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 gelegentliche Benutzer
  - mächtige Sprachkonstrukte für besser ausgebildete Benutzer
  - spezielle Sprachkonstrukte für 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, SQL:2016 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
  - *relational vollständig:* Auswahlvermögen umfasst das der Relationenalgebra
- Grundbaustein

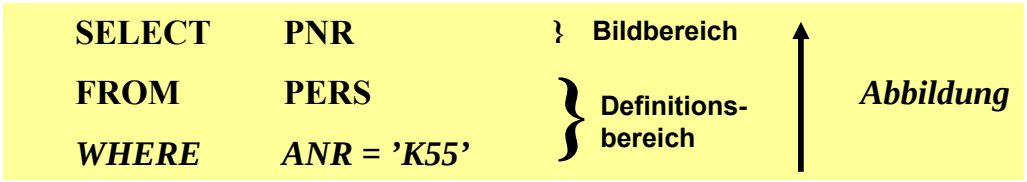


Abbildung: Eingaberelationen (FROM) werden unter Auswertung von Bedingungen (WHERE) in Attribute einer Ergebnistabelle (SELECT) abgebildet

- 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

MERGE
Aggregatfunktionen: COUNT, SUM, AVG, MAX, MIN

Datendefinition (DDL):

CREATE SCHEMA
CREATE DOMAIN
CREATE TABLE
CREATE VIEW
ALTER TABLE
DROP SCHEMA
DROP DOMAIN
DROP TABLE
DROP VIEW

Datenkontrolle:

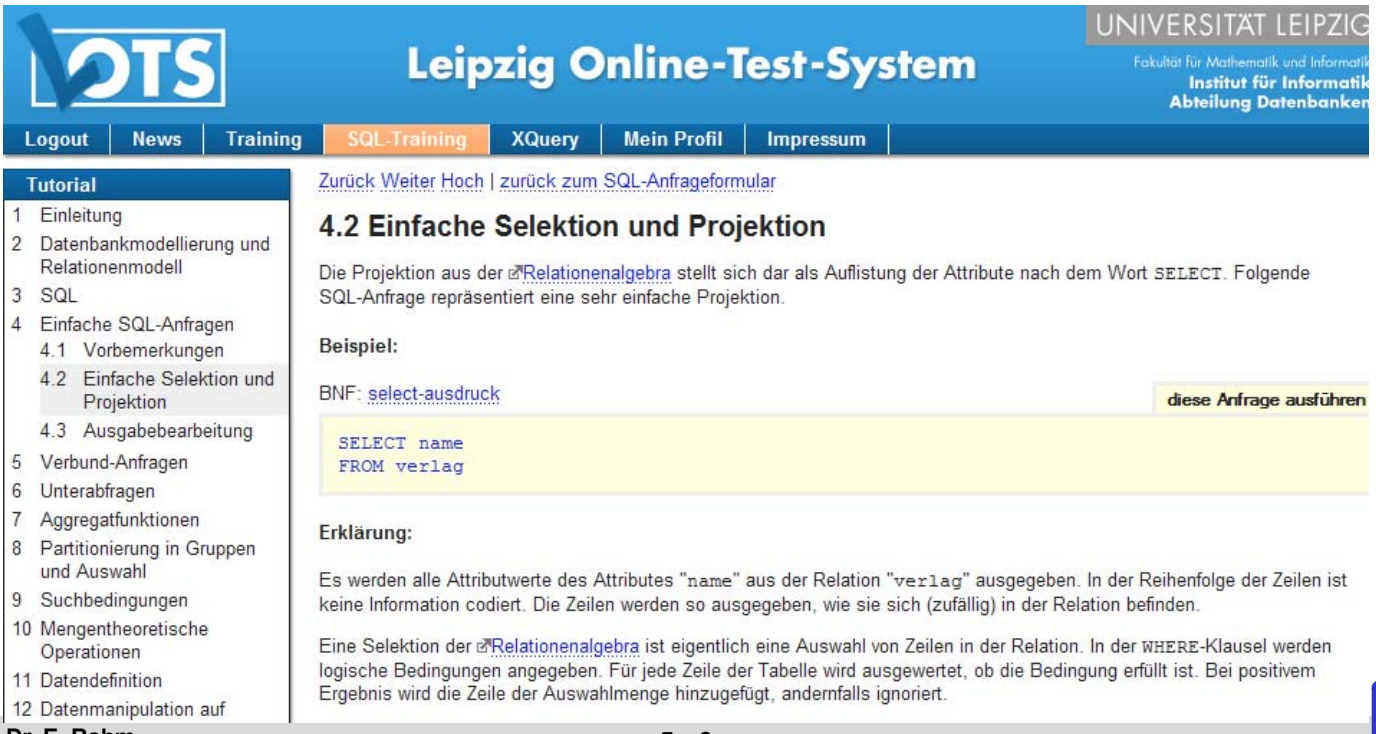
Constraints-Definitionen bei CREATE TABLE
CREATE ASSERTION
DROP ASSERTION
GRANT
REVOKE
COMMIT
ROLLBACK

Eingebettetes SQL:

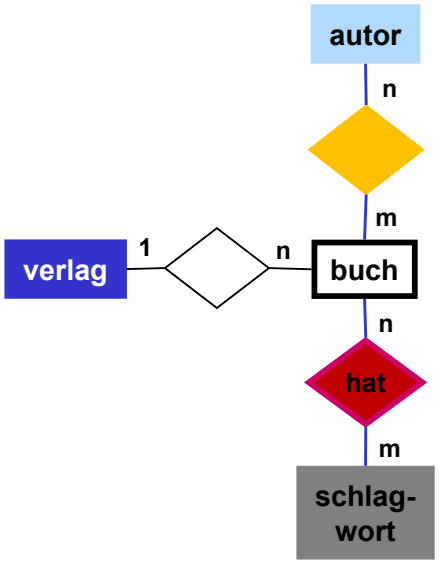
DECLARE CURSOR
FETCH
OPEN CURSOR
CLOSE CURSOR
SET CONSTRAINTS
SET TRANSACTION
CREATE TEMPORARY TABLE

SQL-Training in LOTS (*https://lots.uni-leipzig.de*)

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



Test-Datenbank



Verlag | verlagsid | name | ort

Autor | autorid | nachname | initialen | vornamen | zusatz

Buch_Aut | buchid | autorid | rolle | rang

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

Buch_SW | buchid | swid

Schlagwort | swid | schlagwort

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

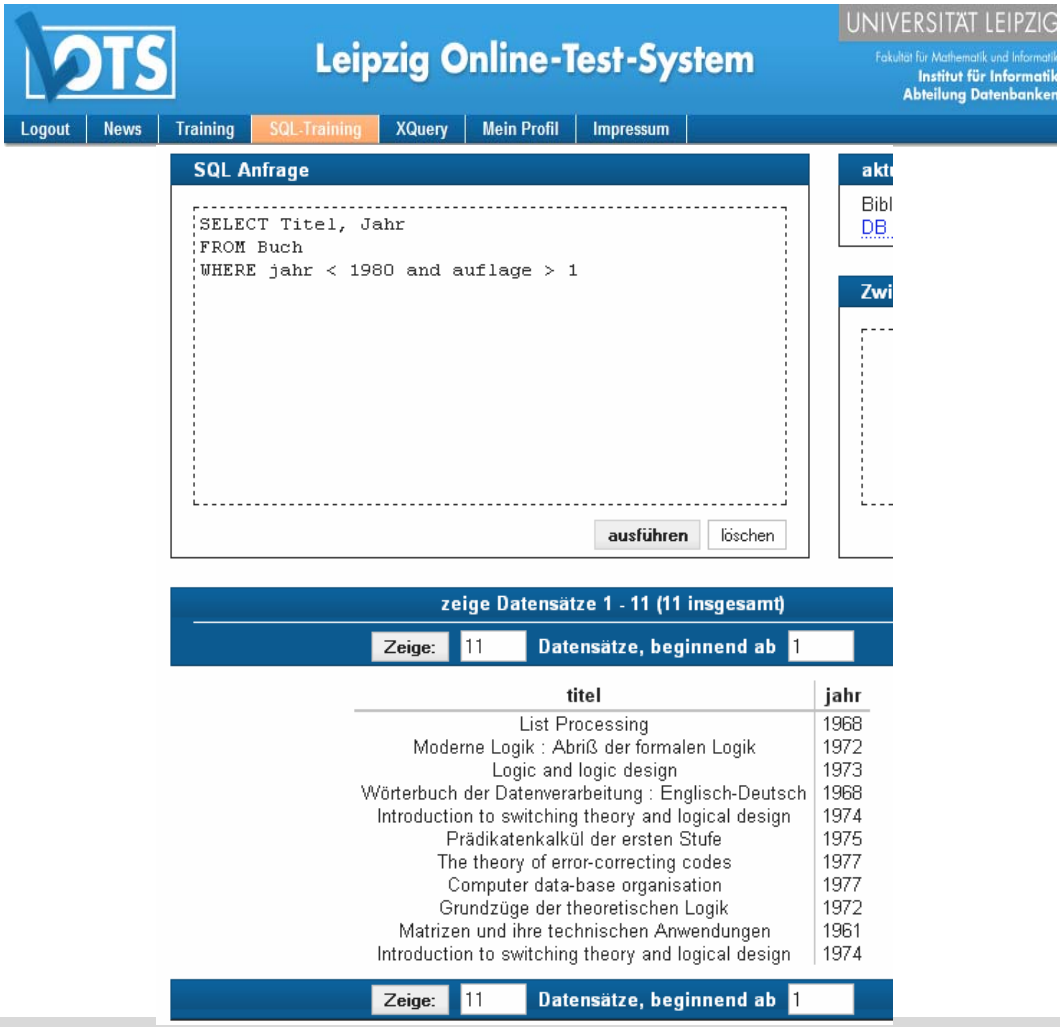
Einfache Selektionen und Projektionen

Q1: Welche (Berliner) Verlage gibt es?

```
SELECT
FROM
WHERE
```

Q2: Welche Bücher erschienen vor 1980 in einer Neuauflage?

```
SELECT
 FROM
 WHERE
```

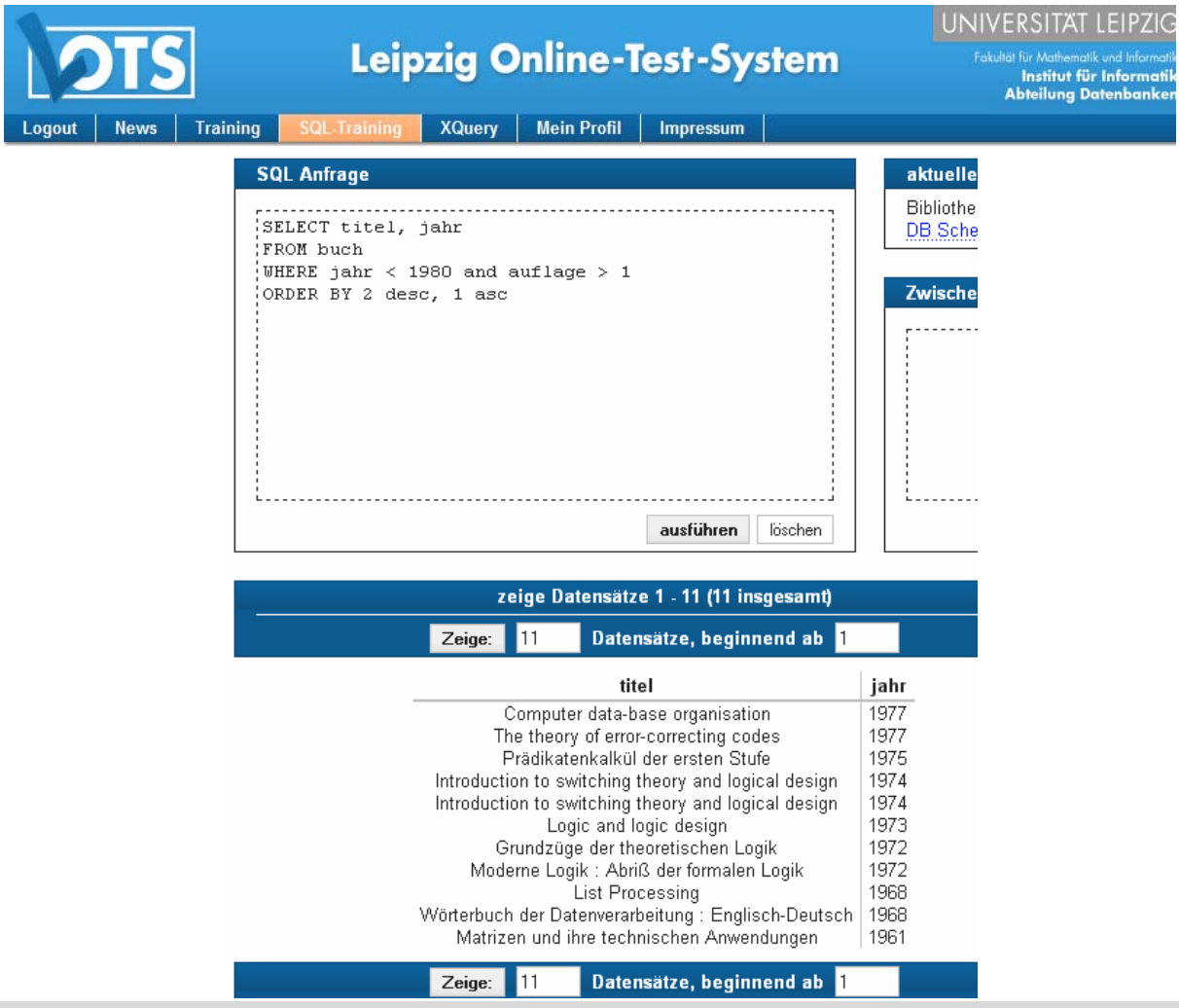



Sortierte Ausgabe

- ORDER BY-Klausel zur Sortierung von Ergebnissätzen
  - aufsteigende (ascending) oder absteigende (descending) Sortierung
  - Voreinstellung: ASCending
  - statt Attributnamen sind in ORDER BY-Klausel auch relative Positionen der Attribute aus Select-Klausel möglich

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

```
SELECT      titel, jahr
FROM        Buch
WHERE           …
ORDER BY
```

Duplikateliminierung

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

Q4: Welche Verlagsorte gibt es?

```
SELECT
FROM    Verlag
```

Umbenennungen

- Benennung von Ergebnis-Spalten

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

Buchtitel	Preis_in_Euro

  - 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

```
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 Rechenanalagen	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 ('2022-04-24' AS DATE) ...

Join-Anfragen

Q5: Welche Buchtitel wurden von Berliner Verlagen herausgebracht?

```
SELECT B.titel
FROM   Buch B, Verlag V
WHERE  V.ort='Berlin' AND
```

- 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 B.titel
FROM   Buch B, Autor A, Buch_Aut BA
WHERE  A.nachname='Rahm' AND
```


Join-Anfragen (2)

- hierarchische Beziehung auf einer Relation (PERS)

Beispielschema:

```
PERS (PNO int, Name, Salary, ..., MNO int, DNO int,
      PRIMARY KEY (PNO), FOREIGN KEY (MNO) REFERENCES PERS)
```

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

```
SELECT
FROM    PERS
WHERE
```

- Verwendung von Korrelationsnamen obligatorisch!

PNO	Name	Salary	MNO
34	Mey	32000	37
35	Schultz	42500	37
37	Abel	41000	-

PNO	Name	Salary	MNO
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?

```
SELECT titel
FROM   Buch NATURAL JOIN (Buch_Aut NATURAL JOIN Autor)
WHERE  nachname= 'Rahm'
```


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

```
SELECT schlagwort, buchid
FROM Schlagwort LEFT OUTER JOIN Buch_SW USING (swid)
```

- kartesisches Produkt:

```
SELECT * FROM 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 > 1990
```

- 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 =2001
```

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 (Salary)
FROM   PERS
```

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

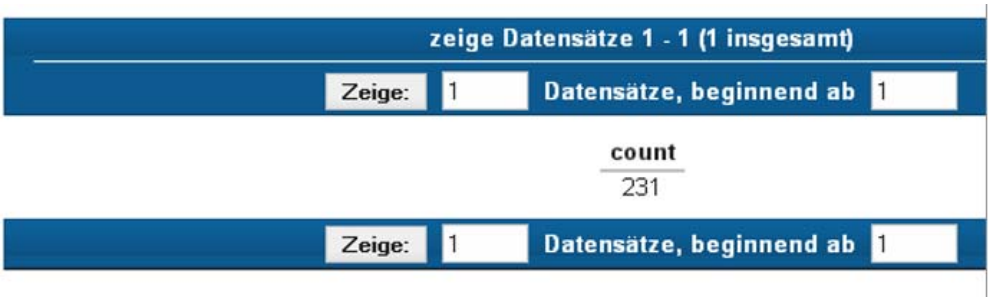
Aggregatfunktionen (2)

Q9: Wie viele Verlage gibt es?

```
SELECT
FROM    Verlag
```

Q10: An wie vielen Orten gibt es Verlage?

```
SELECT
FROM    Verlag
```



Aggregatfunktionen (3)

Q11: Für wie viele Bücher ist der Verlag bekannt?

```
SELECT
FROM    Buch
```

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

```
SELECT
FROM    Buch
```

Q13: Zu wie vielen Verlagen gibt es Bücher ?

```
SELECT
FROM    Buch
```


Aggregatfunktionen (4)

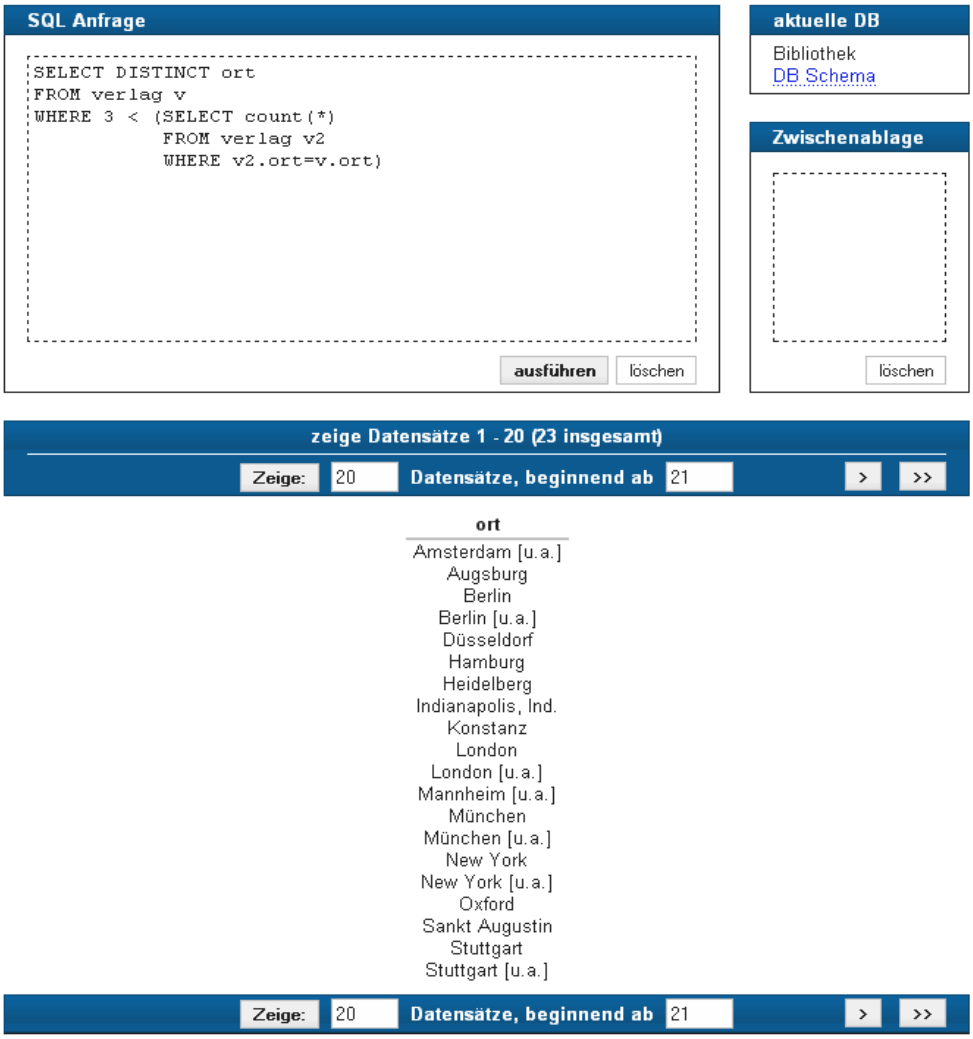
- Aggregatfunktionen können nicht direkt in WHERE-Klausel verwendet werden
  - Einsatz von Sub-Queries

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

```
SELECT  titel, jahr
FROM    Buch
WHERE   jahr = (SELECT MIN(jahr)
                FROM Buch)
```

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

```
SELECT   DISTINCT V.ort
FROM     Verlag V
WHERE    3 < (SELECT COUNT(*)
              FROM Verlag V2
              WHERE V2.ort=V.ort)
```



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
GROUP BY
```

Ausgabe von Verlagsname?

```
SELECT name
FROM   Buch  NATURAL JOIN Verlag
GROUP BY
```

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 DNO, AVG(Salary) AS Avg_Salary, Max(Salary) AS Max_Salary
FROM   PERS
GROUP BY  DNO
```

PERS

PNO	Name	Age	Salary	DNO
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

DNO	Avg_Salary	Max_Salary
K51	38000	43500
K53	35000	42000
K55	38333	52000

Auswahl von Gruppen (HAVING-Klausel)

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

- HAVING: Bedingungen nur bezüglich Sätzen einer Gruppe
  - 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 DNO
FROM   PERS NATURAL JOIN DEPT
WHERE
GROUP BY
HAVING
```


HAVING-Klausel (2)

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

```
SELECT DNO, SUM(Salary)
FROM   PERS
GROUP BY DNO
HAVING COUNT(*) > 5
```

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

„Profi-Version" ohne korrelierte Sub-Query

```
SELECT ort
FROM   Verlag
GROUP BY ort
HAVING COUNT(*) > 3
```

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) für Row Sub-Query
  - darf hier höchstens 1 Tupel als Ergebnis liefern (Kardinalität 1)
- Vergleiche zwischen Tupel-Konstruktoren (row constructor) mit mehreren Attributen
  - (a1, a2, ... an) = (b1, b2, ...bn) ⇔ a 1 = b1 AND a2 = b2 ... AND an = bn
  - (a1, a2, ... an) < (b1, b2, ...bn) ⇔ (a1 < b1) OR ((a1 = b1) AND (a2 < b2)) OR (...)

```
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 '_' selbst durch Voranstellen eines Escape-Zeichens möglich.

LIKE-Klausel	*Wird z.B. erfüllt von*
NAME **LIKE** '%SCHMI%'	'H. SCHMIDT' 'SCHMIED'
ANR **LIKE** '_7%'	'17' 'K75'
NAME **NOT LIKE** '%-%'	
STRING **LIKE** '%_%' **ESCAPE** '\'	'DBS_1' 'X_Y_Z'

BETWEEN-Prädikate

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

- Semantik

  y BETWEEN x AND z ⇔ x ≤ y AND y ≤ z
  y NOT BETWEEN x AND z ⇔ NOT (y BETWEEN x AND z)

- Beispiel

```
SELECT      DNO, count(*)
FROM        PERS
WHERE       DNO NOT BETWEEN 'K50' AND 'K54'
GROUP BY    DNO
HAVING      AVG (Age) 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 IN (a_1, a_j, a_k)
  - *implizite Mengendefinition*: A_i IN (SELECT . . .)
- Semantik

  x IN (a, b, . . ., z) ⇔ x = a OR x = b . . . OR x = z

  x NOT IN erg ⇔ NOT (x 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    swid NOT IN (SELECT swid FROM Buch_SW)
```

NULL-Werte

- pro Attribut kann Zulässigkeit von Nullwerten festgelegt werden
  - unterschiedliche Bedeutungen: Datenwert ist momentan nicht bekannt
  - Attributwert existiert nicht für ein Tupel, z.B. akademischerTitel
- Behandlung von Nullwerten
  - das Ergebnis einer arithmetischen Operation (+, -, *, /) mit einem NULL-Wert ist der 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 (Age <= 50) vereinigt mit PERS (Age > 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  (Salary)  FROM  PERS    →    Ergebnis  X
SELECT  SUM  (Salary)  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 PNO, PNAME
FROM   PERS
WHERE  SALARY IS NULL
```

Quantifizierte Prädikate

- All-or-Any-Prädikat

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

θ ALL: Prädikat wird zu "true" ausgewertet, wenn der θ -Vergleich für alle Ergebniswerte von table-exp "true" ist.

θ ANY/ θ SOME: analog, wenn der θ -Vergleich für einen Ergebniswert "true" ist.

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

Tabelle PERS (PNO, …. Salary, MNO)

```
SELECT M.PNO
FROM   PERS  M
WHERE  M.Salary > ALL (SELECT P.Salary
                       FROM PERS P
                       WHERE P.MNO=M.PNO)
```

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 kann 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))

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

```
SELECT M.PNO
FROM   PERS M
WHERE  NOT EXISTS(SELECT *
                  FROM PERS P
                  WHERE P.MNO=M.PNO AND
                        NOT (M.Salary > P.Salary) )
```

Existenztests (2)

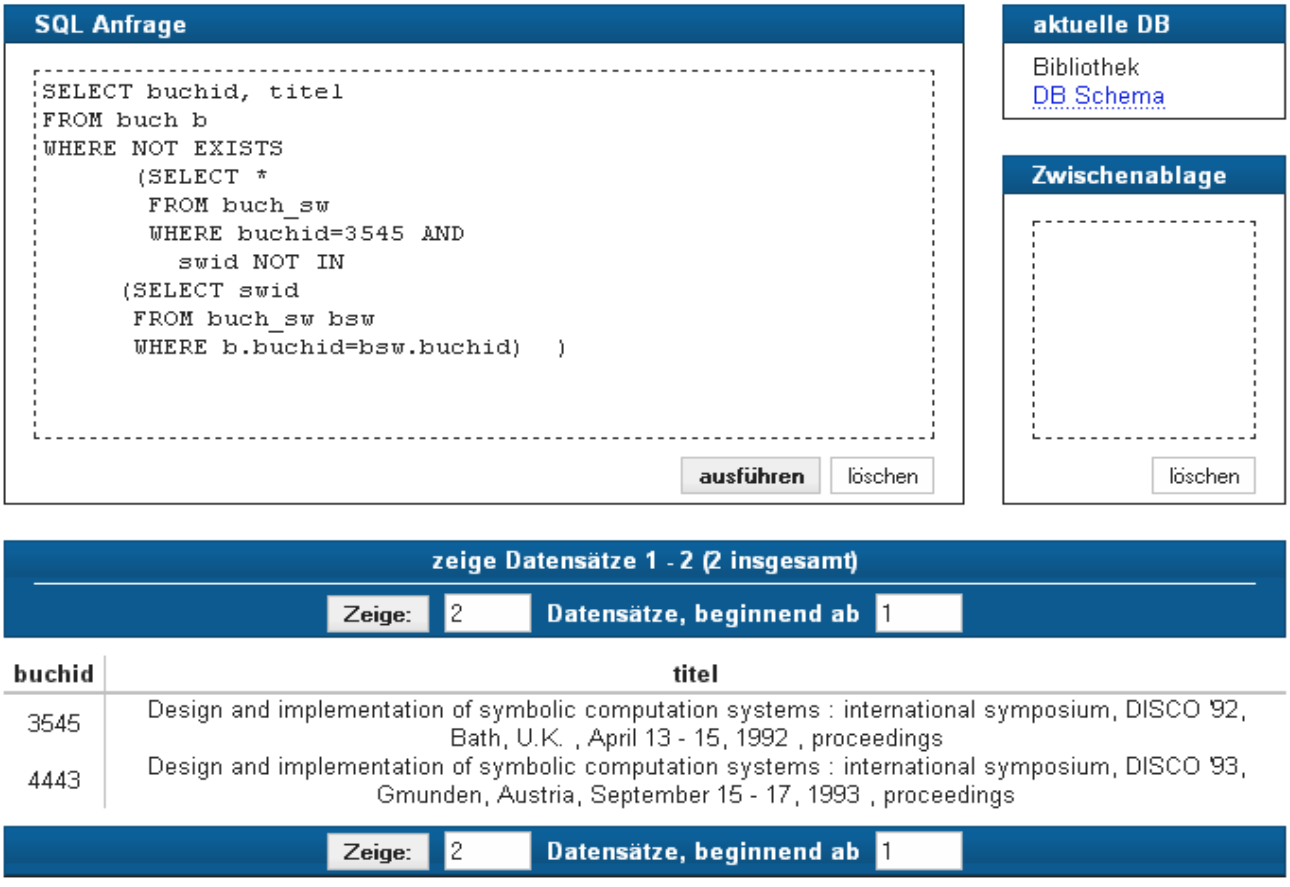
Q24: Finde die Schlagworte, die für mindestens ein (... kein) Buch vergeben wurden

```
SELECT S.*
FROM   schlagwort  S
WHERE      EXISTS  (SELECT *
                    FROM Buch_SW B WHERE B.swid=S.swid)
```

Q25: Finde die Bücher, die alle Schlagworte des Buchs mit ID 3545 abdecken
(*andere Formulierung*: zu denen kein Schlagwort des Buches 3545 existiert, das nicht auch für die gesuchten Bücher existiert).

$$Buch\_SW \div \pi_{swid}(\sigma_{buchid=3545}(Buch\_SW))$$

```
SELECT buchid
FROM   Buch_SW B
WHERE  NOT EXISTS (SELECT  * FROM Buch_SW
                   WHERE buchid=3545 AND
                         swid NOT IN
                            (SELECT swid FROM Buch_SW B2
                             WHERE B2.buchid=B.buchid))
```

Division: Beispiel

Q26: Welche Lieferanten/Supplier (SNO) haben alle Teile geliefert?

$\pi_{SNO, PNO}$ (DELIVERY) ÷ π_{PNO} (PART)

```
SELECT  DISTINCT D.SNO
FROM    DELIVERY D
WHERE   NOT EXISTS  (SELECT *
                     FROM PART
                     WHERE PNO NOT IN
                        (SELECT D2.PNO
                         FROM DELIVERY D2
                         WHERE D2.SNO=D.SNO))
```


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 Ai 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
WHERE isbn IS NULL

EXCEPT

SELECT *
FROM buch
WHERE auflage IS NULL
```

aktuelle DB

Bibliothek

DB Schema

Zwischenablage

zeige Datensätze 1 - 19 (19 insgesamt)

Zeige: 19 Datensätze, beginnend ab 1

buchid	titel	isbn	auflage	jahr	preis	waehrung	signatur	verlagsid
1484	Computer data-base organisation		2. ed. Aufl.	1977			R 3592	30
1315	Neural computers : proceedings of the NATO Advanced Research Workshop on Neural Computers, Neuss, September 20 - october 2, 1987		2. Aufl.	1989			K 4474	6
3888	OSF DCE for AIX, OS/2 and DOS Windows overview		1. ed.	1993			P 5978	299
86	List Processing		2.	1968			R 2055	54
1138	Wörterbuch der Datenverarbeitung : Englisch-Deutsch		1. Aufl.	1968			W 1996	246
2567	Die Programmiersprache Pascal : [ISO-Standard 7185]		15., überarb. Aufl.	1992			VL	56

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 Autor Garfield mit Autorid 4711 ein*

```
INSERT INTO Autor (autorid, vorname)
       VALUES (4711, 'Garfield')
```

  - 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 Verlage aus Leipzig in Relation TEMP ein*

```
INSERT INTO TEMP
SELECT *
FROM Verlag
WHERE ort='Leipzig'
```

  - (leere) Relation TEMP mit kompatiblen Attributen sei vorhanden
  - spezifizierte Tupelmenge wird ausgewählt und in 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 Mitarbeitern der Abteilung „K56“eine Gehaltserhöhung von 2%

```
UPDATE PERS  SET Salary=Salary * 1.02
WHERE DNO='K56'
```

Stelle die DM-Preise für die nach 1997 erschienenen Bücher auf Euro um (Verhältnis 1:1,95)

```
UPDATE Buch
SET preis=preis/1.95, SET waehrung='Euro'
WHERE jahr > 1997 AND waehrung='DM' AND
      preis IS NOT NULL
```

Löschen von Tupeln (DELETE)

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

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

Lösche den Autor mit der Autorid 4711

```
DELETE FROM Autor
WHERE autorid=4711
```

Lösche alle Schlagworte, die nicht verwendet werden.

```
DELETE FROM Schlagwort
WHERE swid NOT IN (SELECT swid FROM Buch_SW)
```


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-exp]
            [ DELETE WHERE cond-exp ]
WHEN NOT MATCHED THEN
      INSERT (column-commalist) VALUES row-constr-commalist
             [WHERE cond-exp]
```

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
$\pi_{A1, A2, \ldots Ak}$ (R)	SELECT **DISTINCT** A1, A2, ... Ak FROM R
σ_P (R)	SELECT * FROM R WHERE P
R × S	SELECT * FROM R,S bzw. FROM R CROSS JOIN S
$R \underset{R.A\ \theta\ S.B}{\bowtie} S$	SELECT * FROM R,S WHERE R.A θ S.B bzw. SELECT * FROM R JOIN S ON (R.A θ S.B)
$R \bowtie S$	SELECT * FROM R NATURAL JOIN S
$R \overset{\bowtie}{\supset} S$	SELECT * FROM R LEFT JOIN S
$R \boxtimes S$	SELECT * FROM R FULL JOIN S
$R \ltimes S$	SELECT R.* FROM R NATURAL JOIN S
R ∪ S	SELECT * FROM R UNION SELECT * FROM S
R ∩ S	SELECT * FROM R INTERSECT SELECT * FROM S
R - S	SELECT * FROM R EXCEPT SELECT * FROM S
R ÷ S	SELECT … FROM R WHERE **NOT EXISTS** (SELECT *…)

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)