



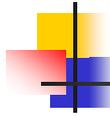
Die Standardsprache SQL

Inhalt

- Grundlagen
- Mengenorientierte Anfragen (Retrieval)
- Möglichkeiten der Datenmanipulation
- Möglichkeiten der Datendefinition
- Beziehungen und referentielle Integrität
- Schemaevolution
- Indexierung
- Sichten

Grundlagen (3)

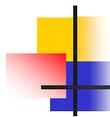
- **Sprachentwicklung von SQL**
 - SQL wurde „**de facto**“-**Standard** in der relationalen Welt (1986 von ANSI, 1987 von ISO akzeptiert)
 - **Weiterentwicklung des Standards in drei Stufen**
 - **SQL2 (1992)**
 - **(SQL3) SQL:1999**
 - **SQL:2003**
- **Mächtigkeit von SQL**
 - Auswahlvermögen äquivalent dem Relationenkalkül und der Relationenalgebra
 - **relational vollständig**



Grundlagen (4)

- Es existieren folgende Standards:
 - SQL-86: erster Standard durch ANSI (ANSI X3.135-1986) und ISO (ISO 9075-1987)
 - SQL-89: Erweiterung um die Möglichkeit zur Definition referentieller Integritätsbedingungen (ANSI X3.135-1989 bzw. X3.168-1989, ISO 9075:1989)
 - **SQL-92** bzw. SQL-2: Vereinheitlichung der bisherigen Lösungen durch Festlegung von drei Sprachschichten (ANSI X3.135-1992, ISO/IEC 9075:1992):
 - Die **Entry SQL-Schicht** enthält die grundlegendsten Anweisungen zum **Anlegen von Tabellen** oder zur **Datenmanipulation**. Der einzig gültige Datentyp ist hier die Zeichenkette (CHAR). Diese Schicht muß von allen relationalen Datenbanksystemen unterstützt werden.
 - In der **Intermediate SQL-Schicht** werden u.a. weitere Anweisungen zur **Datendefinition**, **Funktionen**, **Verbundoperatoren** und Mengenoperatoren beschrieben. Es sind weitere Datentypen definiert.
 - Die **Full SQL-Schicht** enthält alle SQL-Anweisungen und weitere Funktionen z.B. für die Verbindungsverwaltung.
 - SQL-99 bzw. SQL-3: Integration von objektorientierten Konzepten und Aufteilung des Standards in mehrere Teile, die einzeln weiterentwickelt werden sollen. (ISO/IEC 9075:1999)
 - SQL-2003: Aktueller Standard, der u.a. die Verarbeitung von XML-Daten beschreibt. (ISO/IEC 9075:2003)

Rest des Kapitels bezieht sich auf SQL-92

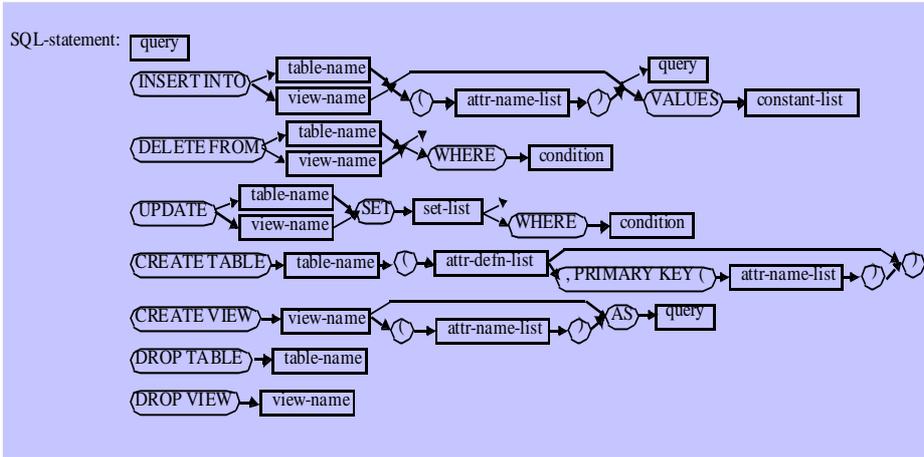


Grundlagen (5)

- **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 (6)

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

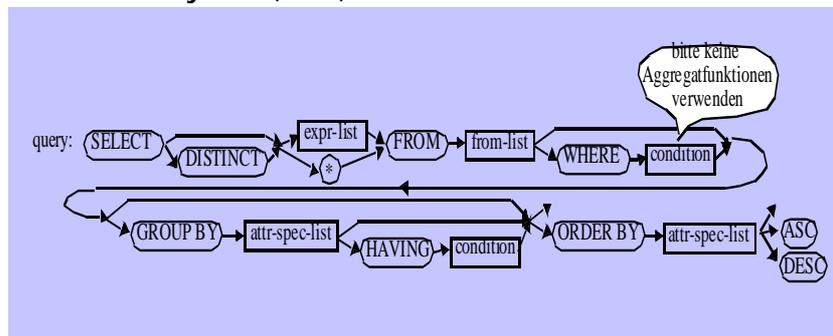


N. Ritter, HMS, 13.09.2007

5

Grundlagen (7)

- **SQL92-Syntax** (Forts.)

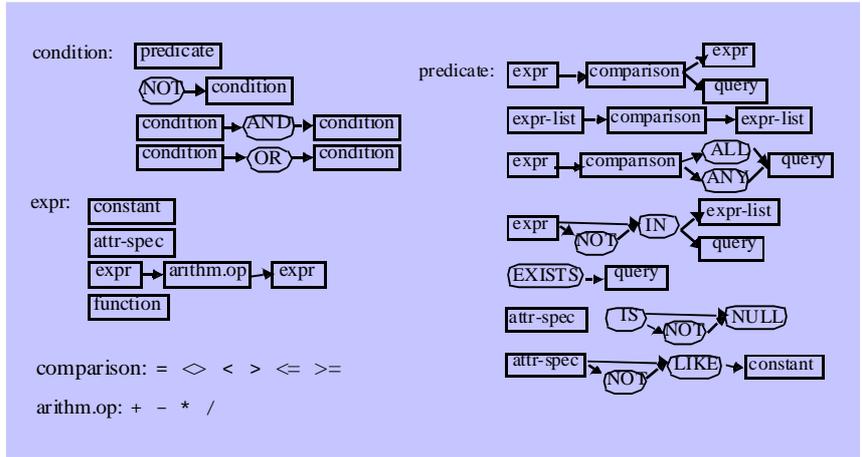


N. Ritter, HMS, 13.09.2007

6

Grundlagen (8)

SQL92-Syntax (Forts.)

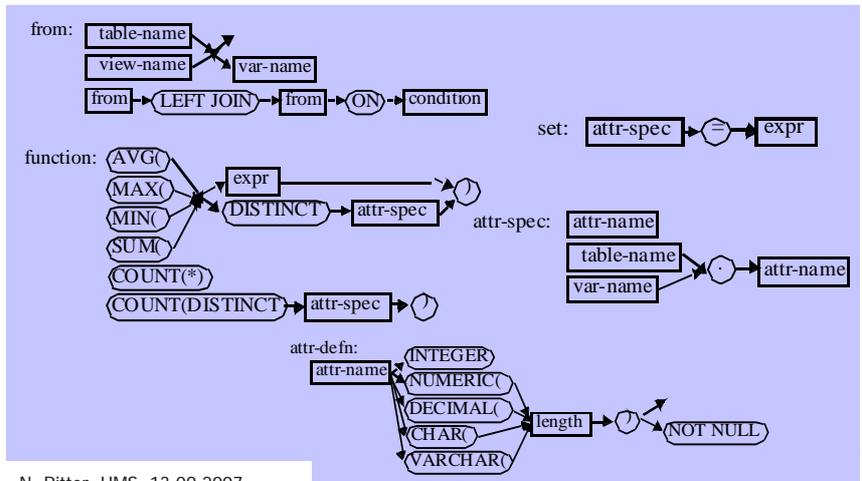


N. Ritter, HMS, 13.09.2007

7

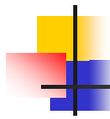
Grundlagen (9)

SQL92-Syntax (Forts.)



N. Ritter, HMS, 13.09.2007

8



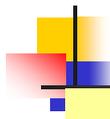
Anfragen (1)



- SELECT-Anweisung

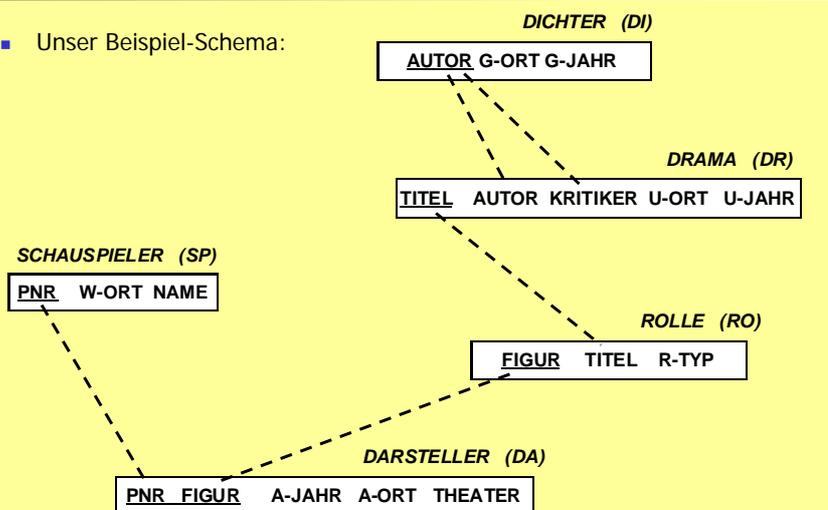
```
select-exp
 ::= SELECT [ALL | DISTINCT] select-item-commalist
    FROM table-ref-commalist
    [WHERE cond-exp]
    [GROUP BY column-ref-commalist]
    [HAVING cond-exp]
```

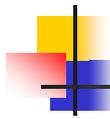
- Grob:
 - SELECT * :** Ausgabe ‚ganzer‘ Tupel
 - FROM-Klausel:** spezifiziert zu verarbeitende Relation bzw.
 - WHERE-Klausel:** Sammlung (elementarer) Prädikate der Form $A_i \Theta a_i$ oder $A_i \Theta A_j$ ($\Theta \in \{ =, <>, <, \leq, >, \geq \}$), die mit *AND* und *OR* verknüpft sein können



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

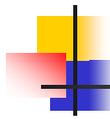
- A_i **IN** (a_1, a_j, a_k) explizite Mengendefinition
- A_i **IN** (**SELECT** . . .) implizite Mengendefinition

■ Beispiel:

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

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

- Duplikateliminierung
 - Default: 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)



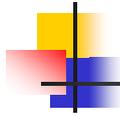
- Symmetrische Abbildung
Finde die Figuren und ihre Autoren, die in Dramen von Schiller oder Goethe vorkommen.

```

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



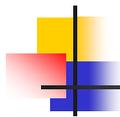
Anfragen (7)

- 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.AUTOR = D.AUTOR
         AND   D.KRITIKER = B.AUTOR
         AND   A.G-ORT = B.G-ORT;
```

Diskussion: Welche Rolle spielen die Bedingungen $A.AUTOR = D.AUTOR$ und $D.KRITIKER = B.AUTOR$ in der erhaltenen Lösung?



Anfragen (8)

- Symmetrische Abbildung (Forts.)

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

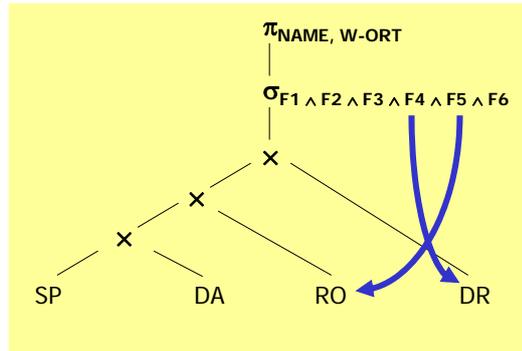
```
A: SELECT S.NAME, S.W-ORT
     FROM   SCHAUSPIELER S, DARSTELLER D, ROLLE R, DRAMA A
     WHERE  S.PNR = D.PNR
           AND   D.FIGUR = R.FIGUR
           AND   R.TITEL = A.TITEL
           AND   A.U-ORT = 'Weimar'
           AND   R.R-TYP = 'Held'
           AND   D.A-ORT = S.W-ORT;
```

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 17)



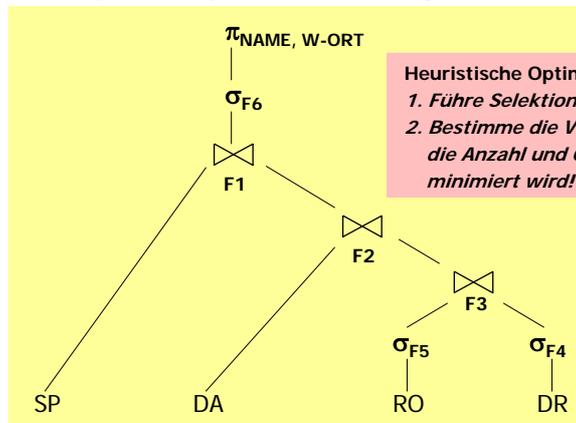
Optimierung?

N. Ritter, HMS, 13.09.2007

17

Anfragen (10)

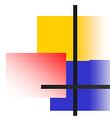
- Auswertungs-/Erklärungsmodell (Forts.)
 - Optimierter Operatorbaum für Anfrage **A** (siehe Folie 17)



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

N. Ritter, HMS, 13.09.2007

18



Anfragen (11)



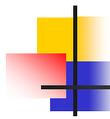
- Benutzer-spezifizierte Reihenfolge der Ausgabe

```
ORDER BY order-item-commalist
```

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 (12)



- 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 SP).

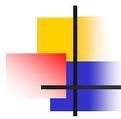
```
SELECT AVG(GEHALT) AS Durchschnittsgehalt
FROM   SCHAUSPIELER
WHERE  ALTER > 50;
```



Anfragen (13)

- 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 Orten wurden Dramen uraufgeführt (U-Ort)?*

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



Anfragen (14)

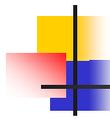
- Aggregatfunktionen (Forts.)
 - *An welchen Orten wurden mehr als zwei Dramen uraufgeführt ?*

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



- keine geschachtelte Nutzung von Funktionsreferenzen !
- Aggregat-Funktionen in WHERE-Klausel unzulässig !

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



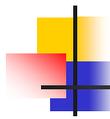
Anfragen (15)

- 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 X);
```



Anfragen (16)

- Partitionierung

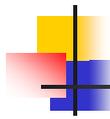
```
GROUP BY column-ref-commalist
```

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 (17)



- 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?



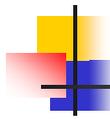
Anfragen (18)

- 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 (19)

- Hierarchische Beziehungen auf einer Relation (Forts.)

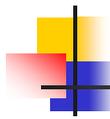
- Erklärung der Auswertung der Formel
 $X.MNR = Y.PNR \text{ AND } X.GEHALT > Y.GEHALT$
in Anfrage B (siehe vorhergehende Folie) am Beispiel

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

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

N. Ritter, HMS, 13.09.2007

27



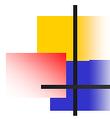
Anfragen (20)

- **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).

N. Ritter, HMS, 13.09.2007

28



Anfragen (21)

- Erklärungsmodell – Beispiel

FROM R

| 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 |

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 |

N. Ritter, HMS, 13.09.2007

29



Anfragen (22)

- 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 |

N. Ritter, HMS, 13.09.2007

30

Anfragen (23)

- 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 |

N. Ritter, HMS, 13.09.2007

31

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(DISTINCT BONUS) > 1)
ORDER BY ANR DESC

```

| ANR | SUM(GEH) |
|-----|----------|
| K45 | 80K |

N. Ritter, HMS, 13.09.2007

32

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 | |

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 |

N. Ritter, HMS, 13.09.2007

33

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 (GEH) | | |
| FROM | PERS | | |
| GROUP BY | ANR | | |
| HAVING | (MAX (ALTER) >= 40) | | |

| ANR | SUM(GEH) |
|-----|----------|
| K45 | 160K |
| K56 | 140K |

N. Ritter, HMS, 13.09.2007

34

Anfragen (27)

- 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-Prädikat
 - ...

```
comparison-cond ::= row-creator
                  ⅀ row-creator
row-creator      ::= scalar-exp |
                  (scalar-exp-commalist) |
                  (table-exp)
```

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

Anfragen (28)

- **IN-Prädikate**

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

 - **x IN (a, b, ..., z)** ⇔ **x = a OR x = b ... OR x = z**
 - **row-constr IN (table-exp)** ⇔ **row-constr = ANY (table-exp)**
 - **x NOT IN erg** ⇔ **NOT (x IN erg)**
- **Beispiel:**
Finde die Namen der Schauspieler, die den Faust gespielt haben.

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

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

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



Anfragen (29)



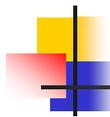
- Ähnlichkeitssuche
 - Unterstützung der Suche nach Objekten, von denen nur Teile des Inhalts bekannt sind oder die einem vorgegebenen Suchkriterium möglichst nahe kommen.
 - Klassen
 - Syntaktische Ähnlichkeitssuche (siehe LIKE-Prädikat)
 - Phonetische Ähnlichkeit (spezielle DBS)
 - Semantische Ähnlichkeit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

`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

N. Ritter, HMS, 13.09.2007

37

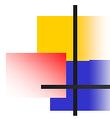


Anfragen (30)

- 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))'

N. Ritter, HMS, 13.09.2007

38



Anfragen (31)



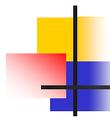
- Prädikate über Nullwerten
 - **Attributspezifikation:** Es kann für jedes Attribut festgelegt werden, ob NULL-Werte zugelassen sind oder nicht
 - **Verschiedene Bedeutungen von Nullwerten:**
 - Datenwert ist momentan nicht bekannt
 - Attributwert existiert nicht für ein Tupel
 - **Auswertung von booleschen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

| NOT | | AND | | | OR | | |
|-----|---|-----|---|---|----|---|---|
| T | F | T | F | ? | T | F | ? |
| T | F | T | T | F | T | T | T |
| F | T | F | F | F | F | T | F |
| ? | ? | ? | ? | F | ? | T | ? |

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

N. Ritter, HMS, 13.09.2007

39



Anfragen (32)

- 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

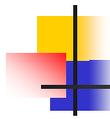
```
row-constr IS [NOT] NULL
```

- Beispiel:

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

N. Ritter, HMS, 13.09.2007

40



Anfragen (33)



- Weiteres zu Nullwerten
 - Eine arithmetische Operation (+, -, *, /) mit einem NULL-Wert führt auf 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**
 - Im allgemeinen ist **AVG (GEH) <> SUM (GEH) / COUNT (PNR)**



Anfragen (34)



- Quantifizierung
 - ALL-or-ANY-Prädikate
 - row-constr Θ { ALL | ANY | SOME } (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
 - [NOT] EXISTS (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 (35)

- Quantifizierung (Forts.)

- Semantik

- $x \in \text{ANY}(\text{SELECT } y \text{ FROM } T \text{ WHERE } p) \Leftrightarrow \text{EXISTS}(\text{SELECT } * \text{ FROM } T \text{ WHERE } (p) \text{ AND } x \in T.y)$

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

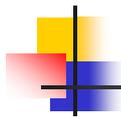
- Beispiele

- 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)
```

N. Ritter, HMS, 13.09.2007

43



Anfragen (36)

- Quantifizierung (Forts.)

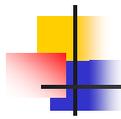
- Beispiele (Forts.)

- 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)
```

N. Ritter, HMS, 13.09.2007

44

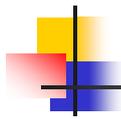


Anfragen (37)

- 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.*



Datenmanipulation (1)

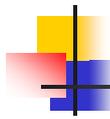


- Einfügen von Tupeln
- ```
INSERT INTO table [(column-commalist)]
 { VALUES row-constr.-commalist |
 table-exp |
 DEFAULT VALUES }
```

- Beispiel:  
*Füge den Schauspieler Garfield mit der PNR 4711 ein.*

```
INSERT INTO SP (PNR, NAME, W-ORT)
VALUES (4711, „Garfield“, DEFAULT);
```

- Anmerkungen (zu satzweises Einfügen)
  - Alle nicht angesprochenen Attribute erhalten Nullwerte.
  - Falls alle Werte in der richtigen Reihenfolge versorgt werden, kann die Attributliste weggelassen werden.
  - Mengenorientiertes Einfügen ist möglich, wenn die einzufügenden Tupel aus einer anderen Relation mit Hilfe einer SELECT-Anweisung ausgewählt werden können.



## Datenmanipulation (2)



- Einfügen von Tupeln (Forts.)
  - Beispiel:  
*Füge die Schauspieler aus KL in die Relation TEMP ein.*

```
INSERT INTO TEMP
 (SELECT *
 FROM SP
 WHERE W-ORT = „KL“);
```

- Anmerkungen (zu mengenorientiertes Einfügen)
  - Im Beispiel sei eine (leere) Relation **TEMP** sei vorhanden. Die Datentypen ihrer Attribute müssen kompatibel zu den Datentypen der ausgewählten Attribute sein.
  - Ein mengenorientiertes Einfügen wählt die spezifizierte Tupelmengung aus und kopiert sie in die Zielrelation.
  - Die kopierten Tupel sind unabhängig von ihren Ursprungstupeln.



## Datenmanipulation (3)



- Löschen von Tupeln mit Hilfe von Suchklauseln

```
searched-delete
 ::= DELETE FROM table [WHERE cond-exp]
```

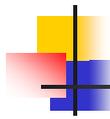
- Aufbau der WHERE-Klausel entspricht dem der SELECT-Anweisung
- Beispiele

*Lösche den Schauspieler mit der PNR 4711.*

```
DELETE FROM SCHAUSPIELER
 WHERE PNR = 4711;
```

*Lösche alle Schauspieler, die nie gespielt haben.*

```
DELETE FROM SCHAUSPIELER S
 WHERE NOT EXISTS
 (SELECT *
 FROM DARSTELLER D
 WHERE D.PNR = S.PNR);
```



## Datenmanipulation (4)



- Ändern von Tupeln mit Hilfe von Suchklauseln

```
searched-update
 ::= UPDATE table SET update-assignment-commalist
 [WHERE cond-exp]
```

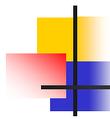
- Beispiel

*Gib den Schauspielern, die am Pfalztheater spielen, eine Gehaltserhöhung von 5% (Annahme: GEHALT in Schauspielern).*

```
UPDATE SCHAUSPIELER S
SET S.GEHALT = S.GEHALT * 1.05
WHERE EXISTS (SELECT *
FROM DARSTELLER D
WHERE D.PNR = S.PNR AND D.THEATER = 'Pfalz');
```

- **Einschränkung:**

Innerhalb der WHERE-Klausel in einer Lösch- oder Änderungsanweisung darf die Zielrelation in einer FROM-Klausel nicht referenziert werden.



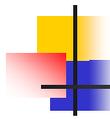
## Datendefinition (4)

- Definition von Schemata

```
CREATE SCHEMA [schema] [AUTHORIZATION user]
 [DEFAULT CHARACTER SET char-set]
 [schema-element-list]
```

- Jedes Schema ist einem Benutzer (*user*) zugeordnet, z.B. DBA
- Schema erhält Benutzernamen, falls keine explizite Namensangabe erfolgt
- Definition aller Definitionsbereiche, Basisrelationen, Sichten (*Views*), Integritätsbedingungen und Zugriffsrechte
- Beispiel

```
CREATE SCHEMA Beispiel-DB AUTHORIZATION DB-Admin
```



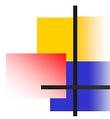
## Datendefinition (5)

### ■ Datentypen

- CHARACTER [ ( length ) ] (Abkürzung: CHAR)
- CHARACTER VARYING [ ( length ) ] (Abkürzung: VARCHAR)
- ...
- NUMERIC [ ( precision [ , scale ] ) ]
- DECIMAL [ ( precision [ , scale ] ) ] (Abkürzung: DEC)
- INTEGER (Abkürzung: INT)
- REAL
- ...
- DATE
- TIME
- ...

N. Ritter, HMS, 13.09.2007

51



## Datendefinition (6)



### ■ Definition von Domains

```
CREATE DOMAIN domain [AS] data type
 [DEFAULT { literal | niladic-function-ref | NULL }]
 [[CONSTRAINT constraint] CHECK (cond-exp) [deferrability]]
```

### ■ Spezifikationsmöglichkeiten

- Optionale Angabe von Default-Werten
- Wertebereichseingrenzung durch benannte CHECK-Bedingung möglich
- CHECK-Bedingungen können Relationen der DB referenzieren; SQL-Domänen sind also dynamisch!

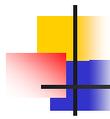
### ■ Beispiele:

```
CREATE DOMAIN ABTNR AS CHAR (6)
```

```
CREATE DOMAIN ALTER AS INT DEFAULT NULL
CONSTRAINT ALTERSBEGRENZUNG
CHECK (VALUE=NULL OR (VALUE > 18 AND VALUE < 70))
```

N. Ritter, HMS, 13.09.2007

52



## Datendefinition (7)



- Definition von Attributen

```
column-def
::= column { data-type | domain }
 [DEFAULT { literal | niladic-function-ref | NULL }]
 [column-constraint-def-list]
```

- Spezifikation von

- Attributname
- Datentyp bzw. Domain
- Defaultwert sowie Constraints

- Beispiele:

PNAME CHAR (30)

PALTER ALTER (siehe Definition von Domain ALTER)



## Datendefinition (8)



- Definition von Attributen (Forts.)

- Als Constraints können definiert werden
  - Verbot von Nullwerten (NOT NULL)
  - Eindeutigkeit (UNIQUE bzw. PRIMARY KEY)
  - FOREIGN-KEY-Klausel
  - CHECK-Bedingungen
- Vorteile der Vergabe von Constraint-Namen
  - Diagnosehilfe bei Fehlern
  - gezieltes Ansprechen bei SET oder DROP des Constraints

```
column-constraint-def ::= [CONSTRAINT constraint]
 { NOT NULL | { PRIMARY KEY | UNIQUE }
 | references-def | CHECK (cond-exp) } [deferrability]
```

## Datendefinition (9)

- Definition von Attributen (Forts.)

- **Beispiel:**

```
Verkaufs_Preis DECIMAL (9, 2),
CONSTRAINT Ausverkauf
 CHECK (Verkaufs_Preis
 <= (SELECT MIN (Preis) FROM Konkurrenz_Preise))
```

- **Überprüfungszeitpunkt**

- Jeder Constraint bzgl. einer SQL2-Transaktion ist zu jedem Zeitpunkt in einem von zwei Modi: „immediate“ oder „deferred“
    - Der Default-Modus ist „immediate“

```
deferrability
 : : = INITIALLY { DEFERRED | IMMEDIATE }
 [NOT] DEFERRABLE
```

## Datendefinition (10)

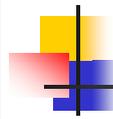
- Definition von Attributen (Forts.)

- Aufbau der **FOREIGN-KEY**-Klausel

```
references-def : : =
 REFERENCES base-table [(column-commalist)
 [ON DELETE referential-action]
 [ON UPDATE referential-action]

referential-action
: : = NO ACTION | CASCADE | RESTRICT | SET DEFAULT | SET NULL
```

- Fremdschlüssel kann auch auf Schlüsselkandidat definiert sein
    - Referentielle Aktionen werden später behandelt



## Datendefinition (11)



- Erzeugung von Basisrelationen

```
CREATE TABLE base-table (base-table-element-commalist)
base-table-element
 ::= column-def | base-table-constraint-def
```

- Definition aller zugehörigen Attribute mit Typfestlegung
- Spezifikation aller Integritätsbedingungen (Constraints)
- Beispiel: Definition der Relationen ABT und PERS

```
CREATE TABLE ABT
(ANR ABTNR PRIMARY KEY,
 ANAME CHAR (30) NOT NULL,
 ANZAHL-ANGEST INT NOT NULL,
 ...)
```



## Datendefinition (12)

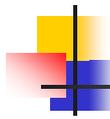


- Erzeugung von Basisrelationen (Forts.)

- Beispiel (Forts.):

```
CREATE TABLE PERS
(PNR INT PRIMARY KEY,
 BERUF CHAR (30),
 PNAME CHAR (30) NOT NULL,
 PALTER ALTER, (* siehe Domaindefinition *)
 MGR INT REFERENCES PERS,
 ANR ABTNR NOT NULL, (* Domaindef. *)
 W-ORT CHAR (25) DEFAULT ' ',
 GEHALT DEC (9,2) DEFAULT 0,00
 CHECK (GEHALT < 120.000,00)

FOREIGN KEY (ANR) REFERENCES ABT)
```



## Beziehungen (16)



- Abbildung von Beziehungen - Zusammenfassung
  - Relationenmodell ,hat' **wertbasierte** Beziehungen  
(objektorientierte Datenmodelle haben hingegen **referenzbasierte** Beziehungen)
  - Fremdschlüssel (FS) und zugehöriger Primärschlüssel/Schlüsselkandidat (PS/SK) repräsentieren eine Beziehung (gleiche Wertebereiche!)
  - Alle Beziehungen (FS ↔ PS/SK) sind binär und symmetrisch
  - Auflösung einer Beziehung geschieht durch Suche
  - Es sind i. allg. k (1:n)-Beziehungen zwischen zwei Relationen möglich
  - **Spezifikationsmöglichkeiten in SQL**

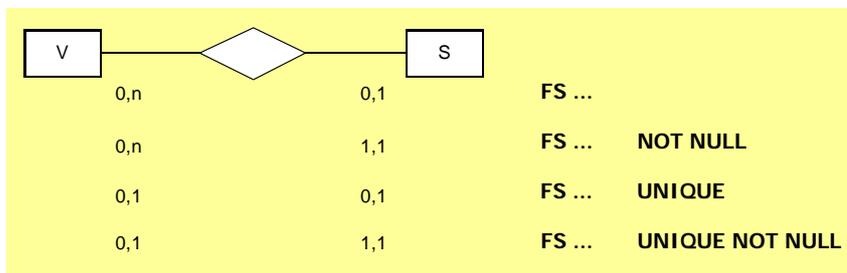
|    |                                                           |
|----|-----------------------------------------------------------|
| PS | <b>PRIMARY KEY</b><br>(implizit: <b>UNIQUE NOT NULL</b> ) |
| SK | <b>UNIQUE [NOT NULL]</b>                                  |
| FS | <b>[UNIQUE] [NOT NULL]</b>                                |

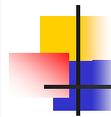


## Beziehungen (17)



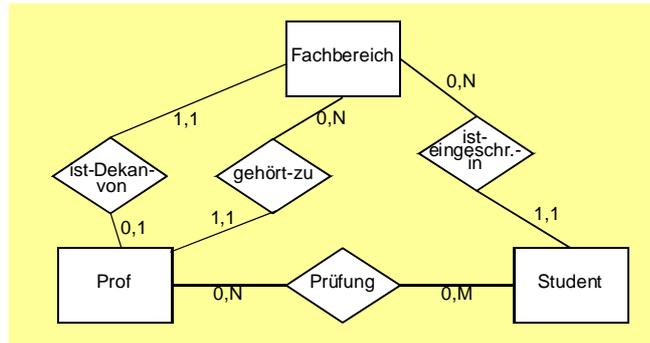
- Abbildung von Beziehungen – Zusammenfassung (Forts.)
  - **Spezifikationsmöglichkeiten in SQL (Forts.)**
    - Fremdschlüsseldeklaration in S:





## Datendefinition (13)

- Beispiel
  - Miniwelt (ER-Diagramm)



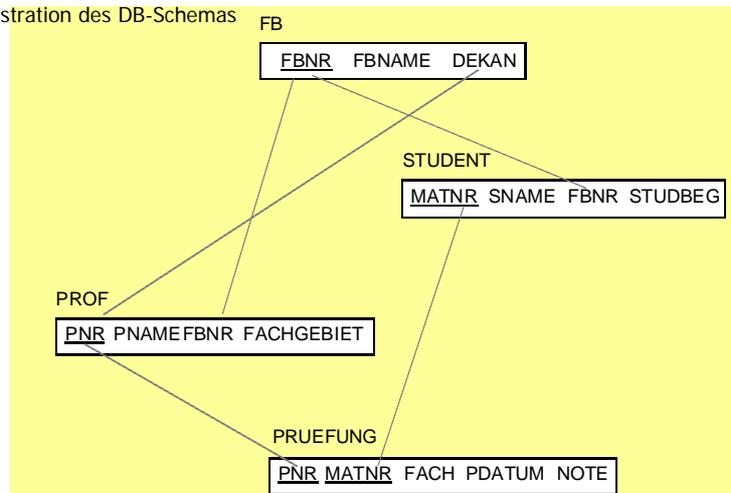
N. Ritter, HMS, 13.09.2007

61

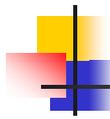


## Datendefinition (14)

- Beispiel (Forts.)
  - Illustration des DB-Schemas



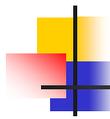
N. Ritter, I



## Datendefinition (15)

- Beispiel (Forts.)
  - Datendefinition - Wertebereiche:

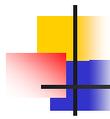
```
CREATE DOMAIN FACHBEREICHSNUMMER AS CHAR (4)
CREATE DOMAIN FACHBEREICHNAME AS VARCHAR (20)
CREATE DOMAIN FACHBEZEICHNUNG AS VARCHAR (20)
CREATE DOMAIN NAMEN AS VARCHAR (30)
CREATE DOMAIN PERSONALNUMMER AS CHAR (4)
CREATE DOMAIN MATRIKELNUMMER AS INT
CREATE DOMAIN NOTEN AS SMALLINT
CREATE DOMAIN DATUM AS DATE
```



## Datendefinition (16)

- Beispiel (Forts.)
  - Datendefinition - Relationen:

```
CREATE TABLE FB (
 FBNR FACHBEREICHSNUMMER PRIMARY KEY,
 FBNAME FACHBEREICHNAME UNIQUE,
 DEKAN PERSONALNUMMER UNIQUE NOT NULL,
 CONSTRAINT FFK FOREIGN KEY (DEKAN)
 REFERENCES PROF (PNR)
 ON UPDATE CASCADE
 ON DELETE RESTRICT)
```



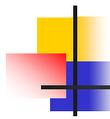
## Datendefinition (17)

- Beispiel (Forts.)
  - Datendefinition – Relationen (Forts.):

```
CREATE TABLE PROF (
 PNR PERSONALNUMMER PRIMARY KEY,
 PNAME NAMEN NOT NULL,
 FBNR FACHBEREICHNUMMER NOT NULL,
 FACHGEBIET FACHBEZEICHNUNG,

 CONSTRAINT PFK1 FOREIGN KEY (FBNR)
 REFERENCES FB (FBNR)
 ON UPDATE CASCADE
 ON DELETE SET DEFAULT)
```

- Es wird darauf verzichtet, die Rückwärtsrichtung der „ist-Dekan-von“-Beziehung explizit als Fremdschlüsselbeziehung zu spezifizieren. Damit fällt auch die mögliche Spezifikation von referentiellen Aktionen weg.

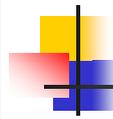


## Datendefinition (18)

- Beispiel (Forts.)
  - Datendefinition – Relationen (Forts.):

```
CREATE TABLE STUDENT (
 MATNR MATRIKELNUMMER PRIMARY KEY,
 SNAME NAMEN NOT NULL,
 FBNR FACHBEREICHNUMMER NOT NULL,
 STUDBEG DATUM,

 CONSTRAINT SFK FOREIGN KEY (FBNR)
 REFERENCES FB (FBNR)
 ON UPDATE CASCADE
 ON DELETE RESTRICT)
```



## Datendefinition (19)

- Beispiel (Forts.)
  - Datendefinition – Relationen (Forts.):

```

CREATE TABLE PRUEFUNG (
 PNR PERSONALNUMMER,
 MATNR MATRIKELNUMMER,
 FACH FACHBEZEICHNUNG,
 PDATUM DATUM NOT NULL,
 NOTE NOTEN NOT NULL,

 PRIMARY KEY (PNR, MATNR),

 CONSTRAINT PR1FK FOREIGN KEY (PNR)
 REFERENCES PROF (PNR)
 ON UPDATE CASCADE
 ON DELETE CASCADE,

 CONSTRAINT PR2FK FOREIGN KEY (MATNR)
 REFERENCES STUDENT (MATNR)
 ON UPDATE CASCADE
 ON DELETE CASCADE)

```

N. Ritter, I

67



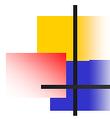
## Datendefinition (20)

- Beispiel (Forts.)
  - Ausprägungen

| PROF | <u>PNR</u> | PNAME    | FBNR | FACHGEBIET          | FB   | <u>FBNR</u>     | FBNAME | DEKAN |
|------|------------|----------|------|---------------------|------|-----------------|--------|-------|
|      | 1234       | HARDER   | FB 5 | DATENBANKSYSTEME    | FB 9 | WIRTSCHAFTSWISS | 4711   |       |
|      | 5678       | WEDEKIND | FB 9 | INFORMATIONSSYSTEME | FB 5 | INFORMATIK      | 2223   |       |
|      | 4711       | MÜLLER   | FB 9 | OPERATIONS RESEARCH |      |                 |        |       |
|      | 6780       | NEHMER   | FB 5 | BETRIEBSSYSTEME     |      |                 |        |       |
|      | 2223       | RICHTER  | FB 5 | EXPERTENSYSTEME     |      |                 |        |       |

| STUDENT | <u>MATNR</u> | SNAME   | FBNR | STUDBEG  | PRUFUNG | <u>PNR</u> | <u>MATNR</u> | FACH | PDATUM   | NOTE |
|---------|--------------|---------|------|----------|---------|------------|--------------|------|----------|------|
|         | 123 766      | COY     | FB 9 | 1.10.95  |         | 5678       | 123 766      | BWL  | 22.10.97 | 4    |
|         | 225 332      | MÜLLER  | FB 5 | 15. 4.87 |         | 4711       | 123 766      | OR   | 16. 1.98 | 3    |
|         | 654 711      | ABEL    | FB 5 | 15.10.94 |         | 1234       | 654 711      | DV   | 17. 4.97 | 2    |
|         | 226 302      | SCHULZE | FB 9 | 1.10.95  |         | 1234       | 123 766      | DV   | 17. 4.97 | 4    |
|         | 196 481      | MAIER   | FB 5 | 23.10.95 |         | 6780       | 654 711      | SP   | 19. 9.97 | 2    |
|         | 130 680      | SCHMID  | FB 9 | 1. 4.97  |         | 1234       | 196 481      | DV   | 15.10.97 | 1    |
|         |              |         |      |          |         | 6780       | 196 481      | BS   | 23.12.97 | 3    |



## Wartung von Beziehungen (1)



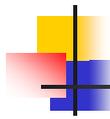
- Relationale Invarianten / referentielle Integrität:
  - Primärschlüsselbedingung: Eindeutigkeit, keine Nullwerte!
  - Fremdschlüsselbedingung: Zugehöriger PS (SK) muss existieren
- Potentielle Gefährdung
  - Operationen in der Sohn-Relation
    - Einfügen eines Sohn-Tupels
    - Ändern des FS in einem Sohn-Tupel
    - Löschen eines Sohn-Tupels
    - Welche Maßnahmen sind erforderlich?
      - Beim Einfügen erfolgt eine Prüfung, ob in einem Vater-Tupel ein PS/SK-Wert gleich dem FS-Wert des einzufügenden Tupels existiert
      - Beim Ändern eines FS-Wertes erfolgt eine analoge Prüfung



## Wartung von Beziehungen (2)



- Potentielle Gefährdung (Forts.)
  - Operationen in der Vater-Relation
    - Löschen eines Vater-Tupels
    - Ändern des PS/SK in einem Vater-Tupel
    - Einfügen eines Vater-Tupels
    - Welche Reaktion ist wann möglich/sinnvoll?
      - Verbiete Operation
      - Lösche/ändere rekursiv Tupel mit zugehörigen FS-Werten
      - Falls Sohn-Tupel erhalten bleiben soll (nicht immer möglich, z.B. bei Existenzabhängigkeit), setze FS-Wert zu NULL oder Default
  - Wie geht man mit NULL-Werten um?
    - Spezielle Semantiken von NULL-Werten
    - Dreiwertige Logik verwirrend: T, F, ?
    - Setzung: NULL  $\neq$  NULL (z. B. beim Verbund)
    - bei Operationen: Ignorieren von NULL-Werten



## Wartung von Beziehungen (3)



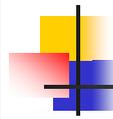
- SQL2-Standard führt „referential actions“ ein
- Genauere Spezifikation der referentiellen Aktionen für jeden Fremdschlüssel (FS)
  - Sind „Nullen“ verboten ?
    - **NOT NULL**
  - Löschregel für Zielrelation (referenzierte Relation)
    - **ON DELETE**  
{**CASCADE** | **RESTRICT** | **SET NULL** | **SET DEFAULT** | **NO ACTION**}
  - Änderungsregel für Ziel-Primärschlüssel (PS oder SK)
    - **ON UPDATE**  
{**CASCADE** | **RESTRICT** | **SET NULL** | **SET DEFAULT** | **NO ACTION**}
  - Die Option **RESTRICT** wird hier explizit aufgeführt; sie entspricht dem Fall, dass die gesamte Klausel weggelassen wird.



## Wartung von Beziehungen (4)



- Genauere Spezifikation der referentiellen Aktionen (Forts.)
  - **RESTRICT**: Operation wird nur ausgeführt, wenn keine zugehörigen Sätze (FS-Werte) vorhanden sind
  - **CASCADE**: Operation „kaskadiert“ zu allen zugehörigen Sätzen
  - **SET NULL**: FS wird in zugehörigen Sätzen zu „Null“ gesetzt
  - **SET DEFAULT**: FS wird in den zugehörigen Sätzen auf einen benutzerdefinierten Default-Wert gesetzt
  - **NO ACTION**: Für die spezifizierte Referenz wird keine referentielle Aktion ausgeführt. Durch eine DB-Operation können jedoch mehrere Referenzen (mit unterschiedlichen Optionen) betroffen sein; am Ende aller zugehörigen referentiellen Aktionen wird die Einhaltung der referentiellen Integrität geprüft

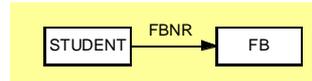


## Wartung von Beziehungen (5)



- Diskussion der Auswirkungen referentieller Aktionen am Beispiel

1. Isolierte Betrachtung von STUDENT-FB



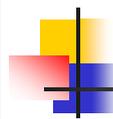
- Beispiel-DB

| STUDENT | <u>MATRNR</u> | SNAME   | FBNR | FB | <u>FBNR</u> | FBNAME          |
|---------|---------------|---------|------|----|-------------|-----------------|
|         | 123766        | COY     | FB9  |    | FB9         | WIRTSCHAFTSWISS |
|         | 225332        | MÜLLER  | FB5  |    | FB5         | INFORMATIK      |
|         | 654711        | ABEL    | FB5  |    |             |                 |
|         | 226302        | SCHULZE | FB9  |    |             |                 |

- Operationen
  - Lösche FB (mit FBNR „FB5“)
  - Ändere FB (FBNR=„FB9“ → FBNR=„FB10“)
- Referentielle Aktionen
  - DC, DSN, DSD, DR, DNA
  - UC, USN, USD, UR, UNA

N. Ritter, HMS, 13.09.2007

73



## Wartung von Beziehungen (6)



- Diskussion der Auswirkungen referentieller Aktionen am Beispiel (Forts.)

2. Isolierte Betrachtung von STUDENT-PRUEFUNG-PROF



- Beispiel-DB

| STUDENT | <u>MATRNR</u> | SNAME | FBNR | PROF | <u>PNR</u> | PNAME   |
|---------|---------------|-------|------|------|------------|---------|
|         | 123766        | COY   | FB9  |      | 1234       | HAERDER |
|         | 654711        | ABEL  | FB5  |      | 4711       | MUELLER |

| PRUEFUNG | <u>PNR</u> | <u>MATRNR</u> | FACH |
|----------|------------|---------------|------|
|          | 4711       | 123766        | OR   |
|          | 1234       | 654711        | DV   |
|          | 1234       | 123766        | DV   |
|          | 4711       | 654711        | OR   |

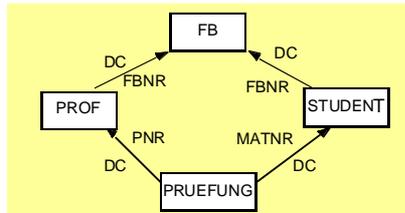
- Einsatz von
  - USN, DSN → Schlüsselverletzung
  - USD, DSD → ggf. Mehrdeutigkeit
  - UNA, DNA → Wirkung identisch mit UR, DR

N. Ritter, HMS, 13.09.2007

74

## Wartung von Beziehungen (7)

- Diskussion der Auswirkungen referentieller Aktionen am Beispiel (Forts.)
  - Unabhängigkeit von Beziehungen hinsichtlich referentieller Aktionen?
- 3. Vollständiges Beispiel



- Lösche FB (mit FBNR „FB9“)

*'erst links':*

- Löschen in FB
- Löschen in PROF
- Löschen in PRUEFUNG
- Löschen in STUDENT
- Löschen in PRUEFUNG

*'erst rechts':*

- Löschen in FB
- Löschen in STUDENT
- Löschen in PRUEFUNG
- Löschen in PROF
- Löschen in PRUEFUNG

- Eindeutigkeit: Ergebnis der Operation ist reihenfolge-unabhängig

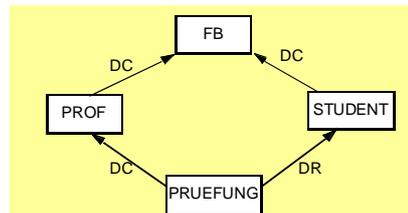
N. Ritter, HMS, 13.09.2007

→ sicheres Schema!

75

## Wartung von Beziehungen (8)

- Diskussion der Auswirkungen referentieller Aktionen am Beispiel (Forts.)
- 3. Vollständiges Beispiel – Modifiziertes Schema



- Lösche FB (mit FBNR „FB9“)

*'erst links':*

- Löschen in FB
  - Löschen in PROF
  - Löschen in PRUEFUNG
  - Löschen in STUDENT
  - Zugriff auf PRUEFUNG
- Wenn ein Student bei einem FB-fremden Professor geprüft wurde  
→ Rücksetzen

*'erst rechts':*

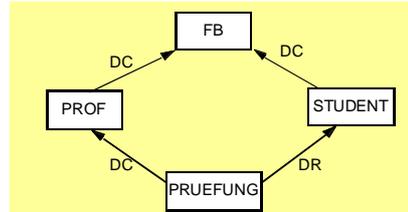
- Löschen in FB
  - Löschen in STUDENT
  - Zugriff auf PRUEFUNG
- Wenn ein gerade gelöschter Student eine Prüfung abgelegt hatte  
→ Rücksetzen  
sonst:  
- Löschen in PROF  
- Löschen in PRUEFUNG

N. Ritter, HMS, 13.09.2007

76

## Wartung von Beziehungen (9)

- Diskussion der Auswirkungen referentieller Aktionen am Beispiel (Forts.)
  3. Vollständiges Beispiel – Modifiziertes Schema (Forts.)



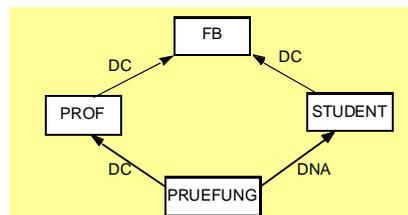
- Es können reihenfolgenabhängige Ergebnisse auftreten!
- Die Reihenfolgenabhängigkeit ist hier wertabhängig

N. Ritter, HMS, 13.09.2007

77

## Wartung von Beziehungen (10)

- Diskussion der Auswirkungen referentieller Aktionen am Beispiel (Forts.)
  3. Vollständiges Beispiel – Nochmalig modifiziertes Schema



- Lösche FB (mit FBNR „FB9“)

*'erst links':*

- Löschen FB
  - Löschen PROF
  - Löschen PRUEFUNG
  - Löschen STUDENT
- Test, ob es noch offene Referenzen in PRUEFUNG auf gelöschte Studenten gibt; wenn ja → Rücksetzen

*'erst rechts':*

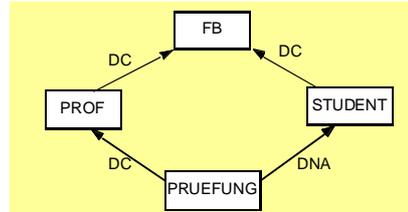
- Löschen FB
  - Löschen STUDENT
  - Löschen PROF
  - Löschen PRUEFUNG
- Test, ob es noch offene Referenzen in PRUEFUNG auf gelöschte Studenten gibt; wenn ja → Rücksetzen

N. Ritter, I

78

## Wartung von Beziehungen (11)

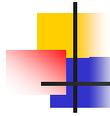
- Diskussion der Auswirkungen referentieller Aktionen am Beispiel (Forts.)
  3. Vollständiges Beispiel –  
Nochmalig modifiziertes  
Schema (Forts.)



- Bei der NA-Option wird der explizite Test der referenzierenden Relation ans Ende **der Operation** verschoben. Eine Verletzung der referentiellen Beziehung führt zum Rücksetzen → **Schema ist immer sicher**

## Wartung von Beziehungen (12)

- Maßnahmen zur Verhinderung von Mehrdeutigkeiten
  - Statische Schemaanalyse zur Feststellung sicherer DB-Schemata
    - nur bei einfach strukturierten Schemata effektiv
    - hohe Komplexität der Analysealgorithmen
    - bei wertabhängigen Konflikten zu restriktiv (konflikträchtige Schemata)
  - Dynamische Überwachung der Modifikationsoperationen
    - hoher Laufzeitaufwand



## Wartung von Beziehungen (13)



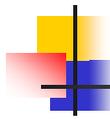
- Maßnahmen zur Verhinderung von Mehrdeutigkeiten - Vorgehensweisen
  1. Falls Sicherheit eines Schemas festgestellt werden kann, ist keine Laufzeitüberwachung erforderlich
  2. Alternative Möglichkeiten zur Behandlung konfliktträchtiger Schemata, nach dem die statische Schemaanalyse die Sicherheit des Schemas nicht feststellen konnte
    - sie werden verboten, oder
    - sie werden erlaubt und
      - die referentiellen Aktionen werden bei jeder Operation dynamisch überwacht
      - falls ein Konflikt erkannt wird, wird die Operation zurückgesetzt



## Schemaevolution (1)



- Wachsender oder sich ändernder Informationsbedarf
  - Erzeugen/Löschen von Tabellen (und Sichten)
  - Hinzufügen, Ändern und Löschen von Spalten
  - Anlegen/Ändern von referentiellen Beziehungen
  - Hinzufügen, Modifikation, Wegfall von Integritätsbedingungen
- Hoher Grad an logischer Datenunabhängigkeit ist sehr wichtig!
- Zusätzliche Änderungen im DB-Schema durch veränderte Anforderungen bei der DB-Nutzung
  - Dynamisches Anlegen von Zugriffspfaden
  - Aktualisierung der Zugriffskontrollbedingungen

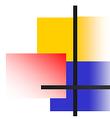


## Schemaevolution (2)



- Dynamische Änderung von Tabellen

```
ALTER TABLE base-table
{ ADD [COLUMN] column-def
| ALTER [COLUMN] column
 {SET default-def | DROP DEFAULT}
| DROP [COLUMN] column {RESTRICT | CASCADE}
| ADD base-table-constraint-def
| DROP CONSTRAINT constraint {RESTRICT | CASCADE}}
```



## Schemaevolution (3)



- Dynamische Änderung von Tabellen - Beispiele:

- Erweiterung der Tabellen Abt und Pers um neue Spalten

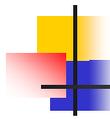
```
ALTER TABLE Pers ADD Svrn INT UNIQUE
```

```
ALTER TABLE Abt ADD Geh-Summe INT
```

- Verkürzung der Tabelle Pers um eine Spalte

```
ALTER TABLE Pers DROP COLUMN Alter RESTRICT
```

- Wenn die Spalte die einzige der Tabelle ist, wird die Operation zurückgewiesen.
- Da RESTRICT spezifiziert ist, wird die Operation zurückgewiesen, wenn die Spalte in einer Sicht oder einer Integritätsbedingung (Check) referenziert wird.
- CASCADE dagegen erzwingt die Folgelöschung aller Sichten und Check-Klauseln, die von der Spalte abhängen.



## Schemaevolution (4)



- Löschen von Schemaelementen

```
DROP {TABLE base-table | VIEW view |
 DOMAIN domain | SCHEMA schema }
 {RESTRICT | CASCADE}
```

- Falls Objekte (Tabellen, Sichten, ...) nicht mehr benötigt werden, können sie durch die DROP-Anweisung aus dem System entfernt werden.
- Mit der **CASCADE**-Option können 'abhängige' Objekte (z.B. Sichten auf Tabellen oder anderen Sichten) mitentfernt werden
- **RESTRICT** verhindert Löschen, wenn die zu löschende Tabelle noch durch Sichten oder Integritätsbedingungen referenziert wird



## Schemaevolution (5)



- Löschen von Schemaelementen - Beispiele

- Löschen von Tabelle Pers

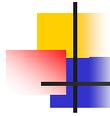
```
DROP TABLE Pers RESTRICT
```

- PersConstraint sei definiert auf Pers

```
ALTER TABLE Pers
 DROP CONSTRAINT PersConstraint CASCADE
```

```
DROP TABLE Pers RESTRICT
```

- Durchführung der Schemaevolution
  - Aktualisierung von Tabellenzeilen des SQL-Definitionsschemas
  - "tabellengetriebene" Verarbeitung der Metadaten durch das DBS



## Indexierung (1)



- **Einsatz von Indexstrukturen**
  - Beschleunigung der Suche: Zugriff über Spalten (Schlüsselattribute)
  - Kontrolle von Integritätsbedingungen (relationale Invarianten)
  - Zeilenzugriff in der logischen Ordnung der Schlüsselwerte
  - Gewährleistung der Clustereigenschaft für Tabellen
  - Aber auch: erhöhter Aktualisierungsaufwand und Speicherplatzbedarf
- **Einrichtung von Indexstrukturen**
  - Datenunabhängigkeit des Relationenmodells erlaubt ein Hinzufügen und Löschen
  - jederzeit möglich, um z. B. bei veränderten Benutzerprofilen das Leistungsverhalten zu optimieren
  - "beliebig" viele Indexstrukturen pro Tabelle und mit unterschiedlichen Spaltenkombinationen als Schlüssel möglich
  - Steuerung der Eindeutigkeit der Schlüsselwerte, der Clusterbildung
  - Freiplatzanteil (PCTFREE) in jeder Indexseite beim Anlegen erleichtert das Wachstum
  - Spezifikation: DBA oder Benutzer



## Indexierung (2)



- Im SQL-Standard keine Anweisung vorgesehen, jedoch in realen Systemen (z. B. IBM DB2):

```
CREATE [UNIQUE] INDEX index
ON base-table (column [ORDER] [, column [ORDER]] ...)
[CLUSTER] [PCTFREE]
```

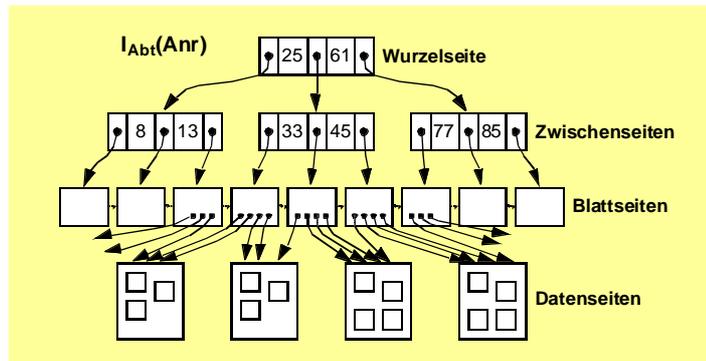
- Nutzung eines vorhandenen Index
  - Entscheidung durch DBS-Optimizer



## Indexierung (3)



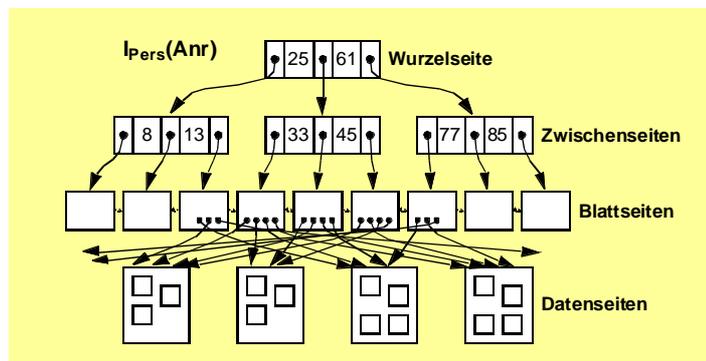
- Index mit Clusterbildung

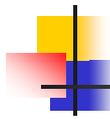


## Indexierung (4)



- Index ohne Clusterbildung

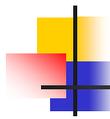




## Indexierung (5)



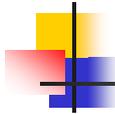
- Beispiele
  - **Erzeugung einer Indexstruktur mit Clusterbildung auf der Spalte Anr von Abt**  
**CREATE UNIQUE INDEX Persind1 ON Abt (Anr) CLUSTER**
    - Realisierung z. B. durch B\*-Baum (oder Hashing, mit verminderter Funktionalität)
    - **UNIQUE**: keine Schlüsselduplikate im Index
    - **CLUSTER**: zeitoptimale sortiert-sequentielle Verarbeitung (Scan-Operation)
  - **Erzeugung einer Indexstruktur auf den Spalten Anr (absteigend) und Gehalt (aufsteigend) von Pers**  
**CREATE INDEX Persind2 ON Pers (Anr DESC, Gehalt ASC)**



## Indexierung (6)



- **Typische Implementierung eines Index: B\*-Baum**  
(wird von allen DBS angeboten!)
  - dynamische Reorganisation durch Aufteilen (Split) und Mischen von Seiten
  - Wesentliche Funktionen
    - direkter Schlüsselzugriff auf einen indexierten Satz
    - sortiert sequentieller Zugriff auf alle Sätze (unterstützt Bereichsanfragen, Verbundoperation usw.)
  - **Balancierte Struktur**
    - unabhängig von Schlüsselmenge
    - unabhängig von Einfügereihenfolge



## Sichten (1)

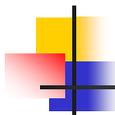


- Ziel: Festlegung
  - welche Daten Benutzer sehen wollen (Vereinfachung, leichtere Benutzung)
  - welche Daten sie nicht sehen dürfen (Datenschutz)
  - einer zusätzlichen Abbildung (erhöhte Datenunabhängigkeit)
- Sicht (*View*)
  - mit Namen bezeichnete, aus Tabellen abgeleitete, virtuelle Tabelle (Anfrage)
- Korrespondenz zum externen Schema bei ANSI/SPARC (Benutzer sieht jedoch i. allg. mehrere Sichten (Views) und Tabellen)
- Syntax

```
CREATE VIEW view [(column-commalist)]
AS table-exp
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

N. Ritter, HMS, 13.09.2007

93



## Sichten (2)



- Beispiele
  - **Sicht, die alle Programmierer mit einem Gehalt < 30.000 umfasst.**

```
CREATE VIEW
Arme_Programmierer (Pnr, Name, Beruf, Gehalt, Anr)
AS SELECT Pnr, Name, Beruf, Gehalt, Anr
FROM Pers
WHERE Beruf = 'Programmierer' AND Gehalt < 30 000
```

- **Sicht für den Datenschutz**

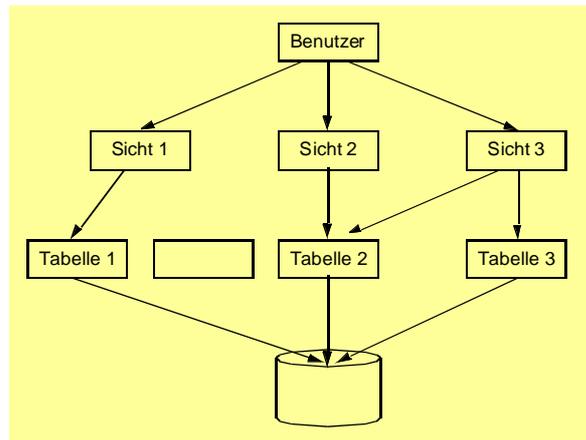
```
CREATE VIEW Statistik (Beruf, Gehalt)
AS SELECT Beruf, Gehalt
FROM Pers
```

N. Ritter, HMS, 13.09.2007

94

## Sichten (3)

- Sichten zur Gewährleistung von Datenunabhängigkeit



N. Ritter, HMS, 13.09.2007

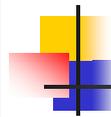
95

## Sichten (4)

- Eigenschaften von Sichten
  - Sicht kann wie eine Tabelle behandelt werden
  - Sichtsemantik: „dynamisches Fenster“ auf zugrundeliegende Tabellen
  - Sichten auf Sichten sind möglich
  - eingeschränkte Änderungen: aktualisierbare und nicht-aktualisierbare Sichten

N. Ritter, HMS, 13.09.2007

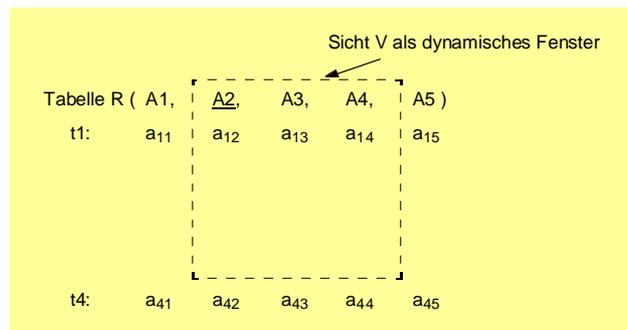
96



## Sichten (5)

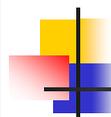


- Semantik von Sichten – ‚dynamisches Fenster‘



N. Ritter, HMS, 13.09.2007

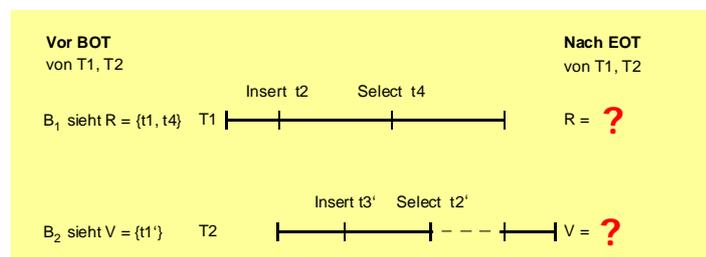
97



## Sichten (6)



- Sichtbarkeit von Änderungen
  - Wann werden welche Datenänderungen in der Tabelle/Sicht für die anderen Benutzer sichtbar? (*Beachte Beispiel auf vorangegangener Folie*)



N. Ritter, HMS, 13.09.2007

98

## Sichten (7)

- Sichtbarkeit von Änderungen
  - Wann werden welche Datenänderungen in der Tabelle/Sicht für die anderen Benutzer sichtbar? (Forts.)

Sicht V als dynamisches Fenster

| Tabelle R ( | A1,             | <u>A2,</u>      | A3,             | A4,             | A5 )            |
|-------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| t1:         | a <sub>11</sub> | a <sub>12</sub> | a <sub>13</sub> | a <sub>14</sub> | a <sub>15</sub> |
| t2:         | a <sub>21</sub> | a <sub>22</sub> | a <sub>23</sub> | a <sub>24</sub> | a <sub>25</sub> |
| t3:         | ≡               | a <sub>32</sub> | a <sub>33</sub> | a <sub>34</sub> | ≡               |
| t4:         | a <sub>41</sub> | a <sub>42</sub> | a <sub>43</sub> | a <sub>44</sub> | a <sub>45</sub> |

N. Ritter, HMS, 13.09.2007

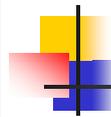
99

## Sichten (11)

- Löschen von Sichten:
  - Beispiel
    - DROP VIEW** Arme\_Programmierer **CASCADE**
      - Alle referenzierenden Sichtdefinitionen und Integritätsbedingungen werden mitgelöscht
      - **RESTRICT** würde eine Löschung zurückweisen, wenn die Sicht in weiteren Sichtdefinitionen oder CHECK-Constraints referenziert werden würde.

N. Ritter, HMS, 13.09.2007

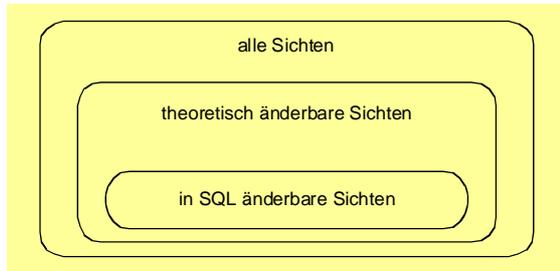
100



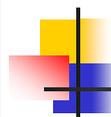
## Sichten (12)



- Änderbarkeit von Sichten



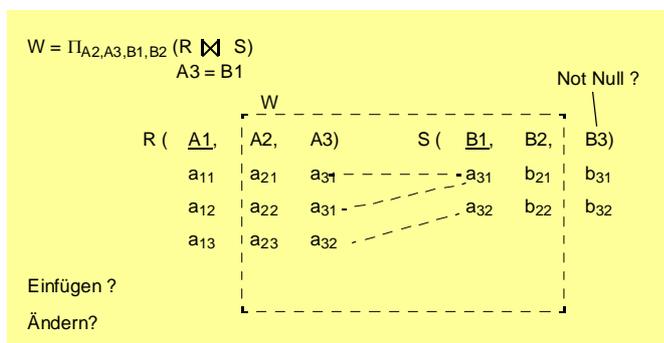
- Änderbarkeit in SQL
  - nur eine Tabelle (Basisrelation oder Sicht)
  - Schlüssel muss vorhanden sein
  - keine Aggregatfunktionen, Gruppierung und Duplikateliminierung



## Sichten (13)



- Änderbarkeit von Sichten (Forts.)
  - Sichten über mehrere Tabellen sind im Allg. nicht änderbar

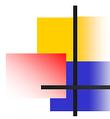




## Sichten (14)



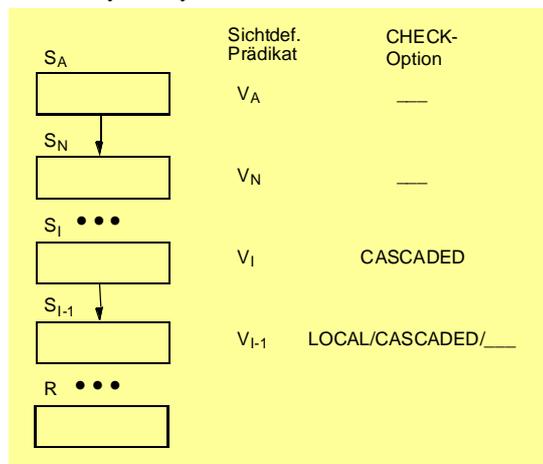
- **WITH CHECK OPTION**
  - Einfügungen und Änderungen müssen das die Sicht definierende Prädikat erfüllen, sonst Zurückweisung
  - nur auf aktualisierbaren Sichten definierbar
- Spezifikationsmöglichkeiten
  - Weglassen der CHECK-Option
  - WITH CASCADED CHECK OPTION oder äquivalent WITH CHECK OPTION
  - WITH LOCAL CHECK OPTION

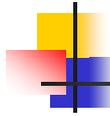


## Sichten (15)



- **WITH CHECK OPTION (Forts.)**

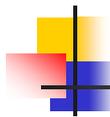




## Sichten (16)



- **WITH CHECK OPTION (Forts.)**
  - **Annahmen**
    - Sicht  $S_A$  mit dem die Sicht definierenden Prädikat  $V_A$  wird aktualisiert
    - $S_I$  ist die höchste Sicht im Abstammungspfad von  $S_A$ , die die Option CASCADED besitzt
    - Oberhalb von  $S_I$  tritt keine LOCAL-Bedingung auf
  - **Aktualisierung von  $S_A$** 
    - als Prüfbedingung wird von  $S_I$  aus an  $S_A$  "vererbt":  
$$V = V_I \wedge V_{I-1} \wedge \dots \wedge V_1$$
    - erscheint irgendeine aktualisierte Zeile von  $S_A$  nicht in  $S_I$ , so wird die Operation zurückgesetzt
    - Es ist möglich, dass Zeilen aufgrund von gültigen Einfüge- oder Änderungsoperationen aus  $S_A$  verschwinden



## Sichten (17)



- **WITH CHECK OPTION (Forts.)**
  - Aktualisierte Sicht besitzt WITH CHECK OPTION
    - Default ist CASCADED
    - Als Prüfbedingung bei Aktualisierungen ergibt sich  
$$V = V_A \wedge V_N \wedge \dots \wedge V_I \wedge \dots \wedge V_1$$
    - Zeilen können jetzt aufgrund von gültigen Einfüge- oder Änderungsoperationen nicht aus SA verschwinden
  - LOCAL hat eine undurchsichtige Semantik
    - wird hier nicht diskutiert
    - Empfehlung: nur Verwendung von CASCADED

## Sichten (18) !

- WITH CHECK OPTION (Forts.)

**Sichtenhierarchie:**

- S<sub>2</sub> mit V<sub>1</sub> ∧ V<sub>2</sub>
- S<sub>1</sub> mit V<sub>1</sub> und CASCADED
- R

- Aktualisierungsoperationen in S<sub>2</sub> (welche sind erlaubt?)
  - I<sub>1</sub> und U<sub>1</sub> erfüllen das S<sub>2</sub>-definierende Prädikat V<sub>1</sub> ∧ V<sub>2</sub> ✓
  - I<sub>2</sub> und U<sub>2</sub> erfüllen das S<sub>1</sub>-definierende Prädikat V<sub>1</sub> ✓
  - I<sub>3</sub> und U<sub>3</sub> erfüllen das S<sub>1</sub>-definierende Prädikat V<sub>1</sub> nicht ⚡

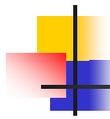
N. Ritter, HMS, 13.09.2007 107

## Sichten (19) !

- WITH CHECK OPTION (Forts.)
- Beispiel
  - Tabelle Pers
  - Sicht1 auf Pers: AP1, mit Beruf = 'Progr' AND Gehalt < '30K'
  - Sicht2 auf AP1: AP2, mit Gehalt > '20K'

|      | Sichtdef. Prädikat | CHECK-Optionen |      |      |      |
|------|--------------------|----------------|------|------|------|
|      |                    | 1              | 2    | 3    | 4    |
| AP2  | > '20K'            | —              | —    | CASC | CASC |
| AP1  | < '30K'            | —              | CASC | —    | CASC |
| PERS |                    |                |      |      |      |

N. Ritter, HMS, 13.09.2007 108



## Sichten (20)



### ■ WITH CHECK OPTION (Forts.)

#### ■ Beispiel (Forts.)

##### ■ Operationen

|                                                                                                     | 1 | 2 | 3 | 4 |
|-----------------------------------------------------------------------------------------------------|---|---|---|---|
| 1. <b>INSERT INTO</b> AP2 (PNR, BERUF, GEHALT, ANR)<br><b>VALUES</b> ( 1234, 'Progr', '25K', 'K55') | ✓ | ✓ | ✓ | ✓ |
| 2. <b>INSERT INTO</b> AP2 (PNR, BERUF, GEHALT, ANR)<br><b>VALUES</b> ( 4711, 'Progr', '15K', 'K55') | ✓ | ✓ | ⚡ | ⚡ |
| 3. <b>UPDATE</b> AP2<br><b>SET</b> Gehalt = Gehalt + '10K'<br><b>WHERE</b> ANR = 'K55'              | ✓ | ⚡ | ⚡ | ⚡ |

AP2: > 20K - - CASC CASC

AP1: < 30K - CASC - CASC

N. Ritter, HMS, 13.09.2007

109



## Zusammenfassung (1)



### ■ SQL-Anfragen

- Mengenorientierte Spezifikation, verschiedene Typen von Anfragen
- Vielfalt an Suchprädikaten
- Auswahlmächtigkeit von SQL ist höher als die der Relationenalgebra.
- Erklärungsmodell für die Anfrageauswertung: Festlegung der Semantik von Anfragen mit Hilfe von Grundoperationen
- Optimierung der Anfrageauswertung durch das DBS

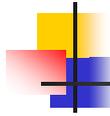
### ■ Mengenorientierte Datenmanipulation

### ■ Datendefinition

- CHECK-Bedingungen für Wertebereiche, Attribute und Relationen
- Spezifikation des Überprüfungszeitpunktes

N. Ritter, HMS, 13.09.2007

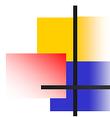
110



## Zusammenfassung (2)



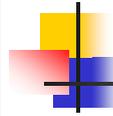
- Kontrolle von Beziehungen
  - SQL erlaubt nur die Spezifikation von binären Beziehungen.
  - Referentielle Integrität von FS --> PS/SK wird stets gewährleistet.
  - Rolle von PRIMARY KEY, UNIQUE, NOT NULL
  - Es ist nur eine eingeschränkte Nachbildung von Kardinalitätsrestriktionen möglich; insbesondere kann nicht spezifiziert werden, dass „ein Vater Söhne haben muss“.
- Wartung der referentiellen Integrität
  - SQL2/3 bietet reichhaltige Optionen für referentielle Aktionen
  - Es sind stets sichere Schemata anzustreben
  - Falls eine statische Schemaanalyse zu restriktiv für die Zulässigkeit eines Schemas ist, muss für das gewünschte Schema eine Laufzeitüberwachung der referentiellen Aktionen erfolgen.



## Zusammenfassung (3)



- Schemaevolution
  - Änderung/Erweiterung von Spalten, Tabellen, Integritätsbedingungen, ...
- Indexstrukturen als B\*-Bäume
  - mit und ohne Clusterbildung spezifizierbar
  - Balancierte Struktur unabhängig von Schlüsselmenge und Einfügereihenfolge
  - dynamische Reorganisation durch Aufteilen (Split) und Mischen von Seiten
  - direkter Schlüsselzugriff auf einen indexierten Satz
  - sortiert sequentieller Zugriff auf alle Sätze (unterstützt Bereichsanfragen, Verbundoperation usw.)



## Zusammenfassung (4)



- Sichtenkonzept
  - Erhöhung der Benutzerfreundlichkeit
  - Flexibler Datenschutz
  - Erhöhte Datenunabhängigkeit
  - Rekursive Anwendbarkeit
  - Eingeschränkte Aktualisierungsmöglichkeiten