

5. Anfragesprache SQL

- Grundlagen: Anfragekonzept, Befehlsübersicht
- SELECT: mengenorientierte Anfragen deskriptiver Art
 - Selektions- und Projektionsanfragen, Join-Anfragen
 - geschachtelte Anfragen (Sub-Queries)
 - Aggregatfunktionen
 - Gruppenanfragen (GROUP BY, HAVING)
 - Prädikate LIKE, BETWEEN, IN, IS NULL
 - quantifizierte Prädikate (ALL/ANY, EXISTS)
 - Mengen-Operationen: UNION, INTERSECT, EXCEPT
- Änderungsoperationen INSERT, DELETE, UPDATE, MERGE
- Vergleich mit der Relationenalgebra

Entwicklung von SQL

- unterschiedliche Entwürfe für relationale Anfragesprachen
 - SEQUEL: Structured English Query Language (System R) -> SQL
 - QUEL (Ingres), . . .
- SQL: vereinheitlichte Sprache für alle DB-Aufgaben
 - 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: „SQL2“ bzw. SQL-92 (Entry, Intermediate, Full Level)
 - 1999 und später: SQL:1999 („SQL3“), SQL:2003, SQL:2008, SQL:2011 u.a. mit objektorientierten Erweiterungen (-> objekt-relationale DBS), XML-Unterstützung, temporalen Anfragen etc.

Abbildungsorientierte Anfragen in SQL

- SQL: strukturierte Sprache, die auf englischen Schlüsselwörtern basiert
 - Zielgruppe umfasst auch Nicht-Programmierer
 - Auswahlvermögen umfasst das der Relationenalgebra (relational vollständig)
- Grundbaustein

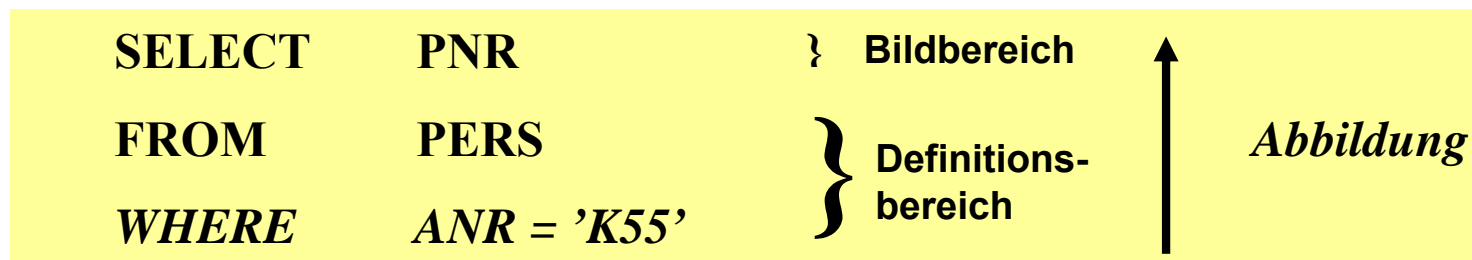


Abbildung: Eingaberelationen (FROM) werden unter Auswertung von Bedingungen (WHERE) in Attribute einer Ergebnistabelle (SELECT) abgebildet

■ Allgemeines Format

<Spezifikation der Operation>

<Liste der referenzierten Tabellen>

[WHERE Boolescher Prädikatsausdruck]

Erweiterungen zu einer vollständigen DB-Sprache

■ Datenmanipulation

- Einfügen, Löschen und Ändern von individuellen Tupeln und von Mengen von Tupeln
- Zuweisung von ganzen Relationen

■ Datendefinition

- Definition von Wertebereichen, Attributen und Relationen
- Definition von verschiedenen Sichten auf Relationen

■ 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	Datendefinition	Datenkontrolle
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

SQL-Training in LOTS (<http://lots.uni-leipzig.de>)

- „freies Üben“ auf einer SQL-Datenbank (SELECT-Anweisungen)
 - Realisierung auf Basis von Postgres
- „aktives“ SQL-Tutorial

Tutorial

- 1 Einleitung
- 2 Datenbankmodellierung und Relationenmodell
- 3 SQL
- 4 Einfache SQL-Anfragen
 - 4.1 Vorbemerkungen
 - 4.2 Einfache Selektion und Projektion
 - 4.3 Ausgabebearbeitung
- 5 Verbund-Anfragen
- 6 Unterabfragen
- 7 Aggregatfunktionen
- 8 Partitionierung in Gruppen und Auswahl
- 9 Suchbedingungen
- 10 Mengentheoretische Operationen
- 11 Datendefinition
- 12 Datenmanipulation auf

[Zurück](#) [Weiter](#) [Hoch](#) | [zurück zum SQL-Anfrageformular](#)

4.2 Einfache Selektion und Projektion

Die Projektion aus der [Relationenalgebra](#) stellt sich dar als Auflistung der Attribute nach dem Wort `SELECT`. Folgende SQL-Anfrage repräsentiert eine sehr einfache Projektion.

Beispiel:

BNF: [select-ausdruck](#) **diese Anfrage ausführen**

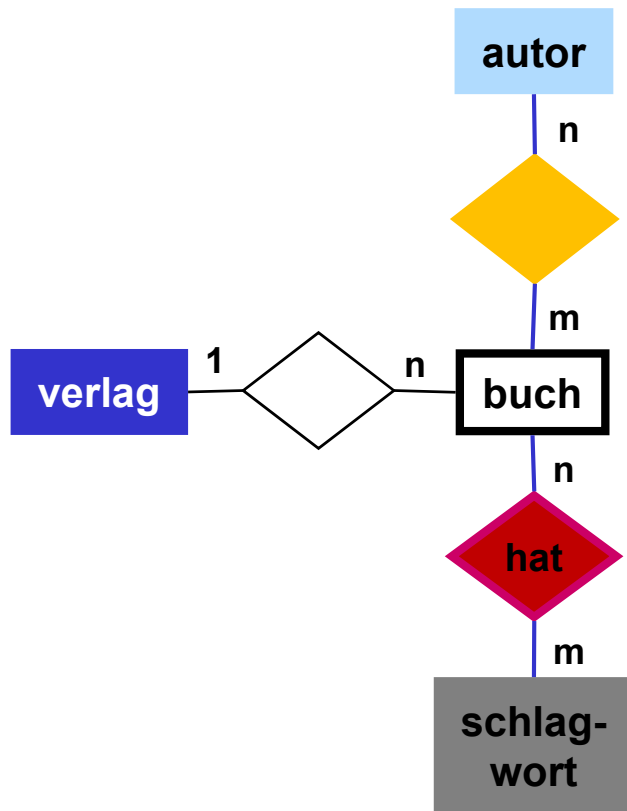
```
SELECT name
FROM verlag
```

Erklärung:

Es werden alle Attributwerte des Attributes "name" aus der Relation "verlag" ausgegeben. In der Reihenfolge der Zeilen ist keine Information codiert. Die Zeilen werden so ausgegeben, wie sie sich (zufällig) in der Relation befinden.

Eine Selektion der [Relationenalgebra](#) ist eigentlich eine Auswahl von Zeilen in der Relation. In der `WHERE`-Klausel werden logische Bedingungen angegeben. Für jede Zeile der Tabelle wird ausgewertet, ob die Bedingung erfüllt ist. Bei positivem Ergebnis wird die Zeile der Auswahlmenge hinzugefügt, andernfalls ignoriert.

Test-Datenbank



autor | autorid | nachname | initialen | vornamen | zusatz

verlag | verlagsid | name | ort

buch | buchid | titel | isbn | auflage | jahr | preis | waehrung | signatur | verlagsid

schlagwort | swid | schlagwort

buch_aut | buchid | autorid | rolle | rang

buch_sw | buchid | swid

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 (ca. 18.000 Sätze)

- „buch“ : 4873 Datensätze, „verlag“ : 414 Datensätze
- „autor“ : 5045 Datensätze, „buch_aut“ : 5860 Datensätze
- „schlagwort“ : 843 Datensätze, „buch_sw“ : 789 Datensätze

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 ]
```

- *SELECT* –Klausel spezifiziert auszugebende Attribute
 - mit *SELECT ** werden alle Attribute der spezifizierten Relation(en) ausgegeben
- *FROM*-Klausel spezifiziert zu verarbeitende Objekte (Relationen, Sichten)
- *WHERE*-Klausel kann eine Sammlung von Suchprädikaten 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
mit $\theta \in \{ =, <>, <, \leq, >, \geq \}$

Einfache Selektionen und Projektionen

Q1: Welche (Berliner) Verlage gibt es?

Q2: Welche Bücher erschienen vor 1980 in einer Neuauflage?

SQL Anfrage

```
SELECT Titel, Jahr
FROM Buch
WHERE jahr < 1980 and auflage > 1
```

ausführen löschen

akt

Bibl
[DB](#)

Zwi

zeige Datensätze 1 - 11 (11 insgesamt)

Zeige: Datensätze, beginnend ab

titel	jahr
List Processing	1968
Moderne Logik : Abriß der formalen Logik	1972
Logic and logic design	1973
Wörterbuch der Datenverarbeitung : Englisch-Deutsch	1968
Introduction to switching theory and logical design	1974
Prädikatenkalkül der ersten Stufe	1975
The theory of error-correcting codes	1977
Computer data-base organisation	1977
Grundzüge der theoretischen Logik	1972
Matrizen und ihre technischen Anwendungen	1961
Introduction to switching theory and logical design	1974

Zeige: Datensätze, beginnend ab



Sortierte Ausgabe

- ORDER BY-Klausel zur aufsteigenden (ascending) oder absteigenden (descending) Sortierung von Ergebnissätzen
 - 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

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

```
SELECT  
FROM  
WHERE
```

SQL Anfrage

```
SELECT titel, jahr
FROM buch
WHERE jahr < 1980 and auflage > 1
ORDER BY 2 desc, 1 asc
```


aktuelle

Bibliothe
[DB Sche](#)

Zwische

zeige Datensätze 1 - 11 (11 insgesamt)

Zeige:

Datensätze, beginnend ab

titel	jahr
Computer data-base organisation	1977
The theory of error-correcting codes	1977
Prädikatenkalkül der ersten Stufe	1975
Introduction to switching theory and logical design	1974
Introduction to switching theory and logical design	1974
Logic and logic design	1973
Grundzüge der theoretischen Logik	1972
Moderne Logik : Abriß der formalen Logik	1972
List Processing	1968
Wörterbuch der Datenverarbeitung : Englisch-Deutsch	1968
Matrizen und ihre technischen Anwendungen	1961

Zeige:

Datensätze, beginnend ab

Duplikateliminierung

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

Q4: Welche Verlagsorte gibt es?

zeige Datensätze 1 - 20 (231 insgesamt)

Zeige: 20 Datensätze, beginnend ab 21 > >>

ort
Aachen
Aldershot
Amsterdam
Amsterdam [u. a.]
Arlington/ Va.
Attenkirchen [u. a.]
Augsburg
Aylesbury
Bad Honnef
Baden-Baden [u. a.]
Baltimore, Md. [u. a.]
Basel, Boston [u. a.]
Bath
Bellingham, Wash.
Belmont, Calif.
Bergheim
Berkeley [u. a.]
Berlin
Berlin [u. a.]

Zeige: 20 Datensätze, beginnend ab 21 > >>



Umbenennungen

■ 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
```

Skalare Funktionen: CASE

■ CASE

```
SELECT titel, jahr, CASE
    WHEN jahr > 2008 THEN 'Aktuell'
    WHEN jahr > 1991 THEN 'Mittel'
    ELSE 'Veraltet' END AS aktualitaet
FROM buch
```

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

zeige Datensätze 1 - 20 (4.877 insgesamt)

Zeige: 20 Datensätze, beginnend ab 21 > >>

titel	jahr	aktualitaet
Algorithmen in C++	1994	Mittel
Spreadsheets: Tabellenkalkulation für Naturwissenschaftler	1992	Mittel
C und Assembler in der Systemprogrammierung	1992	Mittel
Programmiersysteme für elektronische Rechenanlagen	1967	Veraltet
LATEX. Kompaktführer	1991	Mittel
Windows für Workgroups 3.11	1994	Mittel

Weitere skalare Funktionen (Auswahl)

■ String-Funktionen

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

■ Zeit/Datumsfunktionen

- CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP
- EXTRACT (Herausziehen von YEAR, MONTH, ... aus Datum)
- CAST (Typkonversionen) *Bsp.:* CAST ('2009-04-24' AS DATE) ...

Join-Anfragen

Q5: Welche Buchtitel wurden von Berliner Verlagen herausgebracht?

```
SELECT  
FROM  
WHERE
```

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

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

```
SELECT  
FROM buch B, autor A, buch_aut BA  
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!

<u>PNR</u>	NAME	GEHALT	MNR
34	Mey	32000	37
35	Schultz	42500	37
37	Abel	41000	-

<u>PNR</u>	NAME	GEHALT	MNR
34	Mey	32000	37
35	Schultz	42500	37
37	Abel	41000	-

Join-Ausdrücke

■ Join-Spezifikation direkt in FROM-Klausel

```
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
```

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

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

SQL Anfrage

```
SELECT titel, auflage, jahr
FROM buch b, buch_aut ba, autor a
WHERE nachname='Rahm' AND
      b.buchid = ba.buchid AND
      ba.autorid = a.autorid
```

SQL Anfrage

```
SELECT titel, auflage, jahr
FROM buch NATURAL JOIN buch_aut
      NATURAL JOIN autor a
WHERE nachname='Rahm'
```

zeige Datensätze 1 - 7 (7 insgesamt)

Zeige:

7

Datensätze, beginnend ab

1

titel	auflage	jahr
Datenbanksysteme - Konzepte und Techniken der Implementierung	2. Auflage	2001
Mehrrechner-Datenbanksysteme : Grundlagen der verteilten und parallelen Datenbankverarbeitung	1. Aufl.	1994
Synchronisation in Mehrrechner-Datenbank-Systemen : Konzepte, Realisierungsformen und quantitative Bewertung		1988
Datenbanksysteme - Konzepte und Techniken der Implementierung	1. Auflage	1999
Hochleistungs-Transaktionssysteme : Konzepte und Entwicklungen moderner Datenbankarchitekturen		1993
Mehrrechner-Datenbanksysteme. Grundlagen der verteilten und parallelen Datenbankverarbeitung		1997
Web & Datenbanken		2002

Zeige:

7

Datensätze, beginnend ab

1

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
```

Geschachtelte Anfragen (Sub-Queries)

- Auswahlbedingungen können sich auf das Ergebnis einer „inneren“ Anfrage (Sub-Query, Unteranfrage) beziehen

Q5‘: Welche Buchtitel wurden von Berliner Verlagen herausgebracht?

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

- 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

Q5“: 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

- Sub-Queries sind nützlich zur Berechnung komplexer Vergleichswerte in WHERE-Klausel, z.B. durch Anwendung von Aggregatfunktionen in der Sub-Query

Weitergehende Verwendung von Sub-Queries

■ 3 Arten von Sub-Queries

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

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

```
SELECT titel
FROM (Select * from Verlag where Ort='Leipzig') as LVERLAG
      NATURAL JOIN buch
WHERE Jahr > 1980
```

- **skalare Sub-Queries** können auch **in SELECT-Klausel** stehen

```
SELECT titel, (Select name FROM verlag v
              where v.verlagsid=b.verlagsid) AS Verlag
FROM Buch b
WHERE Jahr =1998
```

SQL Anfrage

```
SELECT titel, (Select name FROM verlag v where  
v.verlagsid=b.verlagsid) AS verlag  
FROM buch b  
WHERE jahr =1998
```

[zurück zum Tutorial](#)

aktuelle DB

Bibliothek

[DB Schema](#)

Zwischenablage

zeige Datensätze 1 - 20 (69 insgesamt)

Zeige:

20

Datensätze, beginnend ab

21

>

>>

titel	verlag
Rechnerarchitektur : Aufbau, Organisation und Implementierung	Vieweg
Software-Management, Software-Qulitätssicherung, Unternehmensmodellierung	
Java™ programming with CORBA : [advanced techniques for building distributed applications]	Wiley
Corel WordPerfect 8 im professionellen Einsatz	Hanser
Bestimmung des Stichprobenumfangs für biologische Experimente und kontrollierte klinische Studien	Oldenbourg
INFORMIX Universal Server : das objekt-relationale Datenbanksystem, mit OnLine-XPS und ODS	Addison Wesley

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 aller Angestellten

```
SELECT  AVG (GEHALT)  
FROM    PERS
```

■ 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)

Q9: Wie viele Verlage gibt es?

```
SELECT  
FROM
```

Q10: An wie vielen Orten gibt es Verlage?

```
SELECT  
FROM
```

Q11: Für wie viele Bücher ist der Verlag bekannt?

```
SELECT  
FROM
```

Q12: Für wie viele Bücher ist der Verlag nicht bekannt?

```
SELECT  
FROM
```

Q13: Zu wie vielen Verlagen gibt es Bücher ?

```
SELECT  
FROM
```



SQL Anfrage

```
SELECT count (DISTINCT ort)
FROM verlag
```

ausführen

löschen

zeige Datensätze 1 - 1 (1 insgesamt)

Zeige:

1

Datensätze, beginnend ab

1

count

231

Zeige:

1

Datensätze, beginnend ab

1

Aggregatfunktionen (3)

- Aggregatfunktionen können nicht direkt in WHERE-Klausel verwendet werden
 - Verwendung von Sub-Queries

Q14: Welches Buch (Titel, Jahr) ist am ältesten?

```
SELECT Titel, Jahr
FROM Buch
```

Q15: An welchen Orten gibt es mehr als drei Verlage?

```
SELECT
FROM
WHERE
```


SQL Anfrage

```
SELECT DISTINCT ort
FROM verlag v
WHERE 3 < (SELECT count(*)
          FROM verlag v2
          WHERE v2.ort=v.ort)
```

ausführen

löschen

aktuelle DB

Bibliothek
[DB Schema](#)

Zwischenablage

löschen

zeige Datensätze 1 - 20 (23 insgesamt)

Zeige:

20

Datensätze, beginnend ab

21

>

>>

ort

Amsterdam [u. a.]
Augsburg
Berlin
Berlin [u. a.]
Düsseldorf
Hamburg
Heidelberg
Indianapolis, Ind.
Konstanz
London
London [u. a.]
Mannheim [u. a.]
München
München [u. a.]
New York
New York [u. a.]
Oxford
Sankt Augustin
Stuttgart
Stuttgart [u. a.]

Zeige:

20

Datensätze, beginnend ab

21

>

>>

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 nur:
Gruppierungsattribute, Konstante, Ergebnis von Aggregatfunktionen (-> 1 Satz pro Gruppe)
- die Aggregatfunktion wird jeweils auf die Tupeln einer Gruppe angewendet

Q16: Liste alle Verlage mit der Anzahl ihrer Bücher auf

```
SELECT verlagsid  
FROM buch
```

Ausgabe von Verlagsname?

SQL Anfrage

```
select name, count(*)  
from buch natural join verlag  
group by name order by 2 desc
```

zeige Datensätze 1 - 20 (403 insgesamt)

Zeige: 20 Datensätze, beginnend ab 21

name	count
Springer	983
Addison Wesley	544
Prentice Hall	223
Vieweg	164
Oldenbourg	146
Teubner	141
Markt & Technik	116
Hanser	107
O'Reilly	106
BI-Wiss.-Verl.	102
Thomson	82
MIT Press	80

Group-By (2)

Q17: Liste alle Abteilungen und das Durchschnitts- sowie Spitzengehalt ihrer Angestellten auf.

```
SELECT
FROM PERS
```

PERS

<u>PNR</u>	Name	Alter	Gehalt	ANR
235	Schmid	31	32500	K51
123	Schulz	32	43500	K51
237	Bauer	21	21000	K53
234	Meier	23	42000	K53
829	Müller	36	42000	K53
321	Klein	19	27000	K55
406	Abel	47	52000	K55
574	Schmid	28	36000	K55

Auswahl von Gruppen (HAVING-Klausel)

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

- **HAVING**: Bedingungen nur bezüglich Gruppierungsattribut(en)
 - meist Verwendung von Aggregatfunktionen
- 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

Q18: Für welche Abteilungen in Leipzig ist das Durchschnittsalter kleiner als 30?

```
SELECT  
FROM  
WHERE
```

HAVING-Klausel (2)

Q19: Bestimme die Gehaltssummen der Abteilungen mit mehr als 5 Mitarbeitern

```
SELECT  
FROM
```

Q15': An welchen Orten gibt es mehr als drei Verlage? „Profi-Version“

```
SELECT ORT  
FROM VERLAG
```

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  $\theta$  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) für Row Sub-Query**
 - darf hier höchstens 1 Tupel als Ergebnis liefern (Kardinalität 1)
- **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 ...
```

```
WHERE („Leipzig“, 2000) =  
      (Select Ort, Gründungsjahr FROM Verein ... )
```

LIKE-Prädikate

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

- Suche nach Strings, von denen nur Teile bekannt sind (pattern matching)
- LIKE-Prädikat vergleicht einen Datenwert mit einem „Muster“ („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 z.B. erfüllt von</i>
NAME LIKE '%SCHMI%'	'H. SCHMIDT' 'SCHMIED'
ANR LIKE '_7%'	'17' 'K75'
NAME NOT LIKE '%-%'	
STRING LIKE '%_%' ESCAPE '\'	

BETWEEN-Prädikate

```
row-constr [ NOT] BETWEEN row-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      ANR  
FROM        PERS  
WHERE       ANR NOT BETWEEN 'K50' AND 'K54'  
GROUP BY   ANR  
HAVING      AVG (Alter) BETWEEN 20 AND 35
```

SQL Anfrage

```
select ort, count(*) from verlag
group by ort
having ort between 'Berlin' and 'Leipzig'
```

zeige Datensätze 1 - 20 (108 insgesamt)

Zeige:

20

Datensätze, beginnend ab 21

>

>>

ort	count
Chichester, New York [u.a.]	1
Englewood Cliffs, Englewood Cliffs, NJ, München, London, Englewood Cliffs, N.J., Scarborough, Canada, New York, Upper Saddle River, NJ, Sydney, Mountain View, Calif., Hemel Hempstead, 49, Upple Saddle River, NJ, Ontario, Singapore, Upper Saddle River, Upper Saddle, River, NJ [u.a.]	1
Haar	2
Ghent	1
Düsseldorf	7
Heidelberg, Berlin [u.a.]	1
Chichester [u.a.]	1
Bonn, Reading, Mass., München, Wokingham, Redwood City, Calif., Amsterdam, Redwood, Calif., Sydney, Menlo Park, Calif., Reading, Mass., Berlin, Harlow, Harlowe, Reading, Mass. [u.a.]	1
Korschenbroich	1
Ehningen, Renningen-Malmsheim	1
Frankfurt/M., Frankfurt/Main [u.a.]	1
Bonn, Reading, Mass. [u.a.]	1
Hildesheim [u.a.]	2
Bornheim	1
Landsberg/Lech	1
Frankfurt a.M., Frankfurt am Main [u.a.]	1
Leipzig	1
Hamburg	5
Charleston, SC, Charlston, SC	1
Kaiserslautern	2

Zeige:

20

Datensätze, beginnend ab 21

>

>>

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

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

Q20: Finde die Autoren mit Nachname Maier, Meier oder Müller

```
SELECT *
FROM autor
WHERE NACHNAME IN („Maier“, „Meier“, „Müller“)
```

Q21: Finde die Schlagworte, die nicht verwendet wurden

```
SELECT *
FROM schlagwort
WHERE
```

NULL-Werte

- pro Attribut kann Zulässigkeit von Nullwerten festgelegt werden
 - unterschiedliche Bedeutungen: Datenwert ist momentan nicht bekannt
 - Attributwert existiert nicht für ein Tupel
- Behandlung von Nullwerten
 - das Ergebnis einer arithmetischen Operation (+, -, *, /) mit einem NULL-Wert ist ein NULL-Wert
 - Tupel mit NULL-Werten im Verbundattribut nehmen am Verbund nicht teil
 - Auswertung eines NULL-Wertes in einem Vergleichsprädikat mit irgendeinem Wert ist UNKNOWN (?)
- bei Auswertung von Booleschen Ausdrücken wird 3-wertige Logik eingesetzt
 - das Ergebnis ? bei der Auswertung einer WHERE-Klausel wird wie FALSE behandelt.

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	?	?

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 $X = Y / Z$
(-> ggf. Verfälschung statistischer Berechnungen)

- spezielles SQL-Prädikat zum Test auf NULL-Werte:

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

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

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.

Q22: Finde die Manager, die mehr verdienen als alle ihre Angestellten

```
SELECT M.PNR  
FROM PERS M  
WHERE M.Gehalt
```

Q23: Finde die Manager, die weniger als einer ihrer Angestellten verdienen

Existenztests

[NOT] EXISTS (table-exp)

- Prädikat wird "false", wenn table-exp auf die leere Menge führt, sonst "true"
- im EXISTS-Kontext darf table-exp mit (SELECT * ...) spezifiziert werden (Normalfall)
- Semantik

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

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

$x \theta \text{ ALL } (\text{SELECT } y \text{ FROM } T \text{ WHERE } p) \Leftrightarrow$

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

Q22': Finde die Manager, die mehr verdienen als alle ihre Angestellten.

```
SELECT M.PNR
FROM PERS M
WHERE
```

Existenztests (2)

Q24: Finde die Schlagworte, die für mindestens ein (... kein) Buch vergeben wurden

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

Q25: 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
```

SQL Anfrage

```
SELECT buchid, titel
FROM buch b
WHERE NOT EXISTS
  (SELECT *
   FROM buch_sw
   WHERE buchid=3545 AND
        swid NOT IN
  (SELECT swid
   FROM buch_sw bsw
   WHERE b.buchid=bsw.buchid) )
```

ausführen

löschen

aktuelle DB

Bibliothek
[DB Schema](#)

Zwischenablage

löschen

zeige Datensätze 1 - 2 (2 insgesamt)

Zeige:

2

Datensätze, beginnend ab

1

buchid	titel
3545	Design and implementation of symbolic computation systems : international symposium, DISCO 92, Bath, U.K. , April 13 - 15, 1992 , proceedings
4443	Design and implementation of symbolic computation systems : international symposium, DISCO 93, Gmunden, Austria, September 15 - 17, 1993 , proceedings

Zeige:

2

Datensätze, beginnend ab

1

Division: Beispiel

Q26: Welche Lieferanten (LNR) haben alle Teile geliefert?

$\pi_{\text{LNR, TNR}} (\text{LIEFERUNG}) \div \pi_{\text{TNR}} (\text{TEIL})$

```
SELECT  
FROM  
WHERE
```

Einsatz von Mengen-Operatoren

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

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

- vor Ausführung werden Duplikate entfernt (außer für ALL)
- für die Operanden werden Vereinigungsverträglichkeit und übereinstimmende Attributnamen gefordert (ggf. vorher umbenennen)
- 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

Q21‘: Welche Schlagworte wurden nie verwendet ? (Q24‘)

(SELECT swid FROM schlagwort) **EXCEPT**

(SELECT swid FROM buch-sw)

SQL Anfrage

```

SELECT *
FROM buch as b1
WHERE verlagsid IS NULL

UNION

SELECT *
FROM buch as b2
WHERE isbn IS NULL

```

[zurück zum Tutorial](#)

aktuelle DB

Bibliothek

[DB Schema](#)

Zwischenablage

zeige Datensätze 1 - 20 (427 insgesamt)

Zeige: Datensätze, beginnend ab

buchid	titel	isbn	auflage	jahr	preis	wahrung	signatur	verlagsid
4	Programmiersysteme für elektronische Rechenanlagen			1967			P 2132	5
33	Sparse distributed memory			1988			K 4866	16
39	A comprehensive guide to AI and expert systems			1986			K 4103	29
47	Berechnungstheorie für Informatiker			1988			M 4446	20
62	Handbuch DERIVE : Der Mathematik Assistent für Ihren			1992			M 5202	41

Einfügen von Tupeln (INSERT)

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

■ satzweises Einfügen (direkte Angabe der Attributwerte)

Bsp.: Füge den Schauspieler Garfield mit der PNR 4711 ein

- alle nicht angesprochenen Attribute erhalten Nullwerte
- falls alle Werte in der richtigen Reihenfolge versorgt werden, kann Attributliste entfallen (NICHT zu empfehlen)
- Integritätsbedingungen müssen erfüllt werden

INSERT (2)

- **mengenorientiertes Einfügen**: einzufügende Tupeln werden aus einer anderen Relation mit Hilfe einer SELECT-Anweisung ausgewählt

Bsp.: Füge die Schauspieler aus L in die Relation TEMP ein

- (leere) Relation TEMP mit kompatiblen Attributen sei vorhanden
- die spezifizierte Tupelmenge wird ausgewählt und in die Zielrelation kopiert
- die eingefügten Tupel sind unabhängig von denen, von denen sie abgeleitet/kopiert wurden.

Ändern von Tupeln (UPDATE)

```
searched-update ::=      UPDATE table
                        SET update-assignment-commalist
                        [WHERE cond-exp]
update-assignment ::= column = {scalar-exp | DEFAULT | NULL }
```

*Gib den Schauspielern am Schauspielhaus eine Gehaltserhöhung von 2%
(Attribute GEHALT und THEATER seien in SCHAUSPIELER).*

Löschen von Tupeln (DELETE)

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

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

Lösche den Schauspieler mit der PNR 4711

Lösche alle Schauspieler, die nie gespielt haben.

```
DELETE FROM SCHAUSPIELER  
WHERE
```

MERGE (seit SQL:2003)

- Kombination von Insert/Update/Delete beim Mischen von Tabellen

```
MERGE INTO table1 USING table2 | table-exp ON cond-expr
WHEN MATCHED THEN
    UPDATE SET update-assignment-commalist [WHERE cond-expr]
    [ DELETE WHERE cond-expr ]
WHEN NOT MATCHED THEN
    INSERT (column-commalist) VALUES row-constr-commalist
    [WHERE cond-expr]
```

KURS	AUFGABEN_NR	STUDENTEN_NR	PUNKTE
C1	1	S1	56
C1	2	S1	63
C1	1	S2	88
C1	2	S2	91
C2	1	S4	49

Tabelle 1: Tabelle PUNKTE

KURS	AUFGABEN_NR	STUDENTEN_NR	PUNKTE
C1	1	S1	57
C1	3	S1	69
C1	2	S2	81
C2	1	S3	95

Tabelle 2: Tabelle NEUE_PUNKTE

```
MERGE INTO PUNKTE AS P
USING NEUE_PUNKTE AS N
ON P.KURS = N.KURS
    AND P.AUFGABEN_NR = N.AUFGABEN_NR
    AND P.STUDENTEN_NR = N.STUDENTEN_NR
WHEN MATCHED THEN UPDATE
    SET PUNKTE = N.PUNKTE
WHEN NOT MATCHED THEN INSERT
    VALUES (N.KURS, N.AUFGABEN_NR,
            N.STUDENTEN_NR, N.PUNKTE)
```

Relationenalgebra vs. SQL (Retrieval)

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

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
- Datenänderungen: INSERT, UPDATE, DELETE
- hohe Sprachredundanz
- SQL-Implementierungen weichen teilweise erheblich von Standard ab (Beschränkungen / Erweiterungen)