

4. SQL-Einführung

Grundlagen: Anfragekonzept, Einsatzbereiche, Befehlsübersicht

mengenorientierte Anfragen deskriptiver Art (SELECT)

- einfache Selektions-, Projektions- und Join-Anfragen
- geschachtelte Anfragen, Aggregatfunktionen, Gruppenanfragen
- Suchbedingungen: Vergleichs-, LIKE-, BETWEEN-, IN-Prädikate, Nullwertbehandlung, quantifizierte Prädikate (ALL/ANY, EXISTS)
- mengentheoretische Operationen: UNION, INTERSECT, EXCEPT
- Join-Ausdrücke
- Verallgemeinerte Verwendung von Sub-Queries
- Vergleich mit der Relationenalgebra

Kap. 6: Datendefinition und -manipulation in SQL

- Datendefinition (DDL): Datentypen, Domains
- Erzeugen/Ändern/Löschen von Tabellen, Sichtkonzept
- Änderungsoperationen (INSERT, DELETE, UPDATE)
- Integritätsbedingungen

Kap. 7: Kopplung mit einer Wirtssprache

Entwicklung von SQL

seit 1975 viele Entwürfe für relationale Anfragesprachen

- SEQUEL: Structured English Query Language (System R)
- QUEL (Ingres), . . .

Sprachentwicklung von SQL

- Entwicklung einer vereinheitlichten DB-Sprache für alle Aufgaben der DB-Verwaltung
- leichter Zugang durch verschiedene "Sprachebenen"
- einfache Anfragemöglichkeiten für den gelegentlichen Benutzer, mächtige Sprachkonstrukte für den besser ausgebildeten Benutzer, spezielle Sprachkonstrukte für den DBA

Standardisierung von SQL durch ANSI und ISO

- erster ISO-Standard 1987
- verschiedene Addenda (1989)
- 1992: Verabschiedung von „SQL2“ bzw. SQL-92 (Entry, Intermediate, Full Level)
- 1999/2003: SQL:1999 („SQL3“) und SQL:2003 („SQL4“) mit objektorientierten Erweiterungen etc. (-> objekt-relationale DBS)

SQL "intergalactic data speak"

Möglichkeiten der Anfrage in SQL

SQL: strukturierte Sprache, die auf englischen Schlüsselwörtern basiert

- Zielgruppe umfaßt auch Nicht-Programmierer
- Auswahlvermögen äquivalent der Relationenalgebra (relational vollständig)

Grundbaustein

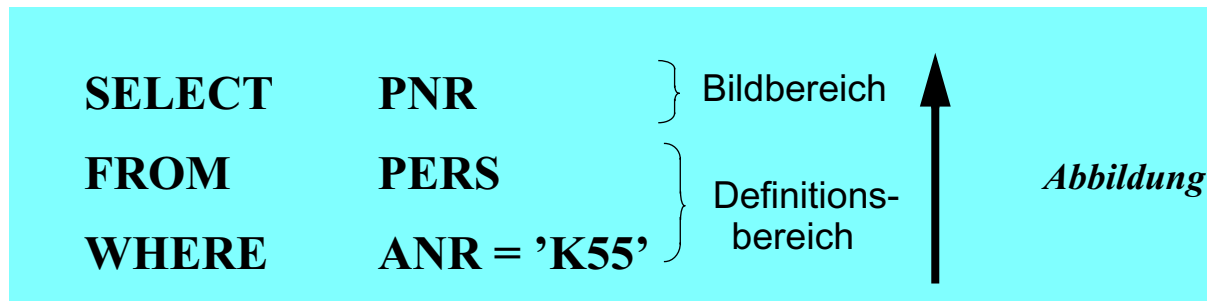


Abbildung: Ein bekanntes Attribut oder eine Menge von Attributen wird durch Angabe von Bedingungen (WHERE) mit Hilfe einer Relation (FROM) in ein gewünschtes Attribut oder eine Menge von Attributen abgebildet

Allgemeines Format

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

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	SQL RA	SQL	SQL	SQL
Eingebettete DB-Sprache	SQL	SQL	SQL	SQL

Befehlsübersicht (Auswahl)

Datenmanipulation (DML):

SELECT

INSERT

UPDATE

DELETE

Aggregatfunktionen: COUNT, SUM, AVG, MAX, MIN

Datenkontrolle:

Constraints-Definitionen bei CREATE TABLE

CREATE ASSERTION

DROP ASSERTION

GRANT

REVOKE

COMMIT

ROLLBACK

Datendefinition (DDL):

CREATE SCHEMA

CREATE DOMAIN

CREATE TABLE

CREATE VIEW

ALTER TABLE

DROP SCHEMA

DROP DOMAIN

DROP TABLE

DROP VIEW

Eingebettetes SQL:

DECLARE CURSOR

FETCH

OPEN CURSOR

CLOSE CURSOR

SET CONSTRAINTS

SET TRANSACTION

CREATE TEMPORARY TABLE

Anfragemöglichkeiten in SQL

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

select-item ::= derived-column | [range-variable.] *
derived-column ::= scalar-exp [AS column]
order-item ::= column [ ASC | DESC ]
```

Mit *SELECT* * werden alle Attribute der spezifizierten Relation(en) ausgegeben

FROM-Klausel spezifiziert die Objekte (Relationen, Sichten), die verarbeitet werden sollen

WHERE-Klausel kann eine Sammlung von Prädikaten enthalten, die mit *NOT*, *AND* und *OR* verknüpft sein können

dabei sind Vergleichsprädikate der Form $A_i \theta a_i$ bzw. $A_i \theta A_j$ möglich ($\theta \in \{=, <>, <, \leq, >, \geq\}$)

SQL-Training

<http://lots.uni-leipzig.de>

entwickelt am Lehrstuhl “Datenbanken”

Inhalt

- „freies Üben“ auf einer SQL-Datenbank
(nur SELECT-Anweisungen)
- „aktives“ SQL-Tutorial
- Übungsblätter mit Online-Bearbeitung

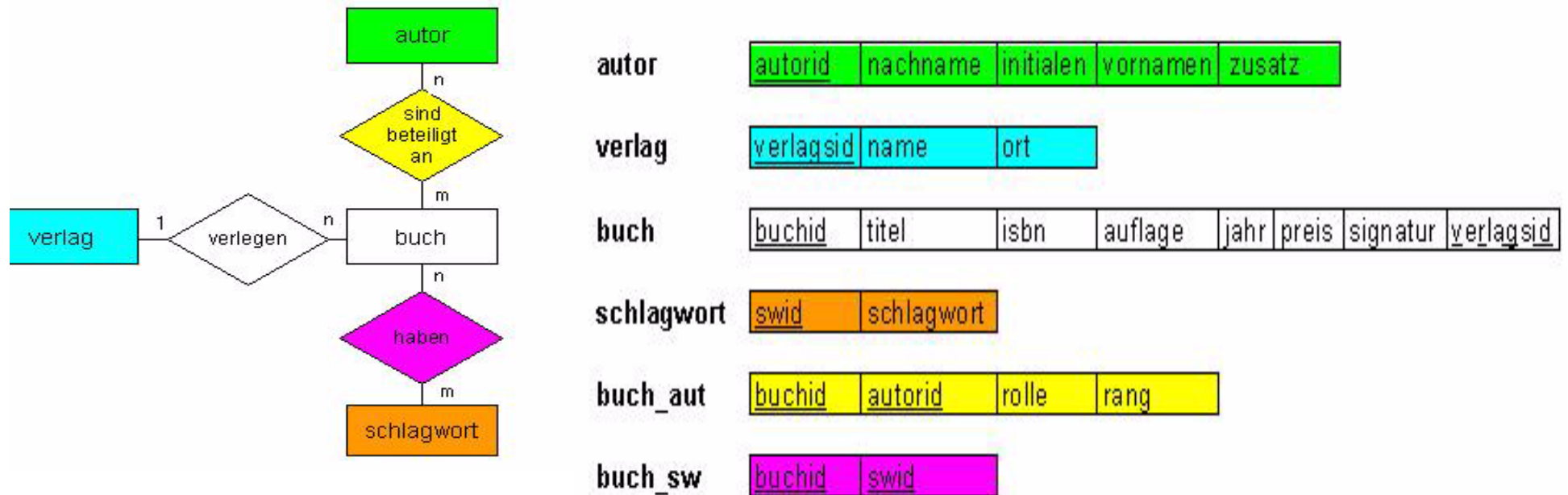
Realisierung auf Basis von Postgres

Gast-Login mit Zugriff auf 1 Datenbank
(user: gast, PW: gast)

Gruppe für Teilnehmer an Vorlesung:
WINF0506



Test-Datenbank



- *buch_aut.rolle* kann sein: Herausgebers (H), Verfasser (V), Übersetzer (U), Mitarbeiter (M)
- *buch_aut.rang*: Position des Autors in der Autorenliste (z.B. 1 für Erstautor)
- *autor.zusatz*: Namenszusatz wie „von“ oder „van“
- *buch.signatur* entspricht der Signatur in der IfI-Bibliothek (Stand 1998)

Mengengerüst (> 18.000 Sätze)

- "buch" : 4873 Datensätze, "verlag" : 831 Datensätze
- "autor" : 5045 Datensätze, "buch_aut" : 5860 Datensätze
- "schlagwort" : 843 Datensätze, "buch_sw" : 789 Datensätze

Einfache Selektionen und Projektionen

Q1: Welche (Berliner) Verlage gibt es?

Q2: Welche Bücher erschienen vor 1980 in einer Neuauflage?

Ausgabebearbeitung

Sortierte Ausgabe (ORDER BY-Klausel)

Q3: Q2, jedoch sortiert nach Jahr (absteigend), Titel (aufsteigend)

```
SELECT  
FROM  
WHERE
```

- ohne ORDER-BY ist die Reihenfolge der Ausgabe durch das DBS bestimmt (Optimierung der Auswertung)
- statt Attributnamen können in der ORDER BY-Klausel auch relative Positionen der Attribute aus der Select-Klausel verwendet werden

Duplikateliminierung

- Default-mäßig werden Duplikate in den Attributwerten der Ausgabeliste nicht eliminiert (ALL)
- *DISTINCT* erzwingt Duplikateliminierung

Q4: Welche Verlagsorte gibt es?

Ausgabebearbeitung (2)

Benennung von Ergebnis-Spalten

```
SELECT titel AS Buchtitel, (preis/2) AS Preis_in_Euro
FROM buch
WHERE waehrung = 'DM'
ORDER BY 2 DESC
```

- Umbenennung von Attributen (AS)
- Vergabe von Attributnamen für Texte und Ausdrücke

Umbenennung von Tabellen (FROM-Klausel)

- Einführung sogenannter Alias-Namen bzw. **Korrelationsnamen**
- Schlüsselwort AS optional
- Alias-Name überdeckt ursprünglichen Namen

```
SELECT B.titel
FROM buch AS B
WHERE B.preis > 300
```

Join-Anfragen

Q5: Welche Buchtitel wurden von Berliner Verlagen herausgebracht?

```
SELECT  
FROM  
WHERE
```

- Angabe der beteiligten Relationen in FROM-Klausel
- WHERE-Klausel enthält Join-Bedingung sowie weitere Selektionsbedingungen
- analoge Vorgehensweise für Equi-Join und allgemeinen Theta-Join
- Formulierung der Join-Bedingung erfordert bei gleichnamigen Attributen Hinzunahme der Relationennamen oder von Alias-Namen (Korrelationsnamen)

Q6: Welche Bücher sind von Autor „Rahm“ vorhanden?

```
SELECT  
FROM  
WHERE
```

Join-Anfragen (2)

Hierarchische Beziehung auf einer Relation (PERS)

Beispielschema:

```
PERS (PNR int, NAME, BERUF, GEHALT, ..., MNR int, ANR int,  
      PRIMARY KEY (PNR), FOREIGN KEY (MNR) REFERENCES PERS)
```

Q7: Finde die Angestellten, die mehr als ihre (direkten) Manager verdienen (Ausgabe: NAME, GEHALT, NAME des Managers)

```
SELECT  
FROM  
WHERE
```

- Verwendung von Korrelationsnamen obligatorisch!

Join-Ausdrücke

unterschiedliche Join-Arten können direkt spezifiziert werden

```
join-table-exp ::= table-ref [NATURAL] [join-type] JOIN table-ref  
                [ON cond-exp | USING (column-commalist) ]  
                | table ref CROSS JOIN table-ref | (join-table-exp)  
table-ref ::= table | (table-exp) | join-table-exp  
join type ::= INNER | { LEFT | RIGHT | FULL } [OUTER] | UNION
```

Beispiel:

```
SELECT *  
FROM buch B, verlag V  
WHERE B.verlagsid = V.verlagsid
```

```
buch NATURAL JOIN verlag  
buch JOIN verlag USING (verlagsid)  
buch B JOIN verlag V  
ON B.verlagsid = V.verlagsid
```

Q6: Welche Bücher sind von Autor „Rahm“ vorhanden?

Join-Ausdrücke (2)

Outer Joins: LEFT JOIN, RIGHT JOIN, FULL JOIN

```
schlagwort LEFT OUTER JOIN buch_sw USING (swid)
```

Kartesisches Produkt:

```
A CROSS JOIN B <=> SELECT * FROM A, B
```

Join-Ausdrücke (3)

Outer Union: UNION JOIN

- Vereinigung von nur teilweise vereinigungsverträglichen Relationen)

Beispiel:

STUDENT

Matnr	Name	Fak	Imm
123	Schmidt	MI	X
234	Müller	WiWi	Y

PERS

Pnr	Name	Fak	Wochenstunden
543	Schulz	WiWi	40
897	Weber	MI	20

STUDENT **UNION JOIN** PERS

Matnr	Name	Fak	Imm	Pnr	Wochenstunden
123	Schmidt	MI	X	-	-
234	Müller	WiWi	Y	-	-
-	Schulz	WiWi	-	543	40
-	Weber	MI	-	897	20

Geschachtelte Anfragen (Sub-Queries)

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

Q5: Welche Buchtitel wurden von Berliner Verlagen herausgebracht?

```
SELECT  
FROM  
WHERE
```

innere und äußere Relationen können identisch sein

eine geschachtelte Abbildung kann beliebig tief sein

Join-Berechnung mit Sub-Queries

- teilweise prozedurale Anfrageformulierung
- weniger elegant als symmetrische Notation
- schränkt Optimierungsmöglichkeiten des DBS ein

Sub-Queries (2)

Einfache Sub-Queries

- 1-malige Auswertung der Sub-Query
- Ergebnismenge der Sub-Query (Zwischenrelation) dient als Eingabe der äußeren Anfrage

Korrelierte Sub-Queries (verzahnt geschachtelte Anfragen)

- Sub-Query bezieht sich auf eine äußere Relation
- Sub-Query-Ausführung erfolgt für jedes Tupel der äußeren Relation
- Verwendung von Korrelationsnamen i.a. erforderlich

Welche Buchtitel wurden von Berliner Verlagen veröffentlicht?

```
SELECT B.titel
FROM buch B
WHERE 'Berlin' IN
  (SELECT V.ort
   FROM verlag V
   WHERE V.verlagsid = B.verlagsid)
```

```
SELECT B.titel
FROM buch B
WHERE B.verlagsid IN
  (SELECT V.verlagsid
   FROM verlag V
   WHERE V.ort = 'Berlin')
```

besser: Join-Berechnung ohne Sub-Queries

Benutzung von Aggregat (Built-in)-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 Angestellten, die mehr als ihre Manager verdienen.

```
SELECT  
FROM  
WHERE
```

Auswertung

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

Aggregatfunktionen (2)

keine Aggregatfunktionen in WHERE-Klausel

keine geschachtelte Nutzung von Funktionsreferenzen!

Q9: Wie viele Verlage gibt es?

```
SELECT  
FROM
```

Q10: An wievielen Orten gibt es Verlage?

```
SELECT  
FROM
```

Q11: An welchen Orten gibt es mehr als drei Verlage?

```
SELECT  
FROM  
WHERE
```

Q12: Welches Buch (Titel, Jahr) ist am ältesten?

```
SELECT  
FROM
```

Aggregatfunktionen (3)

Beispielschema:

```
PERS (PNR, NAME, BERUF, GEHALT, . . . , MNR, ANR)  
      PRIMARY KEY (PNR), FOREIGN KEY (MNR) REFERENCES PERS
```

Q13a: Wieviele Angestellte haben einen Vorgesetzten ?

```
SELECT  
FROM
```

Q13b: Wieviele Angestellte haben keinen Vorgesetzten ?

```
SELECT  
FROM
```

Q13c: Wieviele Angestellte sind Vorgesetzte ?

```
SELECT  
FROM
```

Partitionierung einer Relation in Gruppen

```
SELECT ... FROM ... [WHERE ...]  
[ GROUP BY column-ref-commalist ]
```

Gruppenbildung auf Relationen: GROUP-BY-Klausel

- Tupel mit übereinstimmenden Werten für Gruppierungsattribut(e) bilden je eine Gruppe
- ausgegeben werden können neben Gruppierungsattributen nur Konstanten sowie das Ergebnis von Aggregatfunktionen (-> 1 Satz pro Gruppe)
- die Aggregatfunktion wird jeweils auf die Tupeln einer Gruppe angewendet

Q14: Liste alle Abteilungen und das Durchschnitts- sowie Spitzengehalt ihrer Angestellten auf.

```
SELECT  
FROM
```

Q15: Liste alle Abteilungen (ANR und ANAME) sowie die Anzahl der beschäftigten Angestellten auf

```
SELECT  
FROM  
WHERE
```

Auswahl von Gruppen (HAVING-Klausel)

```
SELECT ... FROM ... [WHERE ...]  
[ GROUP BY column-ref-commalist ]  
[ HAVING cond-exp ]
```

Fragen werden in den folgenden Reihenfolge bearbeitet:

1. Tupeln werden ausgewählt durch die WHERE-Klausel.
2. Gruppen werden gebildet durch die GROUP-BY-Klausel.
3. Gruppen werden ausgewählt, wenn sie die HAVING-Klausel erfüllen

Q16: Für welche Abteilungen zwischen K50 und K60 ist das Durchschnittsalter kleiner als 30?

```
SELECT  
FROM  
WHERE
```

Q17: Bestimme die Namen und Gehaltssummen der Abteilungen mit mehr als 5 Mitarbeitern

```
SELECT  
FROM
```

Suchbedingungen

Sammlung von Prädikaten

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

nicht-quantifizierte Prädikate:

- Vergleichsprädikate
- LIKE-, BETWEEN-, IN-Prädikate
- Test auf Nullwert
- UNIQUE-Prädikat: Test auf Eindeutigkeit
- MATCH-Prädikat: Tupelvergleiche
- OVERLAPS-Prädikat: Test auf zeitliches Überlappen von DATETIME-Werten

quantifizierte Prädikate

- ALL
- ANY
- EXISTS

Vergleichsprädikate

comparison-cond ::= row-constructor θ row-constructor

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

Skalarer Ausdruck (scalar-exp): Attribut, Konstante bzw. Ausdrücke, die einfachen Wert liefern

Tabellen-Ausdruck (table-exp) darf hier höchstens 1 Tupel als Ergebnis liefern (Kardinalität 1, Row Sub-Query)

Vergleiche zwischen Tupel-Konstruktoren (row constructor) mit mehreren Attributen

- $(a_1, a_2, \dots, a_n) = (b_1, b_2, \dots, b_n) \Leftrightarrow a_1 = b_1 \text{ AND } a_2 = b_2 \dots \text{ AND } a_n = b_n$
- $(a_1, a_2, \dots, a_n) < (b_1, b_2, \dots, b_n) \Leftrightarrow (a_1 < b_1) \text{ OR } ((a_1 = b_1) \text{ AND } (a_2 < b_2)) \text{ OR } (\dots)$

SELECT

FROM

WHERE

LIKE-Prädikate

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

Unterstützung der Suche nach Datenstrings, von denen nur Teile bekannt sind (pattern matching).

LIKE-Prädikat vergleicht einen Datenwert mit einem "Muster" bzw. einer "Maske".

Aufbau einer Maske mit Hilfe zweier spezieller Symbole

% bedeutet "null oder mehr beliebige Zeichen"

_ bedeutet "genau ein beliebiges Zeichen"

- Das LIKE-Prädikat ist TRUE, wenn der entsprechende Datenwert der aufgebauten Maske mit zulässigen Substitutionen von Zeichen für "%" und "_" entspricht
- Suche nach "%" und "_" durch Voranstellen eines Escape-Zeichens möglich.

<i>LIKE-Klausel</i>	<i>wird erfüllt von</i>
NAME LIKE '%SCHMI%'	
ANR LIKE '_7%'	
NAME NOT LIKE '%-%'	
STRING LIKE '%_%' ESCAPE '\'	

BETWEEN-Prädikate

row-constr[NOT] BETWEENrow-constr AND row-constr

Semantik

y BETWEEN x AND z $\Leftrightarrow x \leq y$ AND $y \leq z$

y NOT BETWEEN x AND z \Leftrightarrow NOT (y BETWEEN x AND z)

Beispiel

```
SELECT NAME
FROM PERS
WHERE GEHALT BETWEEN 50000 AND 80000
```

IN-Prädikate

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

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

- *explizite Mengendefinition:* $A_i \text{ IN } (a_1, a_j, a_k)$
- *implizite Mengendefinition:* $A_i \text{ IN } (\text{SELECT } \dots)$

Semantik

$$x \text{ IN } (a, b, \dots, z) \quad \Leftrightarrow \quad x = a \text{ OR } x = b \dots \text{ OR } x = z$$
$$x \text{ NOT IN } \text{erg} \quad \Leftrightarrow \quad \text{NOT } (x \text{ IN } \text{erg})$$

Q18: Finde die Autoren mit Nachname Maier, Meier oder Müller

```
SELECT *  
FROM autor  
WHERE
```

Q19: Finde die Schlagworte, die nicht verwendet wurden

```
SELECT *  
FROM schlagwort  
WHERE
```

NULL-Werte

für jedes Attribut kann Zulässigkeit von Nullwerten festgelegt werden
unterschiedliche Bedeutungen:

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

Bei Auswertung von Booleschen Ausdrücken wird 3-wertige Logik eingesetzt

- Das Ergebnis ? bei der Auswertung einer WHERE-Klausel wird wie FALSE behandelt.

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

Behandlung von Nullwerten

- Auswertung eines NULL-Wertes in einem Vergleichsprädikat mit irgendeinem Wert ist UNKNOWN (?). Folglich nehmen Tupel mit NULL-Werten im join-Attribut nicht an einem Verbund teil.
- Das Ergebnis einer arithmetischen Operation (+, -, *, /) mit einem NULL-Wert ist ein NULL-Wert

spezielles Prädikat zum Test auf NULL-Werte:

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

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

NULL-Werte: Problemfälle

3-wertige Logik führt zu unerwarteten Ergebnissen

Bsp.: `PERS (Alter <= 50)` vereinigt mit `PERS (Alter > 50)`

ergibt nicht notwendigerweise Gesamtrelation PERS

Nullwerte werden bei `SUM`, `AVG`, `MIN`, `MAX` nicht berücksichtigt, während `COUNT(*)` alle Tupel (inkl. Null-Tupel, Duplikate) zählt.

```
SELECT  AVG  (GEHALT)  FROM  PERS    → Ergebnis X
```

```
SELECT  SUM  (GEHALT)  FROM  PERS    → Ergebnis Y
```

```
SELECT  COUNT (*)      FROM  PERS    → Ergebnis Z
```

Es gilt nicht notwendigerweise, daß $X = Y / Z$ (-> Verfälschung statistischer Berechnungen)

Quantifizierte Prädikate

All-or-Any-Prädikat

```
row-constr  $\theta$  { 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.

Q20: Finde die Manager, die mehr verdienen als alle ihre Angestellten

```
SELECT  
FROM  
WHERE
```

Q22b: Finde die Manager, die weniger als einer ihrer Angestellten verdienen

Existenztests

```
[ NOT ] EXISTS (table-exp)
```

das Prädikat wird zu "false" ausgewertet, wenn table-exp auf die leere Menge führt, sonst "true"

Im EXISTS-Kontext darf table-exp mit (SELECT * ...) spezifiziert werden (Normalfall)

Semantik

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

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

Q21: Finde die Manager, die mehr verdienen als alle ihre Angestellten.

```
SELECT  
FROM  
WHERE
```

Existenztests (2)

Q22: Finde die Schlagworte, die für mindestens ein (... kein) Buch vergeben wurden

```
SELECT S.*  
FROM schlagwort S  
WHERE
```

Q23: Finde die Bücher, die alle Schlagworte des Buchs mit der ID 3545 abdecken

(andere Formulierung: Finde die Bücher, zu denen kein Schlagwort "existiert", das nicht auch für Buch 3545 existiert).

```
SELECT B.titel  
FROM buch B  
WHERE
```

Skalare Funktionen (Auswahl)

CASE

```
SELECT MATNR, PUNKTE,  
       CASE WHEN PUNKTE > 90 THEN 'Sehr gut'  
            WHEN PUNKTE > 75 THEN 'Gut'  
            WHEN PUNKTE > 60 THEN 'O.K.'  
            ELSE 'SCHLECHT'      END AS NOTE  
FROM ...
```

- fehlender ELSE-Zweig: NULL-Wert für sonstige Fälle

String-Funktionen

- || (String-Konkatenation), CHAR_LENGTH, BIT_LENGTH
- SUBSTRING *Bsp.: SUBSTRING (NAME FROM 1 FOR 20)*
- TRANSLATE, CONVERT
- POSITION, LOWER, UPPER
- TRIM *Bsp.: TRIM (TRAILING ' ' FROM NAME)*

Verschiedene Funktionen

- USER, CURRENT_USER, SESSION_USER, SYSTEM_USER
- CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP
- EXTRACT (Herausziehen von YEAR, MONTH, ... aus Datum)
- CAST (Typkonversionen) *Bsp.: CAST ('2002-04-24' AS DATE)*
- NULLIF, COALESCE, ...

Einsatz mengentheoretischer Operatoren

Vereinigung (UNION), Durchschnitts- (INTERSECT) und Differenzbildung (EXCEPT) von Relationen bzw. Query-Ergebnissen

```
table-exp ::= nonjoin-table-exp | join-table-exp
nonjoin-table-exp ::= nonjoin-table-term | table-exp {UNION | EXCEPT}
    [ALL][CORRESPONDING [BY (column-commalist)]] table-term
nonjoin-table-term ::= nonjoin-table-primary | table-term INTERSECT
    [ALL][CORRESPONDING [BY (column-commalist)]] table-primary
table-term ::= nonjoin-table-term | join-table-exp
table-primary ::= nonjoin-table-primary | join-table-primary
nonjoin-table-primary ::= simple-table | select-exp | (nonjoin-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 ist auf Attribute A_i beschränkt, die in beiden Relationen vorkommen müssen (-> n-stelliges Ergebnis)
- *CORRESPONDING*: Operation ist auf gemeinsame Attribute beschränkt

Q24: Welche Schlagworte wurden nie verwendet ? (Q19, Q22)

Weitergehende Verwendung von Sub-Queries (table expressions)

3 Arten von Sub-Queries

- Table Sub-Queries (mengenwertige Ergebnisse)
- Row Sub-Queries (Tupel-Ergebnis)
- Skalare Sub-Queries (atomarer Wert; Kardinalität 1, Grad 1)

Im SQL-Standard können Table-Sub-Queries überall stehen, wo ein Relationenname möglich ist, insbesondere in der FROM-Klausel.

```
SELECT  ANAME
FROM    (Select ANR, Sum (GEHALT) AS GSUMME FROM PERS GROUP BY ANR)
        AS GSUM JOIN ABT USING (ANR)
WHERE   GSUMME > 100000
```

Skalare Sub-Queries können auch in SELECT-Klausel stehen.

```
SELECT  P.PNAME, (SELECT  A.ANAME FROM ABT A
                  WHERE   A.ANR = P.ANR) AS ABTEILUNG
FROM    PERS P
WHERE   BERUF = "Programmierer"
```

Relationenalgebra vs. SQL (Retrieval)

Relationenalgebra	SQL
$\pi_{A_1, A_2, \dots, A_k}(R)$	
$\sigma_P(R)$	
$R \times S$	
$R \bowtie_{R.A \theta S.B} S$	
$R \bowtie S$	
$R \sqsupseteq S$	
$R \boxtimes S$	
$R \cup S$	
$R \cap S$	
$R - S$	
$R \div S$	

Zusammenfassung

SQL wesentlich mächtiger als Relationenalgebra

Hauptanweisung: SELECT

- Projektion, Selektion, Joins
- Aggregatfunktionen
- Gruppenbildung (Partitionierungen)
- quantifizierte Anfragen
- Unteranfragen (einfache und korrelierte Sub-Queries)
- allgemeine Mengenoperationen UNION, INTERSECT, EXCEPT

hohe Sprachredundanz

SQL-Implementierungen weichen teilweise erheblich von Standard ab
(Beschränkungen / Erweiterungen)