Seab@er Software AG

Oracle Dokumentation

07 - PL / SQL

1. Vorwort

Diese Dokumentation ist entstanden, da ich beruflich mich mit Oracle beschäftigen musste. Was ich sehr gerne übernommen habe und es macht richtig Spaß mit Oracle zu arbeiten. Alle Informationen, die ich zusammentragen konnte, habe ich nun in dieser Dokumentation geschrieben. Ebenso sind meine Erfahrungen in diese Dokumentation eingeflossen.

Oracle wird auf Linux und Windows Servern in unserer Firma betrieben. Die Installation von Oracle wird für die Linux Server beschrieben, da eine Windows Installation nicht so aufwendig ist.

Diese Dokumentation wurde für die Oracle Datenbank 10G R2 und 11G R1 geschrieben und auch getestet.

Die Datenbank in der Version 11G R1 wurde in einer VMWare Session installiert und als Betriebssystem wurde Novel SLES 10 SP2 installiert.

Bei dem Betriebssystem und auch Oracle handelt es sich um die 32 Bit Version. Für die 64 Bit Version werden noch zusätzliche Softwarepakete gebraucht.

Bei Fragen und Anregungen bin ich unter folgender Mail Adresse zu erreichen:

uwe@seabaer-ag.de

2. SQL Befehle

2.1 Tabelle

2.1.1 Leere Tabelle erstellen

```
sql>create table hr.mitarbeiter (vname char(25), nname char(40), alter number(4));
```

Folgende Feldtypen gibt es in Oracle.

Feldtype	Wert	Beschreibung
char(xx)	Maximum 2000 bytes	Die Angabe xx gibt an, wie viele Zeichnen gespeichert werden können. Fixed-length strings. Space padded.
nchar(xx)	Maximum 2000 bytes	Wie char. Fixed-length NLS string- Space padded.
nvarchar2(xx)	Maximum 4000 bytes	Wie char. Variable-length NLS string.
varchar2(xx)	Maximum 4000 bytes	Wie char. Variable-legth string.
long	Maximum 2GB	Variable-length string.
raw	maximum 2000 bytes	Variable-length binary strings.
long raw	maximum 2GB	Variable-length binary strings.
number(p,s)	z.B. (7,2) = 12345,00	
numeric(p,s)	z.B. (5,3) = 12,123	
float		
dec(p,s)	z.B. (3,1) = 12,1	
decimal(p,s)	z.B. (4,1) = 123,1	
date		Kann Datum und Uhrzeit speichern.

2.1.2 Vorhandene Tabelle kopieren

Mit dem nachfolgenden Befehl wird die Struktur und deren Inhalt in eine neu zu erstellende Tabelle übernommen.

```
sql>create table <table_name> as select * from <table_name> where <feld_name> in(select <feld_name> from <table_name>);
```

2.1.3 Tabelle Partition

Mit der Partitionsmöglichkeit kann man große Datenmengen strukturiert ablegen und verwalten.

In dem nachfolgenden Beispiel werden die Daten nach dem Feld `bday` partitioniert.

```
sql>create table adrbook
 2 ( vname char(20) not null
 3 , nname char(30) not null
 4 , bday date not null
 5 , email char(100)
 6 )
 7 partition by range (bday)
 8 ( partition p_before_1_jan_1970 values less than
 9 (to_date('01-01-1962','dd-mm-yyyy'))
10 , partition p_before_1_jan_1980 values less than
11 (to_date('01-01-1965','dd-mm-yyyy'))
12 , partition p_before_1_jan_2007 values less than
13 (to_date('01-01-2008','dd-mm-yyyy'))
14 )
15 enable row movement;
```

Nun können wir die Werte eingeben.

```
sql>insert into adrbook values ('Ralf','Schmidt',to_date('15-07-1963','dd-
mm-yyyy'),'ralf.schmidt@web.de';

sql>insert into adrbook values ('Harry','Hirsch',to_date('08-05-2007','dd-
mm-yyyy'),'hhirsch@freenet.de';

Fehler in Zeile 1:
ORA-14400: Eingefügter Partitionsschlüssel kann keiner Partition zugeordnet
werden.
```

Es wurde bei der Erstellung der Tabelle eine starre Range Partition eingerichtet. Dieses läst sich nachhaltig beheben, indem man auf eine dynamische Intervall Partitionierung umschaltet. Es werden nun automatisch neue Partitionen angelegt. Diese Partitionen bekommen aber einen Systemgenerierten Namen.

```
sql>alter table adrbook set interval (numtoyminterval(1,'year'));
```

Für die Option `numtoyminterval` gibt es außer den Wert `year` auch noch `month`. Es gibt auch noch die Option `numtodsintervall` mit den Werten `day`, `hour`, `minute` und `second`.

Die Informationen über die Partitionen können mit dem nachfolgenden Befehl abgefragt werden.

```
sql>select partition_name, high_value from user_tab_partitions
 2  where table_name = 'ADRBOOK' order by partition_position;

PARTITION_NAME          HIGH_VALUE
P_BEFORE_1_JAN_1970     TO_DATE(' 1970-01-01 00:00:00', 'SYYYY-MM-DD HH24:
P_BEFORE_1_JAN_1980     TO_DATE(' 1980-01-01 00:00:00', 'SYYYY-MM-DD HH24:
P_BEFORE_1_JAN_2007     TO_DATE(' 2007-01-01 00:00:00', 'SYYYY-MM-DD HH24:
SYS_P41                 TO_DATE(' 2008-01-01 00:00:00', 'SYYYY-MM-DD HH24:
```

Die Systemgenerierte Partition kann man nachträglich umbenennen.

```
sql>alter table adrbook rename partition sys_p41 to p_before_1_jan_2008;
```

Man kann auch mehrere Partitions zu einer mit dem `merge` Befehl zusammen fügen.

```
sql>alter table adrbook
 2> merge partitions for (to_date('01-01-1970','dd-mm-yyyy')),
 3> for (to_date('01-01-1980','dd-mm-yyyy'))
 4> into partition p_before_1_jan_1990;
```

Nur bei den Feldtypen `date` oder `number` kann die Intervall Partitionierung eingeschaltet werden. Das nächste Beispiel zeigt es.

```
sql>create table city
 2>(lfdnr number(6) not null
 3>,city char(30) not null
 4>,plz number(5) not null)
 5>partition by range (plz)
 6>(partition plz_0 values less than (1000)
 7>,partition plz_1 values less than (2000)
 8>,partition plz_2 values less than (3000))
 9>enable row movement;

sql>alter table city set interval (1000);
```

Außer einer `Range` Partitionierung gibt es auch noch eine `List` Partitionierung. Bei diesem Beispiel würde das Anlegen eines Datensatzes mit dem Kürzel KS fehlschlagen.

```
sql>create table <table_name>
 2>(lfdnr number(6) not null
 3>,text char(50)
 4>,kuerzel varchar2(2))
 5>partition by list (kuerzel)
 6>(partition north values ('HH', 'HB', 'FL')
 7>,partition west values ('K', 'DU', 'OB'))
 8>enable row movement;
```

2.1.4 Tabellenstruktur anzeigen

```
sql>describe hr.mitarbeiter;
```

2.1.5 Tabelle umbenennen

```
sql>alter table <table_name> rename to <new_table_name>;
```

2.1.6 Tabelle löschen

```
sql>drop table <table_name>;
```

Mit dem nachfolgenden Befehl wird die Tabelle `unwiederruflich` gelöscht, sie kann dann auch nicht mehr mit dem `flashback` Befehl zurückgeholt werden.

```
sql>drop table <table_name> purge;
```

Mehrere Tabellen in einem Rutsch löschen, kann man mit einem PL-Block erledigen. Nachfolgend sind zwei Beispiele für so einen Block beschrieben.

```
sql>begin
 2> for a in (select ‘drop table ‘ ||table_name as tbl from user_tables
 3> where table_name like ‘SAL%’)
 4> loop
 5>  execute immediate a.tbl;
 6> end loop;
 7>end;
 8>/
```

```
sql>begin
 2> for a in (select table_name as tbl from user_tables
 3> where table_name like ‘SAL%’)
 4> loop
 5>  execute immediate (‘drop table ‘ ||a.tbl);
 6> end loop;
 7>end;
 8>/
```

2.1.7 Tabelle komprimieren

```
sql>create table <table_name> compress for all operations
 2>as select * from emp;

sql>alter table <table_name> compress for all operations;
```

2.1.8 Tabellenfeld hinzufügen

```
sql>alter table <table_name> add (<feld_name> <feld_type> <contrain>;
```

Soll mehr als ein Feld hinzugefügt werden, so wird nach der letzten Feldbeschreibung ein Komma gesetzt und dann wird das nächste zu erstellende Feld beschrieben.

```
Sql>alter table hr.mitarbeiter add column (plz char(5) NOT NULL, nickname
char(25), index number);
```

2.1.9 Tabelle Feldtyp ändern

```
sql>alter table <table_name> modify <feld_name> char(30);
```

2.1.10 Tabelle Feldname ändern

```
sql>alter table <table_name> rename column <feld_name> to <feld_name>;
```

2.1.11 Tabelle Kommentar angeben

```
sql>comment on table <table_name> is 'Hier steht der beschreibene Text';

sql>select table_name, comments from user_tab_comments;
```

2.1.12 Tabelle Feldname Kommentar angeben

```
sql>comment on column <field_name> is 'Hier steht ein Text';

sql>select table_name, column_name, comments from user_col_comments;
```

2.1.13 Tabelle in andere DB kopieren

```
sql>copy from scott/tiger@db1 to scott/tiger@db2
 2 insert newtable using select * from db1table;

sql>help copy

COPY
----

Copies data from a query to a table in the same or another
database. COPY supports CHAR, DATE, LONG, NUMBER and VARCHAR2.

COPY {FROM database | TO database | FROM database TO database}
           {APPEND|CREATE|INSERT|REPLACE} destination_table
           [(column, column, column, ...)] USING query

where database has the following syntax:
    username[/password]@connect_identifier
```

2.1.14 Tabelle verschieben

Eine Tabelle kann in einem anderen Tablespace verschoben werden. Sind in der Tabelle Felder mit dem type long, so funktioniert das veschieben nicht.

```
sql>alter table <table_name> move tablespace <new_tablespace>;
```

Wird eine Tabelle nur mit der move Option versehen, so werden die Table Segments neu aufgebaut.

```
sql>alter table <table_name> move;
```

2.1.15 Tabellenfeld löschen

```
sql>alter table <table_name> drop column <feld_name>;
```

2.1.16 Werte eingeben

```
sql>insert into hr.mitarbeiter (vname, nname, year) values ('Uwe',
  2>'Schimanski', 1961);
```

```
sql>insert into hr.mitarbeiter select * from hr.employes where id = 10;
```

```
sql>insert into hr.mitarbeiter values ('Uwe', 'Schimanski',
  2>to_date('03-NOV-09 11:40:00', 'DD-MON-YY HH24:MI:SS'));
```

2.1.17 Werte aus einer anderen Tabelle übernehmen

Werte aus einer anderen Tabelle können mit eine `select` Statement eingefügt werden.

```
sql>insert into hr.mitarbeiter (vname, nname, year)
  2>select first_name, last_name, year from employees;

sql>insert into hr.mitarbeiter (vname, nname, kuerzel, year)
 2>(select first_name, last_name,
 3>substr(last_name,1,1), year from employees);
```

Alle Werte aus einer anderen Tabelle können so übernommen werden.

```
sql>insert into hr.mitarbeiter select * from employees;
```

2.1.18 Join Abfrage

```
sql>select emp.empno, emp.ename, emp.deptno, dept.dname from emp, dept
where emp.deptno = dept.deptno;
```

2.1.19 Abfrage mit SQL-Kommentar

```
sql>select /*Abfrage1*/ vname from mitarbeiter;
```

2.1.20 Abfrage mit Bindevariable

```
sql>var a varchar2(10);
sql>exec :a := 'Uwe';
sql>select vname, nname from mitarbeiter where vname = :a;
```

2.1.21 Abfrage mit like

Bei einer Abfrage kann man auch nach einem Teilstring in einem Datenbankfeld suchen. Hierfür wird in der der `where` Abfrage die `like` Bedingung genommen. Das % Zeichen ist als Platzhalter wie das * zu sehen. Möchte man nur ein Zeichen als Variable definieren, so wird das _ genommen.

```
sql>select * from dba_users where username like 'S%';

sql>select * from dba_users where username like 'S_Y%';
```

2.1.22 Abfrage mit in

```
sql>select * from emp where empno in (10, 20);
```

2.1.23 Abfrage mit Sysdate

```
sql>select sql_text from v$sqlarea where to_date(last_active_time) in
  2>(select to_date(sysdate) from dual);
```

Anzahl der Tage abfragen.

```
sql>select sysdate - to_date('17.01.2012','DD.MM.YYYY') "Days" from dual;

DAYS
---------
126,52341

sql>select start_time, end_time, status from v$rman_backup_job_details
 2>where start_time > truncate(sysdate)-2;

START_TIME  END_TIME    STATUS
----------  ----------- ---------
2013-01-09  2013-01-09  COMPLETED
```

2.1.24 Alias Abfrage

```
sql>select p.empno, p.ename, p.deptno, t.dname from
  2>emp p, dept t
  3>where p.deptno = t.deptno;
```

2.1.25 Abfrage Substring

Möchte man einem Tabellenfeld nur eine bestimmte Anzahl an Zeichen auslesen, so geschieht dieses mit `substr`.

Im ersten Beispiel werden ab dem zweiten Zeichen 5 Zeichen ausgelesen.

```
sql>select substr(platform_name, 2, 5) "OS System" from v$database;

OS SYSTEM
---------
inux
```

Im zweiten Beispiel werden ab dem 10ten Zeichen alle Zeichen bis zum Ende ausgelesen.

```
sql>select substr(platform_name, 10) "OS System" from v$database;

OS SYSTEM
-----------
Bit for AMD
```

Im letzten Beispiel werden die letzten 3 Zeichen ausgelesen.

```
sql>select substr(platform_name, -3) "OS System" from v$database;

OS SYSTEM
---------
AMD
```

2.1.26 Count Abfrage

```
sql>select name, count(street) from city group by name;

NAME        COUNT
----------- -----
Goch            3
Düsseldorf      5
Hambach         2

3 Zeilen ausgewählt
```

2.1.27 Lower / Upper Abfrage

```
sql>select distinct lower(substr(owner,4)) from dba_tables
 2>where owner like 'SP%';
```

Alle Werte ausgeben, die Groß geschrieben sind.

```
sql>select model_file_spec from pdtable_113
 2>where model_file_spec = upper(model_file_spec);
```

2.1.28 Decode Abfrage

```
sql>select decode(backup_type, 'L','Archive LOG', 'D', 'Full', 'Incr.')
 2>from v$backup_set;
DECODE(BACKU
------------
Archive Log
Archive Log
Full
Archive Log

4 Zeilen ausgewählt
```

2.1.29 Abfrage mit Datum

```
sql>select start_time, status from jobs where
 2>start_time > to_date('21.02.2013 04:06:00', 'DD.MM.YYYY hh24:mi:ss');

START_TIME             STATUS
---------------------- ---------
21.02.2013 04:06:23    COMPLETED
21.02.2013 05:34:22    RUNNING
```

2.1.30 Minus Abfrage

```
sql>select model_id, name from model_catalog
 2>minus
 3>(select max(model_id), name from model_catalog where name = 'VW')
 4>group by name
 5>order by name, model_id;

MODEL_ID    NAME
----------- ---------
122536      Opel
122567      Opel
122876      Porsche
```

2.1.31 Null Abfrage

```
sql>select model_id, name from model_catalog where status is not null;
```

2.1.32 Update Table

Möchte man an einen vorhandenen Wert einen Wert hinzufügen, so geschieht das folgendermaßen.

```
sql>update <table_name> set <column_name> = '50' || <column_name>
 2>where <column_name> like 'LBA%';

sql>update pdtable_80_22 set pipe_support_no = '50' || pipe_support_no
 2>where pipe_support_no like 'LBA%';
```

Eingabe von Sonderzeichen.

```
sql>set define off
sql>update hr.mitarbeiter set descry = 'Mobil & Home' where id = 5;
```

Suchen und ersetzten.

```
sql>update <table_name> set <column_name> =
 2>replace (<column_name>,'was','mit')
 3>where <column_name> like '%path%';
```

2.1.33 Max Abfrage

Eine Abfrage mit einem Max Wert in der where Bedingung kann folgendermaßen gemacht werden.

```
sql>select gv.output from gv$rman_output gv,
 2>( select max(session_recid) as maxid
 3>from gv$rman_output) maxrecid
 4>where gv.session_recid = maxrecid.maxid;
```

Eine zweite Möglichkeit wäre:

```
sql>select output from gv$rman_output
 2>where session_recid = (select max(session_recid) from gv$rman_output);
```

2.1.34 Sum Abfrage

```
sql>select
2>(select sum(bytes)/1024/1024/1024 data_size from dba_data_files) +
3>(select nvl(sum(bytes),0)/1024/1024/1024 temp_size from dba_temp_files) +
4>(select sum(bytes)/1024/1024/1024 redo_size from sys.v_$log) +
5>(select sum(BLOCK_SIZE*FILE_SIZE_BLKS)/1024/1024/1024 control_files
6>from v$controlfile) “Size in GB”
7>from dual;

Size in GB
----------
17,7874
```

2.2 External Tables

Mit external Tables kann man Text Dateien in die Datenbank mit einbinden. Diese Text Dateien können als CSV-Dateien mit einem Trennzeichen zwischen den Daten, oder mit einer festen Spaltebreite in der Text Datei vorliegen. Diese Dateien werden mit dem Access Driver Treiber geladen. Die Tabellen werden mit dem Zusatz ORGANIZATION EXTERNAL erstellt. Ebenso ist es Möglich, Views und Synonyme zu erstellen.
Eine Analyse von External Tabellen ist nur mittels DBMS_STATS möglich. Hierzu muss ab der Version 10g der Parameter ESTIMATE_PERCENT auf den Wert NULL gestellt werden, ansonsten kommt es zu den nachfolgende Feherlmeldung:

```
ORA-2000: Unable to analyse TABLE "<owner>"."<table>", sampling on external
table is not supported.
```

2.2.1 Vorarbeiten

Damit der Zugriff auf die externen Daten funktioniert, muss in der Datenbank Directories angelegt werden. Das Data Directory muss angelegt werden, für die Log- und Bad-Dateien ist es optional.

```
sql>create or replace data_path as '/u01/ext/data';

sql> create or replace log_path as '/u01/ext/log';

sql> create or replace bad_path as '/u01/ext/bad';

sql>grant read on directory data_path to uws;

sql>grant write on directory log_path to uws;

sql>grant write on directory bad_path to uws;
```

2.2.2 Tabelle erstellen

In dem ersten Beispiel liegen die Daten in der Text Datei in Spalten vor.

```
sql>create table user_load
 2>(id char(4),
 3> first_name char(20),
 4> last_name char(30),
 5> city char(20),
 6> birthday date)
 7>organization external
 8> (type oracle_loader
 9>  default directory data_path
10>  access parameters
11>  (records delimited by newline
12>   fields (id char(2),
13>           first_name char(15),
14>           last_name char(20),
15>           city char(15),
16>           birthday char(10) date_format date mask "dd.mm.yyyy"
17>           )
18>   )
19> location ('user.dat')
20>);
```

Der Inhalt der Text Datei user.dat sieht folgendermaßen aus.

```
uws@tux>cat user.dat
10Harry          Hirsch              Waldsiedlung   16.02.1967
33Paulchen       Panther             Fernsehstudio  31.05.1953
98Karl-Heinz     May                 Wild-West      05.11.1912
```

In dem nächsten Bespiel sind die Daten in der Text Datei mit einem Simikolon voneinander getrennt.

```
sql>create table users_load
 2>(id number(4),
 3> first_name char(20),
 4> last_name char(25),
 5> city char(20),
 6> birthday date)
 7>organization external
 8>(type oracle_loader
 9> default directory data_path
10> access parameters
11> (records delimited by newline
12>  logfile log_path:'extern%a_%p.log'
13>  badfile bad_path: 'extern%a_%p.bad'
14>  fields terminated by ';'
15>  missing field values are null
16> )
17> location ('user1.dat','user2.txt')
18>)
19>reject limit unlimited
20>parallel;
```

Der Inhalt der Text Datei user1.dat und user2.txt sieht folgendermaßen aus.

```
uws@tux>cat user1.dat
32;Klaus;Vogel;Baumhausen;12.08.1958
114;Walter;Kraus;Waldsee;29.02.1988
54;Susanne;Mops;Berlin;18.12.1965

uws@tux>cat user2.txt
72;Bernd;Müller;Hamburg;04.08.1977
12;Anton;Meise;München;19.11.1932
```

Die Parameter haben folgende Bedeutungen:

Parameter	**Beschreibung**
TYPE	Hier wird der Treiber festgelegt. Standard ist der `Oracle_loader`. Mit diesem Treiber kann nur lesend auf die Dateien zugegriffen werden. Mit dem Treiber `Oracle_datapump` kann auch schreibend auf die Daten zugegriffen werden. Nur hier müssen die Daten in Binär Format vorliegen.
DEFAULT DIRECTORY	Hier wird der Ort der Dateien angegeben.
ACCESS PARAMETERS	Hier kann beschrieben werden, was aus den Dateien geladen werden soll.
LOCATION	Name der Dateien.
REJECT LIMIT	Anzahl der erlaubten verworfenen Datensätze.
PARALLEL	Das parallele Laden wird erlaubt, bei großen Datenmengen it das von Vorteil.

Beispiel einer Abfrage.

```
sql>select * from users_load order by id;

ID  FIRST_NAME    LAST_NAME   CITY        BIRTHDAY
--- ------------- ----------- ----------- ----------
12  Anton         Meise       München     19.11.1932
32  Klaus         Vogel       Baumhausen  12.08.1958
54  Susanne       Mops        Berlin      18.12.1965
72  Bernd         Müller      Hamburg     04.08.1977
114 Walter        Kraus       Waldsee     19.02.1988
```

2.2.3 Ändern

Die Eigenschaften der Tabellen kann man mit `alter table` ändern.

Default Directory

```
sql>alter table user_load default directory users_path;
```

Access Parameters

```
sql>alter table user_load access parameters (fields terminated by ':');
```

Location

```
sql>alter table user_load location ('liste.txt','liste2.dat');
```

Reject Limit

```
sql>alter table user_load reject limit 400;
```

Parallel

```
sql>alter table user_load noparallel;
```

2.2.4 External Tabelle laden

External Tabellen können auch in einer bestehenden Tabelle geladen werden.

```
sql>insert into users_intern (
 2>id,
 3>first_name,
 4>last_name,
 5>city,
 6>birthday)
 7>(select id,
 8> first_name,
 9> last_name,
10> city,
11> to_date(birthday, 'DD.MM.YYYY')
12> from users_load);
```

2.2.5 Infos

In den nachfolgenden Tabellen kann man sich die Informationen über die external Tables sich anzeigen lassen.

Für die Anzeige der Attribute gibt es folgende Tabellen:

USER_EXTERNAL_TABLES
ALL_EXTERNAL_TABLES
DBA_EXTERNAL_TABLES

Informationen über die Datenquellen kann man in den folgenden Tabellen finden:

USER_EXTERNAL_LOCATIONS
ALL_EXTERNAL LOCATIONS
DBA_EXTERNAL_LOCATIONS

```
sql>select table_name, location, directory_owner, directory_name
 2>from user_external_locations:

TABLE_NAME LOCATION  DIR DIRECTORY_NAME
---------- --------- --- --------------
user_load  user.dat  SYS DATA_PATH
users_load user1.dat SYS DATA_PATH
users_load user2.txt SYS DATA_PATH
```

```
sql>select table_name, type_owner, type_name, access_type
2>from user_external_tables;

TABLE_NAME  TYP TYPE_NAME      ACCESS_
----------- --- -------------- --------
USER_LOAD   SYS ORACLE_LOADER  CLOB
USERS_LOAD  SYS ORACLE_LOADER  CLOB
```

2.3 Datatype Long RAW

Möchte man eine Tabelle kopieren und in der Quelltabelle befindet sich ein Feld mit dem Datatype Long, so kann man die Tabelle mittels einer select Abfrage nicht erstellen. Es wird die Oracle Fehlermeldung ORA-00997 ausgegeben. Mit dem nachfolgenden Befehl kann man die Tabelle erstellen.

```
sql>copy from cv3dbck/<passwd>@cv3d to cv3dbck/<passwd>@cv3d create
<table_name> using select * from <table_name> where <feld_name> in(select
<feld:name> from <table_name>);
```

2.4 Views

2.4.1 Anzeigen

Erstelle Views kann man sich mit dem folgenden Befehl anzeigen lassen. Ausser die Abfrage auf die `user_views` gibt es auch noch `dba_views` und `all_views`.

```
sql>select view_name, text from user_views;

VIEW_NAME   TEXT
----------- --------------------------------------------------
V$SHOW_ADDR select strName, strStreet, strCity from t_adressen;
```

Materialized Views werden in den Tabellen `all_snapshot`, `all_mviews` und `dba_mviews` verwaltet.

```
sql>set long 2000
sql>select owner, mview_name, query from all_mviews;

OWNER       MVIEW_NAME  QUERY
----------- ----------------------------------------------------------
UWS         MV$ADRESS   select “T_ADRESSE”.”STRNAME” “NAME”
                              ,“T_ADRESSE”.”STRSTREET” “STRASSE”
                              ,“T_ADRESSE”.”STRCITY” “STADT”
                        from “T_ADRESSEN”;
```

2.4.2 Erstellen

Ein View kann mit folgendem SQL Befehl erstellt werden.

```
sql>create view <view_name> as select <feld_name>, <feld_name1> from
<table_name> where <feld_name> = <wert>;
```

Alle Spalten brauchen einen Namen. Ohne in dem nachfolgenden Beispiel die Angabe “JobTime“, würde das erstellen des Views mit einem ORA-00998 Fehler abgebrochen.

```
sql>create view v$batch_job_list as
 2>select j.job_id, j.job_name, to_char(j-job_time , ‘HH24:MI’) “JobTime”,
 3>q.queue_name from batch_job j, queue_name q
 4>where j.job_name = q.queue_id;
```

2.4.3 Löschen

Das löschen eines Views geschieht mit dem `drop` Befehl.

```
sql>drop view <view_name>;

sql>drop materialized view <mview_name>;
```

2.5 Spool Datei erstellen

Eine Spool Datei kann man mit `spool <file_name>` erstellen. Eine vorhandene Datei wird damit überschrieben. Mit `spool off` wird das schreiben in der Datei beendet. Überzälige Leerzeichen in der Ausgabe kann man mit `set trimspool on` abschneiden.

```
Set newpage 0
Set space 0
Set pagesize 0
/* Set linesize 150 */
Set feedback off
Set heading off
Set echo off
Set show off
Set termout off
Set verify off
Set trimspool on
Spool c:\temp\uschi.sql
Select ‘delete from <table_name> where <feld_name> = ‘ || <feld_name> ||
‘;’ from <table_name> where (sysdate – createdate)>5;
Select ‘commit;’ com from dual;
Spool off
Set feedback on
Set heading on
Set echo on
Set termout on
```

In dem nachfolgenden Beispiel wird in der Spool Datei die Anweisung `delete from hr.mitarbeiter where plz = 4711` stehen.

```
Select 'delete from hr.mitarbeiter where plz = ' || plz || ';' from
hr.mitarbeiter where plz = 4711;
```

Möchte man an einer vorhandenen Datei die Sql Ausgaben anhängen, so gibt man nach der Angabe der `Spool Datei` die Option `append` an.

```
sql>set newpage 0 -- no page return
sql>set space 0 -- no space
sql>set pagesize 0
sql>set feedback off
sql>set echo off
sql>set show off -- no display on screen
sql>set termout off
sql>set verify off
sql>set trimspool on
sql>spool <file_name> append
select status from v$instance;
sql>spool off
```

Die Leerzeichen am Ende eines Feldes kann folgendermaßen abgeschnitten werden.

```
Select path_name || ‘‘ || file_name from files;
```

Möchte man ein Hochkommata in einem Feld eingeben, so gibt man zwei Hochkommatas ein.

```
sql>select ‘commit force ‘‘‘ || local_tran_id || ‘‘‘;‘ from
 2>dba_2pc_pending where state = ‘prepared’;
sql>commit force ‘2016.1315.01’;
```

Bei der Eingabe von 4 Hochkommatas, warden 2 ausgegeben.

```
sql>select ‘Paul ‘‘‘‘the‘‘‘‘ goal keeper‘ as text from dual;
TEXT
------------------------
Paul ‘‘the‘‘ goal keeper
```

Eine neuer Form ist.

```
sql>select q’$name’s with$’ name from dual;
NAME
-----------
Name’s with
```

2.6 Index

2.6.1 Index erstellen

```
sql>create unique index <index_name> on <table_name> (feld_name);
```

```
sql>create index <index_name> on <table_name> (feld_name)
  2>tablespace <name> storage (initial 20k next 20k pctincrease 75);
```

2.6.2 Index Abfragen

```
sql>select * from all_ind_columns;
sql>select * from dba_ind_columns;
sql>select * from user_ind_columns;

sql>select * from all_indexes;
sql>select * from dba_indexes;
sql>select * from user_indexes;

sql>select * from index_stats;
sql>select * from index_histogram;
sql>select * from v$object_usage;
```

2.6.3 Index rebuild

```
sql>alter index <index_name> rebuild;
```

2.6.4 Index löschen

```
sql>drop index <index_name>;
```

Ist der Index ein Primary Key auf einer Tabelle, so kann der Index nicht mit dem `drop index` Befehl gelöscht werden, sondern nun mit dem nachfolgenden Befehl.

```
sql>alter table <table_name> drop constraint <index_name> cascade;

sql>alter table mitarbeiter drop constraint i_mitarbeiter_id cascade;
```

2.6.5 Index umbenennen

```
sql>alter index <index_name> rename to <index_name_new>;
```

2.6.6 Index verschieben

```
sql>alter index <index_name> rebuild tablespace <new_tablespace>;
```

2.7 Constraint

Mit Constraints wird verhindert, das es zu ungültigen Einträgen in einer Tabelle kommt. Folgende Constraints sind Oracle erlaubt.

Constraint	Typ	Beschreibung
Not null	C	Keine Null Werte sind erlaubt.
Unique	U	Eindeutigen Wert
Primary key	P	Ist eine Kombination von Not null und Unique
Foreign key	R	Legt eine Beziehung zu einer anderen Tabelle fest.
Check	C	Gibt Bedingungen an, die erfüllt werden müssen.

2.7.1 Constraint erstellen

Constraint kann man bei dem erstellen einer Tabelle gleich mit definieren.

```
sql>create table <name> (nr number not null, name varchar2(10));
```

Das nachträgliche anlagen eines Constraint erfolgt mit `alter table`.

```
sql>alter table <table_name>
  2>add (constraint 'sys_c003398' unique ('value_id'));
```

Anstelle der Option `unique` gibt es noch die Option `primary key, foreign key` und `check`.

```
sql>alter table <table_name>
  2>add (constraint 'sys_c003398' foreign key (<feld_name>)
  3>reference ‘cv3d.ddd_mod_attr’ (<feld_name>));
```

2.7.2 Constraint anzeigen

Für die Anzeige der Constraints gibt es die Tabellen `user_constraints`, `user_cons_columns` und `user_sequences`.

```
sql>select * from user_constraints where table_name = ‘<table_name>’;
```

2.7.3 Constraint löschen

```
sql>alter table <table_name> drop constraint <constraint_name> cascade;
```

2.7.4 Constraint deaktivieren / aktivieren

```
sql>alter table <table_name> disable constraint <constraint_name>;

sql>alter table <table_name> enable constraint <constraint_name>;
```

2.8 Sequence

2.8.1 Sequence erstellen

Sequenzen sind Generatoren für numerische Werte, die automatisch hoch gezählt werden und üblicherweise für Primärschlüssel verwendet werden.

```
sql>create sequence hr.mitarbeiter_s cycle noorder cache 20
2>maxvalue 2000000000 minvalue 1 increment by 1 start with 1;
```

increment by 1	Schrittgröße beim Hochzählen
start with 1	Startwert
minvalue 1	Kleinster Wert
maxvalue 99999	Größter Wert
cycle /nocycle	wieder bei minvalue starten, wenn maxvalue überschritten wird

2.8.2 Sequence löschen

```
sql>drop sequence hr.mitarbeiter_s;
```

2.8.3 Sequence anzeigen

```
sql>select * from user_sequences;
```

```
sql>select <sequence_name>.nextval from dual;
```

```
sql>select <sequence_name>.currval from dual;
```

Der letzte Befehl gibt nur dann einen Wert zurück, wenn vorher ein `nextval` gemacht worden ist.

2.8.4 Sequence verwenden

```
sql>insert into <table_name>
2>(num, name) values (<sequence_name>.nextval, 'Test');
```

2.9 Ausgaben formatieren

2.9.1 Spaltenformatieren

Als Ausgabeformate gibt es Alphanumerische (a20) und Integer (9999).

```
sql>col <field_name> format a20

sql>col <field_name> format 99,999.99

sql>col ”File Size” format a12

sql>select tablespace_name, bytes ”File Size” from dba_data_files;

sql>col session_recid for 999999 heading “SESSION|ID” -- Überschrift 2
zeilig

sql>col <field_name> for a30 heading “Feld1“ justify right -- Ausrichtung
rechts
```

Die Ausrichtung des Feld Namens wird mit `justify left`, `justify right` und `justify center` definiert.

2.9.2 Zeilen / Seite

Ausgaben Breite.

```
sql>set linesize 150
```

Ausgaben Zeilen.

```
sql>set pagesize 50
```

2.9.3 Datum lesbar ausgeben

```
sql>select to_date(<field_name>,’DD.MM.YYYY HH24:MI:SS’) from dual;

TO_CHAR(SYSDATE)
--------------------
08.11.2011 18:12:34
```

```
sql>select to_char(<field_name>,'DD.MM.YYYY') from dual;

TO_CHAR
----------
08.11.2011
```

Das Datum in Oracle wird als Numbers abgespeichert. Mit `trunc` werden die Precision abgeschnitten.

```
sql>select <field_name> from v$database where trunc(date) = trunc(sysdate);

sql>select model_id, name, createdate from ddd_model_catalog
  2>where to_char (createdate) in
  3>(select to_char (sysdate-1) from dual);
```

Eine Auswahl der Datumsformaten stehen in der nachflogenden Liste.

Format	**Beschreibung**
DD	Numerischer Tag (10).
DY	Gekürzter Tag (Mon).
MM	Numerischer Monat (08).
MON	Gekürzter Monat (Jan).
MONTH	Monat (Januar).
YYYY	Jahr mit 4 Stellen (2011).
YY	Jahr mit 2 Stellen (11).
AM oder PM	Vormittags oder Nachmittags.
HH	Stunden (1-12)
HH24	Stunden (18)
MI	Minuten.
SS	Sekunden

2.9.4 Bytes in MB

```
sql>select tablespace_name, file_name, round(bytes / 1024 / 1024) || ' MB'
 2 "Size in MB" from dba_data_files order by tablespace_name;

Tablespace_Name   File_name                    Size in MB
---------------   ----------------------       ----------
SYS               /u02/oradata/sys01.dbf       500 MB
TOOLS             /u02/oradata/tools01.dbf     100 MB
```

2.10 Tools

Den letzten Befehl im Editor bearbeiten. Welcher Editor aufgerufen wird, hängt davon ab, was in der Variable _EDITOR steht. Mit define kann man einen anderen Editor definieren.

```
sql>define _EDITOR = vi

sql>ed
```

Wechseln auf Betriebssystem

```
sql>host
```

Bash Befehle in einer Sql-Session ausführen.

```
sql>! whoami
oracle

sql>define cmd='whoami'
sql>host &&cmd
oracle
```

2.11 Variablen

Die Ausgabe `alt` und `neu` kann man mit `Verify off` auschalten.

2.11.1 Unterschied zwischen &-Variable und &&-Variable

Definiert man eine Variable mit `zwei &,` so wird man bei einer erneuten Ausführung nicht mehr nach einer Eingabe gefragt.

```
sql> select username from dba_users where username =
'&&1';
Geben Sie einen Wert für 1 ein: SYS
alt 1: select username from dba_users where username = '&&1'
neu 1: select username from dba_users where username = 'SYS'

username
--------
SYS

sql>/
alt 1: select username from dba_users where username = '&&1'
neu 1: select username from dba_users where username = 'SYS'

username
--------
SYS
```

2.11.2 Eingabeaufforderung mit Accept und Prompt

Nach der Eingabe des Befehls `accept` (`acc`), kann man den Wert einer Variablen eingeben, ohne dass ein Sql-Befehl ausgeführt wird.

```
sql>acc 2
SYS
sql>select username from dba_users where username = '&2';
alt 1: select username from dba_users where username = '&2'
neu 1: select username from dba_users where username = 'SYS'

username
--------
SYS
```

Mit dem Befehl `prompt` kann man zusätzlich noch einen Kommentar eingeben.

```
sql>acc 2 prompt 'Bitte den Usernamen eingeben: '
Bitte den Usernamen eingeben: SYS
sql>select username from dba_users where username = '&2';
alt 1: select username from dba_users where username = '&2'
neu 1: select username from dba_users where username = 'SYS'

username
--------
SYS
```

Hier sind noch ein paar Beispiele für `accept` und `prompt`. Im ersten Beispiel wird das Password versteckt eingegeben.

```
sql>set verify off
sql>accept pwd char prompt ‘Please type password: ‘ hide
```

Die Einagbe kann auch vorher formartiert werden.

```
sql>accept sal number format ‘999.99‘ default ‘000.00‘ prompt ‘Salary: ‘

sql>accept name char format ‘A20‘ prompt ‘Enter last name: ‘

sql>accept hdate date format ‘dd.mm.yyyy‘ default ’01.01.2016‘
 2>prompt ‘Enter hired date: ‘
```

Diese beiden Befehle eignen sich hervoragend für das Ausführen eines Sql-Scripts.

```
oracle@woby1002>type queryDbaUsers.sql
prompt Wir Starten nun.
acc 2 prompt ’Bitte den Usernamen eingeben: ’
prompt Vielen Dank!
Select username from dba_users where username = ’&2’

sql>@queryDbaUsers.sql
Wir Starten nun.
Bitte den Usernamen eingeben: HR
Vielen Dank!
alt 1: select username from dba_users where username = ’&2’
neu 1: select username from dba_users where username = ’HR’

username
--------
HR
```

2.11.3 Define Deklaration

Mit dem Befehl `define` werden Variablen Werte zugewiesen, die dann bei einem Gebrauch nicht auf eine Eingabe warten.

```
sql>define 3=HR
sql>select username from dba_users where username = ’&3’;
alt 1: select username from dba_users where username = ’&3’
neu 1: select username from dba_users where username = ’HR’

username
--------
HR
```

2.11.4 Trennzeichen

Möchte man an einer Eingabe noch einen Wert anhängen, so wird dieses mit einem Punkt als Trennzeichen gemacht.

```
sql>select &a.000 from dual;
Geben Sie einer Wert für a ein: 33
alt 1: select &a.000 from dual
neu 1: select 33000 from dual

33000
-----
33000
sql>undefined a
sql>select ’%a.laub’ from dual;
Geben Sie einen Wert für a ein: Ur
Alt 1: select ’%a.laub’ from dual
Neu 1: select ’Urlaub’ from dual

Urlaub
------
Urlaub
```

2.11.5 Variablen weiterverarbeiten

Soll das Ergebnis einer Abfrage in einer Variablen abgelegt werden, so wird hierzu der Befehl `into` benutzt.

```
oracle@woby1002>cat FreeSpace.sql
prompt
prompt Loading FreeSpace.sql
prompt
set verify off
variable intFree number;
variable intSize number;
variable intDiff number;
variable intUsed number;
col tablespace_name for a10 heading ’Tablespace’
select tablespace_name from dba_tablespaces order by tablespace_name;
prompt
acc strTS prompt ‘Bitte den Tablespace Namen eingeben: ‘
begin
 select sum(bytes/1024/1024) into :intFree from dba_free_space where
tablespace_name = '&strTS';
 select sum(bytes/1024/1024) into :intSize from dba_data_files where
tablespace_name = '&strTS';
 select sum(:intSize-:intFree) into :intDiff from dual;
 select sum(:intDiff*100/:intSize) into :intUsed from dual;
end;
/
select :intSize "DataFile Size", to_char(:intFree, ’999999.9’) "Free Space
", to_char:(intDiff, ’999999.9’) "Used", to_char(:intUsed, ’99.9’) ” %”
from dual;
```

2.11.6 Variablen umwandeln

Möchte man eine Eingabe in Großbuchstaben umwandeln, so geschieht das mit dem Befehl `upper`. Mit `lower` wird die Eingabe in Kleinbuchstaben umgewandelt.

```
sql>acc 2 prompt 'Bitte den Usernamen eingeben: '
Bitte den Usernamen eingeben: sys
sql>select user_name from dba_users where user_name = upper('&2');

sql>select user_name from dba_users where user_name = lower('&2');
```

2.12 Befehle ausführen

Mit einem Semikolon oder einer Leerzeile wird das SQL-Statement abgeschlossen. Wird das SQL-Statement mit einem Semikolon abgeschlossen, so wird dieses Statement sofort ausgeführt.

```
sql>select username, account_status from dba_users
 2
sql>

sql>select username,account_status from dba_users where username = 'SYS';

username    account_status
--------    --------------
SYS         open

1 Zeile ausgewählt
```

Eine Anweisung kann sich auch über mehrere Zeilen erstrecken. Abgeschlossen wird dieser Block entweder mit einem `Punkt`, der in einer Zeile eingetragen wird, oder mit einem `Slash`. Wird der Block mit einem `Slash` abgeschlossen, so wird die Anweisung sofort ausgeführt. Anstelle des `Slash` kann man auch das `Semikolon` genommen werden.

```
sql>select
 2 username, account_status
 3 from dba_users
 4 .
sql>

sql>select
 2 username, account_status
 3 from dba_users where username = 'SYS'
 4 /

username    account_status
--------    --------------
SYS         open

1 Zeile ausgewählt
sql>
```

Soll eine Anweisung nochmal ausgeführt werden, so kann man den Befehl `Run (r)` oder einen `Slash` eingeben. Bei `Run` wird der Befehl noch mal ausgegeben, bei einem `Slash` nicht.

```
sql>r
 1 select
 2 username, account_status
 3 from dba_users where username = 'SYS'
 4 *

username    account_status
--------    --------------
SYS         open

1 Zeile ausgewählt
sql>/
username    account_status
--------    --------------
SYS         open

1 Zeile ausgewählt
```

```
sql>
```

Den letzten SQL Befehl kann man sich mit `List (l)` anzeigen lassen und mit `Save (sav)` wird dieser Befehl in einer Datei geschrieben. Existiert die Datei bereits, so muss im Anschluss an den Dateiname die Option `replace` eingegeben werden. Zum laden einer Datei in den Speicher kann man den Befehl `Get` nehmen. Nach dem laden der Datei kann man den Befehl mit `Run` ausführen.

In der zu ladenden Datei darf nur eine einzelne Anweisung stehen. Stehen mehrere Anweisungen in der SQL-Datei, so wird diese Datei mit dem Befehl `Start` oder `@` geladen und auch gleich ausgeführt.

```
sql>l
 1 select
 2 username, account_status
 3 from dba_users where username = 'SYS'
 4 *
sql>save /home/oracle/sql/sys [replace]

sql>get /home/oracle/sql/sys.sql
 1 select
 2 username, account_status
 3 from dba_users where username = 'SYS'
 4 *
sql>run
 1 select
 2 username, account_status
 3 from dba_users where username = 'SYS'
 4 *
username    account_status
--------    --------------
SYS         open

1 Zeile ausgewählt
sql>start /home/oracle/sql/sys.sql

username    account_status
--------    --------------
SYS         open

username    account_status
--------    --------------
HR          open

sql>@/home/oracle/sql/sys.sql

username    account_status
--------    --------------
SYS         open

username    account_status
--------    --------------
HR          open
```

2.13 Sql-Scripte

2.13.1 Kommentare

Um Kommentare in einem Sql-Script zuschreiben, gibt es insgesamt drei Möglichkeiten. Wie bei Batch Scripten üblich, wird der Befehl `Rem` benutzt. Möchte man einen Kommentar über mehrere Zeilen schreiben, so beginnt dieser Kommentar mit `/*` und endet mit `*/`. Nach dem /* ist unbedingt ein Leerzeichen zu setzten, sonst wird der Block nicht als Kommentarblock erkannt. Mit `--` kann man einen Kommentar hinter einem Befehl schreiben.

```
rem Diese Zeile ist eine Kommentarzeile
rem

/* Ab hier kann man seitenweise Kommentare schreiben,
die einfach nicht aufhören wollen */

select username from dba_users; -- Welche Oracle Users gibt es.
```

2.13.2 Ausgabe Text

Mit dem Befehl `prompt`, kann man einen Text in einem Sql Fenster ausgeben. Mehrere Zeilen kann man mit DOC ausgeben, wobei am Anfang dann DOC> steht.

```
oracle@woby1002>cat ausgabe.sql
prompt +++ Diese Zeile wird ausgegeben +++
prompt
DOC
 Und nun werden mehrere Zeilen ausgegeben,
 wobei am Anfang ein DOC> steht.
# -- Ende von DOC

oracle@woby1002>sqlplus uws@cad01

sql>@ausgabe
+++ Diese Zeile wird ausgegeben +++

DOC> Und nun werden mehrere Zeilen ausgegeben,
DOC> wobei am Anfang ein DOC> steht.
DOC>#
```

Eine Text Datei kann auch ausgegeben werden.

```
oracle@woby1002>cat ScriptHeader.txt
 Hier steht ein Beispielttext.

 ++++++++++++++++++++++++
 + Author … Uwe Schimanski
 ++++++++++++++++++++++++

oracle@woby1002>cat ShowText.sql
host cat ScriptHeader.txt
```

Anstelle einer Text Datei, kann man auch ein Shell Script ausführen. In diesem Shell Script kann die Ausgabe des Textes mittels `echo` oder `printf` erfolgen. So ist auch eine farbige Ausgabe möglich.

```
oracle@woby1002>cat ShowTextScript.sql
host ~/sql/ScriptHeader.sh
```

Mit LPAD kann man auch Text ausgeben. Der Text wird von einer Position nach Links geschrieben. Als erstes steht der Text der ausgegeben werden soll. Dann die Position des Textes und danach kann optional ein Zeichen stehen, der anstelle der Leerzeichen genommen werden soll.

```
sql>set serveroutput on
sql>set head off
sql>select LPAD('Start Program *****',25) from dual;

      Start Program *****

sql>select LPAD('Start Program *****',25,'*') from dual;

******Start Program *****

sql>select LPAD LPAD('Start Program *****',25) ||
 2>(to_char(sysdate, 'DD.MM.YYYY HH24:MI:SS'),26) from dual;

      Start Program *****       13.02.2015 18:04:34
```

2.13.3 Parameter übergeben

Parameter können in einem Sql-Script oder auch in einem Sql-Fenster mit dem Befehl into übergeben werden.

```
oracle@woby1002>cat userid.sql
/* Auslesen der max. User_id */
variable intId number
begin
 select max(user_id) into :intId from dba_users;
end;
/
print intId
select :intId from dual;
select username from dba_users where user_id = :intid;
```

Möchte man bei dem Ausführen des Sql Scripts Parameter übergeben, so werden diese Parameter hinter dem Aufruf angehangen.

```
oracle@woby1002>cat parameter.sql
select '&1' from &2;

oracle@woby1002>sqlplus / @parameter.sql '"Parameter 1"' dual

oracle@woby1002>cat cmdspool.sql
spool &1/queryuser.sql
select * from dba_users;
spool off

oracle@woby1002>sqlplus / @cmdspool.sql $temp
```

2.13.4 Fehler Abbruch

Tritt in einem Sql_Script ein auf, so wird der Fehler übersprungen und der nächste Befehl wird ausgeführt. Soll die Vearbeitung des Scriptes bei einem Fehler abbrechen, so gibt man am Anfang des Scriptes die `whenever` Anweisung an.

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

Möchte man die Verarbeitung trotz eines Fehlers zulassen, so gibt man die Option `continue` mit an. Diese Anweisung bezieht aich auf alle nachfolgenden Sql-Befehle.

```
whenever sqlerror continue;
```

2.13.5 Return Code auswerten

Wird eine Sql-Session mit `exit` verlassen, so kann dieser Wert ausgewertet werden. Für `Success` wird eine `0`, für `Failure` eine `1` und für `Warning` eine `2` ausgegeben. Mann kann auch Numerische Werte bis 255 übergeben. In einer Sql-Session kann man die Syntax mit dem Befehl `help exit` abfragen.

```
oracle@woby1002>sqlplus /
sql>exit
oracle@woby1002>echo $?
0

oracle@woby1002>sqlplus /
sql>exit 48
oracle@woby1002>echo $?
48

oracle@woby1002>sqlplus /
sql>variable rtc number;
sql>begin
 2 select 66 into :rtc from dual;
 3 end;
 4 /

PL/SQL-Prozedur erfolgreich abgeschlossen.
sql>exit :rtc
oracle@woby1002>echo $?
66

oracle@woby1002>sqlplus /
sql>help exit

{exit|Quit} [SUCCESS|FAILURE|WARNING|n|variable|:BindVariable]
  [COMMIT|ROLLBACK]
sql>
```

2.13.5 Cursor definieren

Es können `Cursor-Variablen` in Sqlplus definiert werden, die dann in einem PL/SQL-Block verwendet werden. Damit die Ausgabe auf dem Bildschirm erfolgt, wird die `autoprint` Option eingeschaltet.

```
sql>variable a refcursor
sql>set autoprint on
sql>begin
 2 open :a for
 3 select username
 4 from dba_users
 5 where username like '%SYS%';
 6 end;
 7 /

PL/SQL Procedure erfolgreich abgeschlossen

Username
--------
SYS
WKSYS
```

```
oracle@woby1002>cat example1.sql
set serveroutput on size 1000000
define logfile= '/temp/example1.log'
declare
 cursor a1 is
  select distinct path_name from table_102 where path_name like 'd:%';
begin
 for a in a1
  loop
   dbms_output.put_line('mkdir ' || a.path_name || '>> &logfile';
  end loop;
end;
/
set serveroutput off
```

2.13.6 Case Anweisung

Mit einer Case Anweisung kann man eine einfache Bedingung erstellen.

```
sql>select case count(object_name)
 2>when 0 then 'There are no Invalid Objects'
 3>else 'There are ' ||count(object_name)|| ' Invalid Objects'
 4>end "Number of Invalid Objects"
 5>from dba_objects
 6>where status='INVALID'
 7>and owner in ('SYS','SYSTEM');

Number of Invalid Objects
-------------------------------------------------
There are 4 Invalid Objects
```

```
sql>select table_name,
 2>case owner
 3> when ‘SYS’ then ‘The owner is SYS’
 4> when ‘SYSTEM’ then ‘The owner is SYSTEM’
 5> else ‘Other owner’
 6>end “Owner”
 7>from all_tables;

TABLE_NAME  Owner
----------- -------------------
DUAL        The owner is SYS
OL$HINTS    The owner is SYSTEM
CITY        Other owner
```

Die obrige Case Anweisung kann man auch so schreiben.

```
sql>select table_name,
 2>case
 3> when owner=‘SYS‘ then ‘The owner is SYS‘
 4> when owner=‘SYSTEM‘ then ‘The owner is SYSTEM‘
 5> else ‘Other owner’
 6>end “Owner”
 7>from all_tables;
```

Vergleichen von zwei Feldern.

```
sql>select street,
 2>case
 3> when company = ’Siemens’ and city = ’Muenchen’ then ’Sued Deutschland’
 4> when company = ’INFO AG’ and city = ’Hamburg’ then ’Nord Deutschland’
 5> end
 6>from suppliers;
```

```
sql>select
 2>case
 3> when field1 < field2 then ‘Found’
 4> when field3 < field4 then ‘Not exist’
 5> end
 6>from suppliers;
```

Noch testen, was das mit %% added.. auf sich hat.

```
sql>select last_name, job_id, salary,
 2>(case
 3> when job_id like ‘SA_MAN’ and salary < 12000 then ‘10%’ or %% added in
the end
 4> when job_id like ‘SA_MAN’ and salary >= 12000 then ‘15%’
 5> when job_id like ‘IT_PROG’ and salary < 9000 then ‘8%’
 6> when job_id like ‘IT_PROG’ and salary >= 9000 then ‘12%’
 7> else ‘Not applicable’
 8> end ) Raise
 9>from employees;
```

```
sql>select model_no, partition_no,
 2>case
 3> when partition_no <50 then 'Low No.'
 4> when partition_no >=50 and partition_no <300 then 'Middle No.'
 5> when partition_no >300 then 'Large No.'
 6>end "Values"
 7>from pdtable_113;
```

2.14 SqlPlus Konfiguration

Wenn eine Sqlplus Session gestartet wird, so wird die Datei `glogin.sql` ausgeführt, die sich im Verzeichnis `$ORACLE_HOME/sqlplus/admin` befindet. Gibt es in diesem Verzeichnis auch noch eine `login.sql`, so wird diese auch geladen.

```
oracle@woby1002>cat glogin.sql
prompt
prompt Loading glogin.sql …
prompt
--
-- Laden von SQL Scripten
--
@c:\Oracle\Admin\Scripts\MeinScript.sql
-- User settings

alter session set nls_date_format='DD.MM.YYYY hh24:mi:ss';

set serveroutput on
set trimspool on
set linesize 200
set pagesize 50
set SQLPROMPT "_USER'@'_CONNECT_IDENTIFIER>"

define _EDITOR=vi

col osuser for a15 heading 'Username'
col username for a15 heading 'Oracle Name'
col client_info for a20
col machine for a25 heading 'Computer Name'
col member for a35
col program for a20 heading 'Programm'
--
-- for Show Parameter
--
col name_col_plus_show_param for a36 heading 'Name'
col value_col_plus_show_param for a30 heading 'Value'
--
-- for Show SGA command
--
col name_col_plus_show_sga for a24
col units_col_plus_show_sga dor a15
```

Folgende Variablen gibt es für den Sqlprompt.

Variable	**Beschreibung**
_CONNECT_IDENTIFIER	Oracle SID
_DATE	Datum
_O_RELEASE	Release Nummer der Datenbank
_PRIVILEGE	Privilegien
_SQLPLUS_RELEASE	Release Nummer von Sqlplus
_USER	Oracle User

Beispiel für die Anzeige der Versionsnummer:

```
define _CURRVER=NN
col ver noprint new_value _CURRVER
set termout off
select
 substr(version, 1, instr(version, '.', 1, 2) -1) as ver
from
 product_component_version
where
 product like 'Oracle Database%';
set termout on
set SQLPROMPT "'<'_CURRVER>':'_USER'@'_CONNECT_IDENTIFIER>"
```

Anstelle der `substr` Anweisung kann auch nur die Abfrage '`version as ver`' stehen.

Alle `set` Optionen mit ihren Einstellungen kann man sich mit `show all` anzeigen lassen.

```
sql>show all

appinfo is OFF and set to „SQL*Plus“
arraysize 15
autocommit OFF
.
.
```

Den Hostnamen in dem Sql Prompt kann man folgendermaßen realisieren.

```
column host new_value host
set termout off
select host_name host from v$instance;
set termout on
set SQLPROMPT "&host':'_USER'@'_CONNECT_IDENTIFIER>"
```

2.15 SqlPlus Einstellungen sichern

Hat man in einer Sqlplus Session die Konfiguration abgeändert und möchte diese Einstellungen in einer Datei abspeichern, so geschieht das mit dem Befehl `store`. Diese Datei kann dann jederzeit geladen werden und die SqlPlus Einstellungen werden wieder auf dem stand zurückgesetzt.

```
sql>store set /home/oracle/sql/sqlsession.sql replace

sql>@/home/oracle/sql/sqlsession.sql
```

2.16 HTML Seite generieren

Möchte man das Ergebnis einer Abfrage als HTML Seite speichern, so nimmt man hierzu den `Markup` Befehl. Die Syntax des Befehls ist `set markup html [on|off] head [text] body [text] table [text] entmap [on|off] spool [on|off] pre[format] [on|off]`.

```
sql>set markup html on
sql>set echo off
sql>spool index.html
sql>select * from dba_users;
sql>spool off
sql>set echo on
sql>set markup html off
```

2.17 Datatype Clob/Nclob/Long

Sind in einer Tabelle Felder erstellt worden, die als Datatype `Clob/Nclob/Long` enthalten und man macht eine Abfrage auf diese Tabelle, so werden standardmäßig nur die ersten 80 Zeichen des Feldes ausgegeben. Mit dem Befehl `set long` kann man die Ausgabe vergrößern.

```
sql>set long 32000
sql>select text from user_tables where rownum = 1;
```

2.18 Proceduren

2.18.1 Erstellen

Eine Procedure wird mit dem Befehl `create or replace procedure` erstellt. Die Anweisung stehen dann zwischen den `begin` und `end` Blöcken. In einer Procedure kann man mehrere `begin` und `end` Blöcke haben.

```
oracle@woby1002>cat procedure1.sql

create or replace procedure <name> is
begin

-- Hier stehen dann die Anweisungen
end;
/
```

2.18.2 Anzeigen

Anzeige der erstellten `Proceduren, Funktionen` und Packages. Anstelle von `all_source` kann der normale Oracle User auch die Abfrage auf die Tabelle `user_source` machen.

```
sql>select * from dba_objects
 2>where owner = 'SCOTT'
 3>and object_type in ('PROCEDURE','PACKAGE','FUNCTION','PACKAGE_BODY');

sql>select text from all_source [user_source]
 2>where name = '<procdure_name>'
 3>order by line;
```

2.18.3 Verschlüsseln

Mit dem Programm `wrap.exe` kann man `Funktionen`, `Proceduren`, `Packages` und `Types` verschlüsseln.

```
oracle@woby1002>wrap iname=myFunction.sql oname=myFunction.plb
```

Die neu erstellte Datei kann dann in die DB geladen werden.

```
sql>@myFunction.plb
```

2.19 Trigger

2.19.1 Erstellen

Trigger werden mit dem Befehl `create or replace trigger` erstellt.

```
oracle@woby1002>cat trigger_reboot.sql
create table start_down_log (
id number,
sid number,
event_time date,
osuser char(20),
machine char(30),
username char(20),
action char(20));

create index start_down_log_n on start_down_log (osuser);
create sequence start_down_log_s cycle noorder cache 1 maxvalue 5000
minvalue 1 increment by 1 start with 1;
create view v$start_down_log as
select id, event_time “Date”, to_char(event_time, ‘HH24:MI:SS’) “Time”,
osuser, machine, username, action from start_down_log;

create or replace trigger shutdown_log before shutdown database
begin
 insert into start_down_log (id, sid, event_time, osuser, machine,
username) select start_down_log_s.nextval, sid, sysdate, osuser, machine,
username from v$session where sid in (select userenv('sid') from dual);
 update start_down_log set action = 'Shutdown' where sid in (select
userenv('sid') from dual);
 commit;
end;
/

create or replace trigger startup_log after startup on database
begin
 insert into start_down_log (id, sid, event_time, osuser, machine,
username) select start_down_log_s.nextval, sid, sysdate, osuser, machine,
username from v$session where sid in (select userenv('sid') from dual);
 update start_down_log set action = 'Startup' where sid in (select
userenv('sid') from dual);
 commit;
end;
/
```

2.19.2 Anzeigen

```
sql>select * from dba_triggers;

sql>select * from user_triggers;
```

2.19.3 Löschen

Mit `drop trigger` werden erstellt trigger gelöscht.

```
sql>drop trigger <name>;
```

2.19.4 Aktivieren / Deaktivieren

Trigger kann man mit `enable` oder `disable` Aktivieren oder Deaktivieren.

```
sql>alter trigger <name> [enable | disable];
```

2.20 Package, Function, Procedure

2.20.1 Invalid

Invalid Packages, Functions und Proceduren kann man sich mit dem nachfolgenden `select` statement alle User Objecte anzeigen lassen.

```
sql>select object_type, object_name, status from user_objects
 2>where status = 'INVALID' and
 3>object_type in ('PACKAGE','FUNCTION','PROCEDURE');
```

Bei einer Abfrage auf die DBA Object Tabelle kann zusätzlich noch der Owner ausgegeben werden.

```
sql>select object_type, object_name, status, owner from dba_objects
 2>where status = 'INVALID' and
 3>object_type in ('PACKAGE','FUNCTION','PROCEDURE');
```

2.20.2 Compile

Werden bei der Abfrage `Invalid Packages`, `Functions` oder `Proceduren` angezeigt, so kann man sie neu `kompilieren`.

```
sql>alter package <schema>.<package_name> compile;
```

2.21 Database Link

Mit Datenbank Liks kann man von einer Lokalen Datenbank auf eine Remote DB zugreifen, ohne sich mittels connect mit der Remote DB zu verbinden. Für die Verwaltung gibt es die Parameter `open_links` und `open_links_per_instance`.

2.21.1 Erstellen

Mit `create database link` wird ein neuer Links angelegt. Die ganze Syntax hierzu lautet:

`Create [shared] [public] database link <link_name> connect to <user> identified by <passwd> using <connect_string>`

```
sql>create database link woby1003
 2>connect to uws identified by password
 3>using 'cad10';
```

2.21.2 Anzeigen

Angelegte Links lassen sich mit den folgenden Views anzeigen.

dba_db_links	Liste aller dba Links
all_db_links	Liste aller Links
user_db_links	User erstellte Links
v$dblink	Liste aller offenen DB Links in der Sesion, wenn IN_TRANSACTION = yes ist.
gv$dblink	

```
sql>col db_link for a30
sql>col username for a15
sql>col host for a20

sql>select * from dba_db_links;

OWNER DB_LINK     USERNAME    HOST        CREATED
----- ----------- ----------- ---------- ----------
SYS   cad10       uws         cad10       12.03.2012
UWS   cad09       hr          cad09       24.04.2012
```

2.21.3 Löschen

```
sql>drop database link <link_name>
```

3. Copyright