Oracle PL/SQL

Book

Uwe Schimanski

Seab@er Software

Goch, den 17. Juli 2019

```
/*
 *  Shutdown Protokollierung einer Instanz.
 *
 *  Author ....... Uwe Schimanski
 *  Company ...... Seab@er Software AG
 *  Date ......... 02-Nov-2011
 *  Version ...... 1.0.2
 *
 * Create Logging Table
 *
 */
prompt
prompt +++++++++++++++++++++++++++++++++++++++++++++++
prompt Create Logging Table, Index, Sequence and View
prompt +++++++++++++++++++++++++++++++++++++++++++++++
prompt
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 2 maxvalue 5000 minvalue 1 increment by 1 start with 1;
create view v$start_down_log as select id, event_time "Date", to_char(event_time, 'HH24:MI:SS') "Time", osuser, machine, username, action from start_down_log;
--
-- Create Trigger Shutdown_Log
--
prompt
prompt ++++++++++++++++++++++++++++
prompt Create Trigger shutdown_log.
prompt ++++++++++++++++++++++++++++
prompt
create or replace trigger shutdown_log before shutdown on database
begin
 insert into start_down_log (id, sid, event_time, osuser, machine, username) select start_down_log_s.nextval, sid, sysdate, osuser, machine, username from v$session where sid
 in (select userenv('sid') from dual);
 update start_down_log set action = 'Shutdown' where sid in (select userenv('sid') from dual);
 commit;
end;
/

--
-- Create Trigger Startup_Log
--
prompt
prompt +++++++++++++++++++++++++
prompt Create Trigger startup_log.
prompt +++++++++++++++++++++++++
prompt
create or replace trigger startup_log after startup on database
begin
```

Published by Publisher

seabaer-ag.de

First printing, May 2018

Inhaltsverzeichnis

Vorwort

Beruflich beschäftige ich mich mit Oracle seit 1999. In dieser Zeit habe ich die Oracle Kurse `Oracle Admin I und II`, den `Oracle Dataguard` und `Oracle Rman` belegt.

Alle Informationen, die ich zusammentragen konnte in der Zeit, sind in diesem Buch eingeflossen. Ebenso sind meine Erfahrungen mit Oracle auch in diesem Buch enthalten.

Hauptsächlich habe ich Oracle auf Linux Servern für unseren Kunden betreut. Die Informationen können auch für Oracle auf Windows übernommen werden.

Diese Dokumentation wurde für die Oracle Datenbanken 10G R2 und 11G R1 geschrieben und auch getestet. Alles sollte auch unter Oracle 12c und auch 18c funktionieren.

Meinen Dank gilt vor allen meiner Familie, die mich bei dem erstellen des Buches immer unterstützt haben.

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

Kapitel 1

PL/SQL

1.1 Tabellen

1.1.1 Leere Tabelle erstellen

```
sql>CREATE TABLE hr.mitarbeiter (
 2>vname char(25),
 3>nname char(40),
 4>alter number(4));
```

Listing 1.1: Tabelle erstellen

Folgende Feldtypen gibt es in Oracle.

Feldtype	*Wert*	*Beschreibung*
char(xx)	Maximum 2000 bytes	Die Angabe xx gibt an, wie viele Zeichen gespeichert werden können. Fixed-length strings. Space padded
nchar(xx)	Maximum 2000 bytes	Wie char. Fixed-length strings. Space padded
nvarchar2(xx)	Maximum 4000 bytes	Wie char. Variable-length NLS string
varchar2(xx)	Maximum 4000 bytes	Wie char. Variable-length 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

Tabelle 1.1: Feldtypen

1.1.2 Vorhandene Tabelle kopieren

In dem nächsten Beispiel wird die Struktur und deren Inhalt in eine neu zu erstellende Tabelle übernommen.

```
sql>CREATE TABLE <TableName> AS
 2>SELECT * FROM <TableName>
 3>WHERE <FeldName> in(SELECT <FeldName> FROM <TableName>);
```

Listing 1.2: Tabelle kopieren

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 estellen.

```
sql>COPY from cv3dbck/<passwd>@cv3d to cv3dbck/<passwd>@cv3d
 2>CREATE <TableName> using SELECT * from <TableName>
 3>WHERE <FeldName> in (SELECT <FeldName> from <TableName>);
```

Listing 1.3: Datatype Long

1.1.3 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 hr.mitarbeiter (
 2>vname char(25) not null,
 3>nname char(40) not null,
 4>bday date not null,
 5>email char(100))
 6>PARTITION BY RANGE (bday)
 7>(PARTITION p_before_1_jan_1970 values less than
 8>(to_date('01-01-1962','dd-mm-yyyy')),
 9>PARTITION p_before_1_jan_1980 values less than
10>(to_date('01-01-1965','dd-mm-yyyy')),
11>PARTITION p_before_1_jan_2007 values less than
12>(to_date('01-01-2008','dd-mm-yyyy')))
13>ENABLE ROW MOVEMENT;
```

Listing 1.4: Tabelle Partition

Nun können wir Werte eingeben.

```
sql>INSERT INTO adrbook VALUES (
 2>'Ralf','Schmidt',to_date('15-07-1963','dd-mm-yyyy'),'ralf.schmidt@mail.xs
    );

sql>INSERT INTO adrbook VALUES (
 2>'Harry','Hirsch',to_date('08-05-2007','dd-mm-yyyy'),'hhirsch@wald.com);

Fehler in Zeile 1:
ORA-14400: Eingefuegter Partitionsschluessel kann keiner Partition
   zugeordnet werden.
```

Listing 1.5: Tabelle Partition

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 INTERVALL (numtoyminterval(1,'year'));
```

Listing 1.6: Tabelle Partition Intervall

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:MI:SS')
P_BEFORE_1_JAN_1980 TO_DATE('1980-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS')
P_BEFORE_1_JAN_2007 TO_DATE('2007-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS')
SYS_P41             TO_DATE('2008-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS')
```

Listing 1.7: Partition Info

Die Systemgenerierten Partitionen kann man nachträglich umbenennen.

```
sql>ALTER TABLE adrbook RENAME PARTITION sys_p41 TO p_before_1_jan_2008;
```

Listing 1.8: Partition umbenennen

Man kann auch mehrere Partitions zu einer Partition mit dem Befehl merge 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;
```

Listing 1.9: Partition Merge

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 INTERVALL (1000);
```

Listing 1.10: Partition Intervall

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

```
sql>CREATE TABLE <TableName} (
 2>lfdnr number(6) not null,
 3>text varchar(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;
```

Listing 1.11: Partition Range

1.1.4 Tabellenstruktur anzeigen

```
sql>DESCRIBE hr.mitarbeiter;
NAME                 NULL?     TYPE
-------------------- --------- ---------------
ID                   NOT NULL  NUMBER(6)
VNAME                          VARCHAR2(50)
NNAME                          VARCHAR2(50)
MAIL                           VARCHAR2(100)
```

Listing 1.12: Tabellenstruktur

1.1.5 Tabelle umbenennen

```
sql>ALTER TABLE <TableName> RENAME TO <NewTableName>
```

Listing 1.13: Tabelle rename

1.1.6 Tabelle löschen

Eine Tabelle kann man mit `drop table` löschen. Eine so gelöschte Tabelle kann man wieder mit dem `flashback` Befehl zurückholen. Gibt man noch `purge` mit an, so kann man die Tabelle nicht mehr wieder herstellen, auch nicht mit einem `flashback`.

```
sql>DROP TABLE <TableName>;

sql>DROP TABLE <TableName> PURGE;
```

Listing 1.14: Tabelle löschen

Mehrere Tabellen in einem Rutsch löschen kann man mit einem PL-Block machen. 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>/
```

Listing 1.15: Mehrere Tabellen löschen

1.1.7 Tabelle komprimieren

```
sql>CREATE TABLE <TableName> COMPRESS for all operations
 2>AS SELECT * from emp;

sql>ALTER TABLE <TableName> COMPRESS for all operations;
```

Listing 1.16: Tabelle komprimieren

1.1.8 Tabellenfeld hinzufügen

```
sql>ALTER TABLE <TableName> add column (<FeldName> <FeldType> <constrain>);
```

Listing 1.17: Tabellenfeld hinzufügen

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 (
 2>plz char(5) NOT NULL,
 3>nickname char(25), index number);
```

Listing 1.18: Tabellenfeld added

1.1.9 Tabelle Feldtyp ändern

```
sql>ALTER TABLE <TableName> MODIFY <FeldName> char(30);
```

Listing 1.19: Tabelle Feldtyp ändern

1.1.10 Tabelle Feldname ändern

```
sql>ALTER TABLE <TableName> RENAME COLUMN <FeldName> TO <FeldName>;
```

Listing 1.20: Tabelle Feldname ändern

1.1.11 Tabelle Kommentar angeben

```
sql>COOMENT ON TABLE <TableName> IS 'Hier steht der beschreibene Text';

sql>SELECT table_name, comments FROM user_tab_comments;
```

Listing 1.21: Tabelle Kommentar angeben

1.1.12 Tabelle Feldname Kommentar

```
sql>COMMENT ON COLUMN <FeldName> IS 'Hier steht ein Text';

sql>SELECT table_name, column_name, comments FROM user_col_comments;
```

Listing 1.22: Tabelle Feldname Kommnetar

1.1.13 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 und 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
```

Listing 1.23: Tabelle kopieren

1.1.14 Tabelle verschieben

Ein Tabelle kann in einem anderen Tablespace verschoben werden. Sind in der Tabelle Felder mit dem Type long, so funktioniert das verschieben nicht.

```
sql>ALTER TABLE <TableName> MOVE tablespace <NewTablespace>;
```

Listing 1.24: Tabelle verschieben

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

```
sql>ALTER TABLE <TableName> move;
```

Listing 1.25: Tabelle verschieben

1.1.15 Tabellenfeld löschen

```
sql>ALTER TABLE <TableName> DROP COLUMN <FeldName>;
```

Listing 1.26: Tabellenfeld löschen

1.2 Abfragen

1.2.1 Join Abfrage

```
sql>SELECT emp.empno, emp.ename, emp.depno, dept.dname
 2>FROM emp, dept
 3>WHERE emp.deptno = dept.deptno;
```

Listing 1.27: Join Abfrage

1.2.2 Abfrage mit SQL-Kommentar

```
sql>SELECT /*Abfrage1*/ vname
 2>FROM mitarbeiter;
```

Listing 1.28: Abfrage mit Kommentar

1.2.3 Abfrage mit Bindevariable

```
sql>var a varchar2(10);
sql>exec :a :='Uwe';
sql>SELECT vname, nname
 2>FROM mitarbeiter
 3>WHERE vname = :a;
```

Listing 1.29: Abfrage mit Bindevariable

1.2.4 Abfrage mit Like

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

```
sql>SELECT *
 2>FROM dba_users
 3>WHERE username like 'S%';

sql>SELECT *
 2>FROM dba_users
 3>WHERE username like 'S_Y%';
```

Listing 1.30: Abfrage mit Like

1.2.5 Abfrage mit In

```
sql>SELECT *
 2>FROM emp
 3>WHERE empno in (10,20);
```

Listing 1.31: Abfrage mit In

1.2.6 Abfrage mit Sysdate

```
sql>SELECT sql_text
 2>FROM v$sqlarea
 3>WHERE to_date(last_active_time)
 4>IN (select to_date(sysdate) FROM dual);

sql>SELECT start_time, end_time, status
 2>FROM v$rman_backup_job_details
 3>WHERE start_time > truncate(sysdate) -2;
```

Listing 1.32: Abfrage mit Sysdate

Anzahl der Tage Abfragen.

```
sql>SELECT sysdate - to_date('17.01.2012','DD.MM.YYYY') "Days"
 2>FROM dual;

Days
---------
126,52341
```

Listing 1.33: Abfrage Anzahl Tage

1.2.7 Abfrage mit Datum

```
sql>SELECT start_time, status
 2>FROM jobs
 3>WHERE 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
```

Listing 1.34: Abfrage mit Datum

1.2.8 Alias Abfrage

```
sql>SELECT p.empno, p.ename, p.deptno, t.dname
 2>FROM emp p, dept t
 3>WHERE p.deptno = t.deptno;
```

Listing 1.35: Alias Abfrage

1.2.9 Abfrage Substring

Möchte man einem Tabellefeld nur eine bestimmte Anzahl an Zeichen auslesen, so geschieht dises mit substr.
In dem ersten Beispiel werden ab dem zweiten Zeichen 5 Zeichen ausgelesen.

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

OS_System
---------
inux
```

Listing 1.36: Abfrage Substr

In dem zweiten Beispiel wird ab dem 10ten Zeichen alle Zeichen bis zum Ende ausgelesen.

```
sql>SELECT substr(platform_name, 10) "OS System"
 2>FROM v$database;

OS_System
------------
Bit for AMD
```

Listing 1.37: Abfrage Substr(10)

In dem letzten Beispiel werden die letzten 3 Zeichen ausgelesen.

```
sql>SELECT substr(platform_name, -3) "OS System"
 2>FROM v$database;

OS_System
---------
AMD
```

Listing 1.38: Abfrage Substr(-3)

1.2.10 Count Abfrage

```
sql>SELECT name, count(street)
 2>FROM city
 3>GROUP BY name;

NAME       COUNT
---------- -----
GOCH           3
Essen          5
Frankfurt      2

3 Zeilen ausgewaehlt
```

Listing 1.39: Count Abfrage

1.2.11 Lower / Upper Abfrage

```
sql>SELECT DISTINCT lower(substr(owner,4))
 2>FROM dba_tables
 3>WHERE owner like 'SP%';
```

Listing 1.40: Lower Abfrage

Alle Werte ausgeben, die Groß geschrieben sind.

```
sql>SELECT model_file_spec
 2>FROM pdtable_113
 3>WHERE model_file_spec = upper(model_file_spec);
```

Listing 1.41: Upper Abfrage

1.2.12 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 ausgewaehlt
```

Listing 1.42: Decode Abfrage

1.2.13 Minus Abfrage

```
sql>SELECT model_id, name
 2>FROM model_catalog
 3>MINUS
 4>(SELECT max(model_id), name FROM model_catalog WHERE name = 'VW')
 5>GROUP by name
 6>ORDER by name, model_id;

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

Listing 1.43: Minus Abfrage

1.2.14 Null Abfrage

```
sql>SELECT model_id, name
 2>FROM model_catalog
 3>WHERE
 4>status is not null;
```

Listing 1.44: Null Abfrage

1.2.15 Max Abfrage

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

```
sql>SELECT gv.output
 2>FROM gv$rman_output gv,
 3>(SELECT max(sessoin_recid) AS maxid FROM gv$rman_output) maxrecid
 4>WHERE gv.sessoin_recid = maxrecid.maxid;
```

Listing 1.45: Max Abfrage

Eine zweite Möglichkeit wäre.

```
sql>SELECT output
 2>FROM gv$rman_output
 3>WHERE sessoin_recid = (SELECT max(session_recid) FROM gv$rman_output);
```

Listing 1.46: Max Abfrage

1.2.16 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
```

Listing 1.47: Sum Abfrage

1.3 Werte eingeben / Ändern

1.3.1 Werte eingeben

Gibt es mehr Tabellenfelder in einer Tabelle als man mit Werten füllen möchte, so gibt man die zu befüllenden Felder an. Werde alle Felder einer Tabelle mit Werten gefüllt, so baucht man die Angabe der Felder nicht zu machen.

```
sql>INSERT INTO hr.mitarbeiter (vname, nname, year)
 2>VALUES ('Harry', 'Hirsch', 1979);

sql>INSERT INTO hr.mitarbeiter SELECT * FROM hr.employes WHERE id = 10;

sql>INSERT INTO hr.mitarbeiter
 2>VALUES ('Paul', 'Panther', to_date('03-NOV-09 11:40:00', 'DD-MM-YYYY HH24
    :MI:SS'));
```

Listing 1.48: Werte eingeben

1.3.2 Insert mit Select und Werte

```
sql>INSERT INTO t_sum_proj_user (ID, sdate, project, s3duser)
 2>VALUES (t_sum.nextval, sysdate, 'Zoll', (SELECT count(distinct user)
 3>FROM v$session
 4>WHERE schemaname like 'ZOLL%MDB'
 5>GROUP BY schemaname;
```

Listing 1.49: Insert Select

1.3.3 Werte aus einer anderen Tabelle übernehmen

```
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, substr(last_name,1,1), year
 3>FROM employees;
```

Listing 1.50: Werte übernehmen

Alle Werte aus einer anderen Tabelle können so übernommen werden.

```
sql>SELECT INTO hr.mitarbeiter SELECT * FROM employees;
```

Listing 1.51: Werte übernehmen

1.3.4 Insert multiple Values

Mehrere Datensätze in einer Tabelle einfügen kann man folgendermaßen machen.

```
INSERT ALL
INTO countries VALUES ('AR','Argentina',2)
INTO countries VALUES ('AU','Australia',3)
INTO countries VALUES ('BE','Belgium',1)
INTO countries VALUES ('BR','Brazil',2)
INTO countries VALUES ('CA','Canada',2)
INTO countries VALUES ('CH','Switzerland',1)
INTO countries (COUNTRY_ID, COUNTRY_NAME) VALUES ('CN','China')
SELECT * FROM dual;
```

Listing 1.52: Insert Multiple Rows

1.4 Update

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

```
sql>UPDATE <TableName> SET <ColumnName> = '50' || <ColumnName>
 2>WHERE <ColumnName> LIKE 'LBA%';

sql>UPDATE pdtable_80_22 SET pipe_support_no = '50' || pipe_support_no
 2>WHERE pipe_support_no LIKE 'LBA%';
```

Listing 1.53: Werte anfügen

Eingabe von Sonderzeichen.

```
sql>set define off
sql>UPDATE hr.mitarbeiter set descry = 'Mobil & Home'
 2>WHERE id=5;
```

Listing 1.54: Sonderzeichen

Suchen und ersetzen.

```
sql>UPDATE <TableName> SET <ColumnName> =
 2>REPLACE (<ColumnName>, 'was', 'mit')
 3>WHERE <ColumnName> LIKE '%path%';
```

Listing 1.55: Suchen und Ersetzen

1.5 Index

1.5.1 Index erstellen

```
sql>CREATE UNIQUE INDEX <IndexName> ON <TableName> (FeldName);

sql>CREATE INDEX <IndexName> ON <TableName> (FeldName)
 2>TABLESPACE <Name> STORAGE (initial 20k next 20k pctincrease 75);
```

Listing 1.56: Index erstellen

1.5.2 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$object_usage$;
```

Listing 1.57: Index Abfragen

```
sql>col index_name for a25 heading 'Index|Name'
sql>col index_type for a10 heading 'Index|Type'
sql>col table_owner for a8 heading 'Table|Owner'
sql>col table_name for a20 heading 'Table|Name'
sql>col table_type for a8 heading 'Table|Type'
sql>col column_name for a20 heading 'Column|Name'
sql>col tablespace_name for a20 heading 'Tablespace|Name'

sql>SELECT
 2> index_name,
 3> index_type,
 4> table_owner,
 5> table_name,
 6> table_type,
 7> uniqueness,
 8> tablespace_name
 9>FROM
10> user_indexes
11>ORDER BY
12> index_name;

sql>SELECT
 2> index_name,
 3> table_name,
 4> column_name
5>FROM
 6> user_ind_columns
7>ORDER BY
 8> index_name;
```

Listing 1.58: Beispiele

1.5.3 Index Rebuild

```
sql>ALTER INDEX <IndexName> rebuild [online];
```

Listing 1.59: Index rebuild

1.5.4 Index löschen

```
sql>DROP INDEX <IndexName>;
```

Listing 1.60: Index löschen

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

```
sql>ALTER TABLE <TableName> DROP CONSTRAINT <IndexName> cascade;

sql>ALTER TABLE mitarbeiter DROP CONSTRAINT mitarbeiter_id cascade;
```

Listing 1.61: Index Primary Key

1.5.5 Index umbenennen

```
sql>ALTER INDEX <IndexName> RENAME TO <IndexNameNew>;
```

Listing 1.62: Index rename

1.5.6 Index verschieben

```
sql>ALTER INDEX <IndexName> REBUILD tablespace <TablespaceNew>;
```

Listing 1.63: Index move

1.6 Constraint

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

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

Tabelle 1.2: Constraints

1.6.1 Constraint erstellen

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

```
sql>CREATE TABLE <TableName> (nr number NOT NULL, name varchar2(10));
```

Listing 1.64: Constraint erstellen

Das nachträgliche anlegen eines Constraint erfolgt mit dem Befehl `alter table`.

```
sql>ALTER TABLE <TableName>
 2>ADD (constraint 'sys_c003398' UNIQUE ('value_id'));
```

Listing 1.65: Constraint hinzufügen

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

```
sql>ALTER TABLE <TableName>
 2>ADD (constraint 'sys_c003398' FOREIGN KEY (<FeldName>)
 3>REFERENCE 'cv3d.ddd_mod_attr' (<FeldName>));
```

Listing 1.66: Constraint Foreign Key

1.6.2 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 = '<TableName>';
```

Listing 1.67: Constraint anzeigen

1.6.3 Constraint löschen

```
sql>ALTER TABLE <TableName> DROP CONSTRAINT <ConstraintName> CASCADE;
```

Listing 1.68: Constraint löschen

1.6.4 Constraint deaktivieren / aktivieren

```
sql>ALTER TABLE <TableName> DISABLE CONSTRAINT <ConstraintName>;

sql>ALTER TABLE <TableName> ENABLE CONSTRAINT <ConstraintName>;
```

Listing 1.69: Constraint

1.7 Views

1.7.1 Anzeigen

Erstellte Views werden in den Tabellen `user_views`, `dba_views` und `all_views` verwaltet.

```
sql>SELECT view_name, text FROM user_views;

VIEW_NAME   Text
----------- --------------------------------------------------
V$SHOW_ADDR SELECT strName, strStreet, strCity FROM t_adressen;
```

Listing 1.70: User Views

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

```
sql>set long 2000
sql>SELECT owner, mview_name, query FROM all_mviews;

OWNER   NVIEW_NAME  query
uws     MV$ADRESS   SELECT "T_ADRESSEN"."STRNAME" "NAME"
                         ,"T_ADRESSEN"."STRSTREET" "STRASSE"
                         ,"T_ADRESSEN"."STRCITY" "STADT"
                    FROM "T_ADRESSEN";
```

Listing 1.71: Materialized Views

1.7.2 Erstellen

Ein View kann mit `create view` erstellt werden.

```
sql>CREATE VIEW <ViewName> AS SELECT <FeldName>, <FeldName>
 2>FROM <TableName>
 3>WHERE <FeldName> = <Wert>;
```

Listing 1.72: Create View

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

```
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;
```

Listing 1.73: Create View

1.7.3 Löschen

Ein View kann mit dem `drop view` Befehl gelöscht werden.

```
sql>DROP VIEW <ViewName;

sql>DROP MATERIAKIZED VIEW <MviewName>
```

Listing 1.74: View löschen

1.8 Sequence

1.8.1 Sequence erstellen

Sequenzen sind Generatoren für nummerische 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 1 start with 1;
```

Listing 1.75: Sequence erstellen

increment by 1	Schrittgröße beim Hochzählen
start with 1	Startwert
minvalue 1	Kleinster Wert
maxvalue 9999	Größter Wert
cycle/nocycle	Wieder bei minvalue starten, wenn maxvalue überschritten wird

Tabelle 1.3: Constraints

1.8.2 Sequence ändern

Mit `alter sequence` kann man Werte der Sequence ändern. Folgende Werte können mit diesem Befehl verändert werden:

- INCREMENT BY <Wert>
- MAXVALUE <Wert>
- NO MAXVALUE
- MINVALUE <Wert>
- NOMINVALUE
- NOCACHE

```
sql>ALTER SEQUENCE hr.mitarbeiter_s INCREMENT BY 2;
```

Listing 1.76: Sequence ändern

1.8.3 Sequence löschen

```
sql>DROP SEQUENCE hr.mitarbeiter_s;
```

Listing 1.77: Sequence löschen

1.8.4 Sequence anzeigen

```
sql>SELECT * FROM user_sequences;

sql>SELECT <SequenceName>.nextval FROM dual;

sql>SELECT <SequenceName>.currval FROM dual;
```

Listing 1.78: Sequence anzeigen

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

1.8.5 Sequence anwenden

```
sql>INSERT INTO <TableName>
 2>(num, name) VALUES (<SequenceName>.nextval, 'Test');
```

Listing 1.79: Sequence anwenden

1.9 Ausgaben formatieren

1.9.1 Spaltenformatieren

Als Ausgabeformate gibt es Alphanumerische (a20) und Integer (9999). Als Abkürzung von Format kann man auch For nehmen.

```
sql>COL <FeldName> FORMAT a20

sql>COL <FeldName> FORMAT 99,99.99

sql>COL "File Size" FORMAT a12

sql>SELECT tablespace_name, bytes "File Size" FROM dba_data__files;

sql>COL session_recid FORMAT 999999 HEADING "SESSION|ID" -- Ueberschrfit 2
   zeilig

sql>COL <FeldName> FORMAT a30 HEADING "Feld1" JUSTIFY right -- Ausrichtung
   rechts

sql>COL <FeldName> FORMAT a20 TRUNC -- Ausgabe wird abgeschnitten
```

Listing 1.80: Beispiele Formatierung

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

1.9.2 Zeilen / Seite

Ausgabe der Breite, wie viele Zeichen in einer Zeile ausgegeben werden sollen, bevor ein Zeilenumbruch kommt.

```
sql>SET linesize 150
```

Listing 1.81: Zeile

Ausgabe der Anzahl der Zeilen, wenn wieder die Tabellenüberschrifen angezeigt werden.

```
sql>SET pagesize 80
```

Listing 1.82: Seite

1.9.3 Datum lesbar ausgeben

```
sql>SELECT to_date(sysdate, 'DD.MM.YYYY HH24:MI:SS') FROM dual;

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

sql>SELECT to_char(sysdate, 'DD.MM.YYYY') FROM dual;

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

Listing 1.83: Datum ausgeben

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

```
sql>SELECT <FeldName> 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);
```

Listing 1.84: Trunc

Eine Auswahl der Datumsformate stehen in der nachfolgenden Tabelle.

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

Tabelle 1.4: Formate

1.9.4 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     100MN
```

Listing 1.85: Bytes in MB

1.9.5 Datatype Clob/Nclob/Long

Sind in einer Tabelle Felder erstellt worden, die als Datatype `Clob, Nclob oder 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;
```

Listing 1.86: Set long

1.10 External Tables

Mit external Tables kann man Text Dateien in die Datenbank einbinden. Diese Text Dateien können als CSV-Dateien mit einem Trennzeichen zwischen den Daten oder mit einer festen Spaltenbreite 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 NULL gestellt werden, ansonsten kommt es zu der Nachfolgenden Fehlermeldung:
ORA-2000: Unable to analyse TABLE «owner>".«table>", sampling on external table is not supported.

1.10.1 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/data';

sql>CREATE OR REPLACE bad_path AS '/u01/ext/data';

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;
```

Listing 1.87: Vorarbeiten

1.10.2 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>);
```

Listing 1.88: Tabelle Spalten

```
uws@woby1002>cat user.dat
10  Harry       Hirsch    Waldsiedlung    16.02.1967
```

```
33  Paulchen    Panther    Fernsehstudio    31.05.1953
98  Karl-Heinz  May        Wild-West        05.11.1912
```

Listing 1.89: user.dat

In dem nächten Beispiel sind die Daten in der Text Datei mit einem Semikolon 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;
```

Listing 1.90: Tabelle Semikolon

Der Inhalt der Text Dateien 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@woby1002>cat user2.txt
72;Bernd;Mueller;Hamburg;04.08.1977
12;Anton;Meise;Muenchen;19.11.1932
```

Listing 1.91: user1.dat user2.txt

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 zugegrifen werden. Mit dem Treiber `Oracle_datapump` kann auch schreibend auf die Date zugegriffen werden. Nur hier müssen die Daten im 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 verworfene Datensätze
PARALLEL	Das parallele Laden wird erlaubt, bei großen datenmengen ist das von Vorteil

Tabelle 1.5: Parameter

1.10.3 Abfrage

Beispiel einer Abfrage.

```
sql>SELECT * from users_load ORDER BY id;

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

Listing 1.92: Abfrage

1.10.4 Ändern

Die Eigenschaften der Tabelle kann mit `alter table` geändert werden.
Beispiel Default Directory

```
sql>ALTER TABLE user_load DEFAULT DIRECTORY users_path;
```

Listing 1.93: Default Directory

Beispiel Access Parameters.

```
sql>ALTER TABLE user_load ACCESS PARAMETERS (fields terminated by ':');
```

Listing 1.94: Access Parameters

Beispiel Location

```
sql>ALTER TABLE user_load LOCATION ('liste.txt', liste2.dat');
```

Listing 1.95: Location

Beispiel Reject Limit

```
sql>ALTER TABLE user_load REJECT LIMIT 400;
```

Listing 1.96: Reject Limit

Beispiel Parallel.

```
sql>ALTER TABLE user_load NOPARALLEL;
```

Listing 1.97: Parallel

1.10.5 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);
```

Listing 1.98: Laden External Datei

1.10.6 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, locations, 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
```

Listing 1.99: Info

1.11 Spool Datei erstellen

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

```
sql>SET newpage 0
sql>SET space 0
sql>SET pagesize 0
sql>/* Set linesize 150 */
sql>SET feedback off
sql>SET heading off
sql>SET echo off
sql>SET show off
sql>SET termout off
sql>SET verify off
sql>SET trimspool on
sql>Spool c:\temp\uschi.sql
sql>SELECT 'delete from <TableName> WHERE <feld_name> = ' || <FeldName> ||
 2>';' FROM <table_name> where (sysdate - createdate)>5;
sql>Select 'commit;' com from dual;
sql>Spool off
sql>SET feedback on
sql>SET heading on
sql>SET echo on
sql>SET termout on
```

Listing 1.100: Spool Datei

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

```
sql>SELECT 'delete from hr.mitarbeiter where plz = ' || plz || ';'
 2>from hr.mitarbeiter where plz = 4711;
```

Listing 1.101: Spool Delete Query

Möchte man an einer vorhandenen Datei die SQL Ausgabe 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
sql>SELECT status FROM v$instance;
sql>spool off
```

Listing 1.102: Spool Append

Die Leerzeichen am Ende eines Feldes kann man folgendermaßen abgeschnitten werden.

```
sql>SELECT path_name || '' || file_name FROM files;
```

Listing 1.103: Spool Leerzeichen

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

```
sql>SELECT 'commit force ''' || local_tran_id || ''';' FROM
 2>dba_2pc_pending where state = 'prepared';
```

```
sql>commit force '2016.1315.01';
```

Listing 1.104: Spool Hochkommata

Bei der Angabe von 4 Hochkommatas werden 2 ausgegeben.

```
sql>SELECT 'Paul ''''the'''' goal keeper' AS text FROM dual;
text
------------------------
Paul ''the'' goal keeper
```

Listing 1.105: Spool 2 Hochkommatas

Einen neuere Form ist.

```
sql>SELECT q'$name's with$' name FROM dual;
NAME
-----------
Name's with
```

Listing 1.106: Spool Hochkomma

1.12 Tools

1.12.1 Befehl editieren

Den letzten Befehl kann man im Editor bearbeiten, wenn man den Befehl `ed` eingibt. 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
```

Listing 1.107: Befehl editieren

1.12.2 Shell aufrufen

Möchte man kurz auf die Shell wechseln, so gibt man den Befehl `host` an. Zurück in der aktuellen SQL-Session geht es dann mit `exit`.

```
sql>host

uws@tux>
```

Listing 1.108: Shell aufrufen

1.12.3 Shell Befehl ausführen

Einen Shell Befehl in einer SQL-Sesion ausführen, kann man in dem nächsten Beispiel sehen.

```
sql>! whoami
oracle

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

Listing 1.109: Shell Befehl ausführen

1.13 Variablen

Die Ausgabe `alt` und `neu` kann man mit `verify off` ausschalten.

1.13.1 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 fuer 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
```

Listing 1.110: Variable mit &

1.13.2 Eingabeaufforderung mit Accept und Prompt

Nach der Eingabe des Befehls `accept (acc)`, kann man den Wert einer Variable eingeben, ohne das 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
uws@tux>
```

Listing 1.111: Accept

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
uws@tux>
```

Listing 1.112: Prompt

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

```
sql>SET verfify off
sql>accept pwd char prompt 'Please enter password: ' hide
```

Listing 1.113: Beispiel Accept

Die Eingabe kann auch vorher formatiert 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: '
```

Listing 1.114: Beispiel Formatierung

Diese beiden Befehle eignen sich hervorragend für das Ausführen eines SQL-Scripts.

```
uws@tux>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
```

Listing 1.115: Beispiel Script

1.13.3 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
```

Listing 1.116: Define

1.13.4 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 einen Wert fuer a ein: 33
alt 1: SELECT &a.000 FROM dual;
neu 1: SELECT 33.000 FROM dual;
```

```
33000
-----
33000

sql>undefined a
sql>SELECT '%a.laub' FROM dual;
Geben Sie einen Wert fuer a ein: Ur
Alt 1: SELECT '%a.laub' FROM dual;
Neu 1: SELECT 'Urlaub' FROM dual;

URLAUB
------
Urlaub
```

Listing 1.117: Trennzeichen

1.13.5 Variablen weiterverarbeiten

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

```
uws@tux>cat FreeSpace.sql
PROMPT
PROMPT Loading FreeSpace.sql
PROMPT
set verify off
variable intFree number;
variable intSize number;
variable intDiff number;
variable intUsed number;
cal 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;
```

Listing 1.118: Trennzeichen

1.13.6 Variablen umwandeln

Möchte man eine Eingabe in Großbuchstaben umwandeln, so geschieht das mit 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');
```

Listing 1.119: Trennzeichen

1.14 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 ausgewaehlt
```

Listing 1.120: Befehle ausführen

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 der Block sofort ausgeführt. Anstelle des Slash kann 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 ausgewaehlt
```

Listing 1.121: Befehle mehrzeilig

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
sql>SELECT
 2>username, account_status
 3>FROM dba_users WHERE username = 'SYS'
 4>*

USERNAME ACCOUNT_STATUS
-------- --------------
SYS      open

1 Zeile ausgewaehlt

sql>/
USERNAME ACCOUNT_STATUS
-------- --------------
SYS      open
```

```
1 Zeile ausgewaehlt
```

Listing 1.122: Befehl wiederholen

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

```
sql>r
sql>SELECT
 2>username, account_status
 3>FROM dba_users WHERE username = 'SYS'
 4>*
sql>SAVE /home/uws/sql/sys.sql [replace]

sql>GET /home/uws/sql/sys.sql
 1>SELECT
 2>username, account_status
 3>FROM dba_users WHERE username = 'SYS'
 4>*
USERNAME ACCOUNT_STATUS
-------- --------------
SYS      open

1 Zeile ausgewaehlt

sql>START /home/uws/sql/sys.sql

USERNAME ACCOUNT_STATUS
-------- --------------
SYS      open

1 Zeile ausgewaehlt

sql>@/home/uws/sql/sys.sql
USERNAME ACCOUNT_STATUS
-------- --------------
SYS      open

1 Zeile ausgewaehlt
```

Listing 1.123: Befehl Save / Load

1.15 SQL-Scripte

1.15.1 Kommentare

Um Kommentare in einem SQL-Script zu schreiben, gibt es insgesammt 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 setzen, 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 aufhoeren wollen */

SELECT username FROM dba_users; -- Welche Oracle User gibt es.
```

Listing 1.124: Kommentare

1.15.2 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 der Ausgabe dann `DOC>` steht.

```
uws@tux>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

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

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

Listing 1.125: DOC Text

Eine Text Datei kann auch ausgegeben werden.

```
uws@tux>cat ScriptHeader.txt
Hier steht ein Beispieltext.

++++++++++++++++++++++++
+ Author: Uwe Schimanski
++++++++++++++++++++++++

uws@tux>cat ShowText.sql
host cat ScriptHeader.txt
```

Listing 1.126: Datei ausgeben

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 Ausgeba möglich.

```
uws@tux>cat ShowTextScript.sql
host ~/sql/ScriptHeader.sh
```

Listing 1.127: Script ausführen

Mit LPAD kann man auch einen 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('Start Program *****',25) || (to_char(sysdate, 'DD.MM.YYYY
   HH24:MI:SS',26) FROM dual;

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

Listing 1.128: LPAD

1.15.3 Parameter übergeben

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

```
uws@tux>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 :int_Id FROM dual;
SELECT username FROM dba_users WHERE user_id = :intId;
```

Listing 1.129: Parameter übergeben

Möchte man bei dem Ausführen des SQL-Scripts Parameter übergeben, so werdem diese Parameter hinter dem Aufruf angehangen.

```
uws@tux>cat parameter.sql
SELECT '&1' FROM &2;

uws@tux>sqlplus / @parameter.sql '"Parameter 1"' dual

uws@tux>cat cmdspool.sql
spool &1/queryuser.sql
SELECT * FROM dba_users;
spool off

uws@tux>sqlplus / @cmdspool.sql $temp
```

Listing 1.130: Parameter

1.15.4 Fehler Abbruch

Tritt in einem SQL-Script ein Fehler auf, so wird der Fehler übersprungen und der nächste Befehl wird ausgeführt. Soll die Verarbeitung des Scripts bei einem Fehler abbrechen, so gibt man am Anfang der Scripts die `whenever` Anweisung an.

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

Listing 1.131: Fehler

Soll die Verarbeitung trotz eines Fehlers zugelassen werden, so gibt man die Option `continue` mit an. Diese Anweisung bezieht auch auf alle achfolgenden SQL-Befehle.

```
whenever sqlerror continue;
```

Listing 1.132: Continue

1.15.5 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.

```
uws@tux>sqlplus /
sql>exit
uws@tux>echo $?
0

uws@tux>sqlplus /
sql>exit 48
uws@tux>echo $?
48

uws@tux>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
uws@tux>echo $?
66

uws@tux>sqlplus /
sql>help exit

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

Listing 1.133: Exit Code

1.15.6 Cursor definieren

Es können `Cursor-Variablen` in Sql 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

 uws@tux>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
```

Listing 1.134: Cursor definieren

1.15.7 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
```

Listing 1.135: Case Anweisung

```
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
```

Listing 1.136: Case Beispiele

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;
```

Listing 1.137: Case Alternativr

Vergleich von zwei Feldern.

```
sql>SELECT street
 2>CASE
 3> WHEN company = 'Siemens' and city = 'Muenschen' 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;
```

Listing 1.138: Case Bedinging

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;
```

Listing 1.139: Case Format

1.16 Proceduren

1.16.1 Erstellen

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

```
CREATE OR REPLACE PROCEDURE set_export_date_navis
IS
/* In einer Procedure darf kein Declare stehen, sonst gibt es
   ein Kompilierungs Fehler
*/
  run_status number(1);
BEGIN
  SELECT auto_run INTO run_status FROM mca_prep WHERE id = 3;
  IF run_status = 1 THEN
    UPDATE
      cae_view.mca_prep
    SET
      start_datum = to_char(sysdate -1, 'DD.MM.YYYY'),
      end_datum = to_char(sysdate -1, 'DD.MM.YYYY'),
      enddd = to_char(sysdate -1, 'DD.MM.YYYY')
    WHERE
      ID = 3;
    INSERT INTO
      cae_view.mca_prep_date_logging (id, ldate, ltext)
    VALUES
      (cae_view.T_MCA_PREP_DATE_LOGGING_S.NEXTVAL, sysdate, 'Change Date to
         yesterday');
  ELSE
    INSERT INTO
      cae_view.mca_prep_date_logging (id, ldate, ltext)
    VALUES
      (cae_view.T_MCA_PREP_DATE_LOGGING_S.NEXTVAL, sysdate, 'Run Status is
         0, date not revised');
  END IF;
  COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    run_status := NULL;
END;
/
```

Listing 1.140: Beispiel Procedure

1.16.2 Exception

Für eine Fehlerbehandlung kann man `exceptions` abfangen. In der nachfolgenden Liste sind die Exceptions aufgelistet.

Exception	Beschreibung
ACCESS_INTO_NULL	Ihr Programm versucht, den Attributen eines nicht initialisierten (atomar null) Objekts Werte zuzuweisen.
CASE_NOT_FOUND	Keine der Auswahlmöglichkeiten in den WHEN-Klauseln einer CASE-Anweisung ist ausgewählt, und es gibt keine ELSE-Klausel.
COLLECTION_IS_NULL	Ihr Programm versucht, andere Auflistungsmethoden als EXISTS auf eine nicht initialisierte (atomar Null) verschachtelte Tabelle oder Varray anzuwenden, oder das Programm versucht, den Elementen einer nicht initialisierten verschachtelten Tabelle oder Varray Werte zuzuweisen.
CURSOR_ALREADY_OPEN	Ihr Programm versucht, einen bereits geöffneten Cursor zu öffnen. Ein Cursor muss geschlossen werden, bevor er wieder geöffnet werden kann. Eine Cursor-FOR-Schleife öffnet automatisch den Cursor, auf den sie verweist. Ihr Programm kann diesen Cursor also nicht innerhalb der Schleife öffnen.
DUP_VAL_ON_INDEX	Ihr Programm versucht, doppelte Werte in einer Datenbankspalte zu speichern, die durch einen eindeutigen Index eingeschränkt ist.
INVALID_CURSOR	Ihr Programm versucht eine unzulässige Cursoroperation wie das Schließen eines ungeöffneten Cursors.
INVALID_NUMBER	In einer SQL-Anweisung schlägt die Konvertierung einer Zeichenfolge in eine Zahl fehl, da die Zeichenfolge keine gü"ltige Zahl darstellt. (In prozeduralen Anweisungen wird VALUE_ERROR ausgelöst.) Diese Ausnahme wird auch ausgelöst, wenn der LIMIT-Klauselausdruck in einer Bulk FETCH-Anweisung nicht zu einer positiven Zahl ausgewertet wird.
LOGIN_DENIED	Ihr Programm versucht, sich mit einem ungültigen Benutzernamen und / oder Passwort an Oracle anzumelden.
NO_DATA_FOUND	Eine SELECT INTO-Anweisung gibt keine Zeilen zurück, oder Ihr Programm verweist auf ein gelöschtes Element in einer geschachtelten Tabelle oder auf ein nicht initialisiertes Element in einer Index-by-Tabelle. SQL-Aggregatfunktionen wie AVG und SUM geben immer einen Wert oder eine Null zurück. Eine SELECT INTO-Anweisung, die eine Aggregatfunktion aufruft, löst daher NO_DATA_FOUND nicht aus. Es wird erwartet, dass die FETCH-Anweisung letztendlich keine Zeilen zurückgibt. Wenn dies geschieht, wird keine Ausnahme ausgelöst.
NOT_LOGGED_ON	Ihr Programm gibt einen Datenbankanruf aus, ohne mit Oracle verbunden zu sein.
PROGRAM_ERROR	PL / SQL hat ein internes Problem.
ROWTYPE_MISMATCH	Die Host-Cursor-Variable und die PL / SQL-Cursor-Variable, die an einer Zuweisung beteiligt sind, haben inkompatible Rückgabetypen. Wenn beispielsweise eine offene Host-Cursor-Variable an ein gespeichertes Unterprogramm übergeben wird, müssen die Rückgabetypen der tatsächlichen und der formalen Parameter kompatibel sein.
SELF_IS_NULL	Ihr Programm versucht, eine MEMBER-Methode für eine Nullinstanz aufzurufen. Das heißt, der integrierte Parameter SELF (der immer der erste Parameter ist, der an eine MEMBER-Methode übergeben wird) ist null.
STORAGE_ERROR	PL / SQL hat nicht genügend Speicher oder Speicher ist beschädigt.

Continued on next page

continued from previous page.

Exception	Beschreibung
SUBSCRIPT_BEYOND_COUNT	Ihr Programm verweist auf eine geschachtelte Tabelle oder ein varray-Element, das eine Indexnummer verwendet, die größer als die Anzahl der Elemente in der Auflistung ist.
SUBSCRIPT_OUTSIDE_LIMIT	Ihr Programm referenziert eine geschachtelte Tabelle oder ein varray-Element unter Verwendung einer Indexnummer (z. B. -1), die außerhalb des zulässigen Bereichs liegt.
SYS_INVALID_ROWID	Die Umwandlung einer Zeichenfolge in eine universelle Zeilennummer schlägt fehl, da die Zeichenfolge keine gültige Zeilennummer darstellt.
TIMEOUT_ON_RESOURCE	Eine Zeitüberschreitung tritt auf, während Oracle auf eine Ressource wartet.
TOO_MANY_ROWS	Eine SELECT INTO-Anweisung gibt mehr als eine Zeile zurück.
VALUE_ERROR	Ein Arithmetik-, Konvertierungs-, Trunkierungs- oder Größenbeschränkungsfehler tritt auf. Wenn Ihr Programm beispielsweise einen Spaltenwert in eine Zeichenvariable auswählt, bricht PL / SQL die Zuweisung ab und löst VALUE_ERROR aus, wenn der Wert länger als die deklarierte Länge der Variablen ist. In prozeduralen Anweisungen wird VALUE_ERROR ausgelöst, wenn die Umwandlung einer Zeichenfolge in eine Zahl fehlschlägt. (In SQL-Anweisungen wird INVALID_NUMBER ausgelöst.)
ZERO_DIVIDE	Ihr Programm versucht, eine Zahl durch Null zu teilen.

Tabelle 1.6: Exceptions

Nun folgt ein Beispiel einer Exception Behandlung.

```
CREATE OR REPLACE PROCEDURE Count_Project_User
IS
  std_sum number :=0;
  v_s3duser T_SUM_PROJ_USER.S3DUSER%TYPE;
  cursor projname is select DISTINCT(substr(schemaname, 1, length(schemaname
     ) -4)) as sproj from v$session where schemaname like '%MDB';
  v_proj varchar(25);
  cuser number(3,0);
BEGIN
  FOR l_proj IN projname LOOP
    BEGIN
      select count(distinct username) into cuser from v$session where
         schemaname like l_proj.sproj||'_MDB';
      EXCEPTION
        WHEN NO_DATA_FOUND THEN
          cuser := NULL;
    END;
    IF cuser > 0 THEN
      insert into T_SUM_PROJ_USER (ID, SDATE, PROJECT, S3DUSER) values (
         T_SUM_PROJ_USER_S.NEXTVAL, SYSDATE, l_proj.sproj, cuser);
    END IF;
    --dbms_output.put_line('Project: ' ||l_proj.sproj);
    --dbms_output.put_line('  Users: ' || cuser);
  END LOOP;
END;
/
```

Listing 1.141: Exception

1.16.3 Anzeigen

Um sich die Proceduren in der Datenbank anzeigen zu lassen, gibt es die Tabellen `dba_objects`, `all_source` und `user_source`.

```
sql>SELECT *
 2>FROM dba_objects
 3>WHERE object_type in ('PROCEDURE','PACKAGE','FUNCTION','PACKAGE_BODY');

sql>SELECT text
 2>FROM all_source
 3>WHERE name = 'ProcedureName'
 4>ORDER BY line;
```

Listing 1.142: Proceduren anzeigen

1.16.4 Verschlüsseln

Mit dem Programm `wrap.exe` kann man `Funktionen, Proceduren, Packages` und Types verschlüsseln, um sie im Anschluß in der Datenbank zu laden.

```
uws@tux>wrap iname=MyFunction.sql oname=myFunction.plb

sql>-- Laden der Datei
sql>@myFunction.plb
```

Listing 1.143: Proceduren verschlüsseln"

1.16.5 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
 2>FROM user_objects
 3>WHERE status = 'INVALID' and
 4>object_type in ('PROCEDURE','PACKAGE','FUNCTION');

sql>SELECT object_type, object_name, status, owner
 2>FROM dba_objects
 3>WHERE status = 'INVALID' and
 4>object_type in ('PROCEDURE','PACKAGE','FUNCTION');
```

Listing 1.144: Invalid Objects

Werden bei der Abfrage `Invalid` Objekte angezeigt, so kann man diese neu kompilieren.

```
sql>ALTER PACKAGE <schema>.<PackageName> compile;
```

Listing 1.145: Invalid Objects compile

1.17 Database Link

Mit einem `Datenbank Link` 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`.

1.17.1 Erstellen

Mit `create database link` wird ein neuer Link angelegt. Die Syntax hierzu lautet:
`CREATE [shared] [public] DATABASE LINK <LinkName> CONNECT to <User> IDENTIFIED BY <Passwd> using <ConnectString>`

```
sql>CREATE DATABASE LINK woby1003
 2>CONNECT TO uws IDENTIFIED by password
 3> USING 'cad10';
```

Listing 1.146: Link erstellen

1.17.2 Anzeigen

Angelegte Links lassen sich mit den folgenden Views anzeigen.

Tabelle	*Beschreibung*
dba_db_links	Liste aller db Links
all_db_links	Liste aller Links
user_db_links	User erstellte Links
v$dblink gv$dblink	Liste aller offenen DB Links in der Session, wenn IN_TRANSACTION = yes ist

Tabelle 1.7: Link Tabellen

```
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
```

Listing 1.147: Trigger anzeigen

1.17.3 Löschen

```
sql>DROP DATABSE LINK <LinkName>;
```

Listing 1.148: Link löschen

1.18 Scheduled Jobs

Soll in einer Datenbank Zeitgesteuert Proceduren ausgeführt werden, so kann man das mit `Scheduled Jobs` erledigen. Um als User Jobs anlegen zu koennen, muss man folgenden Privilegien haben.

Privileg	*Beschreibung*
create job	Create scheduler Job rights
create external job	Create external job rights
manage scheduler	Verwalten von Jobklassen und Windows
scheduler_admin	Admin

Tabelle 1.8: Privilegien

1.18.1 Job erstellen

Einen Scheduler Job kann man mit dem Package `DBMS_SCHEDULER` erstellen.

```
prompt Create Cron Job
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
    job_name    => 'count_user_project',
    job_type    => 'STORED_PROCEDURE',
    job_action => 'PROJUSER.Count_Project_User',
    number_of_arguments => 0,
    start_date      => '07.09.2018 11:40:00 Europe/Berlin', /* NULL */
    repeat_interval => 'FREQ=MINUTELY;INTERVAL=5', /* every 5 minutes */
    end_date        => NULL,
    auto_drop       => FALSE,
    enabled         => TRUE,
    --job_class       => 'count_user_job',
    comments        => 'Count User for each Project'
  );
END;
/
```

Listing 1.149: Beispiel Scheduled Job

In der nachfolgenden Tabelle sind einige Beispiele für die Angabe der Wiederholungen angegeben.

Repeat Interval	*Beschreibung*
freq=secondly	Run every second
freq=minutely; interval=5	Run every 5 minutes
freq=hourly	Run every hour
freq=daily; byhour=3	Run at 3 am every day
freq=daily; byhour=8,20	Run at 8 am and 8 pm every day
freq=daily; interval=10	Run every 10 days
freq=monthly; bymonthday=1	Run on the first day of every month
freq=weekly; byday=fri	Run every friday
freq=weekly; byday=tue; byhour=6; byminute=23	Run every Tuesday at 06:23
freq=monthly; bymonthday=-1	Run on the last day of every month
freq=yearly; bymonth=sep; bymonthday=20	Run yearly on September 20th

Tabelle 1.9: Repeat Interval

1.18.2 Jobs auflisten

Einen einzelnen Job wird mit der nachfolgenden Abfrage angezeigt. Möchte man alle Jobs sehen, läst man die WHERE Bedingung weg.

```
col owner for a10
col job_action for a30
SELECT
  owner,
  job_name,
  job_action,
  to_char(start_date, 'DD.MM:YYYY HH24:MI:SS'),
  to_char(next_run_date, 'DD.MM.YYYY HH24:MI:SS'),
  state
FROM
  dba_scheduler_jobs
WHERE
  job_name = 'J_SET_EXPORT_DATE_NAVIS';
/
```

Listing 1.150: Job auflisten

Details eines Scheduler Jobs kann man sich mit dem nächsten Statement anzeigen lassen.

```
col log_date for a35
col job_name for a30
col additional_info for a40
col status for a10
SELECT
  log_date,
  owner,
  job_name,
  status,
  error#,
  additional_info
FROM
  dba_scheduler_job_run_details
WHERE
  job_name = 'J_SET_EXPORT_DATE_NAVIS';
```

Listing 1.151: Job Detail

1.18.3 Job ändern

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
    name      => 'j_set_export_date_navis',
    attribute => 'start_date',
    value     => '27.09.2018 10:00:00 Europe/Berlin');
END;
/
```

Listing 1.152: Job ändern

1.18.4 Job ausführen

```
BEGIN
  DBMS_SCHEDULER.RUN_JOB (
    JOB_NAME       => 'J_SET_EXPORT_DATE_NAVIS',
    USE_CURRENT_SESSION => FALSE);
END;
/
```

Listing 1.153: Job ausführen

1.18.5 Job löschen

```
BEGIN
  DBMS_SCHEDULER.DROP_JOB ('J_SET_EXPORT_DATE_NAVIS');
END;
/
```

Listing 1.154: Job löschen

1.18.6 Job enable / disable

```
BEGIN
  DBMS_SCHEDULER.ENABLE ('J_SET_EXPORT_DATE_NAVIS');
END;
/
```

Listing 1.155: Job enable

```
BEGIN
  DBMS_SCHEDULER.DISABLE ('J_SET_EXPORT_DATE_NAVIS');
  /* separate with komma for more jobs to disable ('job1, job2') */
END;
/
```

Listing 1.156: Job disable

1.19 Trigger

1.19.1 Erstellen

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

```
uws@tux>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;
/
```

Listing 1.157: Trigger erstellen

1.19.2 Anzeigen

Es gibt zwei Tabellen, indem die Trigger verwaltet werden. In der `dba_triggers` stehen alle Trigger drin und in der `user_triggers` nur des Benutzers.

```
sql>SELECT * FROM dba_triggers;

sql>SELECT * FROM user_triggers;
```

Listing 1.158: Trigger anzeigen

Möchte man sich anzeigen lassen, was für den Trigger definiert worden ist, so kann man das mit der folgenden Abfrage machen.

```
sql>set long 2000
sql>col description for a100
```

```
sql>SELECT description, trigger_body FROM all_triggers WHERE owner = 'UWS';
```

Listing 1.159: Trigger Text

1.19.3 Löschen

Mit `drop trigger` werden erstellte Trigger gelöscht.

```
sql>DROP TRIGGER <TriggerName>;
```

Listing 1.160: Trigger löschen

1.19.4 Aktivieren

Trigger kann man mit `enable / disable` Aktivieren und Deaktivieren.

```
sql>ALTER TRIGGER <TriggerName> [enable|disable];
```

Listing 1.161: Trigger aktivieren

1.20 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
```

Listing 1.162: HTML Seite erstellen

1.21 Reports

1.21.1 Allgemein

In Oracle kann man einfache Reports erstellen. Für die Formatierung des Reports gibt für den Kopf des Reports den Befehl `ttitle` und für Fußtext den Befehl `btitle`. Die Reports kann man hervorragend in einer Spool Datei schreiben.

1.21.2 Report erstellen

Mit `ttitle` wird die Kopf definition eingeleitet. Den Text kann man mit `LEFT, CENTER` und `RIGHT` ausrichten. Mit `SKIP n` fügt man Leerzeilen an. Mehrzeilige Befehle kann man mit einem - an Ende der Zeile einleiten.
An Options gibt es noch `BOLD, COL n, TAB n, FORMAT text`. In der SET Zeile der Parameter `recsep`

Variablen	*Beschreibung*
SQL.LNO	Die aktuelle Zeilen Nummer
SQL.PNO	Die aktuelle Seiten Nummer
SQL.USER	Der aktuelle User Name
SQL.RELEASE	Die aktuelle Oracle Release Nummer
SQL.SQLCODE	Der aktuelle Error Code

Tabelle 1.10: Variablen

ist für den Zeilenumbruch zuständig. Die aktuelle Einstellung kann man mit `show recsep` sich anzeigen lassen. Folgende recsep Werte gibt es:

- wr[apped] => Eine Leerzeile wird eingefügt, wenn ein Zeilenumbruch erfolgt
- ea[ch] => Es wird immer eine Leerzeile eingefügt.
- off => Keine Leerzeile wird eingefügt

Mit dem Parameter `recsepchar` kann man das Zeichen für die Leerzeile definieren. Standard ist `(hex20)` und kann mit `show recsepchar` abgefragt werden.
Mit `underline` kann man ein neues Zeichen definieren, das unter der Feldbeschreibung gesetzt werden soll. Der Standard ist ein `-`.
Weitere Infos über Reports, SET System Variablen gibt es hier:
`https://docs.oracle.com/cd/B19306_01/server.102/b14357/ch6.htm` `https://dbs.informatik.uni-halle.de/Lehre/PrakDB_SS08/5_SQL_Skript.pdf` `https://docs.oracle.com/cd/B19306_01/server.102/b14357/ch12040.htm`

```
sql>SET linesize 80 pagesize 35 recsep off underline =
sql>TTITLE "Report Salary" RIGHT "Page:" FORMAT 99 SQL.PNO SKIP 1-
 2>Center "Salary in $" SKIP 2 -
 3>LEFT "Author: UWS" SKIP 2
 4>BTITLE LEFT "Version: 18.12.19"
 5>SELECT
 6> JOB_ID,
 7> EMPLOYEE_ID,
 8> LAST_NAME,
 9> SALARY
10>FROM
11> employees
12>WHERE
13> JOB_ID in ('ST_CLERK','ST_MAN')
14>ORDER BY
15> Job_id;
```

```
Report Salary                                                          Page
   :  1
                                  Salary in $

Author: UWS

JOB_ID      EMPLOYEE_ID LAST_NAME                     SALARY
========== =========== ======================== ==========
ST_CLERK            125 Nayer                          3200
ST_CLERK            126 Mikkilineni                    2700
ST_CLERK            144 Vargas                         2500
ST_CLERK            143 Matos                          2600
ST_CLERK            142 Davies                         3100
ST_CLERK            141 Rajs                           3500
ST_CLERK            127 Landry                         2400
ST_CLERK            128 Markle                         2200
ST_CLERK            129 Bissot                         3300
ST_CLERK            130 Atkinson                       2800
ST_CLERK            131 Marlow                         2500
ST_CLERK            132 Olson                          2100
ST_CLERK            133 Mallin                         3300
ST_CLERK            134 Rogers                         2900
ST_CLERK            135 Gee                            2400
ST_CLERK            136 Philtanker                     2200
ST_CLERK            137 Ladwig                         3600
ST_CLERK            138 Stiles                         3200
ST_CLERK            139 Seo                            2700
ST_CLERK            140 Patel                          2500
ST_MAN              121 Fripp                          8200
ST_MAN              120 Weiss                          8000
ST_MAN              123 Vollman                        6500
ST_MAN              122 Kaufling                       7900
ST_MAN              124 Mourgos                        5800

Version: 18.12.19
sql>TTITLE OFF;
sql>BTITLE OFF;
```

Listing 1.163: Beispiel Report

1.21.3 Berechnungen mit Compute

Berechnungen wie zum Beispiel einer Summe oder eines Maximal Wertes können mit `compute` durchgeführt werden.
Syntax:
`COMP[ute] [function [LABEL labelname] OF |expr|column|alias ON expr|column|alias|REPORT|FORM]`

Function	Beschreibung
sum	Bildet eine Summe
avg	Bildet einen Durchschnitts Wert
count	Anzahl der Werte
max[imum]	Der maximale Wert
min[imum]	Der minimalste Wert
num[ber]	Anzahl der Einträge
std	Standard Abweichnung
var[iance]	Abweichung

Tabelle 1.11: Liste Function

Die Ausgabe des Count Befehls erfolgt immer unter der Spalte der Bedingung. In dem nachfolgenden Beispiel unter JOB_ID. Würde die JOB_ID nach dem Feld LAST_NAME kommen, würde die Summe dort ausgegeben.

```
sql>BREAK on job_id SKIP 1
 2>COMPUTE sum LABEL 'Total:' of salary on job_id
 3>SELECT
 4> JOB_ID,
 5> EMPLOYEE_ID,
 6> LAST_NAME,
 7> SALARY
 8>FROM
 9> employees
10>WHERE
11> JOB_ID in ('ST_CLERK','ST_MAN')
12>ORDER BY
13> Job_id;

JOB_ID      EMPLOYEE_ID LAST_NAME                     SALARY
---------- ----------- ------------------------- ----------
ST_CLERK            125 Nayer                           3200
                    126 Mikkilineni                     2700
                    144 Vargas                          2500
                    143 Matos                           2600
                    142 Davies                          3100
                    141 Rajs                            3500
                    127 Landry                          2400
                    128 Markle                          2200
                    129 Bissot                          3300
                    130 Atkinson                        2800
                    131 Marlow                          2500
                    132 Olson                           2100
                    133 Mallin                          3300
                    134 Rogers                          2900
                    135 Gee                             2400
                    136 Philtanker                      2200
                    137 Ladwig                          3600
                    138 Stiles                          3200
                    139 Seo                             2700
                    140 Patel                           2500
```

```
**********                                        ----------
Total:                                                55700

ST_MAN              121 Fripp                          8200
                    120 Weiss                          8000
                    123 Vollman                        6500
                    122 Kaufling                       7900
                    124 Mourgos                        5800
**********                                        ----------
Total:                                                36400
```

Listing 1.164: Beispiel Compute

1.22 Logging

Möchte man Informationen über den Ablauf eines Programms / Procedure protokollieren, so kann man es folgendermaßen machen.

```
CREATE OR REPLACE PROCEDURE "SP_LOG" (
    P_MESSAGE_TEXT VARCHAR2
) IS
  pragma autonomous_transaction;
BEGIN

    DBMS_OUTPUT.PUT_LINE(P_MESSAGE_TEXT);

    INSERT INTO PROCESSING_LOG (
        MESSAGE_DATE,
        MESSAGE_TEXT
    ) VALUES (
        SYSDATE,
        P_MESSAGE_TEXT
    );
    COMMIT;

END;
/
-- PROCEDURE
BEGIN
  SP_LOG('Starting task 1 of 2');

  ... code for task 1 ...

  SP_LOG('Starting task 2 of 2');

  ... code for task 2 ...

  SP_LOG('Ending Tasks');

  ... determine success or failure of process and commit or rollback ...

 ROLLBACK;
END;
/
```

Listing 1.165: Beispiel Logging

```
create or replace procedure my_log (action in varchar2, message in varchar2
   )
is

begin
  Insert into my_log_table (ACTION, MESSAGE, EVENT_DATE)
  values (action, message, sysdate);
  commit;
end;
/

CREATE OR REPLACE PROCEDURE "CUSTOMER_INCREMENTAL" ()
IS
   err_num NUMBER;
   err_msg VARCHAR2(4000);
BEGIN
    my_log ('Start','My message');
  INSERT INTO NDB_AML_CUSTOMER
```

```
(ID, TITLE,...)
  SELECT ID, TITLE,...
FROM NDB_CUSTOMER_NEW
WHERE DATE_TIME > (SELECT RUN_DATE FROM CHECK_POINT WHERE TABLE_NAME = '
    NDB_CUSTOMER_NEW');

  UPDATE CHECK_POINT SET RUN_DATE = SYSDATE WHERE TABLE_NAME = '
     NDB_CUSTOMER_NEW';

  COMMIT;
  my_log ('End','My message');
  EXCEPTION
  WHEN OTHERS THEN
err_num := SQLCODE;
err_msg := SQLERRM;
my_log ('Error' , errnum ||' - ' || err_msg);
 END;
/
```

Listing 1.166: Weiteres Beispiel