY können gesamte Tabellen von Datenbank zu Datenbank kopiert werden




```
sqlplus scott/tiger
copy from scott/tiger@B to scott/tiger@A
create emp using select * from emp;
```

Table zu Remote Database kopieren

Hier in umgekehrter Richtung

```
sqlplus scott/tiger
copy to scott/tiger@A
replace emp using select * from emp;
```

13. Transaktionskontrolle

Eine Transaktion ist eine logisch zusammen hängende Einheit von einem oder mehreren SQL Kommandos. Am einfachsten stellt man sich eine Umbuchung eines Geldbetrags von einem Konto auf das andere vor. Die Transaktion umfaßt hier das Abbuchen und Aufbuchen. Ein Abbuchen ohne Aufbuchen ist absolut unzulässig. Um diese atomaren Vorgänge zu sichern stellt jedes DBMS Kommandos zur Transaktionskontrolle zur Verfügung (Commit und Rollback).

Transaktionsbeginn

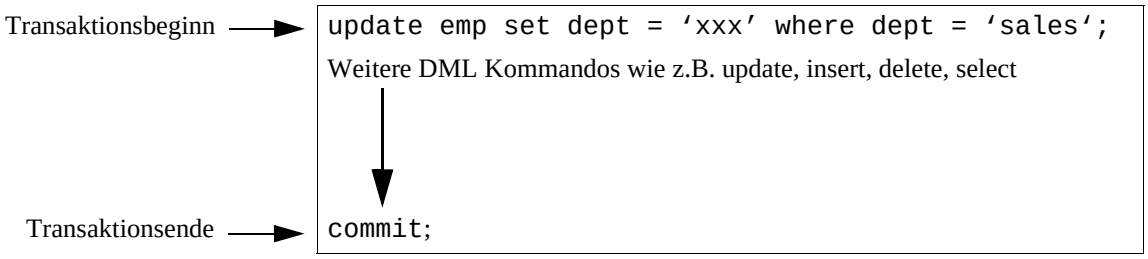
- Beim Logon (CONNECT) eines Benutzers auf die Datenbank
- Nach einem Transaktionsabschluss mit Commit.

Transaktionsende

- Beim Logoff (DISCONNECT) eines Benutzers (Implizites Commit).
- Bei einem DDL Kommando (create, alter, drop)
- Bei einem DCL Kommando (grant, revoke)
- Bei implizitem oder explizitem Rollback
- Bei implizitem oder explizitem Commit

Explizites Commit

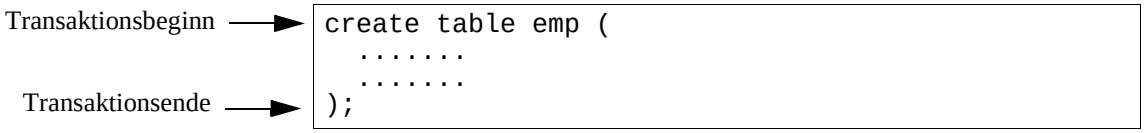
Das explizite Commit wird auf ausdrücklichen SQL Befehl ausgelöst, während das implizite Commit ein «verstecktes», durch Oracle gesteuertes Commit ist.



DML Kommandos wie `update`, `insert`, `delete` benötigen immer ein explizites Commit.

Implizites Commit

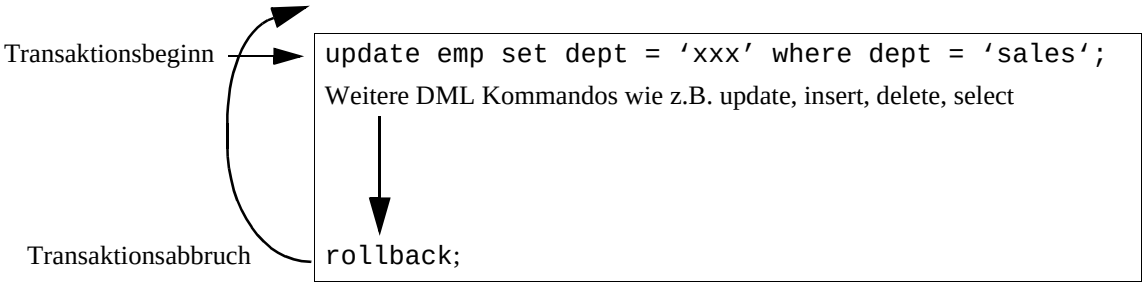
Nach jedem DDL und DCL Kommando erfolgt ein automatisches Commit durch Oracle ausgelöst:



Explizites Rollback

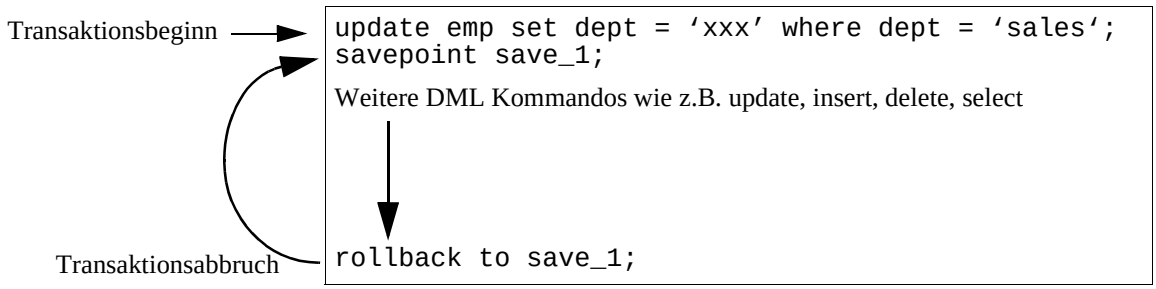
Das explizite Rollback wird auf ausdrücklichen SQL Befehl ausgelöst. Damit wird die gesamte angefangene Transaktion rückgängig gemacht. Oracle führt ein implizites Rollback nach einem Fehler während der Ausführungsphase eines SQL Kommandos durch. Zum Beispiel nach einem ORA-0001 'duplicate key in index'. Bei einem Rollback werden die bereits geschriebenen Blocks anhand der Rollbacksegmente wieder erstellt, wie

zum Zustand vor Transaktionsbeginn. Bestehende Locks werden aufgehoben. Das Kommando `rollback` macht immer die gesamte Transaktion rückgängig.



Savepoints

Manchmal ist es wünschenswert, die Transaktion nur zum Teil rückgängig zu machen. Oracle bietet dazu die sogenannten Savepoints an, dies sind Flags, welche mit dem Kommando `rollback to` *«savepoint»* angesprungen werden können.



Implizites Rollback

Oracle setzt vor jedes SQL Kommando einen «versteckten» Savepoint (impliziter Savepoint). Tritt ein Fehler **während** der EXECUTE Phase des Kommandos auf, so wird automatisch mittels implizitem Rollback zum «versteckten» Savepoint gesprungen.

14. Database Access

14.1 Privilegien

Das Konzept der System- und Objektprivilegien sowie das Rollenkonzept wurde in Oracle7 komplett neu implementiert um Zugriffsrechte sehr gezielt erteilen zu können. Die Zugriffsrechte werden auf drei Ebenen vergeben:

- Auf Datenbankebene (System Privilegien)
- Auf Objektebene (Objekt Privilegien)
- Auf Befehls-, Toolsebene

System Privilegien

System Privilegien definieren welche globalen Datenbankrechte einem Benutzer zugestehen. Zur Zeit unterscheidet Oracle7 mehr als 80 verschiedene Rechte (siehe System Administrator Guide Seite 12-2). Aus historischen Gründen wurden die drei Privilegien von Oracle6 CONNECT, RESOURCE und DBA neu in Form von vordefinierten Rollen weitergeführt.

CONNECT Privileg

Damit hat ein Benutzer das Recht:

- In Datenbank einzuloggen (CREATE / ALTER SESSION).
- Queries auf Tables oder Views anderer Benutzer, sofern «SELECT» für den User erteilt wurde. Es reicht auch, wenn die Rechte auf das Object für den speziellen Datenbank Benutzer **PUPLIC** erteilt wurden.
- Durchführung von INSERT's, UPDATE's und DELETE's auf die Tables, Views anderer Benutzer, sofern der Inhaber des Objects dies erlaubt hat. (Jedes Objekt hat einen Owner!)
- Erstellen von Views, Synonyms und Database-Links.
- Exports / Imports der eigenen Datenbank Objekte.
- Das Erstellen von Tables, Cluster, Sequences oder Indexes ist nicht möglich.

RESOURCE Privileg

Mit dem Resource Privileg kann ein User auch Objekte erstellen. Vom DBA kann das Resource Privileg auf die gesamte Datenbank oder nur auf einen bestimmten Tablespace erteilt werden.

- Beinhaltet alle «CONNECT» Rechte
- Das Erstellen von Tables, Cluster, Sequences oder Indexes

DBA Privileg

Man unterscheidet zwei DBA Privilegien, einem *System-DBA* und *Non-System-DBA* Privileg. Die Unterscheidung erfolgt auf Betriebssystem Ebene, unter UNIX wird dazu die *dba* Gruppenzugehörigkeit verwendet.

Non-System DBA Privileg

Ein *Non-System-DBA* hat alle CONNECT und RESOURCE Privilegien. Er darf zusätzlich:

- Eröffnen eines neuen Datenbank Benutzers
- SQL-Access auf alle Datenbank Objekte außer denjenigen des Datenbank Besitzers SYS.
- Erteilen von Privilegien an Datenbank Benutzer (GRANT, REVOKE)
- Kreieren von PUBLIC Synonyms. Diese sind allen Benutzern zugänglich.
- Full Export / Import
- Database Recovery
- CREATE, ALTER Tablespaces, Rollbacksegmente
- ~~Kein «CONNECT INTERNAL»~~
- ~~Kein STARUP / SHUTDOWN der Datenbank~~
- ~~Kein CREATE DATABASE~~
- ~~Durchführung von GRANTS im Namen anderer Datenbank Benutzer sind nicht möglich.~~

Meistens gibt es einige DBA's mit diesen Rechten, ihnen ist es jedoch nicht erlaubt eine Datenbank zu kreieren, eine Datenbank zu starten/stoppen oder den hochprivilegierten Befehl «CONNECT INTERNAL» abzusetzen.

System-DBA

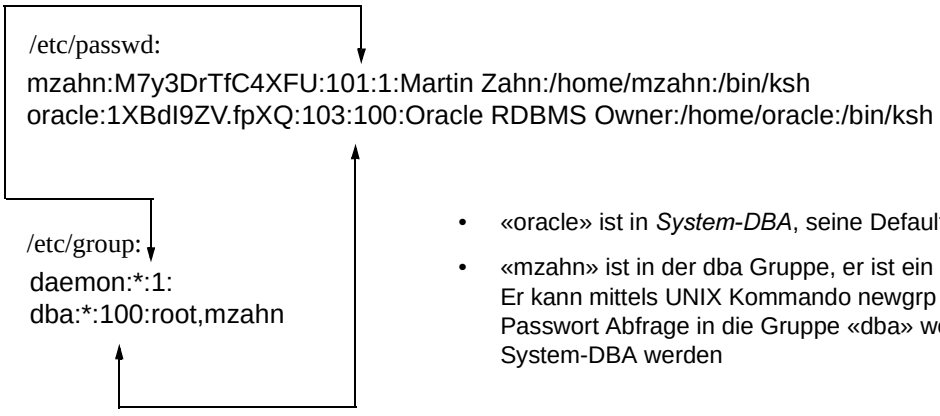
Der *System-DBA* besitzt alle Rechte, grundsätzlich sollte nur ein einziger Benutzer diese Rechte haben, dies ist der Besitzer der Datenbank.

Unterscheidung System-DBA und Non-System-DBA:

Dem *System-DBA* ist die UNIX Group *dba* als Default Gruppe zugeteilt. Non-System-DBA's können mittels newgrp in diese Gruppe wechseln, sie erhalten dann die gleichen Rechte wie ein *System-DBA*.

Wenn ein Benutzer gleichzeitig mehreren UNIX Gruppen angehören kann (zB SUN-OS oder HP-UX mit */etc/logingroup*), so entfällt das Kommando newgrp.

Beispiel: Einträge in */etc/passwd* und */etc/group* für System Privileged DBA «oracle» und Non-System Privileged DBA «mzahn»



- «oracle» ist in *System-DBA*, seine Default Gruppe ist dba.
- «mzahn» ist in der dba Gruppe, er ist ein *Non-System-DBA*. Er kann mittels UNIX Kommando newgrp ohne Group-Passwort Abfrage in die Gruppe «dba» wechseln und so ein System-DBA werden

Objekt Privilegien

Datenbank Objekte wie Tables, Views, Synonyme, Sequences und Prozeduren usw. müssen durch unsachgemäße Benutzung geschützt werden. Die folgende Tabelle zeigt eine Übersicht der möglichen Objekt Privilegien und Objekte:

Privileg	Table	View	Sequence	Procedure
ALTER object	ja		ja	
DELETE FROM object	ja	ja		
EXECUTE object (Procedure oder Function). Referenzieren von public Package Variablen				ja
CREATE INDEX ON object	ja			
INSERT INTO object	ja	ja		
REFERENCES, CREATE or ALTER eines FOREIGN Key Integrity Constraints.	ja			
SELECT ... FROM object	ja	ja	ja	
UPDATE object	ja	ja		

Objekt Privilegien auf Tables

Einige Beispiele:

1. SELECT an User Scott auf Table emp:
   ```
   grant select on emp to scott;
   ```
2. SELECT, INSERT, UPDATE an jedermann:
   ```
   grant select, insert, update on emp to public;
   ```
3. Alle Rechte an jedermann:
   ```
   grant all on emp to public;
   ```
4. SELECT, INSERT, UPDATE von mzahn.emp für jedermann entfernen
   ```
   revoke select, insert, update on emp from public;
   ```
5. SELECT von mzahn.emp für User Scott entfernen:
   ```
   revoke select on emp from scott;
   ```
6. UPDATE auf die spezifischen Columns *sal* und *comm* der Table emp:
   ```
   grant update (sal, comm) on emp to scott;
   ```

Object Privilegien auf Views

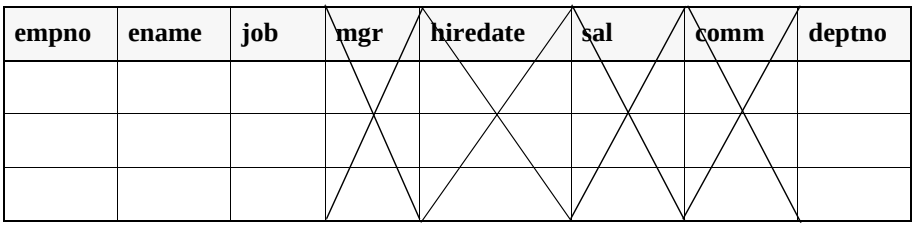
Einige Beispiele:

7. SELECT an User Scott auf View emp_view:
   ```
   grant select on emp_view to scott;
   ```
8. Überprüfung von INSERT und UPDATE in View mittels **WITH CHECK OPTION**
   ```
   create view emp_view
    as select * from emp
    where job in ('SALESMAN','MANAGER','PRESIDENT')
    and sal < (select max(sal) from emp
    where job='PRESIDENT')
    and deptno in (select deptno from dept)
    with check option;
   ```

Im obigen Beispiel werden bei einem INSERT oder UPDATE die Bedingungen (Constraints) getestet.

UPDATE auf Table Columns verhindern

Soll verhindert werden, dass bestimmte Columns der Table verändert werden können, muß nicht zwingendermassen eine View erstellt werden. Für das UPDATE Privileg kann definiert werden, welche Columns verändert werden dürfen.



UPDATE's sollen nur auf den Columns *empno*, *ename*, *job* und *deptno* erlaubt sein.

```
grant update (empno,ename,job,deptno) on emp to test;
```
(Scott)

```
update scott.emp set deptno=20 where empno=7902;
```
(Test)

1 row updated.

```
update scott.emp set sal=3500 where empno=7902;
```
(Test)

ORA-01031: insufficient privileges

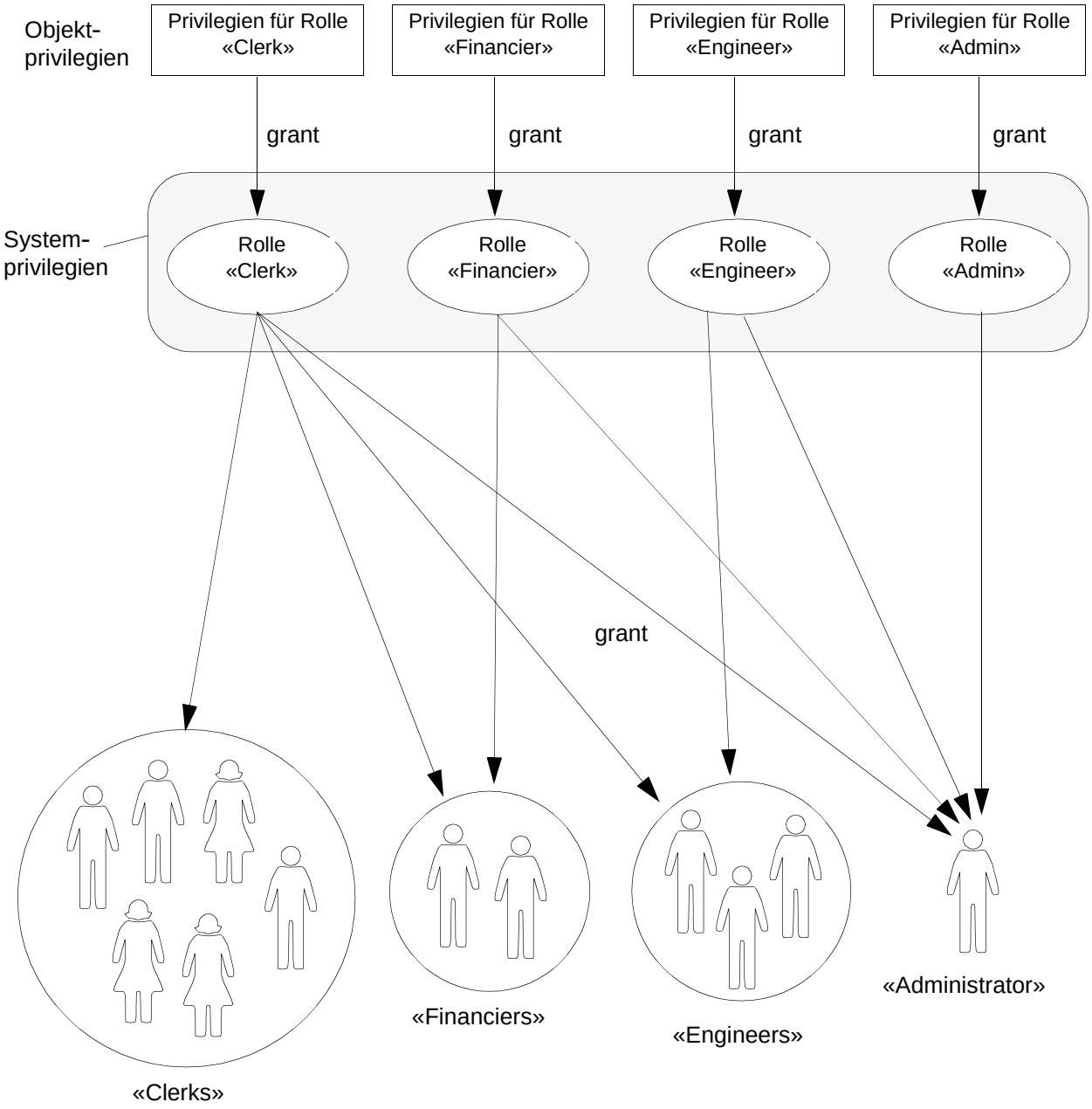
Die Column *sal* kann durch den Benutzer *test* nicht verändert werden.

14.2 Roles

Mit Oracle7 wurden Datenbank Roles eingeführt. Damit können mehrere Privilegien und Roles zusammengefaßt werden und dann als Ganzes einer Gruppe von Benutzern zugeteilt werden. Auf diese Weise wird es möglich eine komplexe Applikation leicht zu verwalten. Es ist nicht mehr notwendig jedem einzelnen Benutzer die Rechte separat zu erteilen.

Übersicht

Die folgende Übersicht zeigt, wie die Rollen «Clerks», «Financiers» und «Engineers» an reale Datenbankbenutzer weitergegeben werden können.



Ablauf zur Erstellung:

```
REM
REM Create Roles
REM
create role apl_clerk;

REM
REM Grant "System" Privileges to Roles
REM
grant create session to apl_clerk;

REM
REM Grant "Object" Privileges to Roles
REM
grant select on EMP to apl_clerk;

REM
REM Grant Roles to real DB Users
REM
create user ops$be identified externally
  default tablespace users
  temporary tablespace temp;
grant connect to ops$be;
grant apl_clerk to ops$be;
alter user ops$be default role apl_clerk;
```

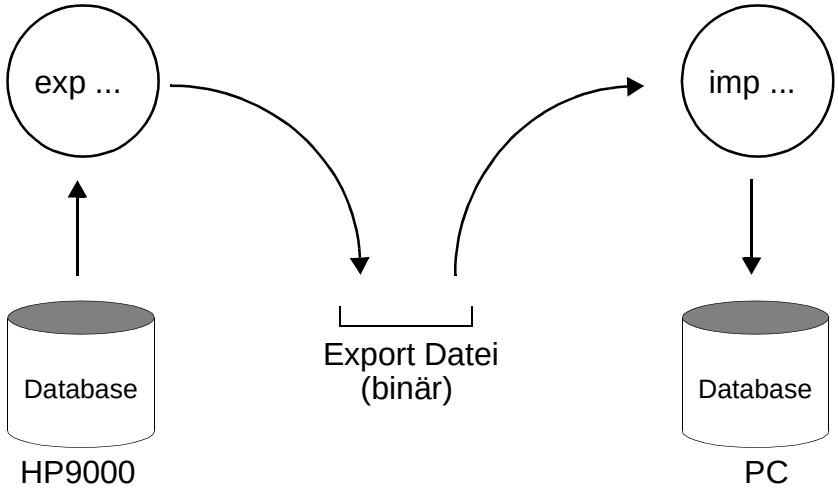
14.3 Datenbank Benutzer als OPS$User einrichten

Führt ein Benutzer das UNIX Kommando `ps -ef` durch, so wird ihm das Passwort unverschlüsselt angezeigt (`mzahn 5306 2152 ttyp3 0:00 sqlplus mzahn/mzahn`). In SQL Scripts muß meistens mittels «connect user/passwort» eine Verbindung zur Datenbank hergestellt werden, dadurch steht das Passwort im Script. Bei Scripts, welche das Data Dictionary abfragen steht somit das Passwort des Users «SYSTEM» oder des *System-DBA* «oracle» unverschlüsselt im Script. Damit dies verhindert werden kann, erlaubt Oracle einen sogenannten OPS$User. Dieser User muß unter dem gleichen Datenbanknamen auch unter UNIX vorhanden sein.

15. Export / Import

Daten aus einer Datenbank können mittels Export in ein File geschrieben werden und von dort mittels Import wieder geladen werden. Bei einem Export/Import File handelt es sich **nicht um eine editierbare ASCII Datei**, sondern um ein plattformunabhängiges Fileformat. Exports und Imports werden immer dann verwendet um Daten von einer Plattform auf eine andere zu zügeln.

Übersicht



Export / Import Modes

Export kennt drei Exportarten (Modes):

- Table Exportieren einzelner Tabellen
- User Exportieren aller Objekte eines Benutzers
- Full Database Exportieren einer gesamten Datenbank

Table Mode	User Mode	Database Mode
~~FULL=Y~~	~~FULL=Y~~	FULL=Y
~~OWNER=...~~	OWNER=...	~~OWNER=...~~
TABLES=...	~~TABLES=...~~	~~TABLES=...~~
Ermöglicht das Exportieren von Tables mit Inhalt oder nur deren Strukturen. DBA kann Tabellen eines anderen Benutzers exportieren.	Backup eines gesamten Benutzers, zum Beispiel um gelöschten Benutzer zu archivieren oder Tables von einem Benutzer auf den anderen zu übertragen.	Gesamter Datenbankexport ohne die Tabellen des Benutzers SYS. Wird vorallem angewendet um gesamte Datenbanken von System zu System zu transferieren.

Export Parameter

Durch die Angabe von Export Parametern kann der Export gesteuert werden. So ist es zum Beispiel möglich das CREATE Table .... Statement zu generieren, ohne dass Daten auch exportiert werden.

Import Parameter

Wie der Export, kann auch der Import Parametergesteuert ablaufen.

Typische Beispiele

Im Table Mode kann ein Benutzer seine Tabellen exportieren / importieren. Ein DBA hat das Recht Tabellen anderer Benutzer zu exportieren / importieren, dazu muss er das Privileg EXP_FULL-DATABASE haben.
Wird der Parameter rows=n gesetzt, so wird nur das CREATE TABLE Statement exportiert ohne die Daten.

Tabelle mit Daten exportieren / importieren

```
exp userid=scott/tiger tables=\(emp\) file=emp.dmp
imp userid=scott/tiger tables=\(emp\) file=emp.dmp
```

Tabellenstrukturen (ohne Daten) exportieren

```
exp userid=scott/tiger rows=n tables=\(emp\) file=emp.dmp
imp userid=scott/tiger tables=\(emp\) file=emp.dmp
```

Indexfile erstellen

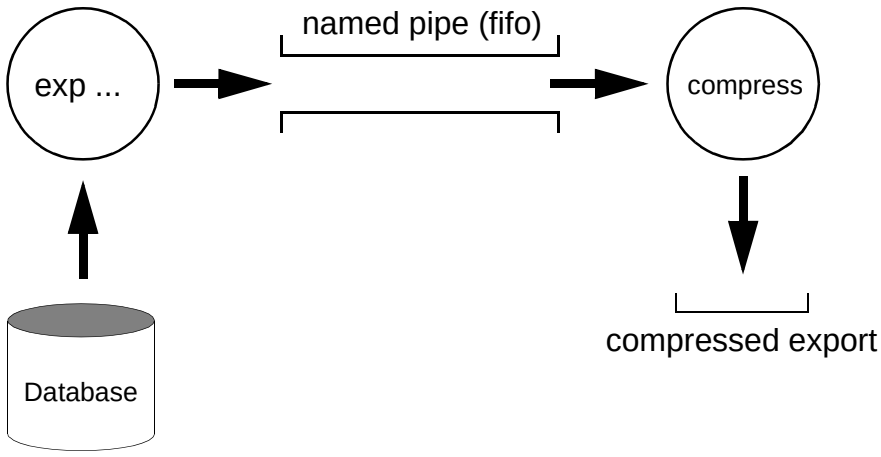
Sollen die Indexe einer Tabelle neu erstellt werden, so ist es am einfachsten wenn man ein Indexfile mittels Import erstellt. Dieses generierte Indexfile kann dann benutzt werden um die Indexe neu anzulegen.

```
exp userid=scott/tiger file=emp.dmp
imp userid=scott/tiger indexfile=y file=emp.dmp
```

Damit ist das Indexfile y.sql kreiert worden, es enthält alle CREATE INDEX Statements. Um die Indexe neu anzulegen muss nur noch das Script y.sql mittels sqlplus scott/tiger @y.sql gestartet werden.

Export / Import in komprimiertes File

Bei sehr großen Export-Dateien kann es vorkommen, dass der Platz auf der Disk nicht ausreicht; es muß komprimiert werden. Das Export-Utility bietet jedoch keine Option zum Komprimieren der Datei an. Eine Möglichkeit bietet sich mittels UNIX Named Pipe an. Man kreiert eine Named Pipe und veranlaßt *exp* hineinzuschreiben. Auf der anderen Seite der Pipe liest *compress* die Daten und komprimiert sie.




```
mknod fifo p
nohup compress < fifo > full_export.dmp &
exp system/... full=y file=fifo grants=y compress=y indexes=y

nohup uncompress < full_export.dmp > fifo &
imp system/... full=y file=fifo ignore=y grants=y
```

Export / Import via named Pipe direkt auf Tape

```
mknod fifo p
nohup dd if=fifo of=/dev/rmt/0m obs=5120 1>/dev/null 2>&1 &
exp system/... full=y file=fifo grants=y compress=y indexes=y

nohup dd if=/dev/rst0 of=fifo ibs=5120 1>/dev/null 2>&1 &
imp system/... full=y file=fifo ignore=y grants=y
```

Einzelne Table ab Full-Export zurücklesen

```
imp userid=system/... tables=HBKONTO fromuser=ERW
file=full_exp.dmp
```

Tabellen in anderen Tablespace verschieben

Export/Import eignet sich sehr gut um Storage Parameter zu optimieren. Sind zB die Anzahl Extents zu klein, so müssen diese auf den Datenbank Objekten geändert werden. Im folgenden Beispiel wird der Tablespace-name geändert.

1. Datenbank exportieren:

```
exp system/... owner=scott ndexes=y grants=y compress=y
file=dbfull.dmp
```

2. Indexfile erstellen:

```
imp system/... full=y indexfile=dbfull.sql file=dbfull.dmp
```

3. «CREATE TABLE ...» Statements kreieren aus Indexfile. Dazu kann das sed Script *create_table_from_indexfile.sh* verwendet werden. Dieses Script filtert die CREATE TABLE Statements heraus. Manuell muss noch das CONNECT user/pw nachgetragen werden. Anschließend wird der Tablespacename geändert.

```
# Generieren der CREATE TABLE Statements aus Indexfile
#
# sed Kommandos
#
# "/^REM/!d"          Loesche alle Zeilen die nicht ein 'REM' am Anfang
#                     der Zeile haben
# "s/^REM  //"        Loesche String 'REM  '
# "/rows/d"           Loesche alle Zeilen mit 'rows'
# "/;/a\\             Fuege nach jedem ';' ein 'DISCONNECT;' ein
# DISCONNECT;"\
# "/CONNECT.*/s//&;/" Fuege ein ';' an jede Zeile die 'CONNECT' enthaelt
#                     am Ende der Zeile an: CONNECT SYSTEM --> CONNECT

# ---------------------------------------------------------------------
echo "Edit Output-File 'create_tables.sql' with Passwords for Users:"
sed -e "/^REM/!d"\
    -e "s/^REM  //"\
    -e "/rows/d"\
    -e "/;/a\\
    DISCONNECT;"\
    -e "/CONNECT.*/s//&;/" $1 > create_tables.sql
echo "Output-File 'create_tables.sql' created"
```

4. Tabellen in altem Tablespace löschen
5. Tables in neuem Tablespace vorkreieren

```
sqlplus scott/tiger @create_tables
```

6. Usertabellen importieren. Es empfiehlt sich die Datenbank ohne Indexe zu importieren, da Fehler auftreten können. Die Indexe werden anschließend aus dem Indexfile erstellt:

```
imp userid=system/osiris fromuser=scott touser=scott ignore=y indexes=n
file=dbfull.dmp
```

7. Indexe aus Indexfile erstellen. Dazu wird das Indexfile *dbfull.sql* editiert.

Auf diese Weise können auch andere Storage Parameter verändert werden.

Probleme mit Export / Import

Für viele DBA's bedeutet ein Export / Import unter Oracle ein Spiel mit dem Feuer. Immer wieder stellt sich die Frage, was exportiert Oracle standardmäßig ? Im obigen Output von Export sieht man, welche Datenbankobjekte exportiert werden, man beachte, dass unter Oracle 7.1.4 die Datenbank Roles (noch) nicht exportiert werden. Ignoriert man diese Tatsache, so gibt es nach einem Import eine böse Überraschung. Um das Schlimmste zu verhindern empfiehlt sich folgendes Vorgehen.

1. Objekte vor dem Export dokumentieren

```
sqlplus scott/tiger
select object_type,count(*) from user_objects group by
object_type;

OBJECT_TYPE    COUNT(*)
-------------  ----------
INDEX          4
SEQUENCE       3
TABLE          10
VIEW           1
```

2. Roles des Benutzers dokumentieren

```
sqlplus scott/tiger
select * from session_roles;

ROLE
------------------------------
CONNECT
RESOURCE
APL_CLERK
```

Die Tabelle *user_objects* und *session_roles* sind sehr wichtige Datendictionary Tables. Die Tabelle *user_objects* enthält die Definitionen der Objekte eines Benutzers. Die Tabelle *session_roles* enthält die aktive Role eines Benutzers wenn er sich auf die Tabelle connected.

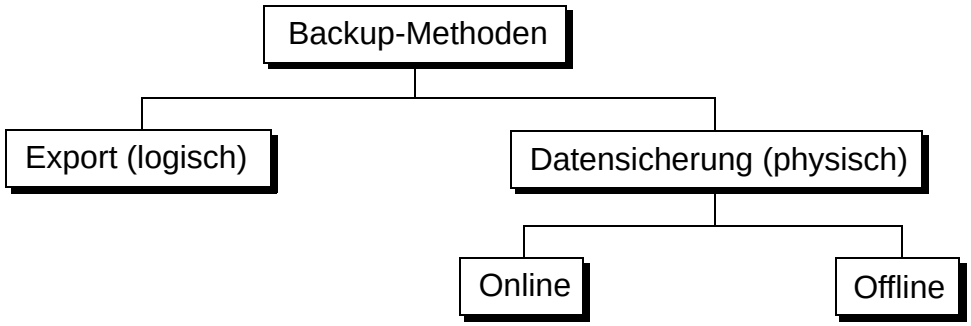
Wichtigste Punkte zu Export / Import

- Daten werden immer in gleichen Tablespace zurückgeladen. Existiert dieser Tablespace nicht, so muss er vorkreiert werden, dies wird durch Import nicht gemacht. Soll in einen anderen Tablespace importiert werden so müssen die Tabellen in diesem Tablespace vorkreiert werden (siehe Beispiel).
- Wird *compress=y* beim Export angegeben so werden die Daten in einen Extent geladen (Möglichkeit um fragmentierte Extents zu reorganisieren).
- Mit *ignore=y* ist es möglich bestehende Rows zu ignorieren, es besteht aber die Gefahr, dass gleiche Rows mehrfach geladen werden.
- Mit *commit=y* wird am Ende eines Array Fetches ein Commit durchgeführt. Dadurch können auch größte Tabellen geladen werden, ohne die Fehlermeldung CAN NOT ALLOCATE .... zu erhalten. Allerdings dauert der Importvorgang dann etwas länger.
- Die Tabellen des Users SYS werden nie exportiert. Diese werden beim Erstellen einer Datenbank neu angelegt. Nur so ist eine Migration von einer alten Version auf eine Neue möglich.

16. Backup und Recovery

Die Notwendigkeit einer Datensicherung einer Datenbank ist sicher unbestritten. Oracle bietet verschiedene Möglichkeiten zur Datensicherung an, eine übersicht zeigt die folgende Zusammenstellung.

Übersicht:



- Sicherung des Datenbank Inhalts (Tables, Indexe, Grants, Constraints, Synonyme usw)
- Sicherung der Strukturen (Indexfile)
- Datenbank muß online sein
- Häufigste Anwendungen sind die Migration auf neue Versionen, Datentransport von Datenbank zu Datenbank und Datenreorganisation
- Oracle Tool exp / imp

- Physische Sicherung der Daten während des laufenden Betriebs.
- Sicherung auf Tablespace-Ebene
- Datenbank befindet sich im ARCHIVELOG Mode.
- Sicherstellung der Datenbankfiles mittels UNIX Kommandos
- Wiedererstellung der Daten bis zur letzten abgeschlossenen Transaktion mittels Rollforward / Rollback

- Physische Sicherung der Daten wenn Datenbank gestoppt ist.
- Datenbank befindet sich im NOARCHIVELOG Mode.
- Sicherstellung der Datenbankfiles mittels UNIX Kommandos.
- Transaktionen seit letzter Sicherung sind verloren.
- Sicherung nur nachts möglich wenn Datenbank gestoppt werden kann.

Wie bereits gesehen sichern die Redologfiles die Konsistenz der Datenbank. Diese Redologfiles werden zyklisch immer wieder neu beschrieben. Der einzige Unterschied zwischen Online Backup (ARCHIVELOG) und Offline Backup (NOARCHIVELOG) besteht also darin, bei einem Logswitch das Redologfile zu sichern. Eine Umstellung auf eine der Betriebsarten kann jederzeit erfolgen.

Umstellung auf ARCHIVELOG

Bei der Umstellung auf ARCHIVELOG Mode sind zwei Sachen zu tun:

- Änderung des Log Modes
- Automatische Archivierung der Redologfiles

Vorgehen:

1. Datenbank stoppen.
2. Datenbankparameter setzen in: *init.ora* und *config.ora*

   ```
   log_archive_start = true
   log_archive_dest  = /oracle7/kurs/arch/log
   ```

   Mit dem Parameter *log_archive_start* wird das automatische Archivieren der Redologfiles aktiviert. Der Parameter *log_archive_dest* definiert das Archivelogdirectory.

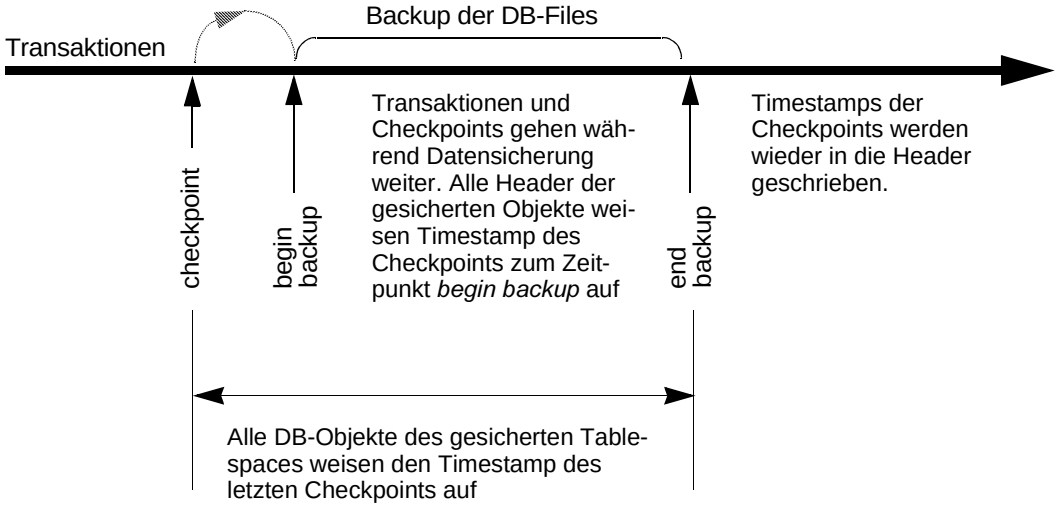
3. ARCHIVELOG Mode im Datadictionary aktivieren

   ```
   svrmgrl
   connect internal
   startup mount
   alter database archivelog
   alter system archivelog start
   archive log list
   ```

16.1 Online Backup

Funktion

Im ARCHIVELOG Mode erfolgt der Backup der Datenbank während des laufenden Betriebes, die Datenbank wird nicht mehr gestoppt. Jeder Tablespace wird einzeln gesichert mit den zugehörigen Datenbankfiles. Während des Online Backups werden die «Checkpoint Timestamps» nicht mehr in die Header der Datenbankfiles geschrieben. Alle Datenbank-Objekte welche gesichert werden weisen also die Zeit (Timestamp) des letzten Checkpoints auf. Damit kann Oracle bei einem Recovery automatisch feststellen, welche Offline Redolog Files zurückappliziert werden müssen.

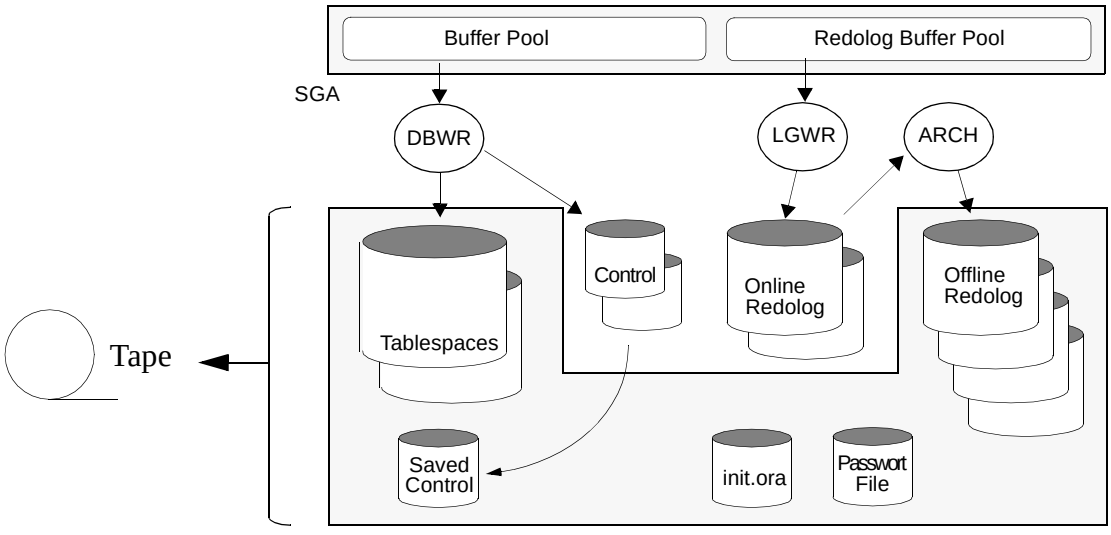


Inhalt des Online Backups

Nebst den Datenbankfiles müssen auch die archivierten Offline Redologfiles und ein Controlfile gesichert werden. **Die Online Redologfiles sind nicht zu sichern, da diese nicht in einem konsistenten Zustand sind**. Sie dürfen grundsätzlich nicht verloren gehen, deshalb sind diese äusserst wichtigen Files allenfalls auf eine andere Disk zu duplizieren (Duplexed Redolog Files). Verliert man die Online Redologfiles, so verliert man immer Transaktionen, ein Cancel based Recovery muss angewendet werden.

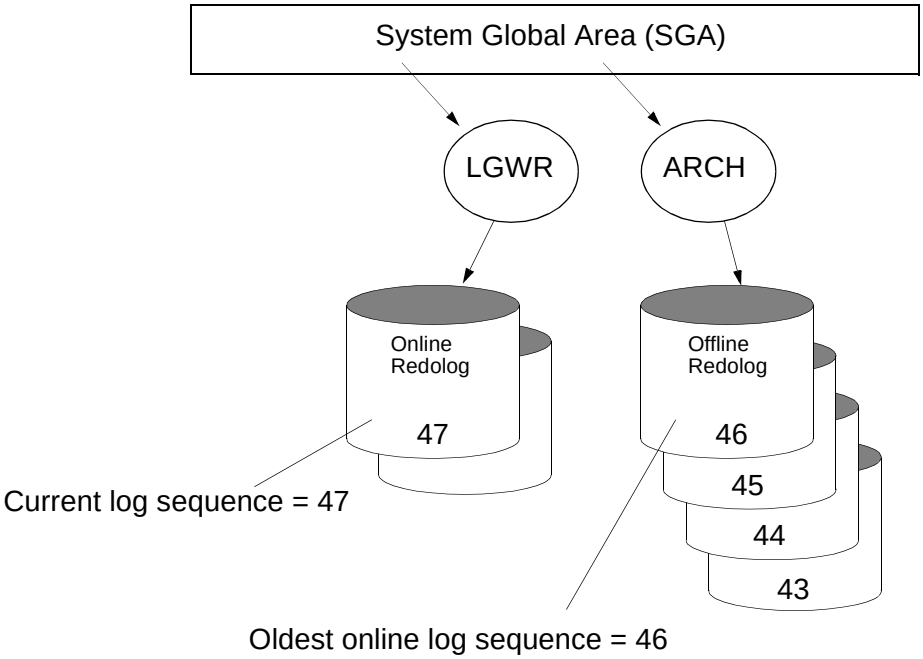
Ein Online Backup besteht also aus

- Allen Datenbankfiles (sämtliche Tablespaces)
- Allen archivierten offline Redologfiles
- Einem archivierten Controlfile
- Parameterfile init<SID>.ora
- Passwortfile



Log Sequence Nummer

Jede Redologfile das archiviert wird bekommt eine Log-Sequence Nummer, die fortlaufend numeriert ist. Diese Nummer ist für ein Recovery von großer Bedeutung, sie muss deshalb zusammen mit dem Online Backup protokolliert werden.



Current log sequence:
Redologfile das aktuell online ist (Online Redolog)

Oldest online log sequence:
Redologfile das zuletzt archiviert wurde.

Wenn der Online Backup mit BEGIN BACKUP gestartet wird muss die Nummer «Oldest online log sequence» protokolliert werden. Sie definiert den Beginn der archivierten Redologfiles. Alle älteren archivierten Redologfiles können nach dem Backup auf der Harddisk gelöscht werden.
Die Online Redologfiles dürfen im Online Backup Mode nicht mitgesichert werden, da sie keinen definierten Zustand aufweisen. Ein konsistenter Online Backup weist zudem immer ein mit
ALTER DATABASE BACKUP CONTOLFILE gesichertes Controlfile auf.
Die Durchführung des Online Backups kann vollautomatisch erfolgen wenn dazu das folgende Script verwendet wird.

Logging des Online Backup

Im *alert.log* wird der Online Backup durch das RDBMS protokolliert. Von großer Wichtigkeit ist, dass zu einem BEGIN BACKUP ... auch immer ein END BACKUP gehört, damit der Online Backup konsistent ist.

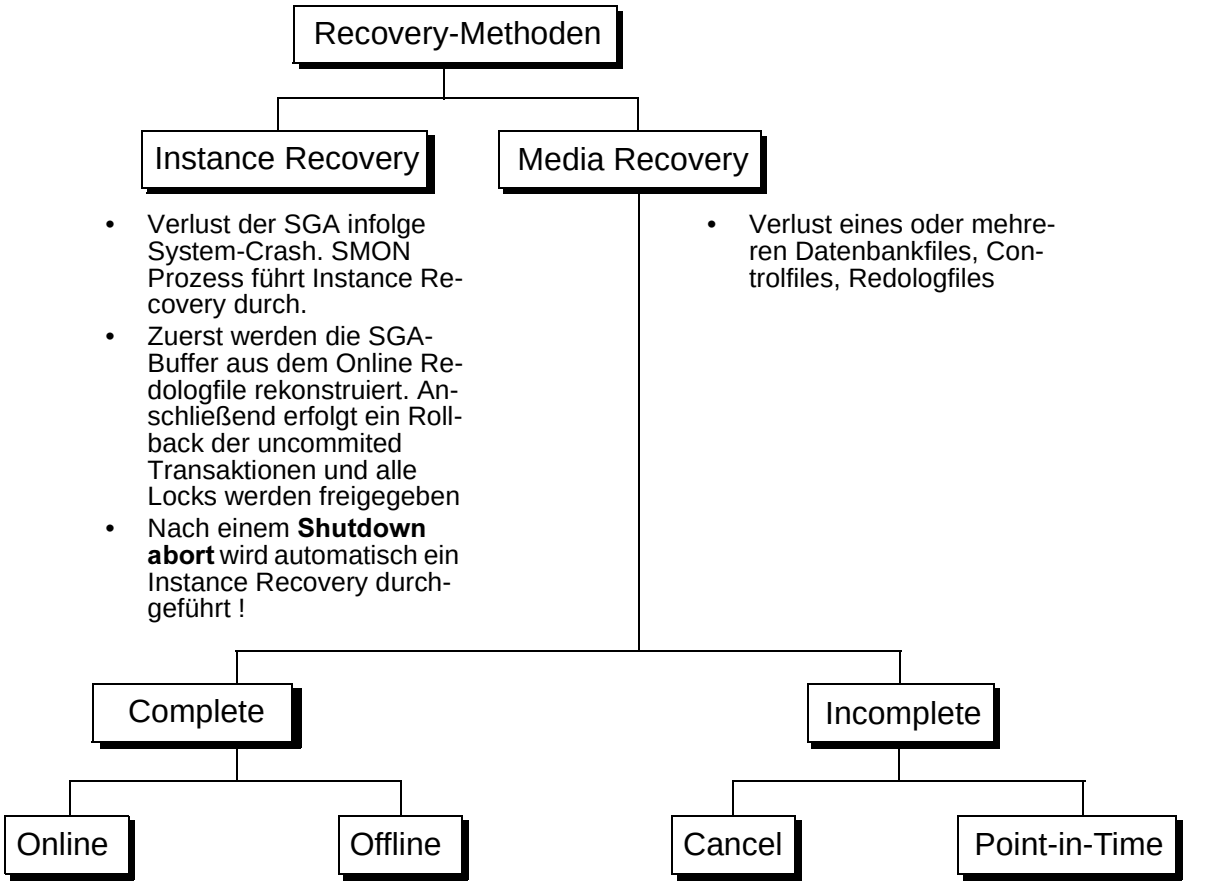
```
Tue Oct 10 12:16:38 1995
Thread 1 advanced to log sequence 588
  Current log# 2 seq# 588 mem# 0: /oracle7/kurs/log_group2_1B.dbf
  Current log# 2 seq# 588 mem# 1: /oracle7/kurs/log_group2_2B.dbf
Tue Oct 10 12:16:38 1995
alter tablespace system begin backup       <------ Start Online Backup für SYSTEM Tablespace
Tue Oct 10 12:16:38 1995
Completed: alter tablespace system begin backup
Tue Oct 10 12:16:38 1995
alter tablespace rbs begin backup
Completed: alter tablespace rbs begin backup
Tue Oct 10 12:16:38 1995
alter tablespace temp begin backup
Completed: alter tablespace temp begin backup
Tue Oct 10 12:16:39 1995
alter tablespace tools begin backup
Completed: alter tablespace tools begin backup
Tue Oct 10 12:16:39 1995
alter tablespace users begin backup
Completed: alter tablespace users begin backup
Tue Oct 10 12:16:39 1995
alter database backup controlfile to
'/oracle7/kurs/back_ctrlfile.ctl' reuse
Completed: alter database backup controlfile to
'/oracle7/kur...
Tue Oct 10 12:17:20 1995
alter tablespace system end backup         <------ End Online Backup für SYSTEM Tablespace
Completed: alter tablespace system end backup
Tue Oct 10 12:17:20 1995
alter tablespace rbs end backup
Completed: alter tablespace rbs end backup
Tue Oct 10 12:17:20 1995
alter tablespace temp end backup
Completed: alter tablespace temp end backup
Tue Oct 10 12:17:20 1995
alter tablespace tools end backup
Completed: alter tablespace tools end backup
Tue Oct 10 12:17:20 1995
alter tablespace users end backup
Completed: alter tablespace users end backup
```

Damit hat man einen konsistenten Online Backup, der für ein Recovery benötigt wird. Man beachte nochmals, dass der Online Backup während dem laufenden Betrieb erfolgen kann. Durch die Checkpoint Timestamps kann Oracle bei einem Recovery die notwendigen Offline Redologfiles anfordern.

16.2 Recovery

Es existieren verschiedene Recoverymethoden, abhängig davon welche Datenbankfiles verloren gehen.

Übersicht:



Instance Recovery

- Verlust der SGA infolge System-Crash. SMON Prozess führt Instance Recovery durch.
- Zuerst werden die SGA-Buffer aus dem Online Redologfile rekonstruiert. Anschließend erfolgt ein Rollback der uncommited Transaktionen und alle Locks werden freigegeben
- Nach einem **Shutdown abort** wird automatisch ein Instance Recovery durchgeführt !

Media Recovery

- Verlust eines oder mehreren Datenbankfiles, Controlfiles, Redologfiles

Complete / Online

- Verlust eines Tablespaces
- Datenbank befindet sich im ARCHIVELOG Mode.
- Datenbank ist geöffnet
- Tablespace darf keine Rollbacksegmente haben
- SYSTEM Tablespace darf nicht betroffen sein.
- Recovery stoppt automatisch, wenn alle Offline Redologfiles appliziert sind.
- Erfolgen Fehlermeldungen, so muß Offline Recovery durchgeführt werden.
- recover tablespace *name ...*

Complete / Offline

- Verlust SYSTEM-Tablespace oder Tablespace mit aktiven Rollbacksegmenten
- Gesamte Datenbank oder einzelne Datenbankfiles können in Recovery einbezogen werden.
- Datenbank ist geschlossen
- Datenbank befindet sich im ARCHIVELOG Mode.
- Recovery stoppt automatisch, wenn alle Offline Redologfiles appliziert sind.
- Häufigste Methode !
- startup mount
  connect internal
  recover database ...

Incomplete / Cancel

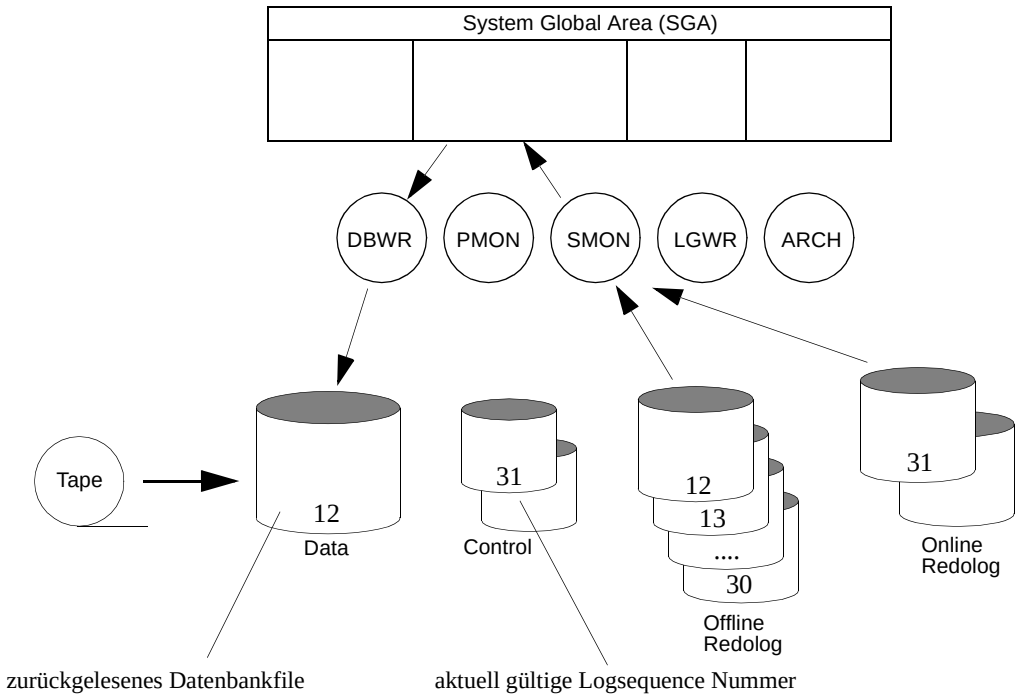
- Verlust Online Redologfile, Controlfile
- Immer dann durchführen, wenn infolge eines korrupten Offline Redologfiles Recovery vor dem Ende abgebrochen werden muß.
- Datenbank ist geschlossen
- Datenbank befindet sich im ARCHIVELOG Mode.
- Abbruch des Recovery mittels CANCEL
- DBA muß wissen, wann Abbruch erfolgen soll !
- startup mount
  connect internal
  recover database ...
  cancel
  alter database open resetlogs

Incomplete / Point-in-Time

- Verlust Online Redologfile, Controlfile
- Immer dann durchführen, wenn Zeitpunkt des Crashes genau bekannt ist.
- Recovery wird bis kurz vor diesen Zeitpunkt durchgeführt
- Datenbank ist geschlossen
- Datenbank befindet sich im ARCHIVELOG Mode.
- Eher selten benutzt.
- startup mount
  connect internal
  recover database
  until *time*

Rollforward und Rollback

Bei einem Recovery werden die archivierten Redologfiles zurückappliziert, dieser Vorgang nennt man Rollforward. Alle Transaktionen werden aufgrund der Redoeinträge rekonstruiert, dabei werden auch uncommited Transaktionen ausgeführt. In einer zweiten Phase werden die uncommited Transaktionen mit einem Rollback zurückgesetzt, so dass sich die Datenbank nach einem Recovery wieder in einem konsistenten Zustand befindet.



Phase 1

Das verlorene Datenbankfile wird zurückkopiert. Jeder Datenfileheader enthält eine tiefere Logsequence Nummer (12) als die aktuelle Logsequence, die im Controlfile vermerkt ist (31).

Phase 2

Nun werden der Reihe nach sämtliche archivierten Redologfiles zurückappliziert (12 ... 30), dieser Vorgang heißt Rollforward. Vorhandene abgeschlossene Transaktionen in anderen Datenbankfiles werden nicht neu generiert, es werden nur Transaktionen rekonstruiert, die verloren gingen, dadurch ist ein Recoveryvorgang in der Regel sehr rasch.

Phase 3

Sobald alle Redologfiles appliziert sind, werden die uncommited Transaktionen mit einem Rollback aus den Rollbacksegmenten zurückgesetzt. Wenn dieser Vorgang abgeschlossen ist haben alle Datenbankheader wieder die gleich aktuelle Logsequence Nummer (31).

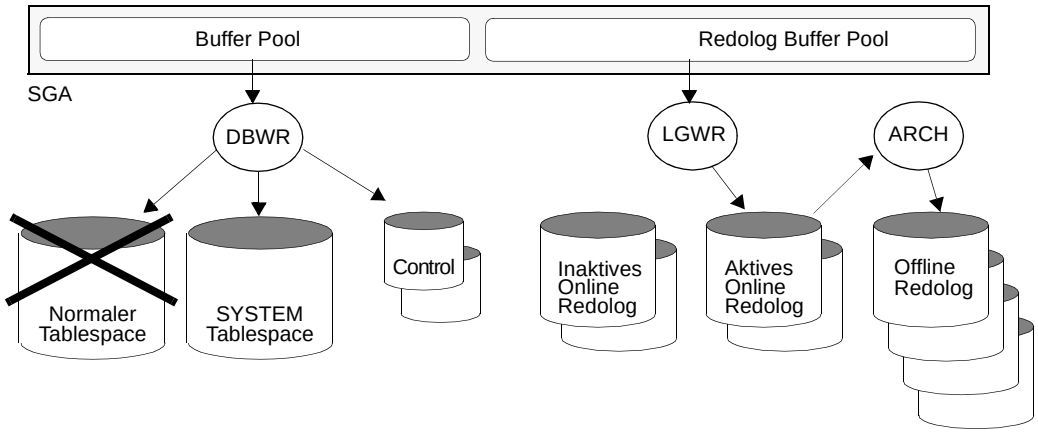
Beim complete Media Recovery werden alle Offline Redologfiles zurückappliziert, beim uncomplete Media Recovery stoppt man das Recovery früher, beispielsweise bei Logsequence 16. Damit kann auf einen älteren Zustand «gefahren» werden.

Complete Media Recovery

Beim Complete Media Recovery können alle commited Transaktionen wiedererstellt werden. Auf den uncommited Transaktionen werden während dem Recovery ein Rollback durchgeführt.

Verlust eines normalen Tablespaces

Die Datenbank wird im ARCHIVELOG Mode betrieben, es ist ein RECOVER TABLESPACE durchzuführen. Es müssen nur die DB-Files, welche zum verlorenen Tablespace gehören zurückgeladen werden.
Ein Online Complete Media Recovery kann benutzt werden wenn ein oder mehrere Datenfiles eines Tablespaces verloren gehen, die keine Rollbacksegmente enthalten. **Der SYSTEM Tablespace kan auf diese Art nicht rekonstruiert werden**.



Die Datenbank muss nicht gestoppt werden, die Transaktionen laufen auf den nicht beteiligten Tablespaces weiter.

1. Datenbank braucht **nicht** gestoppt zu werden

```
svrmgrl
connect internal
```

2. Tablespace offline nehmen

```
alter tablespace users offline immediate;
```

   Wenn dies nicht mehr möglich ist alle betroffenen Datenfiles des Tablespace offline stellen. Die Option «immediate» schreibt keine Checkpoints.

```
alter database datafile 'xxxx.dbf' offline immediate;
```

3. Verlorene Datenfiles und Offline Redologfiles ab Online Backup zurückkopieren mittels UNIX tar, cp. Ebenso alle Archivefiles bereithalten.

4. Recovery starten

```
recover tablespace users;
```

   Nun alle Offline Redologfiles angeben oder Option AUTO

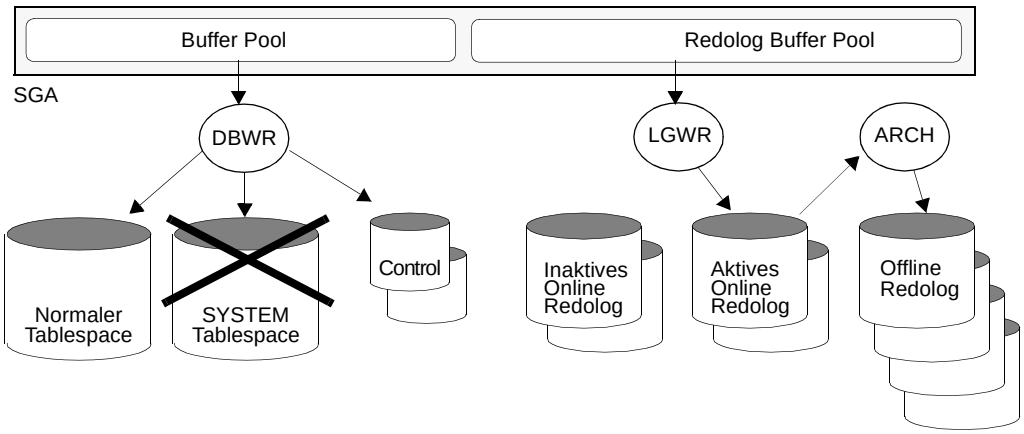
```
auto
```

5. Tablespace online schalten

```
alter tablespace users online;
```

Verlust des SYSTEM oder ROLLBACK Segment Tablespaces

Die Datenbank muss in diesem Fall mit SHUTDOWN ABORT gestoppt werden, es müssen sämtliche DB-Files zurückgeladen werden, es reicht nicht nur die DB-Files des SYSTEM bzw ROLLBACK Tablespaces zurückzuladen. **Beachte, dass das gesicherte Controlfile NICHT zurückgeladen werden darf.** Ein Offline Complete Media Recovery kann immer dann benutzt werden wenn ein oder mehrere Datenfiles eines Tablespaces verloren gehen, sowie wenn der Rollbacksegment- oder SYSTEM Tablespace verloren geht.



1. Die Datenbank muss gestoppt werden !

```
svrmgrl
connect internal
shutdown abort
```

2. Instance mit **bestehenden** Controlfiles starten

```
startup mount;
```

3. Verlorene Datenfiles des SYSTEM- und Rollback Tablespace sowie die Offline Redologfiles ab Online Backup zurückkopieren (tar, cp ...). **Die Controlfiles NICHT zurückladen**.

4. Recovery starten

```
recover database
```

5. Nun alle Offline Redologfiles angeben oder Option AUTO

```
auto
```

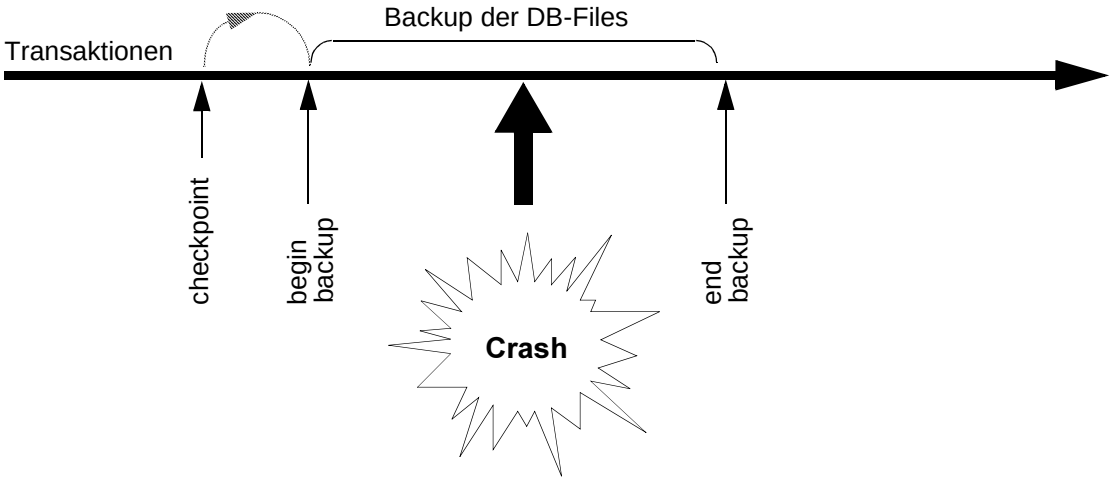
6. Datenbank wieder online nehmen

```
alter database open;
```

Damit ist die Datenbank wieder konsistent.

Crash während Online Backup

Der Online Backup wird bekanntlich mit BEGIN BACKUP gestartet. Nach dem Backup muss die Datenbank zwigend wieder in den END BACKUP Zustand versetzt werden. Was passiert jedoch wenn während dem Online Backup ein Crash erfolgt oder die DB mit shutdown abort heruntergefahren wird. Beim nächsten Startup meldet Oracle, dass die betroffenden Files noch im BEGIN BACKUP Mode sind. Mit dem Kommando alter database datafile 'dbfile(s)' end backup; wird manuell auf END BACKUP geschaltet und der Backup muss wiederholt werden.



1. Beim Startup meldet Oracle

```
connect internal

ORA-01113: Fur Datei '5' ist Datentrager-Recovery notwendig
ORA-01110: Datendatei 5: '/opt/oracle/oradata/TOL2/tab/TOL2_tab1.dbf'
```

2. Kontrolle welche DB-Files im BEGIN Backup Mode sind

```
select substr(a.name,1,30) "DbFile",
       a.file# "File#",
       b.status "Status"
from v$datafile a, v$backup b
where a.file# = b.file#
order by a.file#;

DbFile                         File#      Status
------------------------------ ---------- ------------------
/opt/oracle/oradata/TOL2/sys/T          1 NOT ACTIVE
/opt/oracle/oradata/TOL2/rbs/T          2 NOT ACTIVE
/opt/oracle/oradata/TOL2/tmp/T          3 NOT ACTIVE
/opt/oracle/oradata/TOL2/usr/T          4 NOT ACTIVE
/opt/oracle/oradata/TOL2/tab/T          5 ACTIVE
/opt/oracle/oradata/TOL2/cdr/T          6 NOT ACTIVE
/opt/oracle/oradata/TOL2/cre/T          7 NOT ACTIVE
/opt/oracle/oradata/TOL2/idx/T          8 NOT ACTIVE
/opt/oracle/oradata/TOL2/tab/T          9 ACTIVE
```

3. Die beiden Files des TAB Tablespace auf END BACKUP schalten.

```
alter database datafile '/opt/oracle/oradata/TOL2/tab/
                         TOL2_tab1.dbf' end backup;
alter database datafile '/opt/oracle/oradata/TOL2/tab/
                         TOL2_tab2.dbf' end backup;
```

4. Datenbank öffnen

```
alter database open;
```

Incomplete Media Recovery

In speziellen Fällen muss man einen Recoveryvorgang manuell an einem definierten Punkt abbrechen und die Datenbank so auf einen älteren Zustand setzen. Dies ist der Fall, wenn man alle Online Redologfiles verliert, was bei gespiegelten Redos eigentlich nie vorkommen sollte. Man unterscheidet mehrere Arten wie der Recoveryvorgang abgebrochen werden kann:

Cancel Based

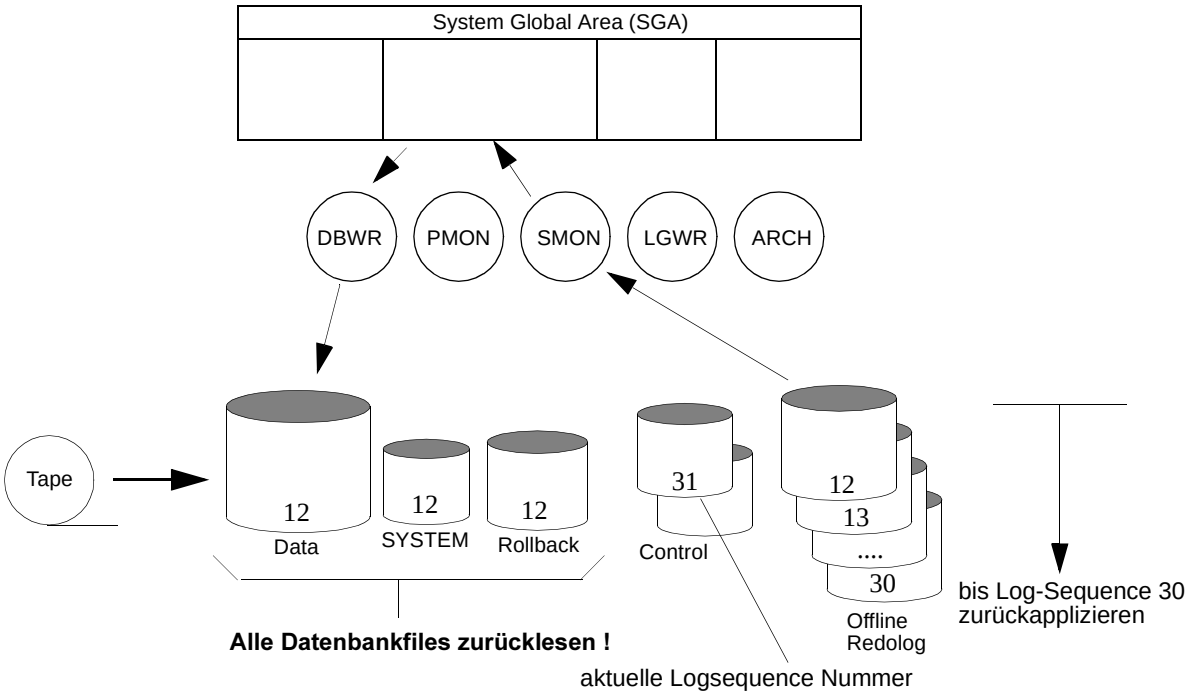
Beim Cancel Based Recovery wird das Recovery an einem definierten Punkt mittels RECOVER CANCEL abgebrochen. Anschließend wird die gesamte Datenbank auf einen älteren Zustand synchronisiert mittels ALTER DATABASE OPEN RESETLOGS.

Time Based

Beim Time Based Recovery wird das Recovery zu einem definierten Zeitpunkt abgebrochen. Diese Art wird vorallem angewendet wenn auf einen älteren Zustand «zurückgefahren» werden soll. Bedingung ist, dass man den Zeitpunkt kennt.

Bedingungen für Incomplete Recovery

Alle Datenfiles müssen vom Backup zurückgeladen werden !
Da man auf einen älteren Zustand zurückkehrt müssen sämtliche Datenbankfiles ab einem definierten Online Backup auf den Abbruchpunkt gefahren werden. Die aktuellen Datenbankfiles sind wertlos, da sie einen jüngeren Zustand aufweisen als der Abbruchpunkt. Man sieht, dass man in diesem Fall mit Sicherheit Transaktionen verliert, bzw. verlieren will.



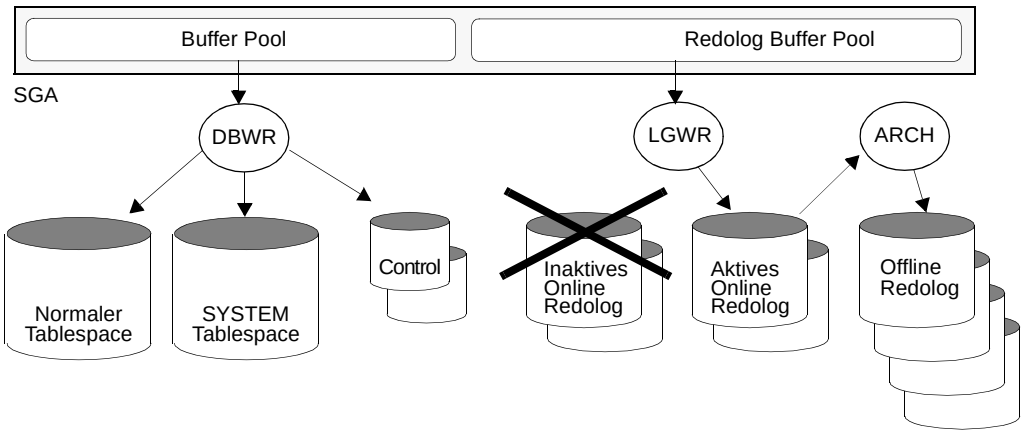
Im obigen Beispiel sind alle Transaktionen des Redologfiles mit der Sequencenummer = 31 definitiv verloren. Man führt nun ein Cancel Based Recovery bis und mit Sequencenummer = 30 durch. Die Sequencenummer im Controlfile wird mittels ALTER DATABASE OPEN RESETLOGS korrigiert.

Cancel Based Media Recovery

Beim Cancel Based Recovery wird der Recovery Vorgang bei einem definierten Punkt mit CANCEL abgebrochen. Bei einem Verlust aller current Online Redologfiles muss immer ein Cancel Based Media Recovery durchgeführt werden. Verliert man die inaktive bereits archivierte Redolog Group so muss kein Recovery durchgeführt werden, es genügt die Redolog Group neu zu kreieren. Man muss sich also zuerst vergewissern welcher Zustand vorliegt, dazu dienen Informationen im Controlfile die abgefragt werden können (Tabelle v$log).

Verlust der inaktiven Redologfiles

Die inaktiven Redologfiles werden vom Logwriter Prozess *lgwr* bis zum nächsten Logswitch nicht beschrieben.



Die Datenbank stürzt beim nächsten Logswitch sofort ab, die Rekonstruktion der Redolog Group muss im NOARCHIVELOG Mode durchgeführt werden. **Es werden keine gesicherten Datenbankfiles zurückkopiert.**

1. Datenbank stoppen

```
svrmgrl
connect internal
shutdown abort
```

2. Instance mit Controlfiles starten

```
startup mount;
```

3. Status der Logfiles kontrollieren (aus Controlfile)

```
SELECT group#, members, status, archived FROM v$log;

    GROUP#    MEMBERS STATUS   ARC
---------- ---------- -------- ---
         1          2 CURRENT  NO    <-- diese existiert noch
         2          2 INACTIVE YES   <-- diese ist verloren !
```

4. Datenbank in NOARCHIVELOG Mode setzten

```
alter database noarchivelog;
```

5. Neue Redolog Group anlegen.

```
alter database add logfile
  group 3
  (
    '/oracle7/kurs/log_group3_1B.dbf',
    '/oracle7/kurs/log_group3_2B.dbf'
  ) size 15M;
```

6. Alte, verlorene Redolog Group löschen

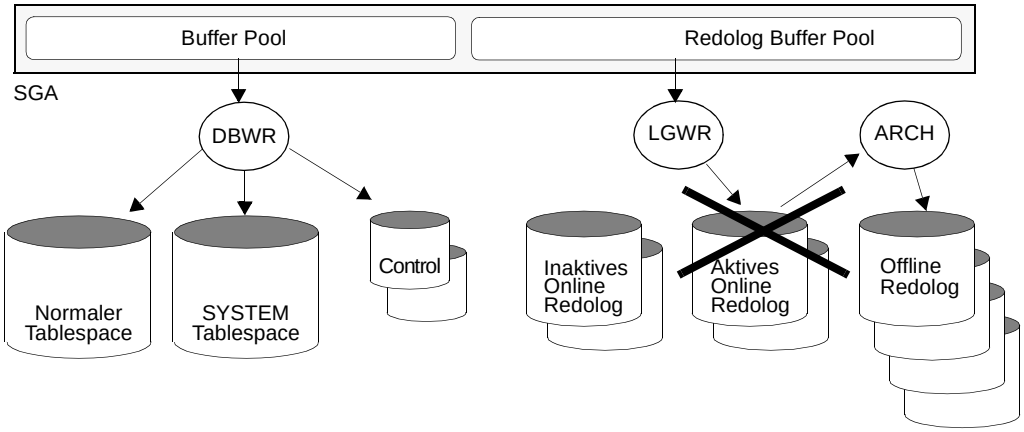
```
alter database drop logfile group 2;
```

7. Datenbank wieder in ARCHIVELOG Mode setzen

```
alter database archivelog;
```

Verlust der aktiven Redologfiles

Die inaktiven Redologfiles werden vom Logwriter Prozess *lgwr* bis zum nächsten Logswitch nicht beschrieben.



Die Datenbank stürzt sofort ab, es muss ein Cancel Based Recovery bis zum letzten Offline Redologfile durchgeführt werden. Die Datenbank muss in diesem Fall mit SHUTDOWN ABORT gestoppt werden, es müssen sämtliche DB-Files zurückgeladen werden, es reicht nicht nur die DB-Files des SYSTEM bzw ROLLBACK Tablespaces zurückzuladen. **Beachte, dass das gesicherte Controlfile NICHT zurückgeladen werden darf.**

1. Datenbank stoppen

```
svrmgrl
connect internal
shutdown abort
```

2. Instance mit bestehenden Controlfiles starten

```
startup mount;
```

3. Status der Logfiles kontrollieren (aus Controlfile)

```
SELECT group#, members, status, archived FROM v$log;

    GROUP#    MEMBERS STATUS   ARC
---------- ---------- -------- ---
         1          2 CURRENT  NO    <-- diese ist verloren !
         2          2 INACTIVE YES   <-- diese existiert noch
```

4. Alle oben gelöschten Datenbankfiles ab Online Backup zurückkopieren (tar, cp ...), diese weisen alle einen älteren Zusatnd auf. Controlfiles nicht zurückladen.
5. Nun erfolgt das Cancel Based Recovery

```
recover database until cancel;
```

   Mit <AUTO> alle Offline Redologfiles applizieren bis Recovery automatisch stoppt, dann ein "Dummy" Recovery durchführen nur um Oracle mitzuteilen, dass ein CANCEL based Recovery durchgeführt wurde.

```
auto
recover database until cancel;
cancel
```

   Ohne <AUTO> bis zum letzten Redologfile des vorhandenen Online Backups zurückapplizieren. Das Ende kann von Oracle nicht selbst erkannt werden.

```
cancel
```

6. Nun wird die Datenbank geöffnet, das verlorene Online Redologfile wird automatisch aufgrund des Controlfile Eintrages wieder erstellt.

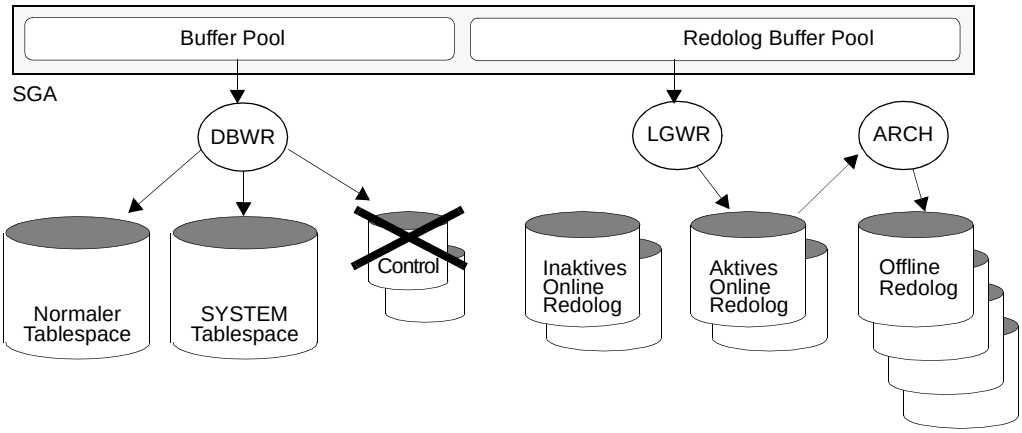
```
alter database open resetlogs;
```

Verlust der Controlfiles, ev auch des SYSTEM und/oder ROLLBACK Segment Tablespaces

Verliert man alle Controlfiles, so bestehen grundsätzlich zwei Möglichkeiten diese wieder zu erstellen.

- Recovery mit Controlfile, das mittels *ALTER DATABASE BACKUP CONTROLFILE ...* gesichert wurde, sofern dieses existiert.
- Controlfile mit dem Kommando *CREATE CONTROLFILE ...* neu erstellen, dazu muss die gesamte Datenbankstruktur bekannt sein.

Die Datenbank muss in diesem Fall mit SHUTDOWN ABORT gestoppt werden, es müssen sämtliche DB-Files zurückgeladen werden, es reicht nicht nur die DB-Files des SYSTEM bzw ROLLBACK Tablespaces zurückzuladen. Beachte, dass auch das gesicherte Controlfile NICHT zurückgeladen werden muss. Nun erfolgt das Cancel Based Recovery, man muss die Option USING BACKUP CONTROLFILE verwenden, damit wird Oracle mitgeteilt, dass Inkonsistenzen bestehen zwischen dem Controlfile und den bestehenden Redologfiles. Nach dem Zurückapplizieren der Offline Redolog Files wird ein "Dummy" Recovery gemacht und die Datenbank mit RESETLOGS geöffnet.



1. Datenbank stoppen

```
svrmgrl
connect internal
shutdown abort
```

2. Zurückladen aller DB-Files **(Controlfiles auch zurückladen !)**
3. Archive Files bereithalten
4. Point-in-Time Recovery der gesamten Datenbank mit zurückgeladenem Controlfile. Mit <AUTO> alle Offline Redologfiles applizieren bis Recovery automatisch stoppt.

```
svrmgrl;
connect internal;
startup mount;
recover database using backup controlfile until cancel;
auto
```

5. "Dummy" Recovery durchführen. Man bricht das Recovery mit <CANCEL> sofort wieder ab, nur um Oracle mitzuteilen, dass ein CANCEL based Recovery durchgeführt wurde.

```
recover database using backup controlfile until cancel;
cancel
```

6. Datenbank mit RESETLOGS öffnen.

```
alter database open resetlogs;
```

Weitere hilfreiche Recovery Tips und Kommands

Datenbank mit fehlendem (normalen) Tablespace starten

Die Datenbank kann ohne diesen fehlenden Tablespace gestartet werden.

1. Datenbank mounten

```
svrmgrl
connect internal
startup mount
```

2. Datenfiles aus Datendictionary und Controlfile entfernen

```
alter database datafile '/opt/oradata/RAB1/tab/RAB1_tab1.dbf'
offline drop;

alter database datafile '/opt/oradata/RAB1/tab/RAB1_tab2.dbf'
offline drop;
```

3. Datenbank öffnen

```
alter database open;
```

4. Tablespace droppen

```
drop tablespace tab including contents;
```

5. Kontrolle der Einträge in DD (dba_data_files) und Controlfile (v$dbfile)

```
select * from dba_data_files order by file_id;
select name,file# from v$dbfile;
```

Archivelog Destination wechseln im laufenden Betrieb

Wird der vorhandene Diskplatz für die Offline Redologfiles auf der Disk knapp, und die Zeit reicht nicht mehr aus um die Offline Redologfiles wegzukopieren, so kann die Archivelog Destination während dem Betrieb gewechselt werden. Mit STOP / START kann das automatische Archivieren durch den ARCH Prozess angehalten werden und wieder gestartet werden

```
alter system archive log stop;
alter system archive log start to '/disk2/data/ARK';
alter system archive log start;
```

Man beachte, dass der String 'ARK' bei der Angabe der neuen Destination kein Directory ist, sondern der Filename-Prefix.

Archivelog Kommandos

Alle momentan nicht archivierten Online Redologfiles manuell archivieren

```
alter system archive log all;
```

Online Redolog Swich erzwingen

```
alter system switch logfile;
```

Status der Redologfiles abfragen (Logsequence)

```
archive log list;
```

Manueller Checkpoint durchführen. Damit werden alle DB Buffer auf die Disk geschrieben.

```
alter system checkpoint;
```

Flush des Shared Pools

Nicht direkt mit dem Recovery zu tun hat der Flush des Shared Pools, zur Vervollständigung der wichtigsten ALTER SYSTEM Kommandos trotzdem an dieser Stelle dokumentiert. Damit werden alle cached Informationen aus dem Shared Pool (Data Dictionary Informationen, Shared PL/SQL und SQL, Stored Proceduren, Functions und Packages) entfernt.

```
alter system flush shared_pool;
```

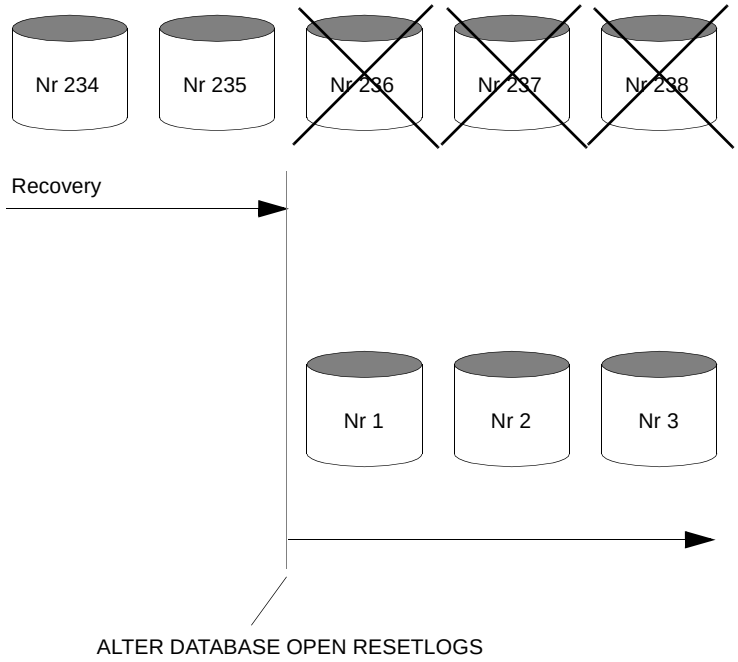
Einfaches Shellscript zum Testen des Online Backups

```
#!/bin/ksh

ORACLE_HOME=/oracle/product/8.0.5; export ORACLE_HOME
ORACLE_SID=RAB1; export ORACLE_SID
TNS_ADMIN=/home/oracle/config/v805; export TNS_ADMIN
NLS_LANG=GERMAN_GERMANY.WE8ISO8859P1; export NLS_LANG
ORA_NLS33=$ORACLE_HOME/ocommon/nls/admin/data; export ORA_NLS33
PATH=/bin:/sbin:/usr/bin:/usr/sbin:$ORACLE_HOME/bin; export PATH

BACKUPLOG=backuplog_${ORACLE_SID}.log

# Mark Tablespaces for the START of Online-Backup

svrmgrl <<-EOF >$BACKUPLOG
connect internal;
alter system switch logfile;
archive log list
alter tablespace rbs begin backup;
alter tablespace system begin backup;
alter tablespace tab begin backup;
alter tablespace temp begin backup;
alter tablespace users begin backup;
alter database backup controlfile to '/data/ctrlRAB1.con' reuse;
alter system switch logfile;
alter system archive log all;
disconnect;
exit;
EOF

# Do the Backup

cp /data/rbs/RAB1_rbs1.dbf /data/rbs/RAB1_rbs1.dbf.backup
cp /data/sys/RAB1_sys1.dbf /data/sys/RAB1_sys1.dbf.backup
cp /data/tab/RAB1_tab1.dbf /data/tab/RAB1_tab1.dbf.backup
cp /data/tmp/RAB1_temp1.dbf /data/tmp/RAB1_temp1.dbf.backup
cp /data/usr/RAB1_users1.dbf /data/usr/RAB1_users1.dbf.backup

# Mark Tablespaces for the END of Online-Backup

svrmgrl <<-EOF >>$BACKUPLOG
connect internal;
alter tablespace rbs end backup;
alter tablespace system end backup;
alter tablespace tab end backup;
alter tablespace temp end backup;
alter tablespace users end backup;
disconnect;
exit;
EOF

exit 0
```

Wichtigste Punkte zum Recovery

alter database open resetlogs

Dies ist ein sehr gefährliches Kommando und muss unbedingt verstanden werden. Wird es falsch angewendet, so kann eine Datenbank irreparabel werden. Es wird nur in Cancel Based Incomplete Recoveries benutzt, also immer dann wenn bei einem Recovery nicht alle Redologfiles zurückappliziert werden. Das Kommando setzt die Logsequence des aktuellen Redologfiles = 1, alle folgenden Offline Redologfiles sind somit wertlos.



Notfall Scenario

Ist eine Datenbank nach einem korrekten Recovery auch nach mehreren Versuchen mit ALTER DATABASE RESETLOGS nicht zu öffnen, so muss der *init.ora* Parameter ***_allow_resetlogs_corruption = true*** gesetzt werden. Ist es dann möglich die Datenabk zu starten. so muss man sofort einen Datenbank Export machen, die Datenbank neu kreieren und das Exportfile importieren.

Offline Datenfiles

Es muss darauf geachtet werden, dass alle Datenfiles der Tabespaces die in ein Recovery einbezogen werden **online** sind. Wird dies missachtet, so können diese Offline Datenbakfiles später nach dem Recovery nicht mehr online genommen werden, da sie einen älteren Zustand aufweisen.

begin und end backup

Beim Online Backup muss zu jedem BEGIN BACKUP auch ein END BAKKUP ausgeführt werden.

DUL Tool

Oracle verfügt über ein Tool mit dem Namen DUL.

DUL stands for Data UnLoader and is a **special tool, available only for Oracle employees**. Configured, DUL work correctly with your database, get the file-numbers and file-names out of the mounted database setup the configuration file to work correctly with Oracle7, unloads the data dictionary unloads the data out of the database in a directory. The provided loader files have a format *<owner>_<table-name>.dat* (data) and *<owner>_<table-name>.ctl* (loader control file).

Wichtigste V$-Views

Wichtigste V$Views Beschreibungen

Informationen bezüglich Transaktionen	
V$THREAD	Ein Thread kennzeichnet eine bestimmte Instance in einem Oracle Parallel Server Umfeld. Da jede Instance eines Parallel Servers ihre eigenen Online Redolog Files hat, kann in dieser View Informationen zu den Log-Group Members, der SCN Nummer des letzten Checkpoints etc eingesehen werden.
V$TRANSACTION	Zeigt die offenen Transaktionen (zB SCN-Nummer der Transaktion)
Informationen bezüglich Redolog Files	
V$LOG	Informationen aus dem Controlfile zu den Redolog Files. Interessant sind lowest SCN und time of first SCN. Man erkennt auch ob ein Redologfile bereits archiviert wurde (ARCHIVED=TRUE) und ob es für ein Instance Recovery noch gebraucht wird (STATUS=ACTIVE).
V$LOGFILE	Namen der Redolog Members
V$LOG_HISTORY	History der archivierten Redolog Files
V$ARCHIVE	Group-Nr, Sequence-Nr, First SCN-Nr der archivierten Redologs
Informationen bezüglich Backup und Recovery	
V$BACKUP	Stati der DB-Files (BEGIN / END Backup). Kann sehr hilfreich sein um herauszufinden, welche DB-Files im BEGIN BACKUP Mode sind.
V$DATAFILE	Informationen zu den DB-Files aus dem Controlfile
V$CONTROLFILE	Namen und Status der Controlfiles
V$DATABASE	Informationen zur DB-Instance wie last SCN checkpointed und last SCN archived sowie Angaben ob DB im Archivelog Mode ist.
V$RECOVERY_LOG	Redologfiles welche für ein Complete Media Recovery benötigt werden
V$RECOVER_FILE	Welche Files brauchen ein Media Recovery ?
V$RECOVERY_STATUS	Informationen zum aktuellen Recovery Prozess