SQL-Befehlsübersicht Oracle-DBMS

Stand: 17.03.2020

Autor: Prof. Dr.-Ing. Heiko Tapken

… Work in Progress …

Inhaltsverzeichnis

1. SELECT - Aufbau eines SQL-Befehls

1.1. Grundsätzlicher Aufbau (SELECT)

```
select [distinct] <ausdruck_1> [alias_1], <ausdruck_2> [alias_2]...
from <quelle_1>, <quelle_2> ...
where <bedingung>
group by <gruppierungs_ausdruck_1>, <gruppierungs_ausdruck_2> ...
having <bedingung>
order by <sortier_ausdruck_1>, <sortier_ausdruck_2> ...
for update of <spalte_1>, <spalte2> ... [nowait];
```

<ausdruck>	beliebiger Ausdruck, i.d.R. mit Spalten gebildet
distinct	doppelte Zeilen werden nur einmal geliefert
alias	Ausdruck erhält Namen in der Ergebnismenge
quelle	Tabelle, View, Inline-View
talias	Aliasnamen für die Quelle in diesem Befehl
<bedingung>	beliebig verschachtelte Ausdrücke möglich
group by	Zusammenfassen von Zeilen
having	Bedingung für zusammengefasste Zeilen
order by	Sortierkriterien
for update	Lesen mit gleichzeitigem Sperren der Ergebnismenge
nowait	Zusatz zu "for update": Ist der Satz bereits gesperrt, wird nicht gewartet, sondern ein Fehler produziert.

1.2. Beispiel

Folgende Formulierung eine Joins ist veraltet und sollte nicht mehr genutzt werden:

```
SELECT PNUMMER, PNAME, COUNT(*)
FROM PROJEKT, ARBEITET_AN
WHERE PNUMMER=PNR
GROUP BY PNUMMER, PNAME
HAVING COUNT(*) > 2;
```

Stattdessen sollte folgende Schreibweise genutzt werden:

```
SELECT PNUMMER, PNAME, COUNT(*)
FROM PROJEKT JOIN ARBEITET_AN ON PNUMMER=PNR
GROUP BY PNUMMER, PNAME
HAVING COUNT(*) > 2;
```

1.3. Anmerkungen

- Es werden Tabellen verarbeitet: Ergebnis: wieder eine Tabelle
- Joins sind teure Operationen
- SQL liefert Multimengen (durch DISTINCT zu verhindern)
- Mengenoperationen (UNION, INTERSECT, MINUS) sortieren aufsteigend und Ergebnis enthält keine doppelten Einträge. Sie sind ebenfalls sehr teuer!

2. SELECT - Joins

Sollen Informationen mehrerer Tabellen in einer Abfrage zusammengefasst werden, so benötigt man einen **Join**.

2.1. Join-Typen

Inner join	Nur korrespondierende Datensätze
Left outer join	Korrespondierende Datensätze und alle Datensätze der linken Tabelle (ergänzt durch NULL bei nichtkorrespondierenden Spalten)
Right outer join	Korrespondierende Datensätze und alle Datensätze der rechten Tabelle (ergänzt durch NULL bei nichtkorrespondierenden Spalten)

2.2. Beispiele:

```
SELECT VNAME, NNAME, ADRESSE
FROM (ANGESTELLTER JOIN ABTEILUNG ON ABT=ABTNUMMER)
WHERE ANAME='Research';
```

Left Outer Join:

```
SELECT A.NNAME AS ANGEST_NAME, S.NNAME AS VORGES_NAME
FROM (ANGESTELLTER A LEFT OUTER JOIN
                ANGESTELLTER S ON
                A.SUPERSSN=S.SSN);
```

3. SELECT - Unterabfragen (Subqueries)

Um komplexere Abfragen zu erstellen, kann es notwendig sein, in der Bedingung Abfragen zu verschachteln.

3.1. Operator "in" bzw. "not in"

Zeige die Sozialversicherungsnummern aller Angestellten, die in einer gleichen Kombination von Projekt und Stunden an einem Projekt arbeiten, an dem auch der Angestellte ‘John Smith‘ mit der SSN 123456789 beschäftigt ist.

```
SELECT DISTINCT ESSN
FROM ARBEITET_AN
WHERE (PNR, STUNDEN) IN (
                  SELECT PNR, STUNDEN
                  FROM ARBEITET_AN
                  WHERE ESSN='123456789' );
```

Dieser Operator sollte aus Performancegründen möglichst vermieden werden.

3.2. Operator "ALL" bzw. "ANY"

- = ANY liefert TRUE, falls v irgendeinem Wert aus V entspricht
- = ANY ist somit äquivalent zu IN.
- = ALL liefert TRUE, falls v allen Werten aus V entspricht.
- Die Schlüsselworte ANY und ALL können ebenfalls mit den Operatoren >, >=, <=, < und <> kombiniert werden.
- v > ALL V liefert bspw. dann TRUE, wenn v größer ist als alle Werte der Menge V.

3.2.1. Beispiele:

Liefere die Namen der Angestellten, deren Gehalt größer ist als das Gehalt aller Angestellten aus Abteilung 5.

```
SELECT NNAME, VNAME
FROM ANGESTELLTER
WHERE GEHALT > ALL (
      SELECT GEHALT
      FROM ANGESTELLTER
```

```
        WHERE ABT=5 );
```

3.3. Operator "exists" bzw. "not exists"

Alle Artikel zu denen es keine Aufträge gibt:

```
Liefere die Namen der Angestellten, die keine Angehörigen haben.

    SELECT NNAME, VNAME
    FROM ANGESTELLTER
    WHERE NOT EXISTS (   SELECT *
                FROM ANGEHOERIGER
                WHERE SSN=ESSN );
```

Wiederum wird für jede Zeile eine Zwischenmenge erzeugt. Diese sind jedoch wesentlich kleiner.

3.4. Operator "="

```
    SELECT NNAME, VNAME
    FROM ANGESTELLTER
    WHERE NNAME = ( SELECT NNAME
                FROM ANGESTELLTER
                WHERE SSN = 1704);
```

3.5. Subqueries mit einer Zeile und einer Spalte

Subqueries, die genau eine Zeile und eine Spalte liefern, können an allen Stellen eingesetzt werden, an denen auch andere Platzhalter (Spalten, Literale) verwendet werden können. Dabei sind Querverbindungen möglich:

3.6. Beispiele:

```
select gehalt ( select avg(gehalt)
                        from angstellter) Durchschnitt
from artikel a;

insert into angestellter (SSN, vname, nname) values (
(select nvl(max(SSN),1) from angestellter) + 1, 'Heiko', 'Tapken'));
```

oder

```
insert into angestellter (SSN, vname, nname) values (
(select nvl(max(SSN),1)+1 from angestellter), 'Heiko', 'Tapken'));
```

4. SELECT – Skalare Funktionen

4.1. Numerische Funktionen

4.1.1. Übersicht

abs(a)	absoluter Wert von a
ceil(a)	kleinste ganze Zahl größer als a
floor(a)	größte ganze Zahl kleiner als a
mod(m,n)	m Modulo n (Rest von m geteilt durch n)
power(m,n)	m hoch n
round(n[,m])	n auf m Stellen gerundet, m per default 0, • m positiv: Stellen vor dem Komma • m negativ: Stellen nach dem Komma
sign(a)	Vorzeichen von a (0, 1 oder -1)
sin(a)	Sinus von a (weitere trigonometrische Funktionen verfügbar)
sqrt(a)	Wurzel aus a
trunc(a[,m])	a auf m Stellen abgeschnitten
exp(n)	liefert e hoch n (e=2,17828...)

ln(n)	natürlicher Algorithmus von zu e
log(m,n)	Logarithmus von n zu Basis m

4.1.2. Beispiele

```
select round (21.76) from dual;
```

Ergebnis: 22

```
select trunc (21.76) from dual;
```

Ergebnis: 21

```
select trunc (21.76, -1) from dual;
```

Ergebnis: 20

4.2. Stringfunktionen

4.2.1. Übersicht

concat(s1, s2)	s1 und s2 konkateniert (entspricht s1||s2)
lower(s)	s in Kleinbuchstaben
upper(s)	s in Großbuchstaben
initcap(s)	erstes Zeichen eines Wortes groß, Rest klein
lpad(s1, n [,s2])	s1 auf n Zeichen mit s2 von links aufgefüllt (Defaultwert für s2 ist ein Blank)
rpad(s1, n [,s2])	s1 auf n Zeichen mit s2 von rechts aufgefüllt (Defaultwert für s2 ist ein Blank)
ltrim(s1[,s2])	alle führenden Zeichen aus s2 in s1 entfernen (Defaultwert für s2 ist ein Blank)
rtrim(s1[,s2])	alle endenden Zeichen aus s2 in s1 entfernen (Defaultwert für s2 ist ein Blank)
trim(s1)	Kombination aus ltrim und rtrim
replace(s1,s2[,s3])	suche s2 in s1 und ersetze ihn durch s3 bzw. NULL
translate(s1,s2,s3)	in s1 werden alle Zeichen aus s2 durch solche aus s3 ersetzt
substr(s,m[,n])	Teilstring von s ab Stelle m, n Zeichen lang (n nicht angegeben => bis Stringende)
instr(s1, s2[,n[,m]])	suche s2 in s1 und zwar ab der n-ten Stelle das m-te Auftreten (Defaultwerte für n und m sind 1), Ergebnis ist die gefundene Position in s1 oder 0
length(s)	die Länge von s

4.2.2. Wichtige Formatzeichen

9	Zahl 0 bis 9 ohne führende Null
0	Zahl 0 bis 9 mit führender Null
S	Vorzeichen + oder -
D	Dezimalpunkt oder Komma
G	Tausenderpunkt oder Komma
FM	Abschneiden von führenden Blanks

4.2.3. Beispiele

Teilstring bestimmen

```
substr('Medien-Informatik', 1, 6)
```

Ergebnis: `'Medien'`

Suchen

```
instr('Medien-Informatik', 'Info')
```

Ergebnis: `8`

Zeichen erzeugen

```
chr(65)
```

Ergebnis: `'A'`

Auf diese Weise können auch nicht druckbare Sonderzeichen in einen String eingefügt werden. Allerdings muss dann der Zeichensatz der Datenbank bekannt sein.

Zeichenkodierung erzeugen

```
ascii('A')
```

Ergebnis: `65`

Teilstring ersetzen

```
replace('SCHADE', 'D', 'LK')
```

Ergebnis: `'SCHALKE'`

Zeichen austauschen

```
translate('ABC67LR5', '0123456789','**********')
```

Ergebnis: `'ABC**LR*'`

```
translate('ABC67LR5', '*0123456789','*')
```

Ergebnis: `'ABCLR'`

Auf diese Weise lassen sich unerwünschte Zeichen elegant aus einer Zeichenkette entfernen.

4.3. Datum-Funktionen

4.3.1. Übersicht

add_months(d,n)	Datum d plus n Monate
last_day(d)	Datum des letzten Tages des Monats, in dem d enthalten ist
months_between(d1, d2)	Anzahl der Monate zwischen d1 und d2
round(d[,fmt])	Datum d gerundet je nach Format (Defaultwert für fmt ist 'dd' (Tag))
sysdate	aktuelles Datum und Uhrzeit
trunc(d[,fmt])	Datum d abgeschnitten je nach Format (Defaultwert für fmt ist 'dd' (Tag))

4.3.2. Formatzeichen

DD	Tag des Monats (1 - 31)
DAY	Name des Tages ('MONTAG' bis 'SONNTAG')
day	Name des Tages ('montag' bis 'sonntag')
Day	Name des Tages ('Montag' bis 'Sonntag')
MM	Monat des Jahres (1 - 12)
MON	Monatsname dreistellig ('JAN' bis 'DEZ')
mon	Monatsname dreistellig ('jan' bis 'dez')
Mon	Monatsname dreistellig ('Jan' bis 'Dez')
MONTH	Monatsname ('JANUAR' bis 'DEZEMBER')
month	Monatsname ('januar' bis 'dezember')
Month	Monatsname ('Januar' bis 'Dezember')
YY	Jahr zweistellig (00 bis 99)
YYYY	Jahr vierstellig
HH24	Uhrzeit: Stunde (0 - 24)

MI	Uhrzeit: Minute (0-60)
SS	Uhrzeit: Sekunde (0-60)
IW	Kalenderwoche nach ISO
Q	Quartal (1, 2, 3, 4)
- / , . ; : .	Formatierungszeichen

4.3.3. Beispiele

```
sysdate+1
```
Ergebnis: morgen um die gleiche Zeit

```
round(sysdate)
```
Ergebnis: heute um 00:00:00 Uhr

```
last_day(to_date('10.12.2002', 'dd.mm.yyyy'))
```
Ergebnis: `31.12.2002 00:00:00`

```
months_between(to_date('25.12.2002', 'dd.mm.yyyy'),
to_date('10.11.2002', 'dd.mm.yyyy'))
```
Ergebnis: `1,48387097`

```
months_between(to_date('25.12.2002', 'dd.mm.yyyy'),
to_date('25.11.2002', 'dd.mm.yyyy'))
```
Ergebnis: `1`

Als Basis zur Berechnung werden immer 31 Tage je Monat zugrunde gelegt.

4.4. Sonstige Funktionen

4.4.1. Übersicht

greatest(e1[,e2] ...)	größter Wert der Ausdrücke
least(e1[,e2] ...)	kleinster Wert der Ausdrücke
nvl(e1, e2)	ist e1 NULL dann e2 sonst e1
nvl2(e1, e2, e3)	ist e1 NULL dann e3 sonst e2 (ab 8i)
user	aktueller Datenbankbenutzername
userenv(s)	Informationen zur Benutzerumgebung
dump(e)	interne Kodierung von e
vsize(e)	benötigter Speicherplatz in Bytes

4.4.2. Beispiele

```
decode (status,'A','Angelegt','E','Erledigt',
'S','Storniert','Unbekannt')
```

Je Nach Status werden unterschiedliche Zeichenketten zurück geliefert. Z.b. Bei 'E' 'Erledigt'. Ist der Status nicht 'A', 'E' oder 'S' liefert `decode` 'Unbekannt'.

Mit `decode` lassen sich Berechnungen durchführen, die sonst nur mittels Programmierung realisierbar wären. Typische Anwendungen sind Kategorisierungen.

```
nvl(artikel_nr, 999999)
```

Ergebnis: 999999 wenn die Artikelnummer nicht gefüllt ist sonst die Artikelnummer

```
greatest(4, 7, 1)
```

Ergebnis: `7`

```
vsize(sysdate)
```

Ergebnis: `8`

4.5. Konvertierungsfunktionen

4.5.1. Übersicht

to_char(a[,fmt])	Umwandlung der Zahl in eine Zeichenkette je nach Format fmt.
to_char(d[,fmt])	Umwandlung des Datums d in eine Zeichenkette je nach Format fmt.
to_date(s[,fmt])	Umwandlung der Zeichenkette s in ein Datum
to_number(s[,fmt])	Umwandlung der Zeichenkette s in eine Zahl
hextoraw(s)	Umwandlung einer Zeichenkette s in Binärdaten
rawtohex(b)	Umwandlung von Binärdaten b in eine Zeichenkette mit entsprechenden Hex-Ziffern

4.5.2. Beispiele

```
insert into druckersteuerung (befehl, code) values ('6 Zeilen/Zoll',
hextoraw('1B266C3644'));
```

```
to_char(23012.9, '000G000D00')
```
Ergebnis: `'023.012,90'`

```
to_char(to_date('24.12.2002','dd.mm.yyyy'),'hh24:mi:ss')
```
Ergebnis: `'00:00:00'`

```
select to_char (anzahl * preis, '999G990D00') Umsatz
from auftrag_pos
where auftrag_nr = 1;
```

```
select preis * to_number ('1,8', '9D9')
from auftrag_pos
where auftrag_nr = 1;
```

4.5.3. Anmerkungen

Konvertierung von Datum in Zeichenkette (to_char) und umgekehrt (to_date)

- Datum in Zeichenkette: `to_char (datum, format)`
- Zeichenkette in Datum: `to_date (zeichenkette, format)`

5. SELECT – Baumstrukturen

Es gibt ein einfaches Konstrukt, um Baumstrukturen beliebiger Tiefe abzubilden. Dazu wird eine Spalte eingefügt, in der der Schlüssel des Vorgängers eingetragen wird. Über die "**CONNECT BY**"-Klausel werden dann die Zeilen verknüpft. Die Pseudospalte "**LEVEL**" enthält die Ebene eines Objektes innerhalb der Baumstruktur.

```
CREATE TABLE Angestellter (
    VName VARCHAR(25) NOT NULL,
    MName CHAR,
    NName VARCHAR(25) NOT NULL,
    SSN CHAR(9),
    GDatum DATE,
    Adresse VARCHAR(60),
    Geschlecht CHAR,
    Gehalt DECIMAL(10,2),
    SuperSSN CHAR(9),
    Abt INT,
    PRIMARY KEY(SSN),
    FOREIGN KEY(SuperSSN) REFERENCES Angestellter(SSN)
);
```

Die Pseudospalte `LEVEL` kann verwendet werden, um die Hierarchieebene optisch durch Einrückung darzustellen:

```
SELECT level ,SSN, LPAD(' ',2*(LEVEL-1)) || name name, chef_pnr
FROM Angestellter
START WITH SSN=1
CONNECT TO PRIOR SSN = SuperSSN;
```

6. SELECT – Mengenoperationen

Mengenoperationen dienen dazu, die Ergebnismengen zweier Abfragen zu einer zusammenzufassen. Die Mengen können vereinigt (UNION / UNION ALL), geschnitten (INTERSECT) oder von einander subtrahiert (MINUS) werden. Damit diese Operatoren angewendet werden können, müssen die selektierten Spalten in gleicher Anzahl und vom gleichen Typ ein. Die Zuordnung erfolgt über die Spaltenreihenfolge.

Es ist nahezu beliebig möglich diese Operatoren zu verketten oder zu verschachteln.

6.1. Operator UNION / UNION ALL

Die Ergebnismenge einer UNION-Operation enthält alle Zeilen aus beiden Abfragen. Bei "UNION ALL" werden vollständig identische Zeilen nicht entfernt.

Beispiel: Saisonale Verteilung der Aufträge:

```
(SELECT DISTINCT PNUMMER
      FROM PROJEKT, ABTEILUNG, ANGESTELLTER
      WHERE ABTNR=ABTNUMMER AND MGRSSN=SSN AND NNAME='Smith')
UNION
(SELECT DISTINCT PNUMMER
      FROM PROJEKT, ARBEITET_AN, ANGESTELLTER
      WHERE PNUMMER=PNR AND ESSN=SSN AND NNAME='Smith');
```

.

6.2. Operator INTERSECT

Die Ergebnismenge enthält die Schnittmenge der Teilmengen, d.h. die Datensätze müssen in beiden Abfragen enthalten sein. Oft ist dieser Operator auch durch "EXISTS" oder "IN" realisierbar.

Beispiel:

```
(SELECT DISTINCT PNUMMER
      FROM PROJEKT, ABTEILUNG, ANGESTELLTER
      WHERE ABTNR=ABTNUMMER AND MGRSSN=SSN AND NNAME='Smith')
```

```
INTERSECT
(SELECT DISTINCT PNUMMER
      FROM PROJEKT, ARBEITET_AN, ANGESTELLTER
      WHERE PNUMMER=PNR AND ESSN=SSN AND NNAME='Smith');
```

6.3. Operator MINUS

Die Ergebnismenge enthält die Differenz der Teilmengen, d.h. die Datensätze müssen in der ersten, dürfen aber nicht in der zweiten Abfragen enthalten sein. Oft ist dieser Operator auch durch "NOT EXISTS" oder "NOT IN" realisierbar

Beispiel: Artikel, die zwar eingekauft, nicht aber verkauft wurden:

```
select distinct artikel_nr
from bestell_pos p
MINUS
select distinct artikel_nr
from auftrag_pos;
```

7. SQL - Insert

7.1. Syntax für das Einfügen (INSERT) einer Zeile

```
INSERT INTO <tabelle> (<spalte_1>,..,<spalte_n>)
values (<wert_1>,..,<wert_n>);
```

7.2. Syntax für das Einfügen mehrerer Zeilen

```
INSERT INTO <tabelle> (<spalte_1>,..,<spalte_n>) <abfrage>
```

Alle definierten Indizes werden automatisch aktualisiert.

7.3. Beispiele

```
INSERT INTO einheit (einheit_kurz, bezeichnung)
VALUES ('ml', 'Milliliter');
INSERT INTO einheit VALUES ('ml', 'Milliliter');
```

7.4. Einfügen von mehreren Zeilen mittels einer Abfrage

```
INSERT INTO (einheit_kurz, bezeichnung)
SELECT einheit_ref, 'Bezeichnung von ' || einheit_ref
FROM artikel;
```

7.5. Verwenden eines Nummerngenerators zur automatischen Erzeugung eines Schlüssels

```
INSERT INTO artikel (artikel_nr, bezeichnung, einheit_ref)
VALUES (sq_generator.nextval, 'Butter', 'gr');
```

7.6. Anmerkungen

- Besser dedizierte Sequenzen mir create sequence … erzeugen. Nächster Wert mit sequenzname.nextval

```
CREATE SEQUENCE abteilungs_seq
 START WITH     1
```

```
    INCREMENT BY   1
    NOCACHE
    NOCYCLE;
```

8. Update

8.1. Syntax zum Ändern (UPDATE) von Daten:

```
update <tabelle> set
    <spalte_1>=<wert_1>, ... ,
    <spalte_n>=<wert_n>
where <bedingung>;
```

- Alle definierten Indizes werden automatisch aktualisiert.
- Alle geänderten Datensätze werden automatisch bis zum Transaktionsende gesperrt.

8.2. Beispiele

Das Datum aller Aufträge um einen Tag verschieben:

```
UPDATE auftrag SET datum=datum+1;
```

Den Status eines Auftrags auf 'E' (erledigt) setzen:

```
UPDATE auftrag SET status='E' WHERE auftrag_nr=1;
```

Alle Aufträge mit Artikel 4711 stornieren, die noch nicht erledigt sind:

```
UPDATE auftrag SET status='S'
WHERE status='A' AND
auftrag_nr IN
(SELECT auftrag_nr FROM auftrag_pos WHERE artikel_nr=4711);
```

8.3. Anmerkungen

9. Löschen

9.1. Syntax für das Löschen von Tabelleninhalten

9.1.1. Syntax für das Löschen (DELETE) aller Zeilen einer Tabelle

```
DELETE <tabelle>;
```

9.1.2. Syntax für das Löschen ausgewählter Zeilen einer Tabelle

```
DELETE FROM <tabelle> WHERE <bedingung>;
```

9.1.3. Beispiel

Alle Auftragspositionen zu Artikel 4711 löschen:

```
DELETE FROM auftrag_pos WHERE artikel_nr=4711;
```

9.2. Syntax für das Löschen (DELETE) einer Tabelle

DROP table <tabelle>

9.3. Anmerkungen

- Es gibt standardmäßig kein UNDO!
- Es folgt keine Sicherheitsabfrage
- Alle definierten Indizes werden automatisch aktualisiert.
- Alle gelöschten Datensätze und Tabellen werden automatisch bis zum Transaktionsende gesperrt.

10. Oracle-Datentypen

10.1. Übersicht:

VARCHAR2 (n)	Variable Zeichenkette der maximalen Länge n, n zwischen 1 und 4000
VARCHAR (n)	wie VARCHAR2
CHAR (n)	Feste Zeichenkette von n Byte, n zwischen 1 und 2000
NCHAR, NVARCHAR	Zeichenketten mit anderem Zeichensatz als dem der Datenbank
NUMBER (p, s), DECIMAL(p, s)	p von 1 bis 38 (Gesamtzahl der Stellen) und s von -84 bis 127 (Vor- bzw. Nachkommastellen)
DATE	Gültiger Datumsbereich von -4712 bis 31.12.9999 enthält immer auch die sekundengenaue Uhrzeit
LONG	Variable Zeichenkette bis zu 2 GB
RAW (n)	Binärdaten der Länge n, n zwischen 1 und 2000 Bytes
LONG RAW	Binärdaten bis zu 2 GB
CLOB	Zeichenketten bis 4 GB
BLOB	Binärdaten bis 4 GB
CFILE, BFILE	Zeiger auf Dateien (Text, Binär)

10.2. Beispiele

```
CREATE TABLE Angestellter (
    VName VARCHAR(25) NOT NULL,
```

```
    MName CHAR,
    NName VARCHAR(25) NOT NULL,
    SSN CHAR(9),
    GDatum DATE,
    Adresse VARCHAR(60),
    Geschlecht CHAR,
    Gehalt NUMBER(10,2),
    SuperSSN CHAR(9),
    Abt INT,
    PRIMARY KEY(SSN),
    FOREIGN KEY(SuperSSN) REFERENCES Angestellter(SSN)
);
```

10.3. Anmerkungen

- Bei Fremdschlüssel auf Typkompatibilität achten.
- Alle **ANSI-Datentypen** sind verfügbar und werden auf die obigen Datentypen abgebildet (z.B. **CHARACTER, DECIMAL, INTEGER, FLOAT**).
- Die Datentypen für unstrukturierte Daten (LONG, LONG RAW, LOBs unterliegen starken Einschränkungen. Die Manipulation solcher Objekte ist nur mit einer geeigneten Programmiersprache und nicht mit SQL möglich (PL/SQL, OCI).
- Nicht relationale Objekte, sogenannte objekt-relationale Datentypen (zum Beispiel nested Tables) sind ebenfalls nur per Programmierung verwendbar.

11. Oracle-Constraints

11.1. Übersicht (weitere im Skript)

NOT NULL	Spalte muss stets gefüllt sein.
UNIQUE	Spalte oder Spaltenkombination ist eindeutig.
PRIMARY KEY	Spalte oder Spaltenkombination ist Primärschlüssel.
FOREIGN KEY	Spalte oder Spaltenkombination muss in einer separaten Tabelle als Schlüssel vorhanden sein
ON DELETE CASCADE	Löschen eines Datensatzes führt zum kaskadierenden Löschen der, über foreign key constraints verbundenen Datensätze.
CHECK	Boolscher Ausdruck ist wahr

11.2. Beispiele:

```
create table einheit (
  einheit_kurz varchar2(10) constraint pk_einheit primary key,
  bezeichnung varchar2(40)
);

create table artikel (
  artikel_nr number(5) not null,
  bezeichnung varchar2(40) not null,
  einheit_ref varchar2(10) not null,
  constraint pk_artikel primary key (artikel_nr),
  constraint fk_artikel_einheit foreign key (einheit_ref)
```

```
  references einheit(einheit_kurz)
);

alter table auftrag_pos add constraint pk_auftrag_pos
primary key (auftrag_nr, pos);

alter table auftrag_pos add constraint ck_auftr_pos_artikel_text
check (artikel_nr is not null or text is not null);

alter table auftrag_pos add constraint fk_auftrag_pos_auftrag
foreign key (auftrag_nr) references auftrag(auftrag_nr);
```

12. NULL-Werte

Der Wert **NULL** steht für den nicht definierten Wert. Vergleiche mit diesem Wert liefern immer FALSE.

So liefert die folgende Abfrage niemals ein Ergebnis:

```
select auftrag_nr, pos
from auftrag_pos
where artikel_nr=NULL;
```

Für den korrekten Vergleich sind deshalb die Operatoren "**is NULL**" bzw. "**is not null**" zu verwenden. Die obige Abfrage lautet korrekt:

```
select auftrag_nr, pos
from auftrag_pos
where artikel_nr is NULL;
```

Gründe für NULL-Werte:

- Wert nicht bekannt
- Wert noch nicht bekannt/eingetragen
- Wert kann nicht existieren (z.B. Brustkrebs beim Mann, Prostatakrebs bei der Frau)