

## 5. Die Standardsprache SQL

- **Grundlagen**
  - Funktions- und Einsatzbereiche
  - Befehlsübersicht und SQL-Grammatik
- **Mengenorientierte Anfragen (Retrieval)<sup>1</sup>**
  - Anfragetypen
  - Aggregatfunktionen, Vergleichsprädikate
  - Erklärungsmodell für die Anfrageauswertung
- **Möglichkeiten der Datenmanipulation (DML)**
- **Möglichkeiten der Datendefinition (DDL)**
  - Basisrelationen
  - Integritätsbedingungen
- **Abbildung von Beziehungen**
  - Rolle des Fremdschlüssels
  - Umsetzung der verschiedenen Beziehungstypen
- **Wartung von Beziehungen**
  - Relationale Invarianten
  - Auswirkungen referentieller Aktionen

---

1. Eine **ungenau**e Antwort auf die **richtige Frage** ist viel mehr wert als eine **präzise** Antwort auf die **falsche Frage**. (John W. Tukey)

## Abbildungsorientierte Sprachen am Beispiel von SQL

- **Seit 1974 viele Sprachentwürfe**
  - SQUARE: Specifying Queries As Relational Expressions
  - SEQUEL: Structured English Query Language
  - Weiterentwicklung zu **SQL (Structured Query Language)**
  - QUEL, OLQ, PRTV, . . .
- **Sprachentwicklung von SQL<sup>2</sup>**
  - Entwicklung einer vereinheitlichten DB-Sprache für alle Aufgaben der DB-Verwaltung
  - Lehrexperimente mit Studenten mit und ohne Programmiererfahrung
  - Erweiterung der Anfragesprache zur „natürlichen“ Formulierung bestimmter Fragen
  - gezielte Verbesserungen verschiedener Sprachkonstrukte zur Erleichterung des Verständnisses und zur Reduktion von Fehlern
  - leichter Zugang durch verschiedene „Sprachebenen“ anwachsender Komplexität:
    - einfache Anfragemöglichkeiten für den gelegentlichen Benutzer
    - mächtige Sprachkonstrukte für den besser ausgebildeten Benutzer
- **Spezielle Sprachkonstrukte für den DBA**
- **SQL wurde „de facto“-Standard in der relationalen Welt**  
(X3H2-Vorschlag wurde 1986 von ANSI, 1987 von ISO akzeptiert)
- **Weiterentwicklung des Standards<sup>3</sup>**  
**SQL2 mit drei Stufen (1992),**  
**SQL:1999, SQL:2003, SQL:2007 (XQuery, SQL/XML, and the Semantic Web)**

---

2. Information Technology – Database Language SQL - Part 1 and Part 2: Framework (for SQL:1999) and Foundation (SQL:1999), International Standard (www.jtc1sc32.org)

Information Technology – Database Language SQL - Technical Corrigendum xxx for SQL:1999, ... 2008-01-27: >1700 Dokumente

3. The nice thing about standards is that there are so many of them to choose from (Andrew S. Tanenbaum)

## Anfragen in SQL

- Eigenschaften

- Auswahlvermögen **äquivalent** dem Relationenkalkül und der **Relationenalgebra**
- Vermeidung von mathematischen Konzepten wie Quantoren

↳ trotzdem: relational vollständig

**SQL: strukturierte Sprache, die auf englischen Schlüsselwörtern basiert<sup>4</sup>**

### Grundbaustein

```
SELECT  PNR
FROM    PERS
WHERE   ANR = 'K55'
```



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>

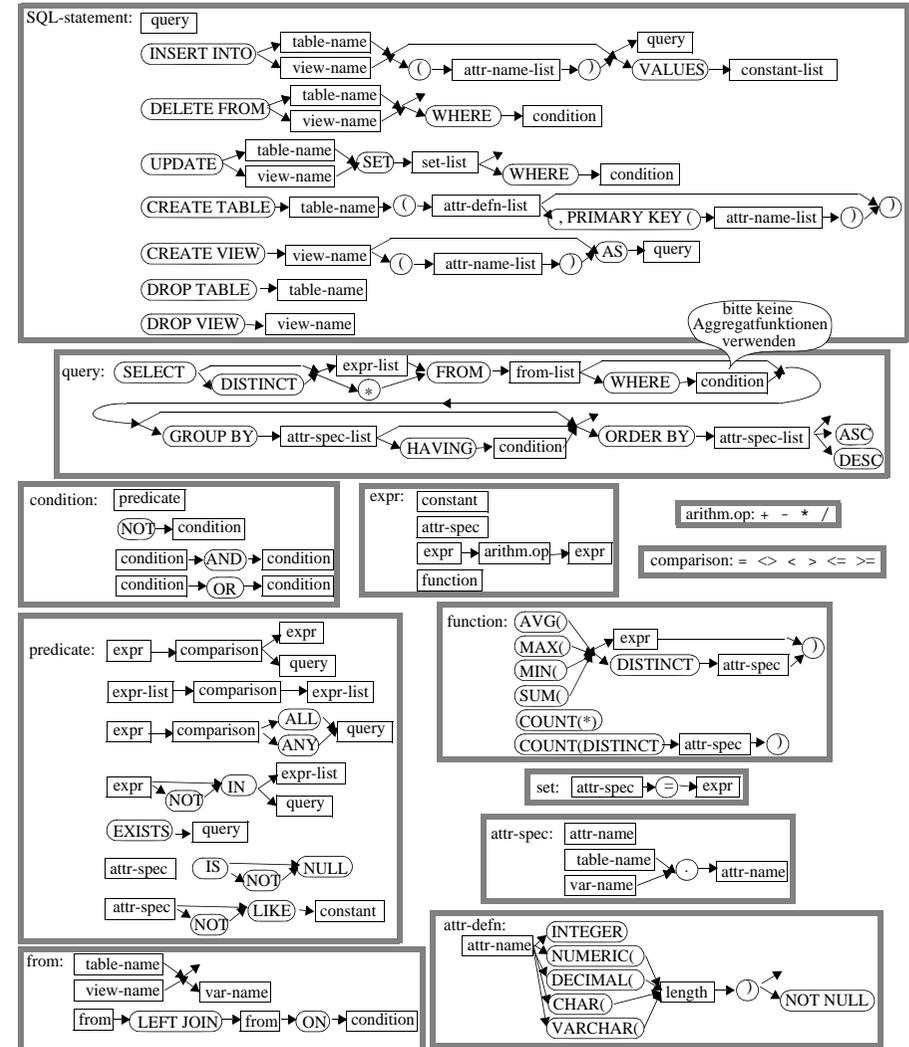
[WHERE Boolescher Prädikatsausdruck]

4. Ausführliche Behandlung in vielen Lehrbüchern, z. B.:  
 Pernul, G., Unland, R.: Datenbanken im Unternehmen — Analyse, Modellbildung und Einsatz, Oldenbourg-Verlag, 2001;  
 Türker, C.: SQL:1999 & SQL:2003, dpunkt.verlag, 2003

## SQL2-Grammatik

- Semantik durch „allgemeine Regeln“ in natürlicher Sprache

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



## Anfragemöglichkeiten in SQL<sup>5</sup>

```

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

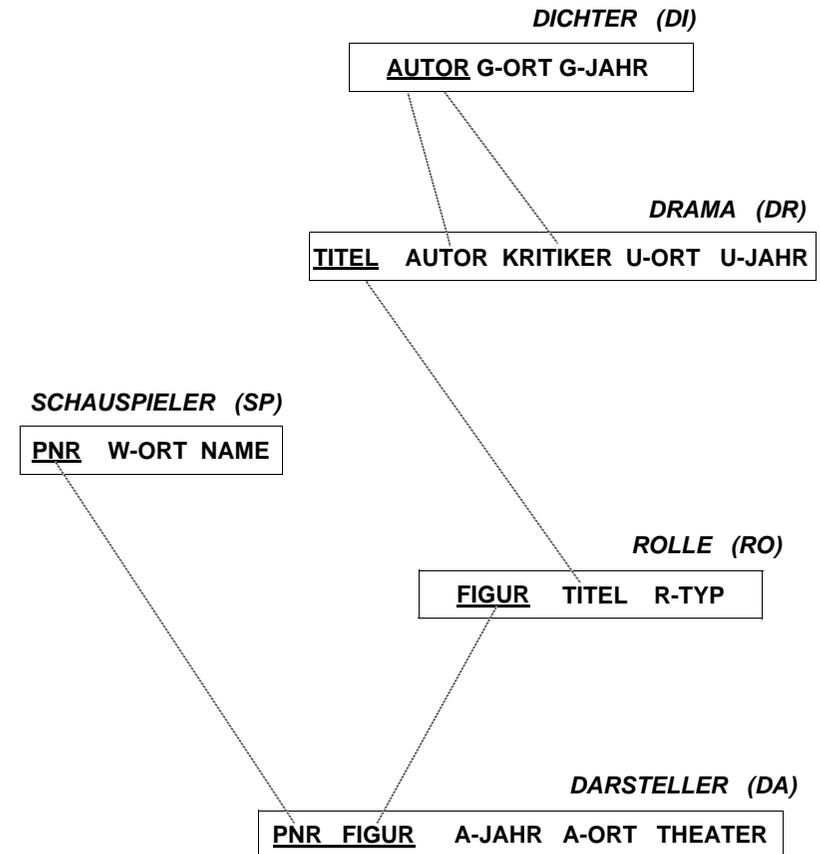
- Mit **SELECT \*** kann das ganze Tupel ausgegeben werden
- **FROM-Klausel spezifiziert das Objekt (Relation, Sicht),** das verarbeitet werden soll (hier durch SELECT)
- **WHERE-Klausel** kann eine Sammlung von Prädikaten enthalten, die mit **AND** und **OR** verknüpft sein können
- **Folgende Prädikate (Verbundterme) sind möglich:**

$$A_i \Theta a_i$$

$$A_i \Theta A_j$$

$$\Theta \in \{=, <>, <, \leq, >, \geq\}$$

## Beispiel-DB: BÜHNE



5. <http://www.sql-und-xml.de/sql-tutorial/>

## Untermengenbildung in einer Relation

**Q1: Welche Dramen von Goethe wurden nach 1800 uraufgeführt?**

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

- **Benennung von Ergebnis-Spalten**

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

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

- Ein Prädikat in einer *WHERE*-Klausel kann ein Attribut auf Zugehörigkeit zu einer Menge testen:

$A_i \text{ IN } (a_1, a_j, a_k)$  explizite Mengendefinition

$A_i \text{ IN } (\text{SELECT } \dots)$  implizite Mengendefinition

**Q2: Finde die Schauspieler (PNR), die Faust, Hamlet oder Wallenstein gespielt haben.**

```
SELECT      PNR
FROM        DARSTELLER
WHERE       FIGUR IN ('Faust', 'Hamlet', 'Wallenstein')
```

- Duplikate in der Ausgabeliste werden nicht eliminiert (Default)
- *DISTINCT* erzwingt Duplikateliminierung

→ Die Menge, die zur Qualifikation herangezogen wird, kann Ergebnis einer geschachtelten Abbildung sein.

## Geschachtelte Abbildung

**Q3: Finde die Figuren, die in Dramen von Schiller oder Goethe vorkommen.**

- innere und äußere Relationen können identisch sein
- eine geschachtelte Abbildung kann beliebig tief sein

## Symmetrische Notation

**Q4: Finde die Figuren und ihre Autoren, die in Dramen von Schiller oder Goethe vorkommen.**

- 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

## Symmetrische Notation (2)

**Q5: 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
```

- Welche Rolle spielen die Bedingungen **A.AUTOR = D.AUTOR** und **D.KRITIKER = B.AUTOR** in der erhaltenen Lösung?

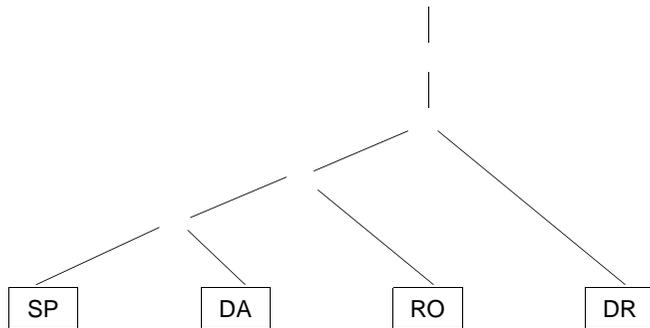
**Q6: Finde die Schauspieler (NAME, W-ORT), die bei in Weimar uraufgeführten Dramen an ihrem Wohnort als 'Held' mitgespielt haben.**

```
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
```

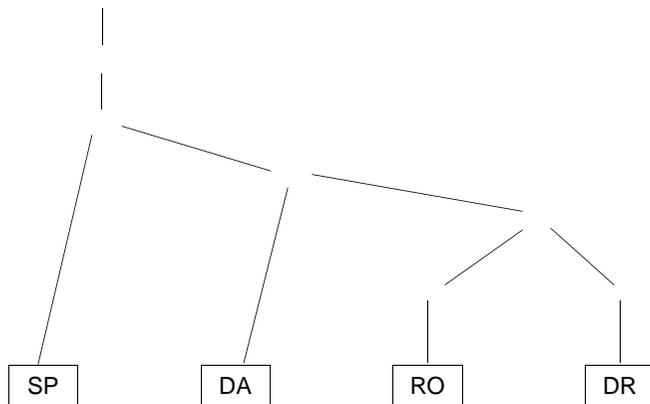
- Wie sieht das Auswertungsmodell (Erklärungsmodell) bei symmetrischer Notation aus?

## Ausführung von SQL-Anweisungen

- Abstraktes Erklärungsmodell für Q6



- Verbessertes Operatorbaum für Q6



- Heuristische Optimierungsregeln

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

## Benutzerspezifizierte Reihenfolge der Ausgabe

ORDER BY order-item-commalist

- Q7: **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)

## Aggregat-Funktionen

```
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

**Q8: Bestimme das Durchschnittsgehalt der Schauspieler, die älter als 50 Jahre sind. (GEHALT und ALTER seien Attribute von SP)**

```
SELECT  
FROM   SCHAUSPIELER  
WHERE  ALTER > 50
```

- **Auswertung**

- Aggregat-Funktion (AVG) wird angewendet auf einstellige Ergebnisliste (GEHALT)
- keine Eliminierung von Duplikaten
- Verwendung von arithmetischen Ausdrücken ist möglich: AVG (GEHALT/12)

## Aggregat-Funktionen (2)

**Q9: An wievielen Orten wurden Dramen uraufgeführt (U-Ort)?**

```
SELECT  
FROM   DRAMA
```

**Q10: An welchen Orten wurden mehr als zwei Dramen uraufgeführt?**

**Versuch:**

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

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

**Q11: Welches Drama (Titel, U-Jahr) wurde zuerst aufgeführt?**

```
SELECT TITEL, U-JAHR  
FROM   DRAMA  
WHERE  U-JAHR =
```

## Partitionierung einer Relation in Gruppen

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

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

**Q12:** Liste alle Abteilungen und das Durchschnittsgehalt ihrer Angestellten auf (Monatsgehalt).

```
SELECT ANR,  
FROM PERS  
GROUP BY ANR
```

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

## Auswahl von Gruppen

```
HAVING cond-exp
```

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

**Q13:** 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  
HAVING
```

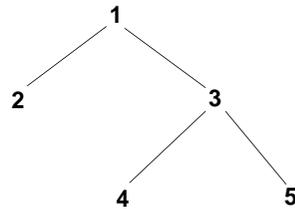
→ Wie sieht ein allgemeines Erklärungsmodell für die Anfrageauswertung aus?

## Hierarchische Beziehung auf einer Relation

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

**Q14: Finde die Angestellten, die mehr als ihre (direkten) Manager verdienen (Ausgabe: NAME, GEHALT, NAME des Managers)**

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



• **Erklärung der Auswertung der Formel**

X.MNR = Y.PNR AND X.GEHALT > Y.GEHALT

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	-		574	May	50 K	-

AUSGABE	X.NAME	X.GEHALT	Y.NAME

## Hierarchische Beziehung auf einer Relation (2)

• **Alternatives Erklärungsmodell für Q14:**  
 Verbund von PERS mit sich selbst und anschließende Selektion

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	-		574	May	50 K	-

**Verbundbedingung:** MNR = PNR'

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

**Selektionsbedingung:** GEHALT > GEHALT'

AUSGABE	NAME	GEHALT	NAME'
	Maier	60 K	Müller
	Müller	55 K	May

## Auswertung von SQL-Anfragen – Erklärungsmodell

1. Die auszuwertenden Relationen werden durch die **FROM**-Klausel bestimmt.  
Aliasnamen erlauben die mehrfache Verwendung derselben Relation
2. Das **Kartesische Produkt** aller Relationen der FROM-Klausel wird gebildet.
3. Tupeln werden ausgewählt durch die **WHERE**-Klausel.
  - Prädikat muss zu „true“ evaluieren
4. Aus den übrig gebliebenen Tupeln werden Gruppen gemäß der **GROUP-BY**-Klausel derart gebildet, dass eine Gruppe aus allen Tupeln besteht, die hinsichtlich aller in der GROUP-BY-Klausel aufgeführten Attribute gleiche Werte enthalten.
5. Gruppen werden ausgewählt, wenn sie die **HAVING**-Klausel erfüllen.
  - Prädikat in der HAVING-Klausel muss zu „true“ evaluieren.
  - Prädikat in der HAVING-Klausel darf sich nur auf Gruppeneigenschaften beziehen (Attribute der GROUP-BY-Klausel oder Anwendung von Aggregat-Funktionen).
6. 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).
7. Die Ausgabereihenfolge wird gemäß der **ORDER-BY**-Klausel hergestellt.
  - Wurde keine ORDER-BY-Klausel angegeben, ist die Ausgabereihenfolge systembestimmt (indeterministisch).

## Erklärungsmodell von SQL-Anfragen – Beispiele

		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
<b>FROM</b>	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
<b>WHERE</b>	B <= 50					
		R''		A	B	C
				Rot	10	10
				Rot	20	10
				Rot	10	20
				Gelb	10	50
				Blau	10	10
				Blau	20	200
<b>GROUP BY</b>	A					
		R'''		A	B	C
				Rot	10	10
				Rot	20	10
				Rot	10	20
				Gelb	10	50
				Blau	10	10
				Blau	20	200
<b>HAVING</b>	MAX(C) > 100					
		R''''		A	SUM(B)	12
				Blau	30	12
<b>SELECT</b>	A, SUM(B), 12					
		R'''''		A	SUM(B)	12
<b>ORDER BY</b>	A			Blau	30	12

## Erklärungsmodell von SQL-Anfragen – Beispiele (2)

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

**Q151:** **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

**Q152:** **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

**Q153:** Die Summe der Gehälter pro Abteilung, in der mindestens ein Mitarbeiter 40 Jahre oder älter ist, soll berechnet werden.

ANR	SUM(GEH)
-----	----------

**Q154:** Warum löst folgende Anfrage nicht Q153?

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

ANR	SUM(GEH)
K45	160K
K56	100K

## Suchbedingungen

- **Sammlung von Prädikaten**

- Verknüpfung mit AND, OR, NOT
- Auswertungsreihenfolge ggf. durch Klammern

- **Nicht quantifizierte Prädikate**

- Vergleichsprädikate  $\Theta$

```
comparison-cond
::= row-constructor  $\Theta$  row-constructor

row-constructor
:= scalar-exp | (scalar-exp-commalist) | (table-exp)
```

- BETWEEN-Prädikate

```
row-constr [NOT] BETWEEN row-constr
AND row-constr
```

Beispiel: GEHALT BETWEEN 80K AND 100K

- IN-Prädikate
- Ähnlichkeitssuche: LIKE-Prädikat
- Behandlung von Nullwerten

- **Quantifizierte Prädikate: ALL, ANY, EXISTS**

- **Weitere Prädikate**

- MATCH-Prädikat für Tupelvergleiche
- UNIQUE-Prädikat zur Bestimmung von Duplikaten

## IN-Prädikate

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

- $x \text{ IN } (a, b, \dots, z) \Leftrightarrow x = a \text{ OR } x = b \dots \text{ OR } x = z$
- $\text{row-constr IN (table-exp)} \Leftrightarrow \text{row-constr} = \text{ANY (table-exp)}$
- $x \text{ NOT IN erg} \Leftrightarrow \text{NOT (x IN erg)}$

**Q16: 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'
```

## Ähnlichkeitssuche

### • Unterstützung der Suche nach Objekten

- von denen **nur Teile des Inhalts** bekannt sind oder
- die einem **vorgegebenen Suchkriterium möglichst nahe** kommen

### • Aufbau einer Maske mit Hilfe zweier spezieller Symbole

% bedeutet „null oder mehr beliebige Zeichen“

\_ bedeutet „genau ein beliebiges Zeichen“

%TENB\_ \_ \_

### • Klassen der Ähnlichkeitssuche

#### 1. Syntaktische Ähnlichkeit (Einsatz von Masken)

LIKE M\_ \_ ER

#### 2. Phonetische Ähnlichkeit (Codierung von Lauten)

? MAIER

#### 3. Semantische Ähnlichkeit (Ontologien, Synonyme, Oberbegriffe, ...)

POLITIKER ~

## LIKE-Prädikate

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

- **Unschärfe 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.

- **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))'
```

## NULL-Werte

- **Attributspezifikation:** Es kann für jedes Attribut festgelegt werden, ob NULL-Werte zugelassen sind oder nicht

- **Verschiedene Bedeutungen**

- Datenwert ist momentan nicht bekannt
- Attributwert existiert nicht für ein Tupel

- **Auswertung von Booleschen Ausdrücken mit einer dreiwertigen Logik**

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

- **Die Auswertung eines NULL-Wertes** in einem Vergleichsprädikat mit irgendeinem Wert ist UNKNOWN (?)

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:  
1234:

**GEH > 70K OR PROV > 50K**

0815:  
1111:  
1234:

➔ Das Ergebnis ? nach vollständiger Auswertung einer WHERE-Klausel wird wie FALSE behandelt

## NULL-Werte (2)

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

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

```
SELECT PNR, GEH + PROV  0815:  
FROM PERS              4711:  
                      ....:
```

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

- **Achtung**  
Im allgemeinen ist

**AVG (GEH) <=> SUM (GEH) / COUNT (PNR)**

- Spezielles Prädikat zum **Test auf NULL-Werte**

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

Beispiel

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

## Quantifizierte Prädikate

### All-or-Any-Prädikate

```
row-constr  $\Theta$  { ALL | ANY | SOME } (table-exp)
```

$\Theta$  **ALL**: Prädikat wird zu „true“ ausgewertet, wenn der  $\Theta$ -Vergleich für alle Ergebniswerte von table-exp „true“ ist

$\Theta$  **ANY** /  $\Theta$  **SOME**:  
analog, wenn der  $\Theta$ -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 werde (Normalfall)

### Semantik

$x \Theta$  ANY (SELECT y FROM T WHERE p)  $\Leftrightarrow$   
EXISTS (SELECT \* FROM T WHERE (p) AND x  $\Theta$  T.y)

$x \Theta$  ALL (SELECT y FROM T WHERE p)  $\Leftrightarrow$   
NOT EXISTS (SELECT \* FROM T WHERE (p) AND NOT (x  $\Theta$  T.y))

## Quantifizierte Prädikate (2)

Q17: Finde die Manager, die mehr verdienen als direkte Untergebenen

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

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	70 K	-		574	May	70 K	-

AUSGABE	M.PNR

Q17a: Finde die Manager, die mehr verdienen als alle direkten Untergebenen

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

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	70 K	-		574	May	70 K	-

AUSGABE	M.PNR

## Quantifizierte Prädikate (3)

Q18: Finde die Namen der Schauspieler, die mindest. einmal gespielt haben

```
SELECT S.NAME
FROM SCHAUSPIELER S
WHERE EXISTS
  (SELECT *
   FROM DARSTELLER D
   WHERE D.PNR = S.PNR)
```

SCHAUSPIELER (PNR, NAME, ...)

S1 ...  
S2  
S3

EXISTS D  
S1 S2 S3

DARSTELLER (PNR, FIGUR, ...)

S1 F  
S2 F  
S2 H  
S2 W  
S3 H

Q18a: Finde die Namen der Schauspieler, die nie gespielt haben

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

SCHAUSPIELER (PNR, NAME, ...)

S1 ...  
S2  
S3

NOT EXISTS D  
S1 S2 S3

DARSTELLER (PNR, FIGUR, ...)

S1 F  
S2 F  
S2 H  
S2 W

## Quantifizierte Prädikate (4)

SCHAUSPIELER (PNR, NAME, ...)

S1 ...			
S2			
S3			
	<b>NOT</b>	<b>EXISTS</b>	<b>R</b>
	S1	S2	S3
	F	T	F

ROLLE (FIGUR, TITEL, ...)

F ...			
H			
W			
	<b>NOT</b>	<b>EXISTS</b>	<b>D</b>
	S1	S2	S3
	F	F	T
	T	F	F
	T	F	T

DARSTELLER (PNR, FIGUR, ...)

S1	F
S2	F
S2	H
S2	W
S3	H

**Q19: 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.

## Es gibt immer viele Möglichkeiten!

**Q20: Finde die Messstation mit der niedrigsten gemessenen Temperatur**

Gegeben:

station (snr, name, ...)	wettert (datum, snr, mintemp, ...)
S1 ...	0101 S1 -373 ...
S2	0101 S2 -284 ...
S3	0201 S1 -175 ...
...	...

In wettert stehen die täglich gemessenen Minimaltemperaturen der verschiedenen Messstationen.<sup>6</sup>

**Gute Lösung:** (Aggregat-Funktion in Subquery)

```
SELECT s.name FROM station s, wettert w
WHERE s.snr=w.snr and w.mintemp=
  (SELECT MIN(ww.mintemp) FROM wettert ww);
```

**Schlechte Lösung:** Keine Joins

```
SELECT name FROM station WHERE snr=(
  SELECT DISTINCT snr FROM wettert WHERE mintemp=(
    SELECT MIN(mintemp) FROM wettert));
```

**Naja, worst case?!**: Keine Aggregat-Funktion

```
SELECT DISTINCT name FROM station
WHERE snr IN (
  SELECT W1.snr FROM wettert W1
  WHERE NOT EXISTS (
    SELECT * FROM wettert W2
    WHERE W2.mintemp < W1.mintemp));
```

6. Zusatz: Die Temperaturen werden als Integer in Zehntelgraden aufgezeichnet. Manche Stationen können bei der Temperatur Nullwerte aufweisen, die als '-2732' (0 Kelvin) (oder als NULL) codiert sind. Bei allen Lösungen fehlt die Behandlung des Nullwertes.

## Auch das ist eine SQL-Anfrage

- Durch Tool zur Entscheidungsunterstützung (*OnLine Analytical Processing, OLAP*) und GUI-Nutzung automatisch erzeugt.

```
select distinct a.fn
from T1 a
where a.owf =
  (select min (b.owf)
   from T1 b
   where (1=1) and (b.aid='SAS' and
    b.fc in (select c.cid
             from T2 c
             where c.cn='HKG') and
    b.tc in (select d.cid
             from T2 d
             where d.cn='HLYD') and
    b.fid in (select e.fid
              from T3 e
              where e.did in
                (select f.did
                 from T4 f
                 where f.dow='saun')) and
    b.fdid in (select g.did
               from T4 g
               where g.dow='saun')))) and
(1=1) and (a.aid='SAS' and
a.fc in (select h.cid
         from T2 h
         where h.cn='HKG') and
a.tc in (select i.cid
         from T2 i
         where i.cn='HLYD') and
a.did in (select j.fid
          from T3 j
          where j.did in
            (select k.did
             from T4 k
             where k.dow='saun')) and
a.fdid in (select l.did
           from T4 l
           where l.dow='saun'))
```

## Möglichkeiten der Datenmanipulation

### Einfügen von Tupeln

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

- M1: Füge den Schauspieler Garfield mit der PNR 4711 ein**  
(satzweises Einfügen)

```
INSERT INTO SCHAUSPIELER (PNR, NAME, W-ORT
VALUES ( 4711, 'Garfield', DEFAULT)
```

- 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.

- M2: Füge die Schauspieler aus KL in die Relation TEMP ein**

```
INSERT INTO TEMP
```

- 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 Tupelmengenaus und kopiert sie in die Zielrelation.
- Die kopierten Tupel sind unabhängig von ihren Ursprungstupeln.

## Löschen von Tupeln durch Suchklauseln

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

- Der Aufbau der WHERE-Klausel entspricht dem in der SELECT-Anweisung

**M3: Lösche den Schauspieler mit der PNR 4711.**

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

**M4: Lösche alle Schauspieler, die nie gespielt haben.**

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

## Ändern von Tupeln durch Suchklauseln

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

**M5: Gib den Schauspielern, die am Pfalztheater spielen, eine Gehaltserhöhung von 5% (Annahme: GEHALT in Schauspieler)**

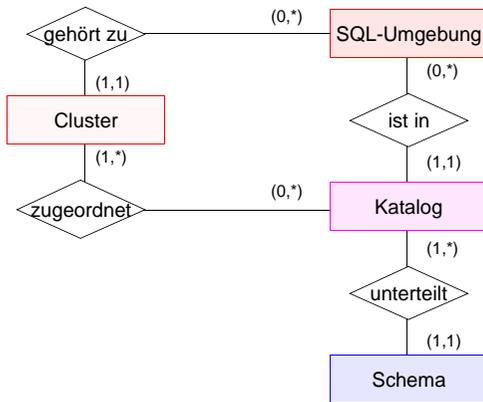
```
UPDATE SCHAUSPIELER S  
SET S.GEHALT = S.GEHALT * 1.05  
WHERE EXISTS  
(SELECT *  
FROM DARSTELLER D  
WHERE D.PNR = S.PNR AND D.THEATER = 'Pfalz')
```

- **Einschränkung (SQL-92 Entry/Intermediate)**

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

## Datendefinition nach SQL

- Was ist alles zu definieren, um eine "leere DB" zu erhalten?
- SQL-Umgebung (environment) besteht aus
  - einer Instanz eines DBMS zusammen mit
  - einer Menge von Daten in Katalogen (als Tabellen organisiert)
  - einer Reihe von Nutzern (authorization identifiers) und Programmen (modules)
- Wichtige Elemente der SQL-Umgebung



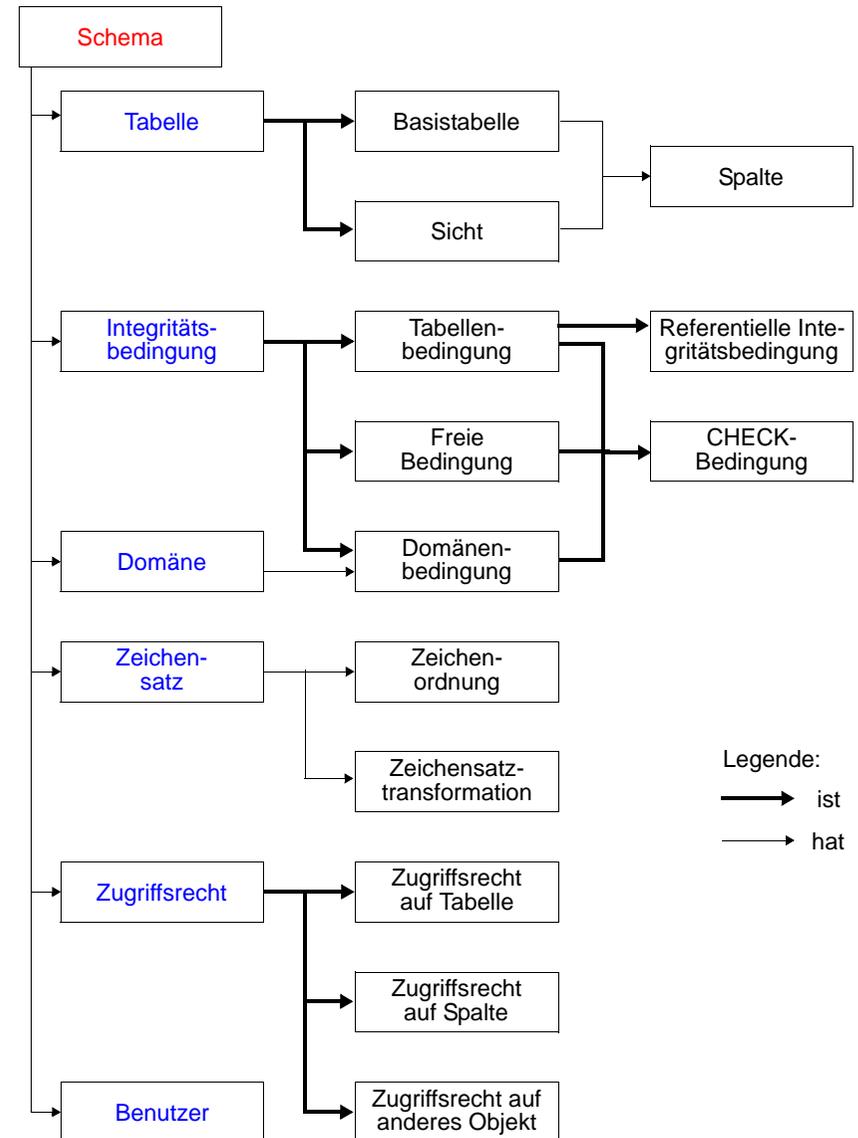
➔ Kataloge bestehen aus SQL-Schemata und können innerhalb einer SQL-Umgebung optional auf ein oder mehrere Cluster<sup>7</sup> verteilt werden

### SQL-Schema

- Katalog kann man als DB (in der DB) ansehen
- SQL-Schemata sind Hilfsmittel zur logischen Klassifikation von Objekten innerhalb einer solchen DB
- Datendefinitionsteil von SQL enthält Anweisungen zum Erzeugen, Verändern und Löschen von Schemaelementen

7. Sinn dieser Clusterbildung ist die Zuordnung von genau einem Cluster zu jeder SQL-Sitzung und dadurch wiederum die Zuordnung einer Menge von Daten bzw. Katalogen zu dieser Sitzung

## Elemente des SQL-Schemas



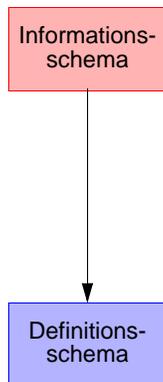
Legende:  
 —▶ ist  
 —▶ hat

## Informations- und Definitionsschema

- Ziel der SQL-Normierung

- möglichst große Unabhängigkeit der DB-Anwendungen von speziellen DBS
- einheitliche Sprachschnittstelle genügt **nicht!**
- Beschreibung der gespeicherten Daten und ihrer Eigenschaften (**Metadaten**) nach einheitlichen und verbindlichen Richtlinien ist genauso wichtig

- Zweischichtiges Definitionsmodell zur Beschreibung der Metadaten<sup>8</sup>



- bietet **einheitliche Sichten** in normkonformen Implementierungen
- ist **für den Benutzer zugänglich** und somit die definierte Schnittstelle zum Katalog

- beschreibt **hypothetische** Katalogstrukturen, also Meta-Metadaten
- erlaubt „Altsysteme“ mit abweichenden Implementierungen normkonform zu wer-

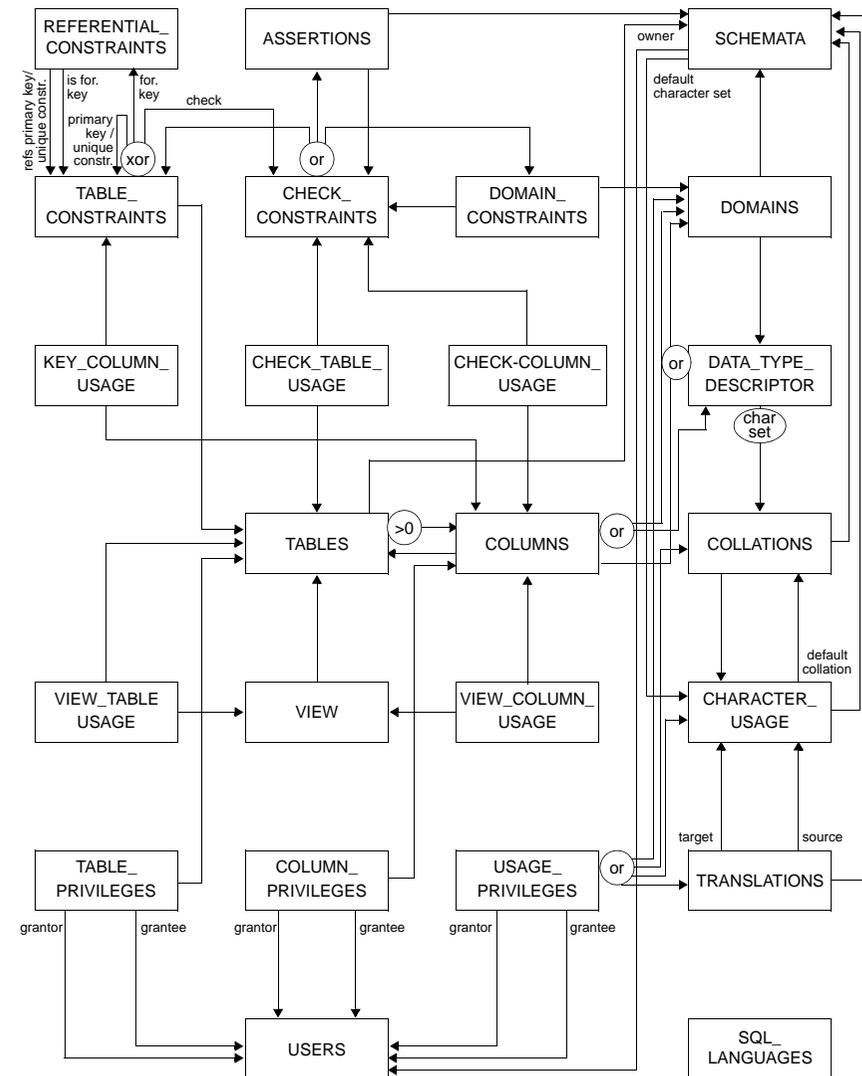
- Welche Meta-Metadaten enthält ein „generisches“ SQL-DBMS?<sup>9</sup>

- DEFINITION\_SCHEMA umfasst 24 Basistabellen und 3 Zusicherungen
- In den Tabellendefinitionen werden ausschließlich 3 Domänen verwendet: SQL\_IDENTIFIER, CHARACTER\_DATA und CARDINAL\_NUMBER

8. Als Definitionsgrundlage für die Sichten des Informationsschemas spezifiziert die SQL-Norm das Definitionsschema, das sich auf ein ganzes Cluster von SQL-Katalogen bezieht und die Elemente aller darin enthaltenen SQL-Schemata beschreibt.

9. Das nicht normkonforme Schema SYSCAT von DB2 enthält 37 Tabellen

## Das Definitionsschema



## Definition von Schemata

- **Anweisungssyntax** (vereinfacht)

```
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

### D1: Benennung des Schemas

- **CREATE SCHEMA Beispiel-DB AUTHORIZATION DB-Admin**

- **Datentypen**

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

## Definition von Wertebereichen

- **Domänen-Konzept zur Festlegung zulässiger Werte**

```
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))

## Definition von Attributen

- Bei der Attributdefinition (column definition) können folgende Angaben spezifiziert werden

- Attributname
- Datentyp bzw. Domain
- Defaultwert sowie Constraints

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

- Beispiele

- PNAME CHAR (30)
- PALTER ALTER (siehe Definition von Domain ALTER)

- Als Constraints können

- Verbot von Nullwerten (NOT NULL)
- Eindeutigkeit (UNIQUE bzw. PRIMARY KEY)
- FOREIGN-KEY-Klausel
- CHECK-Bedingungen definiert werden

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

- Constraint-Namen sind vorteilhaft

- Diagnosehilfe bei Fehlern
- gezieltes Ansprechen bei SET oder DROP des Constraints

## Definition von Attributen (2)

- Beispiel

- Verkaufs\_Preis DECIMAL (9, 2),  
CONSTRAINT Ausverkauf  
CHECK (Verkaufs\_Preis  
<= (SELECT MIN (Preis) FROM Konkurrenz\_Preise))

- Überprüfungszeitpunkt

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

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

- 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 | SET DEFAULT | SET NULL
```

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

## Erzeugung von Basisrelationen

```
CREATE TABLE base-table
  (base-table-element-commalist)
```

```
base-table-element
  ::= column-def | base-table-constraint-def
```

### • Definition einer Relation

- Definition aller zugehörigen Attribute mit Typspezifikation
- Spezifikation aller Integritätsbedingungen (Constraints)

### D2: Erzeugung der neuen Relationen PERS und ABT

#### 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)

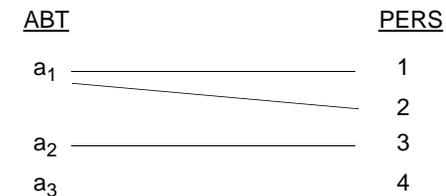
#### CREATE TABLE ABT

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

↳ **Wie kann ANZAHL-ANGEST überprüft werden?**

## Abbildung von Beziehungen

### • Beispiel: Darstellung einer (1:n)-Beziehung



### • Abbildungsversuch (FS auf welche Seite?)

ABT (ABTNR, PNR, ...)      PERS (PNR, ...)

### • Abbildung im Relationenmodell

ABT (ABTNR, ...)      PERS (PNR, ANR, ...)

## Abbildung von Beziehungen (2)

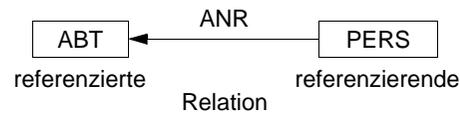
- ER-Diagramm: (1:n)-Beziehung



- Umsetzung ins Relationenmodell

ABT (ABTNR ..., ... PRIMARY KEY (ABTNR))	PERS (PNR ..., ANR ..., PRIMARY KEY (PNR), FOREIGN KEY (ANR) REFERENCES ABT)
--	---

- Referenzgraph



- Zusätzliche Einschränkungen

Jeder Angestellte (PERS) muss in einer Abteilung beschäftigt sein ([1,1]).

→ PERS.ANR ... NOT NULL

Jeder Abteilung (ABT: [0,1]) darf höchstens einen Angestellten beschäftigen.

→ PERS.ANR ... UNIQUE

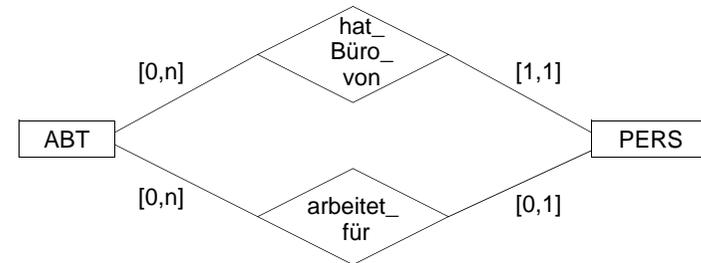
- Bemerkung

In SQL2 kann nicht spezifiziert werden, dass ein Vater einen Sohn haben muss, z. B. [1,n]. Die Anzahl der Söhne lässt sich nicht einschränken (außer [0,1]).

- Vorschlag für späteren Standard: PENDANT-Klausel, mit welcher der Fall [1,n] abgedeckt werden kann.
- Bei der Erstellung müssen solche Beziehungen verzögert überprüft werden.

## Abbildung von Beziehungen (3)

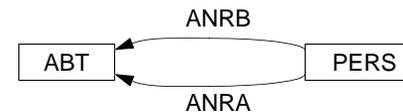
- ER-Diagramm



- Umsetzung ins Relationenmodell

ABT (ABTNR ..., ... PRIMARY KEY (ABTNR))	PERS (PNR ..., ANRA ..., ANRB... NOT NULL,  PRIMARY KEY (PNR), FOREIGN KEY (ANRA) REFERENCES ABT, FOREIGN KEY (ANRB) REFERENCES ABT)
--	--

- Referenzgraph

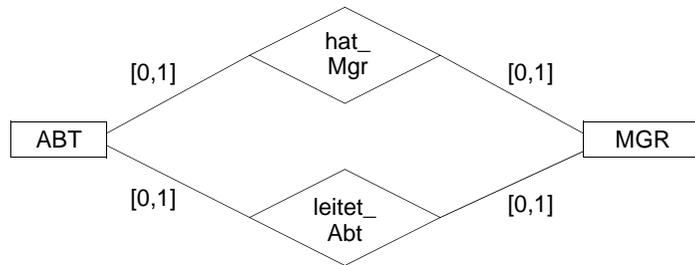


- Bemerkung

- Für jede FS-Beziehung benötigt man ein separates FS-Attribut.
- Mehrere FS-Attribute können auf dasselbe PS/SK-Attribut verweisen.

## Abbildung von Beziehungen (4)

- Ziel: Darstellung einer symmetrischen (1:1)-Beziehung
- Erster Versuch: ER-Diagramm



- Umsetzung ins Relationenmodell

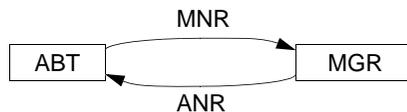
ABT (ANR ..., MNR ... UNIQUE, ... PRIMARY KEY (ANR), FOREIGN KEY (MNR) REFERENCES MGR)	MGR (MNR ..., ANR ...UNIQUE, ... PRIMARY KEY (MNR), FOREIGN KEY (ANR) REFERENCES ABT)
---	--

↳ Es sind alternative Lösungen möglich

- Zusätzliche Einschränkungen

- Jede Abteilung hat einen Manager → ABT.MNR ... UNIQUE NOT NULL
- Jeder Manager leitet eine Abteilung → MGR.ANR ... UNIQUE NOT NULL

- Referenzgraph



↳ Kann durch die beiden (n:1)-Beziehungen eine symmetrische (1:1)-Beziehung ausgedrückt werden?

## Abbildung von Beziehungen (5)

- Beispiel: Darstellung einer (1:1)-Beziehung

<u>ABT</u>	<u>MGR</u>
a <sub>1</sub>	1
a <sub>2</sub>	2
a <sub>3</sub>	3
a <sub>4</sub>	4

- Versuch

ABT (ABTNR, MNR, ...)      PERS (MNR, ABTNR, ...)

- Abbildung im Relationenmodell

ABT (ABTNR, MNR, ...)      PERS (MNR, ...)

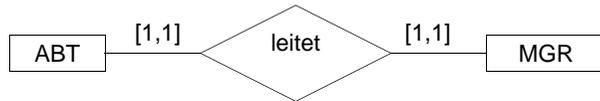
- Abbildung im Relationenmodell

(Variation über Schlüsselkandidaten)

ABT (ABTNR, MNR, ...)      PERS (SVNR, MNR, ...)

## Abbildung von Beziehungen (6)

- ER-Diagramm: Symmetrische (1:1)-Beziehung



- Umsetzung ins Relationenmodell

ABT (ANR ..., MNR ... UNIQUE NOT NULL, ... PRIMARY KEY (ANR), FOREIGN KEY (MNR) REFERENCES MGR)	MGR (MNR ..., ... PRIMARY KEY (MNR), FOREIGN KEY (MNR) REFERENCES ABT(MNR))
--	---

↳ Es sind alternative Lösungen möglich

- Referenzgraph



- Die Nutzung des MNR-Attributes für beide FS-Beziehungen gewährleistet hier die Einhaltung der (1:1)-Beziehung
- Der Fall ([0,1], [0,1]) ist so nicht darstellbar

- Variation über Schlüsselkandidaten

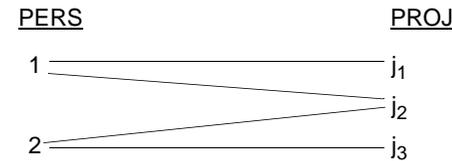
ABT (ANR ..., MNR ... UNIQUE, ... PRIMARY KEY (ANR), FOREIGN KEY (MNR) REFERENCES MGR(MNR))	MGR (SVNR ..., MNR ... UNIQUE, ... PRIMARY KEY (SVNR) FOREIGN KEY (MNR) REFERENCES ABT(MNR))
--	---

- Die Nutzung von Schlüsselkandidaten mit der Option NOT NULL erlaubt die Darstellung des Falles ([1,1], [1,1])
- Alle Kombinationen mit [0,1] und [1,1] sind möglich

↳ Es sind alternative Lösungen möglich

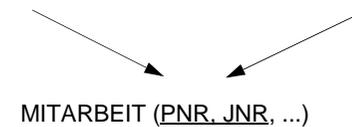
## Abbildung von Beziehungen (7)

- Beispiel: Darstellung einer (n:m)-Beziehung



- Abbildung im Relationenmodell

PERS (PNR, ...)      PROJ (JNR, ...)



- Direkte (n:m)-Abbildung?

PERS (PNR, JNR, ...)      PROJ (JNR, PNR, ...)

## Abbildung von Beziehungen (8)

- ER-Diagramm: (n:m)-Beziehung



- Umsetzung ins Relationenmodell

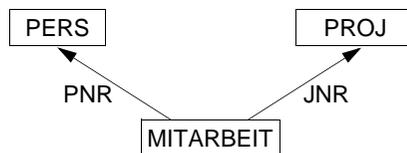
PERS (PNR ...,  
 ...  
 PRIMARY KEY (PNR))

PROJ (JNR ...,  
 ...  
 PRIMARY KEY (JNR))

MITARBEIT (PNR ...,  
 JNR ...,  
 PRIMARY KEY (PNR,JNR),  
 FOREIGN KEY (PNR) REFERENCES PERS,  
 FOREIGN KEY (JNR) REFERENCES PROJ)

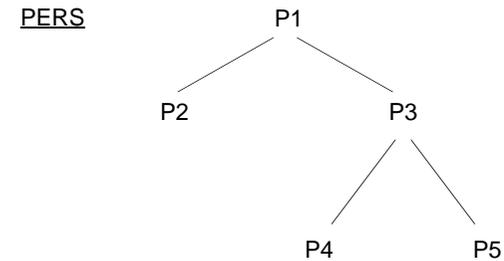
- Diese Standardlösung erzwingt „Existenzabhängigkeit“ von MITARBEIT. Soll dies vermieden werden, dürfen die Fremdschlüssel von MITARBEIT nicht als Teil des Primärschlüssels spezifiziert werden.
- Ist die Realisierung von [1,n] oder [1,m] bei der Abbildung der (n:m)-Beziehung möglich?

- Referenzgraph



## Abbildung von Beziehungen (9)

- Beispiel: Darstellung einer (1:n)-Beziehung als Selbstreferenz



- Mögliche Abbildung (Redundanz!)

PERS' (PNR, ...)

HAT\_MGR (PNR, MNR...)

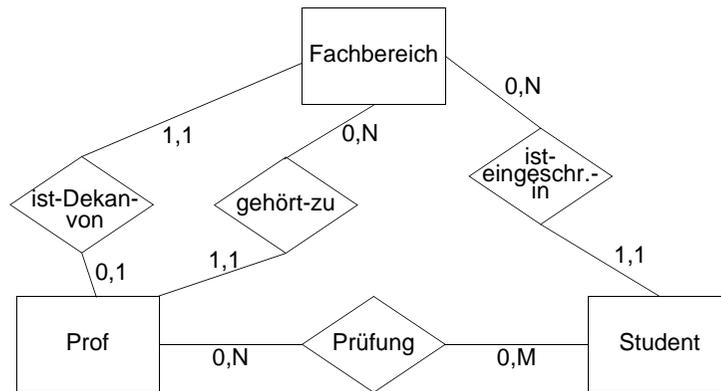
- Abbildung im Relationenmodell

PERS (PNR, ..., MNR)

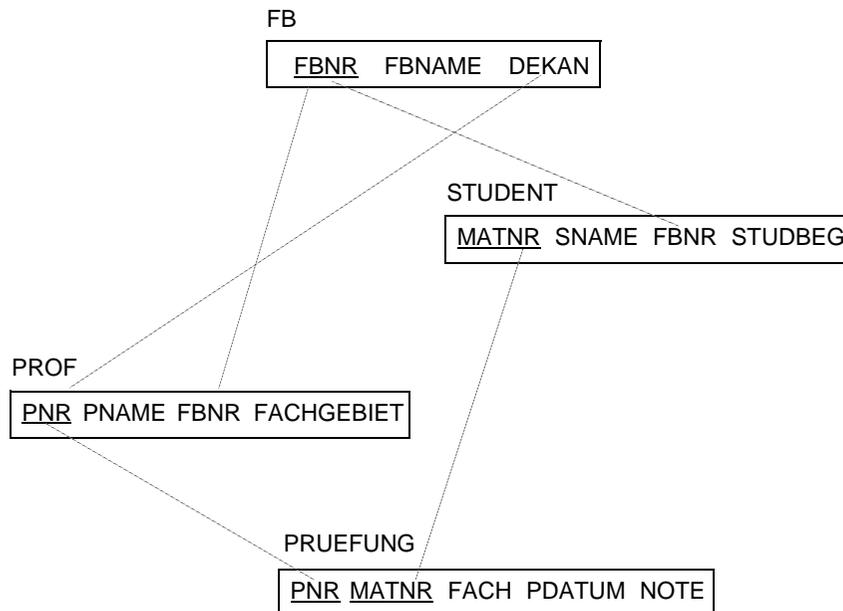


## Beispiel-Miniwelt

- ER-Diagramm



- Graphische Darstellung des Relationenschemas



## Spezifikation des relationalen DB-Schemas (nach dem SQL2-Standard)

### Wertebereiche

CREATE DOMAIN	FACHBEREICHSNUMMER	AS	CHAR (4)
CREATE DOMAIN	FACHBEREICHSNAMEN	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

### Relationen

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

```
CREATE TABLE PROF (
    PNR          PERSONALNUMMER        PRIMARY KEY,
    PNAME        NAMEN                 NOT NULL,
    FBNR         FACHBEREICHSNUMMER    NOT NULL,
    FACHGEBIET   FACHBEZEICHNUNG,
    CONSTRAINT   PFK1  FOREIGN KEY (FBNR)
                REFERENCES FB (FBNR)
                ON UPDATE CASCADE
                ON DELETE SET DEFAULT)
```

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

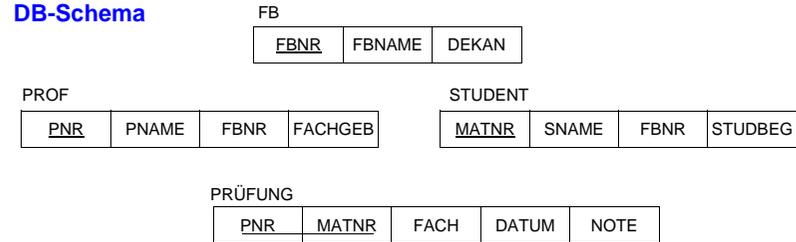
## Spezifikation des relationalen DB-Schemas (Fortsetzung)

```
CREATE TABLE STUDENT (
  MATNR      MATRIKELNUMMER      PRIMARY KEY,
  SNAME      NAMEN                NOT NULL,
  FBNR       FACHBEREICHSNUMMER  NOT NULL,
  STUDBEG    DATUM,
  CONSTRAINT SFK FOREIGN KEY (FBNR)
               REFERENCES FB (FBNR)
               ON UPDATE CASCADE
               ON DELETE RESTRICT)
```

```
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)
```

## Darstellung des „Inhalts“ der Miniwelt in Relationen

### DB-Schema



### Ausprägungen

FB	<u>FBNR</u>	FBNAME	DEKAN
	FB 9	WIRTSCHAFTSWISS	4711
	FB 5	INFORMATIK	2223

PROF	<u>PNR</u>	PNAME	FBNR	FACHGEB
	1234	HÄRDER	FB 5	DATENBANKSYSTEME
	5678	WEDEKIND	FB 9	INFORMATIONSSYSTEME
	4711	MÜLLER	FB 9	OPERATIONS RESEARCH
	6780	NEHMER	FB 5	BETRIEBSSYSTEME

STUDENT	<u>MATNR</u>	SNAME	FBNR	STUDBEG
	123 766	COY	FB 9	1.10.05
	225 332	MÜLLER	FB 5	15.04.02
	654 711	ABEL	FB 5	15.10.04
	226 302	SCHULZE	FB 9	1.10.05
	196 481	MAIER	FB 5	23.10.05
	130 680	SCHMID	FB 9	1.04.07

PRUEFUNG	<u>PNR</u>	<u>MATNR</u>	FACH	PDATUM	NOTE
	5678	123 766	BWL	22.10.07	4
	4711	123 766	OR	16.01.08	3
	1234	654 711	DV	17.04.08	2
	1234	123 766	DV	17.04.07	4
	6780	654 711	SP	19.09.07	2
	1234	196 481	DV	15.10.07	1
	6780	196 481	BS	23.10.07	3

## Wartung von Beziehungen

### • Relationale Invarianten

1. **Primärschlüsselbedingung:** Eindeutigkeit, keine Nullwerte!
2. **Fremdschlüsselbedingung:** Zugehöriger PS (SK) muss existieren

### • Welche PROBLEME sind zu lösen?

#### 1. Operationen in der Sohn-Relation

- a) Einfügen eines Sohn-Tupels
- b) Ändern des FS in einem Sohn-Tupel
- c) 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

#### 2. Operationen in der Vater-Relation

- a) Löschen eines Vater-Tupels
- b) Ändern des PS/SK in einem Vater-Tupel
- c) 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

#### 3. Wie geht man mit NULL-Werten um?

- Dreiwertige Logik verwirrend: T, F, ?
- Vereinbarung: NULL ≠ NULL (z. B. beim Verbund)
- bei Operationen: Ignorieren von NULL-Werten

→ Spezielle Semantiken von NULL-Werten erforderlich

## Wartung der referentiellen Integrität

### • SQL2-Standard führt „referential actions“ ein

- genauere Spezifikation der referentiellen Aktionen
- für jeden Fremdschlüssel (FS) separat festzulegen

#### 1. Sind „Nullen“ verboten?

NOT NULL

#### 2. Löschregel für Zielrelation (referenzierte Relation)

ON DELETE

{NO ACTION | CASCADE | SET NULL | SET DEFAULT | RESTRICT<sup>10</sup>}

#### 3. Änderungsregel für Ziel-Primärschlüssel (PS oder SK)

ON UPDATE

{NO ACTION | CASCADE | SET NULL | SET DEFAULT | RESTRICT}

- **NO ACTION** Sie ist die Defaulteinstellung. 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
- **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
- **RESTRICT** Operation wird nur ausgeführt, wenn keine zugehörigen Sätze (FS-Werte) vorhanden sind

10. Die Option RESTRICT kam in SQL:1999 neu hinzu; sie ist restriktiver als NO ACTION, da eine integritätsverletzende Anweisung sofort zurückgewiesen wird

## Auswirkungen referentieller Aktionen

### Referentielle Aktionen

ON DELETE {CASCADE | RESTRICT | SET NULL | SET DEFAULT | NO ACTION}  
 ON UPDATE {CASCADE | RESTRICT | SET NULL | SET DEFAULT | NO ACTION}

### 1. Isolierte Betrachtung von STUDENT – FB



### Beispiel-DB

FB	<u>FBNR</u>	FBNAME
FB9		WIRTSCHAFTSWISS
FB5		INFORMATIK

STUDENT	<u>MATNR</u>	SNAME	FBNR
	123 766	COY	FB 9
	225 332	MÜLLER	FB 5
	654 711	ABEL	FB 5
	226 302	SCHULZE	FB 9

### Operationen

- Lösche FB (FBNR=FB5)
- Ändere FB ((FBNR=FB9) → (FBNR=FB10))

### Referentielle Aktionen

- DC, DSN, DSD, DR, DNA
- UC, USN, USD, UR, UNA

### Eindeutigkeit der Operationen?

## Auswirkungen referentieller Aktionen (2)

### 2. Isolierte Betrachtung von STUDENT – PRUEFUNG – PROF



### Beispiel-DB

PROF	<u>PNR</u>	PNAME	STUDENT	<u>MATNR</u>	SNAME
	1234	HÄRDER		123 766	COY
	4711	MÜLLER		654 711	ABEL

PRÜFUNG	<u>PNR</u>	<u>MATNR</u>	FACH
	4711	123 766	OR
	1234	654 711	DV
	1234	123 766	DV
	4711	654 711	OR

### Einsatz von

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

### Auswirkungen von Aktualisierungsoperationen

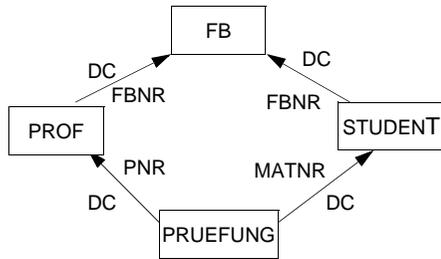
- Verträglichkeit der Referentiellen Aktionen

Student \ Prof	DC	DR	UC	UR
DC				
DR				
UC				
UR				

- Unabhängige referentielle Beziehungen können unabhängig definiert und gewartet werden

### Auswirkungen referentieller Aktionen (3)

#### 3. Vollständiges Beispiel



- Lösche FB (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

➔ Ergebnis der Operation unabhängig von der Reihenfolge der referentiellen Aktionen

➔ Eindeutigkeit des erreichten DB-Zustandes

- Was heißt Unabhängigkeit der referentiellen Beziehungen?

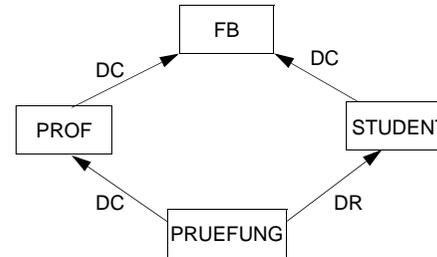
- Es sind mehrere Kombinationen von referentiellen Aktionen möglich: z. B. DSD, UC oder DC, USN

- Eindeutigkeit bei allen Aktualisierungsoperationen

➔ sicheres Schema

### Auswirkungen referentieller Aktionen (4)

#### 4. Modifikation des Schemas



- Lösche FB (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

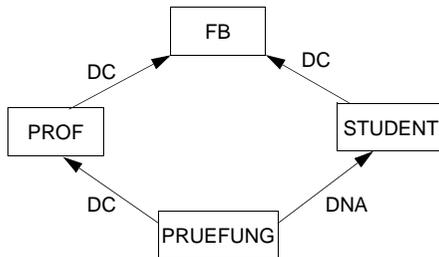
- Löschen in FB
- Löschen in STUDENT
- Zugriff auf PRUEFUNG
- Wenn ein gerade gelöschter Student eine Prüfung abgelegt hatte
- Rücksetzen
- sonst:
- Löschen in PROF
- Löschen in PRUEFUNG

➔ Es können reihenfolgenabhängige Ergebnisse auftreten!

➔ Die Reihenfolgenabhängigkeit ist hier wertabhängig

## Auswirkungen referentieller Aktionen (5)

### 5. Weitere Modifikation des Schemas



#### • Lösche FB (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

- 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

## Eindeutigkeit referentieller Aktionen

### • Aufgabe

Verhinderung von mehrdeutigen DB-Operationen

### • Maßnahmen

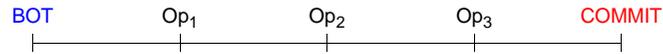
- Statische Schemaanalyse zur Feststellung sicherer DB-Schemata
  - ➔ nur bei einfach strukturierten Schemata effektiv
  - ➔ bei wertabhängigen Konflikten zu restriktiv (konfliktrichtige Schemata)
  - ➔ Hohe Komplexität der Analysealgorithmen
- Dynamische Überwachung der Modifikationsoperationen
  - ➔ hoher Laufzeitaufwand

### • Vorgehensweise

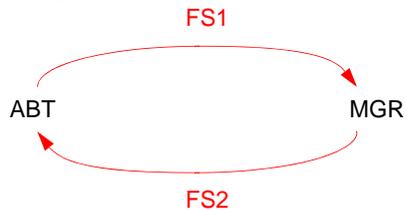
1. Falls Sicherheit eines Schemas festgestellt werden kann, ist keine Laufzeitüberwachung erforderlich
2. Alternative Möglichkeiten zur Behandlung konfliktträchtiger Schemata
  - a) Sie werden verboten:
    - Statische Schemaanalyse kann Sicherheit eines Schemas nicht feststellen. Dabei sind ggf. pessimistische Annahmen zu treffen, je nachdem, ob bei der Analyse nur Relationen oder auch ihre Attribute (Attributkonflikte) betrachtet werden.
  - b) Sie werden erlaubt:
    - Die referentiellen Aktionen werden bei jeder Operation dynamisch überwacht
    - Falls ein Konflikt erkannt wird, wird die Operation zurückgesetzt

## Durchführung der Änderungsoperationen

- Prüfung der referentiellen Integrität (IMMEDIATE/DEFERRED)



- Bei zyklischen Referenzpfaden



- wenigstens ein Fremdschlüssel im Zyklus muss „NULL“ erlauben **oder**
- Prüfung der referentiellen Integrität muss verzögert (DEFERRED) werden (z. B. bei COMMIT)

- Durchführung der referentiellen Aktionen (RA)

- Benutzeroperationen (Op) sind in SQL immer *atomic*
- mengenorientiertes oder satzorientiertes Verarbeitungsmodell



- IMMEDIATE-Bedingungen müssen erfüllt sein an Anweisungsgrenzen (→ **mengenorientierte Änderung**)

## Beispiel – Data Warehousing

```

Create Table Produkt
(Pnr      P-Nummer  (*Domaindef.*),
 Bezeichnung Char(40),
 ...)
Primary Key (Pnr)

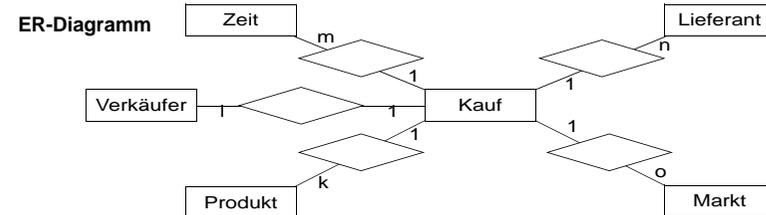
Create Table Lieferant
(Lnr      L-Nummer  (*Domaindef.*),
 LName    Char(20),
 Ort      ... ,
 ...)
Primary Key (Lnr)

Create Table Verkäufer
(Vnr      V-Nummer  (*Domaindef.*),
 VName    Char(20),
 ...)
Primary Key (Vnr)

Create Table Markt
(Mnr      M-Nummer  (*Domaindef.*),
 Adresse  ... ,
 ...)
Primary Key (Mnr)

Create Table Zeit
(Znr      Z-Nummer  (*Domaindef.*),
 Datum    Date      NOT NULL,
 Besonderheit Char(80)
 ...)
Primary Key (Znr)
    
```

- Kauf als Entity mit unabhängigen Beziehungen



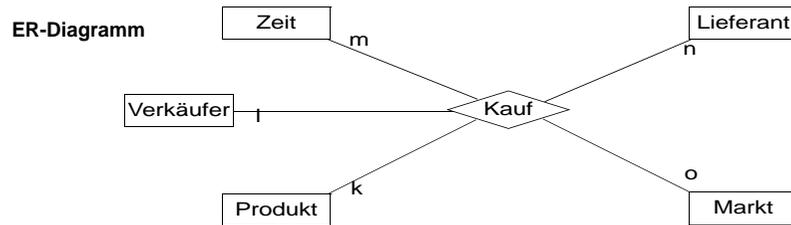
```

Create Table Kauf
(Knr      K-Nummer          (* Unique Not Null*),
 Pnr      P-Nummer          (*siehe Domaindefinition*),
 Vnr      V-Nummer          (*siehe Domaindefinition*),
 Znr      Z-Nummer          (*siehe Domaindefinition*),
 Lnr      L-Nummer          (*siehe Domaindefinition*),
 Mnr      M-Nummer          (*siehe Domaindefinition*),
 Menge    Integer           NOT NULL,
 Preis    Money             NOT NULL,
 ...)

Primary Key (Knr),
Foreign Key (Pnr)      References      Produkt
ON DELETE SET DEFAULT (oder SET NULL),
Foreign Key (Vnr)      References      Verkäufer
ON DELETE SET DEFAULT (oder SET NULL),
Foreign Key (Znr)      References      Zeit
ON DELETE SET DEFAULT (oder SET NULL),
Foreign Key (Lnr)      References      Lieferant
ON DELETE SET DEFAULT (oder SET NULL),
Foreign Key (Mnr)      References      Markt
ON DELETE SET DEFAULT (oder SET NULL)
    
```

## Beispiel – Data Warehousing (2)

- Kauf als 5-stellige Beziehung



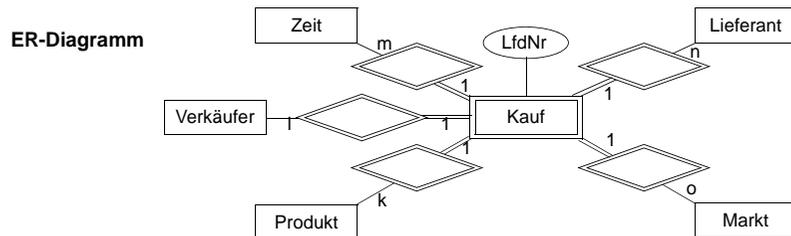
**Create Table Kauf**

(Pnr	P-Nummer	(*siehe Domaindefinition*),
Vnr	V-Nummer	(*siehe Domaindefinition*),
Znr	Z-Nummer	(*siehe Domaindefinition*),
Lnr	L-Nummer	(*siehe Domaindefinition*),
Mnr	M-Nummer	(*siehe Domaindefinition*),
Menge	Integer	<b>Not Null,</b>
Preis	Money	<b>Not Null,</b>

**Primary Key** (Pnr, Vnr, Znr, Lnr, Mnr),

<b>Foreign Key</b> (Pnr)	References	Produkt	<b>ON DELETE CASCADE,</b>
<b>Foreign Key</b> (Vnr)	References	Verkäufer,	<b>ON DELETE CASCADE,</b>
<b>Foreign Key</b> (Znr)	References	Zeit,	<b>ON DELETE CASCADE,</b>
<b>Foreign Key</b> (Lnr)	References	Lieferant,	<b>ON DELETE CASCADE,</b>
<b>Foreign Key</b> (Mnr)	References	Markt	<b>ON DELETE CASCADE)</b>

- Kauf als existenzabhängiges Entity mit 5 begründ. binären Beziehungen



Existenzabhängigkeit wird über den Primärschlüssel von Kauf ausgedrückt:  
Er besteht aus LfdNr und von den konstituierenden Entities ererbten Schlüsselteilen

## Beispiel – Data Warehousing (3)

- Kauf als existenzabhängiges Entity mit 5 begründenden binären Beziehungen (Forts.)

**Create Table Kauf**

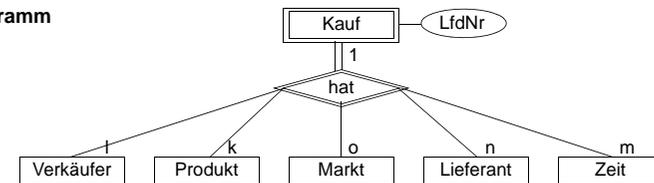
(LfdNr	Integer,	
(Pnr	P-Nummer	(*siehe Domaindefinition*),
Vnr	V-Nummer	(*siehe Domaindefinition*),
Znr	Z-Nummer	(*siehe Domaindefinition*),
Lnr	L-Nummer	(*siehe Domaindefinition*),
Mnr	M-Nummer	(*siehe Domaindefinition*),
Menge	Integer	<b>Not Null,</b>
Preis	Money	<b>Not Null,</b>

**Primary Key** (LfdNr, Pnr, Vnr, Znr, Lnr, Mnr),

<b>Foreign Key</b> (Pnr)	References	Produkt	<b>ON DELETE CASCADE,</b>
<b>Foreign Key</b> (Vnr)	References	Verkäufer	<b>ON DELETE CASCADE,</b>
<b>Foreign Key</b> (Znr)	References	Zeit	<b>ON DELETE CASCADE,</b>
<b>Foreign Key</b> (Lnr)	References	Lieferant	<b>ON DELETE CASCADE,</b>
<b>Foreign Key</b> (Mnr)	References	Markt	<b>ON DELETE CASCADE)</b>

- Kauf als existenzabhängiges Entity mit einer begr. 6-stelligen Beziehung

ER-Diagramm



**Create Table Kauf**

(LfdNr	Integer,	
Pnr	P-Nummer	(*siehe Domaindefinition*),
Vnr	V-Nummer	(*siehe Domaindefinition*),
Znr	Z-Nummer	(*siehe Domaindefinition*),
Lnr	L-Nummer	(*siehe Domaindefinition*),
Mnr	M-Nummer	(*siehe Domaindefinition*),
Menge	Integer	<b>Not Null,</b>
Preis	Money	<b>Not Null,</b>

**Primary Key** (LfdNr, Pnr, Vnr, Znr, Lnr, Mnr)

<b>Foreign Key</b> (Pnr)	References	Produkt	<b>ON DELETE CASCADE,</b>
<b>Foreign Key</b> (Vnr)	References	Verkäufer	<b>ON DELETE CASCADE,</b>
<b>Foreign Key</b> (Znr)	References	Zeit	<b>ON DELETE CASCADE,</b>
<b>Foreign Key</b> (Lnr)	References	Lieferant	<b>ON DELETE CASCADE,</b>
<b>Foreign Key</b> (Mnr)	References	Markt	<b>ON DELETE CASCADE)</b>

➔ Beide existenzabhängigen Modellierungen unterscheiden sich in SQL nicht!

# Zusammenfassung

- **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

- **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.

# SQL als richtungsweisender DB-Standard

- **Standardisierungsprozess<sup>11</sup>**

- Teilnehmer: DB-Hersteller und Anwender, mehr als 20 Länder, ANSI
- Konsens zwischen Teilnehmern wird angestrebt
- Entwicklungsrichtungen: objekt-relationale DBS, SQL/XML, ...

- **SQL:1999 hat viele Teile**

- SQL/Foundation (Part 2), SQL/CLI (Part 3), SQL/PSM (Part 4)
- SQL/Language Bindings (Part 5), **SQL/MED** (Mgmt. of External Data) (Part 9)
- SQL Object Language Bindings (Part 10)
- SQL/Schemata (Part11), SQL/JRT (Part 13), SQL/XML (Part 14), . . .

- **Weiterer auf SQL:1999 aufbauender Standard:**

- SQL Multimedia and Application Packages (SQL/MM)**

- Framework, Full-Text
- Spatial, Still Image
- Data Mining

---

11. Information Technology – Database Language SQL - Part 1 and Part 2: Framework (for SQL:200n) and Foundation (SQL:200n), International Standard, Januar 2009 ([www.jtc1sc32.org](http://www.jtc1sc32.org))  
Information Technology – Database Language SQL - Technical Corrigendum xxx for SQL:1999, ... [2010-02-05: >1960 Dokumente](#)  
<http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html>

## DBS-Markt

Daten- strukturen	komplex	OO $n \cdot 10^8$ \$	OR ?
	einfach	Dateisysteme Video-Server	RM (SQL) $k \cdot 10^{10}$ \$
		einfach	komplex
		Anfragen	

### • Einfache Daten, einfache Anfragen

- Datenstruktur ist dem System nicht bekannt
- Künftig werden solche Systeme wahrscheinlich mit Anfragemöglichkeiten (z. B. SQL) ausgestattet

### • Einfache Daten, komplexe Anfragen

- RDBS: skalierbar, robust, Zugriff über Struktur und Inhalt
- Begrenzte Unterstützung für komplexe, als BLOBs gespeicherte Daten
- RDBS können diese BLOBs nicht indexieren, manipulieren oder über ihren Inhalt suchen

### • Komplexe Daten, einfache Anfragen

- Persistente komplexe Objekte, die durch Java, C++, Smalltalk, ... manipuliert werden
- Begrenzte Skalierbarkeit in Bezug auf große Datenvolumina und große Anzahlen von Benutzer

### • Komplexe Daten, komplexe Anfragen

- OR-Server können komplexe Daten als Objekte handhaben
- Benutzerdefinierte Funktionen lassen sich zur Manipulation der Daten im Server heranziehen
- Erweiterbarkeit ist für Datentypen und Funktionen möglich

## Objekt-Relationale DBS – Entwicklungstrend

Daten- strukturen	komplex	OODBS	Universal Server
	einfach	Dateisysteme Video-Server	
		einfach	komplex
		Anfragen	

### • DBS, die VITA-Daten (*Video, Image, Text, Audio*) handhaben können,

- werden auch **Universal Server** genannt;
- viele Erweiterungen (*spatial types, time series, ...*) werden laufend entwickelt

### • Erweiterbare DBS erfordern erweiterbare Konzepte

- Integration von AW-Funktionen (in 3GL) in den DB-Server (Weiterentwicklung des Konzeptes der Stored Procedures)
- Benutzung einer CALL-Schnittstelle oder von eingebettetem SQL
- 4G-Sprachen (z. B. NewEra) lassen sich erweitern mit C++ und OLE
- Plattformunabhängigkeit lässt sich durch Web-Applikationen erzielen
- Java-Client-Applikationen können für sich Anwendungscode in Form von Java-Applets aus dem Web laden
  - Java-Applets werden in Intranets oder im Internet gespeichert
  - Applets laufen als Client-Applikationen ab

### • Wettbewerber

- Oracle (Express Edition für Studenten), Microsoft SQLServer, IBM DB2 Universal DB (5 Produktlinien u. a. Enterprise Server Edition, hybride Version mit DB-Verwaltung von nativen XML-Dokumenten)
- Sybase Adaptive Server, CA Associates (OpenIngres ++?)
- SAG (Adabas D++), Informix Dynamic Server, . . .