4. SQL-Einführung

- Grundlagen: Anfragekonzept, Befehlsübersicht
- SELECT: mengenorientierte Anfragen deskriptiver Art
  - einfache Selektions- und Projektionsanfragen
  - Join-Anfragen
  - geschachtelte Anfragen (Sub-Queries)
  - Aggregatfunktionen
  - Gruppenanfragen (GROUP BY, HAVING)
  - LIKE-, BETWEEN-, IN-Prädikate
  - Nullwertabfragen
  - quantifizierte Prädikate (ALL/ANY, EXISTS)
  - mengentheoretische Operationen: UNION, INTERSECT, EXCEPT
- Änderungsoperationen INSERT, DELETE, UPDATE
- 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 u.a. mit objektorientierten Erweiterungen (-> objekt-relationale DBS) und XML-Unterstützung

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

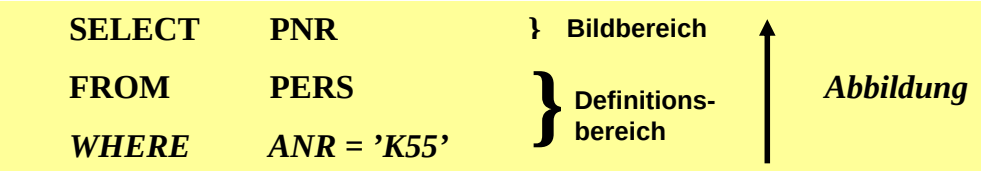


Abbildung: Eingaberelationen (FROM) werden unter Auswertung von Bedingungen (WHERE) in Attribute einer Ergebnistabelle (SELECT) abgebildet

- 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

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

Anfragemöglichkeiten in SQL

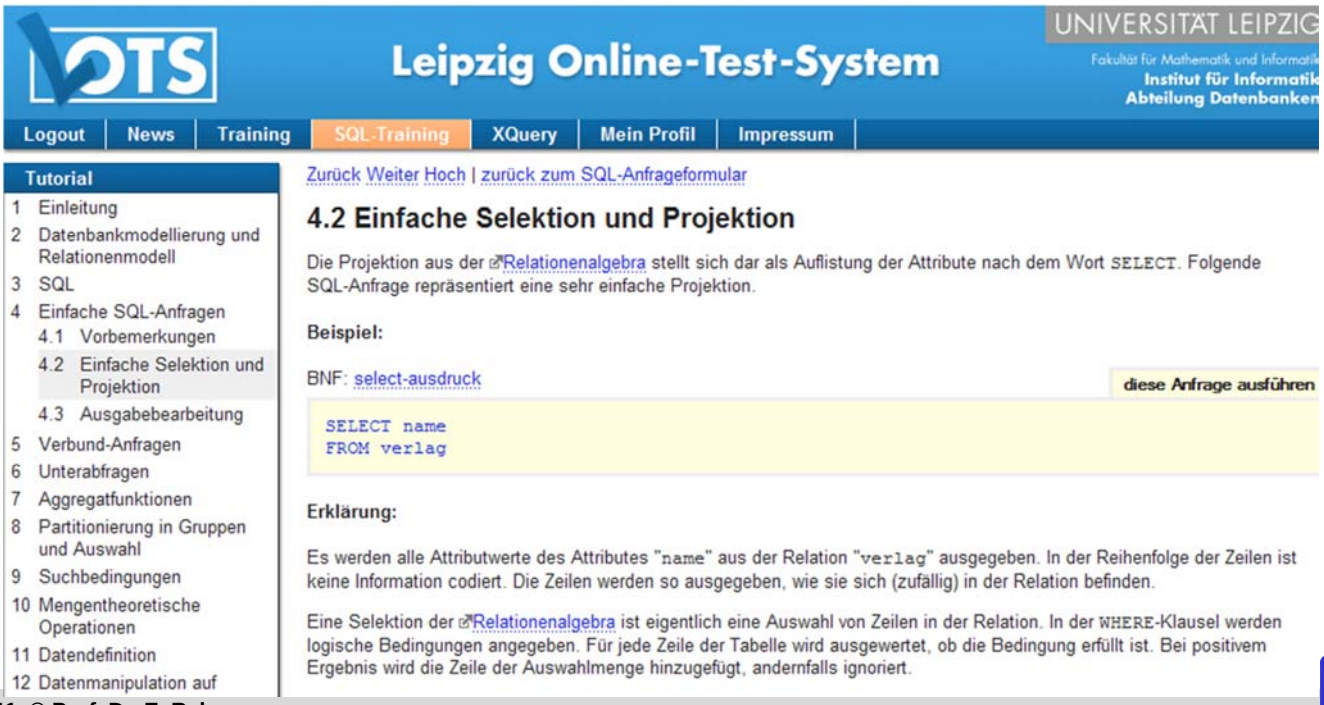
```
select-expression ::=
      SELECT [ALL | DISTINCT] select-item-list
      FROM table-ref-commalist
      [WHERE cond-exp]
      [GROUP BY column-ref-commalist]
      [HAVING cond-exp]
      [ORDER BY order-item-commalist ]

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

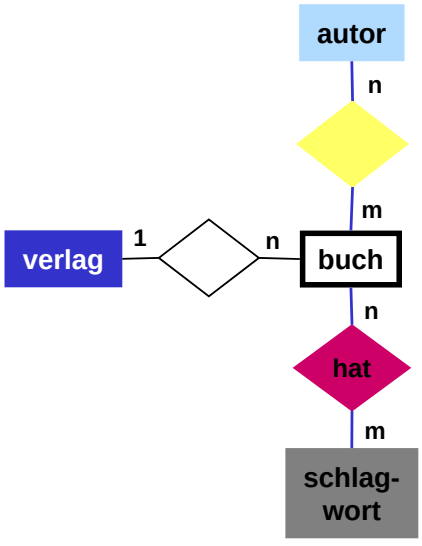
- *SELECT* –Klausel spezifiziert auszugebende Attribute
  - mit SELECT * werden alle Attribute der spezifizierten Relation(en) ausgegeben
- FROM-Klausel spezifiziert zu verarbeitende Objekte (Relationen, Sichten)
- *WHERE*-Klausel kann eine Sammlung von Suchprädikaten enthalten, die mit *NOT*, *AND* und *OR* verknüpft sein können
  - dabei sind Vergleichsprädikate der Form $A_i\ \theta\ a_i$ bzw. $A_i\ \theta\ A_j$ möglich
    mit $\theta \in \{ =, <>, <, \leq, >, \geq \}$

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



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

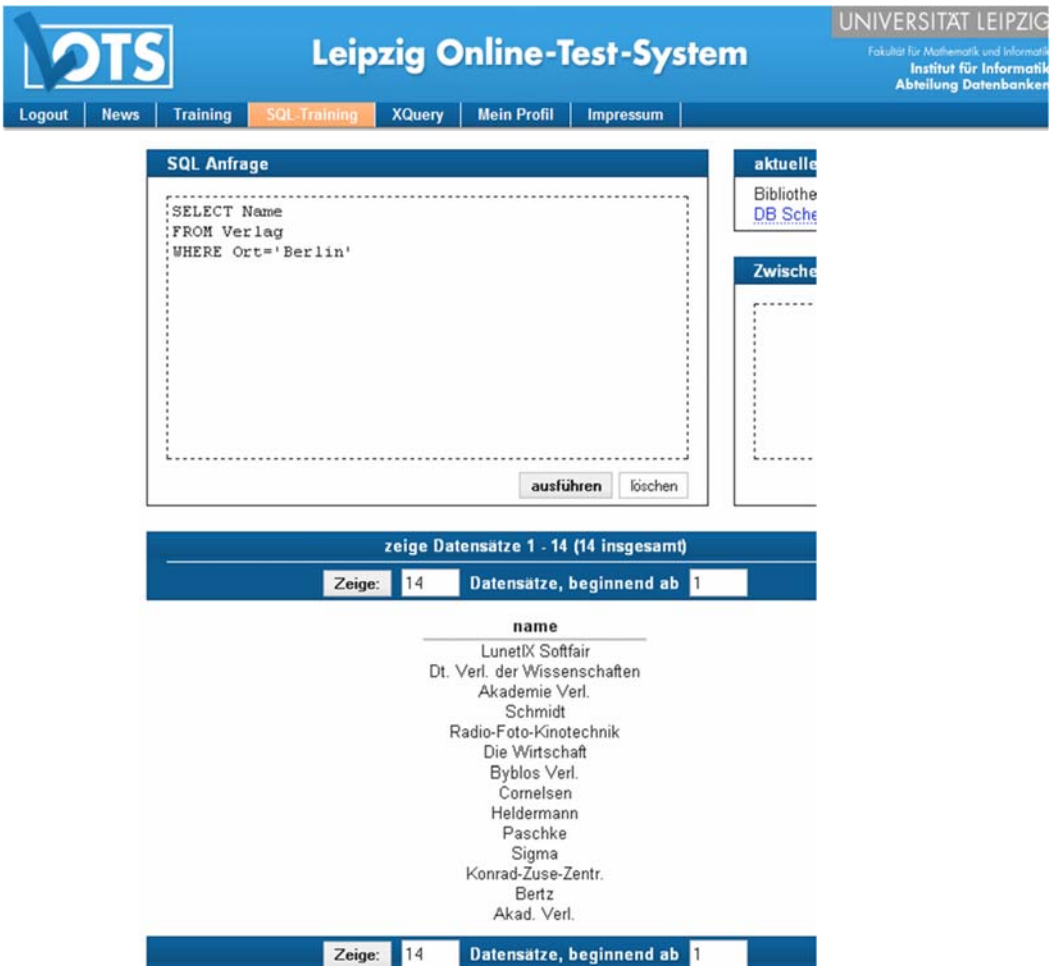
buch_aut.rolle kann sein: Herausgebers (H), Verfasser (V), Übersetzer (U), Mitarbeiter (M)
buch_aut.rang: Position des Autors in der Autorenliste (z.B. 1 für Erstautor)
autor.zusatz: Namenszusatz wie „von“ oder „van“
buch.signatur entspricht der Signatur in der IfI-Bibliothek (Stand 1998)

- 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

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)

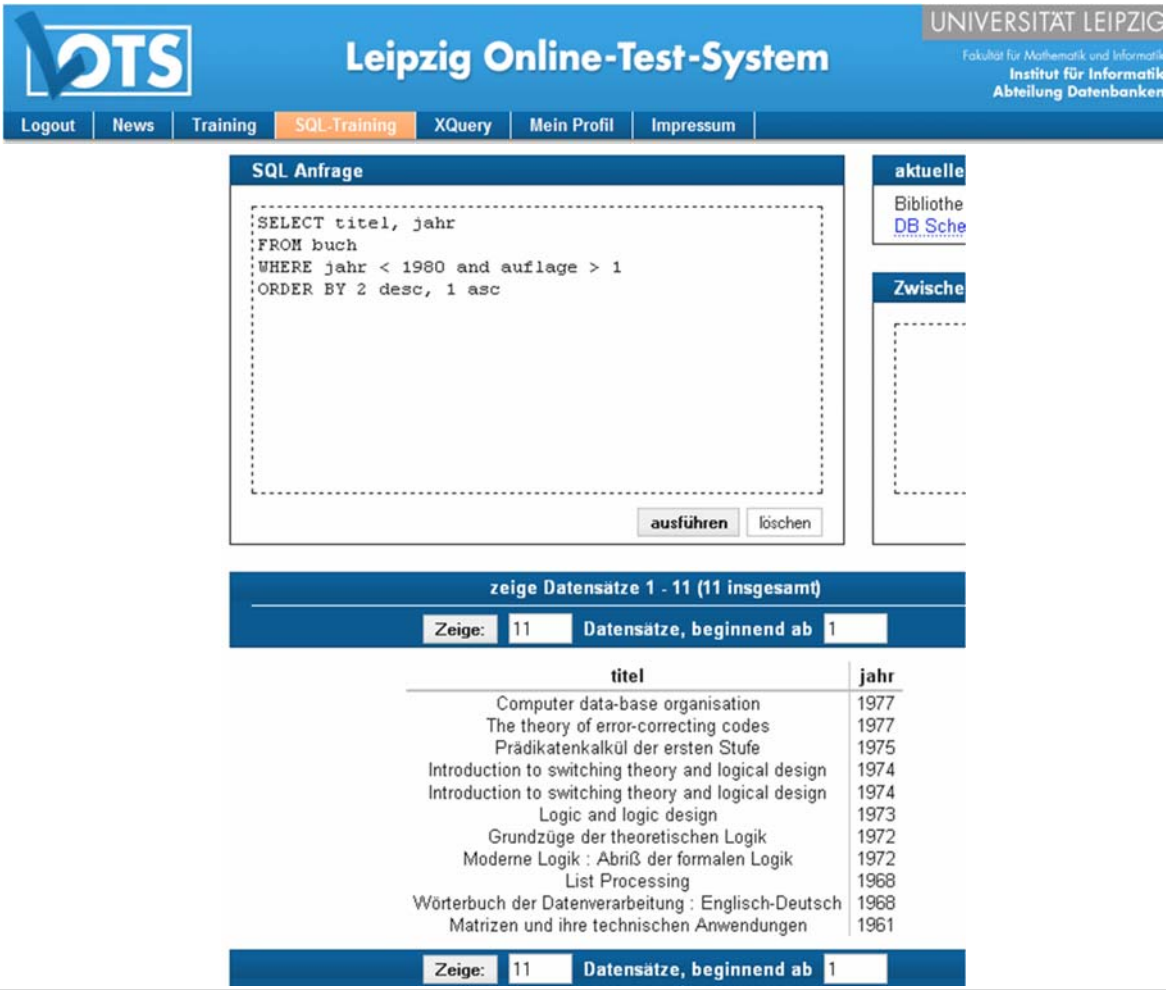
Q3: wie Q2, jedoch sortiert nach Jahr (absteigend), Titel (aufsteigend)

```
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 Ausgabe 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
```


Skalare Funktionen (Auswahl)

- CASE

```
SELECT titel, jahr, CASE
        WHEN jahr >2005 THEN 'Aktuell'
        WHEN jahr > 1995 THEN 'Mittel'
        ELSE 'Veraltet' END AS  Aktualitaet
 FROM buch
```

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

- 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)
- 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 ('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 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!

PNR	GEHALT	MNR
34	32000	37
35	42500	37
37	41000	-

PNR	GEHALT	MNR
34	32000	37
35	42500	37
37	41000	-

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
```

```
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

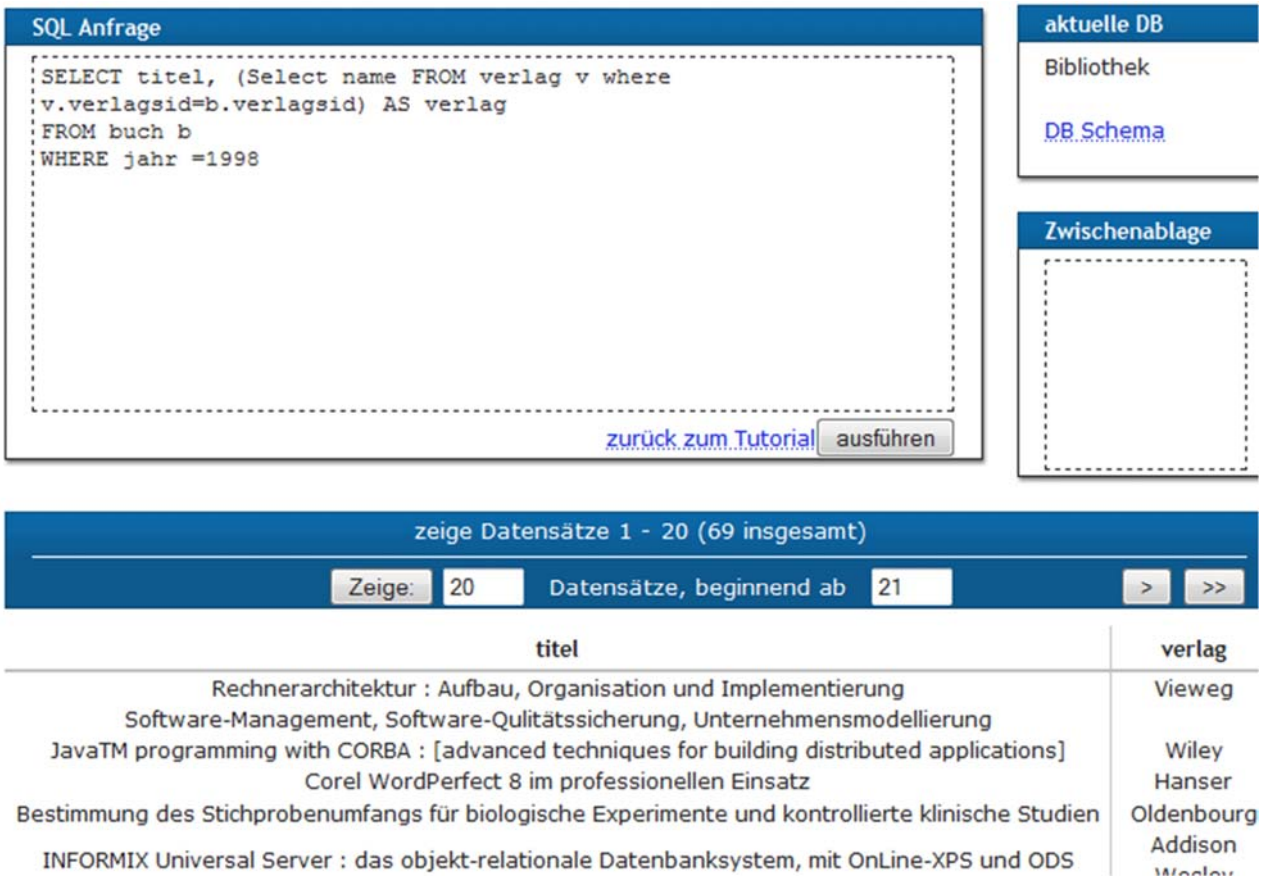
Weitergehende Verwendung von Sub-Queries

- 3 Arten von Sub-Queries
  - Table Sub-Queries (mengenwertige Ergebnisse)
  - Row Sub-Queries (Tupel-Ergebnis)
  - Skalare Sub-Queries (atomarer Wert; Kardinalität 1, Grad 1)
- Im SQL-Standard können Table-Sub-Queries überall stehen, wo ein Relationenname möglich ist, insbesondere in der FROM-Klausel.

```
SELECT count(*)
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
```



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
```


Aggregatfunktionen (3)

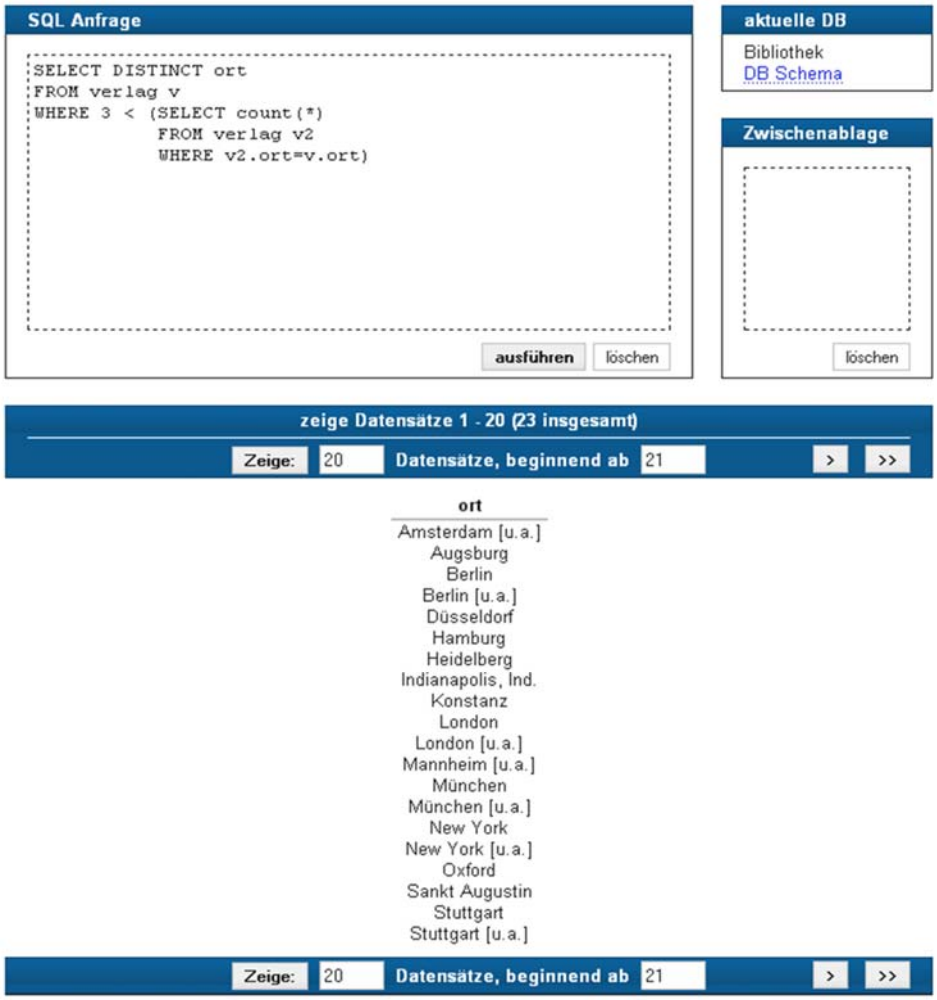
- Aggregatfunktionen können nicht direkt in WHERE-Klausel verwendet werden
  - Verwendung von Sub-Queries

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

```
SELECT
FROM
```

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

```
SELECT
FROM
WHERE
```



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 Abteilungen und das Durchschnitts- sowie Spitzengehalt ihrer Angestellten auf.

```
SELECT
FROM
```

Q17: Liste alle Abteilungen (ANR und ANAME) sowie die Anzahl der beschäftigten Angestellten auf

```
SELECT
FROM
```

Group-By: Beispiel

PERS

PNR	Name	Alter	Gehalt	ANR
234	Meier	23	42000	K53
235	Schmid	31	32500	K51
237	Bauer	21	21000	K53
321	Klein	19	27000	K55
406	Abel	47	52000	K55
123	Schulz	32	43500	K51
829	Müller	36	42000	K53
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 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
  - (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 „_“ 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** ’\’	

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      ANR
FROM        PERS
WHERE       ANR NOT BETWEEN `K50´ AND `K54´
GROUP BY    ANR
HAVING      AVG (Alter) BETWEEN 20 AND 35
```


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
```

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 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

```
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 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.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

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)

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)

```
searched-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

Relationenalgebra vs. SQL (Retrieval)

Relationenalgebra	SQL
$\pi_{A1, A2, \ldots Ak}$ (R)	SELECT
σ_P (R)	SELECT
R × S	SELECT
$R \underset{R.A\ \theta\ S.B}{\bowtie} S$	SELECT
$R \bowtie S$	
$R \overrightarrow{\bowtie} S$	
$R \boxtimes S$	
$R \cup S$	
$R \cap S$	
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
- Datenänderungen: INSERT, UPDATE, DELETE
- hohe Sprachredundanz
- SQL-Implementierungen weichen teilweise erheblich von Standard ab (Beschränkungen / Erweiterungen)