



5. Die Standardsprache SQL

Inhalt

Grundlagen

Mengenorientierte Anfragen (Retrieval)

Datenmanipulation

Datendefinition

Beziehungen und referentielle Integrität

Schemaevolution

Sichten

Indexierung





Grundlagen (1)

- **Sprachentwicklung von SQL**
 - SQL wurde „**de facto**“-**Standard** in der relationalen Welt (1986 von ANSI, 1987 von ISO akzeptiert)
 - **Wesentliche Stufen der Weiterentwicklung des Standards**
 - **SQL2 (1992): rein relational**
 - **(SQL3) SQL:1999: objekt-relational**
- **Mächtigkeit von SQL**
 - Auswahlvermögen umfasst das des Relationenkalküls und der Relationenalgebra: **relational vollständig**



Grundlagen (2)

- **Englische Aussprache von SQL**

- SQL kann als Nachfolger von SEQUEL (Structured English Query Language) betrachtet werden und wird daher oft ['sɪkwəl] ausgesprochen
- Speziell im Umfeld von:
 - Oracle
 - Microsoft
- Gemäß Standard wird es als [ɛskju:'ɛl] ausgesprochen
- So auch im Umfeld von:
 - MySQL
 - PostgreSQL



Grundlagen (3)

- **SQL: abbildungsorientierte Sprache**

- Grundbaustein: **SELECT ...**
FROM ...
WHERE ...



Abbildung

- Ein bekanntes Attribut oder eine Menge von Attributen wird mit Hilfe einer Relation in ein gewünschtes Attribut oder einer Menge von Attributen abgebildet.

- **Allgemeines Format**

<Spezifikation der Operation>

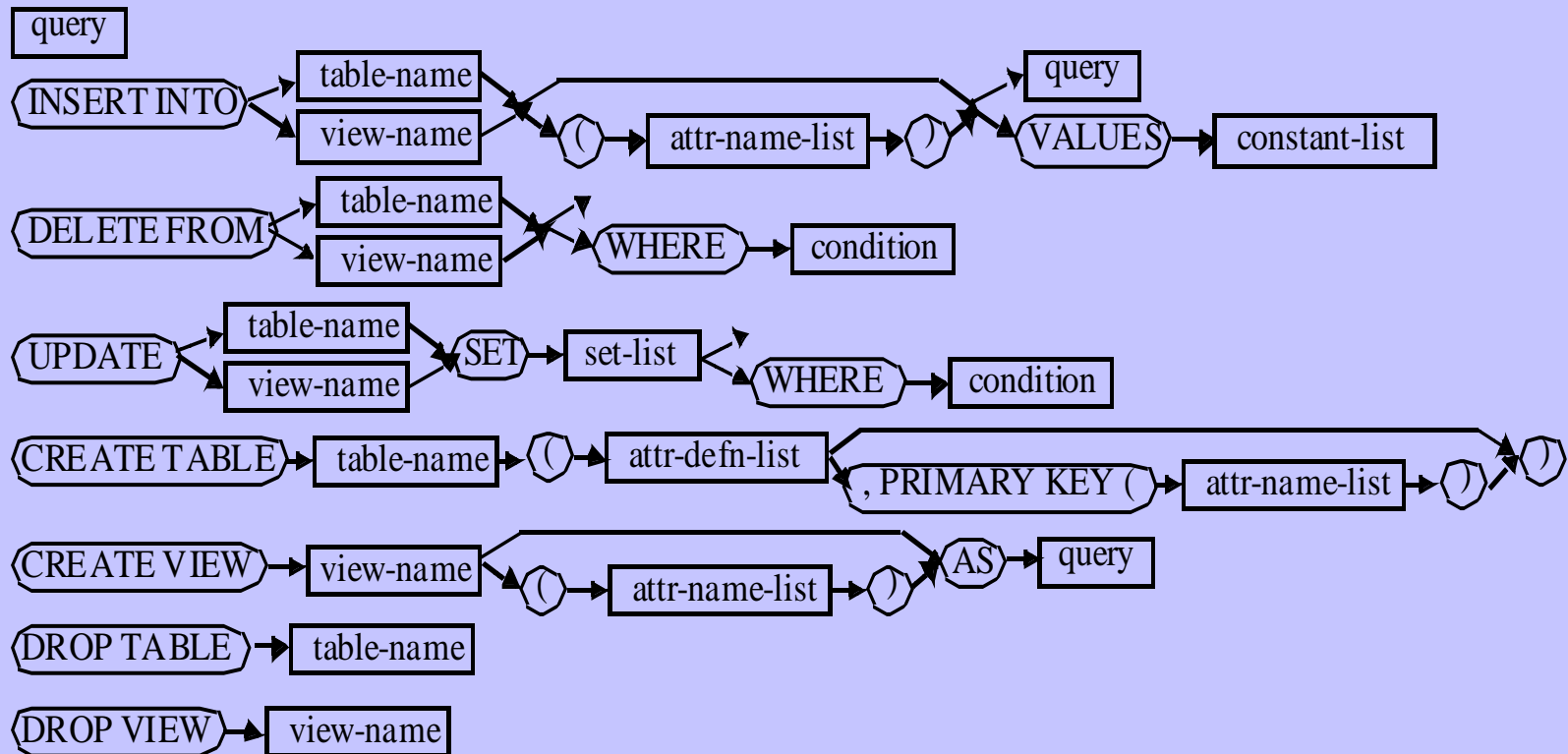
<Liste der referenzierten Tabellen>

[WHERE Boolescher Prädikatsausdruck]

Grundlagen (4)

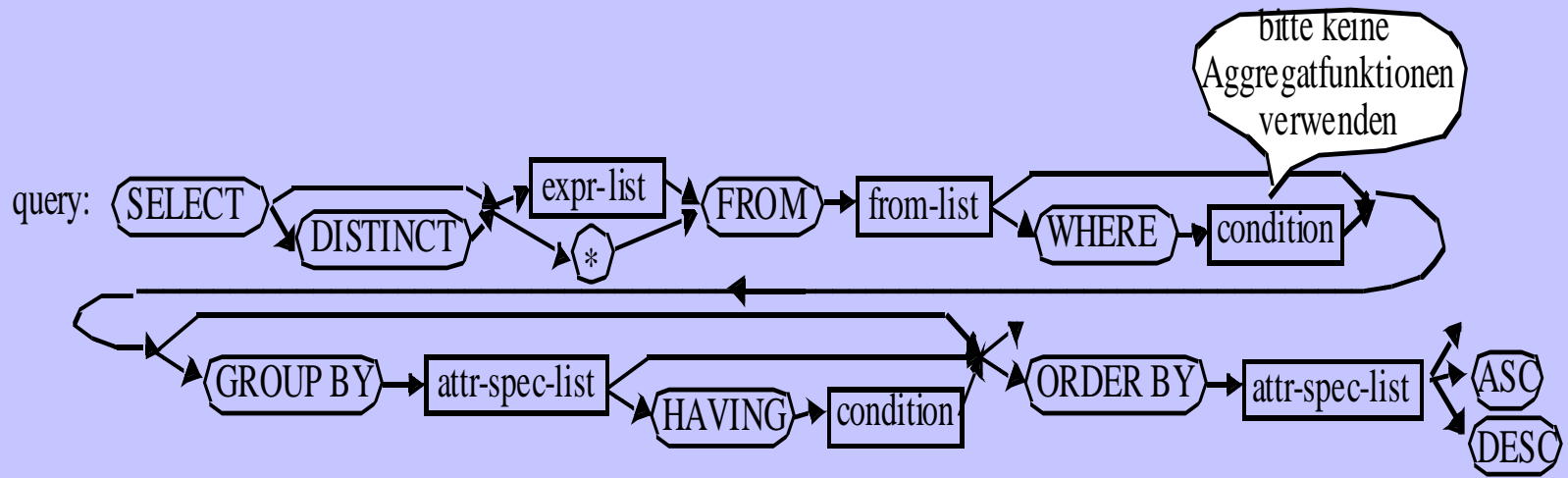
- **SQL92-Syntax** (Auszug, Table=Relation, Column=Attribut, Listenelemente durch Komma getrennt)

SQL-statement:



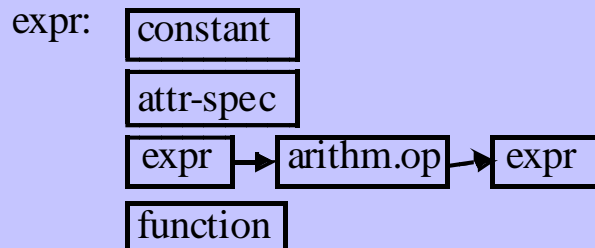
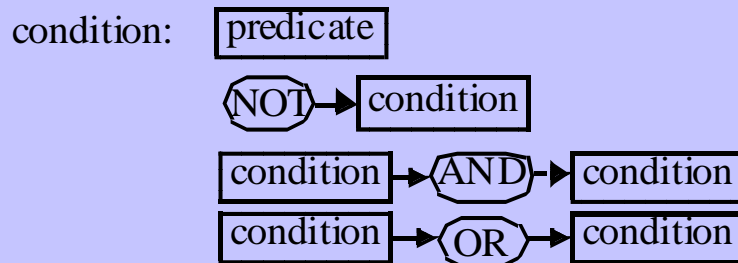
Grundlagen (5)

■ SQL92-Syntax (Forts.)



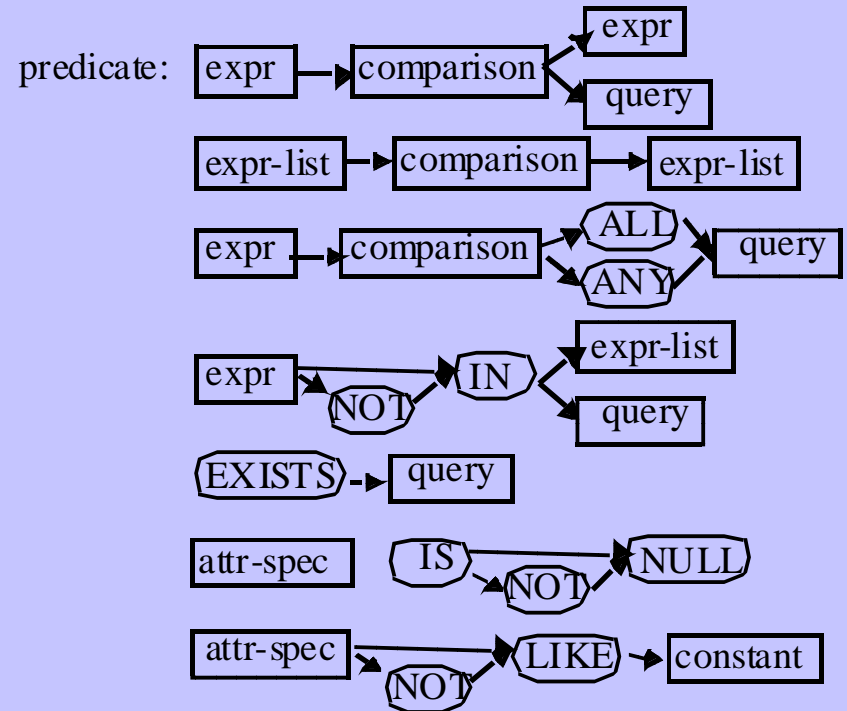
Grundlagen (6)

■ SQL92-Syntax (Forts.)



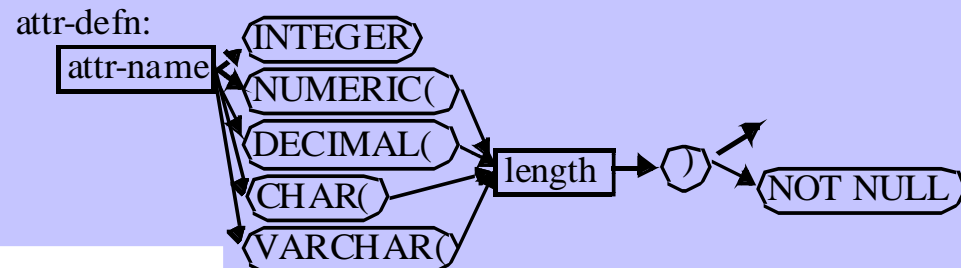
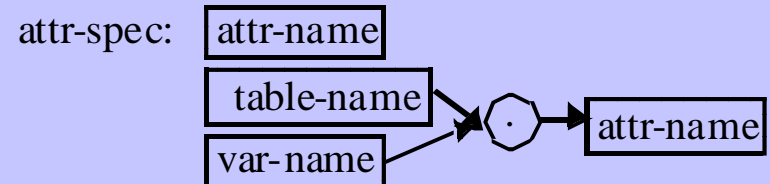
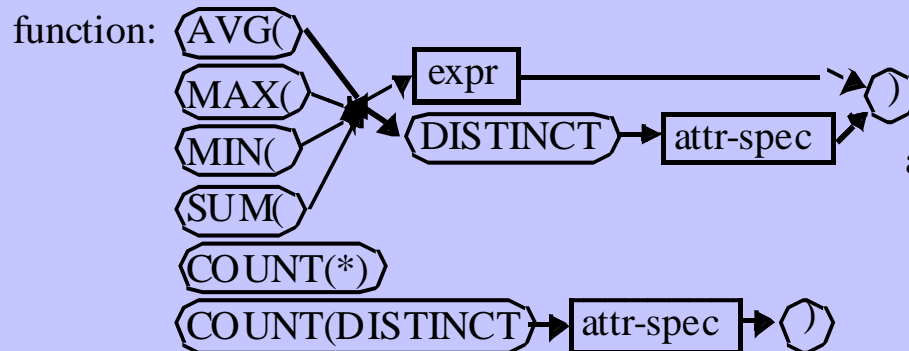
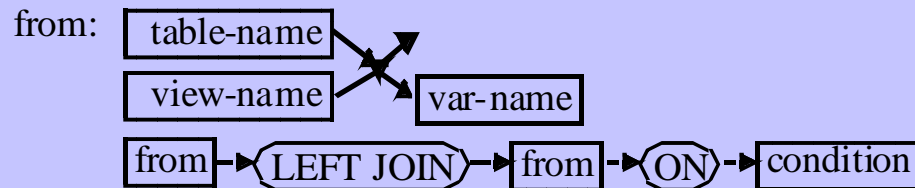
comparison: = < > <= >=

arithm.op: + - * /



Grundlagen (7)

■ SQL92-Syntax (Forts.)





Anfragen (1)

- SELECT-Anweisung

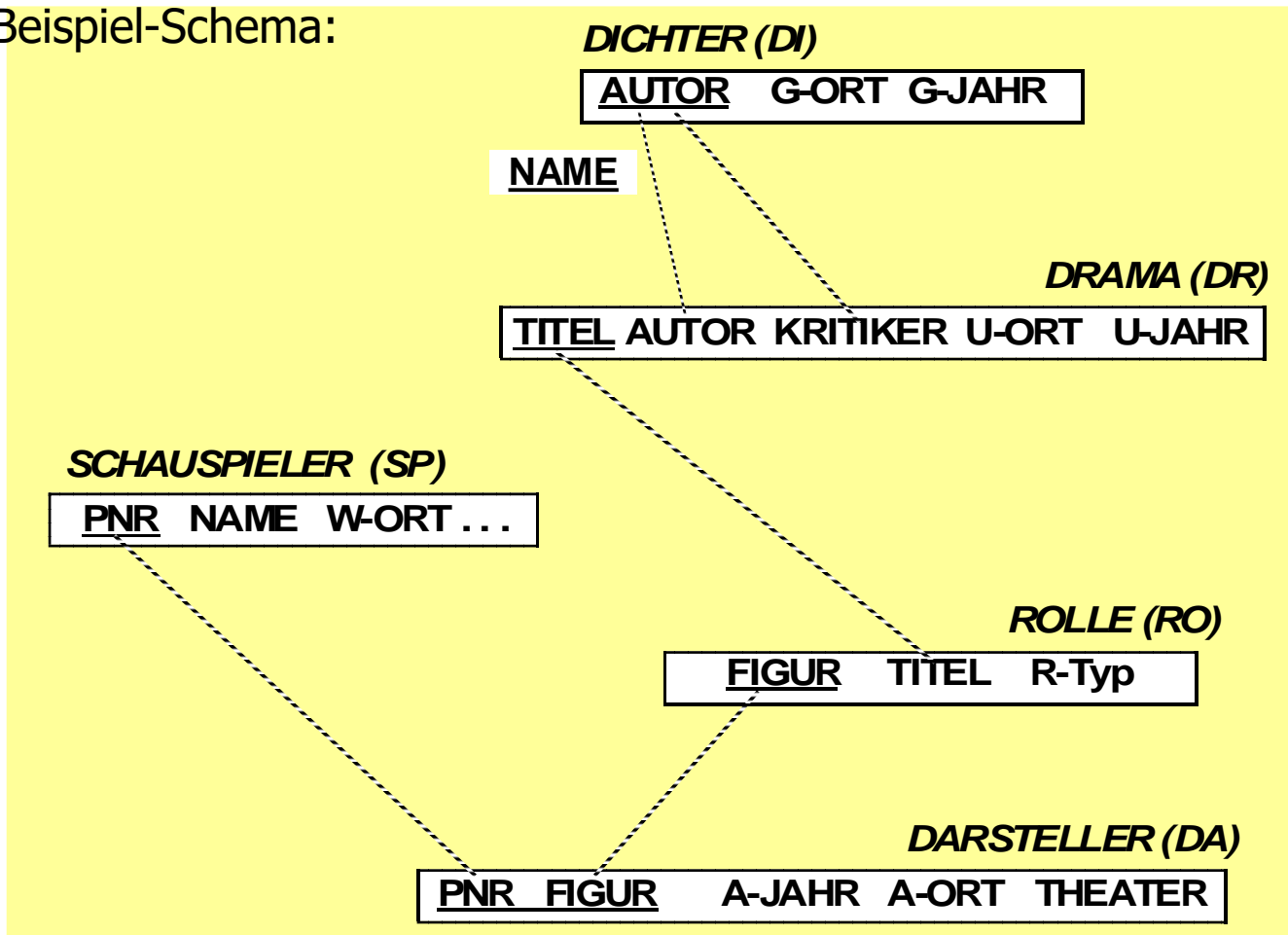
```
select-exp  
 ::= SELECT [ALL | DISTINCT] select-item-commalist  
    FROM table-ref-commalist  
    [WHERE cond-exp]  
    [GROUP BY column-ref-commalist]  
    [HAVING cond-exp]
```

- Grob:

- **SELECT * :** Ausgabe ‚ganzer‘ Tupel
- **FROM-Klausel:** spezifiziert zu verarbeitende Relation bzw.
- **WHERE-Klausel:** Sammlung (elementarer) Prädikate der Form $A_i \Theta a_i$ oder $A_i \Theta A_j$ ($\Theta \in \{ =, <>, <, \leq, >, \geq \}$), die mit *AND* und *OR* verknüpft sein können

Anfragen (2)

- Unser Beispiel-Schema:





Anfragen (3)

- **Untermengenbildung**

Welche Dramen von Goethe wurden nach 1800 uraufgeführt?

```
SELECT   *  
FROM     DRAMA  
WHERE    AUTOR = 'Goethe' AND U-JAHR > 1800;
```

- **Benennung von Ergebnis-Spalten**

- Ausgabe von Attributen, Text oder Ausdrücken
- Spalten der Ergebnisrelation können (um)benannt werden (AS)
- Beispiel:

```
SELECT    NAME,  
            'Berechnetes Alter: ' AS TEXT,  
            CURRENT_DATE – GEBDAT AS ALTER  
FROM      SCHAUSPIELER;
```



Anfragen (4)

- Test auf Mengenzugehörigkeit

- A_i **IN** (a_1, a_j, a_k) explizite Mengendefinition
 A_i **IN** (**SELECT** . . .) implizite Mengendefinition

- **Beispiel:**

Finde die Schauspieler (PNR), die Faust, Hamlet oder Wallenstein gespielt haben.

```
SELECT DISTINCT PNR
FROM DARSTELLER
WHERE FIGUR IN („Faust“, „Hamlet“, „Wallenstein“);
```

- Duplikateliminierung
 - **ALL** (Defaultwert): keine Duplikateliminierung
 - **DISTINCT** erzwingt Duplikateliminierung

Anfragen (5)

- Geschachtelte Abbildung

Welche Figuren kommen in Dramen von Schiller oder Goethe vor?

äußere
Abbildung

```
SELECT  DISTINCT FIGUR  
FROM    ROLLE  
WHERE    TITEL IN (
```

innere
Abbildung

```
SELECT  TITEL  
FROM    DRAMA  
WHERE    AUTOR IN („Schiller“, „Goethe“);
```

- Innere und äußere Relationen können identisch sein
- Eine geschachtelte Abbildung kann beliebig tief sein



Anfragen (6)

- Symmetrische Abbildung

Finde die Figuren und ihre Autoren, die in Dramen von Schiller oder Goethe vorkommen.

```
SELECT    FIGUR, AUTOR
FROM      ROLLE RO, DRAMA DR
WHERE     (RO.TITEL=DR.TITEL) AND
            (DR.AUTOR=„Schiller“ OR DR.AUTOR=„Goethe“);
```

- Einführung von **Tupelvariablen** (*correlation names*) erforderlich
- **Vorteile der symmetrischen Notation**
 - Ausgabe von Größen aus inneren Blöcken
 - keine Vorgabe der Auswertungsrichtung (DBS optimiert !)
 - (direkte Formulierung von Vergleichsbedingungen über Relationengrenzen hinweg möglich)
 - (einfache Formulierung des Verbundes)



Anfragen (7)

- Symmetrische Abbildung (Forts.)

Finde die Dichter (AUTOR, G-ORT), deren Dramen von Dichtern mit demselben Geburtsort (G-ORT) kritisiert wurden.

```
SELECT A.AUTOR, A.G-ORT
FROM   DICHTER A, DRAMA D, DICHTER B
WHERE  A.NAME = D.AUTOR
          AND    D.KRITIKER = B.NAME
          AND    A.G-ORT = B.G-ORT;
```

- Beispiel als geschachtelte Abbildung (versch. Möglichkeiten)

```
SELECT A.AUTOR, A.G-ORT
FROM   DICHTER A, DRAMA D
WHERE  A.NAME = D.AUTOR
AND    D.KRITIKER IN (SELECT B.NAME
                        FROM DICHTER B
                        WHERE A.G-ORT = B.G-ORT;
```

Anfragen (8)

- Symmetrische Abbildung (Forts.)

Finde die Schauspieler (NAME, W-ORT), die bei in Weimar uraufgeführten Dramen an ihrem Wohnort als 'Held' mitgespielt haben.

```
A:  SELECT  S.NAME, S.W-ORT
FROM    SCHAUSPIELER S, DARSTELLER D, ROLLE R, DRAMA A
WHERE      S.PNR = D.PNR
AND       D.FIGUR = R.FIGUR
AND       R.TITEL = A.TITEL
AND       A.U-ORT = 'Weimar'
AND       R.R-TYP = 'Held'
AND       D.A-ORT = S.W-ORT;
```

F1

F2

F3

F4

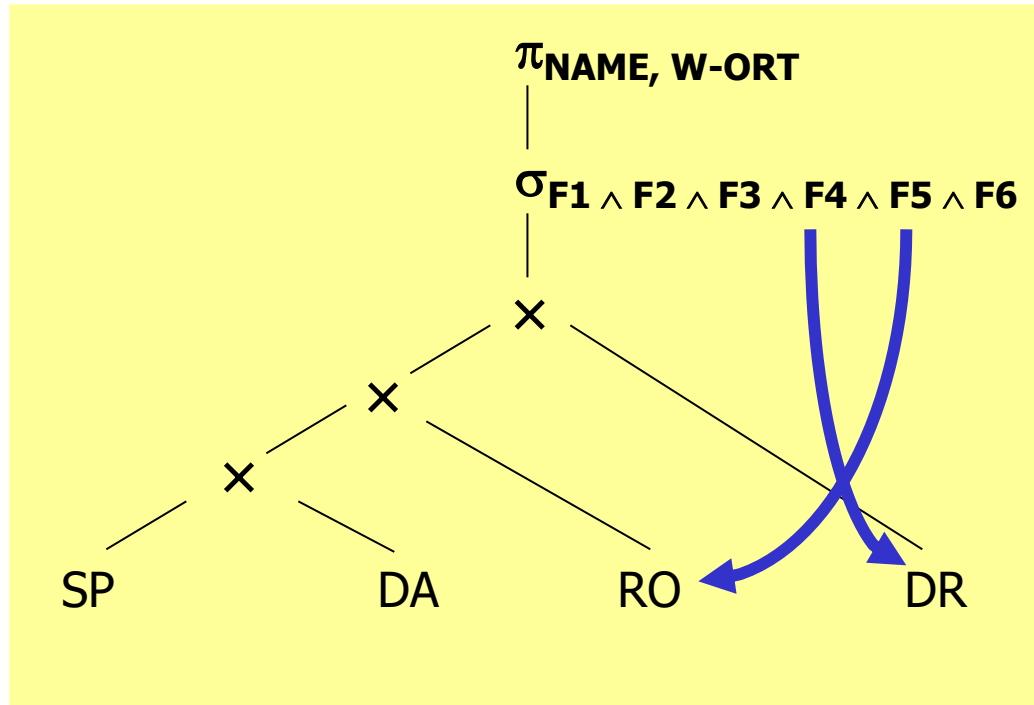
F5

F6

Diskussion: Wie sieht das Auswertungsmodell (Erklärungsmodell) bei symmetrischer Notation aus?

Anfragen (9)

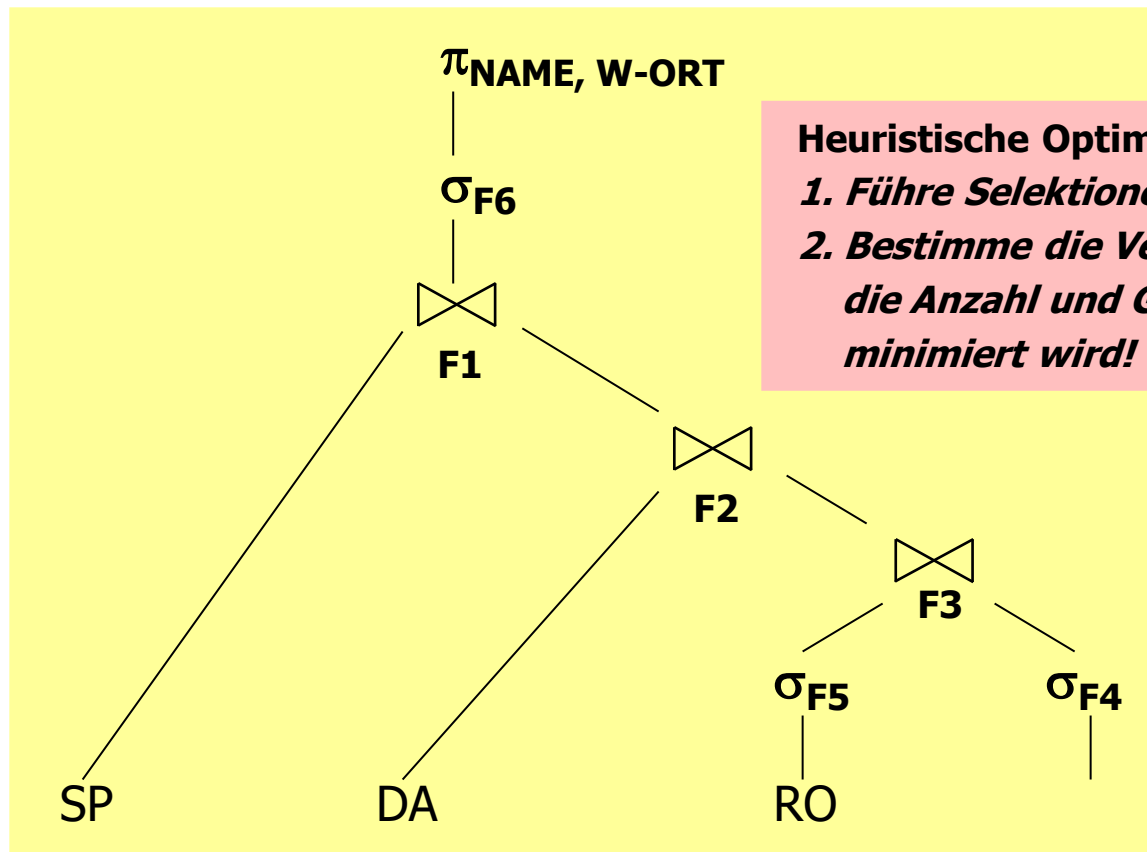
- Auswertungs-/Erklärungsmodell
 - Einfacher Operatorbaum für Anfrage **A** (siehe Folie 15)



Optimierung?

Anfragen (10)

- Auswertungs-/Erklärungsmodell (Forts.)
 - Optimierter Operatorbaum für Anfrage **A** (siehe Folie 15)

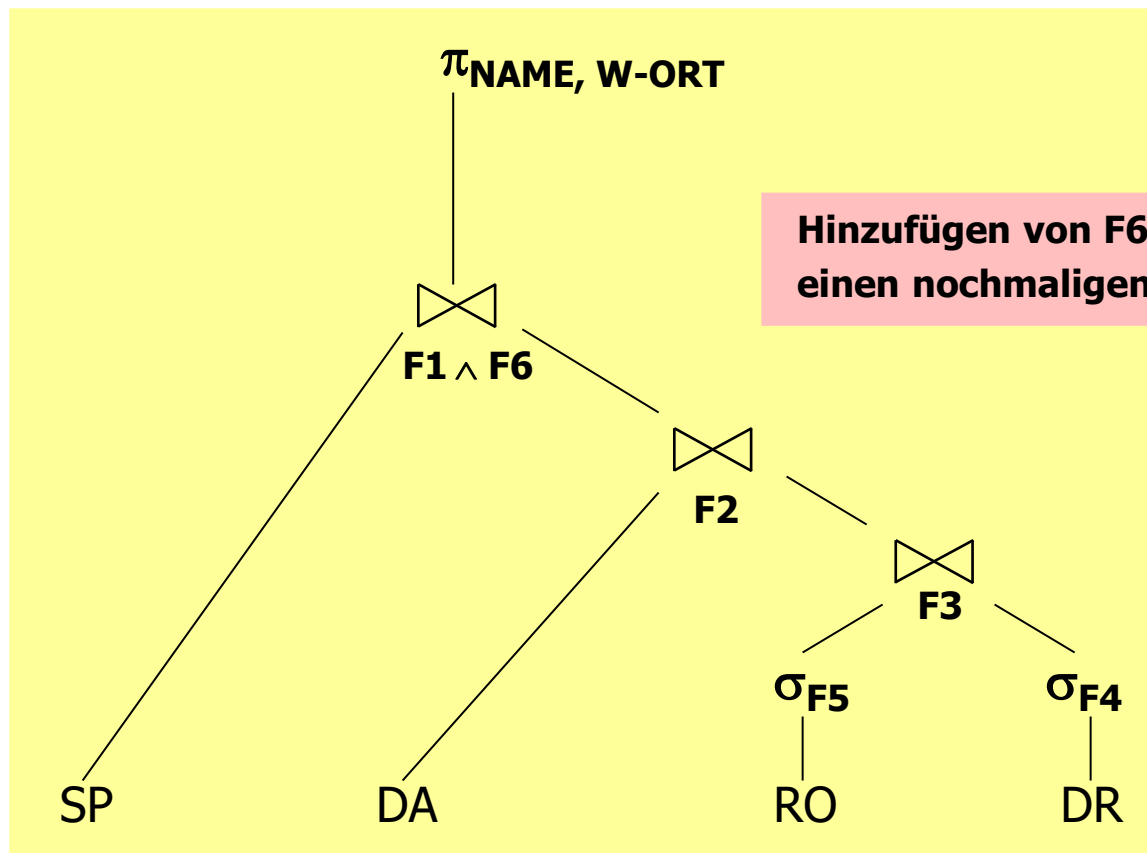


Heuristische Optimierungsregeln:

- 1. Führe Selektionen so früh wie möglich aus!**
- 2. Bestimme die Verbundreihenfolge so, dass die Anzahl und Größe der Zwischenobjekte minimiert wird!**

Anfragen (11)

- Auswertungs-/Erklärungsmodell (Forts.)
 - Optimierter Operatorbaum für Anfrage **A** (siehe Folie 15)



Hinzufügen von **F6** zur Join-Bedingung spart einen nochmaligen Scan des Join-Ergebnisses



Anfragen (12)

- Benutzer-spezifizierte Reihenfolge der Ausgabe

ORDER BY order-item-commalist [ASC | DESC]

Finde die Schauspieler, die an einem Ort wohnen, an dem sie gespielt haben, sortiert nach Name (aufsteigend), W-Ort (absteigend).

```
SELECT      S.NAME, S.W-ORT
FROM        SCHAUSPIELER S, DARSTELLER D
WHERE        S.PNR = D.PNR AND S.W-ORT = D.A-ORT
ORDER BY    S.NAME ASC, S.W-ORT DESC;
```

- Ohne Angabe der ORDER-BY-Klausel wird die Reihenfolge der Ausgabe durch das System bestimmt (*Optimierung der Auswertung*).



Anfragen (13)

- Aggregatfunktionen

```
aggregate-function-ref  
  ::= COUNT(*)  
    | {AVG | MAX | MIN | SUM | COUNT}  
      ([ALL | DISTINCT] scalar-exp)
```

- Standard-Funktionen: AVG, SUM, COUNT, MIN, MAX
 - Elimination von Duplikaten : DISTINCT
 - keine Elimination : ALL (Defaultwert)
 - Typverträglichkeit erforderlich
- *Bestimme das Durchschnittsgehalt der Schauspieler, die älter als 50 Jahre sind (GEHALT und ALTER seien Attribute von SCHAUSPIELER).*

```
SELECT  AVG(GEHALT) AS Durchschnittsgehalt  
FROM    SCHAUSPIELER  
WHERE   ALTER > 50;
```



Anfragen (14)

- Aggregatfunktionen (Forts.)
 - Auswertung
 - Aggregat-Funktion (AVG) wird angewendet auf einstellige Ergebnisliste (GEHALT)
 - keine Eliminierung von Duplikaten
 - Verwendung von arithmetischen Ausdrücken ist möglich:
AVG (GEHALT/12)
 - *An wievielen (unterschiedlichen) Orten wurden Dramen uraufgeführt (U-Ort)?*

```
SELECT    COUNT (DISTINCT U-ORT)  
FROM      DRAMA;
```

Anfragen (15)

- Aggregatfunktionen (Forts.)

- *An welchen Orten wurden mehr als zwei Dramen uraufgeführt ?*

```
SELECT  DISTINCT U-ORT
FROM    DRAMA
WHERE    COUNT(U-ORT)>2;
```




- keine geschachtelte Nutzung von Funktionsreferenzen !
- Aggregat-Funktionen die sich auf Werte verschiedener Tupel beziehen sind in der WHERE-Klausel unzulässig !

```
SELECT  DISTINCT U-ORT
FROM    DRAMA D
WHERE    2 <      (SELECT COUNT(*)
                   FROM    DRAMA X
                   WHERE    X.U-ORT = D.U-ORT);
```

Anfragen (16)

- Aggregatfunktionen (Forts.)
 - *Welches Drama wurde zuerst aufgeführt ?*

```
SELECT  TITEL, MIN(U-JAHR)
FROM    DRAMA;
```



```
SELECT  TITEL, U-JAHR
FROM    DRAMA
WHERE    U-JAHR = (SELECT MIN(U-JAHR)
                   FROM DRAMA);
```

oder

```
SELECT  TITEL, U-JAHR
FROM    DRAMA
WHERE    U-JAHR <= ALL (SELECT U-JAHR
                       FROM DRAMA);
```




Anfragen (17)

- Partitionierung

GROUP BY column-ref-commalist

Beispielschema: **PERS (PNR, NAME, GEHALT, ALTER, ANR)**
PRIMARY KEY (PNR)

Liste alle Abteilungen und das Durchschnittsgehalt ihrer Angestellten auf (Monatsgehalt).

```
SELECT      ANR, AVG(GEHALT)
FROM        PERS
GROUP BY    ANR;
```

Die GROUP-BY-Klausel wird immer zusammen mit einer Aggregat-Funktion benutzt. Die Aggregat-Funktion wird jeweils auf die Tupeln einer Gruppe angewendet. Die Ausgabe-Attribute müssen verträglich miteinander sein!



Anfragen (18)

- Partitionierung (Forts.)

HAVING cond-exp

Liste die Abteilungen zwischen K50 und K60 auf, bei denen das Durchschnittsalter ihrer Angestellten kleiner als 30 ist.

```
SELECT ANR
FROM PERS
WHERE ANR > K50 AND ANR < K60
GROUP BY ANR
HAVING AVG(ALTER) < 30;
```

Diskussion: Allgemeines Erklärungsmodell?



Anfragen (19)

- Hierarchische Beziehungen auf einer Relation

Beispielschema: **PERS (PNR, NAME, GEHALT, MNR)**
 PRIMARY KEY (PNR)
 FOREIGN KEY MNR REFERENCES PERS

Finde die Angestellten, die mehr als ihre (direkten) Manager verdienen (Ausgabe: NAME, GEHALT, NAME des Managers).

```
B:    SELECT X.NAME, X.GEHALT, Y.NAME  
         FROM   PERS X, PERS Y  
         WHERE X.MNR = Y.PNR AND  
                 X.GEHALT > Y.GEHALT;
```

Anfragen (20)

- Hierarchische Beziehungen auf einer Relation (Forts.)
 - Erklärung der Auswertung der Formel
 $X.MNR = Y.PNR$ **AND** $X.GEHALT > Y.GEHALT$
in Anfrage B (siehe vorhergehende Folie) am Beispiel

X = PERS	PNR	NAME	GEH.	MNR
	406	Abel	50 K	829
	123	Maier	60 K	829
	829	Müller	55 K	574
	574	May	50 K	999

Y = PERS	PNR	NAME	GEH.	MNR
	406	Abel	50 K	829
	123	Maier	60 K	829
	829	Müller	55 K	574
	574	May	50 K	999

AUSGABE	X.NAME	X.GEHALT	Y.NAME
	Maier	60 K	Müller
	Müller	55 K	May



Anfragen (21)

- **Auswertung von SELECT-Anweisungen – Erklärungsmodell**
 - Die auszuwertenden Relationen werden durch die FROM-Klausel bestimmt. Alias-Namen erlauben die mehrfache Verwendung derselben Relation.
 - Das Kartesische Produkt aller Relationen der FROM-Klausel wird gebildet.
 - Tupeln werden ausgewählt durch die WHERE-Klausel.
 - Qualifizierte Tupeln werden gemäß der GROUP-BY-Klausel in Gruppen eingeteilt.
 - Gruppen werden ausgewählt, wenn sie die HAVING-Klausel erfüllen. Prädikat in der HAVING-Klausel darf sich nur auf Gruppeneigenschaften beziehen (Attribute der GROUP-BY-Klausel oder Anwendung von Aggregat-Funktionen).
 - Die Ausgabe wird durch die Auswertung der SELECT-Klausel abgeleitet. Wurde eine GROUP-BY-Klausel spezifiziert, dürfen als SELECT-Elemente nur Ausdrücke aufgeführt werden, die für die gesamte Gruppe genau einen Wert ergeben (Attribute der GROUP-BY-Klausel oder Anwendung von Aggregat-Funktionen).
 - Die Ausgabereihenfolge wird gemäß der ORDER-BY-Klausel hergestellt. Wurde keine ORDER-BY-Klausel angegeben, ist die Ausgabereihenfolge systembestimmt (indeterministisch).

Anfragen (22)

- Erklärungsmodell – Beispiel

FROM R

WHERE B <= 50

R	A	B	C
	Rot	10	10
	Rot	20	10
	Gelb	10	50
	Rot	10	20
	Gelb	80	180
	Blau	10	10
	Blau	80	10
	Blau	20	200

R'	A	B	C
	Rot	10	10
	Rot	20	10
	Gelb	10	50
	Rot	10	20
	Gelb	80	180
	Blau	10	10
	Blau	80	10
	Blau	20	200

Anfragen (23)

- Erklärungsmodell – Beispiel (Forts.)

GROUP BY A

R''	A	B	C
	Rot	10	10
	Rot	20	10
	Rot	10	20
	Gelb	10	50
	Blau	10	10
	Blau	20	200

HAVING MAX(C) > 100

R''	A	B	C
	Rot	10	10
	Rot	20	10
	Rot	10	20
	Gelb	10	50
	Blau	10	10
	Blau	20	200

SELECT A, SUM(B), 12

R'''	A	SUM(B)	12
	Blau	30	12

ORDER BY A

R''''	A	SUM(B)	12
	Blau	30	12

Anfragen (24)

- Erklärungsmodell – weitere Beispiele

PERS	PNR	ANR	GEH	BONUS	ALTER
	0815	K45	80K	0	52
	4711	K45	30K	1	42
	1111	K45	50K	2	43
	1234	K56	40K	3	31
	7777	K56	80K	3	45
	0007	K56	20K	3	41

```
SELECT  ANR, SUM(GEH)
FROM    PERS
WHERE   BONUS <> 0
GROUP BY ANR
HAVING  (COUNT(*) > 1)
ORDER BY ANR DESC
```

ANR	SUM(GEH)
K56	140K
K45	80K

Anfragen (25)

- Erklärungsmodell – weitere Beispiele

PERS	PNR	ANR	GEH	BONUS	ALTER
	0815	K45	80K	0	52
	4711	K45	30K	1	42
	1111	K45	50K	2	43
	1234	K56	40K	3	31
	7777	K56	80K	3	45
	0007	K56	20K	3	41

```
SELECT      ANR, SUM(GEH)
FROM        PERS
WHERE       BONUS <> 0
GROUP BY   ANR
HAVING     (COUNT(DISTINCT BONUS) > 1)
ORDER BY   ANR DESC
```

ANR	SUM(GEH)
K45	80K

Anfragen (26)

- Erklärungsmodell – weitere Beispiele

PERS	PNR	ANR	GEH	BONUS	ALTER
	0815	K45	80K	0	52
	4711	K45	30K	1	42
	1111	K45	50K	2	43
	1234	K56	40K	3	31
	7777	K56	80K	3	45
	0007	K56	20K	3	41

Die Summe der Gehälter pro Abteilung, in der mindestens ein Mitarbeiter 40 Jahre oder älter ist, soll berechnet werden:

```
SELECT ANR, SUM(GEHALT)
FROM PERS
WHERE ALTER >= 40
GROUP BY ANR
HAVING (COUNT(*) >= 1)
```



ANR	SUM(GEH)
K45	160K
K56	100K

Anfragen (27)

- Erklärungsmodell – weitere Beispiele

PERS	PNR	ANR	GEH	BONUS	ALTER
	0815	K45	80K	0	52
	4711	K45	30K	1	42
	1111	K45	50K	2	43
	1234	K56	40K	3	31
	7777	K56	80K	3	45
	0007	K56	20K	3	41

Die Summe der Gehälter pro Abteilung, in der mindestens ein Mitarbeiter 40 Jahre oder älter ist, soll berechnet werden:

```
SELECT    ANR, SUM(GEH)
FROM      PERS
GROUP BY  ANR
HAVING    (MAX(ALTER) >= 40)
```

ANR	SUM(GEH)
K45	160K
K56	140K

Anfragen (28)

- Suchbedingungen
 - Sammlung (elementarer) Prädikate
 - Verknüpfung mit **AND, OR, NOT**
 - Ggf. Bestimmung der Auswertungsreihenfolge durch Klammerung
- Nicht-quantifizierte Prädikate
 - Vergleichsprädikate
 - BETWEEN-Prädikate
 - IN-Prädikate
 - Ähnlichkeitssuche
 - Prädikate über Nullwerten
- Quantifizierte Prädikate mit Hilfe von ALL, ANY, EXISTS
- Weitere Prädikate
 - UNIQUE
 - ...

comparison-cond	::=	row-constructor ⊕ row-constructor
row-creator	::=	scalar-exp (scalar-exp-commalist) (table-exp)

Beispiel: GEHALT **BETWEEN** 80K **AND** 100K

Anfragen (29)

- **IN-Prädikate**

row-constr [NOT] IN (table-exp)

- **x IN (a, b, . . . , z)** \Leftrightarrow **x = a OR x = b . . . OR x = z**
- **row-constr IN (table-exp)** \Leftrightarrow **row-constr = ANY (table-exp)**
- **x NOT IN erg** \Leftrightarrow **NOT (x IN erg)**

- **Beispiel:**

Finde die Namen der Schauspieler, die den Faust gespielt haben.

```
SELECT S.NAME
FROM   SCHAUPIELER S
WHERE  'Faust' IN
      (SELECT D.FIGUR
       FROM   DARSTELLER D
       WHERE  D.PNR = S.PNR)
```

```
SELECT S.NAME
FROM   SCHAUPIELER S
WHERE  S.PNR IN
      (SELECT D.PNR
       FROM   DARSTELLER D
       WHERE  D.FIGUR = 'Faust')
```

```
SELECT S.NAME
FROM   SCHAUPIELER S,
      DARSTELLER D
WHERE  S.PNR = D.PNR AND
      D.FIGUR = 'Faust'
```



Anfragen (30)

- Ähnlichkeitssuche

- Unterstützung der Suche nach Objekten, von denen nur Teile des Inhalts bekannt sind oder die einem vorgegebenen Suchkriterium möglichst nahe kommen.
- Klassen
 - Syntaktische Ähnlichkeitssuche (siehe LIKE-Prädikat)
 - Phonetische Ähnlichkeit (spezielle DBS)
 - Semantische Ähnlichkeit (benutzerdefinierte Funktionen)

char-string-exp [NOT] LIKE char-string-exp
 [ESCAPE char-string-exp]

- **Unscharfe Suche:** LIKE-Prädikat vergleicht einen Datenwert mit einem „Muster“ bzw. einer „Maske“
- Das LIKE-Prädikat ist TRUE, wenn der entsprechende Datenwert der Maske mit zulässigen Substitutionen von Zeichen für % und _ entspricht



Anfragen (31)

- Ähnlichkeitssuche (Forts.)
 - LIKE-Prädikat (Forts.) – Beispiele
 - **NAME LIKE '%SCHMI%'** wird z. B. erfüllt von
'H.-W. SCHMITT', 'SCHMITT, H.-W.', 'BAUSCHMIED', 'SCHMITZ'
 - **ANR LIKE '_7%'** wird erfüllt von Abteilungen mit einer 7 als zweitem Zeichen
 - **NAME NOT LIKE '%-%'** wird erfüllt von allen Namen ohne Bindestrich
 - Suche nach '%' und '_' durch Voranstellen eines Escape-Zeichens möglich:
STRING LIKE '%_%' ESCAPE '\'
wird erfüllt von STRING-Werten mit Unterstrich
 - **SIMILAR-Prädikat** in SQL:1999
 - erlaubt die Nutzung von regulären Ausdrücken zum Maskenaufbau
 - Beispiel: **NAME SIMILAR TO** '(SQL-(86 | 89 | 92 | 99)) | (SQL(1 | 2 | 3))'



Anfragen (32)

- Quantifizierung

- ALL-or-ANY-Prädikate

$\text{row-constr } \Theta \{ \text{ALL} \mid \text{ANY} \mid \text{SOME} \} (\text{table-exp})$

- Θ **ALL**: Prädikat wird zu „true“ ausgewertet, wenn der Θ -Vergleich für alle Ergebniswerte von table-exp „true“ ist
 - Θ **ANY** / Θ **SOME**: analog, wenn der Θ -Vergleich für einen Ergebniswert „true“ ist

- Existenztests

$[\text{NOT}] \text{ EXISTS } (\text{table-exp})$

- Das Prädikat wird zu „false“ ausgewertet, wenn table-exp auf die leere Menge führt, sonst zu „true“
 - Im EXISTS-Kontext darf table-exp mit (SELECT * ...) spezifiziert werden (Normalfall)

Anfragen (33)

- Quantifizierung (Forts.)

- Semantik

- $x \ominus \text{ANY} (\text{SELECT } y \text{ FROM } T \text{ WHERE } p) \Leftrightarrow$
EXISTS (SELECT * FROM T WHERE (p) AND $x \ominus T.y$)
 - $x \ominus \text{ALL} (\text{SELECT } y \text{ FROM } T \text{ WHERE } p) \Leftrightarrow$
NOT EXISTS (SELECT * FROM T WHERE (p) AND NOT ($x \ominus T.y$))

- Beispiele

- *Finde die Manager, die mehr verdienen als alle ihre direkten Untergebenen*

```
SELECT      M.PNR
FROM        PERS M
WHERE        M.GEHALT > ALL (SELECT P.GEHALT
                                FROM   PERS P
                                WHERE  P.MNR = M.PNR)
```



Anfragen (34)

- Quantifizierung (Forts.)
 - Beispiele (Forts.)
 - *Finde die Namen der Schauspieler, die mindestens einmal gespielt haben (... nie gespielt haben)*

```
SELECT      SP.NAME
FROM        SCHAUSPIELER SP
WHERE        (NOT) EXISTS      (SELECT *
                                FROM   DARSTELLER DA
                                WHERE  DA.PNR = SP.PNR)
```



Anfragen (35)

- Quantifizierung (Forts.)
 - Beispiele (Forts.)
 - *Finde die Namen aller Schauspieler, die alle Rollen gespielt haben.*

```
SELECT S.NAME
FROM   SCHAUSPIELER S
        WHERE NOT EXISTS
            (SELECT *
             FROM   ROLLE R
             WHERE NOT EXISTS
                 (SELECT *
                  FROM   DARSTELLER D
                  WHERE D.PNR = S.PNR
                      AND D.FIGUR = R.FIGUR))
```

Andere Formulierung: *Finde die Namen der Schauspieler, so dass keine Rolle „existiert“, die sie nicht gespielt haben.*

Anfragen (36)

- Prädikate über Nullwerten
 - **Attributspezifikation:** Es kann für jedes Attribut festgelegt werden, ob NULL-Werte zugelassen sind oder nicht
 - **Verschiedene Bedeutungen von Nullwerten:**
 - Datenwert ist momentan nicht bekannt
 - Attributwert existiert nicht für ein Tupel
 - **Auswertung von boolschen Ausdrücken anhand 3-wertiger Logik**

NOT		AND	T	F	?	OR	T	F	?
T	F	T	T	F	?	T	T	T	T
F	T	F	F	F	F	F	T	F	?
?	?	?	?	F	?	?	T	?	?

- Elementares Prädikat wird zu **UNKNOWN (?)** ausgewertet, falls Nullwert vorliegt (z.B. NULL = NULL => UNKNOWN)
- nach vollständiger Auswertung einer WHERE-Klausel wird das Ergebnis ? wie FALSE behandelt

Anfragen (37)

- Prädikate über Nullwerten (Forts.)

- Beispiele

PERS	PNR	ANR	GEH	PROV
	0815	K45	80K	-
	4711	K45	30K	50K
	1111	K45	20K	-
	1234	K56	-	-
	7777	K56	80K	100K

- GEH = PROV: 0815: ?, 1111: ?, 1234: ?
- GEH > 70K AND PROV > 50K: 0815: ?, 1111: F, 1234: ?
- GEH > 70K OR PROV > 50K: 0815: T, 1111: ?, 1234: ?

- Test auf Nullwert

row-constr IS [NOT] NULL

- Beispiel:

```
SELECT PNR, PNAME
FROM PERS
WHERE GEH IS NULL;
```



Anfragen (38)

- Häufige genutzte Alternativen zu Nullwerten
 - Defaultwerte (Risiko: Eingabe eines falschen Wertes)
 - Leere Zeichenkette, „None“, „Unbekannt“ oder spezielle Symbole wie „-“ oder „#“ bei Stringwerten
 - „- 1“ bei Attributen mit positiven Zahlen wie z.B. Alter, Gehalt
 - Untypische Werte wie „01.01.0000“ bei Datumsangaben
- Generelle Aspekte:
 - Welche Auswirkungen haben solche Alternativwerte auf Wertvergleiche, Aggregationen oder Statistikanalysen?
 - Inkonsequente Nutzung führt zu Inkonsistenzen (Konsequente Nutzung kann aber nur begrenzt vom System kontrolliert werden)
 - Bis auf Defaultwerte nicht auf Fremdschlüsselattribute anwendbar

Anfragen (39)

- Alternative Modellierung von optionalen Eigenschaften
 - Verwendung eines zusätzlichen Booleschen Attributes pro optionaler Eigenschaft
 - Belegung mit 1 wenn Eigenschaft existiert
 - Belegung mit 0 wenn Eigenschaft nicht existiert
 - Belegung mit Nullwert falls Existenz der Eigenschaft unbekannt ist
- Anm.: In den beiden letzten Fällen beinhaltet das eigentl. Attribut auch einen Nullwert
- Vorteil:
 - Löst Konflikt zwischen den Nullwertsemantiken „unbekannt“ und „nicht existent“
 - Nachteil:
 - Erhöht Komplexität von Schema und Anfragen sowie den Speicherbedarf

PERS	PNR	ABT	ANR	GEH	PROV
	0815	-	-	80K	-
	4711	1	K45	30K	50K
	1111	0	-	20K	-
	1234	1	K56	-	-
	7777	1	-	80K	100K

```
SELECT *  
FROM PERS  
WHERE ABT = 0
```

Personen ohne
Abteilung

```
SELECT *  
FROM PERS  
WHERE ABT = 1 AND ANR IS NULL
```

Personen mit
unbekannter Abteilung

Anfragen (40)

- Weiteres zu Nullwerten

- Eine arithmetische Operation (+, -, *, /) mit einem NULL-Wert führt zu einen NULL-Wert

- **SELECT** PNR, GEH + PROV
FROM PERS:
0815: ?,
4711: 80K,
...

PERS	PNR	ANR	GEH	PROV
	0815	K45	80K	-
	4711	K45	30K	50K
	1111	K45	20K	-
	1234	K56	-	-
	7777	K56	80K	100K

- **Verbund**

- Tupel mit NULL-Werten im Verbundattribut nehmen **nicht** am Verbund teil

- **Achtung**

- Aggregatfunktionen ignorieren Nullwerte
- Im allgemeinen gilt daher: **AVG(GEH) <> SUM(GEH) / COUNT(PNR)**

Anfragen (41)

- **Vermeintliche Anfrage-Äquivalenzen**
- **Beispiel:** *Finde alle Abteilungen deren Leiter nicht mehr als 50k verdient.*

ABT	<u>ANR</u>	Name	Leiter
	K51	Planung	0815
	K53	Einkauf	-
	K55	Vertrieb	1111

PERS	PNR	ANR	GEH	PROV
	0815	K45	80K	-
	4711	K45	30K	50K
	1111	K45	20K	-
	1234	K56	-	-
	7777	K56	80K	100K

```
SELECT *  
FROM ABT A  
WHERE A.Leiter NOT IN  
      (SELECT P.PNR  
       FROM PERS P  
       WHERE P.Gehalt > 50k)
```

K53 fehlt!



```
SELECT *  
FROM ABT A  
WHERE NOT EXISTS  
      (SELECT P.PNR  
       FROM PERS P  
       WHERE P.Gehalt > 50k  
       AND P.PNR = A.Leiter)
```

K53 ist
im Ergebnis!

Anfragen (42)

- **Vermeintliche Anfrage-Äquivalenzen**
- **Beispiel:** *Finde die Person mit dem maximalen Gehalt.*

PERS	PNR	ANR	GEH	PROV
	0815	K45	80K	-
	4711	K45	30K	50K
	1111	K45	20K	-
	1234	K56	-	-
	7777	K56	80K	100K

```
SELECT  *
FROM    PERS p1
WHERE    NOT EXISTS (SELECT *
                       FROM    PERS p2
                       WHERE    p2.GEH > p1.GEH)
```

1234 ist
im Ergebnis!

```
SELECT  *
FROM    PERS p1
WHERE    p1.GEH =
           (SELECT MAX(p2.GEH)
           FROM    PERS p2)
```

1234 fehlt!



Anfragen (43)

- **Vermeintliche Anfrage-Äquivalenzen**
- **Beispiel:** *Finde die Person mit dem maximalen Gehalt.*

PERS	PNR	ANR	GEH	PROV
	0815	K45	80K	-
	4711	K45	30K	50K
	1111	K45	20K	-
	1234	K56	-	-
	7777	K56	80K	100K

```
SELECT  *  
FROM    PERS p1  
WHERE    p1.GEH IS NOT NULL  
AND      NOT EXISTS (SELECT *  
              FROM    PERS p2  
              WHERE    p2.GEH > p1.GEH)
```

1234 fehlt!

=

```
SELECT  *  
FROM    PERS p1  
WHERE    p1.GEH =  
          (SELECT MAX(p2.GEH)  
          FROM    PERS p2)
```

1234 fehlt!

Anfragen (44)

- **Vermeintliche Anfrage-Äquivalenzen**
- **Beispiel:** *Finde alle Personen aus Abteilung K56 die mehr als 40k verdienen.*

PERS	PNR	ANR	GEH	PROV
	0815	K45	80K	-
	4711	K45	30K	50K
	1111	K45	20K	-
	1234	K56	-	-
	7777	K56	80K	100K

```
SELECT *  
FROM PERS  
WHERE ANR = K56  
AND GEH > 40k
```

1234 fehlt!



```
SELECT *  
FROM PERS  
WHERE ANR = K56  
MINUS  
SELECT *  
FROM PERS  
WHERE GEH <= 40k
```

1234 ist
im Ergebnis!

Anfragen (45)

- **Vermeintliche Tautologien**
- **Beispiel:** *Finde alle Personen die weniger, gleich oder mehr als 40k verdienen.*

PERS	PNR	ANR	GEH	PROV
	0815	K45	80K	-
	4711	K45	30K	50K
	1111	K45	20K	-
	1234	K56	-	-
	7777	K56	80K	100K

```
SELECT    *  
FROM      PERS  
WHERE     GEH <= 40k  
OR        GEH > 40k
```

1234 fehlt!

```
SELECT    *  
FROM      PERS  
WHERE     GEH <= 40k  
OR        GEH > 40k  
OR        GEH IS NULL
```

1234 ist
im Ergebnis!

Anfragen (46)

- **Vermeintliche Tautologien**
- **Beispiel:** *Finde alle Personen die entweder in Abteilung K45 oder nicht in Abteilung K45 arbeiten*

PERS	PNR	ANR	GEH	PROV
	0815	K45	80K	-
	4711	K45	30K	50K
	1111	-	20K	-
	1234	K56	-	-
	7777	K56	80K	100K

```
SELECT *  
FROM PERS  
WHERE ANR = K45  
OR NOT (ANR = K45)
```

1111 fehlt!

```
SELECT *  
FROM PERS  
WHERE ANR = K45  
OR NOT (ANR = K45)  
OR ANR IS NULL
```

1111 ist
im Ergebnis!