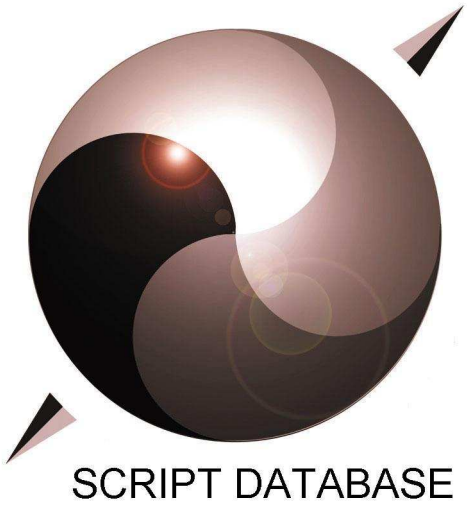

http://www.therealgang.de/

Titel :	Oracle
Author :	S.Schmidbauer, S.Mayer
Kategorie :	DATENBANKTECHNIK-ALLGEMEIN

Oracle Anwenderscript

Created by Susanne Mayer und Sandra Schmidbauer

Inhaltsverezeichnis

1. Erste Schritte

a. Anmeldung an Datenbank

Benutzername: System(DBA), Scott(DBA), Public
Kennwort: Manager, Tiger, kein Passwort

b. Datenbank füllen bzw. Testdaten importieren

start crdemodb;
start cralle;
start insalle;

c. Neuen Benutzer anlegen

grant dba to *benutzername* identified by *passwort*;

d. Benutzer DBA-Rechte erteilen

Dies funktioniert nur wenn man die nötigen Rechte dazu hat, sprich Administratorrechte
grant dba to *benutzername*;

e. Benutzer löschen

Dies ist erst möglich, wenn die Childs-Tabellen bereits gelöscht sind
drop user *benutzername*;
Wenn der Benutzer mit allen Beziehungen und Tabellen gelöscht werden soll
drop user *benutzername* cascade;

f. Benutzer wechseln

connect *benutzername*
Anschließend wird nach dem Passwort gefragt

g. Objekte der User anzeigen

select * from user_users;
select * from user_tables;
select table_name from user_tables;
select view.name from user_views;
select username from dba_users;

h. Tabellenaufbau bzw. –definition anzeigen

desc *tabellenname*;

i. Speichern

Hierbei wird alles bisher Erstelltes oder Geändertes abgespeichert
commit;

j. Rückgängig

rollback;

2. Tabellen

a. Allgemeines

Kommando zum Erstellen einer Tabelle:
CREATE TABLE name (Spaltenname Datentyp(Constraints));

Nur der Datenbankadministrator kann Tabellen für einen Benutzer anlegen!

b. Primary Key und Foreign Key; Index

Erstellen einer Tabelle mit Primary und Foreign Key an einem konkreten Bsp:

```
CREATE TABLE auftrag(auftragsnr CHAR(5),
                     CONSTRAINT pk_auftragsnr
                     PRIMARY KEY(auftragsnr),
                     aufdatum DATE,
                     kundennr  NUMBER(5)
                          CONSTRAINT fk_kundennr
                     FOREIGN KEY (kundennr)
                     REFERENCES kunden(kundennr));
CREATE UNIQUE INDEX pk_auftragsnr_index ON auftrag(auftragsnr);
CREATE INDEX fk_kundennr_index ON auftrag(kundennr);
```

c. Tabelle bearbeiten

Umbenennen
RENAME altername TO neuername

Spalte hinzufügen
ALTER TABLE name ADD(Spaltenname Datentyp);

Datentyp ändern
ALTER TABLE name MODIFY(spaltenspezifikation)

Spalte löschen
In Oracle fehlt der Befehl um eine Spalte zu löschen. Daher muss zunächst eine neue Tabelle mit den benötigten Feldern erstellt werden, die alte Tabelle gelöscht werden und die neue Tabelle dann noch umbenannt werden.
CREATE TABLE name AS SELECT feldername FROM alteTabelle;

Tabelle löschen
DROP TABLE name;

3. Datenmanipulation

a. *Insert – Datensätze einfügen*

Mit dem Insert-Befehl wird eine Zeile oder mehrere Zeilen in eine bestehende Tabelle eingefügt.

Grundform:

insert into *tabellennamen* [(*spaltenname1*, …)]
values (‚*wert1, wert2*, ….);

Beispiel:

insert into personal (persnr, nachname, vorname, abtnr, geb_datum, prov_satz)
values (10, ‚Maier', ‚Armin', 21, ‚11-Jan-34', 0.03);

Zugriff auf Informationen anderer Tabellen:

insert into *tabellenname1* [(*spaltenname1*, …)]
select *spaltenname1, Wert bzw. Konstante*
from *tabellenname2*, …
where …;

Beispiel:

insert into mitarbeiter (persnr, name, projekt_nr, abtnr)
select 501, ‚Mohr', projekt_nr, 20
from projekt
where projektname = ‚Wirkstoff ABC';

b. *Select – Sätze bzw. Werte suchen*

Dieser Befehl wird benutzt, um sich gesuchte Daten anzeigen zu lassen.

Grundform:

Select *Angabe der gewünschten Spalten*
From *Angabe der Tabelle(n)*
Where *Bedingung(en)*
Group by *Gruppenbildung bei gleichen Werden in der angegebenen Spalte*
Having *Selektion spezifischer Gruppen*
Order by *Sortierfolge der Sätze der Ergebnistabelle*

Einfache Abfrage:

Select * from kunde;
Select auftragsnr, aufdatum, aufstatus, kundennr from auftrag;

Unterdückung von Mehrfach-Anzeige:

Select **distinct** kundennr from auftrag;
Distinct kann nur erfolgreich ausgeführt werden, wenn nur ein Spaltenname ausgegeben werden soll.
Select kundennr from auftrag group by kundennr;

Einfache Berechnungen und Spaltenüberschriften:

Select auftragsnr, artikelnr, menge, preis, (menge*preis)Artikelumsatz from auftragspos
Die Überschrift Artikelumsatz wird in Großbuchstaben angezeigt
Select persnr, nachname, (gehalt*1.05)“Gehaltneu“ from personal;
Die Überschrift Gehaltneu wird so angezeigt wie in den Anführungszeichen
Select persnr, nachname, 'Jahresgehalt=', gehalt*12, 'Projekt=', 501 from personal;

Jahresgehalt und Projekt sind neue Spalten und enthalten diese Bezeichnungen

Verknüpfung von Zeichenketten:

Mit dem Operator || können mehrere Zeichenketten miteinander verknüpft werden
select (artikelnr || artikelbez)artikelkennung from artikel;

Sortierung der Ausgabe:

```
Select persnr, nachname
from personal
order by nachname;
    oder
    order by nachname asc, nachname desc;
```

desc bedeutet absteigend
asc oder nichts bedeutet aufsteigend

Auswahl von Zeilen:

=	gleich
< >	ungleich
>	größer
>=	größer gleich
<	kleiner
<=	kleiner gleich
NOT	Verneinung einer Bedingung
AND	alle Bedingungen müssen erfüllt sein, damit selektiert wird
OR	mindestens eine Bedingung muss erfüllt sein
Datum + Zahl	Fügt eine Anzahl von n Tagen zu einem Datum hinzu
Datum – Zahl	Subtrahiert eine Anzahl von n Tagen vom Datum
Datum – Datum	Subtrahiert ein Datum von einem anderen; Ergebnis ist Anzahl
Between … And …	zwischen … und …
Not between … and.	Nicht zwischen … und…
IN (Liste)	in der Liste enthalten
NOT IN	nicht in der Liste enthalten
LIKE	Übereinstimmung mit Zeichenkettenfragmenten
NOT LIKE	keine Übereinstimmung „
IS NULL	Feldinhalt leer
IS NOT NULL	Feldinhalt nicht leer

Beispiele:

```
Select * from personal where persnr = 18;
Select * from auftrag where kundennr IN (24005, 24006, 24010) AND liefnr
        NOT IN (1011, ..);
Select persnr, nachname from personal where nachname LIKE ‚M__er';
Select auftragsnr, aufdatum, sysdate, sysdate – aufdatum from auftrag
where sysdate – aufdatum < 90;
Select sysdate from dual;
```

Arithmetische Funktionen:

Select ABS (-15.37) from dual;
ABS macht aus negativen Werten positive 15.37

Select CEIL (10.5789) from dual; 11
CEIL rundet positive Werte
Select FLOOR (-10.4789) from dual; -11
FLOOR rundet positive Werte ab und negative Werte auf
Select MOD (7,5) from dual; 2
MOD ist der ganzzahlige Rest einer Division
Select POWER (5,2) from dual; 25
POWER ist das Quadrat einer Zahl
Select ROUND (10.7389, 2) from dual; 10.47
ROUND rundet so wie man es angibt
Select SIGN (-3.14), SIGN (2.5), SIGN (0) from dual; -1, 1, 0
SIGN ist die Kennzeichnung von negativen, positiven und null-Werten
Select SQRT (144) from dual; 12
SQRT berechnet die Wurzel der Werte
Select TRUNC (10.5433), TRUNC (10.4536, 2) from dual; 10, 10.45
TRUNC ist die Darstellung eines Wertes. Schneidet Nachkommastellen ab.

Zeichenkettenfunktionen:

Select ASCII (‚A') from dual; 65
ASCII liefert den dezimalen Wert eines Buchstabens
Select CHR (67) from dual; C
CHR liefert das ASCII Zeichen
Select INITCAP (kurzbez), kurzbez from kunde where kurzbez LIKE ‚M%';
INITCAP ist die Groß- und Kleinschreibung einer Zeichenkette
Select UPPER (kurzbez), kurzbez from kunde where kurzbez LIKE ‚M%';
UPPER ist die Großschreibung einer Zeichenkette
Select LOWER (kurzbez), kurzbez from kunde where kurzbez LIKE ‚M%';
LOWER ist die Kleinschreibung einer Zeichenkette
INSTR ist die Ausgabe der Position eines Wertes
LENGTH gibt die Länge einer Zeichenkette an
Select nachname, LPAD(einst_datum, 20, ‚.') from personal;
LPAD ist das Auffüllen von links mit angegebenem Wert
RPAD ist das Auffüllen von rechts mit angegebenem Wert
Select LTRIM (‚HALLO__', ‚_') from dual;
LTRIM schneidet von links die angegebenen Werte ab
RTRIM schneidet von rechts die angegebenen Werte ab
SOUNDEX bedeutet, dass alles ausgegeben werden soll, was ähnlich klingt
Select auftragsnr, aufdatum, SUBSTR (aufdatum, 1, 2)"Tag" from auftrag;
SUBSTR bedeutet, dass Teile einer Zeichenkette ausgegeben werden sollen; der Wert 1 bedeutet, dass die erste Stelle der Zeichenkette aus aufdatum angezeigt werden soll; der Wert 2 bedeutet, dass es auf zwei Stellen abgeschnitten werden soll.
Select artikelnr, d_ek_preis, TRANSLATE(d_ek_preis, ‚.', ‚,') from artikel;
TRANSLATE bedeutet, dass der angegebene Wert der Zeichenkette . durch , ersetzt werden soll.

Konvertierungsfunktionen:

Select persnr, nachname, TO_CHAR(gehalt, ‚9999.99') || 'EUR' "Gehalt" from personal;
TO_CHAR konvertiert eine Zahl in eine Zeichenkette. Hier steht die 9 für ein Kann-Feld und die 0 für eine Muss-Feld.

Select nachname, TO_CHAR(geb_datum, ‚DD.MM.YY')Datum,
TO_CHAR(geb_datum, ‚HH24:Mi)Uhrzeit from personal;
Select nachname, geb_datum from personal where geb_datum <
TO_DATE(‚45', ‚YY');

Datumsfunktionen:

ADD_MONTHS(d,n)	*Addition von n Monaten zum Datum d*
LAST_DAY(d)	*Datum des letzten Tages im Monat*
MONTHS_BETWEEN(d1,d2)	*Anzahl der Monate zwischen d1 und d2*
NEW_TIME(d, c1, c2)	*Liefert das Datum und die Zeit der Zeitzone c2 bezogen auf die Zeitzone c1 des Datums d*
NEXT_DAY(d,c)	*Liefert das Datum des nächstfolgenden Wochentages c, bezogen auf das Datum d. c muss eine zulässige Tagesbezeichnung sein*.

Sonstige Funktionen:

NVL(a, a1)	*Wenn a = 0 dann wird a1 ausgegeben, wenn a=1 dann wird a ausgegeben*
DECODE	select persnr, nachname, abtnr, DECODE if abtnr=21 then ‚Verk.' Else if abtnr=22 then ‚Eink' else if abtnr=24 then'Entw.' Else ‚SONST' end-if end-if end-if from personal where abtnr IN(21, 22, 24, 25);

Gruppen-Funktionen:

COUNT	*Anzahl der Zeilen einer Tabelle*
MAX	*Maximaler Wert des Ausdrucks in der Tabelle*
MIN	*Minimaler Wert des Ausdrucks in der Tabelle*
AVG	*Mittelwert der Werte des Ausdrucks der Tabelle*
SUM	*Summe der Werte des Ausdrucks in der Tabelle*
VARIANCE	*Varianz der Werte des Ausdrucks in der Tabelle, ohne Null-Werte*

Select mit GROUP BY:

Select *Angabe der gewünschten Spalten*
From *Angabe der Tabelle(n)*
Where *Bedingung(en)*
Group by *Gruppenbildung bei gleichen Werden in der angegebenen Spalte*
Having *Selektion spezifischer Gruppen*
Order by *Sortierfolge der Sätze der Ergebnistabelle*

Select produktgruppe_kz, sum(d_ek_preis*bestand), max(bestand),
min(bestand) from artikel group by produktgruppe_kz;

Logische Abarbeitungsfolge eines SQL-Befehls:

SELECT SUBSTR(TO_CHAR(einst_datum), 8, 2), AVG(prov_satz*gehalt)
FROM personal
WHERE abtnr <> 24
GROUP BY SUBSTR(TO_CHAR(einst_datum), 8, 2)
HAVING BY AVG(prov_satz*gehalt);

1. Schritt	*Mit FROM wird zunächst die Tabelle ausgewählt*
2. Schritt	*Diese Tabelle wird um die Zeilen vermindert, die die inder WHERE-Klausel aufgeführten Bedingung nicht erfüllen.*
3. Schritt	*Dur die GROUP BY Klausel werden die Zeilen mit dem gemeinsamen Merkmal zusammengefasst, wobei die HAVING-Klausel überprüft wird.*
4. Schritt	*Beimm Zusammenfassen wird geprüft, ob die Gruppe auch die in der HAVING-Klausel angengebenen Bedingungen erfüllt*
5. Schritt	*Für die verbleibenden Zeilen wird in der Ergebnistabelle auf die in der Select-Klausel angegebenen Spalten projiziert.*
6. Schritt	*Diese projizierte Ergebnistabelle wird entsprechend den Angaben der ORDER BY-Klausel sortiert zur Anzeige gebracht.*

4. Hierachische Strukturen

a. Bearbeitung

SQL-Kommando zur Bearbeitung von hierarischer Strukturen
SELECT * FROM tabellenname
CONNECT BY elternspalte = PRIOR kindspalte
START WITH spalte = bedingung;

PRIOR legt die Sicht auf den Baum fest; von oben nach unten PRIOR Unterteil bzw. von unten nach oben PRIOR OBERTEIL

ACHTUNG: Die Bearbeitung von hierarchischen Strukturen in Verbindung mit dem Befehl JOIN ist nicht erlaubt!

b. Union, Intersect, Minus in SELECT-Statements

Minus:
Gib alle Teile an die zum Damenrad, aber nicht zum Herrenrad gehören
SELECT unterteil FROM est
CONNECT BY PRIOR unterteil = oberteil
START WITH oberteil=1000
MINUS
SELECT unterteil FROM est
CONNECT BY PRIOR unterteil = oberteil
START WITH oberteil = 1001;

Intersect:
Gib alle Teile an die sowohl zum Damenrad als auch zum Herrenrad gehören
SELECT unterteil FROM est
CONNECT BY PRIOR unterteil = oberteil
START WITH oberteil=1000

INTERSECT
SELECT unterteil FROM est
CONNECT BY PRIOR unterteil = oberteil
START WITH oberteil = 1001;

Union:
Gib alle Teile aus, die zum Damenrad oder zum Herrenrad gehören
SELECT unterteil FROM est
CONNECT BY PRIOR unterteil = oberteil
START WITH oberteil=1000
UNION
SELECT unterteil FROM est
CONNECT BY PRIOR unterteil = oberteil
START WITH oberteil = 1001;

c. *Update Befehl*

Allgemein:
UPDATE tabellenname
SET (spaltenname = Wert, … , spaltenname = Wert)
WHERE bedingung;
Falls keine Where-Klausel vorhanden ist, ändern sich alle Werte in der betreffenden Spalte

d. *Delete Befehl*

Allgemein:
DELETE FROM tabellenname
WHERE bedingung;
Delete löscht eine oder mehrere Zeilen aus einer bestehenden Tabelle. Die Tabelle selbst wird nicht gelöscht und bleibt im DATA DICTIONARY definiert.

5. Datenverbund

a. *Allgemein*

Der Join ist eine Verbindung von Zeilen meherer Tabellen in einem SELECT-Befehl. Dabei wird jede Zeile einer Tabelle mit jeder Zeile einer anderen Tabellen verknüpft. Die Verbindung der Tabellen wird in der WHERE-Klausel definiert.

b. *Equi-Join*

Der Equi-Join verbindet sachlich zusammengehörende Feldinhalte.
z.B.
SELECT kunde.kundennr, auftrag.auftragsnr
FROM kunde, auftrag
WHERE kunde.kundennr = auftrag.kundennr
GROUP BY kunde.kundennr;

c. *Outer-Join*

Liegt bei einem Join in einer Tabelle kein Vergleichswert vor, existiert dort auch keine Zeile. Z.B. existieren Kunden in der Tabelle KUNDE, die keinen Auftrag erteilt haben. Der Outer-Join behandelt auch die Tabelle, in denen kein Vergleichswert existiert, indem dort virtuell eine Zeile mit NULL-Wert eingefügt und dann angezeigt wird.
z.B:

```
SELECT kunde.kundennr, auftragsnr, name
FROM auftrag, kunde
WHERE kunde.kundennr = auftrag.kundennr
AND
auftrag.kundennr IS NULL;
```

6. Unterabfragen

a. *Allgemein*

Es gibt Abfragen, deren Bedingungen vom augenblicklichen Inhalt der Tabelle abhängen, d.h. die Bedingungen müssen zum Abfragezeitpunkt mit einem weiteren SELECT-Kommando erst ermittelt werden. Die Gesamtabfrage gliedert sich in eine Hautpabfrage und eine oder mehrere Unterabfragen.
Eine Unterabfrage darf ein GROUP BY, aber kein ORDER BY enthalten.
z.B.

```
SELECT persnr, nachname, gehalt
FROM personal
WHERE gehalt >(SELECT AVG(gehalt) FROM personal);
```

b. *Unterabfragen die ein Ergebnis liefern*

z.B.:Welche Abteilungen haben ein mittleres Gehalt, das größer ist als das mittlere Gehalt der gesamten Firma.

```
SELECT abtnr, AVG(gehalt)
FROM personal
GROUP BY abtnr
HAVING AVG(gehalt)>(SELECT AVG(gehalt) FROM personal);
```

c. *Unterabfragen die mehrere Ergebnisse liefern*

z.B.: Welche Mitarbeiter haben einen Provisionssatz, der auch in der Abteilung mit der Nr. 21 vorkommt?

```
SELECT abtnr, persnr, prov_satz
FROM personal
WHERE prov_satz IN(SELECT prov_satz FROM personal WHERE abtnr=21);
```

7. Views

Views sind virtuelle Tabellen bzw. Fenster in einer Datenbank, in denen man Inforamtionen aus einer oder mehreren Tabellen sehen und bearbeiten kann. Die View-Tabelle existiert nicht physikalisch, sondern nur für den Benutzer. Die Zeilen

dieser virtuellen Tabelle sind abgeleitet aus den zugrunde liegenden Basistabellen. Die Ableitung geschieht mit Hilfe des Select-Kommandos in der View-Definition, welches des RDBMS jedes mal ausführt, wenn eine View benutzt wird.

a. Grundform:

CREATE
OR REPLACE *ersetzt die View, wenn sie bereits vorhanden ist*
VIEW benutzer.viewname()
As Select befehl;
Select view.name from user_views; select view.name, text from user_views;

b. Einfache Views mit einer Tabelle:

Views mit realen Spalten:

Create VIEW kuview1 as select kundennr, name1, plz, ort from kunde;
Create or replace VIEW persview1 as select persnr, nachname, gehalt from personal where gehalt between 3000 and 5000;

Einfügen und Verändern von Datensätzen:

Create VIEW persview2 as select persnr, nachname, gehalt, abtnr from prestest where gehalt between 3000 and 5000;
Update persview2 set gehalt = 2000 where persnr = 13;

With Check Option:

SQL macht es möglich, Änderungen an einem View auszuschließen, die einen Widerspruch zur Selektion in der View-Definition darstellen. Denn aufgrund der Veränderungen des Gehaltes des Mitarbeiters mit der persnr = 13 auf 2000, wird dieser in der View nicht mehr berücksichtigt.
Um solche Updates zu verhindern, verwendet man in der Create-Anweisung den Zusatz WITH CHECK OPTION!
Update perstest set gehalt = 3000 where persnr = 13; oder ROLLBACK
Dann View neu erstellen
Create VIEW persview2 as select persnr, nachname, gehalt, abtnr from perstest where gehalt between 3000 and 5000 WITH CHECK OPTION;
Anschließend die View updaten. Welche Meldung erscheint?
Update persview2 set gehalt = 2000 where persnr = 13;

VIEWS mit virtuellen Spalten:

Create VIEW artumsview1 as select artikelnr, auftragsnr, menge, preis, (menge*preis)Umsatz from auftragspos;
Insert into artumsview1 values (4711, ‚A9999', 5, 10, 50);
Welche Meldung erscheint?

VIEWS mit Gruppenfunktionen

Create VIEW abtgehview1 as select abtnr, sum(gehalt)gehaltssumme from personal group by abtnr;

c. VIEW mit Join

Mit Hilfe eines Join, kann man die Informationen wieder zusammenführen, die im Normalisierungsprozess auf unterschiedliche Tabellen aufgeteilt wurden.

View-Werte aus mehreren Tabellen, die aufgrund der Primär- und Fremdschlüsselrelationen zusammengestellt wurden, können nicht verändert werden.

```
Create VIEW aufkuview1
        as select auftrag.auftragsnr, aufdatum, sum(menge*preis)Umsatz
        from auftrag, auftragspos
        where auftrag.auftragspos = auftragspos.auftragsnr
        group by auftrag.auftragsnr, aufdatum;
```

Löschen einer View:

Eine View muss man mit der Create View-Anweisung erzeugen. Zwecks Änderung eines Views muss diese entweder mit dem Befehl DROP VIEW tabellenname gelöscht und anschließend noch mal neu erstellt werden oder man verwendet bei der Definition der View den REPLACE-Zusatz.

```
Create or Replace VIEW aufkuview1 as select auftrag.auftragsnr, aufdatum,
...
```

8. PL/SQL

Wichtig: Um eine Bildschirmausgabe zu ermöglichen: set serveroutput on

a. Erstellen von einfachen anonymen Blöcken

Ein anonymer Block ist sozusagen eine Procedure bei der der Definitionskopf weggelassen wurde und die damit keine Parameterliste besitzt.

Grundform:

```
Declare
Begin
End;
/
```

Beispiel:

```
Declare

max_records constant int:=10;
i int := 1;
Beging
Delete from test1;
For I IN 1 .. max_records
Loop
    If I = 8 then
    Insert into test1 (record_id; sysdatum)
    Values(i+10, sysdate);
    Else
    Insert into test1 values(i, sysdate);
    End if;
End loop;
Commit;
```

```
End;
/
```

Beispiel mit Bildschirmanzeige:

```
Declare temp_gehaltssumme real;
Begin dbms_output.enable;
Select sum(gehalt) into temp_gehaltssumme from personal;
Dbms_output.put_line('Gehaltssumme: ' || temp_gehaltssumme);
End; /
```

b. Erstellen von einfachen Prozeduren

Prozeduren ohne Parameterübergabe:

```
Create or Replace procaufumsatz IS temp_aufumsatz real;
Begin
    Dbms_output.enable;
    Select sum(menge*preis) into temp_aufumsatz from auftragspos;
    Dbms_ouput.put_line(,Auf-Umsatz: , || temp_aufumsatz);
End; /
```

Aufruf der Prozedur nach start mit dem Befehl EXECUTE prozedurname!

Prozeduren mit Parameterübergabe:

Deklaration von Variablen:

Name_var1 mitarbeiter.mit_name%type

➔ Der Variable Name_var1 wird zur Laufzeit der gleiche Datentyp zugewiesen, wie der Spalte mit_name in der Tabelle mitarbeiter.

Var2 mitarbeiter%rowtype

➔ Es wird zur Laufzeit eine Struktur mit Namen Var2 aufgebaut, mit gleichen Namen und Datentypen wie die Tabelle mitarbeiter.

```
Create or Replace procaufumsatz(tmpaufnr char)
IS
temp_aufumsatz real;
Begin
    Dbms_output.enable;
    Select sum(menge*preis) into temp_aufumsatz from auftragspos
    Where auftragsnr = tmpaufnr;
    Dbms_ouput.put_line(,Auf-Umsatz: , || temp_aufumsatz);
End; /
```

Cursortechnik

Ein Cursor dient zur Verwaltung des Zugriffs auf einen Satz von Datenzeilen, der das Ergebnis einer Select-Anweisung ist. Hierbei kann der Satz eine oder mehrere Zellen umfassen. Wie Variablen werden auch Cursor im Deklarationsteil definiert. Es wird ein Name und die Select-Anweisung zugewiesen. Als erstes muss der Cursor geöffnet werden um auf die Daten zugreifen zu können. Das Öffnen bewirkt auch die Durchführung des Select-

Befehls. Die Verarbeitung erfolgt zeilenweise. Durch FETCH wird die Zeile auf die der Zeiger des Cursor verweist in die angegebene Variable übertragen und der Zeiger auf den nächsten Datensatz gestellt.

```
DECLARE
CURSOR artcursor IS
     SELECT artikelnr, bestand FROM artikel WHERE bestand >0;
artsatz artcursor%rowtype;
BEGIN
OPEN artcursor;
LOOP
  FETCH artcursor INTO artsatz;
  EXIT WHEN artcursor%NOTFOUND; /* Beendet wenn kein Datensatz mehr vorhanden ist */
END LOOP;
CLOSE artcursor;
```

Prozeduraufrufe in anonymen Blöcken

Ruft die Prozedur mit dem Namen procmaxgehalt für die Abteilungsnr. 21 auf.

```
DECLARE
BEGIN
  dmbs_output.enable;
  dbms_output.put_line(, Aufruf der Prozedur');
   procmaxgehalt(21);
END;
/
```

Prozeduren mit Übergabeparametern

Jedes Argument einer Prozedur oder Funktion kann optional aus einer der folgenden Möglichkeiten bestehen: in, out, in out (Standard ist: in)

Nachfolgende Prozedur ermittelt für einen beliebigen Auftrag(über Auftragsnr) den Auftragsumsatz. Falls der Auftrag existiert, wird der Umsatz über einen Übergabeparameter an das aufgerufene Programm zurückgeliefert. Falls ein Auftrag nicht existiert ist dies anhand des Wertes des Parameters tmpfehlerstatus ersichtlich.

```
CREATE OR REPLACE Procedure procaufumsatz1(tmpaufnr char, tmpaufumsatz out real, tmpstatus  out number)
IS
   temp_aufumsatz number(9,2);
   temp_status number(9,2);
BEGIN
   Select sum(menge*preis) into temp_aufumsatz FROM auftragspos
   WHERE auftragsnr = tmpaufnr;
    IF temp_aufumsatz IS Null Then
        temp_status := 1;
         tmpstatus := temp_status;
    ELSE
         temp_status := 0;
         tmpstatus := temp_status;
         tmpaufumsatz  := temp_aufumsatz;
    END IF;
```

```
END;
/
```

➔ dem “out”Parameter kann direkt kein Wert zugewiesen werden. Zunächst muss einer Variable der ermittelte Wert zugewiesen werden und diese dann wieder an den Parameter übergeben werden.

➔ “out”Parameter können nur Werte entgegen nehmen. In diesem Beispiel wird dem Parameter tmpaufumsatz also der mittels des Select-Befehls ermittelte Auftragsumsatz zugewiesen. Weiterhin bedeutete „out“, dass der Parameter dem aufrufenden Programm etwas zurückübergibt(der Wert der in tmpaufumsatz steht.)

Die Prozedur ist im Moment noch nicht ausführbar, da das aufrufende Programm noch fehlt. Dieses wird einfach in einen anonymen Block gepackt.

```
DECLARE
  temp_aufumsatz number(9,2);
  temp_fehlerstatus number;
BEGIN
  dbms_output.enable;
  procaufumsatz1(,A1003', temp_aufumsatz, temp_fehlerstatus);
  dbms_output.put_line(, Der Aufumsatz beträgt: , || temp_aufumsatz);
  dbms_output.put_line(, Der Fehlerstatus ist: , || temp_fehlerstatus);
END;
/
```

c. Erstellen von Funktionen

Funktionen sind gespeicherte Prozeduren, die einen Wert liefern. Alle Anweisungen usw. gelten daher auch für die Erstellung von Funktionen.

```
CREATE OR REPLACE Function euro(tmpbetrag real)
RETURN Real
IS
     tmpeurosatz CONSTANT Real:=1.958538;
     tmpeurobetrag number(9,2);
BEGIN
      tmpeurobetrag = tmpbetrag / tmpeurosatz;
       return tmpeurobetrag;
END;
/
```

d. Function ruft Procedure

Eine Function ruft zur Ermittlung des Auftragsumsatzes eine Prozedur auf, welche prüft ob der Auftrag überhaupt vorhanden ist.

```
CREATE OR REPLACE Function fncaufumsatz2 (tmpaufnr char)
Return Real;
IS
temp_aufumsatz real;
temp_vorhanden boolean;
BEGIN
        dmbs_output.enable;
        procaufsuche(tmpaufnr, temp_vorhanden)
        IF temp_vorhanden THEN
```

```
SELECT sum(menge*preis) INTO temp_aufumsatz FROM
auftragspos WHERE auftragsnr = tmpaufnr;
Return temp_aufumsatz;
ELSE
Return 0;
END IF;
END;
/
```

```
CREATE OR REPLACE Procedure procaufsuche(tmpaufnr char, tmpvorhanden
OUT boolean)
temp_aufnr auftrag.auftragsnr%type;
IS
BEGIN
SELECT auftragsnr INTO temp_aufnr FROM auftrag WHERE auftragsnr =
tmpaufnr;
tmpvorhanden := True;

Exception WHEN NO DATA FOUND THEN
tmpvorhanden := FALSE;
END;
/
```

e. Procedure benutzt Function

Die Prozedur bestimmt Auftragsumsatz eines existierenden Auftrags. Zur Überprüfung ob der Auftrag existiert wird die Funktion in der Prozedur aufgerufen.

```
CREATE OR REPLACE Function fncaufvorhanden(tmpaufnr char)
RETURN boolean
temp_aufnr auftrag.auftragsnr%type;
IS
BEGIN
SELECT auftragsnr INTO temp_aufnr FROM auftrag WHERE
auftragsnr = tmpaufnr;
RETURN True;
Exception WHEN NO DATA FOUND THEN
RETURN False;
END;
/

CREATE OR REPLACE Procedure procfncauf1(tmpaufnr char)
IS
temp_aufumsatz number(9,2);
BEGIN
dbms_output.enable;
IF fncaufvorhanden(tmpaufnr) THEN
SELECT Sum(menge*preis) INTO temp_aufumsatz FROM auftragspos
WHERE auftragsnr = tmpaufnr;
dbms_output.put_line('Der Auftragsumsatz beträgt: ' || temp_aufumsatz);
ELSE
dbms_output.put_line(,Es existiert kein Auftrag');
END IF;
END;
```

/

f. Erstellen von Triggern

Trigger sind Programme, die für einzelne Tabellen definiert werden könne und automatisch bei Datenmanipulationen gestartet werden.
Es wird unterschieden ob ein Trigger pro Befehl einmal ausgelöst werden soll(Befehls-Typ), unabhängig davon, wie viel Datensätze durch das Ereignis geändert werden oder, ob ein Trigger pro geändertem Datensatz(Datensatz-Typ) ausgelöst werden soll. Durch die Angabe der ‚for each row'-Klausel wird ein Datensatz-Trigger definiert.
Bei zeilenorientierten Triggern ist es möglich, auf Werte vor und nach einer Änderung mit :OLD.spaltenname und :NEW.spaltenname zuzugreifen. Zu Beginn einer Update-, Insert- oder Delete-Aktion wird die jeweilige Tablelle gesperrt(keine schreibenden und lesenden Zugriffe auf die Tabelle).

Protokoll-Trigger

```
CREATE OR REPLACE Trigger triggmitarbeiter
BEFORE DELETE OR INSERT OR UPDATE ON temp_mitarbeiter
FOR EACH ROW
DECLARE
temp_klausel varchar2(20);
BEGIN
        IF DELETING THEN
                temp_klausel:='DELETE';
        END IF;
        IF UPDATING THEN
                temp_klausel:='UPDATE';
        END IF;
        IF INSERTING THEN
                temp_klausel:='INSERT';
        END IF;
        INSERT INTO protokoll values(USER, temp_klausel, SYSDATE);
END;
/
```

➔ Die Tabelle protokoll muss vorher noch angelegt werden =)

Update-Trigger – Auslösen eines Bestellsatzes –Mindestbestand >Bestand

```
CREATE OR REPLACE Trigger bestandstrigger1
AFTER UPDATE OF bestand ON artikel
FOR EACH ROW
BEGIN
        IF UPDATING AND :old.artikelnr = :new.artikelnr
        AND :new.bestand < :new.mindest_bestand THEN
        INSERT INTO temp_bestell1 values(:old.artikelnr,
        :new.bestand, :new.mindes_bestand);
        END IF ;
END;
/
```

Delete-Trigger – Bestand >0 – kein Löschen

```
CREATE OR REPLACE TRIGGER triggdelartikel
BEFOR DELETE ON artikel
FOR EACH ROW
```

```
BEGIN
        IF DELETING AND :old.bestand > 0 THEN
        RAISE_APPLICATION_ERROR(-20005, ‘Bestand größer null’);
END IF;
END;
/
```

Delete-Trigger – Bestand>0 – Trigger ruft Function/Procedure auf

Wenn der Bestand allerdings = 0 soll die Function aufgerufen werden die wiederum prüft, ob an den einzelnen Lagerorten(Tabelle lager1) keine Bestände des Artikels mehr vorhanden sind. Falls auch dort keine Bestände mehr sind wird die Procedur aufgerufen, die wiederum die Bestände an den einzelnen Lagerorten löscht und der Trigger wird erfolgreich abgeschlossen(ohne Raise_Application_Error).

Procedur:

```
CREATE OR REPLACE Procedure procdelartikel(tmpartnr char)
IS
BEGIN
Delete FROM Lager 1 WHERE artikelnr = tmpartnr;
END;
/
```

Function:

```
CREATE OR REPLACE Function fncsumbestand(tmpartikelnr number)
RETURN number
temp_summenge number;
IS
BEGIN
SELECT sum(bestand) INTO temp_summenge FROM lager1 WHERE
artikelnr = tmpartikelnr;
IF temp_summenge = 0 THEN
RETURN 0;
ELSE
RETURN temp_summenge;
END IF;
END;
/
```

Trigger:

```
CREATE OR REPLACE TRIGGER triggdelartikel1
BEFOR DELETE ON artikel
FOR EACH ROW
DECLARE
temp_bestand number;
BEGIN
        IF :old.bestand = 0 THEN
                temp_bestand := fncsumbestand(:old.artikelnr)
                IF temp_bestand=0 THEN
                procdelartikel(:old.artikelnr);
                ELSE
                UPDATE artikel SET Bestand  = temp_bestand
                WHERE artikelnr = :old.artikelnr
                RAISE_APPLICATION_ERROR(-20010, ‘Artikel im
                Zentrallager noch vorhanden’);
```

```
            END IF;
        ELSE
        RAISE_APPLICATION_ERROR(-20005, ‘Bestand größer null’);
        END IF;
END;
/
```

Befehlstrigger – Ändern einer Tabelle nur zwischen 08:00 und 16:00 Uhr möglich !

Ein befehlsorientierter Trigger wird genau einmal vor oder nach der Ausführung desjenigen Befehls ausgeführt, der das Ereignis auslöst, und ein zeilenorientierter Trigger je einmal pro Datensatz ausgeführt, der durch den Befehl verändert wurde.

```
CREATE OR REPLACE Trigger triggzugang1
BEFORE DELETE OR INSERT OR UPDATE on artikel
DECLARE
keine_Eingabe Exception;
BEGIN
IF to_char(sysdate,’HH24:MI)
NOT BETWEEN ’08:00’ AND ’16:00’ THEN
raise keine_Eingabe;
END IF;
EXCEPTION
WHEN keine_Eingabe Then Raise_Application_Error(-21000,’Änderung nur
zwischen 08.00 und 16.00 Uhr möglich’);
END;
/
```

Login-/Logoff Trigger

```
CREATE OR REPLACE Trigger trigglogon1
After Logon ON Database
Begin
delete from protokoll;
Insert into protokoll values(user, ‘LOGON’, sysdate, null);
END;
/
```

(Logoff funktioniert genauso)

g. Zugriff auf MS Access