Folien zur Vorlesung Datenbanken

Kapitel 3

Datenbanksprache SQL - Abfrage

Fachhochschule Wedel

Prof. Dr. Ulrich Hoffmann

basierend auf den Lehrmaterialien von
Prof. Dr. Hans–Detlef Gerhardt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

3. Datenbanksprache SQL - Abfrage

3.1 Datenbank - Tabellen

3.2 Einführung in die vollständige SELECT - Anweisung

3.3 Komponenten der SELECT - Anweisung

fhwedel
UNIVERSITY OF APPLIED SCIENCES

3. Datenbanksprache SQL - Abfrage

3.1 Datenbank – Tabellen

PNR	NAME	VOR-NAME	GEH_ STUFE	ABT_ NR	KRANKEN-KASSE
167	Krause	Gustav	it3	d12	dak
168	Hahn	Egon	it4	d11	bek
123	Lehmann	Karl	it3	d13	aok
133	Schulz	Harry	it1	d13	aok
124	Meier	Richard	it5	d13	aok
125	Wutschke	Oskar	it3	d13	aok
126	Schroeder	Karl-Heinz	it4	d13	aok
227	Wagner	Walter	it2	d13	dak
234	Krohn	August	it4	d13	aok
135	Tietze	Lutz	it2	d13	tkk
156	Hartmann	Juergen	it1	d14	bek
127	Ehlert	Siegfried	it1	d15	kkh
157	Schultze	Hans	it1	d14	aok
159	Osswald	Petra	it2	d15	dak
137	Haase	Gert	it1	d11	kkh
134	Meier	Gerd	it5	d11	tkk

Tabelle 1: PERSONAL

Tabelle 2: MASCHINE

MNR	NAME	PNR	ANSCH_ DATUM	NEU-WERT	ZEIT-WERT
1	Bohrmaschine	123	01-feb-99	30000	15000
2	Bohrmaschine	123	01-jul-02	30000	18000
3	Fraesmaschine	124	04-jan-98	40000	10000
11	Hobelmaschine	127	15-jan-02	29000	19000
12	Drehbank	126	01-aug-99	31000	21000
14	Hobelmaschine	123	01-nov-98	32000	22000
16	Drehbank	134	25-nov-01	32000	23000
17	Bohrmaschine	127	01-feb-03	31000	25000

Tabelle 3:KIND

PNR	K_NAME	K_VORN	K_GEB
167	Krause	Fritz	1997
167	Krause	Ida	1999
123	Lehmann	Sven	2002
123	Lehmann	Karl	2004
168	Hahn	Hans	1993
133	Wendler	Klaus	1996
124	Meier	Gustav	1999
124	Meier	Susi	2002
124	Meier	Dirk	2004

Tabelle 4: GEHALT

GEH_STUFE	BETRAG
it1	2523
it2	2873
it3	3027
it4	3341
it5	3782

Tabelle 5: ABTEILUNG

ABT_NR	NAME
d11	Verwaltung
d12	Projektierung
d13	Produktion
d14	Lagerung
d15	Verkauf

Tabelle 6: PRAEMIE

PNR	P_BETRAG
227	550
227	610
227	250
124	250
234	600
234	500
127	300
168	600
168	700

3.2 Einführung in die vollständige SELECT-Anweisung

Syntax der SELECT - Anweisung:

SELECT [ALL|DISTINCT] ausdruck

FROM {tabellenname|viewname}[aliasname]
[,{tabellenname|viewname}[aliasname]. . .]

[**WHERE** suchbedingungen]

[**GROUP BY**
nicht aggregierender ausdruck
[,nicht aggregierender ausdruck]...]
[HAVING suchbedingungen]

[**ORDER BY**
{ spaltenname | nummer in spaltenliste }[{ASC|DESC}]
[,{spaltenname|nummer in spaltenliste} [{ASC|DESC}]]...]

ausdruck:
*|spaltenname[,spaltenname][,"echter" ausdruck]

spaltenname:
Spaltenname | tabellenname.Spaltenname |
viewname.Spaltenname

Wir betrachten die Tabelle MASCHINE:

MNR	NAME	PNR	ANSCH_ DATUM	NEU- WERT	ZEIT- WERT
1	Bohrmaschine	123	01-feb-99	30000	15000
2	Bohrmaschine	123	01-jul-02	30000	18000
3	Fraesmaschine	124	04-jan-98	40000	10000
11	Hobelmaschine	127	15-jan-02	29000	19000
12	Drehbank	126	01-aug-99	31000	21000
14	Hobelmaschine	123	01-nov-98	32000	22000
16	Drehbank	134	25-nov-01	32000	23000
17	Bohrmaschine	127	01-feb-03	31000	25000

Tabelle 7: Tabelle MASCHINE

Gesucht sind die Personalnummern derjenigen Mitarbeiter, die an mindestens 2 Maschinen mit einem Zeitwert jeweils über 17.000,- € arbeiten dürfen.

Ergebnis ist aufsteigend sortiert auszugeben.

Anfrage:

```
SELECT      PNR
FROM        MASCHINE
WHERE       ZEITWERT > 17000
GROUP BY    PNR
HAVING      COUNT(ZEITWERT)>1
ORDER BY    PNR;
```

Bearbeitung:

Jede Komponente liefert als **Zwischenergebnis eine Tabelle**, die als Eingabe für die nächste Komponente dient. (Für den Nutzer nicht sichtbar.)

1. **FROM**-Komponente ⇒ mit Tabelle MASCHINE muss gearbeitet werden.
   Ergebnis: Kopie der Tabelle MASCHINE

2. **WHERE**-Komponente ⇒ alle Zeilen, deren Spalte ZEITWERT die Bedingung erfüllt

MNR	NAME	PNR	ANSCH_ DATUM	NEU- WERT	ZEIT- WERT
2	Bohrmaschine	123	01-jul-02	30000	18000
11	Hobelmaschine	127	15-jan-02	29000	19000
12	Drehbank	126	01-aug-99	31000	21000
14	Hobelmaschine	123	01-nov-98	32000	22000
16	Drehbank	134	25-nov-01	32000	23000
17	Bohrmaschine	127	01-feb-03	31000	25000

3. GROUP BY-Komponente

⇒ gruppiert die Zeilen auf der Basis von Werten in der Spalte PNR.

Gruppieren heißt, die Zeilen zusammenzufassen, die den gleichen Wert in mehreren Zeilen der ausgewählten Spalte haben.

MNR	NAME	PNR	ANSCH_ DATUM	NEU-WERT	ZEIT-WERT
2, 14	Bohrmaschine Hobelmaschine	123	01-jul-02, 01-nov-98	30000, 32000	18000, 22000
11 17	Hobelmaschine Bohrmaschine	127	15-jan-02 01-feb-03	29000 31000	19000 25000
12	Drehbank	126	01-aug-99	31000	21000
16	Drehbank	134	25-nov-01	32000	23000

Mit Ausnahme der Spalte PNR können alle anderen Spalten **mehrere Werte** enthalten.

fhwedel
UNIVERSITY OF APPLIED SCIENCES

4. HAVING-Komponente

⇒ bezieht sich auf das gruppierte Zwischenergebnis der

GROUP BY-Komponente,

wirkt wie **WHERE**-Komponente:

MNR	NAME	PNR	ANSCH_ DATUM	NEU- WERT	ZEIT- WERT
2, 14	Bohrmaschine Hobelmaschine	**123**	01-jul-02, 01-nov-98	30000, 32000	18000, 22000
11 17	Hobelmaschine Bohrmaschine	**127**	15-jan-02 01-feb-03	29000 31000	19000 25000

5. SELECT-Komponente

⇒ wählt genannte Spalten aus

PNR: 0123, 0127

6. ORDER BY-Komponente

⇒ sortiert die Zeilen in die gewünschte Reihenfolge

Ergebnis auf dem Bildschirm:

PNR

0123
0127

3.3 Komponenten der SELECT - Anweisung

3.3.1 FROM-Komponente

Gibt die verwendeten Tabellen in der **SELECT -** Anweisung an:

Dabei gilt:

- Es können **mehrere Tabellennamen** angegeben werden
- Nach einem Tabellennamen kann ein *Alias*-Name angegeben werden, das ist insbesondere sinnvoll, wenn die beteiligten Tabellen gleichnamige Spalten haben.
- Nach der Angabe eines Alias-Namen darf in der Anweisung der ursprüngliche Tabellenname nicht mehr verwendet werden

Beispiel:

1. Gib für jeden Mitarbeiter Name und Gehalt an:

```
SELECT    NAME, BETRAG
FROM      PERSONAL, GEHALT
WHERE     PERSONAL.GEH_STUFE = GEHALT.GEH_STUFE;
```

FROM PERSONAL, GEHALT

⇒ die **FROM**-Komponente berechnet *kartesisches Produkt*:

Tabelle 8: Tabelle Personal

PNR	NAME	GEH_ STUFE	GEH_ STUFE	BE- TRAG	...
123	Lehmann	it3	it1	2.523	
123	Lehmann	it3	it2	2.873	
123	Lehmann	it3	it3	3.027	
123	Lehmann	it3	it4	3.341	
123	Lehmann	it3	it5	3.782	
124	Meier	it5	it1	2.523	
124	Meier	it5	it2	2.873	
124	Meier	it5	it3	3.027	
124	Meier	it5	it4	3.341	
124	Meier	it5	it5	3.782	
125	Wutschke	it3	it1	2.523	
...					

Alle Zeilen der Tabelle PERSONAL bilden mit allen Zeilen der Tabellen GEHALT eine neue Zwischentabelle mit

Anzahl Spalten = **Summe** der Spaltenanzahlen beider Tabellen
Anzahl Zeilen = **Produkt** aus Zeilenanzahlen beider Tabellen

```
WHERE     PERSONAL.GEH_STUFE = GEHALT.GEH_STUFE;
```

⇒ die **WHERE**-Komponente wählt Zeilen aus (Selektion):

PNR	NAME	GEH_-STUFE	GEH_-STUFE	BE-TRAG	...
123	Lehmann	it3	it3	3027	
124	Meier	it5	it5	3782	
125	Wutschke	it3	it3	3027	
...					

Die Auswahl der Zeilen, in denen die Werte der genannten Spalte GEH_STUFE übereinstimmen.
Vor den Attributnamen muss der Name der Tabelle stehen.

```
SELECT      NAME, BETRAG
```

⇒ die **SELECT**-Komponente wählt Spalten aus (Projektion):

NAME	BETRAG
Lehmann	3027
Meier	3782
Wutschke	3027
...	

2. Gib die Namen aller Mitarbeiter an, die die gleiche Gehaltsstufe wie Krause haben.

Beispiel dafür, dass mit Alias-Namen gearbeitet werden muss.

```
SELECT    X.NAME
FROM      PERSONAL X, PERSONAL Y
WHERE     Y.NAME = 'Krause'
AND       X.GEH_STUFE = Y.GEH_STUFE;
```

⇒ nach **FROM**-Komponente

X				Y			
PNR	**NAME**	**GEH_ STUFE**	**...**	**PNR**	**NAME**	**GEH_- STUFE**	**...**
167	Krause	it3		167	Krause	it3	
168	Hahn	it4		167	Krause	it3	
...				...			
167	Krause	it3		168	Hahn	it4	
168	Hahn	it4		168	Hahn	it4	
...				**...**			

Man sieht: Multiplikation der Tabelle PERSONAL mit sich selbst, zur Unterscheidung wird ein Alias-Name eingeführt.

⇒ nach **SELECT**-Komponente

NAME
Krause
Lehmann
Wutschke

Zur Unterscheidung reicht es aus, nur einer Tabelle einen Alias-Namen zu geben. Also ist ebenfalls richtig:

```
SELECT    X.NAME
FROM      PERSONAL X, PERSONAL
WHERE     PERSONAL.NAME = 'Krause'
AND       X.GEH_STUFE = PERSONAL.GEH_STUFE;
```

Die Reihenfolge der Tabellenspezifikation in der **FROM-**Komponente ist ohne Einfluss auf das Ergebnis**.**

3.3.2 WHERE-Komponente

Durch die **Angabe einer Bedingung** wird festgelegt, welche Zeilen in das Resultat der Anfrage aufgenommen werden.

Mögliche Bedingungen

- der einfache Vergleich
- Vergleichsoperator mit Unterabfrage
- Bedingungen mit logischen Operatoren (**AND**, **OR**, **NOT**)
- **BETWEEN** - Operator
- **IN** - Operator mit Unterabfrage
- **LIKE** - Operator
- **NULL** - Operator
- **ANY** - Operator
- **ALL** - Operator
- **EXISTS** - Operator

Ergebnis einer Bedingung kann sein : **wahr, falsch, NULL**

Verarbeitung:
Nacheinander werden alle Zeilen geprüft, die als Zwischenergebnis der **FROM**-Komponente entstanden sind. Jede Zeile, für welches das Resultat WAHR ermittelt wird, wird als Zwischenergebnis der **WHERE**-Komponente in die Tabelle eingetragen.

3.3.2.1 Der einfache Vergleich

Ausdruck Operator (=, <=, >=, <>, <, >) Ausdruck

Numerische Werte als Wert angegeben.
Zeichenfolge in Apostroph, dabei wird Klein- und Großschreibweise streng unterschieden.
Kommt der Wert **Null** vor, so ist das Ergebnis der Bedingung gleich NULL.

Beispiele: Name und Vorname der Mitarbeiter, deren Personalnummer größer als 200 ist.

```
SELECT    NAME, VORNAME
FROM      PERSONAL
WHERE     PNR > 200;
```

Alle Maschinen (Nummer, Name), deren Abschreibung über 10.000 € liegt.

```
SELECT    MNR, NAME
FROM      MASCHINE
WHERE     ZEITWERT+10000 < NEUWERT;
```

Name und Vorname der Mitarbeiter, bei denen eine Krankenkasse eingetragen ist.

```
SELECT    NAME, VORNAME
FROM      PERSONAL
WHERE     KRANKENKASSE = KRANKENKASSE;
```

3.3.2.2 Bedingungen mit AND, OR, NOT

Zur Verkopplung mehrerer Bedingungen bzw. zur Umkehrung einer Aussage.

Reihenfolge: NOT vor **AND** vor **OR**, Änderung durch Klammerung möglich

A	*B*	*A AND B*	*A OR B*	*NOT A*
W	W	W	W	F
W	F	F	W	F
W	NULL	NULL	W	F
F	W	F	W	W
F	F	F	F	W
F	NULL	F	NULL	W
NULL	W	NULL	W	NULL
NULL	F	F	NULL	NULL
NULL	NULL	NULL	NULL	NULL

Tabelle 9: Wahrheitstabelle : AND, OR, NOT

fhwedel
UNIVERSITY OF APPLIED SCIENCES

Beispiele:

```
SELECT      NAME, VORNAME
FROM        PERSONAL
WHERE       GEH_STUFE = 'it3' AND KRANKENKASSE = ' AOK';
```

```
SELECT      NAME, VORNAME
FROM        PERSONAL
WHERE       KRANKENKASSE <>'AOK';
```

oder

```
SELECT      NAME, VORNAME
FROM        PERSONAL
WHERE       NOT KRANKENKASSE = ' AOK ';
```

3.3.2.3 IN-Operator

Ausdruck ist Element einer Menge, deren Elemente als Konstante vom gleichen Datentyp aufgezählt werden.

A sei Ausdruck; B, C, D, seien Konstanten

A **IN** (B, C, D) ⇔ (A=B) **OR** (A=C) **OR** (A=D)

A **NOT IN** (B, C, D) ⇔ **NOT** (A IN (B, C, D) ⇔ (A<>B) **AND** (A<>C) **AND** (A<>D)

Beispiel:

Name, Vorname und Krankenkasse aller Mitarbeiter, die in der AOK, KKH bzw. TKK sind.

```
SELECT      NAME, VORNAME, KRANKENKASSE
FROM        PERSONAL
WHERE       KRANKENKASSE IN ('AOK', 'KKH','TKK');
```

3.3.2.4 BETWEEN - Operator

A **BETWEEN** B **AND** C ⇔ (A>=B) **AND** (A<=C)

Voraussetzung: B <= C, sonst kein Ergebnis

NOT A **BETWEEN** B **AND** C ⇔ **NOT** (A **BETWEEN** B **AND** C) ⇔ (A<B) **OR** (A>C)

Beispiel:

Gib die Maschinen an, deren Zeitwert zwischen 20000 € und 24000 € liegt.

```
SELECT    MNR, NAME, ZEITWERT
FROM      MASCHINE
WHERE     ZEITWERT BETWEEN 20000 AND 24000;
```

3.3.2.5 LIKE-Operator

Vergleicht Datenwerte einer Spalte mit einem vorgegebenen Muster.

Allgemeine Form:

attribut_name [**NOT**] **LIKE** ’muster’ ,

wobei ‘muster’ eine Zeichenkette ist, in der % und _ eine besondere Rolle spielen:

% steht für kein, ein oder mehrere Zeichen,

_ steht für genau ein Zeichen.

Kommen im Suchstring % oder _ selbst vor, so wird die ursprüngliche

Bedeutung durch \% bzw. _ gesichert.

Beispiele:

Nummer und Name aller Maschinen, deren Name

- mit maschine endet
- als 2. Buchstaben ein r hat
- Drehbank ist

```
SELECT      MNR, NAME
FROM        MASCHINE

WHERE       NAME LIKE '%maschine'
WHERE       NAME LIKE '_r%'
WHERE       NAME LIKE 'Drehbank%'
```

3.3.3 NULL-Operator

Stehen in einer Spalte Nullwerte, so können mittels Nulloperator **die Zeilen ermittelt werden**, in deren Spalte ein Nullwert steht.

Sei z.B. Tabelle MASCHINE ergänzt durch eine Spalte TEL, (TEL: Telefonnummer des Wartungsteams, das außerbetrieblich für Reparatur und Wartung laut Wartungsvertrag zuständig ist).

Nullwert, wenn kein Wartungsvertrag abgeschlossen ist.

Tabelle 10: NULL-Operator

MNR	NAME	PNR	ANSCH_ DATUM	NEU- WERT	ZEIT- WERT	TEL
1	Bohrmaschine	123	01-feb-99	30000	15000	123456
2	Bohrmaschine	123	01-jul-02	30000	18000	NULL
3	Fraesmaschine	124	04-jan-98	40000	10000	234567
11	Hobelmaschine	127	15-jan-02	29000	19000	NULL
12	Drehbank	126	01-aug-99	31000	21000	NULL
14	Hobelmaschine	123	01-nov-98	32000	22000	345678
16	Drehbank	134	25-nov-01	32000	23000	456789
17	Bohrmaschine	127	01-feb-03	31000	25000	NULL

Beispiele:

Gib Name und Nummer aller Maschinen ohne Wartungsvertrag an.

```
SELECT    MNR, NAME
FROM      MASCHINE
WHERE     TEL IS NULL
```

Gib Name, Nummer und Telefonnummer aller Maschinen mit Wartungsvertrag an.

```
SELECT    MNR, NAME, TEL
FROM      MASCHINE
WHERE     TEL IS NOT NULL
```

a **IS NOT** NULL ist gleichwertig mit **NOT** (a **IS** NULL)

IS darf aber NICHT durch '=' ersetzt werden !

3.3.4 WHERE-Komponente mit Unterabfrage

3.4.1 Auswertung von Unterabfragen

```
SELECT  . . .       S1
FROM                A
WHERE               . . .

        (SELECT  . . .      S2
        FROM                B
        WHERE               . . .

                (SELECT  . . .      S3
                FROM        C

AND

        (SELECT  . . .      S4
        FROM                D

UNION

SELECT  . . .               S5
FROM            E
WHERE           . . .
```

Wo sind welche FROM-Tabellen-Deklarationen sichtbar?

Spalten der Tabelle A gelten in S1, S2, S3, S4.

Spalten der Tabelle B gelten in S2, S3.

Spalten der Tabelle C gelten in S3.

Spalten der Tabelle D gelten in S4.

Spalten der Tabelle E gelten in S5.

S1, S2, S3, S4 heißt **Reichweite** der Tabelle A.

3.3.4.2 IN-Operator mit Unterabfrage

Konstanten, die beim IN-Operator eingegeben werden müssen, werden erst ermittelt, wenn SQL die Anweisung abarbeitet.

⇒ nach IN-Operator folgt eine **SELECT**-Anweisung
zur Berechnung,
als ***Unterabfrage*** **(Subquery)** bezeichnet.

Abarbeitung:

- Ergebnis der Unterabfrage wird berechnet
- Ergebnis wird in die Abfrage eingesetzt

Das von der Unterabfrage ermittelte Zwischenergebnis wird nicht angezeigt.

Tabelle **KIND** mit Attributen

PNR	(von PERSONAL)
K_NAME	Name des Kindes
K_VORN	Vorname des Kindes
K_GEB	Geburtsjahr des Kindes

Tabelle 11: In-Operator

PNR	K_NAME	K_VORN	K_GEB
167	Krause	Fritz	1997
167	Krause	Ida	1999
123	Lehmann	Sven	2002
123	Lehmann	Karl	2004
168	Hahn	Hans	1993
133	Wendler	Klaus	1996
124	Meier	Gustav	1999
124	Meier	Susi	2002
124	Meier	Dirk	2004

Beispiel:

Gib Name, Vorname aller Mitarbeiter an, die Kinder haben, die nach 1994 geboren sind.

```
SELECT      NAME,VORNAME
FROM        PERSONAL
WHERE       PNR IN
        (SELECT     PNR
        FROM        KIND
        WHERE       K_GEB > 1994);
```

1. Berechnung der Unterabfrage:
   Bestimmt die Personalnummern der Mitarbeiter, die Kinder haben.
2. Name und Vorname der Mitarbeiter, die Kinder haben, werden bestimmt.

Sei k eine Spalte und seien w1, w2, ..., wn Werte, die das Zwischenergebnis einer Unterabfrage U sind, dann gilt :

k **IN** (U) ⇔ (k=w1) **OR** (k=w2) **OR** ... **OR** (k=wn)

k **NOT IN** (U) ⇔ (k<>w1) **AND** (k<>w2) **AND** ... **AND** (k<>wn)

3.3.4.3 Vergleichsoperator mit Unterabfrage

Unterabfragen nach
Vergleichsoperatoren **=, >, <, <=, >=**
sind möglich.

Berechnung entsprechend IN-Operator.

Beispiel:

Gib Personalnummer, Name, und Vorname des Mitarbeiters an, dessen Kind Klaus Wendler heißt.

```
SELECT      PNR, NAME, VORNAME
FROM        PERSONAL
WHERE       PNR=
        (SELECT     PNR
        FROM        KIND
        WHERE       K_NAME='Wendler' AND K_VORN='Klaus');
```

Gib die Namen aller Kinder an, die älter als Klaus Wendler sind.

```
SELECT      K_NAME
FROM        KIND
WHERE       K_GEB <
      (SELECT      K_GEB
      FROM         KIND
      WHERE        K_NAME = 'Wendler' AND K_VORN = 'Klaus');
```

Das Ergebnis der Unterabfrage muss genau ein Wert sein.

3.3.4.4 ANY- und ALL- Operator

Ähnlich wie IN-Operator mit Unterabfrage, nur an Stelle

"ist Element von" bei **IN** steht hier

"gilt für irgendein" **ANY** bzw.
"gilt für alle" **ALL**.

Seien w1, w2, ... wn das Ergebnis einer Unterabfrage U und K eine Spalte, so ist

K > ANY(U) ⇔ (K > w1) **OR** (K > w2) **OR** ... **OR** (K > wn)

K <= ALL(U) ⇔ (K <= w1) **AND** (K <= w2) **AND** ... **AND** (K <= wn)

Äquivalent sind auch :

K = ANY(U) ⇔ K IN (U) und

K <> ALL(U) ⇔ K **NOT** IN(U) ⇔ **NOT** (K IN (U)).

Beispiele:

Gib Name und Geburtsjahr des/der ältesten Kinder an.

```
SELECT      K_NAME, K_GEB
FROM        KIND
WHERE       K_GEB <= ALL
      (SELECT       K_GEB
      FROM          KIND);
```

Gib Name und Geburtsjahr der Kinder an, die nicht zu den ältesten zählen.

```
SELECT      K_NAME, K_GEB
FROM        KIND
WHERE       K_GEB > ANY
      (SELECT       K_GEB
      FROM          KIND);
```

3.3.4.5 Korrelierte Unterabfrage

Eine Unterabfrage heißt ***korreliert***, wenn die innere SELECT-Anweisung eine Spalte enthält, deren Werte in der äußeren SELECT - Anweisung festgelegt sind.

EXISTS – Operator:

Es wird zeilenweise geprüft, ob eine **korrelierte Unterabfrage** ein Ergebnis liefert oder nicht, in Abhängigkeit davon werden die Werte TRUE oder FALSE zurückgegeben.

Die Unterabfrage liefert also **keinen** Tabellenwert, sondern einen Wahrheitswert.

EXISTS-Operator

Beispiel:
Gib die Namen der Mitarbeiter an, die ein Kind haben.

```
SELECT          NAME
FROM            PERSONAL
WHERE EXISTS
        (SELECT         *
        FROM            KIND
        WHERE           PNR = PERSONAL.PNR);
```

Variable

Abarbeitung:

Für **jede Zeile** der Tabelle PERSONAL wird **einzeln**
überprüft, ob die Unterabfrage Zeilen liefert (=> Kind vorhanden) oder nicht (kein Kind vorhanden), also ein Resultat existiert (=TRUE) oder nicht.

Ist das Ergebnis TRUE, dann ist die entsprechende Zeile der Hauptabfrage Zwischenresultat.

Anders ausgedrückt:

PERSONAL.PNR durchläuft alle vorhandenen Personalnummern.

Die Wahrheitswerte werden durch **NOT** verändert, also z.B.

Gib die Namen der Mitarbeiter an, die **keine** Kinder haben.

```
SELECT      NAME
FROM        PERSONAL
WHERE NOT EXISTS
      (SELECT     *
      FROM        KIND
      WHERE       PNR = PERSONAL.PNR);
```

Weitere Beispiele:

Finden Sie die Namen aller Mitarbeiter, die an einer Bohrmaschine arbeiten dürfen.

Nennen Sie die Namen aller Mitarbeiter, die nicht in der Verwaltung arbeiten.

Durchschnitt:
Finden Sie die Vornamen, die sowohl Vornamen der Mitarbeiter als auch Vornamen der Kinder sind.

Differenz:
Finden Sie die Vornamen der Mitarbeiter, die keine Vornamen der Kinder sind.

ANY:
Nennen Sie Namen und Vornamen aller Mitarbeiter, die in der Abteilung Lagerung arbeiten.

ALL:
Nennen Sie den Prämienbetrag des Mitarbeiters, der die größte PNR hat.

3.3.5 GROUP BY-Komponente

Entsprechend der Spaltenspezifikation der
GROUP BY-Komponente, werden die Zeilen bei der
Berechnung des Zwischenergebnisses gruppiert.

Beispiel:

In welchen Jahren sind die Kinder der Mitarbeiter geboren ?

```
SELECT      K_GEB
FROM        KIND
GROUP BY    K_GEB;
```

Alle Zeilen mit demselben Wert von K_GEB bilden
eine Gruppe. Außer in K_GEB können in allen anderen Spalten mehrere Werte stehen.

Gruppiert werden kann auch nach zwei oder mehr Spaltenspezifikationen, dabei ist die Reihenfolge ohne Einfluss auf das Resultat.

Tabelle 12: GROUP BY

PNR	K_NAME	K_VORN	K_GEB
167	**Krause**	**Fritz**	1997
167 **124**	Krause Meier	Ida Gustav	1999
123 124	Lehmann Meier	Sven Susi	2002
123 124	Lehmann Meier	Karl Dirk	2004
168	Hahn	Hans	1993
133	Wendler	Klaus	1996

Beispiel:

```
SELECT      GEH_STUFE, KRANKENKASSE
FROM        PERSONAL
GROUP BY    GEH_STUFE, KRANKENKASSE;
```

⇔

```
SELECT      GEH_STUFE, KRANKENKASSE
FROM        PERSONAL
GROUP BY    KRANKENKASSE, GEH_STUFE;
```

Beachte:

Alle Nullwerte bilden eine Gruppe.

Die Einordnung der Nullwerte ist implementationsspezifisch.

3.3.6 HAVING-Komponente

- entspricht **WHERE**-Komponente
- bezieht sich aber auf Gruppen
- selektiert Gruppen auf der Basis von Gruppeneigenschaften
- Ausdruck der **HAVING**-Komponente kann eine oder mehrere Funktionen enthalten
- Funktionen beziehen sich auf die Werte in jeder Gruppe
- 5 **Standardfunktionen**:
  - **COUNT** ermittelt die Anzahl der Werte in jeder Gruppe der angegebenen Spalte.
  - **MIN** ermittelt den kleinsten Wert in jeder Gruppe der angegebenen Spalte.
  - **MAX** ermittelt den größten Wert in jeder Gruppe der angegebenen Spalte.
  - **SUM** ermittelt die Summe der Werte in jeder Gruppe der angegebenen Spalte.
  - **AVG** ermittelt das arithmetische Mittel der Werte in jeder Gruppe der angegebenen Spalte.

Beispiel:

Gib die Geburtsjahre an, in denen mehr als ein Kind geboren wurde.

```
SELECT      K_GEB
FROM        KIND
GROUP BY    K_GEB
HAVING      COUNT (PNR) >1;
```

Tabelle 13 : Having

K_GEB
1999
2002
2004

3.3.6.1 COUNT-Funktion

- Zählt die Anzahl der verschiedenen Werte - Nullwerte werden nicht mitgezählt.

Beispiel:

Liste alle Abteilungsnummern auf, in denen die Mitarbeiter nach mehr als 2 Gehaltstufen bezahlt werden.

```
SELECT      ABT_NR
FROM        PERSONAL
GROUP BY    ABT_NR
HAVING      COUNT (DISTINCT GEH_STUFE) > 2;
```

Tabelle 14 : COUNT

ABT_NR
d11
d13

3.3.6.2 MIN- und MAX-Funktion

- damit wird der kleinste / größte Wert in jeder Gruppe der angegebenen Spalte bestimmt
- kann auch der NULL-Wert sein.

Beispiel:

Gib die Nummern der Abteilungen an, die nur Mitarbeiter enthalten, deren Personalnummer unter 200 ist.

```
SELECT      ABT_NR
FROM        PERSONAL
GROUP BY    ABT_NR
HAVING      MAX (PNR) < 200;
```

Für jede Gruppe wird in der Spalte PNR der größte Wert bestimmt und mit 200 verglichen.

Tabelle 15 : Min- / Max

ABT_NR
d11
d12
d14
d15

3.3.6.3 SUM-Funktion

- berechnet die Summe aller Werte in jeder Gruppe der angegebenen Spalte
- ist nur für Spalten mit numerischen Datentyp zugelassen
- bei **DISTINCT** werden mehrfach auftretende gleiche Werte zu einem summiert (nur einmal gezählt)
- die Summe von nur NULL- Werten ist NULL

Beispiel:

Gib die PNR der Mitarbeiter an, die bisher mehr als 1000 € Prämie erhalten haben.

```
SELECT      PNR
FROM        PRAEMIE
GROUP BY    PNR
HAVING      SUM(P_BETRAG) > 1000;
```

Lautet die letzte Zeile

HAVING SUM(DISTINCT P_BETRAG) > 1000;

werden Prämien in gleicher Höhe nur einmal gezählt.

3.3.6.4 AVG-Funktion

- berechnet das arithmetische Mittel der Werte in jeder Gruppe der angegebenen Spalte
- ist nur für Spalten mit einem numerischen Datentyp zugelassen
- bei DISTINCT werden mehrfach auftretende gleiche Werte nur einmal berücksichtigt
- sind in einer Spalte nur NULL - Werte, ist das Ergebnis NULL, ansonsten werden NULL - Werte von der Berechnung ausgeklammert.

Beispiel:

Gib die PNR aller Mitarbeiter an, deren Mittelwert für bisher erhaltene Prämien über 500,- € liegt.

```
SELECT      PNR
FROM        PRAEMIE
GROUP BY    PNR
HAVING      AVG (P_BETRAG) > 500;
```

Tabelle 16 : Average

PNR	P_BETRAG	AVG(P_BETRAG)
227	550,610,250	470
168	600,700	650
124	250	250
234	600,500	550
127	300	300

PNR
168
234

3.3.7 SELECT-Komponente

- realisiert den Projektionsoperator, d.h. Auswahl von Spalten
- abhängig in Wirkungsweise von **GROUP BY**-Komponente

3.3.7.1 SELECT ohne GROUP BY

bezieht sich auf alle Zeilen

Syntax:

SELECT-Komponente::=
SELECT{DISTINCT|ALL}[* | [Spaltenausdruck]...]

Spaltenausdruck::= [table_name.* | numerischer Ausdruck | alphanumerischer Ausdruck | Funktion | NULL]

- verkürzte Bezeichnung ∀ Spalten einer Tabelle, enthält **FROM**-Komponente mehrere Tabellen, muss vor * der Tabellenname stehen.

COUNT(*) zählt Anzahl der Zeilen im Zwischenergebnis

gleichwertig zu:

```
SELECT      *
FROM        KIND;
```

```
SELECT      PNR, K_NAME, K_VORN, K_GEB
FROM        KIND;
```

und

```
SELECT      KIND.*
FROM        KIND, PERSONAL   ...;
```

```
SELECT      KIND.K_NAME, PERSONAL.NAME
FROM        KIND, PERSONAL     ...;
```

gleichwertig zu:

```
SELECT      K.K_NAME, P.NAME
FROM        KIND K, PERSONAL P ...;
```

Enthalten die Spaltenausdrücke Funktionen, beziehen sich diese immer auf alle Zeilen.

```
SELECT    MNR, NAME, '2007', NEUWERT- ZEITWERT FROM  MASCHINE;

SELECT    MNR, ZEITWERT-1000
FROM      MASCHINE
```

```
SELECT    GEH_STUFE, BETRAG*1.05
FROM      GEHALT;
```

Tabelle 17: Differenz

MNR	ZEITWERT-1000
1	14000
2	17000
3	9000
11	18000
12	20000
14	21000
16	22000
17	24000

Tabelle 18: Prozent

GEH_STUFE	BETRAG*1.05
IT1	2649,15
IT2	3016,65
IT3	3178,35
IT4	3508,05
IT5	3971,10

Die 5 Standardfunktionen beziehen sich jetzt immer auf alle Zeilen einer Spalte:

- **COUNT** ermittelt die Anzahl der Werte in einer Spalte/ die Anzahl der Zeilen in einer Tabelle.
- **MIN** ermittelt den kleinsten Wert in einer Spalte
- **MAX** ermittelt den größten Wert in einer Spalte
- **SUM** ermittelt die Summe der Werte in einer Spalte
- **AVG** ermittelt das arithmetische Mittel der Werte in einer Spalte.

Beispiele:

Wie viel Mitarbeiter hat der Betrieb?

```
SELECT    COUNT (PNR)
FROM      PERSONAL
```

oder

```
SELECT    COUNT (*)
FROM      PERSONAL;
```

Wie viel Mitarbeiter mit der Gehaltsstufe IT4 hat der Betrieb?

```
SELECT    COUNT (PNR)
FROM      PERSONAL
WHERE     GEH_STUFE = 'IT4';
```

Wie hoch ist die höchste Prämie?

```
SELECT    MAX (P_BETRAG)
FROM      PRAEMIE;
```

Wie oft wurde der minimale Prämienbetrag gezahlt?

```
SELECT    COUNT (P_BETRAG)
FROM      PRAEMIE
WHERE           P_BETRAG  =
      (SELECT   MIN(P_BETRAG)
      FROM      PRAEMIE);
```

Wie viel Gehaltsstufen liegen vor ?

```
SELECT    COUNT(DISTINCT GEH_STUFE)
FROM      PERSONAL;
```

Welche Gehaltsstufen sind dies?

```
SELECT    DISTINCT GEH_STUFE
FROM      PERSONAL;
```

In

```
SELECT    DISTINCT GEH_STUFE, KRANKENKASSE
FROM      PERSONAL;
```

bezieht sich DISTINCT auf alle dahinter stehenden Attribute.

3.3.7.2 SELECT mit GROUP BY

Die Funktion wird für jede Zeile, die als Ergebnis der **GROUP BY**-Komponente entsteht, auf der genannten Spalte ausgeführt.

⇒ Bei jedem Spaltenausdruck in der **SELECT**-Komponente muss eine Funktion, eine Konstante oder eine Spalte stehen, die in der **GROUP BY**-Komponente angegeben ist.

Gib für jeden Mitarbeiter, der mindestens eine Prämie erhalten hat, die Personalnummer und die Anzahl der Prämien an.

```
SELECT      PNR, COUNT (P_BETRAG)
FROM        PRAEMIE
GROUP BY    PNR;
```

Tabelle 19: Count

PNR	COUNT(P_BETRAG)
124	1
127	1
168	2
227	3
234	2

Gib für jeden Mitarbeiter, der mindestens 2 Prämien erhalten hat, die Personalnummer und die Summe der Prämien an.

```
SELECT      PNR, SUM(P_BETRAG)
FROM        PRAEMIE
GROUP BY    PNR
HAVING      COUNT(P_BETRAG)>=2;
```

Tabelle 20: Sum

PNR	SUM(P_BETRAG)
168	1.300
227	1.410
234	1.100

Gib für jeden Mitarbeiter, der mindestens eine Prämie erhalten hat, die PNR, den Text Prämiendurchschnitt anstelle des Mittelwertes der erhaltenen Prämie an.

```
SELECT      PNR, AVG(P_BETRAG) "Prämiendurchschnitt"
FROM        PRAEMIE
GROUP BY    PNR;
```

3.3.8 ORDER BY-Komponente

- **sortiert die Zeile** auf der Grundlage der angegebenen Attribute
- die Attribute werden durch ihre Namen oder durch die Angabe der Spaltennummer gekennzeichnet
- Nummer muss stehen, wenn Spaltenausdruck aus einer Funktion, einer Konstanten oder einem numerischen Ausdruck besteht
- standardmäßig aufsteigend (**ASC**) sortiert
- absteigend sortiert bei Angabe von **DESC**
- NULL-Werte je nach Implementierung eingeordnet z.B.:
  - NULL- Werte immer zuerst ausgegeben
  - NULL- Werte immer zuletzt ausgegeben
  - **NULL- Wert ist kleinster Wert (Oracle, MySQL)**
  - NULL- Wert ist größter Wert

Beispiel:

Gib die PNR der Mitarbeiter sortiert nach Gesamtprämienhöhe aus.

```
SELECT      PNR, SUM(P_BETRAG)
FROM        PRAEMIE
GROUP BY    PNR
ORDER BY    2
```

oder

```
ORDER BY    2 DESC;
```

Sortierung ist auch nach mehreren Spaltenausdrücken möglich,
z.B. absteigende Liste aller gezählten Prämien,
bei gleicher Höhe aufsteigend sortiert nach PNR

```
SELECT      PNR, P_BETRAG
FROM        PRAEMIE
ORDER BY    P_BETRAG DESC, PNR;
```

3.4 Zusammenfügen von Ergebnissen mit UNION (Vereinigung), INTERSECT (Durchschnitt), MINUS (Differenz)

Zusammenfügen mehrerer Tabellen zu einer Tabelle.

Regeln:

- alle SELECT-Anweisungen besitzen die gleiche Anzahl Spalten
- die Spalten besitzen die gleichen Datentypen
- nur die letzte SELECT - Anweisung darf die ORDER BY-Komponente enthalten, sortiert wird auf der Grundlage des Endergebnisses
- doppelte Zeilen werden automatisch gelöscht, DISTINCT darf nicht vorkommen.

Beispiel:

Gib Name und Vorname der Mitarbeiter mit den Personalnummern 167, 168, 227 an, sortiert nach NAME

```
SELECT      NAME, VORNAME
FROM        PERSONAL
WHERE       PNR=167
UNION
SELECT      NAME, VORNAME
FROM        PERSONAL
WHERE       PNR=168
UNION
SELECT      NAME, VORNAME
FROM        PERSONAL
WHERE       PNR=227
ORDER BY    1;
```

oftmals durch OR-Operator zu ersetzen:

```
SELECT      NAME, VORNAME
FROM        PERSONAL
WHERE       PNR=167 OR PNR=168 OR PNR=227
ORDER BY    PNR;
```

3.5 Subquery

- vollständige **SELECT**-Anweisung, die als rechtsseitiger Ausdruck in einer **WHERE**-Bedingung verwendet wird.

- bei Operatoren wie IN, ANY, ALL, **EXISTS** benötigt

  - **SELECT**-Komponente darf nur einen Spaltenausdruck enthalten
  - **DISTINCT** nicht erlaubt
  - **ORDER BY** nicht erlaubt

- Attributnamen gelten in der **SELECT**-Anweisung, in der ihre Tabelle angegeben ist, und in allen zugehörigen Unterabfragen.

3.6 Join

- mit Join werden Zeilen von Tabellen aneinandergefügt
- **FROM**-Komponente enthält zwei oder mehr Tabellen
- **WHERE**-Komponente enthält mindestens eine Bedingung, mit der Spalten verschiedener Tabellen verglichen werden
  (so genannte Join-Spalten)

Beispiele:

```
SELECT      *
FROM        PERSONAL, GEHALT
WHERE       PERSONAL.GEH_STUFE =
            GEHALT.GEH_STUFE;
```

```
SELECT      *
FROM        PERSONAL JOIN GEHALT
            USING (GEH_STUFE);
```

```
SELECT      *
FROM        PERSONAL NATURAL JOIN GEHALT;
```

Beispiel:

```
SELECT    PERSONAL.NAME, GEHALT.BETRAG
FROM      PERSONAL, GEHALT
WHERE     PERSONAL.GEH_STUFE =
          GEHALT.GEH_STUFE;
```

```
SELECT    PERSONAL.NAME, GEHALT.BETRAG
FROM      PERSONAL JOIN GEHALT
          USING  (GEH_STUFE);
```

Zwischen 2 Join - Spalten S1 und S2 besteht immer eine bestimmte Beziehung.

1. Die Wertemengen von S1 und S2 sind gleich (S1=S2).
2. Die Wertemenge von S1 ist Teilmenge von S2 (S1⊂ S2)
3. Die Wertemengen besitzen gemeinsame Werte (S1∩S2≠∅)
4. Die Wertemengen sind disjunkt (S1∩S2=∅).

Mit SQL ausführbar sind

- Thetajoin
- Equijoin
- natürlicher Join

3.6.1 Thetajoin

Seien R1 und R2 zwei Tabellen,
sei A ein Attribut von R1,
sei B ein Attribut von R2.

```
SELECT    *
FROM      R1, R2
WHERE     R1.A Θ R2.B;
```

```
SELECT    *
FROM      R1 JOIN R2
          ON (R1.A Θ R2.B);
```

Ist Θ der Vergleich =, dann nennt sicher der Thetajoin auch Equijoin.

Wird nur eine der Join–Spalten (die den gleichen Namen haben müssen)
in die **SELECT**-Komponente aufgenommen

⇒ natürlicher Join

⇒ alle Spalten sind anzugeben.

Habe R1 die Spalten A, A1, A2, A3,und R2 die Spalten B, B1, B2, B3, B4,

dann ist

```
SELECT    R1.A1, R1.A2, R1.A3, R2.*
FROM      R1, R2
WHERE     R1.A=R2.B;
```

gleichwertig mit

```
SELECT    R1.*, R2.B1, R2.B2, R2.B3, R2.B4
FROM      R1, R2
WHERE     R1.A=R2.B;
```

```
SELECT    R1.A1, R1.A2, R1.A3, R2.*
FROM      R1 JOIN R2
          ON (R1.A=R2.B);
```

gleichwertig mit

```
SELECT    R1.*, R2.B1, R2.B2, R2.B3, R2.B4
FROM      R1 JOIN R2
          ON (R1.A=R2.B);
```

Habe R1 die Spalten A, A1, A2, A3, und R2 die Spalten A, B1, B2, B3, B4,

dann ist

```
SELECT    R1.*, R2.B1, R2.B2, R2.B3, R2.B4
FROM      R1, R2
WHERE     R1.A=R2.A;
```

gleichwertig mit

```
SELECT    R1.*, R2.B1, R2.B2, R2.B3, R2.B4
FROM      R1 JOIN R2
          ON (R1.A=R2.A);
```

gleichwertig mit

```
SELECT    *
FROM      R1 JOIN R2  USING (A);
```

gleichwertig mit

```
SELECT    *
FROM      R1  NATURAL JOIN R2;
```

3.6.2 Equijoin

- Inner - Equijoin
- Outer - Equijoin

Beispiel:
Gib für jeden Mitarbeiter den Namen, die PNR und die erhaltene Prämie an.

```
SELECT    PERSONAL.NAME, PERSONAL.PNR,
          PRAEMIE.P_BETRAG
FROM      PERSONAL, PRAEMIE
WHERE     PERSONAL.PNR=PRAEMIE.PNR;
```

```
SELECT    NAME, PNR, PRAEMIE.P_BETRAG
FROM      PERSONAL JOIN PRAEMIE USING (PNR);
```

Tabelle 21 : Equijoin

NAME	PNR	P_BETRAG
Wagner	227	550
Wagner	227	610
Wagner	227	250
Meier	124	250
Krohn	234	600
Krohn	234	500
Ehlert	127	300
Hahn	168	600
Hahn	168	700

```
SELECT    NAME, PNR,
          PRAEMIE.P_BETRAG
FROM      PERSONAL NATURAL JOIN PRAEMIE;
```

Das Ergebnis entspricht nicht dem Gewünschten, da nur Mitarbeiter erfasst sind, die mindestens eine Prämie erhalten haben.

Es sind nur die Daten der Mitarbeiter angegeben, die in der Tabelle PRAEMIE stehen

⇒ Name **Inner-Equijoin.**

Gewünschte Antwort erhalten wir durch den

Outer-Equijoin:

Beim **Outer-Join** werden alle Datensätze der ersten Tabelle in der Join-Bedingung, denen kein Datensatz der zweiten Tabelle zugeordnet werden kann, mit einem imaginären Datensatz, der nur aus Nullwerten besteht, verbunden.

Outer-Joins können in Oracle durch die Zeichenkette **(+)** in der Join-Bedingung gekennzeichnet. Joins ohne diese Kennzeichnung sind Inner-Joins. In MySQL verwenden wir **OUTER JOIN** in der **FROM**-Klausel.

Zwei Outer-Join Attribute in einem Prädikat sind verboten.

Beispiel:

```
SELECT    NAME, PERSONAL.PNR, PRAEMIE.P_BETRAG
FROM      PERSONAL, PRAEMIE
WHERE     PERSONAL.PNR=PRAEMIE.PNR

UNION

SELECT    PERSONAL.NAME, PERSONAL.PNR, 0
FROM      PERSONAL
WHERE     PNR NOT IN
          (SELECT PNR
          FROM   PRAEMIE);
```

Einfacher zu formulieren in Oracle (aber nicht MySQL):

```
SELECT    P.NAME, P.PNR, PRAEMIE.P_BETRAG
FROM      PERSONAL P, PRAEMIE
WHERE     P.PNR=PRAEMIE.PNR(+);
```

bzw. (auch in MySQL)

```
SELECT    NAME, PERSONAL.PNR, PRAEMIE.P_BETRAG
FROM      PERSONAL  LEFT OUTER  JOIN PRAEMIE
          ON (PERSONAL.PNR = PRAEMIE.PNR);
```

Weitere Möglichkeiten von OUTER-JOINS:

RIGHT-OUTER-JOIN:

```
SELECT   NAME, PERSONAL.PNR, PRAEMIE.P_BETRAG
FROM     PRAEMIE RIGHT OUTER JOIN PERSONAL
         ON (PERSONAL.PNR = PRAEMIE.PNR);
```

FULL OUTER JOIN:

```
SELECT   NAME, PERSONAL.PNR, PRAEMIE.P_BETRAG
FROM     PERSONAL FULL OUTER JOIN PRAEMIE
         ON (PERSONAL.PNR = PRAEMIE.PNR);
```

Nur sinnvoll, wenn für die Join-Attribute A und B von R1 bzw. R2 gilt:

$$\{ A \setminus B \} \neq \emptyset \text{ und } \{ B \setminus A \} \neq \emptyset.$$

3.7 Views

- Ist eine aus einer Basistabelle abgeleitete **virtuelle Tabelle**
- View wird erst berechnet, wenn sie benötigt wird
- Nutzer hat den Eindruck, mit echter Tabelle zu arbeiten.

Wird angewandt :

- zur Vereinfachung von Anweisungen, die häufig verwendet werden
- wenn zwei oder mehrere Tabellen häufig verbunden werden
- zur schrittweisen Ausführung großer **SELECT** - Anweisungen
- um Teile von Tabellen zu schützen (Datensicherheit).

Viewdefinition:

CREATE VIEW view_name **AS**
SELECT - Anweisung;

REPLACE VIEW view_name **AS**
SELECT - Anweisung;

CREATE OR REPLACE VIEW view_name **AS**
SELECT - Anweisung;

SELECT - Anweisung ist View-Formel der View.

Löschen einer View:

DROP VIEW view_name;

Die View view_name und alle Views, auf die in der View - Formel verwiesen wird, werden gelöscht.

Aber es werden keine Daten in Tabellen gelöscht.

Beispiel:
View aller Mitarbeiter der Gehaltsstufe IT1.

```
CREATE VIEW     PERSONALIT1   AS
       SELECT   *
       FROM     PERSONAL
       WHERE    GEH_STUFE='IT1';
```

Gib die Namen aller Mitarbeiter an, die Gehaltsstufe IT1 erhalten.

```
SELECT     NAME
FROM       PERSONALIT1;
```

Die Integrität wird automatisch gewährleistet, da Views jeweils neu berechnet werden, wenn sie benötigt werden.

Einschränkung der **SELECT** - Anweisung in Viewformel:

- **UNION** - Operator nicht erlaubt
- **ORDER BY** - Komponente nicht erlaubt

Schachteln von Views ist erlaubt.

```
CREATE VIEW     PERSONALIT1AOK  AS
       SELECT   *
       FROM     PERSONALIT1
       WHERE    KRANKENKASSE = 'AOK';
```

Die Name der Spalten für VIEW

- werden übernommen
- können explizit definiert werden (sind dann bindend für VIEW)

```
CREATE VIEW     BOHRM(NR, NWERT, ZWERT) AS
       SELECT   MNR, NEUWERT, ZEITWERT
       FROM     MASCHINE
       WHERE    NAME = 'Bohrmaschine';
```

```
SELECT      *
FROM        BOHRM
WHERE       ZWERT > 16000;
```

Tabelle 22 : View

NR	NWERT	ZWERT
2	30000	18000
17	31000	25000

Bearbeitung:

Viewformel wird in die **SELECT -** Anweisung aufgenommen.