Oracle/ Druckversion

Aus Wikibooks
< Oracle

Oracle

Wikibooks

Inhaltsverzeichnis

Einleitung

Zurück zu " Buchanfang " | Hoch zu " Inhaltsverzeichnis " | Vor zu " Einführung SQL*Plus "

Grundlagen von SQL sollten vorhanden sein. Dieses Buch versteht sich mehr als Nachschlagewerk und Einführung auf einem sehr einfachen Level. Es ist auch ideal für Umsteiger anderer Programmiersprachen weil hier versucht wird, auf einfachste Art und Weise, das Datenbanksystem der Firma Oracle etwas näher zu bringen.

SQL*PLUS

Zurück zu " Einleitung " | Hoch zu " Inhaltsverzeichnis " | Vor zu " Einführung Oracle SQL Developer "

Aufruf-Möglichkeiten

iSQL

iSQL ist eine Möglichkeit, mit dem Datenbankserver Kontakt aufzunehmen alleine über einen beliebigen Web-Browser. Es muss keine Software auf dem Client installiert werden. Diese Möglichkeit gibt es jedoch nur bei der Version 10g. Schon bei der Version 11g gibt es diese Möglichkeit nicht mehr.

Aufruf bei Oracle 10g über die URL: **http://<host>:5560/isqlplus** Dabei muss anstelle von <host> der Computername angegeben werden. Den Computernamen eines Servers, an dem man gerade angemeldet ist, kann man bei Windows herausfinden durch:

Start / Einstellungen / Systemsteuerung / System

In Reiter Netzwerkidentifikation findet man den Computernamen angegeben.

Über den Web-Browser kann man dadurch auf alle Datenbank-Server zugreifen, deren Servername man kennt und bei denen eine Anmeldung über iSQL und den Port 5560 nicht explizit deaktiviert ist.

Natürlich ist eine Identifikation an diesem Server mit einer Oracle-User-ID und Passwort auch noch erforderlich, aber viele Datenbanken haben ja den User SCOTT mit dem Passwort TIGER...

Über diesen Aufruf kann man sich nicht als SYSDBA oder SYSOPER anmelden.

iSQL/dba

Um administrative Arbeiten auszuführen, die eine Anmeldung AS SYSDBA oder AS SYSOPER erfordern, kann die folgende URL benutzt werden: **http://<host>:5560/isqlplus/dba**

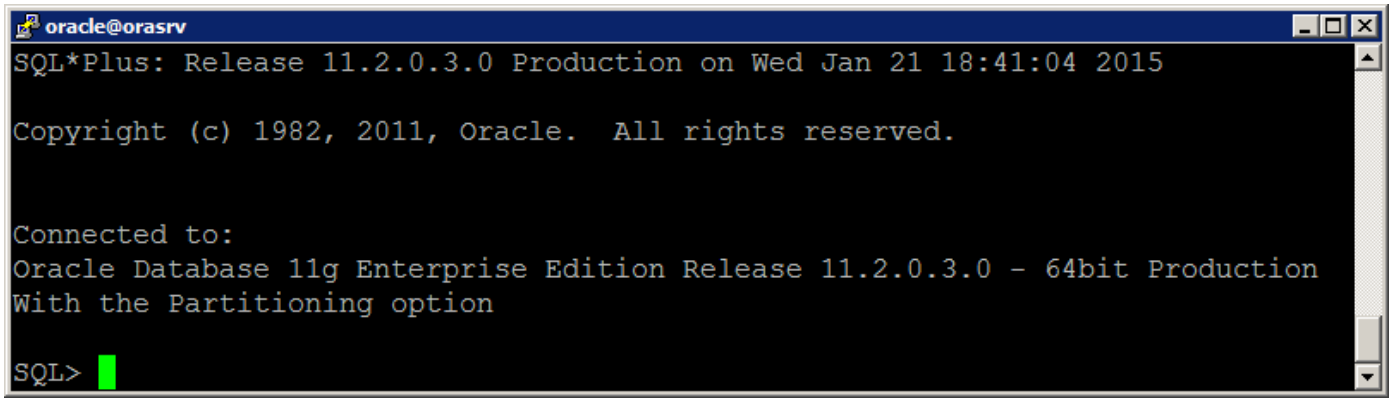
Nach einem erfoglreichen Connect mit einem User in der SYSDBA-Rolle kann die Datenbank z.B. heruntergefahren werden oder neu gestartet werden.

SQL*Plus als Windows-Tool

Das SQL*Plus-Worksheet war lange Zeit die im Oracle-Lieferumfang gehörende Standard-Client-Software, um SQL-Befehle ausführen zu lassen. Seit der Version 11g gibt es diese Möglichkeit nicht mehr. An die Stelle dessen tritt nun der Oracle SQL Developer.

Aufrufmöglichkeiten:

- Als Befehl in einem CMD-Fenster: SQLPLUSW
- Start / Ausführen / SQLPLUSW
- Start / Programme / Oracle / Anwendungsentwicklung / SQL Plus
- Oder C:\oraclexe\app\oracle\product\11.2.0\server\bin\sqlplus.exe



SQL*Plus als Command-Line-Tool

Als Befehl in einem CMD-Fenster: SQLPLUS. Dadurch wird die Verwendung von SQL*Plus per Shell-/Batchskript ermöglicht. Diese Variante gab es schon immer und sie ist auch in der aktuellen Version (11g) noch vorhanden.

Als Befehlsparameter kann man einen Connect-String angeben. Angenommen, man möchte eine Anmeldung als User SCOTT (mit dem Passwort TIGER) an einer lokalen Datenbank vornehmen, dann sind folgende Aufrufe möglich:

```
SQLPLUS SCOTT/TIGER
```

```
SQLPLUS SCOTT
```

Wenn man das Passwort nicht direkt angibt, wird man anschließend aufgefordert, das Passwort einzugeben. Der Vorteil dabei ist, dass das eingegebene Passwort nicht auf dem Bildschirm sichtbar ist. Da wir in diesem Beispiel den Datenbank-Namen nicht angegeben haben, verwendet SQL*Plus den in der Registry angegebenen Datenbank-Namen.

Aufbau zu einer entfernten Datenbank d.h. zu einem Datenbank-Server, der über das Netzwerk zu erreichen ist und der in der lokal gespeicherten Datei TNSNAMES.ORA eingetragen ist. Der Name der Datenbank ist z.B. ORCL. Folgende Aufrufe sind möglich:

```
SQLPLUS SCOTT/TIGER@ORCL
```

```
SQLPLUS SCOTT@ORCL
```

Man kann auch SQL*Plus starten ohne ein Login auszuführen:

```
SQLPLUS /NOLOG
```

Danach ist SQL*Plus gestartet, aber es besteht noch keine Verbindung zu einer Datenbank. In diesem Zustand kann man lediglich die SQL*Plus-Umgebungsvariablen abfragen oder ändern und eben einen Connect-Befehl anweisen z.B.:

```
CONNECT SCOTT/TIGER@ORCL
INSERT ...;
COMMIT;
DISCONNECT
EXIT
```

Das gibt einem die Möglichkeit, mehrere SQL-Befehle in einer Datei als Skript zu speichern und den Connect-Befehl ebenfalls im Skript anzugeben. Wenn das Skript dann noch mit dem Befehl disconnect endet, dann ist es eine in sich abgeschlossene Einheit, die eine Verbindung zu einer Datenbank aufbaut, eine Verarbeitung ausführt und die Verbindung wieder beendet. Wenn man beim Aufruf von SQL*Plus gleich schon das Passwort mitangibt, dann hat das den Nachteil, dass bei MS-Windows das Passwort im Window-Header sichtbar ist.

SQL*Plus-Session beenden

Alle SQL*Plus-Varianten können mit dem Befehl EXIT oder QUIT wieder beendet werden.

Sowohl das Beenden über EXIT oder QUIT, als auch das sofortige schliessen (in Windows) über *Fenster schliessen* / x-Button führt implizit ein **commit** auf dem Datenbank-Server aus.

Interaktives Arbeiten

Allgemeine Formatierung ändern

Nach dem Aufruf einer SQL*Plus-Session ist es in vielen Fällen erforderlich, die Formatierung der Server-Ausgaben zu ändern.

Die Formatierung wird im SQL*Plus in Systemvariablen (auf dem Client-Computer) gespeichert.

Die Systemvariable **LINESIZE** gibt an, wie viele Zeichen in einer Zeile maximal ausgegeben werden sollen. Wenn die Ausgabezeile länger ist, dann wird der Rest in der nächsten Zeile ausgegeben. Der Default-Wert dieser Systemvariablen ist 80.

Die Systemvariable **PAGESIZE** gibt an, nach wievielen Ausgabezeilen die Überschrift erneut ausgegeben werden soll. Der Default-Wert dieser Systemvariablen ist 14.

Die Inhalte der SQL*Plus-Systemvariablen kann man mit dem Befehl SHOW ausgeben lassen:

```
SQL> show linesize
linesize 80
SQL> show pagesize
pagesize 14
SQL>
```

Wenn viele Tabellenspalten ausgegeben werden sollen, dann können die vielen Zeilenumbrüche störend sein. Auch die wiederholte Ausgabe der Spaltenüberschriften kann störend wirken.

Die SQL*Plus-Systemvariablen kann man mit dem SET-Befehl ändern. Die Änderung ist jedoch nur für die aktuelle Session aktiv:

```
SQL> set linesize 9000
SQL> set pagesize 9000
```

Es gibt noch wesentlich mehr Systemvariablen, mit denen man die Formatierung beeinflussen kann. Eine vollständige Beschreibung findet man im SQL*Plus-Manual. (Siehe Literatur-Liste)

Formatierung bestimmter Spalten ändern

Wenn Daten durch den SELECT-Befehl ausgegeben werden, dann wird für jede Spalte so viel Platz gelassen, dass auch ein Wert mit maximaler Länge ausgegeben werden kann.

Beispiel:

```
select empno, ename, sal from emp;

     EMPNO ENAME             SAL
---------- ---------- ----------
      7369 SMITH             800
      7499 ALLEN            1600
      7521 WARD             1250
      7566 JONES            2975
      7654 MARTIN           1250
      7698 BLAKE            2850
      ...
```

Wenn man die Formatierung ändern will, dann kann man mit dem Befehl COLUMN FORMAT das Ausgabeformat ändern. Die Befehle können auch abgekürzt werden als COL FOR.

Beispiel: Die Spalte ENAME soll in maximal 6 Zeichen ausgegeben werden und die Spalte SAL soll mit Tausenderpunkten und zwei Nachkommastellen ausgegeben werden:

```
column ename format a6
column sal format 99,999.99
```

```
select empno, ename, sal from emp;

     EMPNO ENAME         SAL
---------- ------ ----------
      7369 SMITH      800.00
      7499 ALLEN    1,600.00
      7521 WARD     1,250.00
      7566 JONES    2,975.00
      7654 MARTIN   1,250.00
      7698 BLAKE    2,850.00
      ...
```

Wenn man der Wert einer Spalte in dem vorgegebenen Format nicht ausgegeben werden kann (wenn der Platz nicht ausreicht), dann wird automatisch ein Zeilenumbruch ausgeführt.

Beispiel:

```
column ename format a5
select empno, ename from emp;

     EMPNO ENAME
---------- -----
      7369 SMITH
      7499 ALLEN
      7521 WARD
      7566 JONES
      7654 MARTI
           N

      7698 BLAKE
      ...
```

und bei numerischen Werten wird bei Platzmangel der vorhandene Platz mit dem Hash-Zeichen aufgefüllt.

```
column sal format 999.99
select empno, sal from emp;

     EMPNO     SAL
---------- -------
      7369  800.00
      7499 #######
      7521 #######
      7566 #######
      7654 #######
      7698 #######
      ...
```

Eine Formatierung kann wieder aufgehoben werden mit dem Schlüsselwort CLEAR.

Login-Skript erstellen

Um die Änderungen an den Systemvariablen des SQL*Plus-Client nicht bei jedem Start einer Session erneut setzen zu müssen, können die Änderungen in einem Login-Skript eingetragen werden, das bei jedem Aufbau einer neuen Session automatisch ausgeführt wird. Bei einer Standard-Installation der Oracle-Version 10g unter Windows befindet sich das Login-Skript in dem folgenden Verzeichnis:

```
C:\oracle\product\10.2.0\db_1\sqlplus\admin\glogin.sql
```

Bei der Version 11g liegt es hier:

```
C:\app\<User-ID>\product\11.1.0\db_1\sqlplus\admin\glogin.sql
```

Sollte kein schreibender Zugriff auf die oben angegebene Datei möglich sein, kann man in dem aktuellen Verzeichnis oder in dem per Umgebungsvariable gesetzten Verzeichnis `$ORACLE_PATH` eine Datei namens `login.sql` erstellen, die durch SQL*Plus beim Start **nach** der `glogin.sql` eingelesen wird. Eventuell dopppelte Angaben überschreiben demnach die Werte, die beim Einlesen der `glogin.sql` gesetzt worden sind.

Editor

Im SQLPLUS und im SQLPLUSW kann ein Editor aufgerufen werden, um den letzten SQL-Befehl bzw. den letzten

PL/SQL-Block zu verändern.

Der Aufruf erfolgt mit dem Befehl **EDIT** (Abkürzung: ED). Nach dem Editieren des Befehls muss die Seite gespeichert werden und der Editor beendet werden. Dann kann der veränderte Befehl mit / oder mit **RUN** (Abkürzung: R) ausgeführt werden.

In der Systemvariablen _EDITOR ist der Name des Editors eingetragen, der bei **EDIT** aufgerufen wird.

Man kann auch einen anderen Editor angeben, z.B. den VI-Editor:

```
DEFINE _EDITOR = vi
```

Readline Funktionalität

Will man dieselbe Funktionalität wie in einer Shell haben (alle vorhergehenden Befehle, durchsuchen der Befehle etc.) hilft das Programm **rlwrap** (readline wrapper). Das Programm liegt den meisten Distributionen als Paket bei, unter Ubuntu installiert man es mit **apt-get install rlwrap**. Der Aufruf von SQLPLUS erfolgt dann so:

```
rlwrap sqlplus
```

Befehl / Skript ausführen

Ein SQL-Statement wird mit einem Semikolon oder einer Leerzeile abgeschlossen. Bei einem Abschluss durch ein Semikolon wird der SQL-Befehl auch gleich ausgeführt.

Ein PL/SQL-Block kann aus mehreren SQL-Statements bestehen. Er wird erst ausgeführt, wenn der Block abgeschlossen wurde. Das geschieht durch eine Zeile, in der nur ein einzelner Punkt oder ein Slash eingetragen ist. Bei einem Slash wird der PL/SQL-Block auch gleich ausgeführt.

Mit dem Befehl RUN (Abkürzung: R) oder einem Slash (eine Eingabezeile, in der nur ein Slash eingegeben wird) wird der letzte SQL-Befehl bzw. der letzte PL/SQL-Block erneut ausgeführt. Der Unterschied ist nur, dass bei RUN der SQL-Befehl bzw. der PL/SQL-Block, der ausgeführt werden soll, noch einmal ausgegeben wird, während bei dem Slash nur ausgeführt wird.

Mit dem Befehl LIST (Abkürzung: L) kann man den letzten SQL-Befehl bzw. den letzten PL/SQL-Block ausgeben ohne ihn auszuführen.

Mit dem Befehl SAVE <Dateiname> (Abkürzung: SAV <Dateiname>) kann man den letzten SQL-Befehl bzw. den letzten PL/SQL-Block in die Datei <Dateiname> ausgeben. Wenn man dabei keine Extension angibt, dann wird automatisch .sql ergänzt.

Mit dem Befehl GET <Dateiname> kann man SQL-Statements aus einer Datei in den Buffer laden. Die gelesenen Statements werden dabei nicht ausgeführt.

Wenn man SQL-Statements aus einer Datei lesen und gleichzeitig ausführen will, dann kann man GET und danach RUN verwenden oder START <Dateiname> (Abkürzung: @<Dateiname>) angeben. Der Unterschied ist nur, dass GET und RUN nur ein einziges SQL-Statement oder einen einzigen PL/SQL-Block in den Buffer übertragen können, während mit START auch beliebig viele SQL-Statements und PL/SQL-Blöcke aus der Datei ausgeführt werden können.

In der Datei, die mit START oder @ ausgeführt wird, können weitere Skripte eingebunden werden, indem diese ebenfalls mit START oder @ benannt werden. Die weiteren Skripte werden bei einem Aufruf mit @ <Dateiname> in dem Verzeichnis gesucht, aus dem SQLPLUS gestartet wurde. Man kann ein weiteres Sktipt auch mit @@ <Dateiname> aufrufen. Der Unterschied besteht darin, dass <Dateiname> in dem Verzeichnis gesucht wird, in dem das aufrufende Skript liegt.

Tabellenstruktur ausgeben

Mit dem DESCRIBE-Befehl (Abkürzung: DESC) kann die Struktur einer Tabelle, Prozedur oder Funktion ausgegeben werden.

Beispiel:

```
SQL> describe emp
```

```
Name                 Null?    Typ
-------------------- -------------------------------
EMPNO                NOT NULL NUMBER(4)
ENAME                         VARCHAR2(10)
JOB                           VARCHAR2(9)
MGR                           NUMBER(4)
HIREDATE                      DATE
SAL                           NUMBER(7,2)
COMM                          NUMBER(7,2)
DEPTNO                        NUMBER(2)
```

Sollten nach den Spaltennamen zuviele Leerstellen kommen, ist die Variable LINESIZE zu hoch eingestellt, dann mit `set linesize 80` diese kleiner setzen, und den describe Befehl wiederholen.

Die selben Informationen erhält man auch mit einem Zugriff auf die Katalog-Tabelle USER_TAB_COLUMNS (ohne störende 1000 Leerzeichen dazwischen). Wie bei allen Zugriffen auf Katalog-Tabellen muss man darauf achten, dass der Tabellenname in Grossbuchstaben angegeben wird:

```
select *
from user_tab_columns
where table_name = 'EMP'
```

SQL*Plus-Austausch-Variablen - erstes Beispiel

Wenn man eine Austausch-Variable das erste Mal mit & verwendet, dann wird bei jeder Skriptausführung eine Eingabeaufforderung ausgeführt.

```
SQL> select empno, ename from emp where empno = &x;
Geben Sie einen Wert für x ein: 7369
alt   1: select empno, ename from emp where empno = &x
neu   1: select empno, ename from emp where empno = 7369

     EMPNO ENAME
---------- ----------
      7369 SMITH
```

Wenn das Skript ein zweites Mal ausgeführt wird, dann muss erneut ein Wert für die Austauschvariable eingegeben werden, allerdings kann dann ein anderer Wert eingegeben werden.

```
SQL> /
Geben Sie einen Wert für x ein: 7566
alt   1: select empno, ename from emp where empno = &x
neu   1: select empno, ename from emp where empno = 7566

     EMPNO ENAME
---------- ----------
      7566 JONES
```

Unterschied zwischen & und &&

Wenn die Austausch-Variable das erste Mal mit && verwendet wurde, dann wird nur bei diesem ersten Auftreten eine Eingabe verlangt. Wenn man die Ausführung mit / wiederholt, dann erfolgt keine Eingabeaufforderung mehr.

```
SQL> select empno, ename from emp where empno = &&y;
Geben Sie einen Wert für y ein: 7698
alt   1: select empno, ename from emp where empno = &&y
neu   1: select empno, ename from emp where empno = 7698

     EMPNO ENAME
---------- ----------
      7698 BLAKE

SQL> /
alt   1: select empno, ename from emp where empno = &&y
neu   1: select empno, ename from emp where empno = 7698

     EMPNO ENAME
---------- ----------
      7698 BLAKE
```

Eingabeaufforderung mit PROMPT und ACCEPT

Die Eingabe einer Austausch-Variablen kann explizit durch den Befehl ACCEPT (Abkürzung: ACC) angewiesen werden, ohne dass auf dem Datenbank-Server ein SQL-Befehl ausgeführt werden muss.

```
SQL> accept x
7369
SQL> select empno, ename from emp where empno = &x;
alt   1: select empno, ename from emp where empno = &x
neu   1: select empno, ename from emp where empno = 7369

     EMPNO ENAME
---------- ----------
      7369 SMITH
```

Mit ACCEPT PROMPT kann noch ein Kommentar zu der Eingabeaufforderung definiert werden.

```
SQL> accept x prompt 'Bitte eine Personennummer eingeben '
Bitte eine Personennummer eingeben 7698
SQL> select empno, ename from emp where empno = &x;
alt   1: select empno, ename from emp where empno = &x
neu   1: select empno, ename from emp where empno = 7698

     EMPNO ENAME
---------- ----------
      7698 BLAKE
```

PROMPT und ACCEPT eignen sich vor allem für die Ausführung von SQL-Skripten, die in einer Datei abgelegt sind.

Beispiel: In der Datei s.sql stehen das folgende Skript:

```
prompt jetzt geht es los
accept x prompt 'Bitte eine Personennummer eingeben '
prompt Danke für die Eingabe
select empno, ename from emp where empno = &x;
```

Das Skript wird ausgeführt:

```
SQL> @s
jetzt geht es los
Bitte eine Personennummer eingeben 7698
Danke für die Eingabe
alt   1: select empno, ename from emp where empno = &x
neu   1: select empno, ename from emp where empno = 7698

     EMPNO ENAME
---------- ----------
      7698 BLAKE
```

Deklaration mit DEFINE

Austausch-Variable können auch mit DEFINE (Abkürzung: DEF) bekannt gemacht werden. Wenn sie danach verwendet werden, dann keine Eingabeaufforderung mehr ausgeführt.

```
SQL> define s=2990
SQL> select ename, sal from emp where sal > &s;
alt   1: select ename, sal from emp where sal > &s
neu   1: select ename, sal from emp where sal > 2990

ENAME             SAL
---------- ----------
SCOTT            3000
KING             5000
FORD             3000
```

Selbst wenn man den SQL-Befehl erneut ausführt, wird keine Eingabeaufforderung mehr ausgeführt. Der einmal definierte Wert wird wieder verwendet.

```
SQL> /
alt   1: select ename, sal from emp where sal > &s
neu   1: select ename, sal from emp where sal > 2990
```

```
ENAME             SAL
---------- ----------
SCOTT            3000
KING             5000
FORD             3000
```

Eingabeaufforderung für einen String-Wert

Wenn man mit Austausch-Variablen String-Werte verarbeiten will, dann muss man sich grundsätzlich entscheiden, ob in der Variablen der String mit oder ohne Anführungszeichen gespeichert werden soll. Im ersten Fall muss auch der Anwender bei der Eingabeaufforderung den String mit - und um zweiten Fall ohne - Anführungszeichen eingeben.

Beispiel für die Speicherung mit Anführungszeichen:

```
SQL> select empno, ename from emp where ename = &n;
Geben Sie einen Wert für n ein: 'SMITH'
alt   1: select empno, ename from emp where ename = &n
neu   1: select empno, ename from emp where ename = 'SMITH'

     EMPNO ENAME
---------- --------------------------------------------------
      7369 SMITH
```

Beispiel für die Speicherung ohne Anführungszeichen:

```
SQL> select empno, ename from emp where ename = '&n';
Geben Sie einen Wert für n ein: SMITH
alt   1: select empno, ename from emp where ename = '&n'
neu   1: select empno, ename from emp where ename = 'SMITH'

     EMPNO ENAME
---------- --------------------------------------------------
      7369 SMITH
```

Gibt es ein Escape-Zeichen?

Wie kann man denn in einem SQL*Plus-Skript nach einem String suchen, der mit & beginnt ohne dass die nachfolgenden Buchstaben als Austauschvariable interpretiert werden?

Mit **set define** kann das **&** Zeichen geändert werden:

```
>show define
>define "&" (hex 26)
>select * from dual where '1'='&testvariable';
Enter value for testvariable: 2
old   1: select * from dual where '1'='&testvariable'
new   1: select * from dual where '1'='2'

no rows selected

>set define #
>select * from dual where '1'='&testvariable';

no rows selected

>select * from dual where '1'='#testvariable';
Enter value for testvariable: 2
old   1: select * from dual where '1'='#testvariable'
new   1: select * from dual where '1'='2'

no rows selected

>show define
>define "#" (hex 23)
>set define &
```

Mit **set define off** kann die Suche nach Austauschvariablen ausgeschaltet werden. Dann werden alle Zeichen so belassen, wie sie angegeben sind. Das kann dann sinnvoll sein, wenn man in einem Skript Strings mit allen möglichen Sonderzeichen verwenden möchte, ohne dass eines davon als Austauschvariable interpretiert werden soll.

Punkt als Trennzeichen

Wenn man eine Variable ohne Leerzeichen an ein anderes Literal anschließen möchte, dann muss man den Punkt als Trennzeichen verwenden.

Beispiel für numerische Werte

```
SQL> undefine x
SQL> select &x.000 from dual;
Geben Sie einen Wert für x ein: 33
alt   1: select &x.000 from dual
neu   1: select 33000 from dual

     33000
----------
     33000
```

Beispiel für Zeichenketten

```
SQL> undefine x
SQL> select '&x.abc' from dual;
Geben Sie einen Wert für x ein: 12345
alt   1: select '&x.abc' from dual
neu   1: select '12345abc' from dual

'12345AB
--------
12345abc
```

Nutzung von SQL*Plus in Skripten

einfaches Beispiel

Man kann in einer Datei SQL-Befehle und PL/SQL-Blöcke speichern, die häufig ausgeführt werden müssen.

Beispiel: In der Datei ana.sql stehen die folgenden Befehle:

```
analyze TABLE SALGRADE compute statistics;
analyze TABLE BONUS compute statistics;
analyze TABLE EMP compute statistics;
analyze TABLE DEPT compute statistics;
```

Diese Datei kann ausgeführt werden in einer SQL*Plus-Session:

```
SQL> @ana
Tabelle wurde analysiert.
Tabelle wurde analysiert.
Tabelle wurde analysiert.
Tabelle wurde analysiert.
SQL>
```

Wenn das Skript sehr umfangreich ist und man nicht möchte, dass die Meldungen 'Tabelle wurde analysiert.' auf dem Bildschirm ausgegeben werden, dann kann man die Systemvariable FEEDBACK auf OFF setzen. Das Skript ana.sql wird dann wie folgt erweitert:

```
set feedback off
analyze TABLE SALGRADE compute statistics;
analyze TABLE BONUS compute statistics;
analyze TABLE EMP compute statistics;
analyze TABLE DEPT compute statistics;
```

Kommentare

Es gibt drei Möglichkeiten, in einem Skript Kommentare zu verfassen:

```
rem Zeilen, die mit REM beginnen, werden nicht ausgeführt
```

```
rem

/* Mit Slash-Stern wird ein Kommentar begonnen, der über mehrere
Zeilen gehen kann.
Mit Stern-Slash wird der Kommentar beendet.
*/

-- Mit zwei Minusstrichen kann man den Rest einer Zeile
-- als Kommentar definieren.
-- Alles, was vor den Kommentarstrichen steht, wird ausgeführt

select * -- Hier sollen alle Spalten gelesen werden
from emp -- Diese Tabelle liegt im Schema scott
;
```

Parameter übergeben

Bind-Variablen

Das kann man in einem SQL*Plus-Fenster ausführen. (Mit dem Programm TOAD geht das leider nicht)

```
variable a number
begin
  select min(id) into :a from auftrag_tabelle;
end;
/
print a
select :a
from dual
;
select *
from auftrag_tabelle
where id = :a
;
```

Whenever-Befehl

Wenn bei der Ausführung der Befehle eines Skripts Fehler auftreten, dann wird normalerweise der fehlerhafte Befehl übersprungen und mit den nachfolgenden Befehlen fortgesetzt. Wenn mann jedoch bei einem Fehler die weitere Verarbeitung abbrechen will, dann kann man an den Anfang des Skripts eine WHENEVER-Anweisung setzen:

```
whenever sqlerror exit 1;
whenever oserror  exit 1;
```

Oserror betrifft Fehler aus dem umgebenden Betriebssystem. Sqlerror betrifft Fehler bei der Ausführung der SQL-Statements vom Oracle-Server.

Wenn bei einem Fehler die Verarbeitung fortgesetzt werden soll, dann kann man das auch explizit anweisen:

```
whenever sqlerror continue
```

Die Whenever-Anweisungen beziehen sich immer auf alle nachfolgenden SQL-Befehle.

Return-Code abfragen

SQL*Plus kann beim EXIT Returncodes übergeben.

```
SQL> help exit
 ...
{EXIT|QUIT} [SUCCESS|FAILURE|WARNING|n|variable|:BindVariable]
  [COMMIT|ROLLBACK]
```

ein SUCCESS liefert als RC 0, FAILURE als RC 1 und WARNING als RC 2. Desweiteren kann ein numerischer Wert (0..255) übergeben werden, der dann im RC steht:

```
sqlplus -s test/test
exit

In der shell:
```

```
-> echo $?
0

sqlplus -s test/test
exit 99

In der shell:
-> echo $?
99


sqlplus -s test/test
variable RTC number;
begin
select 5 into :RTC from dual;
end;
/
exit :RTC

-> echo $?
5
```

Cursor definieren

Mit SQL*Plus können Cursor-Variablen definiert werden, die in einem PL/SQL-Block verwendet werden. In dem nachfolgenden Beispiel wird eine Cursor-Variable mit dem Namen C deklariert, geöffnet und - nach Beendigung des PL/SQL-Blocks - ausgegeben. Die Ausgabe wird durch die Autoprint-Option aktiviert.

```
SQL> variable c refcursor
SQL> set autoprint on
SQL> begin
  2  open :c for
  3  select ename
  4  from scott.emp
  5  where ename like '%E%';
  6  end;
  7  /

PL/SQL-Prozedur erfolgreich abgeschlossen.

ENAME
----------
ALLEN
JONES
BLAKE
TURNER
JAMES
MILLER

6 Zeilen ausgewählt.
```

Cursor-Variablen kann man auch dafür verwenden, um Prozeduren oder Funktionen aufzurufen, die einen Cursor als Parameter erwarten.

Tipps und Tricks

Skript generieren

Ein Administrator muss oft verschiedene Wartungs-Arbeiten für alle Tabellen ausführen, die bestimmte Kriterien erfüllen. So zum Beispiel das Aktualisieren von Statistikdaten.

```
analyze table <name> compute statistics;
```

Dabei kann es sein, dass diese SQL-Befehle für hunderte oder gar tausende Tabellen ausgeführt werden müssen. Natürlich wird ein Administrator solche Befehle nicht für jede Tabelle von Hand eintippen, sondern er könnte sich ein Skript dafür generieren.

Beispiel: Mit dem folgenden Befehl werden Analyze-Befehle für alle Tabellen des Users SCOTT generiert.

```
select 'analyze table '
    || table_name
    || ' compute statistics;'
from user_tables;
```

```
'ANALYZETABLE'||TABLE_NAME||'COMPUTESTATISTICS;'
------------------------------------------------
analyze table SALGRADE compute statistics;
analyze table BONUS compute statistics;
analyze table EMP compute statistics;
analyze table DEPT compute statistics;
```

Nun kann man den Output entweder mit der Maus kopieren und erneut ausführen lassen, oder man läßt den Output gleich in eine Datei mitschreiben. Das kann man mit dem SPOOL-Befehl machen. Das hat den Vorteil, dass man das Skript noch in einem Editor nachbearbeiten kann, bevor man es ausführt. Nach der Ausführung steht das Skript immer noch zur Verfügung und man kann zu einem späteren Zeitpunkt noch einmal nachschauen, welche Befehle ausgeführt wurde.

Wenn man den Output in eine Datei protokolliert, dann wäre es nützlich, die Spaltenüberschriften und die Ergebnis-Meldung zu deaktivieren. Die Befehle dafür sehen dann so aus:

```
SQL> set newpage 0
SQL> set space 0
SQL> set pagesize 0
SQL> set echo off
SQL> set feedback off
SQL> set verify off
SQL> set heading off
SQL> spool x.sql
SQL> set trimspool on
SQL> select 'analyze table '
  2   || table_name || ' compute statistics;'
  3   from user_tables;
SQL> spool off
```

In der Datei x.sql wurde das Skript generiert. Hier muss man evtl. noch die Protokollierung des SPOOL OFF entfernen, danach kann man es ausführen:

```
@x.sql
```

Serveroutput vergrößern

Um die Ausgaben von ausgeführten Packages/Prozeduren auch zu sehen, muss diese Ausgabe auch eingeschaltet werden, und gross genug sein:

```
>show serveroutput
serveroutput OFF

>exec dbms_output.put_line('Das ist ein Test');

PL/SQL procedure successfully completed.
```

So wird eine Prozedur zwar ausgeführt, es wird aber nichts ausgegeben. Um die Ausgabe zu erhalten, muss der Serveroutput eingeschaltet werden:

```
>set serveroutput on
>exec dbms_output.put_line('Das ist ein Test');
Das ist ein Test

PL/SQL procedure successfully completed.
```

Oracle schreibt diese Ausgaben erst mal in einen Buffer, und gibt diesen Buffer erst am Ende aus. Dabei kann es passieren, dass der Fehler **ORA-20000: ORU-10027: buffer overflow, limit of 2000 bytes** auftritt, der besagt, dass dieser Buffer übergelaufen ist. Diesen kann man bis zu einer Grösse von 1000000 vergrössern:

```
>set serveroutput on size 1000001
SP2-0547: size option 1000001 out of range (2000 through 1000000)
>set serveroutput on size 1000000
>show serveroutput
serveroutput ON size 1000000 format WORD_WRAPPED
```

Datenbankserver starten und stoppen

Der Vorteil von SQL*Plus ist, dass man damit auch auf einen Datenbank-Server zugreifen kann, bei dem die Datenbank-

Instanz noch nicht gestartet ist. Das liegt daran, dass das sqlplus Programm auf einem Client-Computer ausgeführt wird. Sqlplus nimmt zunächst Kontakt mit dem Listener auf dem Datenbank-Server auf. Der Listener ist ein Prozess, der in der Regel aktiv bleibt, auch wenn die Datenbank heruntergefahren ist. Der Listener hat die Möglichkeit, ein Hochfahren der Datenbank-Instanz zu bewirken.

Der User SCOTT hat nicht das SYSDBA-Privileg, daher kann er sich an einer inaktiven Instanz nicht anmelden:

```
C:\Oracle_test>sqlplus scott/tiger

SQL*Plus: Release 11.1.0.6.0 - Production on Sa Feb 2 17:04:52 2008

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

ERROR:
ORA-01034: ORACLE not available
ORA-27101: shared memory realm does not exist
Prozess-ID: 0
Session-ID: 0 Seriennummer: 0
```

Jedoch der User SYS hat das SYSDBA-Privileg, daher kann er sich anmelden und die Instanz hochfahren:

```
C:\Oracle_test>sqlplus sys/geheim as sysdba
Bei einer nicht hochgefahrenen Instance angemeldet.

SQL> startup
ORACLE-Instance hochgefahren.

Total System Global Area  313860096 bytes
Fixed Size                  1332892 bytes
Variable Size             226494820 bytes
Database Buffers           79691776 bytes
Redo Buffers                6340608 bytes
Datenbank mounted.
Datenbank geöffnet.
SQL>
```

Das Stoppen und Herunterfahren einer Datenbank-Instanz kann genauso durch SQL*Plus ausgeführt werden - vorausgesetzt, der User hat die SYSDBA-Berechtigung:

```
SQL> shutdown
Datenbank geschlossen.
Datenbank dismounted.
ORACLE-Instance heruntergefahren.
SQL>
```

Das Starten und Stoppen einer Datenbank wird ausführlicher beschrieben im Kapitel Oracle: Datenbank starten.

Literatur

- download.oracle.com SQL*Plus User's Guide and Reference der Oracle Version 11.1 (http://download.oracle.com/docs/cd/B28359_01/server.111/b31189/toc.htm)
- download.oracle.com SQL*Plus Quick Reference der Oracle Version 11.1 (http://download.oracle.com/docs/cd/B28359_01/server.111/b31190/toc.htm)
- www.orafaq.com/wiki/SQL*Plus_FAQ (http://www.orafaq.com/wiki/SQL*Plus_FAQ)

Zurück zu " Einleitung " | Hoch zu " Inhaltsverzeichnis " | Vor zu " Einführung Oracle SQL Developer "

SQL Developer

Zurück zu " SQL*Plus " | Hoch zu " Inhaltsverzeichnis " | Vor zu " Einführung Toad "

Der SQL Developer (http://www.oracle.com/technology/software/products/sql/index.html) von Oracle (zuvor Projekt Raptor) ist ein grafisches Entwicklerwerkzeug (IDE) für alle Operationen welche an einer Oracle Datenbank vorgenommen werden können:

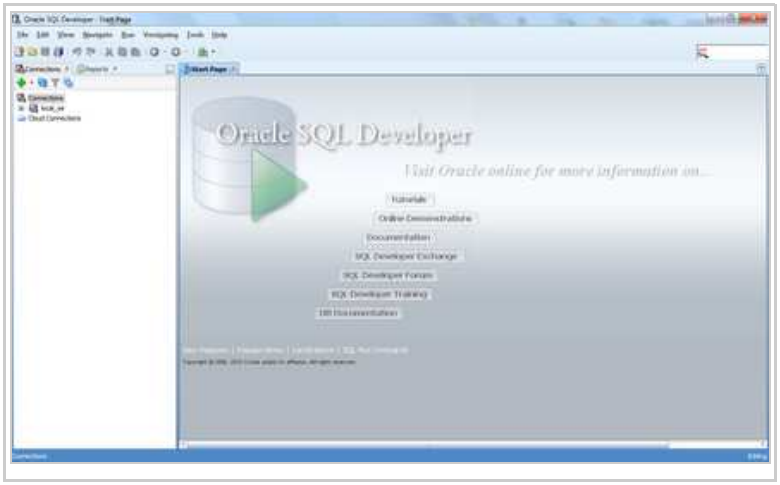
- Anlegen und Bearbeiten von Tabellen und Indices
- Entwickeln und debuggen von PL/SQL Code
- etc.

Er ist komplett in Java geschrieben und verwendet den JDBC Thin-Driver (dh. er benötigt keinen extra installierten Treiber - lediglich ein lauffähiges JDK). Da er kostenlos verwendet werden kann, ist er besonders bei kleineren Firmen sehr beliebt.

Der SQL Developer ersetzt das SQL*Plus-Worksheet, das seit der Version 11g nicht mehr dabei ist.

Beim ersten Start des SQL Developers muss der Pfad für die Java-Version angegeben werden. Bei der Oracle-Version 11g ist immer eine Java-Version mit dabei. Man findet sie im Verzeichnis:

```
C:\app\<User-ID>\product\11.1.0\db_1\jdk\bin\java.exe
```



Toad

 Zurück zu " Einführung Oracle SQL Developer " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Einführung Aqua Data Studio "

Allgemeines

TOAD (**T**ool for **O**racle **A**pplication **D**eveloper) ist ein Werkzeug zur Entwicklung und Administration von Oracle-Datenbanken und -Anwendungen aus dem Haus Quest Software. Er zeichnet sich hauptsächlich durch seine Vielfältigkeit und umfangreichen Funktionen aus und ist dabei übersichtlich gestaltet und leicht zu bedienen. Toad enthält folgende Bereiche: SQL Editor, Schema Browser, Procedure Editor, SQL Modeler. Daneben bietet es Funktionen für Datenbankadministratoren.

SQL Editor

Editor mit Syntaxhervorhebung, in dem SQL- und PL/SQL-Skripte formuliert, gespeichert und ausgeführt werden können.

Schema Browser

Der Schema Browser erlaubt die Pflege und Verwaltung aller Oracle-Objekte (Tabelle, Index, View, Sequence, Package, Trigger etc.). Zur Tabellenpflege gehört auch die editierbare, tabellarische Darstellung von Tabelleninhalten/Daten.

Procedure Editor

Der Procedure Editor dient der Entwicklung und dem Debugging von PL/SQL-Prozeduren, -Funktionen und -Packages. Seit der Version 9 von Toad sind SQL Editor und Procedure Editor zu einem Editor zusammengefasst.

ER Diagram

ER Diagram erstellt eine grafische Darstellung des Tabellenschemas mit Fremdschlüsselbeziehungen und liefert damit weniger ein ER-Diagramm sondern eher ein Servermodell-Diagramm vergleichbar mit dem Oracle Designer.

Aqua Data Studio

 Zurück zu " Einführung Toad " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Datenbank-Objekte#Table "

Aqua Data Studio

Aqua Data Studio ist ein Datenbank Abfrage und Administration Programm für Sql-Datenbanken wie zum Beispiel Oracle, welches auch ohne SQL*PLUS funktioniert. Es basiert auf JDBC und hat dadurch den Vorteil, dass man nicht erst alles bei Oracle lange zusammen suchen muss, sondern relativ einfach und schnell „loslegen“ kann.

Aqua Data Studio ist für Schulungszwecke, also für nicht kommerzielle Zwecke kostenlos. Selbst die kommerziellen Lizenzen sind wesentlich günstiger als eine Lizenz von Quests Toad.

Table

Zurück zu " ORACLE Datenbank-Objekte " | Hoch zu " Inhaltsverzeichnis " | Vor zu " View "

Tabellendefinition

Es gibt drei Arten von Tabellen:

- Relationale Tabellen
- Objekttabellen
- XML Tabellen

Die relationale Tabelle ist die Grundstruktur um Daten zu halten. Sie ist in der Form von Spalten (Columns) und Zeilen (Rows) aufgebaut.

Syntax

```
CREATE TABLE <schema.table>(
  <relational properties>
);
```

Beispielscript

```
CREATE TABLE reltable(
  id         NUMBER,
  username   VARCHAR2(50),
  email      VARCHAR2(50),
  job        VARCHAR2(50)
);
```

RELTABLE

ID	USERNAME	EMAIL	JOB
1	BBLOCKSBERG	bb@besen.hex	Hexe
2	BBLÜMCHEN	bb@z.oo	Dickhäuter

Objekttabellen erweitern relationale Tabelle um die Möglichkeit der Nutzung von Objekttypen. Mindestens eine Spalte der Tabelle beinhaltet die Definition eines Objekttyps.

OBJTABLE

ID	USERINFO
1	(Objectdata: BBLOCKSBERG bb@besen.hex Hexe)
2	(Objectdata: BBLÜMCHEN bb@z.oo Dickhäuter)

OBJTYPE

USERNAME	EMAIL	JOB

. . .

Check-Constraints

Bereits bei der Definition einer Tabelle, kann man festlegen, welche Werte oder Wertebereiche für einen Spaltenwert erlaubt oder verboten sind. Dadurch wird die Integrität der Datenbank erhöht, da nur Werte in die Datenbank eingepflegt werden

können, die diesen Regeln entsprechen. Beispiel:

```
CREATE TABLE TEST (
  T1      VARCHAR(10) NOT NULL,
  T2      NUMBER(2)   DEFAULT 10,
  T3      NUMBER(3,2) CHECK T3 >=5
) TABLESPACE T;
```

NOT NULL	Bestimmt, dass der Spalte im Datensatz ein Wert zugeordnet werden muss.
DEFAULT	Setzt einen Vorgabewert
CHECK	Ist eine Boolesche Regel, die Wahr sein muß, damit der Datensatz akzeptiert wird

Primärschlüssel

```
ALTER TABLE reltable ADD CONSTRAINT reltable_pk PRIMARY KEY (id);

ALTER TABLE reltable ADD PRIMARY KEY (id) DISABLE;
```

Fremdschlüssel

```
ALTER TABLE reltable
ADD CONSTRAINT fk_reltable2
  FOREIGN KEY (reltable2_id)
  REFERENCES reltable2(id);
```

Einfügereihenfolge ermitteln

Wenn viele Tabellen existieren, die mit Fremdschlüsseln miteinander verknüpft sind, dann kann man sich die Einfüge-Reihenfolge der Tabellen aus den Informationen im Datenbank-Distionary generieren lassen.

Beispiel für Oracle:

```
WITH rel AS
(
-- rel start
SELECT
  child_c.table_name child
, parent_c.table_name parent
FROM  user_constraints child_c, user_constraints parent_c
WHERE child_c.constraint_type = 'R'
AND   child_c.r_constraint_name = parent_c.constraint_name
-- rel ende
)
, ebenen AS
(
-- ebenen start
SELECT level+1 ebene, child tabelle
FROM rel
CONNECT BY PRIOR child = parent
START WITH parent IN
(
SELECT parent
FROM rel
WHERE parent NOT IN (SELECT child FROM rel)
)
-- Ebene 1 hinzufügen
UNION
SELECT 1, parent
FROM rel
WHERE parent NOT IN (SELECT child FROM rel)
-- ebenen ende
)
-- Ermitteln und ausgeben der einzelnen Ebenen
SELECT MAX(ebene) ebene, tabelle
FROM ebenen
GROUP BY tabelle
```

Diese Query verwendet Informationen aus der Dictionary-View USER_CONSTRAINTS. Durch mehrere inline-Views und einem rekursivem SQL-Statement werden die gesuchten Informationen ermittelt. Die Formulierung 'CONNECT BY PRIOR'

ist in Oracle eine Möglichkeit, rekursiv vorzugehen. Es werden durch 'START WITH parent IN' zuerst alle Tabellen gesucht, die keine eigenen Fremdschlüssel haben, die aber Detail-Tabellen haben (das ist die Ebene 1). Dann werden alle Tabellen gesucht, die Fremdschlüssel zu den bereits gefundenen Tabellen haben (das ist die Ebene 2). Dann werden alle neu gefundenen Tabellen untersucht, ob sie wiederum Detail-Tabellen haben. Genau das ist die Rekursion. Sie wird so lange weiter fortgesetzt, bis alle Detail-Tabellen gefunden wurden. (Ebenen 3, 4, 5, u.s.w) Die Pseudo-Spalte LEVEL gibt in Oracle bei rekursiven SQL-Statements an, wie viele Rekursions-Schritte bereits ausgeführt wurden.

Bei der oben beschriebenen Query werden Tabellen ohne Fremdschlüssel nicht ausgegeben. Falls es Ring-Verkettungen gibt, dann werden die daran beteiligten Tabellen auch nicht ausgegeben, da sie bei der 'START WITH' Klausel nicht gefunden werden. Wenn Ring-Verkettungen vorhanden sind, dann kann man diese mit der folgenden Query ausgeben. Sie verwendet auch wieder die Inline-Views REL und EBENEN und sucht schließlich nach Tabellen in USER_TABLES, die zwar in REL enthalten sind (die also Fremdschlüssel haben), die aber nicht in der View EBENEN enthalten sind. Falls eine Ring-Verkettung nur eine einzige Tabelle betrifft, also die Tabelle einen Fremdschlüssel hat, der auf die eigene Tabelle verweist, dann ist dafür noch ein weiterer Schritt erforderlich. Solche Tabellen werden gefunden mit dem Zugriff auf REL mit der Bedingung 'WHERE parent = child'.

```
WITH rel AS
(
-- rel start
SELECT
  child_c.table_name child
, parent_c.table_name parent
FROM  user_constraints child_c, user_constraints parent_c
WHERE child_c.constraint_type = 'R'
AND   child_c.r_constraint_name = parent_c.constraint_name
-- rel ende
)
, ebenen AS
(
-- ebenen start
SELECT level+1 ebene, child tabelle
FROM rel
CONNECT BY PRIOR child = parent
START WITH parent IN
(
SELECT parent
FROM rel
WHERE parent NOT IN (SELECT child FROM rel)
)
-- Ebene 1 hinzufügen
UNION
SELECT 1, parent
FROM rel
WHERE parent NOT IN (SELECT child FROM rel)
-- ebenen ende
)
SELECT table_name tabelle
FROM user_tables
WHERE table_name NOT IN (SELECT tabelle FROM ebenen)
AND   table_name IN (SELECT parent FROM rel)
-- Und noch alle Cycle mit nur einem Element hinzufügen
UNION
SELECT parent tabelle
FROM rel
WHERE parent = child
```

Die REL-View kann man auch dazu verwenden, um die dritte Gruppe von Tabellen zu finden, nämlich diejenigen, die mit Fremdschlüsseln nichts zu tun haben. Diese Tabellen haben keine eigenen Fremdschlüssel und werden auch von keiner anderen Tabelle als als Master-Tabelle referenziert. Dafür sucht man nach allen Tabellen in USER_TABLES, die weder als Parent, noch als Child in der REL-View vorkommen.

```
WITH rel AS
(
-- rel start
SELECT
  child_c.table_name child
, parent_c.table_name parent
FROM  user_constraints child_c, user_constraints parent_c
WHERE child_c.constraint_type = 'R'
AND   child_c.r_constraint_name = parent_c.constraint_name
-- rel ende
)
SELECT table_name tabelle
FROM user_tables
WHERE table_name NOT IN (SELECT parent FROM rel
                  UNION SELECT  child FROM rel)
```

Partitionierung

Partitionierung nennt man den Vorgang aus einer großen Tabelle zwecks Performanzoptimierung in kleinere Teiltabellen zu zerlegen.

Es gibt verschiedene Arten der Partitionierung:

- Range-Partitionierung
- Hash-Partitionierung
- List-Partitionierung
- Interval-Partitionierung

Ferner kann man Partitionen noch weiter in Sub-Partitionen unterteilen.

Range-Partitionierung

Beispiel

```
CREATE TABLE t_range
( t1      VARCHAR2(10) NOT NULL,
  t2      NUMBER       NOT NULL,
  t3      NUMBER
)
PARTITION BY RANGE (t2)
( PARTITION part1 VALUES LESS THAN (1),
  PARTITION part2 VALUES LESS THAN (11),
  PARTITION part3 VALUES LESS THAN (MAXVALUE)
);
```

Eine Partition mit MAXVALUE nuss nicht angegeben werden. Dann allerdings können keine Werte größer oder gleich 11 für die Spalte t2 eingefügt werden.

Die Partitionierung kann auch von mehreren Spalten abhängig gemacht werden:

```
CREATE TABLE t_range2
( jahr      NUMBER  NOT NULL,
  monat     NUMBER  NOT NULL,
  tag       NUMBER  NOT NULL,
  zeitpunkt DATE    NOT NULL,
  umsatz    NUMBER(14,2)
)
PARTITION BY RANGE (jahr, monat, tag)
( PARTITION p_2013       VALUES LESS THAN (2013, MAXVALUE, MAXVALUE),
  PARTITION p_2014_01    VALUES LESS THAN (2014,  1, MAXVALUE),
  PARTITION p_2014_02    VALUES LESS THAN (2014,  2, MAXVALUE),
  PARTITION p_2014_03    VALUES LESS THAN (2014,  3, MAXVALUE),
  PARTITION p_2014_04_01 VALUES LESS THAN (2014,  4,  2),
  PARTITION p_2014_04_02 VALUES LESS THAN (2014,  4,  3),
  PARTITION p_2014_04_03 VALUES LESS THAN (2014,  4,  4),
  PARTITION p_2014_04_04 VALUES LESS THAN (2014,  4,  5),
  PARTITION p_max        VALUES LESS THAN (MAXVALUE, MAXVALUE, MAXVALUE)
);
```

In einer Query kann man sich auf die einzelnen Partitionen beziehen, indem man entweder den Namen der Partition angibt, oder die Werte, die die Partitionierung beeinflussen. Beispiel:

```
select * from t_range2 partition( p_2013 );
select * from t_range2 partition for ( 2013, 12, 1 );
```

Hash-Partitionierung

Beispiel

```
CREATE TABLE t_hash
( t1      VARCHAR2(10) NOT NULL,
  t2      NUMBER       NOT NULL,
  t3      NUMBER
PARTITION BY HASH (t2)
PARTITIONS 4
;
```

List-Partitionierung

Beispiel

```
CREATE TABLE t_list
( ort     VARCHAR2(30) NOT NULL,
  t2      NUMBER,
  t3      NUMBER
)
PARTITION BY LIST(ort)
( PARTITION part_nord VALUES IN ('Hamburg','Berlin'),
  PARTITION part_sued VALUES IN ('Muenchen', 'Nuernberg'),
  PARTITION part_west VALUES IN ('Koeln','Duesseldorf'),
  PARTITION part_ost VALUES IN ('Halle'),
  PARTITION part_def VALUES (DEFAULT)
);
```

Wenn keine Default-Partition angelegt wird, dann können für die Spalte ort nur die angegebenen Werte eingetragen werden.

Interval-Partitionierung

Intervall-Partitionierung ist eine besondere Form der Range-Partitionierung, wobei die Spalte, von der die Partitionierung abhängt, den Datentyp DATE haben muss.

Beispiel

```
CREATE TABLE t_interval
( buchungs_datum  DATE NOT NULL,
  buchungs_text   VARCHAR2(100),
  betrag          NUMBER(10,2)
)
PARTITION BY RANGE (buchungs_datum)
INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'))
( PARTITION p_historie VALUES LESS THAN (TO_DATE('2014.01.01', 'YYYY.MM.DD')),
  PARTITION p_2014_01  VALUES LESS THAN (TO_DATE('2014.02.01', 'YYYY.MM.DD')),
  PARTITION p_2014_02  VALUES LESS THAN (TO_DATE('2014.03.01', 'YYYY.MM.DD'))
);
```

Bei dieser Tabelle werden initial die drei angegebenen Partitionen angelegt. Sobald in die Tabelle Sätze eingefügt werden, bei denen der Wert für buchungs_datum nicht in die bestehenden Partitionen passt, wird automatisch weitere Partitionen erzeugt. Dabei wird pro Monat eine neue Partition erzeugt.

Man kann die Bildung neuer Intervalle jederzeit ändern. Die Änderung beeinflusst nicht die bereits existierenden Partitionen. Nur wenn neue Partitionen nach der letzten existierenden Partition gebraucht werden, kommt die Intervall-Angabe zur Anwendung.

Beispiel zur Definition, dass neue Partitionen nur im Rhytmus von jeweils 2 Jahren angelegt werden sollen:

```
ALTER TABLE t_interval
SET INTERVAL (NUMTOYMINTERVAL(2,'YEAR'))
;
```

Beispiel zur Definition, dass neue Partitionen nur im Rhytmus von jeweils 5 Tagen angelegt werden sollen:

```
ALTER TABLE t_interval
SET INTERVAL (NUMTODSINTERVAL(5,'DAY'))
;
```

temporäre Tabellen

...

externe Tabellen

...

LOBs

...

Datadictionary-Views für Tabellen

In Oracle gibt es fast alle Dictionary-Views in dreifacher Ausführung:

- Views mit dem Präfix USER_ zeigen die eigenen Objekte an, also die Objekte, die im eigenen Schema erstellt sind.
- Views mit dem Präfix ALL_ zeigen alle Objekte an, für die man eine Zugriffsberechtigung hat. Das sind die Objekte im eigenen Schema und auch Objekte in anderen Schemata, für die man durch den GRANT-Befehl eine Zugriffsberechtigung erhalten hat.
- Auf Views mit dem Präfix DBA_ kann man nur zugreifen, wenn man das Administrations-Recht hat. In dieser View werden alle Objekte der gesamten Datenbank angezeigt, also auch die, auf die man keine Zugriffsrechte besitzt.

Alle Oracle Dictionary-Views sind in dem Manual Reference (nicht: SQL-Reference) beschrieben.

Einige Beispiele

Alle eigenen Tabellen einer Datenbank anzeigen:

```
SELECT TABLE_NAME FROM USER_TABLES
```

Alle Tabellen anzeigen, auf die man zugriffsberechtigt ist:

```
SELECT TABLE_NAME FROM ALL_TABLES
```

Alle Tabellen, die es in der gesamten Datenbank gibt, anzeigen:

```
SELECT TABLE_NAME FROM DBA_TABLES
```

Alle Spaltennamen anzeigen mit den dazugehörigen Datentypen einer Tabelle:

```
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE, DATA_LENGTH,
      DATA_PRECISION, DATA_SCALE, NULLABLE, COLUMN_ID
FROM USER_TAB_COLUMNS
WHERE TABLE_NAME = 'Tabellenname'
ORDER BY COLUMN_ID
```

Seit der Version 11 gibt es zusätzlich die View ALL_TAB_COLS. Der Unterschied zu ALL_TAB_COLUMNS besteht darin, dass versteckte Spalten in dieser View nicht ausgefiltert werden:

```
SELECT * FROM ALL_TAB_COLS
```

Alle Fremdschlüssel-Beziehungen und anderen Constraints anzeigen. Fremdschlüssel-Beziehungen haben den Typ 'R'. Bei DELETE_RULE = 'CASCADE' handelt es sich um eine Beziehung mit Löschweitergabe und bei 'NO ACTION' um eine Beziehung mit Lösch-Restriktion.

```
SELECT CONSTRAINT_NAME, TABLE_NAME, R_CONSTRAINT_NAME REFERENCED_TABLE, DELETE_RULE
FROM USER_CONSTRAINTS
WHERE CONSTRAINT_TYPE = 'R'
```

Tabellen-Kommentare werden in einer eigenen View bereitgestellt:

```
SELECT TABLE_NAME, TABLE_TYPE, COMMENT
FROM ALL_TAB_COMMENTS
```

Ab der Version 11g gibt es eine eigene View zur Ausgabe von Statistik-Daten:

```
SELECT * FROM ALL_TAB_STATISTICS
```

Zusätzliche Informationen über partitionierte Tabellen. Jede partitionierte Tabelle hat hier einen Eintrag:

```
SELECT * FROM ALL_PART_TABLES
```

Informationen über Partitionen und Sub-Partitionen. Jede Partition bzw. Sub-Partition hat hier eine eigene Zeile:

```
SELECT * FROM ALL_TAB_PARTITIONS
SELECT * FROM ALL_TAB_SUBPARTITIONS
```

Tabellen verkleinern

Wenn man Sätze aus einer Tabelle löscht, dann wird der Speicherplatz im Tablespace trotzdem nicht freigegeben. Alle Datenblöcke, die einmal für die Speicherung von Daten für eine bestimmte Tabelle verwendet wurden, bleiben dieser Tabelle zugeordnet und können nicht zur Speicherung von Daten in anderen Tabellen benutzt werden. Wenn man eine Tabelle nur vorübergehend mit einem großen Datenvolumen gefüllt wurde und der Speicherplatz danach für andere Tabellen zur Verfügung gestellt werden soll, dann gibt es verschiedene Möglichkeiten:

- ALTER TABLE MOVE TABLESPACE
- EXPORT / drop Table / create Table / IMPORT
- ALTER TABLE SHRINK SPACE

Die Varianten 1 und 2 haben den Nachteil, dass es eine Ausfallzeit gibt, in der andere Programme, die auf diese Tabelle zugreifen wollen, eine Fehlermeldung bekommen.

Die Variante 3 gibt es seit der Version 10g. Sie hat den Vorteil, dass die Tabelle nur durch einen Lock gesperrt wird. Andere Programme müssen so lange warten, bis die Verkleinerung fertig ist, sie bekommen aber keine Fehlermeldung.

Der Befehl ALTER TABLE SHRINK SPACE ist an bestimmte Bedingungen geknüpft:

Verkleinern kann man damit

- Tabellen
- Indizes
- indexorganisierte Tabellen
- Partitionen
- Subpartitionen
- LOB Segmente (ab Version 10.2)
- Materialized Views

Voraussetzungen

- Oracle RDBMS ab Version 10g.
- Der Tablespace muss mit der Option SEGMENT MANAGEMENT AUTO angelegt sein.
- Bei der Tabelle, die verkleinert werden soll, muss ROW MOVEMENT aktiviert sein.

Einschränkungen

- Die Tabelle darf nicht komprimiert sein.
- Die Tabelle darf keine FUNCTION BASED Indizes besitzen.
- Die Mastertabelle einer ON COMMIT MATERIALIZED VIEW kann nicht verkleinert werden.
- ROWID MATERIALIZED VIEWS müssen nach dem Verkleinern neu aufgebaut werden.
- Tabelle darf keine LOB oder LONG Spalten besitzen (nur in Version 10.1).
- Der Befehl SHRINK SPACE ist eine Art der Reorganisation. Dadurch ändern sich die ROWIDs.

```
 alter table test ENABLE ROW MOVEMENT;
 alter table test SHRINK SPACE;
```

 Zurück zu " ORACLE Datenbank-Objekte " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " View "

View

Zurück zu " Tabelle " | Hoch zu " Inhaltsverzeichnis " | Vor zu " Materialized View "

Permanent definierte Views

Views sind virtuelle Tabellen. Das heißt sie werden aus Tabellen zur Laufzeit gebildet. Views lösen viele praktische Probleme, da sie zum einen SELECT-Abfragen überflüssig machen können, oder den Zugriff auf eine Tabelle einschränken können, um Spalten für Anwender zu verbergen. Eine View speichert selber jedoch keine Daten, sondern wendet nur das SQL-Statement an, das bei der View-Definition hinterlegt wurde.

Beispiel:

```
CREATE OR REPLACE VIEW my_view AS
   SELECT *
   FROM tbl_bsp
```

Wenn eine View ein komplexeres SQL-Statement verwendet (Join, Group-by, Funktionen) dann kann diese View nicht mehr für Schreib-Befehle (INSERT, UPDATE, DELETE) verwendet werden. Abhilfe können hier INSTEAD-OF-TRIGGER leisten.

Wenn die Tabelle, auf die sich die View bezieht, geändert wird, dann ist die View ungültig. Sie kann dann mit dem folgenden Befehl wieder validiert werden:

```
ALTER VIEW my_view COMPILE
```

Inline-Views

Seit der Version 10g ist es möglich, eine View nur für die Ausführung eines einzelnen SQL-Statements zu definieren und dann in diesem zu verwenden. Solche Views werden nicht im Dictionary eingetragen.

Beispiel:

```
WITH abc AS
(
SELECT name, id
FROM mitarbeiter
WHERE geschlecht = 'W'
)
SELECT name, projekt_nr
FROM abc, projekt
WHERE abc.is = projekt.pl_id
```

Datadictionary-Views für Views

In der Spalte VIEW_TEXT ist das SQL-Statement angegeben, dass zum Erstellen der View verwendet wurde.

```
SELECT * FROM USER_VIEWS
SELECT * FROM ALL_VIEWS
SELECT * FROM DBA_VIEWS
```

Materialized View

Zurück zu " View " | Hoch zu " Inhaltsverzeichnis " | Vor zu " Index "

Materialized Views

Im Gegensatz zu normalen Views, sind Materialized Views physikalisch gespeichert um so die Zugriffe zu beschleunigen. Dies ist vor allem bei großen Datenmengen und komplexen Abfragen von Vorteil.

CREATE

```
CREATE MATERIALIZED VIEW MV1
AS SELECT * FROM client1
```

Mit täglich REFRESH:

```
CREATE MATERIALIZED VIEW MV2
REFRESH FAST
START WITH SYSDATE
NEXT SYSDATE + 1
AS SELECT * FROM client1;
```

Eigentlich, `select SYSDATE from DUAL` gibt das Datum.

SELECT MATERIALIZED VIEW

```
SELECT QUERY FROM ALL_MVIEWS
WHERE MVIEW_NAME='MV1'
```

Index

 Zurück zu " Materialized View " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Tablespace "

Ein Index ist ein Datenbanksegment, dass über ein oder mehrere Spalten einer Tabelle erstellt wird. Der Datenzugriff einer Tabelle wird somit schneller. Auf eine bestimmte Spaltenfolge kann nur ein Index erstellt werden.

Es gibt verschiedene Arten von Indexen:

- B-tree Index
- funktionsbasierter B-tree Index
- B-tree Cluster Index
- Hash Cluster Index
- Reverse Key Index
- Bitmap Index
- Bitmap Join Index

Eine spezielle Möglichkeit sind Index-Organized Tables. Hier gibt es nur die Index-Struktur und die sonst zugrundeliegende Tabelle entfällt.

Wenn die Tabelle, auf die sich ein Index bezieht, partitioniert ist, dann kann der Index in der selben Weise partitioniert werden (lokale Partitionierung). Er kann auch ohne Partitionierung als eine einzige Struktur erstellt werden oder er kann eine ganz eigene Partitionierung haben (globale Partitionierung).

Datadictionary-Views für Indices

Alle Indices anzeigen.

In der Spalte UNIQUENESS ist angegeben, ob es sich um einen eindeutigen Index handelt (UNIQUE) oder die Index-Werte in der Tabelle mehrmals vorkommen dürfen (NONUNIQUE)

```
SELECT INDEX_NAME, TABLE_NAME, UNIQUENESS FROM USER_INDEXES
```

Index-Partitionen anzeigen:

```
SELECT * FROM USER_IND_PARTITIONS
```

Tablespace

Zurück zu " Index " | Hoch zu " Inhaltsverzeichnis " | Vor zu " Prozeduren "

Ein Tablespace ist eine logische Einheit zum Speichern von Datenobjekten (z. B. Tabellen). Ein Tablespace muss aus mindestens einem und kann aus mehreren Datafiles bestehen. Ein Tablespace kann vergrößert werden, indem ein Datafile angehängt oder vergrößert wird. Ein einmal zugeordnetes Datafile kann nicht mehr entfernt werden. Es gibt nur die Möglichkeit, alle Tabellen und sonstigen Objekte aus diesem Tablespace zu entfernen, den Tablespace als Ganzes zu löschen (DROP) und neu zu erstellen.

Es gibt Standard Tablespaces mit besonderer Funktion:

- SYSTEM = für Systemobjekte
- SYSAUX = ab 10g für bestimmte Systemobjekte (z. B. Statspack, Advisor, Scheduler)
- TEMP = temporär verwendeter Bereich für Sortierungen
- UNDO = ab 9i, Name kann abweichen, dient der Speicherung von Rollback-Informationen
- USERS = für die Benutzer

Zusätzlich können individuelle Tablespaces für Benutzerdaten angelegt werden. Für verschiedene Aufgabenbereiche können spezifische Tablespaces erstellt werden, um die Datenbank zu strukturieren.

Jeder Benutzer bekommt einen Tablespace zugewiesen, in dem er seine Objekte ablegen darf. Man sollte Benutzerdaten nicht im SYSTEM-Tablespace, sondern in einem Benutzer-Tablespace ablegen.

Tablespaces können auf Read-Only gesetzt werden, um Datenänderungen zu verhindern. Dies kann z. B. für historische Daten sinnvoll sein. Ein Read-Only-Tablespace kann auch wieder in den Schreibmodus versetzt werden.

Die Verfügbarkeit von Tablespaces kann Online (Standard) und Offline sein. Ein Tablespace kann z. B. offline gesetzt werden, um ein Backup durchzuführen oder bestimmte Daten nicht zugreifbar zu machen.

Dictionary-Views zu Tablespaces

```
select * from v$tablespace;
select * from dba_tablespaces;
```

Zuordnung der Datafiles zu TS

```
select tablespace_name, file_name, bytes
from dba_data_files
order by  tablespace_name, file_name
```

Datafiles vom Temp-Tablespace

```
select  tablespace_name, file_name, bytes
from dba_temp_files
```

Prozeduren

 Zurück zu " Tablespace " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Funktionen "

Einstieg

Möglichkeiten, eine Prozedur zu erstellen:

- Eine Prozedur kann erstellt werden, indem ein compiliertes Programm als Aktion definiert wird.
- Eine Procedur kann auch eine PL/SQL-Script als Verarbeitungsteil erhalten.

Prozeduren sind nützlich, um regelmäßig wiederkehrende Arbeitsabläufe zu automatisieren z.B. um Installationsarbeiten durchzuführen.

Beispiel:

```
CREATE OR REPLACE PROCEDURE spins IS
BEGIN
   INSERT INTO tdept (deptno, deptname, mgrno, admrdept)
   VALUES ('A00', 'SPIFFY COMPUTER DIV.', '000010', 'A00');
   INSERT INTO tdept (deptno, deptname, mgrno, admrdept)
   VALUES ('B01', 'PLANNING           ', '000020', 'A00');
   INSERT INTO tdept (deptno, deptname, mgrno, admrdept)
   VALUES ('C01', 'INFORMATION CENTER  ', '000030', 'A00');
END;
```

Die Procedur wird im SQLPLUS aufgerufen mit dem Befehl:

```
EXECUTE spins;
```

Innerhalb eines PL/SQL-Scripts wird die Procedur nur durch Angabe ihres Namens aufgefufen.

```
BEGIN
   spins;
END;
```

Parameterübergabe

Wenn man der Prozedur Parameter übergeben will, dann wird unterschieden in:

- IN: Die aufrufende Umgebung übergibt einen Wert an die Prozedur. Dieser Parameter kann innerhalb der Verarbeitung nicht verändert werden. IN ist der default Parameter-Modus
- OUT: Die aufrufende Umgebung übergibt eine Variable an die Prozedur, die innerhalb der Prozedur als nicht initialisiert betrachtet wird. Wenn innerhalb der Verarbeitung diesem Parameter ein Wert zugewiesen wird, dann wird dieser an die aufrufende Umgebung zurückgegeben.
- IN OUT: Die Aufrufende Umgebung übergibt eine Variable an die Prozedur. Diese kann innerhalb der Prozedur verwendet werden und auch geändert werden. Der geänderte Wert wird der aufrufenden Umgebung mitgeteilt.

Einfache Prozedur, die einen Parameter übergeben bekommt:

```
CREATE OR REPLACE PROCEDURE myfirstproc(Parm1 IN NUMBER)
IS
BEGIN
   NULL;
END;
```

Es ist auch möglich, Werte von Prozeduren verändern zu lassen:

```
CREATE OR REPLACE PROCEDURE mysecondproc(param1 IN OUT NUMBER)
```

```
IS
BEGIN
   Param1 := Param1 + 42;
END;
```

Wird dieses zweite Beispiel verwendet, liefert der entsprechende Test:

```
DECLARE
  x NUMBER := 13;
BEGIN
   DBMS_OUTPUT.PUT_LINE( x );
   mysecondproc( x );
   DBMS_OUTPUT.PUT_LINE( x );
END;
```

wie zu erwarten die Werte 13 und 55.

Für die Übergabe der Daten per Referenz gibt es die **NOCOPY** Option bei den Parametern.

Verschlüsselung des Quelltextes

...

Übersicht über Proceduren, die bei der Installation mitgeliefert werden

...

Funktionen

Zurück zu " Prozeduren " | Hoch zu " Inhaltsverzeichnis " | Vor zu " Package "

Neben den vielen Funktionen, die die SQL-Sprache schon bietet, kann man eigene Funktionen definieren. Funktionen haben immer einen oder mehrere Input-Parameter und genau einen Ergebniswert.

Hier ein ganz einfaches Beispiel einer Funktion, die einen String als Input erwartet und einen String als Ergebnis liefert. Die Verarbeitung der Funktion besteht darin, den Input-Parameter um ein 'x' zu erweitern.

```
CREATE OR REPLACE FUNCTION myfunction (in_parm VARCHAR2) RETURN VARCHAR2 IS
BEGIN
   RETURN in_parm || 'x';
END;
/
```

Diese Funktion kann man nun genauso verwenden wie die anderen bereits vordefinierten Funktionen. Man muss nur darauf achten, dass man die richtige Anzahl an Parametern angibt und die passenden Datentypen wählt. Beispielaufruf:

```
SELECT loc, myfunction(loc)
FROM scott.dept;
```

Ergebnis:

```
LOC           MYFUNCTION(LOC)
------------- ---------------
NEW YORK      NEW YORKx
DALLAS        DALLASx
CHICAGO       CHICAGOx
BOSTON        BOSTONx
```

Im Anweisungsteil der Funktion können beliebig komplexe PL/SQL-Anweisungen angegeben werden. Ein Beispiel für eine etwas komplexere Funktion:

```
CREATE OR REPLACE FUNCTION anzma (abteilung VARCHAR2)
RETURN INTEGER IS
h_anzahl INTEGER;
BEGIN
   SELECT COUNT(*) INTO h_anzahl
   FROM ben01.templ
   WHERE workdept = abteilung;
   RETURN h_anzahl;
END;
/
```

Funktionen kann man gut testen, indem man sie in einem SELECT einsetzt, der aus der Tabelle dual liest. Diese Tabelle enthält genau einen Satz mit einer Spalte. Eigentlich ist man aber an diesem Satz gar nicht interessiert, sondern man will durch diesen SELECT nur bewirken, dass die Funktion genau einmal aufgerufen wird. Aufrufbeispiel:

```
SELECT anzma('A00')
FROM dual;
```

Ergebnis:

```
ANZMA('A00')
------------
           3
```

Hinweise:

- Die 1. Funktion ist deterministisch, d.h. wenn man die Funktion zweimal mit demselben Eingabe-Parameter aufruft,

dann liefert sie immer denselben Output.

- Die 2. Funktion ist nicht-deterministisch, d.h. ihr Ergebnis ist auch von anderen Informationen abhängig als nur den übergebenen Parametern. Wenn jemand die Sätze in der Tabelle ben01.templ ändert, dann kann die Funktion anzma bei gleichem Eingabe-Parameter ein anderes Ergebnis liefern.

Package

 Zurück zu " Funktionen " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Trigger "

Syntax

Ein Package ist eine Zusammenfassung auf verschiedenen Typen, Prozeduren und Funktionen. Die Definition eines Package geschieht in zwei Schritten. Erst wird die *Package Specification* erstellt. Das ist die Schnittstelle, die von außen "sichtbar" ist. In einem zweiten Schritt kann danach der *Package Body* erstellt werden, der die Verarbeitung beschreibt.

```
CREATE OR REPLACE PACKAGE TEST_PACKAGE AS
  PROCEDURE xy;
  FUNCTION abc(p_var VARCHAR2);
END TEST_PACKAGE;
```

Ausführung:

```
CALL TEST_PACKAGE.xy;
```

Syntax zum Erstellen eines *Package Bodys*:

```
CREATE OR REPLACE PACKAGE BODY TEST_PACKAGE AS
  PROCEDURE xy IS
  BEGIN
     ...
  END;
  FUNCTION abc(p_var VARCHAR2) IS
  BEGIN
     ...
  END;
END TEST_PACKAGE;
```

Globale Variablen und Typen in einem Package definieren ...

Überlagern von Prozeduren und Funktionen ...

Trigger

Zurück zu " Package " | Hoch zu " Inhaltsverzeichnis " | Vor zu " Sequenzen "

Was ist und macht ein Trigger

Trigger sind eventgesteuerte Prozeduren, die automatisch bei bestimmten Ereignissen durchgeführt werden.

Es gibt 3 Auslöser

- INSERT
- UPDATE
- DELETE

Zusätzlich kann noch der Ausführungszeitpunkt bestimmt werden

- BEFORE - vor der Änderung
- AFTER - nach der Änderung
- INSTEAD OF - anstelle der Änderung

Seit Oracle 9i können Trigger für folgende weitere Ereignisse definiert werden:

- DDL-Statements: CREATE, ALTER, DROP
- An- und Abmeldungen: LOGON, LOGOFF
- Start/Stop der Datenbank: STARTUP, SHUTDOWN
- Bei Systemfehler: SERVERERROR

Ein weiteres Kriterium ist, wie oft der Trigger gestartet werden soll

- ROW-Trigger: for each row

  werden pro geänderter Zeile ausgeführt
  Haben dadurch Zugriff auf die Attribute des Tupels vor und nach Ausführung des Triggers
  Anwendungsbeispiel: Protokollierung, Überprüfung von Aktionen, ...

- Statement-Trigger: for each statement

  werden pro ausgeführtem Statement einmalig ausgeführt, egal wieviele Zeilen betroffen sind
  Inhalt der Tupel ist nicht bekannt
  Anwendungsbeispiel: Zugriffsschutz, ...

Syntax eines Triggers

```
create or replace trigger <triggername>
before/after insert or update or delete
on <tablename>
REFERENCING NEW AS <newROW> OLD AS <oldROW>
for each row/for each statement
when (<Bedingung>)
DECLARE
   variablen deklaration
BEGIN
   if INSERTING then
    ...
   end if;
   if UPDATING then
    ...
   end if;
   if DELETING then
    ...
   end if;
```

```
EXCEPTION
   Fehlerbehandlung
END <triggername>;
```

Nur in ROW-Triggern werden die Alten (old) und Neuen (new) Werte der Tabelle zur Verfügung gestellt

Syntax bei Bedingungen: **new.spaltenname**, **old.spaltenname**

Syntax bei Aktionen: **:new.spaltenname**, **:old.spaltenname**

Bei BEFORE-Triggern besteht die Möglichkeit die NEW-Werte zu ändern

Weitere Infos: http://www.psoug.org/reference/instead_of_trigger.html

Dictionary-View zu Triggern

```
select * from user_triggers
```

Sequenzen und Timestamps in den Triggern (Vorlage)

```
-- (ersetze Platzhalter <% %>.)

CREATE SEQUENCE SQ_<%tableName%> START WITH 1 INCREMENT BY 1 MINVALUE 1;

CREATE OR REPLACE TRIGGER TS_<%tableName%>
BEFORE INSERT OR UPDATE
ON <%tableName%>
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
BEGIN
  IF (INSERTING) THEN
    --SELECT SYSDATE INTO :NEW.CHG_DATE FROM DUAL;
    --SELECT SYSDATE INTO :NEW.CRE_DATE FROM DUAL;
    IF (:NEW.<%pkFieldName%> IS NULL) THEN
      SELECT SQ_<%tableName%>.NEXTVAL INTO :NEW.<%pkFieldName%> FROM DUAL;
    END IF;

  ELSIF (UPDATING) THEN
    --SELECT SYSDATE INTO :NEW.CHG_DATE FROM DUAL;
  END IF;
END;

-- Kein Setzen von SEQUENCE_NAME.nextval und von Timestamp in den Insert / Update Abfragen ist nun notwendig.

-- INSERT:
 -- statt:
INSERT INTO tabelle1 (id, name, CRE_DATE, CHG_DATE) VALUES (SEQUENCE_NAME.nextval, 'Test', SYSDATE, SYSDATE );
 -- nun:
INSERT INTO tabelle1 (name) VALUES ('Test');

-- UPDATE:
 -- statt:
UPDATE tabelle1 SET name='Test 2', CHG_DATE = SYSDATE WHERE id = 1;
 -- nun:
UPDATE tabelle1 SET name='Test 2' WHERE id = 1;
```

Sequenzen

 Zurück zu " Trigger " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " sonstige Objekte "

Sequenzen sind Generatoren für numerische Werte, die automatisch hochgezählt werden und üblicherweise für Primärschlüsselwerte verwendet werden. Sinn einer Sequenz ist die Vermeidung des beliebten Anfängerfehlers "select max(id)+1 from xyz" zur Erzeugung des nächsten Primärschlüsselwertes.

Syntax zum Erzeugen einer Sequenz

```
CREATE SEQUENCE SEQUENCE_NAME
INCREMENT BY 1   -- Schrittgröße beim Hochzählen
START WITH 1     -- Startwert
MINVALUE 1       -- Kleinster Wert
MAXVALUE 999999 -- Größter Wert
NOCYCLE / CYCLE -- wieder bei MINVALUE starten wenn MAXVALUE überschritten wurde
CACHE 20
NOORDER;
```

Verwenden von Sequenzen

```
select SEQUENCE_NAME.nextval from dual
```

liefert den nächsten Wert der Sequenz

```
select SEQUENCE_NAME.currval from dual
```

liefert den aktuellen Wert der Sequenz, das heißt, genau den Wert, der beim letzten Aufruf von SEQUENCE_NAME.nextval zurückgeliefert wurde. SEQUENCE_NAME.currval kann erst aufgerufen werden, wenn vorher mindestens einmal SEQUENCE_NAME.nextval aufgerufen worden ist. Mit Currval kann man jedoch nur Werte abfragen, die in der eigenen Session erzeugt wurden. Selbst wenn in anderen parallel laufenden Sessions von der selben Sequence bereits weitere Werte generiert wurden, dann liefert Currval immer noch den zuletzt für die eigene Session generierten Wert. Wenn man in einer Session noch nicht mit Nextval einen Wert generiert hat, dann kann man auch nicht mit Currval den zuletzt generierten Wert abfragen.

Anwendungsbeispiel

```
INSERT INTO tabelle1 (num, name)
VALUES (SEQUENCE_NAME.nextval, 'Test');
```

Oft steht man vor dem Problem, dass man den Wert, der soeben in die Datenbank geschrieben wurde weiterverwenden will, z.B. um einen Detaildatensatz in einer Untergeordneten Tabelle anzulegen, der über den Fremdschlüssel num verbunden ist. Hierzu gibt es 2 Möglichkeiten:

1. Referenzieren über SEQUENCE_NAME.currval. Nachteil hierbei ist, dass insbesondere bei längeren Programmen oftmals nicht sichergestellt werden kann, dass der Wert der Sequenz unverändert ist.
2. Innerhalb eines PL/SQL-Programms kann man den Wert über die RETURNING-Klausel direkt in eine Variable speichern:

```
INSERT INTO tabelle1 (num, name)
VALUES (SEQUENCE_NAME.nextval, 'Test')
RETURNING num INTO v_aktuellerSequenzwert;
```

Alternativ Speicherung in einer PL/SQL-Variablen:

```
SELECT SEQUENCE_NAME.nextval INTO v_aktuellerSequenzwert FROM DUAL;
```

Beim Verwenden einer Sequenz kann nicht sichergestellt werden, dass die eingetragenen Werte lückenlos aufeinanderfolgen. Das liegt schon daran, dass immer mehrere Werte im Voraus generiert werden und in einem Cache gespeichert werden. Wenn die Datenbank heruntergefahren wird, dann gehen die im Cache gespeicherten Werte verloren.

Dictionary-View zu Sequenzen

```
select * from user_sequences;
```

Trigger (Vorlage) für eine Sequenz

====(ersetze Platzhalter <%%>. ====

==== Zusätzlich (& nützlich): Timestamp's im Trigger. Siehe z.B.: --SELECT SYSDATE INTO :NEW.CHG_DATE FROM DUAL; // ggf. entfernen.====

```
CREATE SEQUENCE SQ_<%tableName%> START WITH 1 INCREMENT BY 1 MINVALUE 1;
/
CREATE OR REPLACE TRIGGER TS_<%tableName%>
BEFORE INSERT OR UPDATE
ON <%tableName%>
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
BEGIN
  IF (INSERTING) THEN
    --SELECT SYSDATE INTO :NEW.CHG_DATE FROM DUAL;
    --SELECT SYSDATE INTO :NEW.CRE_DATE FROM DUAL;
    IF (:NEW.<%pkFieldName%> IS NULL) THEN
      SELECT SQ_<%tableName%>.NEXTVAL INTO :NEW.<%pkFieldName%> FROM DUAL;
    END IF;

  ELSIF (UPDATING) THEN
    --SELECT SYSDATE INTO :NEW.CHG_DATE FROM DUAL;
  END IF;
END;
/

-- Kein Setzen von SEQUENCE_NAME.nextval in der Abfrage ist nun notwendig, Trigger z.B. TS_tabelle1 macht es nun beim Ins
-- Bsp.:
INSERT INTO tabelle1 (name) VALUES ('Test');

-- Nachträgliches holen von gerade erstellten Id mit:
SELECT  SQ_<%tableName%>.currval as value FROM dual;
```

sonstige Objekte

 Zurück zu " Sequenzen " |  Hoch zu " Inhaltsverzeichnis " |

Synonym

Ein Synonym (anderer Name) für Datenbankobjekte wie Tabellen, Stored Procedures oder andere Synonyms.

```
CREATE SYNONYM table1 FOR reltable;
```

Database Link

Ein Database Link ist eine gespeicherte Verbindung zu einer anderen Datenbank mit einem definierten Benutzer. Es werden die Berechtigungen dieses Benutzers in der Remote-Datenbank verwendet. Die Auflösung der Remote-Datenbank erfolgt über die tnsnames.ora.

Alle DB-Links können über folgende Abfrage ermittelt werden: `select * from dba_db_links;`

Die Abfrage der Remote-Datenbank erfolgt über das Anhängen von @<DB_LINK> an eine normale Abfrage. Zum Beispiel:

```
select user from dual@remote.world;
```

oder

```
select * from abakus@db3;
```

wenn `abakus` der Name einer Tabelle, einer View oder eines Synonyms auf der Remote-Datenbank ist und `db3` der Name eines Database Link.

Ein Beispiel für das Anlegen eines Database Link (Fixed User Database Link):

```
create database link db3 connect to user51 identified by tz4ut using 'tomate';
```

- `db3` ist hier der Name des Database Link, der angelegt werden soll
- `user51` der vorgegebene User auf der Remote-Datenbank
- `tz4ut` das Passwort dieses Users user51
- `tomate` ist der Servicename der Remote-Datenbank entsprechend dem Eintrag in der lokalen tnsnames.ora

sonstige Objekte#Synonym

 Zurück zu " Sequenzen " |  Hoch zu " Inhaltsverzeichnis " |

Synonym

Ein Synonym (anderer Name) für Datenbankobjekte wie Tabellen, Stored Procedures oder andere Synonyms.

```
CREATE SYNONYM table1 FOR reltable;
```

Database Link

Ein Database Link ist eine gespeicherte Verbindung zu einer anderen Datenbank mit einem definierten Benutzer. Es werden die Berechtigungen dieses Benutzers in der Remote-Datenbank verwendet. Die Auflösung der Remote-Datenbank erfolgt über die tnsnames.ora.

Alle DB-Links können über folgende Abfrage ermittelt werden: `select * from dba_db_links;`

Die Abfrage der Remote-Datenbank erfolgt über das Anhängen von @<DB_LINK> an eine normale Abfrage. Zum Beispiel:

```
select user from dual@remote.world;
```

oder

```
select * from abakus@db3;
```

wenn `abakus` der Name einer Tabelle, einer View oder eines Synonyms auf der Remote-Datenbank ist und `db3` der Name eines Database Link.

Ein Beispiel für das Anlegen eines Database Link (Fixed User Database Link):

```
create database link db3 connect to user51 identified by tz4ut using 'tomate';
```

- `db3` ist hier der Name des Database Link, der angelegt werden soll
- `user51` der vorgegebene User auf der Remote-Datenbank
- `tz4ut` das Passwort dieses Users user51
- `tomate` ist der Servicename der Remote-Datenbank entsprechend dem Eintrag in der lokalen tnsnames.ora

sonstige Objekte#Database Link

 Zurück zu " Sequenzen " |  Hoch zu " Inhaltsverzeichnis " |

Synonym

Ein Synonym (anderer Name) für Datenbankobjekte wie Tabellen, Stored Procedures oder andere Synonyms.

```
CREATE SYNONYM table1 FOR reltable;
```

Database Link

Ein Database Link ist eine gespeicherte Verbindung zu einer anderen Datenbank mit einem definierten Benutzer. Es werden die Berechtigungen dieses Benutzers in der Remote-Datenbank verwendet. Die Auflösung der Remote-Datenbank erfolgt über die tnsnames.ora.

Alle DB-Links können über folgende Abfrage ermittelt werden: `select * from dba_db_links;`

Die Abfrage der Remote-Datenbank erfolgt über das Anhängen von @<DB_LINK> an eine normale Abfrage. Zum Beispiel:

```
select user from dual@remote.world;
```

oder

```
select * from abakus@db3;
```

wenn `abakus` der Name einer Tabelle, einer View oder eines Synonyms auf der Remote-Datenbank ist und `db3` der Name eines Database Link.

Ein Beispiel für das Anlegen eines Database Link (Fixed User Database Link):

```
create database link db3 connect to user51 identified by tz4ut using 'tomate';
```

- `db3` ist hier der Name des Database Link, der angelegt werden soll
- `user51` der vorgegebene User auf der Remote-Datenbank
- `tz4ut` das Passwort dieses Users user51
- `tomate` ist der Servicename der Remote-Datenbank entsprechend dem Eintrag in der lokalen tnsnames.ora

DB-Architektur

 Zurück zu " ORACLE Datenbank Administration " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Anmeldung "

Übersicht Datenbankinstanz

In Oracle besitzt jede Datenbank eine eigene Instanz der DBMS-Software. Das ist unterschiedlich zu einigen anderen DBMS-Systemen, wo eine Instanz mehrere Datenbanken unterstützt.

SID System-Identifer der Oracle-Datenbankinstanz

Hintergrundprozesse

- DBWR (Database Writer) Schreibt in der SGA modifizierte Datenblöcke zurück in die Datenbank.
- SMON (System Monitor) Überwacht die Wiederherstellung der Datenbank bei einem Neustart. Ferner registriert dieser Prozess freiwerdende Bereiche in der Datenbank und vereinfacht somit deren Wiederbelegung.
- LGWR (Log Writer) Schreibt die im Rahmen von Transaktionen anfallenden Protokollinformationen in die zugehörigen Plattenbereiche (Redo-Log-Einträge).
- CKPT (Checkpoint) Generiert die sogenannten Checkpoints, zu denen modifizierte Datenblöcke aus der SGA in die Datenbank zurückgeschrieben werden.
- PMON (Process Monitor) Überwachung der Benutzerprozesse. Freigabe der Ressource von abgebrochenen Benutzerprozessen.

Jede Oracle-Instanz besitzt einen eigenen Speicherbereich. Diese sogenannte SGA (System Global Area) ist wiederum in mehrere Bereiche aufgeteilt.

Diese Speicherbereiche werden während der Initialisierung der Instanz angelegt und können in der Größe zur Laufzeit nicht geändert werden.

Database Buffer Cache

In diesem Speicherbereich werden die gerade benötigten Datenblöcke vorgehalten. Da üblicherweise nicht alle Daten gleichzeitig in diesen Puffer passen, findet ein permanenter Ein- und Auslagerungsprozess zwischen aktuell benötigten und länger nicht mehr gebrauchten Daten statt. Hierbei werden die am längsten ungenutzten Puffer aus diesem SGA-Bereich ausgelagert (LRU-Algorithmus, LRU = Least Recently Used). Zur Erinnerung: zum Wegschreiben von Änderungen aus diesem Puffer war der Prozess DBWR zuständig.

Redo Log Buffer

Die während einer Transaktion anfallenden Protokollinformationen werden in diesem Puffer zwischengespeichert.

Shared Pool

SQL-Befehle jeglicher Art, Funktionen oder Prozeduren werden in diesem Pool zwischengelagert, wobei diese hier gelagerten Abfragen direkt ausführungsfähig sind, d.h. sie werden hier mitsamt ihren Ausführungsplänen gespeichert. Ähnlich wie beim Database Buffer Cache wird auch dieser Bereich nach dem LRU-Algorithmus verwaltet, d.h., häufig benutzte Abfragen oder Prozeduren stehen direkt zur Ausführung zur Verfügung.

Dictionary-Views über den Zustand der Datenbank

Neben den Dictionary-Tabellen USER_*, ALL_* und DBA_* gibt es noch eine weitere Gruppe von Dictionary-Views. Diese geben über den Zustand der Datenbank Auskunft. Diese Views haben den Präfix V$. Sie sind hauptsächlich für die Administration wichtig.

Anzeigen aller Sessions, die gerade aktiv sind oder zuletzt aktiv waren:

```
SELECT * FROM V$SESSION
```

Anzeigen von Informationen über die aktuelle Datenbank-Instanz.

```
-- Datenbank-Name, auf welchem Server die Datenbank läuft, Oracle-Version,
-- seit wann die Datenbank aktiv ist, ob aktuell Logins möglich sind
-- und in welchem Status sich die Datenbank befindet.
SELECT INSTANCE_NAME, HOST_NAME, VERSION, STARTUP_TIME,
       LOGINS, DATABASE_STATUS
FROM V$INSTANCE
```

Anzeige der Versionen der Komponenten:

```
SELECT * FROM V$VERSION
```

Es gibt noch einige weitere Dictionary-Views, die zu keiner der oben genannten Gruppen passen:

Wenn man die exakte Version einer Oracle-Installation ermitteln will, dann läßt man sich am besten die Database-Properties anzeigen.

```
SELECT * FROM database_properties
WHERE PROPERTY_NAME = 'NLS_RDBMS_VERSION'
```

 Zurück zu " ORACLE Datenbank Administration " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Anmeldung "

Anmeldung

 Zurück zu " DB-Architektur " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Dateien "

Local Naming

Beim Local Naming werden die Descriptoren verwendet, die in der Datei tnsnames.ora eingetragen sind. Diese Datei steht in $ORACLE_HOME/network/admin.

Wenn man sich als User "scott" (mit dem Passwort "tiger") an der Datenbank "testdb" anmelden möchte, dann muss man den folgenden Connect-String in einem Client-Tool eingeben:

```
connect scott/tiger@testdb
```

Wenn der Datenbankname bei der Installation in der Windows-Registry hinterlegt wurde, dann kann man diesen auch weglassen:

```
connect scott/tiger
```

Wenn man das Kennwort nicht in lesbarer Form eingeben will, dann kann man es (z.B. bei SQLPLUS) auch in dem Connect-String weglassen. Es erscheint dann eine Eingabeaufforderung, über die man das Passwort ohne Darstellung auf dem Bildschirm erfassen kann:

```
connect scott@testdb
connect scott
```

Wenn der User das sysdba-Recht hat, dann kann er sich auch als Administrator anmelden. Nur wenn man als Administrator angemeldet ist, kann man bestimmte administrative Veränderungen an der Datenbank vornehmen z.B. eine Datenbank löschen.

```
connect scott/tiger@testdb as sysdba
```

Der Befehl "connect" darf auch abgekürzt werden als "conn"

```
conn scott/tiger@testdb
```

Anmelden im SQLPLUS aus einem DOS-Fenster:

```
c:\>sqlplus "scott/tiger@testdb"
```

Andere Möglichkeiten der Namensauflösung

- **local Naming** wurde oben beschrieben.
- **Directory Naming** verwendet eine Namensauflösung durch einen zentralen LDAP-Server.
- **Easy Connect Naming** verwendet einen Connect-String in der Form: CONNECT username/Passwort@host:port /service-name. Die Parameter port und service-name sind optional. Bei dieser Methode wird die Datei tnsnames.ora nicht verwendet.
- **External Naming** Es gibt noch weitere Möglichkeiten der Namensauflösung durch Oracle-fremde Tools.

Local Naming ist die default-Einstellung nach der Installation.

 Zurück zu " DB-Architektur " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Dateien "

Dateien

 Zurück zu " Anmeldung " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Datenbank starten "

Arten von Datenbank-Dateien

Die physikalische Datenbank besteht aus vier Typen von Dateien:

- Parameter-Dateien: Dateien, in denen Parameter das Hochfahren der Datenbank-Instanz und der Client-Verbindung abgelegt sind.
- Control-Dateien: Steuer-Dateien, in denen abgelegt wird, welche Datenbank-Dateien zu der physikalischen Datenbank gehören
- Tablespace-Dateien: Dateien, in denen die Daten des System-Katalogs und der User-Tabellen gespeichert sind.
- Redolog-Dateien: Informationen über Datenänderungen. Diese werden benötigt für Backup und Recover.

Die Tablespace-, Redolog- und Control-Dateien sind nach der Standard-Installation unter Windows in einem einzigen Verzeichnis angelegt: C:\Oracle\oradata\<SID>

Datenspiegelung

Aus Sicherheitsgründen sollten die Dateien der Oracle-Datenbank auf mehrere Festplatten gespiegelt gespeichert werden. Das ist einerseits möglich durch den Einsatz von RAID-Platten, die hardwaremäßig die Spiegelung vornehmen. Die Datenbank-Dateien kann man auch von der Oracle-Datenbank auf mehreren Platten spiegeln lassen. Man sollte aber darauf achten, dass die Verbindung vom Server zu diesen Platten schnell genug ist (möglichst über den Datenbus und nicht über eine langsame Netzwerkverbindung)

Offline-Sicherung

Zusätzlich sollten diese Dateien regelmäßig gesichert werden. Am einfachsten ist eine Sicherung der Dateien, wenn gerade keine Instanz mit dieser Datenbank verbunden ist. Eine Solche Sicherung kann man jederzeit wieder zurückspielen, man muss nur darauf achten, dass die Dateien vollständig sind und von der selben Version stammen. Danach kann man die Instanz wieder starten, und mit der zurückgesicherten Version weiterarbeiten.

Online-Sicherung

Oft müssen Datenbanken 7 Tage zu je 24 Stunden verfügbar sein. Alle Datenbank-Dateien kann man auch während des laufenden Betriebes sichern. Dafür werden spezielle Tools eingesetzt (BACKUP, RECOVER, ARCH-Prozess)

Parameter-Dateien

Die Parameter- Dateien liegen jeweils unter dem Pfad, in dem die Datenbank installiert wurde. Dies kann ein beliebiger Pfad sein und ist z.B bei einer Unix-/Linux- Installation in der Umgebungsvariablen ORACLE_HOME abgelegt. Zur Abbildung dieses "Home" Verzeichnisses und dessen Variabilität wird im Folgenden den Pfaden diese Variable in der Windows- Notierung vorangestellt.

Init.ora bzw. Init<SID>.ora

Datei: %ORACLE_HOME%\database\init<SID>.ora

In früheren Versionen waren hier die Initialisierungs-Parameter eingetragen. Seit der Version 8 steht hier nur noch ein Verweis auf eine Datei mit den Parametern drin, und zwar auf:

%ORACLE_HOME%\..\admin\<SID>\pfile\init.ora

Inhalt: Parameter der Datenbank und der Datenbank-Instanz.

tnsnames.ora

Datei: %ORACLE_HOME%\network\ADMIN\tnsnames.ora

Inhalt: Verbindungsdaten für den Client. Hier müssen alle Server eingetragen sein, auf die der Client zugreifen kann.

listener.ora

Datei: %ORACLE_HOME%\network\ADMIN\listener.ora

Inhalt: Parameter für den Listener-Prozess auf dem Server, die die Verbindung zu den Clients bedient.

sqlnet.ora

Datei: %ORACLE_HOME%\network\ADMIN\sqlnet.ora

Inhalt: Parameter der Netzwerkverbindung

Es gibt noch eine Vielzahl von weiteren Parameter-Dateien, die man anlegen kann, doch diese drei sind die wichtigsten.

Weitere Parameterdateien

- %ORACLE_HOME%\sqlplus\ADMIN\init.sql SQL*PLUS-Parameter

Tablespace-Dateien

Eine Tablespace-Datei ist ein Bereich auf der Festplatte, der vom Betriebssystem für die ORACLE-Datenbank zur Verfügung gestellt wird. Die Oracle-Datenbank verwendet einen oder mehrere Tablespace-Dateien zur Speicherung der Daten in dem Tablespace.

Die Datei wird in Extents aufgeteilt. Die Größe eines Extents beträgt ein ganzzahliges Vielfaches eines Betriebssystem-Blocks. Jeder Extent beinhaltet mehrere Datenblöcke. Die Größe eines Datenblocks ist in der init.ora - Datei eingetragen (Parameter db_block_size) und beträgt normalerweise 8K. Dieser Parameter wird beim Anlegen einer Database einmal festgelegt und kann danach nicht mehr verändert werden.

Bei Lese- und Schreib-Zugriffen wird immer ein ganzer Block gelesen oder geschrieben. Es wird nie auf einzelne Bytes der Speicherplatte zugegriffen. Bei Schreib-Zugriffen wird ein Block in die SGA gelesen. Dort werden einzelne Werte oder Sätze aus dem Block verändert. Der DBWR-Prozess hat die Aufgabe, in regelmäßigen Abständen, alle geänderten Blöcke wieder auf die Speicherplatte zu schreiben.

Aufbau eines Datenblocks

Der Parameter PCTFREE gibt an, wieviel Prozent des Datenbereichs frei bleiben muss für Datenänderungen (UPDATE's). Dieser Bereich wird nicht für INSERT's verwendet. Wenn der Block durch INSERT's mit Daten bis zur PCTFREE-Grenze gefüllt wurde, dann wird er als "voll" markiert. Wenn dann durch UPDATE oder DELETE-Anweisungen wieder freier Platz entsteht, dann wird dieser Block aber nicht gleich wieder als "Frei" markiert. Erst wenn der Füllgrad unter die PCTUSED -Marke kommt, erst dann wird dieser Block wieder als "Frei" markiert und kann beim nächsten INSERT wieder neue Datensätze aufnehmen. Die Adressen von allen "freien" Blöcken werden in der Freispeicherliste notiert. PCTUSED sorgt dafür, dass ein Block nicht zu häufig in die Freispeicherliste eingetragen wird und wieder von dort entfernt werden muss.

Die Parameter PCTFREE und PCTUSED werden beim Erstellen eines Tablespace einmal für alle Datenblöcke festgelegt. Sie können auch nachträglich geändert werden. Wenn die Daten selten geändert werden, dann kann man PCTFREE / PCTUSED auf z.B. 5/80 festlegen. (Gute Speicherplatzausnutzung) Wenn die Daten sehr oft geändert werden, dann sollte man PCTFREE / PCTUSED auf z.B. 20/40 festlegen. (Es wird viel freier Platz gelassen, um die Performance der Schreib-Zugriffe zu verbessern)

Row-Fragmentierung

Die Parameter PCTFREE und PCTUSED haben die Aufgabe, Row-Migration so weit wie möglich zu vermeiden.

ROW Migration entsteht, wenn ein Datensatz durch ein UPDATE so groß wird, dass er nicht mehr in den Datenbereich des Blocks passt. Dann wird ein Verweis auf einen neuen Block an die Stelle eingetragen. Beim Lesen muss dann der Original-Block gelesen werden und der neue Block. Row-Chaining ist erforderlich, wenn die Record-Länge eines Satzes größer ist, als der Datenbereich eines Blocks. Dann muss dieser Datensatz auf einen zweiten Block fortgeschrieben werden.

Wenn der zweite Block auch nicht ausreicht, dann wird ein dritter, vierter ... Block dafür verwendet. So entsteht eine Kette von Blöcken, die zur Speicherung eines einzelnen Datensatzes benutzt werden. Row-Chaining kann man nur dadurch verhindern, wenn man beim Anlegen der Database die db_block_size entsprechend größer festlegt.

Über die Data-Dictionary-View v$datafile kann man die Größe der einzelnen Tablespace-Dateien anzeigen lassen.

```
select bytes, blocks, block_size, name from v$datafile;

     BYTES     BLOCKS BLOCK_SIZE NAME
---------- ---------- ---------- ---------------------------------------
 287309824      35072       8192 C:\ORACLE\ORADATA\ORACLE\SYSTEM01.DBF
 256901120      31360       8192 C:\ORACLE\ORADATA\ORACLE\RBS01.DBF
   9437184       1152       8192 C:\ORACLE\ORADATA\ORACLE\TEST2.DBF
   5242880        640       8192 C:\ORACLE\ORADATA\ORACLE\TEST3.DBF
```

weitere Data-Dictionary-Views zur Speicherplatzverwaltung:

```
dba_data_files
dba_extents
dba_free_space
dba_segments
dba_tablespaces
```

Rollback-Segmente

Rollback-Segmente sind spezielle Segmente in einem Tablespace, die Datenänderungen protokollieren.

Wenn eine Transaktion eine Änderung an den Daten vornimmt, dann wird die Änderung zunächst an dem Datensegment vorgenommen, aber die alte Version des Satzes wird in dem Rollback-Segment vermerkt. Solange die ändernde Transaktion ihre Änderung noch nicht durch COMMIT freigegeben hat, bekommen alle anderen lesenden Transaktionen die alte Version des Datensatzes zu lesen. Diese Informationen werden dann aus den Rollback-Segmenten gelesen. Wenn die ändernde Transaktion ihre Änderung durch COMMIT freigibt, bekommen andere lesende Transaktionen die neue Version des Satzes zu lesen. Nur solche Transaktionen, die ihren Lesezugriff noch vor dem COMMIT-Zeitpunkt begonnen haben, bekommen immer noch die alte Version des Satzes zu lesen, denn alle Lesezugriffe werden als Snapshot ermittelt d.h. als Sicht auf die Datenbank zu einem bestimmten Zeitpunkt. Wenn die Informationen in den Rollback-Segmenten nicht mehr gebraucht werden, können sie wieder überschrieben werden. Durch den System-Parameter UNDO-RETENTION wird festgelegt, wie lange die Informationen in den Rollback-Segmenten nach ihre Freigabe noch erhalten bleiben sollen für Lesezugriffe, die vor der Commit-Schreibung mit ihrem Zugriff begonnen haben.

Snapshot too old

Falls ein Programm z.B. mit einem Select alle vorhandenen Sätze aus einer Tabelle auslesen muss und für jeden Satz eine zeitaufwändige Verarbeitung ausführen muss, dann kann es vorkommen, dass für dieses Programm auf Informationen aus den Rollback-Segmenten zugegriffen will, die aber bereits überschrieben wurden, weil die UNDO-RETENTION-Zeit schon vorbei ist. Dann bricht diesen Programm ab mit der Fehlermeldung „ORA-01555: Snapshot too old“. In diesem Fall kann man die UNDO-RETENTION-Zeit hochsetzen. Wenn das Programm drei Stunden braucht, dann muss die UNDO-RETENTION-Zeit auf 3 * 60 * 60 = 10800 gesetzt werden, d.h. alle Informationen in den Rollback-Segmenten bleiben nach ihrer Freigabe noch drei Stunden erhalten. Erst dann dürfen sie überschrieben werden.

```
ALTER SYSTEM SET UNDO_RETENTION = 10800;
```

Bei einer so hohen UNDO-RETENTION-Zeit wird viel Platz für die Rollback-Informationen gebraucht.

Nach einer Standard-Installation der Oracle-Datenbank gibt es nur den SYSTEM-Tablespace, in dem sowohl die User-Daten, als auch die Rollback-Daten abgelegt werden. Um die Dictionary-Tabellen nicht unnötig zu belasten und um das Recovery zu erleichtern, sollte man die User-Tabellen in eigenen Tablespaces ablegen. Ebenso ist es empfehlenswert, mindestens einen eigenen Tablespace anzulegen, in denen ausschließlich Rollback-Segmente gespeichert werden. Oracle empfielt, die Tablespace für die Rollback-Segmente auf einer anderen Platte zu plazieren, als die Tablespace in denen die Daten-Segmente gespeichert werden. Das fördert Parallelzugriffe auf Daten- und Rollback-Informationen.

Ab Version 10 spricht man von UNDO-Segmenten und UNDO-Tablespaces.

Neue Rollback-Segmente anlegen

Um neue Rollbacksegmente anzulegen, müssen diese einem bestimmten Tablespace zugewiesen werden. Danach müssen die Rollback-Segmente ONLINE gesetzt werden.

```
CREATE PUBLIC ROLLBACK SEGMENT RBS0 TABLESPACE RBS
STORAGE ( OPTIMAL 4096K );
CREATE PUBLIC ROLLBACK SEGMENT RBS1 TABLESPACE RBS
STORAGE ( OPTIMAL 4096K );
```

```
ALTER ROLLBACK SEGMENT "RBS0" ONLINE;
ALTER ROLLBACK SEGMENT "RBS1" ONLINE;
```

Falls eigene Rollback-Segmente angelegt werden, dann müssen diese in der Parameter-Datei init.ora bekanntgegeben werden z.B.

```
rollback_segments = ( RBS0, RBS1 )
```

Redolog-Dateien

Redolog-Dateien speichern Informationen über alle Datenänderungen in den Tablespace-Dateien. Sie können im Falle eines Speichermediums-Fehlers zur Rekonstruktion der Daten herangezogen werden. Es müssen mindestens zwei Redolog-Dateien (mindestens zwei Gruppen mit jeweils mindestens einer Datei) für jede Database definiert sein. Jede Redolog-Datei sollte zusätzlich auf mehrere Platten gespiegelt werden.

Die Änderungen an den TS-Dateien werden immer in eine Redolog-Datei ausgegeben, bis diese voll ist. Dann führt das System einen Logswitch durch, d.h. es wird eine neue Redolog-Datei begonnen. Da aber nur eine begrenzte Anzahl von Redolog-Dateien vorhanden ist, wird nach dem Beschreiben der letzten Datei, wieder mit der ersten fortgesetzt. Die bisherigen Einträge der Datei, werden dabei überschrieben. Damit die Redolog-Informationen durch das Überschreiben nicht verloren gehen, hat der ARCH-Prozess die Aufgabe, nach einem Logswitch die gerade fertig geschriebene Redolog-Datei zu archivieren.

Nach der Installation ist der ARCH-Prozess nicht aktiv, d.h. die Datenbank befindet sich im NOARCHIVELOG-Modus. In diesem Zustand kann nur manuelles Offline-Backup und Recovery ausgeführt werden. (s. Seite 3) Um die Datenbank-Funktionen BACKUP und RECOVER ausführen zu können, muss die Datenbank in den ARCHIVELOG-Modus versetzt werden. Dann nimmt der ARCH-Prozess seine Arbeit auf.

Redolog-Gruppen

Da die Redolog-Dateien elementare Voraussetzung für ein Recover nach einem Plattencrash sind, ist es dringend empfohlen, die Redolog-Dateien von der Datenbank gespiegelt zu beschreiben.

Anzeigen der aktuell vorhandenen Redolog-Dateien:

```
select * from v$logfile;
```

```
    GROUP# STATUS  MEMBER
---------- ------- ----------------------------------
         1         C:\ORACLE\ORADATA\ORCL\REDOC1.LOG
         1         D:\ORACLE\ORADATA\ORCL\REDOD1.LOG
         2         C:\ORACLE\ORADATA\ORCL\REDOC2.LOG
         2         D:\ORACLE\ORADATA\ORCL\REDOD2.LOG
         3         C:\ORACLE\ORADATA\ORCL\REDOC3.LOG
         3         D:\ORACLE\ORADATA\ORCL\REDOD3.LOG
```

```
SQL> select * from v$log;
```

```
GROUP#  THREAD#  SEQUENCE#    BYTES  MEMBERS ARC STATUS
------ -------- ---------- -------- -------- --- --------
     1        1          7  1048576        2 NO  INACTIVE
     2        1          8  1048576        2 NO  CURRENT
     3        1          6  1048576        2 NO  INACTIVE
```

Redolog-Gruppen verändern

Man kann nur Loggruppen ändern, die gerade nicht aktiv sind. Einen Wechsel zur nächsten Loggruppe kann man erzwingen durch den Befehl:

```
alter system switch logfile;
```

Einrichten einer zusätzlichen Redolog-Gruppe:

```
ALTER DATABASE orcl ADD LOGFILE GROUP 4
('C:\ORACLE\ORADATA\ORCL\REDOC4.LOG',
  'D:\ORACLE\ORADATA\ORCL\REDOD4.LOG' ) SIZE 1M;
```

Redolog-Gruppe erweitern um eine zusätzliche Datei auf der E-Platte:

```
ALTER DATABASE orcl ADD LOGFILE
MEMBER 'E:\ORACLE\ORADATA\ORCL\REDOE1.LOG' reuse to group 1;
```

Redolog-Gruppe löschen:

```
ALTER DATABASE orcl DROP LOGFILE GROUP 4;
```

Eine einzelne Datei aus einer Redolog-Gruppe löschen:

```
ALTER DATABASE orcl DROP LOGFILE
MEMBER 'E:\ORACLE\ORADATA\ORCL\REDOE1.LOG';
```

Eine einzelne Datei umbenennen:

```
ALTER DATABASE orcl RENAME FILE 'E:\ORACLE\ORADATA\ORCL\REDOXX.LOG'
TO 'E:\ORACLE\ORADATA\ORCL\REDOE1.LOG';
```

Die Datenbank entfernt die Dateien nur aus ihrem Verzeichnis. Das Entfernen der physischen Datei auf der Betriebssystem-Ebene muss man selber machen.

Ebenso muss beim Umbenennen die Datei auf Betriebssystem-Ebene bereits unter dem neuen Namen existieren.

Empfehlungen für die Handhabung von Redolog-Dateien

- Redolog-Dateien sollten auf einer anderen Platte gespeichert werden, als die Tablespace-Dateien
- Jede Redolog-Gruppe sollte aus mindestens 2 Dateien bestehen
- Um Zugriffskollisionen zwischen dem LGWR und dem ARCH-Prozessen zu vermeiden, sollte sich das Verzeichnis, in den die Redolog-Dateien archiviert werden, auf einer anderen Platte befinden, als die Redolog-Dateien geschrieben werden.
- Wenn viele Datenänderungen ausgeführt werden, dann sollten die Redolog-Dateien entsprechend größer dimensioniert werden. Fall eine Redolog-Gruppe voll ist und der ARCH-Prozess die nächste Redolog-Gruppe noch nicht fertig archiviert hat, dann muss das System so lange warten, bis die Archivierung abgeschlossen ist.
- Bis Version 9 ist maximal eine fünffache Spiegelung der Redolog-Dateien möglich.
- Größe von Redolog-Dateien:

```
  50 KB   Minimalwert
       500 KB  default-Wert
       1 MB    das ist immer noch „klein“
       50 MB   schon ganz gut für produktive Anwendungen
       500 MB  bei großen Anwendungen keine Seltenheit
```

Control-Dateien

Die Control-Dateien enthalten die Informationen über alle physikalischen Datanbankdateien

Sie beinhalten die Verwaltungs- und Strukturinformationen

- Datenbankname
- Namen der Tablespace-Dateien
- Namen der Redolog-Dateien
- die aktuelle LOG-Sequenznummer
- weitere Informationen, die für ein Recover erforderlich sind.

Die Control-Dateien sollten ebenfalls auf unterschiedlichen Platten abgelegt werden.

Die aktuell vorhandenen Control-Dateien kann man sich ausgeben lassen mit:

```
select * from v$controlfile;
```

```
STATUS  NAME
------- -------------------------------------
        C:\ORACLE\ORADATA\ORCL\CONTROL01.CTL
        C:\ORACLE\ORADATA\ORCL\CONTROL02.CTL
        C:\ORACLE\ORADATA\ORCL\CONTROL03.CTL
```

Control-Dateien wiederherstellen

Falls die Controldateien zerstört wurden (und alle anderen Dateien noch ok sind), dann kann man die Control-Dateien neu erstellen lassen mit folgender Befehlsfolge:

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE
SQL> SHUTDOWN IMMEDIATE
```

Dadurch wird in dem Trace-Verzeichnis ein Script erstellt, mit dem man die Control-Dateien sichern kann. Die Trace-Datei findet man in dem Verzeichnis:

```
C:\oracle\admin\<SID>\udump
```

Datei mit dem letzten Datum suchen. In der Datei muss man die Header-Zeilen und die Kommentar-Zeilen entfernen. Dann kann man die Datei z.B. speichern unter c:\con_neu.sql und ausführen:

```
SQL> @c:\con_neu.sql
```

Nun sind die Controldateien neu angelegt worden.

 Zurück zu " Anmeldung " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Datenbank starten "

Datenbank starten

Zurück zu " Dateien " | Hoch zu " Inhaltsverzeichnis " | Vor zu " Benutzerverwaltung "

Verwalten der Datenbank-Instanz

Eine Oracle-Datenbank besteht aus den Dateien und einem Programm, das im Arbeitsspeicher aktiv ist. Das aktive Programm bezeichnet man auch als Oracle-Instanz.

Die Instanz kann gestartet werden

- mit dem Enterprise Manager
- dem SQL*Plus-Befehl STARTUP
- bei Windows durch Starten der Dienste

Beim Starten der Instanz werden verschiedene Zustände durchlaufen:

- SHUTDOWN
- NOMOUNT
- MOUNT
- OPEN

Diese Zustände können alle nacheinander durchlaufen werden bis zum Status OPEN, in dem die Instanz die normale Arbeit ausführen kann. Man kann die Status auch einzeln durchlaufen, um bestimmte administrative Arbeiten ausführen zu können.

Der aktuelle Status der Datenbank kann ermittelt werden mit dem Befehl:

```
select status from v$instance;
```

DB-Status SHUTDOWN

Sowohl die physische Datenbank, als auch die Instanz ist heruntergefahren.

In diesem Zustand kann sich nur ein User als SYSDBA anmelden. Die Anmeldung wird bestätigt mit der Meldung:

Bei einer nicht hochgefahrenen Instanz angemeldet.

Zum Hochfahren der Datenbank müssen noch drei weitere Phasen durchlaufen werden:

Zustand NOMOUNT

Zustand MOUNT

Zustand OPEN

Eine gestoppte Datenbank kann komplett hochgefahren werden (und alle Phasen des Start-Prozesses zu durchlaufen) mit dem Befehl:

startup

oder

startup normal

DB-Status NOMOUNT

Eine Database kann vom Administrator in verschiedene Status versetzt werden:

Der erste Schritt zum Starten einer Datenbank ist das Starten der Instanz im Zustand NOMOUNT.

```
Startup nomount
```

In diesem Zustand ist die Instanz hochgefahren, hat aber noch keine Verbindung zur physischen Database aufgenommen.

Nur in diesem Zustand kann eine neue Datenbank erstellt werden.

In diesem Zustand sind nur Befehle zulässig, die auf keine Datenbank-Dateien zugreifen:

```
show sga;
```

```
Total System Global Area   73701404 bytes
Fixed Size                    75804 bytes
Variable Size              56770560 bytes
Database Buffers           16777216 bytes
Redo Buffers                  77824 bytes
```

Folgende Dictionary-Views sind selektierbar:

```
v$controlfile
v$instance
v$option
v$parameter
v$session
v$sga
v$version
```

DB-Status MOUNT

Um die physische Datenbank zu öffnen, wird sie in den Status MOUNT versetzt.

Wenn die Database im Zustand SHUTDOWN ist, dann kann sie in den Status MOUNT gebracht werden durch den Befehl:

```
startup mount
```

Wenn die Database im Zustand NOMOUNT ist, dann kann sie in den Status MOUNT gebracht werden durch den Befehl:

```
alter database mount
```

Um die Datenbank in den Zustand MOUNT zu versetzen, müssen die Controldateien gelesen werden, um zu ermitteln, welche Dateien zur physischen Datenbank gehören.

Falls die Control-Dateien nicht gefunden werden oder inkonsistent sind, dann kann die Datenbank nicht in den MOUNT-Zustand versetzt werden.

In diesem Zustand sind weitere Dictionary-Views selektierbar:

```
v$database
v$controlfile
v$datafile
v$logfile
v$datafile_header
```

Zugriffe auf die Benutzer- oder Dictionary-Tabellen ist in diesem Zustand noch nicht möglich

DB-Status OPEN

Um das reguläre Arbeiten mit der Datenbank zu ermöglichen, muss die Datenbank noch geöffnet werden.

Wenn die Database im Zustand SHUTDOWN ist, dann kann sie in den Status OPEN gebracht werden durch den Befehl:

```
STARTUP OPEN
```

Wenn die Datenbank im Zustand NOMOUNT oder MOUNT ist, dann kann sie in den Status OPEN gebracht werden durch den Befehl:

```
ALTER DATABASE OPEN
```

Erst jetzt sind die Benutzer- und Dictionary-Tabellen zugreifbar. Nur in diesem Zustand können sich User ohne SYSDBA-Privileg an der Datenbank anmelden.

Ist eine Datenbank aus einem Onlinebackup zurückgespielt worden, werden die ArchivedRedoLogs mit

```
RECOVER DATABASE
```

in die Datenbank eingepflegt. Sind alle ArchivedRedoLogs eingepflegt muß die Datenbank noch mit

```
ALTER DATABASE OPEN RESETLOGS
```

geöffnet werden. Da die OnlineRedoLogs nicht mehr aktuell sind müßen sie neu initialisiert werden.

Weitere Startmöglichkeiten

```
STARTUP FORCE
```

Entspricht einem SHUTDOWN ABORT und anschließendem STARTUP NORMAL

```
STARTUP RECOVER
```

Öffnen der Datenbank nach vorherigem Recover

```
STARTUP READ ONLY
```

Öffnen der Datenbank nur für Lese-Operationen.

DB-Status RESTRICT

Um Wartungsarbeiten an der Datenbank auszuführen, kann die Datenbank im Zustand RESTRICT geöffnet werden. Dieser Zustand ist derselbe, wie der OPEN-Zustand, aber es dürfen sich nur User mit dem Systemprivileg "restricted Session" anmelden.

```
STARTUP RESTRICT
```

Voraussetzung: Die Datenbank muss im Zustand SHUTDOWN sein. Sie wird nun hochgefahren in den OPEN Zustand. Der Administrator kann jetzt auf alle Tabellen zugreifen, aber ein Anmelden anderer Benutzer ist ausgeschlossen. Genauer ausgedrückt: Nur Benutzer mit dem Systemprivileg "restricted session" können sich jetzt anmelden.

"normale" Benutzer, die sich in diesem Zustand anmelden wollen, erhalten die Fehlermeldung:

```
ERROR:
ORA-01035: ORACLE only available to users with RESTRICTED SESSION privilege
```

Falls die Datenbank aus dem laufenden Betrieb in den RESTRICT-Zustand versetzt werden soll, dann kann das mit dem folgenden Befehl geschehen:

```
ALTER SYSTEM ENABLE RESTRICTED SESSION
```

Achtung: Falls jedoch noch andere Benutzer noch mit der Datenbank verbunden sind, dann werden diese Verbindungen nicht getrennt. Nur Neuanmeldungen von "normalen" Benutzern sind ausgeschlossen. Falls die anderen Benutzer zwangsweise abgemeldet werden sollen, dann ist ein Shutdown erforderlich.

Die Datenbank kann für den normalen Betrieb wieder freigegeben werden durch den Befehl:

```
ALTER SYSTEM DISABLE RESTRICTED SESSION
```

SHUTDOWN-Befehl

Beim Herunterfahren der Datenbank muss angegeben werden, wie mit anderen noch mit der Datenbank arbeitenden Benutzern und den noch aktiven Transaktionen umgegangen werden soll:

```
SHUTDOWN
```

oder

```
SHUTDOWN NORMAL
```

wartet solange, bis alle aktiven Benutzer mit "EXIT" ihre Arbeiten beendet haben. Ein erneuter LOGON ist nicht möglich.

```
SHUTDOWN TRANSACTIONAL
```

Alle Transaktionen dürfen ihre Arbeiten bis zum nächsten Commit-Punkt fortsetzen, die geänderten Daten werden gespeichert und dann wird der Benutzer von der Datenbank getrennt. Falls eine Transaktion "klemmt", weil sie z.B. auf eine andere Transaktion wartet z.B. in einem Deadlock, dann kann die Datenbank nicht herunterfahren. Wenn die datenbank durch einen SHUTDOWN TRANSACTIONAL heruntergefahren werden konnte, dann sind die Daten in jedem Fall konsistent.

```
SHUTDOWN IMMEDIATE
```

Nun wird nicht mehr bis zum nächsten Commit gewartet, sondern alle Aktivitäten der noch aktiven Benutzer werden sofort gestoppt. Alle Datenänderungen werden mit Rollback zurückgerollt. Auf diese Weise können auch Deadlock-Situationen aufgelöst werden. Alle Daten sind durch den Rollback in einem konsistenten Zustand.

```
SHUTDOWN ABORT
```

Bei diesem Befehl wird sofort der Kontakt der Instanz zur physischen Datenbank unterbrochen. Falls andere Transaktionen noch aktiv waren, dann sind die Daten inkonsistent. Beim nächsten Hochfahren muss erst mal ein Recover ausgeführt werden. SHUTDOWN ABORT hat die selbe Wirkung, wie eine Unterbrechung der Stromversorgung.

Bei Ausführung des Shutdown-Befehls wird folgende Meldung ausgegeben:

```
Datenbank geschlossen.
Datenbank abgehängt.
ORACLE-Instanz heruntergefahren.
```

Dadurch wird exakt protokolliert, wie die Datenbank die einzelnen Phasen des Startup-Befehle in genau der umgekehrten Reihenfolge durchläuft. Wenn als Letztes die Instanz heruntergefahren ist, dann befindet sich die Datenbank im SHUTDOWN-Zustand.

Sollte beim Herunterfahren der Datenbank ein Problem auftreten, dann besteht die Möglichkeit für den Administrator, sich in einer weiteren Session anzumelden und mit dem Befehl

```
SHUTDOWN IMMEDIATE
```

oder

```
SHUTDOWN ABORT
```

das Herunterfahren zu beschleunigen.

In der folgenden Tabelle sind die relevanten Aktionen dargestellt.

Modus für das Herunterfahren	**A**	**I**	**T**	**N**
Lässt neue Anmeldungen zu	Nein	Nein	Nein	Nein
Wartet, bis aktuelle Sessions beendet sind	Nein	Nein	Nein	Ja
Wartet, bis aktuelle Transaktionen beendet sind	Nein	Nein	Ja	Ja
Erzwingt Checkpoint und schliesst Dateien	Nein	Ja	Ja	Ja

A = ABORT, I = IMMEDIATE, T = TRANSACTIONAL, N = NORMAL,

 Zurück zu " Dateien " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Benutzerverwaltung "

Benutzerverwaltung

Zurück zu " Datenbank starten " | Hoch zu " Inhaltsverzeichnis " | Vor zu " Tablespace verwalten "

Durch Installation angelegte Benutzer

Wenn eine Oracle-Datenbank installiert wird, dann werden die folgenden Benutzer eingerichtet:

User-ID	durch die Installation vergebenes Passwort	Verwendung
SYS	CHANGE_ON_INSTALL	Systemstart und Betrieb
SYSTEM	MANAGER	Administration

Der User **SYS** hat die DBA-Rolle und das SYSDBA-Recht. Er kann sich an der Datenbank-Instanz anmelden, wenn diese noch nicht hochgefahren ist. SYS darf den STARTUP-Befehl ausführen. SYS kann nicht nur die Datenbank administrieren, sondern darf auch alle Tabelleninhalte der anderen Benutzer anzeigen lassen. Der User SYS kann selber Tabellen erstellen. Diese Möglichkeit sollte nur dafür genutzt werden, um Tabellen zu erstellen, die vom System oder für die Systemverwaltung genutzt werden. Benutzer die dieses Privileg erlangen wollen müssen in der Betriebssystemgruppe `dba` sein und müssen sich mit dem Kommandozeilenzusatz `as sysdba` anmelden. Aus Sicherheitsgründen sollte dieses Konto ge`LOCKED` sein, damit keine Loginmöglichkeit mehr über den `Listener`, trotz bekanntem Passwort, besteht. Ein lokaler Login ist weiterhin möglich.

Der User **SYSTEM** hat ebenfalls die DBA-Rolle und das SYSDBA-Recht. Er kann jedoch - im Vergleich zum SYS - nicht die Tabellen der anderen Benutzer anzeigen. Auch das SYSTEM Konto sollte aus Sicherheitsgründen ge`LOCKED` sein, damit keine Loginmöglichkeit mehr über den Listener, trotz bekanntem Passwort, besteht. Ein lokaler Login ist weiterhin möglich.

Die oben angegebenen Passwörter werden bei einer Installation bis zur Version 9 automatisch vergeben. Ab der Version 10g muss man die Initial-Passwörter während der Installation selber bestimmen.

Weitere User-IDs, die bei der Installation zusätzlicher Komponenten erstellt werden:

User-ID	durch die Installation vergebenes Passwort	Verwendung
SCOTT	TIGER	Bis zur Version 8 wurde der User SCOTT immer als Beispiel-User mitinstalliert. Seit der Version 9 (?) wird er nur noch dann erstellt, wenn die Beispiel-Datenbank SAMPLE installiert wird. Handelt es sich bei der Instanz nicht um eine Test- Beispieldatenbank sollte der User, inklusiv aller Objekte, gelöscht werden. Dies gilt erst recht für Produktivdatenbanken.
SYSMAN	CHANGE_ON_INSTALL	für die Administration der Datenbank aus dem Oracle Enterprise Manager
DBSNMP	DBSNMP	für Zugriffe aus dem Oracle Enterprise Manager
CTXSYS	CTXSYS	für die Administration der Text-Extender-Funktionen

DIP	DIP	Administration der Directory Integration Platform (DIP), die Änderungen im OID (Oracle Internet Directory) mit Applikationen in der Datenbank synchronisieren kann
DMSYS	DMSYS	für die Administration der data mining-Funktionen
EXFSYS	EXFSYS	für die Administration der Expression-Filter-Indizierungs-Funktionen
MDDATA	MDDATA	für die Administration der Oracle Spatial- und Geodaten-Funktionen
MDSYS	MDSYS	für die Administration der Oracle Spatial- und interMedia-Locator-Funktionen
MGMT_VIEW	wird generiert	wird für die Kontrolle durch den Oracle Enterprise Manager benutzt
OLAPSYS	MANAGER	für die Administration der OLAP metadata-Strukturen
ORDPLUGINS	ORDPLUGINS	der Oracle interMedia User
ORDSYS	ORDSYS	der Oracle interMedia Administrator Account
OUTLN	OUTLN	für die Administration der Tuning-Statistiken
SI_INFORMTN_SCHEMA	SI_INFORMTN_SCHEMA	für die Administration der Oracle interMedia-Funktionen

Einen neuen Benutzer anlegen

Dem Benutzer muss ein Default-Tablespace zugewiesen werden. Falls dieser noch nicht existiert, oder falls der User einen eigenen Default-Tablespace erhalten soll, dann muss dieser zuvor eingerichtet werden.

```
create tablespace sjm
   DATAFILE 'c:\oracle\oradata\oracle\sjm.dbf' size 5M reuse
   DEFAULT STORAGE (INITIAL 10K NEXT 50K
                    MINEXTENTS 1 MAXEXTENTS 999)
   ONLINE;
```

```
create user jm               -- Die User-ID lautet: jm
   identified by geheim      -- Passwort
   default tablespace sjm
   temporary tablespace temp
   profile default
   account unlock            -- Der Account soll nicht gesperrt sein
   quota 1M on sjm           -- Der User darf 1MB Platz verbrauchen
;
```

Die Minimalangaben zum Einrichten eines neuen Users sind der Name des Users und sein Passwort.

Einige Berechtigungen für den User jm vergeben:

```
grant connect to jm;               -- Connect-Berechtigung (erst dann ist ein Connect möglich)
grant create procedure to jm;      -- Prozeduren erstellen
grant create trigger to jm;        -- Trigger erstellen
grant create sequence to jm;       -- Sequence erstellen
grant create public synonym to jm; -- Synonyme erstellen
grant drop public synonym to jm;   -- Synonyme löschen
grant execute on sys.sp1 to jm;    -- Prozedur sys.sp1 ausführen
grant select on pr_tab to jm;      -- Lesezugriffe auf Tabelle pr_tab
grant create tablespace to jm;     -- Tablespace erstellen
```

Anmeldung als der neu erstellte User jm:

```
connect jm/geheim;
```

Bei der **externen Autorisierung** übernimmt das Betriebssystem die Passwortprüfung für die Autentifizierung. In diesem Fall wird innerhalb der Oracle-Datenbank kein Passwort für einen Benutzer gespeichert bzw. überprüft. Der Benutzer muss lediglich der Oracle-Datenbank bekannt gemacht werden.

```
create user jmex
   identified externally
   default tablespace sjm
   temporary tablespace sjm
   profile default
   account unlock
   quota 1M on sjm
;
```

Passwort ändern:

```
alter user jm identified by tiger;
```

Das Passwort des eigenen Users kann man immer ändern. Das Passwort eines anderen Users kann nur der Administrator ändern.

Einen bestehenden Benutzer löschen

Sollten für den Benutzer bereits abhängige Daten existieren, muss die 'cascade' Option angegeben werden um alle Abhängigkeiten mitzulöschen.

```
drop user username cascade;
```

Systemprivilegien

Systemprivilegien sind Rechte zur Administration der Datenbank. Sie sind unabhängig von bestimmten Objekten (z.B. Tabellen, Spalten, Indices)

Welche Systemprivilegien hat ein User erhalten?

```
select * from user_sys_privs;
select * from dba_sys_privs;
select * from session_privs;
```

Objektprivilegien

Objektprivilegien sind Berechtigungen zur Administration bestimmter Datenbank-Objekte. Bei der Vergabe dieser Berechtigungen muss immer das betreffende Objekt genannt werden, auf das sich das Recht bezieht.

Welche Objektprivilegien hat ein User erhalten?

```
select * from all_tab_privs;
select * from user_tab_privs;
select * from dba_tab_privs;
```

Durch die Quota wird festgelegt, wie viel Platz ein Benutzer in Anspruch nehmen darf für die Tabellen, die er erstellt und mit Daten füllt. Die Quota kann ein Administrator ändern:

```
alter user jm quota 15M on stest;
```

Welche Quotas wurden vergeben:

```
select * from user_ts_quotas;
select * from dba_ts_quotas;
```

Welche Spaltenprivilegien wurden vergeben?

```
select * from all_col_privs;
select * from user_col_privs;
select * from dba_col_privs;
```

Welche Objektprivilegien hat er weitergegeben?

```
select * from all_tab_privs_made;
select * from user_tab_privs_made;
```

Welche Objektprivilegien hat er von anderen Usern erhalten?

```
select * from all_tab_privs_recd;
select * from user_tab_privs_recd;
```

Welche Spaltenprivilegien hat er erhalten?

```
select * from all_col_privs_recd;
select * from user_col_privs_recd;
```

Welche Spaltenprivilegien hat er weitergegeben?

```
select * from all_col_privs_made;
select * from user_col_privs_made;
```

Welche Quotas hat er erhalten?

```
select * from user_ts_quotas;
select * from all_ts_quotas;
select * from dba_ts_quotas;
```

Profil

Jeder User hat ein Profil, in dem verschiedene Ressource- und Passwort-Parameter festgelegt sind. Über die Ressource-Parameter kann ein Benutzer eingeschränkt werden in der Benutzung der Systemressourcen (z.B. maximal zulässige Anzahl Sessions, CPU-Nutzung) Die Passwort-Parameter legen z.B. fest, wie lange ein Passwort verwendet werden darf und die zulässige Anzahl der Fehleingaben. Welches Profil ein User hat, ist in DBA_USERS eingetragen.

```
select profile from dba_users where username = 'JM';
```

Welche Profile gibt es und welche Einstellungen sind z.B. in dem Profil 'DEFAULT' festgelegt?

```
select * from dba_profiles where profile = 'DEFAULT'
```

Der Systemadministrator kann auch weitere Profile anlegen. Beispiel:

```
create profile pr_develop limit
   password_reuse_max 10
   password_reuse_time 30
   sessions_per_user unlimited
   cpu_per_session unlimited
   cpu_per_call 3000
   connect_time 45
   ;
```

Rolle

Meistens gibt es in einem Unternehmen verschiedene Gruppen von Mitarbeitern, die bestimmte Berechtigungen haben. Beispiel: Softwareentwickler, Administratoren, Anwender. Jeder Oracle-User braucht eine Vielzahl von Berechtigungen, die davon abhängig ist, in welcher Gruppe er tätig ist. Damit nun diese vielen Einzel-Berechtigungen nicht bei jeder personellen Veränderung einzeln zugewiesen oder entzogen werden müssen, gibt es in Oracle das Rollen-Konzept.

Eine Rolle wird eingerichtet und die dafür vorgesehenen Berechtigungen werden dieser Rolle gegeben.

```
create role anwend_a;
grant select on a01.umsatz to anwend_a;
grant select, delete on a01.log to anwend_a;
grant execute on prog035 to anwend_a;
```

Dann wird diese Rolle an die einzelnen User vergeben.

```
grant anwend_a to jm;
```

Jedem User kann eine Default-Rolle zugewiesen werden. Beispiel:

```
alter user jm default role r_default;
```

Wenn ein User mehrere Rollen erhalten hat, dann kann er diese aktivieren und deaktivieren durch den Befehl SET:

```
set role r01, r02, r03; -- Nur diese Rollen werden aktiviert
set role all; -- Alle Rollen werden aktiviert
set role none; -- Alle Rollen werden deaktiviert
```

Welche Rollen sind in der aktuellen Session gerade aktiv?

```
select * from session_roles;
```

Welche Rollen gibt es?

```
select * from dba_roles;
```

Welche System- und Objektprivilegien wurden einer Rolle gegeben? Wurden einer Rolle noch weitere Rollen zugewiesen?

```
select * from role_sys_privs;
select * from role_tab_privs;
select * from role_role_privs;
```

Welche Rollen hat der User JM erhalten?

```
select * from dba_role_privs where grantee = 'JM';
```

Was machen die einzelnen User gerade?

```
select EXECUTIONS, USERS_EXECUTING, username, sql_text
from v$session se , v$sql sq
where se.sql_address = sq.address;
```

In der Katalog-View v$session sind alle gerade aktiven Sessions verzeichnet. In der View v$sql sind die zuletzt ausgeführten SQL-Statements eingetragen. Bei einer Verknüpfung dieser beiden Views kann man sich ausgeben lassen, welche SQL-Statements die einzelnen Benutzer ausgeführt haben und wie oft diese Statements ausgeführt wurden.

Übersicht über das User-Profil

```
select * from user_users;
select * from all_users;
```

```
select * from dba_users;
```

 Zurück zu " Datenbank starten " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Tablespace verwalten "

Tablespace verwalten

 Zurück zu " Benutzerverwaltung " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Backup und Recover "

Konfiguration von Tablespaces

Nach der Installation von Oracle gibt es nur den SYSTEM-Tablespace (bei Oracle Version < 10g), ab Oracle Version 10g gibt es mind. den SYSTEM und den SYSAUX Tablespace.

Es ist empfehlenswert, die System-Daten und die Anwendungsdaten in getrennten Tablespaces zu speichern.

Für die Systemdaten kann man eigene Tablespaces einrichten für

- Datadictionary (existiert schon)
- Tools
- Rollback-Segmente
- Sort-Bereich (temporärer Tablespace)

Für die Anwendungsdaten kann man Tablespaces einrichten für

- die einzelnen Fachgebiete
- Indices

Informationen von Tablespaces auslesen

```
    select
    file_name, tablespace_name,
    bytes/1048576, maxbytes/1048576, CASE WHEN maxbytes = 0 THEN 0 ELSE 100*bytes/maxbytes END,
    blocks/1024, maxblocks/1024, CASE WHEN maxblocks = 0 THEN 0 ELSE 100*blocks/maxblocks END,
    status, autoextensible, online_status
    from
    dba_data_files order by tablespace_name;
```

Neuen Tablespace erstellen

```
CREATE TABLESPACE user_ts
    DATAFILE 'c:\oracle\oradata\ora\userts.dbf' SIZE 10M;
```

Der Tablespace wird mit einer festen Größe von 10MB angelegt.

```
CREATE TABLESPACE user_ts
    DATAFILE 'c:\oracle\oradata\ora\userts.dbf' SIZE 10M
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 100K;
```

Die Extent-Verwaltung erfolgt bei diesem Tablespace lokal im Tablespace durch ein BITMAP und nicht über das Datadictionary. Alle Extents haben die selbe Größe.

```
CREATE TABLESPACE user_ts
    DATAFILE 'c:\oracle\oradata\ora\userts.dbf' SIZE 10M
    AUTOEXTEND ON NEXT 200K  MAXSIZE 200M;
```

Dieser Tablespace wird zunächst mit 10MB angelegt. Er kann sich bei Bedarf in Schritten von 200 KB bis zu einer Maximalgröße von 200MB selbst erweitern.

Tablespace erweitern

```
ALTER TABLESPACE user_ts1
```

```
    ADD DATAFILE 'c:\oracle\oradata\ora\userts1.dbf' SIZE 10M;
```

```
ALTER TABLESPACE user_ts2
    ADD DATAFILE 'c:\oracle\oradata\ora\userts2.dbf' SIZE 10M
    AUTOEXTEND ON NEXT 200K  MAXSIZE 200M;
```

Die beiden Tablespaces werden erweitert, indem eine weitere Datei hinzugefügt wird. Im ersten Fall ist es eine Datei mit fester Größe, im zweiten Fall ist es eine Datei, die sich bei Bedarf selbst erweitern kann.

```
ALTER TABLESPACE user_ts1
    ADD DATAFILE 'c:\oracle\oradata\ora\userts1.dbf' SIZE 10M
    AUTOEXTEND ON NEXT 200K  MAXSIZE 200M;
```

```
ALTER TABLESPACE user_ts2
    ADD DATAFILE 'c:\oracle\oradata\ora\userts2.dbf' SIZE 10M
    AUTOEXTEND OFF;
```

```
ALTER DATABASE
    DATAFILE 'c:\oracle\oradata\ora\userts2.dbf'
    AUTOEXTEND ON NEXT 200K MAXSIZE 100M;
```

Wenn der TEMP Tablespace erweitert werden soll, dann wird TEMPFILE anstelle von DATAFILE angegeben:

```
ALTER TABLESPACE temp
    ADD TEMPFILE 'c:\oracle\oradata\ora\temp2.dbf' SIZE 10M
    AUTOEXTEND ON NEXT 200K  MAXSIZE 200M;
```

Die AUTOEXTEND-Einstellung kann man nachträglich mit Hilfe des ALTER-DATABASE-Befehls verändern.

```
ALTER DATABASE
    DATAFILE 'c:\oracle\oradata\ora\userts1.dbf'
   resize 10M;
```

Die Tablespace-Datei wird wieder verkleinert auf 10 MB. Dieser Befehl kann nur dann ausgeführt werden, wenn das bestehende Datenvolumen nicht den abzutrennenden Speicherbereich verwendet.

```
ALTER TABLESPACE user_ts1 COALESCE;
```

Die einzelnen hintereinanderliegenden freien Extents in den Tablespace-Dateien werden zu ganzen Blöcken zusammengefasst. Dadurch können neue größere Extents in dem freien Platz angelegt werden.

Tablespace administrieren

Für bestimmte administrative Tätigkeiten müssen Tablespaces offline gesetzt werden.

In einer NOARCHIVELOG-Umgebung (d.h. es ist kein ARCH-Prozess aktiv, dementsprechend werden die Redolog-Dateien nicht archiviert):

```
ALTER TABLESPACE user_ts1 OFFLINE DROP;
```

```
ALTER TABLESPACE user_ts1 ONLINE;
```

In einer ARCHIVELOG-Umgebung (die Redolog-Dateien werden vom ARCH-Prozess archiviert):

```
ALTER TABLESPACE user_ts1 OFFLINE NORMAL;
ALTER TABLESPACE user_ts1 OFFLINE IMMEDIATE;
ALTER TABLESPACE user_ts1 OFFLINE TEMPORARY;
```

```
ALTER TABLESPACE user_ts1 ONLINE;
```

Schlüsselwort NORMAL: die Anweisung wird nur dann ausgeführt, wenn sich der Tablespace in einer Fehlersituation befindet.

Schlüsselwort IMMEDIATE: der Tablespace wird sofort in den OFFLINE-Modus gesetzt, auch wenn eine Transaktion diesen gerade benutzt. Beim ONLINE-Setzen muss ein Recover stattfinden.

Schlüsselwort TEMPORARY: die Dateien, die sich nicht in einer Fehlersituation befinden, werden durch einen Checkpoint gesichert. Beim ONLINE-Setzen muss ein Recover nur für die fehlerhaften Dateien stattfinden.

```
ALTER DATABASE DATAFILE <Dateiname> OFFLINE;
ALTER DATABASE DATAFILE <Dateiname> OFFLINE DROP;
ALTER DATABASE DATAFILE <Dateiname> ONLINE;
```

Man kann auch einzelne Datenbankdateien online / offline setzen. In einer NOARCHIVELOG-Umgebung muss der Parameter DROP angegeben werden. Dies kann es ggf. erforderlich machen ein recover auf die Datenbankdatei auszuführen, bevor sie wieder online gesetzt werden kann.

```
ALTER DATABASE RECOVER DATAFILE <Dateiname>;
```

Datenbankdatei umbenennen

1. Schritt: Tablespace OFFLINE setzen

2. Schritt: Datei auf Betriebssystemebene **kopieren** (Oracle schaut in beide Dateien nach)

3. Schritt: Neuen Namen im Oracle bekannt geben:

```
ALTER TABLESPACE user_ts
   RENAME DATAFILE <Dateiname-alt> TO <Dateiname-neu>;
```

oder:

```
ALTER TABLESPACE user_ts
   RENAME DATAFILE <Dateiname1-alt>, <Dateiname2-alt>, ...
   TO <Dateiname1-neu>, <Dateiname2-neu>, ... ;
```

4. Schritt: Tablespace ONLINE setzen

5. Kontrollieren:

```
SQL> select * from v$datafile;
...
Online
...
```

6. Wenn die neue Datenbankdatei Online ist kann die alte Datenbankdatei auf Betriebssystemebene gelöscht werden

Falls mehrere Datenbankdateien oder Dateien aus dem SYSTEM-Tablespace umbenannt werden sollen, dann können folgende Schritte ausgeführt werden:

1. Schritt: Datenbank in den MOUNT-Status versetzen

2. Schritt: Datei mit Betriebsystemmitteln umbenennen

3. Schritt: Neuen Namen im Oracle bekannt geben:

```
ALTER DATABASE RENAME FILE <Dateiname-alt> TO <Dateiname-neu>;
```

oder:

```
ALTER DATABASE RENAME
```

```
   FILE <Dateiname1-alt>, <Dateiname2-alt>, ...
   TO <Dateiname1-neu>, <Dateiname2-neu>, ... ;
```

4. Schritt: Datenbank in den OPEN-Status versetzen

Tablespace löschen

Falls der Tablespace leer ist (keine Tabellen, Indexe, Rollback-Segmente vorhanden):

```
DROP TABLESPACE user_ts;
```

Falls sich noch Objekte im Tablespace befinden:

```
DROP TABLESPACE user_ts INCLUDING CONTENTS;
```

Ab Version 9 können auch die Datendateien auf Betriebssystemebene mit gelöscht werden:

```
DROP TABLESPACE user_ts INCLUDING CONTENTS AND DATAFILES;
```

Falls sich noch Objekte im Tablespace befinden und von anderen Objekten Referenzen auf die Objekte im zu löschenden Tablespace existieren:

```
DROP TABLESPACE user_ts INCLUDING CONTENTS CASCADE CONSTRAINTS;
```

Ohne die Angabe "AND DATAFILES" müssen die Dateien noch manuell mit Betriebssystemmitteln gelöscht werden.

Temporary Tablespace

Wenn bei der Ausführung von SQL-Statements Sortierungen gefordert werden, und diese Daten für die SGA zu umfangreich sind, dann werden sie in einem Tablespace auf der Platte zwischengespeichert und sortiert. Dabei können kurzfristig viele Extents angefordert werden, die danach wieder freigegeben werden. Um die Speicherplatzverwaltung im SYSTEM-Tablespace und in den Daten-Tablespaces nicht unnötig zu belasten, sollten temporäre Tablespaces zur Aufnahme dieser Daten bereitgestellt werden.

Temporary Tablespace anlegen

```
CREATE TEMPORARY TABLESPACE user_temp_ts TEMPFILE <Dateiname> SIZE 10M;
```

Ein temporärer Tablespace hat ein TEMPFILE und KEIN DATAFILE.

Man kann auch einen "normalen" Tablespace in einen temporären Tablespace umwandeln:

```
ALTER TABLESPACE user_ts TEMPORARY;
```

Und man kann einen temporären Tablespace in einen "normalen" Tablespace umwandeln:

```
ALTER TABLESPACE user_temp_ts PERMANENT;
```

Datadictionary-Views für Tablespaces

Anzeige aller Tablespaces:

```
select * from user_tablespaces;
select * from dba_tablespaces;
```

Anzeige der Datendateien:

```
select * from dba_data_files;
select * from v$datafile;
```

Anzeige der Temp-Dateien:

```
select * from dba_temp_files;
```

Freiplatzverwaltung:

```
select * from dba_free_space;
```

Tablespaces mit AUTOEXTEND ON:

```
select * from filext$;
```

Anzeige der Objekte in den Tablespaces:

```
select * from dba_segments;
```

Wie groß ist der Redolog-Bereich? Redologs werden nicht in einem Tablespace gespeichert, sondern als "Archive-Log-Dateien" in einem dafür vorgesehenen Verzeichnis ausgegeben.

```
select group#, bytes / 1024 / 1024 Redo_mb from v$log;
```

 Zurück zu " Benutzerverwaltung " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Backup und Recover "

Backup und Recover

Zurück zu " Tablespace verwalten " | Hoch zu " Inhaltsverzeichnis " | Vor zu " National Language Support "

Backup-Strategie

Offline-Backup

Die Datenbank-Dateien werden gesichert, während die Datenbank nicht geöffnet ist.

Vorteile:

- einfache Handhabung
- das Backup kann mit Betriebsystem-Mitteln durchgeführt werden

Nachteile:

- die Datenbank muss heruntergefahren werden
- falls ein Recover erforderlich ist, dann kann nur die Sicherung wiederhergestellt werden. Alle danach erfolgten Änderungen können nicht automatisch nachgezogen werden.
- Die Datenbank kann nur als Ganzes wiederhergestellt werden.

Online-Backup

Während der Benutzung der Datenbank von Anwendern kann auch eine Datensicherung hergestellt werden. Diese Sicherung enthält aber keinen Snapshot, sondern die Änderungen, die während der Erstellung der Sicherung ausgeführt wurden, sind teilweise in der Sicherung enthalten.

Online-Sicherungen sind nur möglich, wenn auch die Redolog-Dateien durch den ARCH-Prozess gesichert werden.

Vorteile:

- Kein SHUTDOWN erforderlich zur Erstellung der Sicherung
- beim Recover kann eine Sicherung eingespielt werden und dann die Änderungen automatisch nachvollzogen werden bis zu einem bestimmten Punkt z.B. bis vor den Start eines bestimmten Programms
- Einzelne Tablespaces können gesichert werden

Nachteile:

- Das Erstellen einer online-Sicherung belastet die anderen Aktivitäten der Instanz
- ein Recover auf die Datensicherung alleine ist nicht möglich, da die Sicherung keine konsistenten Daten eines Snapshots enthält.

Offline-Backup

Beispiel-Script für die Durchführung eines offline-Backup (unter Windows)

```
set heading off
set feedback off
set underline off
set termout off

spool c:\backup\backup.bat

select 'copy ' || name   || '  c:\backup' from v$datafile;
select 'copy ' || name   || '  c:\backup' from v$controlfile;
select 'copy ' || member || '  c:\backup' from v$logfile;

spool off
```

```
shutdown immediate

host c:\backup\backup.bat

startup
```

Aufruf des Scripts:

```
SQLPLUS "sys/<Passwort> as sysdba" @c:\<Name des Scripts>
```

Zur Durchführung eines Recover müssen alle Dateien wieder an ihren Ursprungsort kopiert werden. Falls die Datenbank-Dateien auf unterschiedlichen Platten gespeichert waren, dann muss darauf geachtet werden, dass alle Dateien wieder an ihren richtigen Ort kopiert werden. Es darf keine einzige Datei ausgelassen werden. Am besten hebt man sich die generierte Backup-Datei auf oder man erstellt sich gleich eine geeignete Recover-Datei, mit der ein Recover automatisch abläuft.

Online-Backup

Ein Online-Backup ist nur zu gebrauchen, wenn die Redolog-Dateien über den ARCH-Prozess gesichert werden. Denn die einzelnen Dateien werden parallel zu den Änderungen der Datenbank erstellt. Bestimmte Änderungen sind in den Dateien bereits enthalten, andere noch nicht.

Wenn man später eine Online-Sicherung für das Recover verwenden will, dann müssen nach dem Einspielen der Sicherung die archivierten Redolog-Dateien ausgewertet werden, um alle Datenänderungen bis zu einem bestimmten Timestamp nachzuvollziehen. Erst dann ist ein bestimmter, konsistenter Zustand der Datenbank wieder hergestellt.

Ein Online-Backup kann durch folgende Befehlsfolge ausgeführt werden:

1. Beginn der Online-Sicherung dem System mitteilen

```
ALTER TABLESPACE <Name> BEGIN BACKUP
```

2. Mit Betriebssystem-Mitteln wird die Datei in das Backup-Verzeichnis kopiert

3. Das Ende der Online-Sicherung dem System mitteilen

```
ALTER TABLESPACE <Name> END BACKUP
```

Für die Durchführung von Online-Backups kann man sich auch Skripte generieren, die dann automatisiert ablaufen. Dabei muss man darauf achten, dass immer nur ein Tablespaces im Backup-Status ist. Für jeden Tablespace, der gesichert werden soll, müssen die oben genannten drei Schritte durchlaufen werden.

Control-Dateien wiederherstellen

Falls die Controldateien zerstört wurden (und alle anderen Dateien noch ok sind), dann kann man die Control-Dateien neu erstellen lassen mit folgender Befehlsfolge:

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE
```

oder

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE AS 'zu_speichernde_pfad_mit_name'
SQL> SHUTDOWN IMMEDIATE
```

Dadurch wird in dem Trace-Verzeichnis ein Script erstellt, mit dem man die Control-Dateien sichern kann. Die Trace-Datei findet man in dem Verzeichnis:

```
C:\oracle\admin\<SID>\udump
```

Datei mit dem letzten Datum suchen.

In der Datei muss man die Header-Zeilen und die Kommentar-Zeilen entfernen. Dann kann man die Datei z.B. speichern unter c:\con_neu.sql und ausführen:

```
SQL> @c:\con_neu.sql
```

Nun sind die Controldateien neu angelegt worden.

ARCH-Prozess

Der ARCH-Prozess hat die Aufgabe, die voll geschriebenen LOG-Dateien zu sichern. Der LGWR (Log-Writer-Prozess) überschreibt und hat nur eine bestimmte Anzahl von Log-Dateien zur Verfügung. Wenn die eine Datei voll ist, dann schreibt er die Log-Informationen in die nächste Datei weiter. Dabei wird der alte Inhalt der Log-Datei überschrieben. Der ARCH-Prozess sorgt dafür, dass die Log-Dateien nach ihrem Beschreiben gesichert werden.

Der ARCH-Prozess wird aktiviert durch:

Einträge in der Datei init.ora: (Beispiel)

```
log_archive_start = true
log_archive_dest = C:\oracle\oradata\ora1g\archive
log_archive_format = %%ORACLE_SID%%T%TS%S.ARC
```

Dann muss der ARCH-Prozess noch aktiviert werden. Das geht nur, wenn die Datenbank in der MOUNT-Phase ist, (nicht in der OPEN-Phase)

```
shutdown immediate;
startup mount;
alter database archivelog;
alter database open;
```

Nun kann man die Arbeit des ARCH-Prozesses beobachten.

Die Data-Dictionary-Views geben über seine Aktivitäten Auskunft:

```
V_$ARCHIVED_LOG
V_$ARCHIVE_DEST
V_$ARCHIVE_PROCESSES
```

Man kann einen Logswitch erzwingen, auch wenn die aktuelle Logdatei noch nicht voll ist mit dem Befehl

```
alter system switch logfile;
```

Recover

Recover allgemeine Bemerkungen

Vor dem Starten eines Recover sollte immer eine genaue Fehleranalyse stehen. Dabei können grundsätzlich zwei Fehlerarten unterschieden werden:

logische Fehler

- Datenfehler durch fehlerhafte Programme
- Ein Programm oder Skript ist in seiner Verarbeitung abgebrochen und hinterläßt die Daten in einem inkonsistenten Zustand.
- Operator-Fehler z.B. ein Programm zur Erhöhung der Gehälter wurde aus Versehen zwei mal gestartet
- Anwender oder Administratoren haben aus Versehen wichtige Tabellen oder Datensätze gelöscht
- es stehen keine ausreichenden Systemressourcen zur Verfügung (Systemüberlastung, Tablespace-Dateien sind voll, Rollbacksegmente sind zu klein, Deadlocks)

technische Fehler

- Stromausfall
- Hardwarefehler z.B. bei den Speicherplatten
- Die Ausführung von Tools ist fehlgeschlagen, so dass sich die Daten in einem inkonsistenten Zustand befinden

Recover planen

Wenn ein Datenfehler festgestellt wurde, dann sollten die einzelnen Schritte für das Recover genau geplant werden.

Erst wenn

- die Fehlerursache genau eingegrenzt ist und
- der Umfang der erforderlichen Recover-Massnahme genau ermittelt ist

ist es sinnvoll, mit dem Recover zu beginnen.

Recover ausführen mit einer offline-Sicherung

Offline-Sicherungen können nur als Ganzes verwendet werden. Einzelne Teile oder einzelne Dateien können nicht wiederhergestellt werden. Es müssen alle beteiligten Dateien einschließlich dem System-Tablespace, den Control-Dateien und den Redolog-Dateien aus der Sicherung übernommen werden. Alle Datenänderungen, die seit dem Zeitpunkt der Erstellung der offline-Sicherung ausgeführt wurden, müssen wiederholt werden.

Recover ausführen mit einer online-Sicherung

Einzelne Dateien können wiederhergestellt werden. Eine viel granularere Steuerung ist möglich. Die Änderungen an den Daten, die seit dem Herstellen der Sicherung ausgeführt wurden sind, können durch Auslesen der Log-Dateien nachvollzogen werden, ohne dass die Anwendungsprogramme neu ablaufen müssen. Für das Recover mit online-Sicherungen gibt es eine große Vielzahl von Möglichkeiten, auf die in diesem Seminar nicht eingegangen werden kann. Beispielhaft soll das recovern einer einzelnen Tablespace-Datei dienen, das auf der nächsten Seite erläutert ist.

Recover Beispiel 1

Es wurde eine Datei, gelöscht, in der die Daten eines Tablespace mit Anwendungstabellen gespeichert waren. Es handelt sich um einen Tablespace, dessen Anlegen noch in den zur Verfügung stehenden LOG-Dateien verzeichnet ist.

1. Beim Hochfahren der Datenbank wird ausgegeben, dass eine Datei fehlt.

```
SQLPLUS  "SYS/<Passwort> as SYSDBA"
STARTUP
```

Ausgabe:

```
ORA-01110: Datendatei 3: c:\ORACLE\ORADATE\ORCL\USERTS.DBF'
```

Die Datenbank konnte nicht geöffnet werden. Sie befindet sich im Status MOUNT

```
ALTER DATABASE DATAFILE 3 OFFLINE;
ALTER DATABASE OPEN
```

Nun ist die Datenbank geöffnet. Man kann auf alle Objekte zugreifen, ausser auf die Daten in dem defekten Tablespace.

Nun wird der Name des Tablespace ermittelt, der den Datafile 3 verwendet:

```
SELECT tablespace_name from dba_extents where file_id = 3;
```

Es handelt sich um den Tablespace "USERS"

```
RECOVER TABLESPACE USERS;
```

Die Datenbank kann selbständig auf die Daten der Sicherung zugreifen und die fehlende Datei wieder herstellen. Nun ist der Tablespace wieder in Ordnung.

```
ALTER TABLESPACE USERS ONLINE;
```

Anmerkung: Falls der Tablespace vor einem Jahr angelegt worden ist und die Log-Dateien nicht mehr vorhanden sind für den gesamten Zeitraum dazwischen, dann ist der oben skizzierte Ablauf nicht möglich. Es muss mindestend die Log-Datei vorhanden sein, die das Anlegen des Tablespace protokolliert und alle zeitlich nachfolgenden Log-Dateien.

Recover Beispiel 2

Es wurde eine Datei, gelöscht, in der die Daten eines Tablespace mit Anwendungstabellen gespeichert waren. Es steht eine Offline-Sicherung zur Verfügung und alle Log-Dateien seit der letzten Offline-Sicherung stehen auch zur Verfügung.

Dann kann man die betroffene Daten-Datei von der Offline-Sicherung kopieren und anschließend die Datenbank wieder hochfahren. Die Meldungen des STARTUP Befehls und des anschließenden Recover sind in dem nachfolgenden Script wiedergegeben:

```
SQL> startup
ORACLE-Instanz hochgefahren.
...
Datenbank mit MOUNT angeschlossen.
ORA-01113: Fur Datei '1' ist Datenträger-Recovery notwendig
ORA-01110: Datendatei 1: 'C:\ORACLE\ORADATA\ORA1G\SYSTEM01.DBF'
```

```
SQL> recover
Wiederherstellung des Datenträgers abgeschlossen.
```

```
SQL> alter database open;
Datenbank wurde geändert.
```

 Zurück zu " Tablespace verwalten " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " National Language Support "

NLS

 Zurück zu " Backup und Recover " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Import Export "

NLS

Oracle bietet verschiedene Möglichkeiten, eine Client-Session zu parametrisieren, um auf länderspezifische Besonderheiten Rücksicht zu nehmen. Beim nächsten Connect müssen Änderungen an den Session-Parametern erneut ausgeführt werden.

```
ALTER SESSION SET nls_language = 'German';
```

Dieser Parameter bestimmt die Sprache, in der Server-Meldungen ausgegeben werden.

```
ALTER SESSION SET nls_territory = 'Germany';
```

Durch diesen Parameter werden verschiedene andere Parameter verändert. So wird z.B. das Format der Datums- und Uhrzeit-Ausgabe, sowie die Zeitzone festgelegt. Diese Einzelparameter kann man anschließend trotzdem noch individuell verändern.

```
ALTER SESSION SET TIME_ZONE = '-1:0';
ALTER SESSION SET nls_date_format = 'dd.mm.yyyy hh24:mi:ss';
ALTER SESSION SET NLS_NUMERIC_CHARACTERS = '.,';
```

Das erste Zeichen der NLS_NUMERIC_CHARACTERS legt fest, ob ein Dezimalpunkt oder ein Komma ausgegeben werden soll. Das zweite Zeichen legt fest, mit welchen Zeichen die Abtrennung der Tausender-Stellen vorgenommen wird.

Beispiel:

```
SQL> ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ',.';

Session wurde geändert.

SQL> select to_char(1234.56,'999g999d999') zahl from dual;

ZAHL
------------
   1.234,560

SQL> ALTER SESSION SET NLS_NUMERIC_CHARACTERS = '.,';

Session wurde geändert.

SQL> select to_char(1234.56,'999g999d999') zahl from dual;

ZAHL
------------
   1,234.560
```

Anführungszeichen ausgeben

```
SELECT 'Er sagte: "Es reicht."' FROM dual;
SELECT 'das war' 's' FROM dual;
-- nach war kommen 2 Anführungszeichen ohne Leerzeichen dazwischen
```

 Zurück zu " Backup und Recover " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Import Export "

Import Export

 Zurück zu " National Language Support " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " Datenbank Tuning "

Es gibt das Kommandozeilenwerkzeug **exp** zum Exportieren einer Oracle Datenbank.

Dieser Export kann dann mit dem Kommandozeilenwerkzeug **imp** wieder eingespielt werden.

Ab der Version 10 kann Alternativ dazu auch der Enterprisemanager verwendet werden (Export und Import mit Hilfe des Webbrowsers, mir persönlich zu umständlich und unübersichtlich).

Bitte beachten sie, dass bei Oracle 8 und 9 die Datenbank (hinter dem @) angegeben werden muss. Bei der Version 10 ist das nicht erforderlich, hierbei muss allerdings der Datenbankname als Umgebungsvariable zur Verfügung stehen. z.B. unter Linux/Unix export ORACLE_SID=$dbname

Desweiteren besteht die Möglichkeit bei vielen Parametern y oder yes zu verwenden.

Grundsätzlich gibt es vier verschiedene Modi:

- full - vollständiger Export bzw. Import, meistens durch den User system durchgeführt.
- user - nur die Datenbankobjekte exportieren bzw. importieren, die diesem User gehören
- table - einzelne Tabellen exportieren bzw. importieren
- tablespace - einzelne Tablespaces exportieren bzw. importieren

exp

Syntax:

```
exp $username/$password@$dbname $expModes $expOpts
```

Hilfe:

```
exp help=y
```

$expModes:

```
full=y | owner=$username | tables=$tableName1,$tableName2 | tablespaces=$tableSpaceName1,$tableSpaceName2
```

$expOpts:

```
LOG=filename COMPRESS=Y|N ROWS=Y|N QUERY=SQL_string DIRECT=Y|N FEEDBACK=integer STATISTICS=COMPUTE|ESTIMATE|NONE INDEXES=
TRIGGERS=Y|N CONSISTENT=Y|N OBJECT_CONSISTENT=Y|N FLASHBACK_SCN=SCN_number FLASHBACK_TIME=DATE BUFFER=Integer RESUMABLE=Y
RESUMABLE_NAME=resumable_string RESUMABLE_TIMEOUT=integer
```

Beispiele:

Einen kompletten Export einer Oracle Datenbank kann beispielsweise hiermit erstellt werden:

```
exp system/$password@$dbname file=$exportFile.dmp log=$exportLogFile.log full=yes
```

Einen userspezifischen Export kann beispielsweise hiermit erstellt werden:

```
exp $username/$password@$dbname file=$exportFile.dmp log=$exportLogFile.log owner=$username
```

Falls der Export mit dem Benutzer system durchgeführt wird kann der User mit dem Parameter "owner" angegeben werden.

Ein Export von einer oder mehreren Tabellen kann beispielsweise folgendermaßen aus sehen:

```
exp $username/$password@$dbname file=$exportFile.dmp log=$exportLogFile.log compress=no
tables=$tableName1,$tableName2 statistics=none consistent=yes
```

Ein Export von Tablespace(s) kann beispielsweise folgendermaßen aus sehen:

```
exp $username/$password@$dbname file=$exportFile.dmp log=$exportLogFile.log compress=no
tablespaces=$tableSpaceName1,$tableSpaceName2 statistics=none consistent=yes
```

imp

Der Import funktioniert korrespondierend zum Export, d.h. es gibt auch wieder die vier Modi und etliche Optionen. Hinzu kommt, dass auch "in" einen anderen Benutzer importieren kann.

Syntax:

```
imp $username/password@dbname $impModes $impOpts
```

Hilfe:

```
imp help=y
```

$impModes:

```
FULL=Y | FROMUSER=$username1 TOUSER=$username2 TABLES=$tableName1,tableName2 | TABLES=$tableName1,tableName2 |TRANSPORT_T
```

$impOpts:

```
ROWS=Y|N | COMMIT=Y|N | FEEDBACK=integer | BUFFER=integer | IGNORE=Y|N | DESTROY=Y|N | INDEXES=Y|N | CONSTRAINTS=Y|N |
SKIP_UNUSABLE_INDEXES=Y|N | STREAMS_CONFIGURATION=Y|N | STREAMS_INSTANTIATION=Y|N GRANTS=Y|N | STATISTICS=always|safe|rec
TOID_NOVALIDATE=([schemaname.]typename[,[schemaname.]typename]...) | SHOW=Y|N | RESUMABLE=Y|N | RESUMABLE_NAME=resumable_
RESUMABLE_TIMEOUT=integer | COMPILE=Y|N
```

Beispiele: Einen kompletten Import einer Oracle Datenbank kann beispielsweise hiermit erstellt werden.

```
imp system/$password@dbname file=$exportFile.dmp log=$importLogFile.log full=yes ignore=yes
```

Datenbank Tuning

Zurück zu " Import Export " | Hoch zu " Inhaltsverzeichnis " | Vor zu " CharacterSet ändern "

Es gibt viele Bereiche für das Tuning einer Oracle-Datenbank.

Lock-Erkennung und -Behebung

Locks auf der Datenbank ermitteln

Ausgeben aller Sessions, die gerade von anderen Sessions blockiert werden.

```
select *
from v$session
where blocking_session is not null
```

Locks auf allen Objekten in der Datenbank anzeigen

```
SELECT a.session_id, a.oracle_username, a.os_user_name, b.object_name
FROM   v$locked_object a, sys.all_objects b
WHERE  b.object_id = a.object_id
ORDER BY 2, 3;
```

Locks nur für die Objekte des aktuellen Benutzers in der Datenbank anzeigen

```
SELECT a.session_id, a.oracle_username, a.os_user_name, b.object_name
FROM   v$locked_object a, sys.user_objects b
WHERE  b.object_id = a.object_id
ORDER BY 2, 3;
```

SQL-Statement-Cache der aktuellen Sessions anzeigen

```
SELECT se.username, se.osuser, sq.sql_text
FROM   v$sql sq, v$session se
WHERE  se.sql_address = sq.address
ORDER BY 1, 2;
```

Skript catblock.sql zur Anzeigen von Sperrungen

In $ORACLE_HOME\RDBMS\ADMIN\catblock.sql befindet sich ein Script zum Erstellen einiger System-Views, mit denen Locks angezeigt werden können. Folgende Views werden erstellt:

- DBA_KGLLOCK
- DBA_LOCKS
- DBA_LOCK
- DBA_LOCK_INTERNAL
- DBA_DML_LOCKS
- DBA_DDL_LOCKS
- DBA_WAITERS
- DBA_BLOCKERS

Blockiert jemand eine andere Transaktion?

```
SQL> select * from dba_waiters
;
WAITING_SESSION HOLDING_SESSION
--------------- ---------------
             13              19
```

Wer wird blockiert?

```
SQL> select * from dba_waiters
;
WAITING_SESSION HOLDING_SESSION
--------------- ---------------
             13              19
```

Wie lange wartet die Session 13 schon?

```
SQL> select session_id, LAST_CONVERT Sekunden, LAST_CONVERT/60 Minuten
from dba_locks where Session_id in (13, 19)
;
SESSION_ID   SEKUNDEN    MINUTEN
---------- ---------- ----------
        13       2011 33,5166667
        13       2011 33,5166667
        13       2011 33,5166667
        19       2057 34,2833333
        19       2057 34,2833333
        19       2061      34,35
```

Welche User arbeitet als Session 13 bzw. 19?

```
SQL> select sid, serial#, username from v$session where sid in (13,19)
;
       SID    SERIAL# USERNAME
---------- ---------- ------------------------------
        13         71 SCOTT
        19         35 SCOTT
```

Skript utllockt.sql zur Anzeige von wartenden Transaktionen

In $ORACLE_HOME\RDBMS\ADMIN\utllockt.sql findet man ein Script zur Anzeige von wartenden Transaktionen.

In diesem Script wird

- eine Tabelle erstellt,
- mit Daten gefüllt,
- angezeigt
- und dann wird die Tabelle wieder entfernt.

Beispiel für eine Anzeige von Sperren:

- Die Session 9 wartet auf die Session 8.
- Session 7 wartet auf Session 9
- Session 10 wartet ebenfalls auf Session 9

```
WAITING_SESSION   TYPE MODE REQUESTED    MODE HELD         LOCK ID1 LOCK ID2
----------------- ---- ----------------- ----------------- -------- --------
8                 NONE None              None              0         0
   9              TX   Share (S)         Exclusive (X)     65547     16
      7           RW   Exclusive (X)     S/Row-X (SSX)     33554440  2
      10          RW   Exclusive (X)     S/Row-X (SSX)     33554440  2
```

Kill Session

Die Session 19 blockiert die Session 13 schon seit 34 Minuten. Sie soll beendet werden. Bei kill session muss man immer die Session-Id und die Serial-Nummer in Anführungszeichen angeben.

```
SQL> alter system kill session '19, 35'
;
System wurde geändert.
```

Die Session 19 erhält zunächst keine Fehlermeldung. Erst wenn der nächste Befehl eingegeben wird, stellt SQL-PLUS fest, dass die Verbindung zur Server-Session nicht mehr existiert:

```
SQL> select * from intab;
select * from intab
```

```
*
FEHLER in Zeile 1:
ORA-00028: Ihre Sitzung wurde abgebrochen
```

 Zurück zu " Import Export " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " ORACLE Datenbank Administration "

CharacterSet ändern

 Zurück zu " Datenbank Tuning " |  Hoch zu " Inhaltsverzeichnis " |  Vor zu " ORACLE Datenbank Administration "

In diesem Kapitel wird beschrieben, wie man den Zeichensatz in einer Datenbank ändern kann. Der Zeichensatz der Datenbank legt fest, mit welcher Codierung die Text-Informationen auf der Festplatte gespeichert werden. Wenn z.B. ein w:ASCII-Zeichensatz mit deutschen Umlauten festgelegt ist, dann können in den Tabellen dieser Datenbank zwar alle deutschen Schiftzeichen gespeichert werden (incl. äöüß), jedoch nicht die Umlaute aller anderen europäischen Länder. Erst recht können keine Schriftzeichen, die nicht auf dem w:lateinischen Alphabet basieren, gespeichert werden. Wenn man für die Datenbank einen w:Unicode-Zeichensatz festlegt, dann können fast alle Schriftzeichen dieser Welt gespeichert werden, allerdings werden für alle seltener vorkommenden Schriftzeichen (also auch den deutschen Umlauten) mehr als ein Byte Speicherplatz verbraucht.

Eine (Rückwärts-) Konvertierung von UTF8 nach WE8MSWIN1252 ist nur mit Verlusten möglich, da WE8MSWIN1252 eine echte Untermenge von UTF8 ist. Beispielsweise sind die griechischen oder kyrillischen Buchstaben in UTF8 vorhanden, aber nicht in WE8MSWIN1252.

Unabhängig vom Zeichensatz der Datenbank ist der Zeichensatz eines Oracle-Clients. Wenn die Zeichensätze sich unterscheiden, dann werden die Daten konvertiert, soweit das möglich ist.

Vor der Umstellung sollte auf jeden Fall ein FullBackup der Datenbank erstellt werden, damit man diesen wieder zurückladen kann, wenn irgend etwas schief geht.

Als sys mit sysdba-Berechtigung via sqlplus an der OracleInstanz anmelden.

```
sqlplus "sys/password@myDB as sysdba"
```

CharacterSet der Datenbank überprüfen

```
select value from nls_database_parameters where parameter='NLS_CHARACTERSET';
```

bzw. um sämtliche Einstellungen zu sehen

```
select * from nls_database_parameters;
```

Die Datenbank anhalten

```
shutdown immediate;
```

Die Datenbank einhängen aber noch nicht öffnen

```
startup mount;
```

Bei manchen Versionen gab es an dieser Stelle einen Bug, bei dem ein Fehler nach "startup mount;" ausgegeben wurde. Falls dieser noch nicht behoben ist, kann folgender Befehl eventuell helfen "connect / as sysdba".

Session für Operation vorbereiten

```
alter system enable restricted session;
alter system set job_queue_processes=0;
```

Datenbank öffnen

```
alter database open;
```

CharacterSet ändern

```
alter database character set UTF8;
ORA-12712: Der neue Zeichensatz muss eine Obermenge des alten Zeichensatzes
sein
```

hiermit geht es dann

```
alter database character set internal_use utf8;
```

Datenbank wieder schließen und herunterfahren

```
shutdown immediate;
```

Datenbank wieder normal hochfahren

```
startup;
```

Nochmal

```
select * from nls_database_parameters;
```

ausführen und die Einstellungen überprüfen.

Getestet auf Oracle 8.1.7.3 unter Windows 2000 Server

Getestet auf Oracle 10.1.0.2.0 unter Linux (erfolgreich)

Getestet auf Oracle 10.2.0.4 unter Windows 2003 Server (erfolgreich, obwohl dies lt. Oracle SQL Reference 10g nicht mehr möglich sein sollte)

Getestet auf Oracle 11.1.0.7 unter AIX5L Server ; erfolgreich.

Getestet auf Oracle 11.2.0.1.0 unter Windows 2008 Enterprise Server ; erfolgreich.

Getestet auf Oracle 11.2.0.1.0 unter Oracle Enterprise Linux 6.4 ; erfolgreich.

Links

| Hoch zu " Inhaltsverzeichnis " | Vor zu " Autoren "

Verwandte Themen in Wikibooks

http://de.wikibooks.org/wiki/SQL
http://de.wikibooks.org/wiki/PL/SQL
http://de.wikibooks.org/wiki/Relationenalgebra_und_SQL

Verwandte Themen in Wikipedia

http://de.wikipedia.org/wiki/Oracle_(Datenbanksystem)
http://de.wikipedia.org/wiki/PL/SQL
http://de.wikipedia.org/wiki/Oracle_Real_Application_Cluster

Links zum sonstigen WWW

http://www.oracle.com
http://www.sysdba.de
http://www.aquafold.com
http://www.quest.com
http://web.dadanini.com:7980/books/Sql_in_21Tagen/
http://www.orafaq.com/wiki/Main_Page