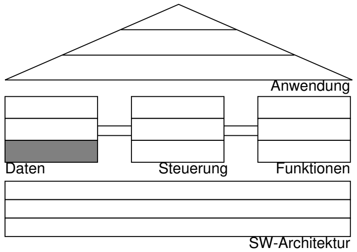
5. Die Standardsprache SQL

- **GBIS-Rahmen: Einordnung**



- **Grundlagen**
  - Funktions- und Einsatzbereiche
  - Befehlsübersicht und SQL-Grammatik
- **Mengenorientierte Anfragen (Retrieval)**
  - Anfragetypen
  - Aggregatfunktionen
  - Erklärungsmodell für die Anfrageauswertung
  - Vergleichsprädikate
- **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

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**[1]
  - 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: SQL2 mit drei Stufen (1992), SQL3 (SQL:1999) und SQL4 (2003?)**

1. http://www.cse.iitb.ernet.in:8000/proxy/db/~dbms/Data/Papers-Other/SQL1999/ (SQL-Standard Dokumente + einige Artikel)
http://www.wiscorp.com/sql99.html (Artikel + Präsentationen zu SQL:1999)

Möglichkeiten der Anfrage in SQL

- Zielgruppe: zunächst Nicht-Programmierer
- **Auswahlvermögen äquivalent dem Relationenkalkül und der Relationenalgebra**
- Vermeidung von mathematischen Konzepten wie Quantoren

➥ trotzdem: relational vollständig

SQL: strukturierte Sprache, die auf englischen Schlüsselwörtern basiert

Grundbaustein

SELECT	PNR
FROM	PERS
WHERE	ANR = 'K55'

Abbildung

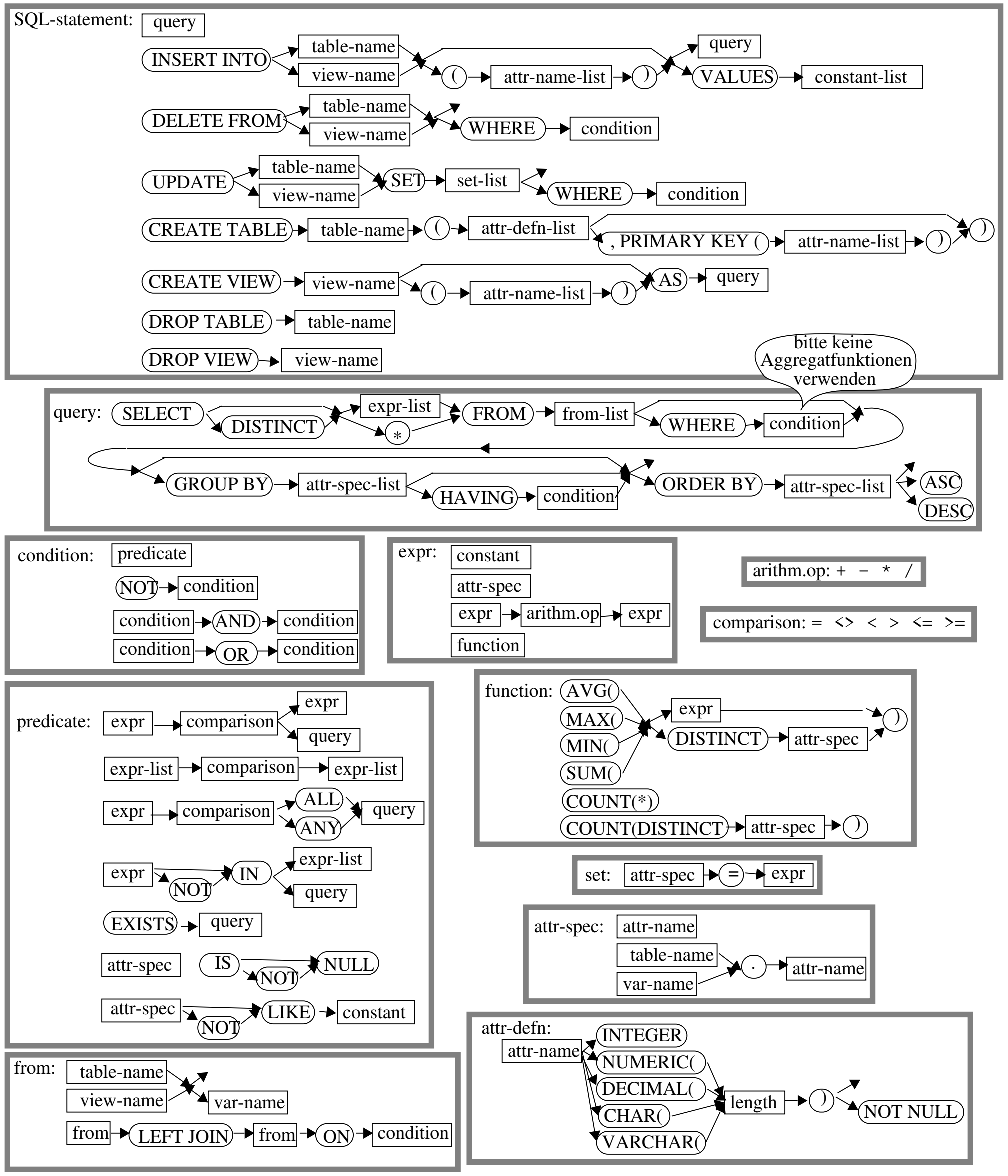
Ein bekanntes Attribut oder eine Menge von Attributen wird mit Hilfe einer Relation in ein gewünschtes Attribut oder einer Menge von Attributen abgebildet.

Allgemeines Format

<Spezifikation der Operation>
<Liste der referenzierten Tabellen>
[*WHERE Boolescher Prädikatsausdruck*]

SQL2-Grammatik

- **Semantik durch „allgemeine Regeln" in natürlicher Sprache**
- **SQL - Syntax** (Auszug, Table=Relation, Column=Attribut, Listenelemente durch Komma getrennt)



Anfragemöglichkeiten in SQL

```
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

DICHTER (DI)

AUTOR	G-ORT	G-JAHR

DRAMA (DR)

TITEL	AUTOR	KRITIKER	U-ORT	U-JAHR

SCHAUSPIELER (SP)

PNR	W-ORT	NAME

ROLLE (RO)

FIGUR	TITEL	R-TYP

DARSTELLER (DA)

PNR	FIGUR	A-JAHR	A-ORT	THEATER

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 IN (a_1, a_j, a_k) explizite Mengendefinition

A_i IN (SELECT . . .) implizite Mengendefinition

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

- 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

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?

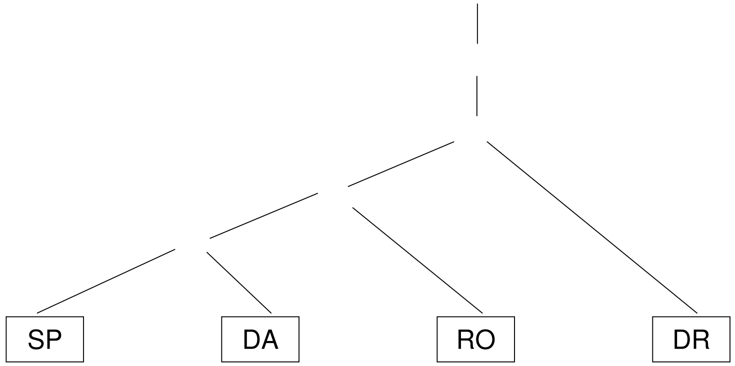
Q6: Finde die Schauspieler (NAME, W-ORT), die bei in Weimar uraufgeführten Dramen an ihrem Wohnort als 'Held' mitgespielt haben.

```
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
```

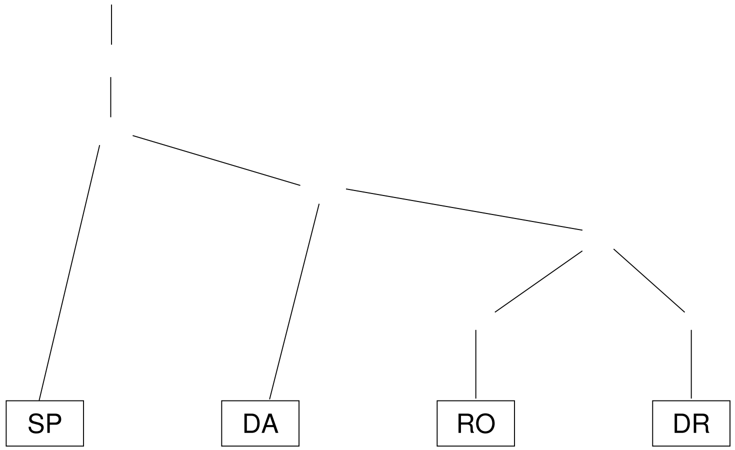
- Wie sieht das Auswertungsmodell (Erklärungsmodell) bei symmetrischer Notation aus?

Ausführung von SQL-Anweisungen

- **Abstraktes Erklärungsmodell für Q6**



- **Verbesserter 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)*

Benutzung von 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

Aggregat-Funktionen

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

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

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

Aggregat-Funktionen (2)

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 ?

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).

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.

Auswahl von Gruppen

```
HAVING cond-exp
```

Q13: Liste die Abteilungen zwischen K50 und K60 auf, bei denen das Durchschnittsalter ihrer Angestellten kleiner als 30 ist.

➥ 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
	406	Abel	50 K	829
	123	Maier	60 K	829
	829	Müller	55 K	574
	574	May	50 K	999

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

AUSGABE	X.NAME	X.GEHALT	Y.NAME

Hierarchische Beziehung auf einer Relation (2)

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

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

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

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	999

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

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

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

Erklärungsmodell von SQL-Anfragen - 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

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 Θ

```
 comparison-cond
::=   row-constructor Θ row-constructor

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

  - BETWEEN-Prädikate:

```
row-constr    [NOT]   BETWEEN   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)

- **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)**

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“

- **Klassen der Ähnlichkeitssuche**

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

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

3. **Semantische Ähnlichkeit** (Synonyme, Oberbegriffe, ...)

LIKE-Prädikate

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.

  - **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	
T	F
F	T
?	?

AND	T	F	?
T	T	F	?
F	F	F	F
?	?	F	?

OR	T	F	?
T	T	T	T
F	T	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

GEH > 70K AND PROV > 50K

GEH > 70K OR PROV > 50K

- 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
FROM PERS

- **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-Prädikat

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)

Semantik

x Θ ANY (SELECT y FROM T WHERE p) ⇔
EXISTS (SELECT * FROM T WHERE (p) AND x Θ T.y)

x Θ ALL (SELECT y FROM T WHERE p) ⇔
NOT EXISTS (SELECT * FROM T WHERE (p) AND NOT (x Θ T.y))

Quantifizierte Prädikate (2)

Q17: Finde die Manager, die mehr verdienen als <u>alle</u> ihre direkten Untergebenen

Q18: Finde die Namen der Schauspieler, die mindestens einmal gespielt haben (... nie gespielt haben)

Q19: Finde die Namen aller Schauspieler, die <u>alle</u> 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 (<u>snr</u>, name, ...); wettert (<u>datum, snr</u>, mintemp, ...)

In wettert stehen die täglich gemessenen Minimaltemperaturen der verschiedenen Meßstationen.[1]

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));
```

1. 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 1.did
                        from T4 1
                        where 1.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)

- 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

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

Löschen von Tupeln durch Suchklauseln

```
searched-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

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

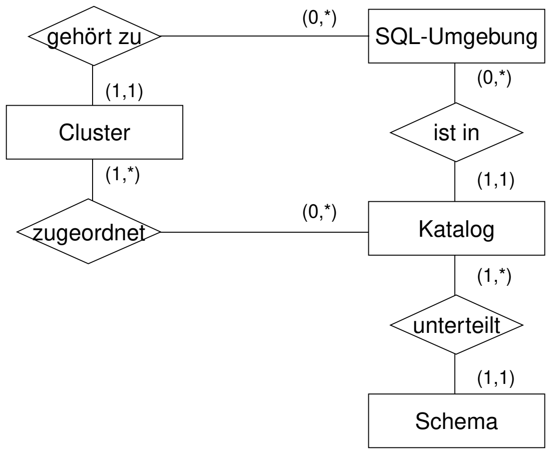
M5: Gib den Schauspielern, die am Pfalztheater spielen, eine Gehalts-erhöhung von 5% (Annahme: GEHALT in Schauspieler)

```
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:**
  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**

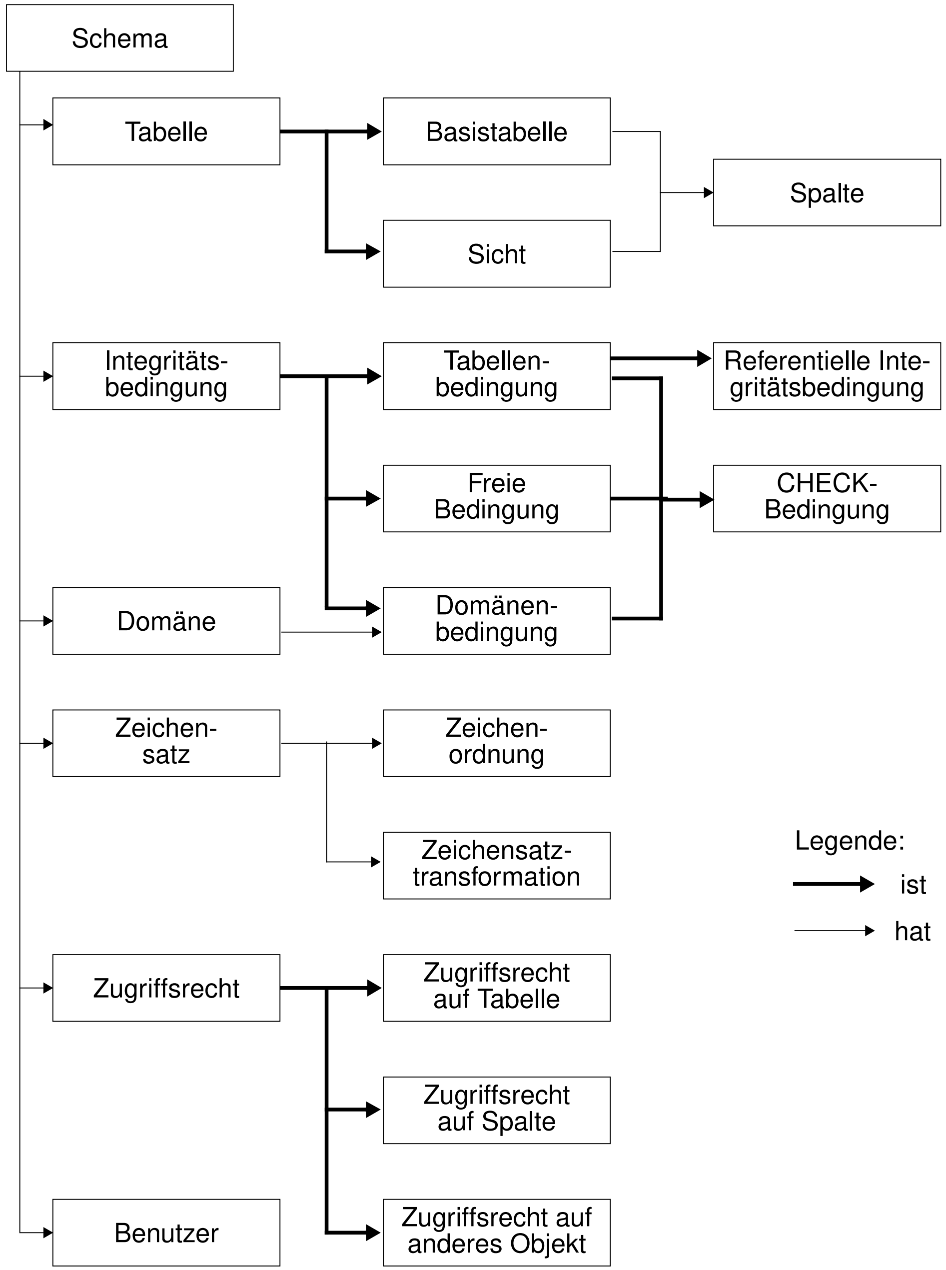


➥ Kataloge bestehen aus SQL-Schemata und können innerhalb einer SQL-Umgebung optional auf ein oder mehrere Cluster[1] verteilt werden

- **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

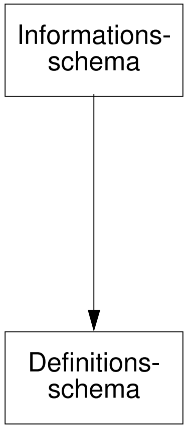
1. Sinn dieser Clusterbildung ist die Zuordnung von genau einem Cluster zu jeder SQL-Sitzung und dadurch wiederum die Zuordnung einer Menge von Daten bzw. Katalogen zu dieser Sitzung

Elemente des SQL-Schemas



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 nach einheitlichen und verbindlichen Richtlinien ist genauso wichtig
- **Zweischichtiges Definitionsmodell für die Beschreibung der Daten**



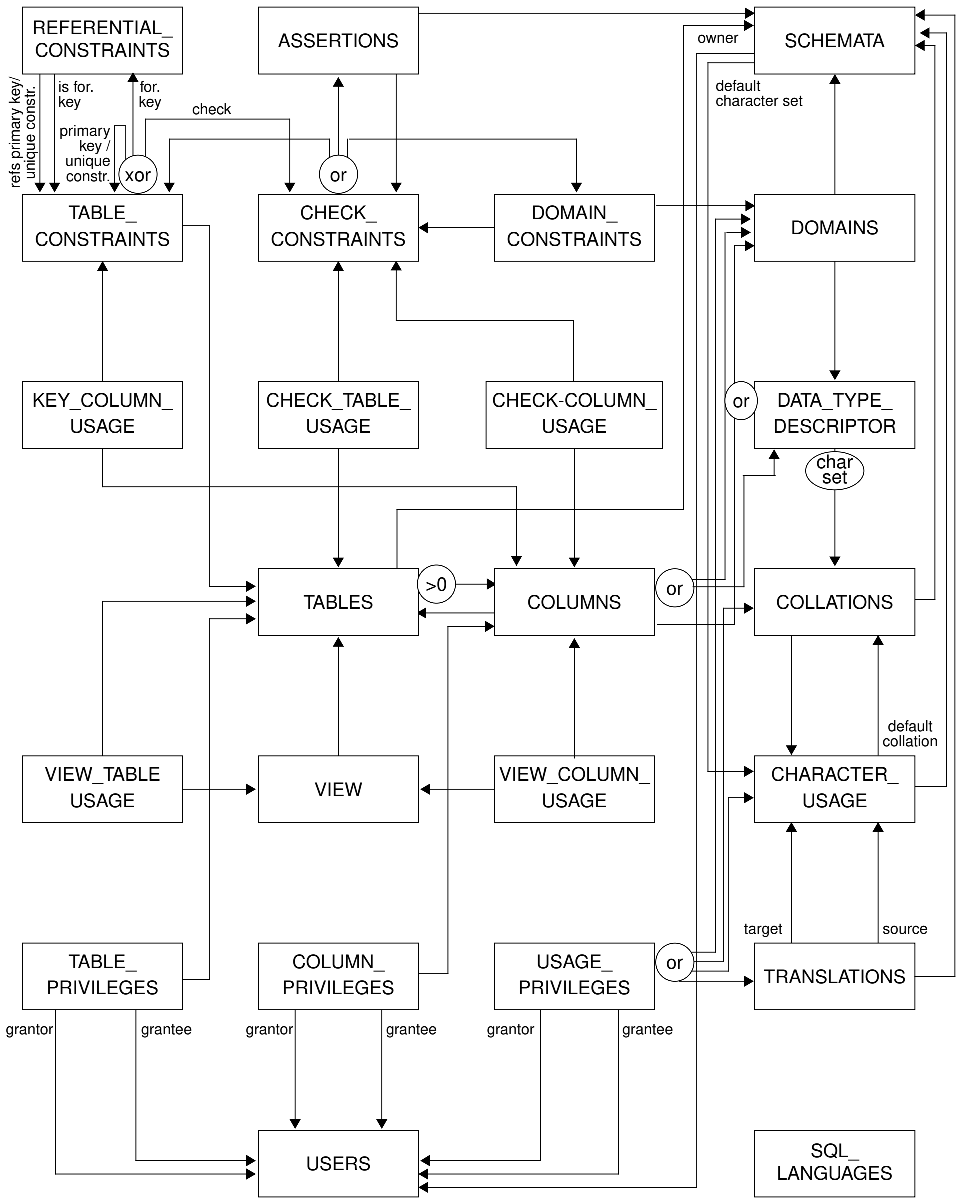
- bietet einheitliche Sichten in normkonformen Implementierungen
- ist für den Benutzer zugänglich und somit die definierte Schnittstelle zum Katalog

- beschreibt hypothetische Katalogstrukturen
- erlaubt „Altsysteme“ mit abweichenden Implementierungen normkonform zu werden

- **Komplexe Strukturen**[1]
  - DEFINITION_SCHEMA umfaßt 24 Basistabellen und 3 Zusicherungen
  - In den Tabellendefinitionen werden ausschließlich 3 Domänen verwendet: SQL_IDENTIFIER, CHARACTER_DATA und CARDINAL_NUMBER

1. 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 Namens-angabe 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 benamte 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 Typfestlegung
  - 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 **überprüf**t werden?

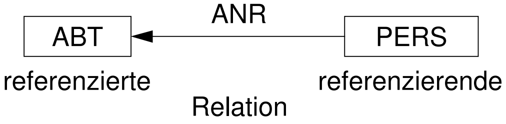
Abbildung von Beziehungen

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

- **Umsetzung ins Relationenmodell**

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

- **Referenzgraph**



- **Zusätzliche Regeln:**

  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

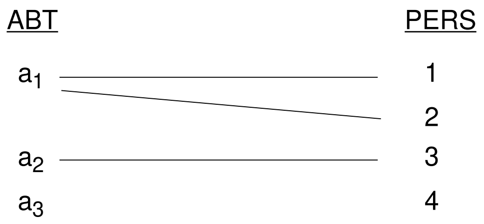
- **Bemerkung:**

  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 der der Fall [1,n] abgedeckt werden kann.
  - Bei der Erstellung müssen solche Beziehungen verzögert überprüft werden.

Abbildung von Beziehungen (2)

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



- **Abbildungsversuch (FS auf welche Seite?)**

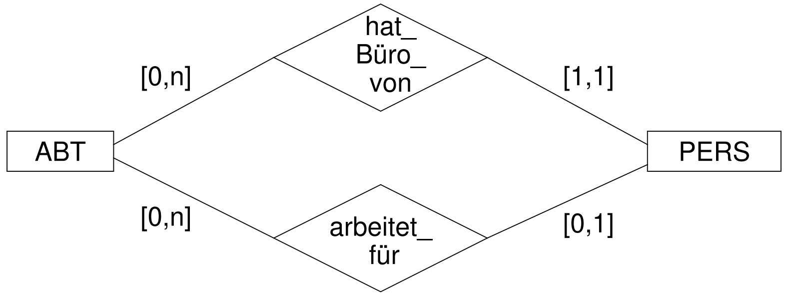
ABT (ABTNR, PNR, ...)   PERS (PNR, ...)

- **Abbildung im Relationenmodell**

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

Abbildung von Beziehungen (3)

- **ER-Diagramm**

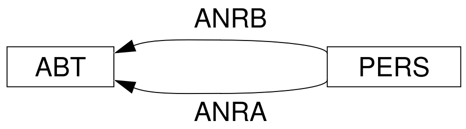


- **Umsetzung ins Relationenmodell**

```
ABT (ABTNR ...,                 PERS (PNR ...,
     ...                              ANRA ...,
     PRIMARY KEY (ABTNR))             ANRB... NOT NULL,


                                      PRIMARY KEY (PNR),
                                      FOREIGN KEY (ANRA) REFERENCES ABT,
                                      FOREIGN KEY (ANRB) REFERENCES ABT)
```

- **Referenzgraph**

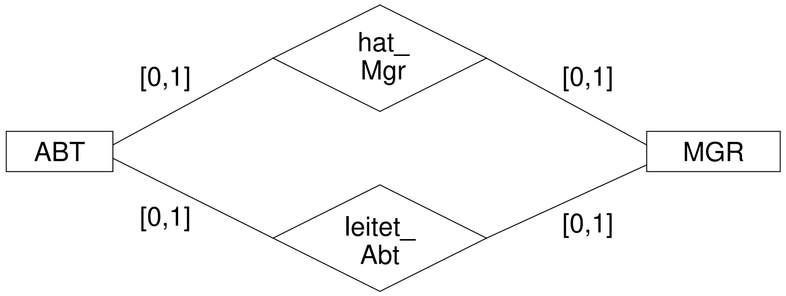


- **Bemerkung:**
  - Für jede FS-Beziehung benötigt man ein separates FS-Attribut.
  - Mehrere FS-Attribute können auf dasselbe PS/SK-Attribut verweisen.

Abbildung von Beziehungen (4)

- **Ziel: Darstellung einer symmetrischen (1:1)-Beziehung**

- **Erster Versuch: ER-Diagramm**



- **Umsetzung ins Relationenmodell**

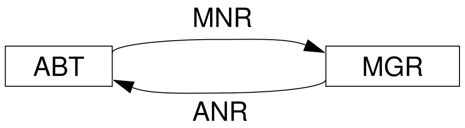
```
ABT (ANR ...,
     MNR ... UNIQUE,
     ...
     PRIMARY KEY (ANR),
     FOREIGN KEY (MNR)
         REFERENCES MGR)
```

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

→ Es sind alternative Lösungen möglich

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

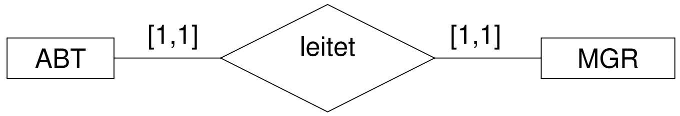
- **Referenzgraph**



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

Abbildung von Beziehungen (5)

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



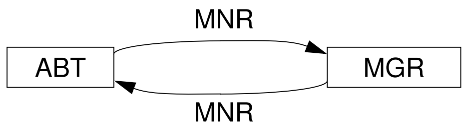
- **Umsetzung ins Relationenmodell**

```
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))
```

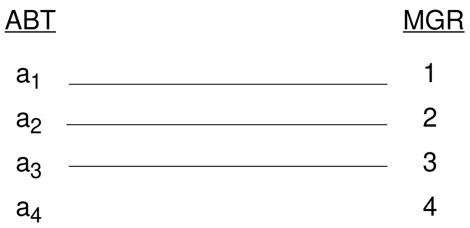
→ Es sind alternative Lösungen möglich

- 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

Abbildung von Beziehungen (6)

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



- **Versuch**

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

- **Abbildung im Relationenmodell**

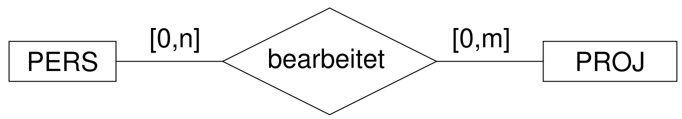
ABT (ABTNR, MNR, ...) PERS (MNR, ...)

- **Abbildung im Relationenmodell (Variation über Schlüsselkandidaten)**

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

Abbildung von Beziehungen (7)

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



- **Umsetzung ins Relationenmodell**

```
PERS (PNR ...,                  PROJ (JNR ...,
     ...                             ...
     PRIMARY KEY (PNR))              PRIMARY KEY (JNR))

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

→ Diese Standardlösung erzwingt eine „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**

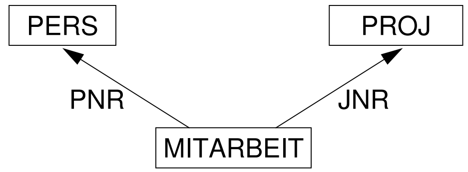
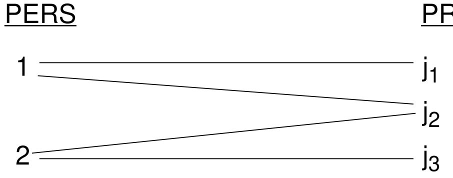


Abbildung von Beziehungen (8)

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



- **Abbildung im Relationenmodell**

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

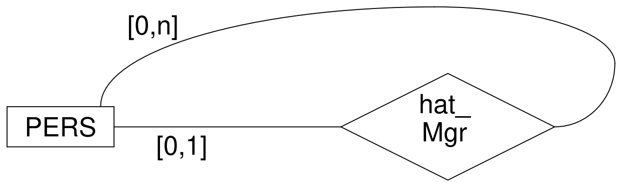
  MITARBEIT (PNR, JNR, ...)

- **Direkte (n:m)-Abbildung?**

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

Abbildung von Beziehungen (9)

- **ER-Diagramm: (1:n)-Beziehung als Selbstreferenz**



- **Umsetzung ins Relationenmodell**

```
PERS (PNR ...,
      MNR ...,
      ...
      PRIMARY KEY (PNR),
      FOREIGN KEY (MNR) REFERENCES PERS (PNR))
```

→ Mit Hilfe dieser Lösung kann die Personal-Hierarchie eines Unternehmens dargestellt werden. Die referentielle Beziehung stellt hier eine partielle Funktion dar, da die „obersten“ Manager einer Hierarchie keinen Manager haben

→ MNR ... NOT NULL läßt sich nur realisieren, wenn die „obersten“ Manager als ihre eigenen Manager interpretiert werden. Dadurch treten jedoch Referenzzyklen auf, was die Frageauswertung und die Konsistenzprüfung erschwert

- Welche Beziehungsstruktur erzeugt MNR ... UNIQUE NOT NULL?
- **Referenzgraph**

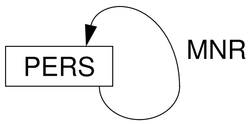
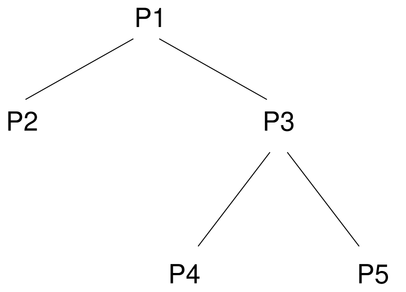


Abbildung von Beziehungen (10)

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

PERS



- **Mögliche Abbildung (Redundanz!)**

PERS‘ (<u>PNR</u>, ...) HAT_MGR (<u>PNR</u>, MNR...)

- **Abbildung im Relationenmodell**

PERS (<u>PNR</u>, ..., MNR)

Abbildung von Beziehungen (11)

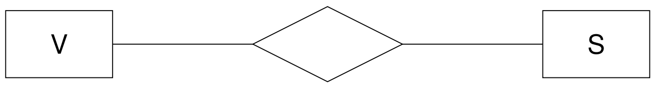
- **Zusammenfassung:** Relationenmodell hat **wertbasierte** Beziehungen
  - Fremdschlüssel (FS) und zugehöriger Primärschlüssel/Schlüsselkandidat (PS/SK) repräsentieren eine Beziehung (gleiche Wertebereiche!)
  - Alle Beziehungen (FS<-->PS/SK) sind binär und symmetrisch
  - Auflösung einer Beziehung geschieht durch Suche
  - Es sind i. allg. k (1:n)-Beziehungen zwischen zwei Relationen möglich

➥ Objektorientierte Datenmodelle haben **referenzbasierte** Beziehungen!

- **Spezifikationsmöglichkeiten in SQL**

PS	PRIMARY KEY (implizit: UNIQUE NOT NULL)
SK	UNIQUE [NOT NULL]
FS	[UNIQUE] [NOT NULL]

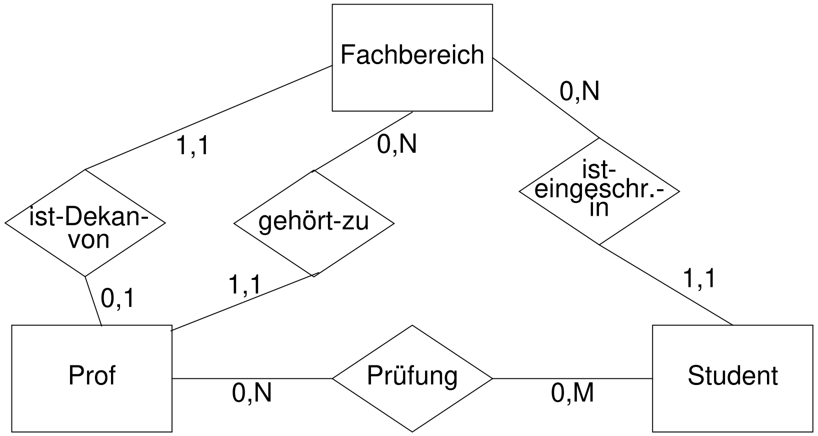
- **Fremdschlüsseldeklaration (in S)**



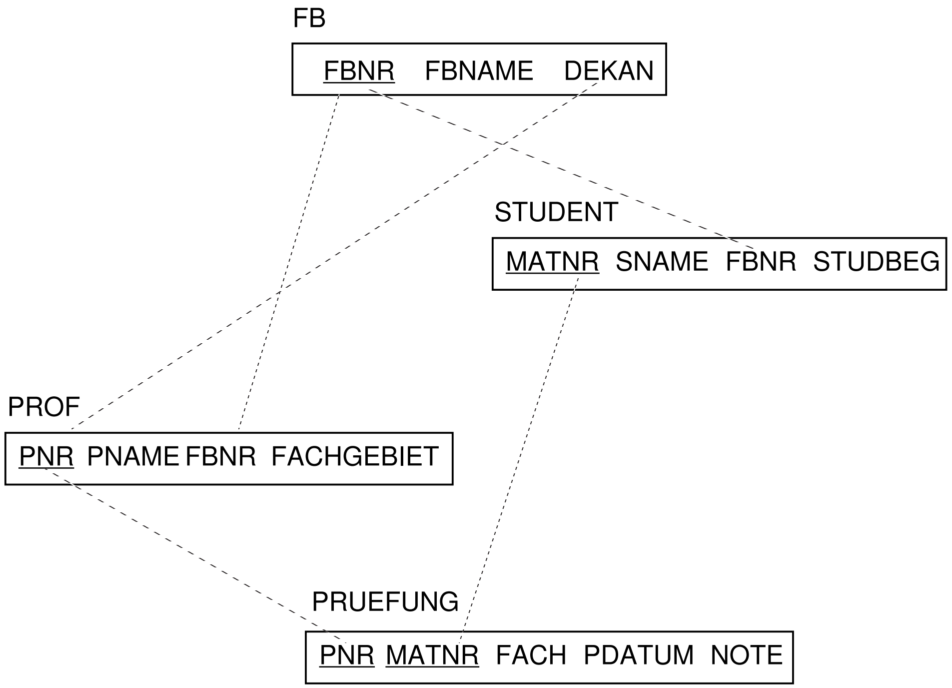
0,n	0,1
0,n	1,1
0,1	0,1
0,1	1,1

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   FACHBEREICHSNAME     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        FACHBEREICHSNAME     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

FB	FBNR	FBNAME	DEKAN
	FB 9	WIRTSCHAFTSWISS	4711
	FB 5	INFORMATIK	2223

PROF	PNR	PNAME	FBNR	FACHGEBIET
	1234	HÄRDER	FB 5	DATENBANKSYSTEME
	5678	WEDEKIND	FB 9	INFORMATIONSSYSTEME
	4711	MÜLLER	FB 9	OPERATIONS RESEARCH
	6780	NEHMER	FB 5	BETRIEBSSYSTEME
	2223	RICHTER	FB 5	EXPERTENSYSTEME

STUDENT	MATNR	SNAME	FBNR	STUDBEG
	123 766	COY	FB 9	1.10.95
	225 332	MÜLLER	FB 5	15. 4.87
	654 711	ABEL	FB 5	15.10.94
	226 302	SCHULZE	FB 9	1.10.95
	196 481	MAIER	FB 5	23.10.95
	130 680	SCHMID	FB 9	1. 4.97

PRÜFUNG	PNR	MATNR	FACH	PDATUM	NOTE
	5678	123 766	BWL	22.10.97	4
	4711	123 766	OR	16. 1.98	3
	1234	654 711	DV	17. 4.97	2
	1234	123 766	DV	17. 4.97	4
	6780	654 711	SP	19. 9.97	2
	1234	196 481	DV	15.10.97	1
	6780	196 481	BS	23.12.97	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**
   d) Löschen eines Vater-Tupels
   e) Ändern des PS/SK in einem Vater-Tupel
   f) 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?**

   → 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 der referentiellen Integrität

- **SQL2-Standard führt „referential actions“ ein**

- **Genauere Spezifikation der referentiellen Aktionen** für jeden Fremdschlüssel (FS)

  1. Sind „Nullen“ verboten ?

     NOT NULL

  2. Löschregel für Zielrelation (referenzierte Relation)

     ON DELETE
     {CASCADE | RESTRICT[1] | SET NULL | SET DEFAULT | NO ACTION}

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

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

- 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 benutzer-definierten 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

---

1. Die Option RESTRICT wird hier explizit aufgeführt; sie entspricht dem Fall, daß die gesamte Klausel weggelassen 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**

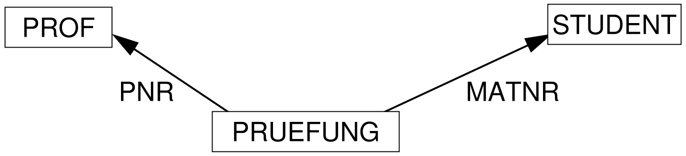
FB	FBNR	FBNAME
	FB9	WIRTSCHAFTSWISS
	FB5	INFORMATIK

STUDENT	MATNR	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	PNR	PNAME
	1234	HÄRDER
	4711	MÜLLER

STUDENT	MATNR	SNAME
	123 766	COY
	654 711	ABEL

PRÜFUNG	PNR	MATNR	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

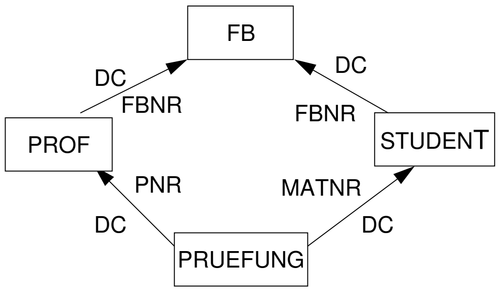
Prof \ Student	DC	DR	UC	UR
DC				
DR				
UC				
UR				

➥ Unabhängige referentielle Beziehungen können unabhängig definiert und gewartet werden

Auswirkungen referentieller Aktionen (3)

- Was heißt Unabhängigkeit der referentiellen Beziehungen?

3. Vollständiges Beispiel



- **Lösche FB (FBNR=FB9)**

<u>erst links</u>	<u>erst rechts</u>
- Löschen in FB	- Löschen in FB
- Löschen in PROF	- Löschen in STUDENT
- Löschen in PRUEFUNG	- Löschen in PRUEFUNG
- Löschen in STUDENT	- Löschen in PROF
- Löschen in PRUEFUNG	- Löschen in PRUEFUNG

→ Ergebnis der Operation unabhängig von der Reihenfolge der referentiellen Aktionen

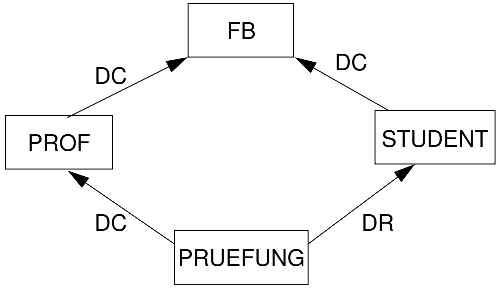
→ Eindeutigkeit des erreichten DB-Zustandes

- 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)

- **Modifikation des Schemas**



- **Lösche FB (FBNR=FB9)**

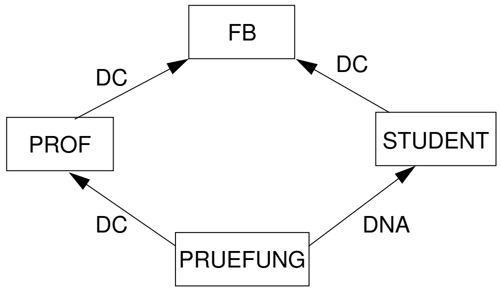
<u>erst links</u>	<u>erst rechts</u>
- Löschen in FB	- Löschen in FB
- Löschen in PROF	- Löschen in STUDENT
- Löschen in PRUEFUNG	- Zugriff auf PRUEFUNG
- Löschen in STUDENT	Wenn ein gerade gelöschter Student
- Zugriff auf PRUEFUNG	eine Prüfung abgelegt hatte
Wenn ein Student bei einem	→ Rücksetzen
FB-fremden Professor geprüft	sonst:
wurde	- Löschen in PROF
→ Rücksetzen	- Löschen in PRUEFUNG

→ Es können reihenfolgenabhängige Ergebnisse auftreten!

→ Die Reihenfolgenabhängigkeit ist hier wertabhängig

Auswirkungen referentieller Aktionen (5)

- **Modifikation des Schemas**



- **Lösche FB (FBNR=FB9)**

erst links	erst rechts
- Löschen FB	- Löschen FB
- Löschen PROF	- Löschen STUDENT
- Löschen PRUEFUNG	- Löschen PROF
- Löschen STUDENT	- Löschen PRUEFUNG
Test, ob es noch offene Referenzen in PRUEFUNG auf gelöschte Studenten gibt; wenn ja → Rücksetzen	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 (konfliktträchtige 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:

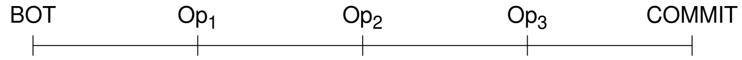
a) Sie werden verboten:
  - Statische Schemanalyse kann Sicherheit eines Schemas nicht feststellen

  Dabei sind ggf. pessimistische Annahmen zu treffen, je nachdem, ob bei der Analyse nur Relationen oder auch ihre Attribute (Attributkonflikte) betrachtet werden.

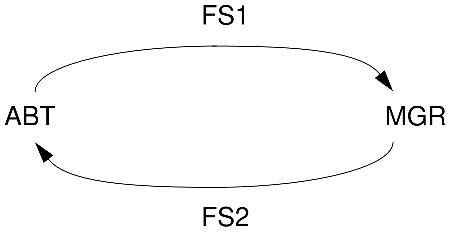
b) Sie werden erlaubt:
  - Die referentiellen Aktionen werden bei jeder Operation dynamisch überwacht
  - Falls ein Konflikt erkannt wird, wird die Operation zurückgesetzt

Durchführung der Änderungsoperationen

- **Prüfung der referentiellen Integrität (IMMEDIATE/DEFERRED)**

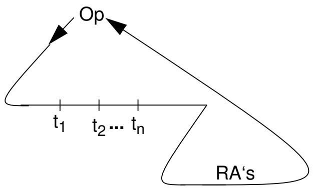


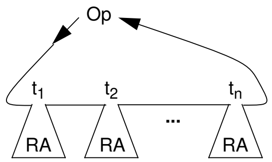
- **Bei zyklischen Referenzpfaden**



  - 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 *atomar*
  - mengenorientiertes oder tupelorientiertes Verarbeitungsmodell





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

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.