4. Die Standardsprache[1] SQL

- **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)**
  - Wertebereiche, Attribute, Basisrelationen
  - Abbildung von Beziehungen
- **Kopplung mit einer Wirtssprache**
  - Eigenschaften relationaler Anwendungs-programmierschnittstellen (API)
  - Cursor-Konzept

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

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[2]**
  - Entwicklung einer vereinheitlichten DB-Sprache für alle Aufgaben der DB-Verwaltung
  - Lehrexperimente mit Studenten mit und ohne Programmiererfahrung
  - 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 (SQL:2003)

2. 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)

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[3]

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*]

3. 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

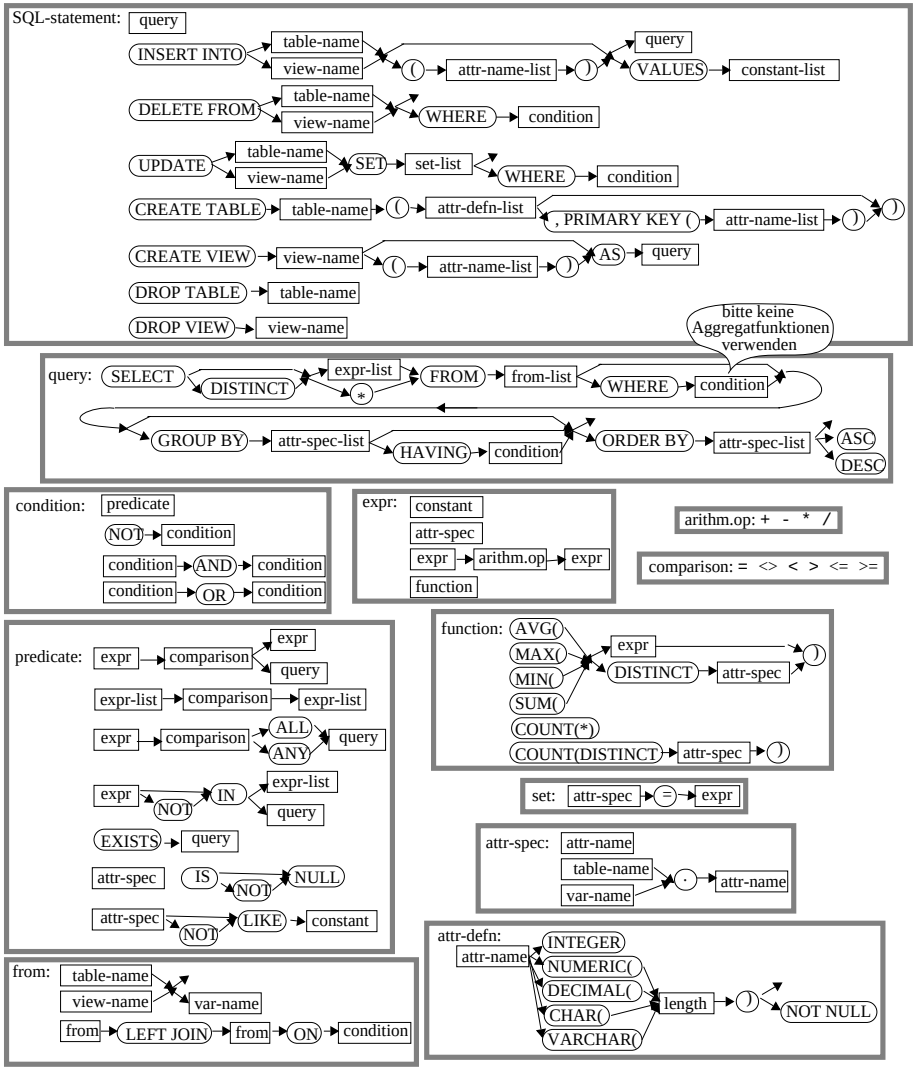
Erweiterungen zu einer vollständigen DB-Sprache

- **Möglichkeiten der Datenmanipulation**
  - Einfügen, Löschen und Ändern von individuellen Tupeln und von Mengen von Tupeln
  - Zuweisung von ganzen Relationen
- **Möglichkeiten der Datendefinition**
  - Definition von Wertebereichen, Attributen und Relationen
  - Definition von verschiedenen Sichten auf Relationen
- **Möglichkeiten der Datenkontrolle**
  - Spezifikation von Bedingungen zur Zugriffskontrolle
  - Spezifikation von Zusicherungen (assertions) zur semantischen Integritätskontrolle
- **Kopplung mit einer Wirtssprache**
  - deskriptive Auswahl von Mengen von Tupeln
  - sukzessive Bereitstellung einzelner Tupeln

	Retrieval	Manipulation	Daten-definition	Daten-kontrolle
Stand-Alone DB-Sprache				
Eingebettete DB-Sprache				

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	U-ORT	U-JAHR	AUTOR

SCHAUSPIELER (SP)

PNR	NAME	W-ORT

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 Relationen-grenzen hinweg möglich
  - einfache Formulierung des Verbundes

Symmetrische Notation (2)

Q5: Finde die Schauspieler, die an einem Ort wohnen, an dem sie gespielt haben.

```
SELECT  S.NAME, S.W-ORT
FROM    SCHAUSPIELER S, DARSTELLER D
WHERE   S.PNR = D.PNR
  AND   S.W-ORT = D.A-ORT
```

- Welche Rolle spielt die Bedingung S.PNR = D.PNR in der erhaltenen Lösung?

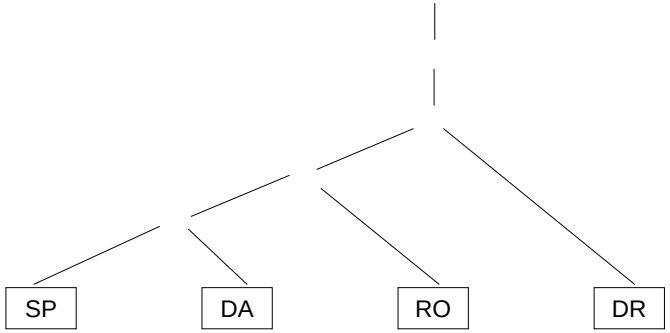
Q6:Finde die Schauspieler (NAME, W-ORT), die bei in Weimar urauf-geführten Dramen an ihrem Wohnort als 'Held' mitgespielt haben.

```
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
```

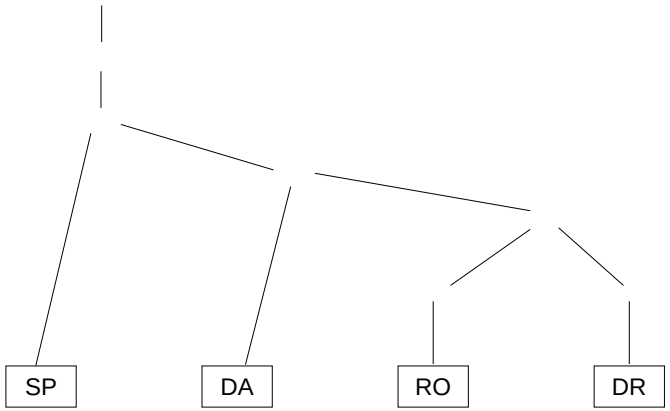
- Wie sieht das Auswertungsmodell (Erklärungsmodell) bei symmetrischer Notation aus?

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)

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)

- **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)?

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).

- 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.

➥ 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	-

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

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

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
                          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 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** (Ontologien, 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-Prädikate

row-constr Θ { ALL | ANY | SOME} (table-exp)

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

Θ **ANY /** Θ **SOME:**
analog, wenn der Θ-Vergleich für einen Ergebniswert „true“ ist

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

Q17: Finde die Manager, die mehr verdienen als alle ihre Angestellten

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


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))

Q18: Finde die Manager, die mehr verdienen als alle ihre Angestellten

```
SELECT M.PNR
FROM PERS M
WHERE NOT EXISTS
    (SELECT *
    FROM PERS P
    WHERE M.PNR = P.MNR
        AND M.GEHALT <= P.GEHALT)
```

Einsatz mengentheoretischer Operatoren

- **Vereinigung** (UNION), **Durchschnitts**- (INTERSECT) und **Differenzbildung** (EXCEPT) von Relationen bzw. Anfrageergebnissen

```
(table-exp) { UNION | INTERSECT | EXCEPT }
                    [ ALL ]
    [ CORRESPONDING  [BY (column-commalist ) ] ]
(table-exp)
```

- Vor Ausführung werden Duplikate aus den Operanden entfernt, außer wenn ALL spezifiziert ist

- Für die Operanden wird Vereinigungsverträglichkeit gefordert
  Abschwächung:
  - *CORRESPONDING BY (A1, A2, ...An)*
    Operation auf Attribute Ai beschränkt, die in beiden Relationen vorkommen müssen (➥ n-stelliges Ergebnis)
  - *CORRESPONDING*
    Operation ist auf gemeinsame Attribute beschränkt

Q19: Welche Schauspieler haben nie gespielt?

```
SELECT S.PNR
FROM  SCHAUSPIELER S

   EXCEPT

SELECT D.PNR
FROM DARSTELLER D
```


Join-Ausdrücke

Beispiel:
```
SELECT *
FROM   SCHAUSPIELER  S, DARSTELLER  D
WHERE  S.PNR =  D.PNR
```

gleichbedeutend mit
```
SELECT *
FROM   SCHAUSPIELER  NATURAL  JOIN  DARSTELLER
```

bzw. ... FROM SCHAUSPIELER **JOIN** DARSTELLER **USING** (PNR)

... FROM SCHAUSPIELER S **JOIN** DARSTELLER D **ON** S.PNR=D.PNR

- **Outer Join:** LEFT JOIN, RIGHT JOIN, FULL JOIN

- **Kartesisches Produkt: CROSS JOIN**

  A **CROSS JOIN** B ⇔ SELECT * FROM A, B

- **Outer Union:** UNION JOIN
  (Vereinigung von nur teilweise vereinigungsverträglichen Relationen)

  Beispiel:
  ```
  STUDENT  (MATNR, NAME, FB, IMMATDAT)
  ANGEST   (PNR, NAME, FB, WOCHENSTUNDEN)

  SELECT *
  FROM   STUDENT UNION JOIN ANGEST
  ```

Es gibt immer viele Möglichkeiten!

Q20: Finde die Messstation mit der niedrigsten gemessenen Temperatur

Gegeben: station (snr, name, ...); wettert (datum, snr, mintemp, ...)

In wettert stehen die täglich gemessenen Minimaltemperaturen der verschiedenen Messstationen.[4]

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));
```

4. 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'))
```

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.


```
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]

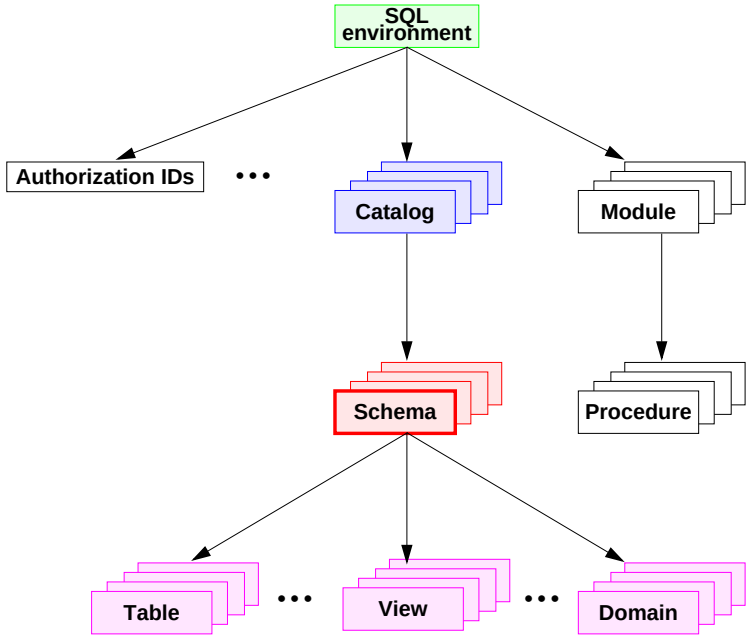
M5: Gib den Schauspielern, die am Pfalztheater spielen, eine Gehaltserhöhung von 5% (Annahme: GEHALT in Schauspieler)

```
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?**
- **SQL92-Architektur**



- **SQL-Umgebung (Environment) besteht aus**
  - Benutzern (authorization identifiers)
  - Katalogen
  - SQL-Module
- **Ein Katalog enthält**
  - für jede Datenbank ein DB-Schema
  - ein INFORMATION_SCHEMA (Metadaten über alle Schemata)

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 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 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 **überprüf**t werden?

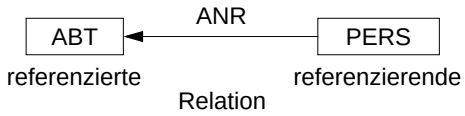
Abbildung von Beziehungen

- **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

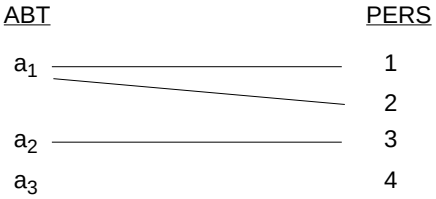
- **Bemerkung:**

  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 welcher 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?)**

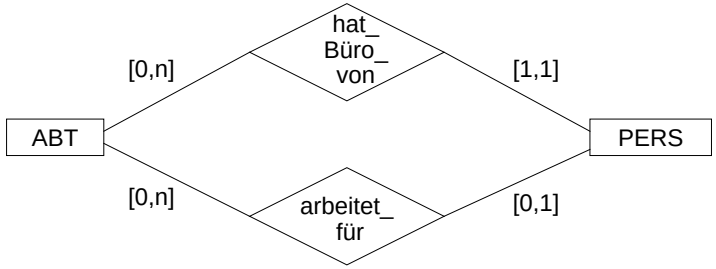
ABT (ABTNR, PNR, ...) PERS (PNR, ...)

- **Abbildung im Relationenmodell**

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

Abbildung von Beziehungen (3)

- **ER-Diagramm**

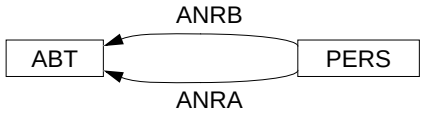


- **Umsetzung ins Relationenmodell**

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

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

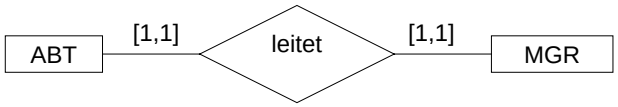
- **Referenzgraph**



- **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)

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

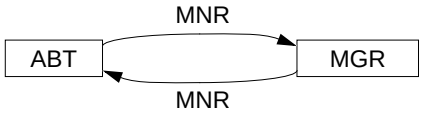


- **Umsetzung ins Relationenmodell**

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

➥ 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**

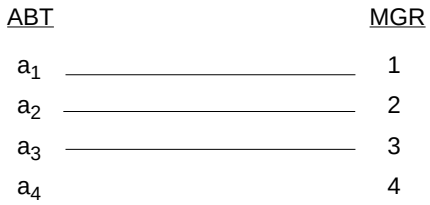
```
ABT (ANR ...,                          MGR (SVNR ...,
     MNR ... UNIQUE,                        MNR ... UNIQUE,
     ...                                    ...
     PRIMARY KEY (ANR),                     PRIMARY KEY (SVNR)
     FOREIGN KEY (MNR)                      FOREIGN KEY (MNR)
         REFERENCES MGR(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 (5)

- **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 (6)

- **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 „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**

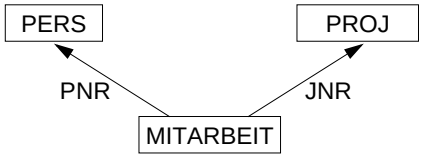
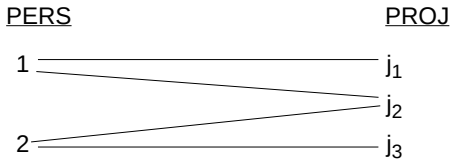


Abbildung von Beziehungen (7)

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



- **Abbildung im Relationenmodell**

PERS (<u>PNR</u>, ...) PROJ (<u>JNR</u>, ...)

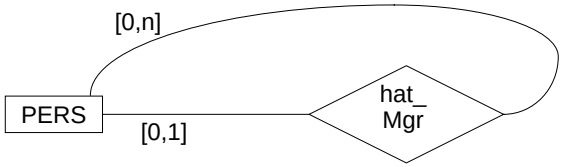
MITARBEIT (<u>PNR, JNR</u>, ...)

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

PERS (<u>PNR</u>, JNR, ...) PROJ (<u>JNR</u>, PNR, ...)

Abbildung von Beziehungen (8)

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



- **Umsetzung ins Relationenmodell**

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

➥ Lösung erlaubt Darstellung der Personal-Hierarchie eines Unternehmens. 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**

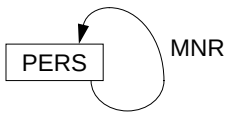
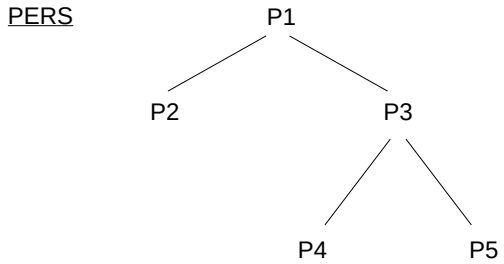


Abbildung von Beziehungen (9)

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



- **Mögliche Abbildung (Redundanz!)**

PERS‘ (PNR, ...) HAT_MGR (PNR, MNR...)

- **Abbildung im Relationenmodell**

PERS (PNR, ..., MNR)

Abbildung von Beziehungen – Zusammenfassung

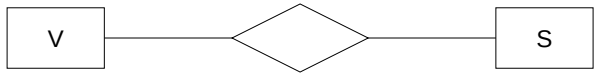
- Relationenmodell hat **wertbasierte** Beziehungen
  - Fremdschlüssel (FS) und zugehöriger Primärschlüssel/Schlüsselkandidat (PS/SK) repräsentieren eine Beziehung (gleiche Wertebereiche!)
  - Alle Beziehungen (FS<—>PS/SK) sind binär und symmetrisch
  - Auflösung einer Beziehung geschieht durch Suche
  - 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]

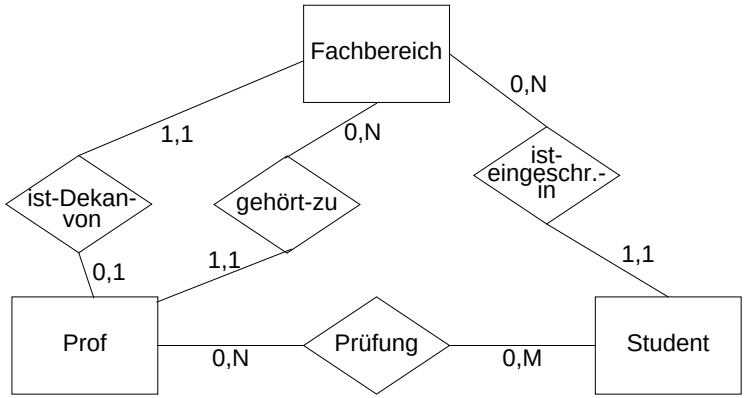
- **Fremdschlüsseldeklaration (in S)**



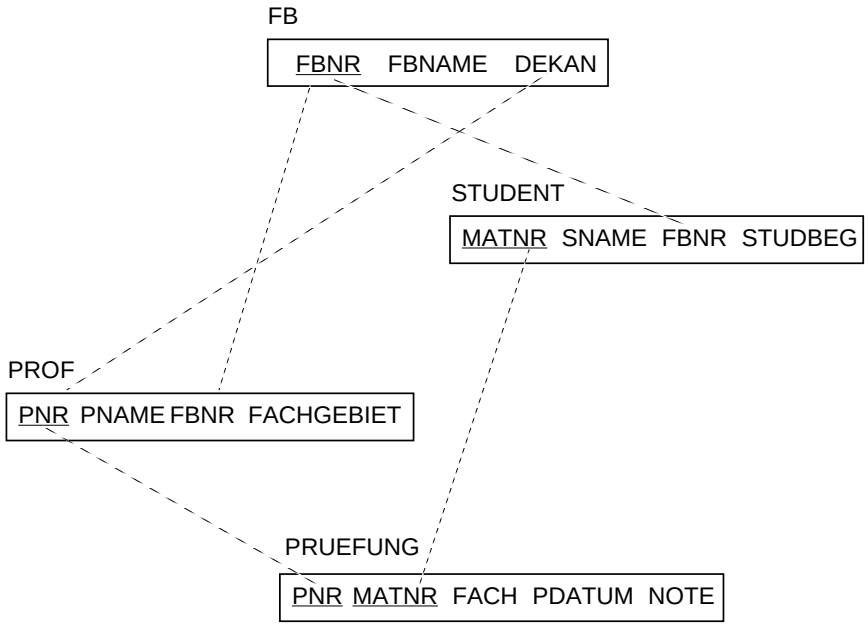
0,n	0,1
0,n	1,1
0,1	0,1
0,1	1,1

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

DB-Schema

FB

FBNR	FBNAME	DEKAN

PROF

PNR	PNAME	FBNR	FACHGEB

STUDENT

MATNR	SNAME	FBNR	STUDBEG

PRÜFUNG

PNR	MATNR	FACH	DATUM	NOTE

Ausprägungen

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

PROF	PNR	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	MATNR	SNAME	FBNR	STUDBEG
	123 766	COY	FB 9	1.10.00
	225 332	MÜLLER	FB 5	15.04.97
	654 711	ABEL	FB 5	15.10.99
	226 302	SCHULZE	FB 9	1.10.00
	196 481	MAIER	FB 5	23.10.00
	130 680	SCHMID	FB 9	1.04.02

PRÜFUNG	PNR	MATNR	FACH	PDATUM	NOTE
	5678	123 766	BWL	22.10.03	4
	4711	123 766	OR	16.01.02	3
	1234	654 711	DV	17.04.03	2
	1234	123 766	DV	17.04.03	4
	6780	654 711	SP	19.09.03	2
	1234	196 481	DV	15.10.03	1
	6780	196 481	BS	23.10.03	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
{CASCADE | RESTRICT | SET NULL | SET DEFAULT |
NO ACTION[5]}

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

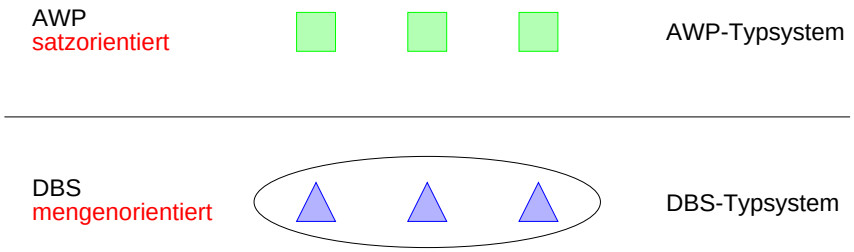
ON UPDATE
{CASCADE | RESTRICT | SET NULL | SET DEFAULT |
NO ACTION}

- 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

5. Die Option NO ACTION wird hier explizit aufgeführt; sie entspricht dem Fall, daß die gesamte Klausel weggelassen wird.

Kopplung mit einer Wirtssprache

- **Relationale AP-Schnittstellen (API)**
  - bieten deskriptive DB-Operationen (mengenorientierter Zugriff)
  - und Mehrsprachenfähigkeit
  - erfordern jedoch Maßnahmen zur Überwindung der sog. Fehlanpassung (impedance mismatch)



- **Kernprobleme der API bei konventionellen Programmiersprachen**
  - Konversion und Übergabe von Werten
  - Übergabe aktueller Werte von Wirtssprachenvariablen (Parametrisierung von DB-Operationen)
  - DB-Operationen sind i. Allg. mengenorientiert:
    Wie und in welcher Reihenfolge werden Tupel/Sätze dem AP zur Verfügung gestellt?

    ➥ Cursor-Konzept

Cursor-Konzept

- **Cursor-Konzept zur satzweisen Verarbeitung von Ergebnismengen**
  - Trennung von Qualifikation und Bereitstellung/Verarbeitung von Zeilen
  - Cursor ist ein Iterator, der einer Anfrage zugeordnet wird und mit dessen Hilfe die Zeilen der Ergebnismenge einzeln (one tuple at a time) im Programm bereitgestellt werden
  - Wie viele Cursor können im AWP sein?

- **Cursor-Deklaration**

```
DECLARE cursor CURSOR FOR table-exp
    [ORDER BY order-item-commalist]
```

```
DECLARE C1 CURSOR FOR
  SELECT Name, Gehalt, Anr FROM Pers WHERE Anr = 'K55'
  ORDER BY Name;
```

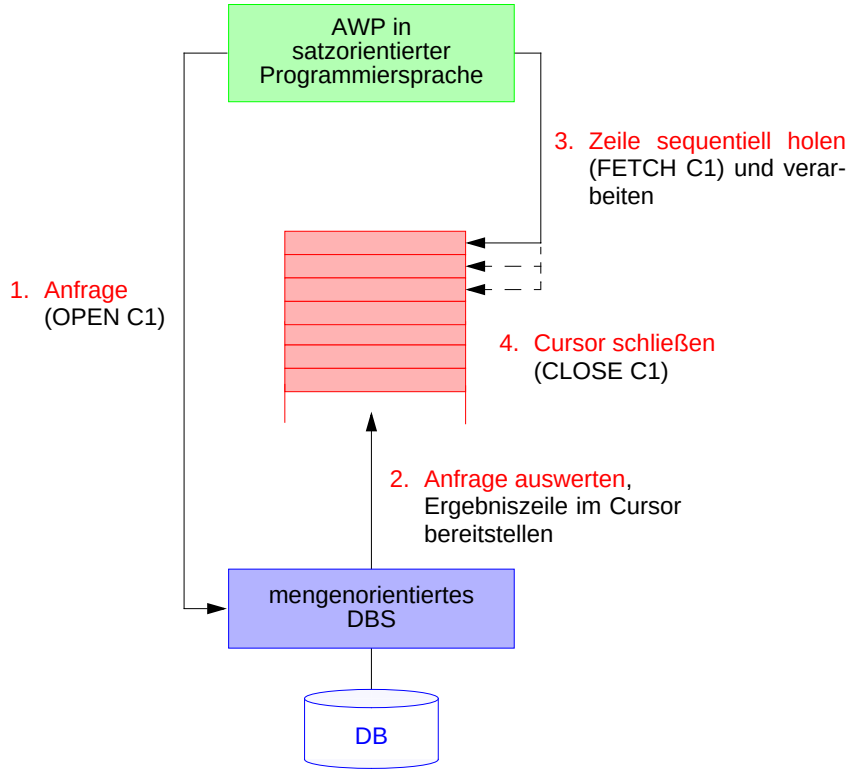
- **Operationen auf einen Cursor C1**

```
OPEN   C1
FETCH C1  INTO  Var1, Var2, . . ., Varn
CLOSE C1
```

- **Reihenfolge der Ergebniszeilen**
  - systembestimmt
  - benutzerspezifiziert (ORDER BY)

Cursor-Konzept (2)

- **Veranschaulichung der Cursor-Schnittstelle**



- **Wann wird die Ergebnismenge angelegt?**
  - **lazy:** schritthaltende Auswertung durch das DBS? Verzicht auf eine explizite Zwischenspeicherung ist nur bei einfachen Anfragen möglich
  - **eager:** Kopie bei OPEN? Ist meist erforderlich (ORDER BY, Join, Aggregat-Funktionen, ...)

Cursor-Konzept (3)

- **Beispielprogramm in C (vereinfacht)**

```
exec sql begin declare section;
char X[50], Y[3];
exec sql end declare section;

exec sql declare C1 cursor for
   select Name from Pers where Anr = :Y;


printf("Bitte Anr eingeben: \n");
scanf("%d", Y);
exec sql open C1;
while (sqlcode == OK)
{
   exec sql fetch C1 into :X;
   printf("Angestellter %d\n", X);
}
exec sql close C1;
```

- **Anbindung einer SQL-Anweisung** an die Wirtssprachen-Umgebung
  - eingebettete SQL-Anweisungen werden durch **exec sql** eingeleitet und durch spezielles Symbol (hier ";") beendet, um dem Compiler eine Unterscheidung von anderen Anweisungen zu ermöglichen
  - Verwendung von AP-Variablen in SQL-Anweisungen verlangt Deklaration innerhalb eines **declare section**-Blocks sowie Angabe des Präfix ":" innerhalb von SQL-Anweisungen
  - OPEN C1 bindet die Werte der Eingabevariablen des Cursors
  - Systemvariable SQLCODE zur Übergabe von Fehlermeldungen
  - Übergabe der Werte eines Tupels mit Hilfe der INTO-Klausel
    - INTO target-commalist (Variablenliste des Wirtsprogramms)
    - Anpassung der Datentypen (Konversion)

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**".
  - SQL2/3 bietet reichhaltige Optionen für referentielle Aktionen
- **Cursor-Konzept zur satzweisen Verarbeitung von Datenmengen**
  - Anpassung von mengenorientierter Bereitstellung und satzweiser Verarbeitung von DBS-Ergebnissen
  - Operationen: DECLARE CURSOR, OPEN, FETCH, CLOSE