Einführung in Datenbanken

Übung 10: Aggregationen: GROUP BY, Relationale Algebra

Prof. Dr. Stefan Brass
PD Dr. Alexander Hinneburg
Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2020/21

http://www.informatik.uni-halle.de/~brass/db20/

Inhalt

Hausaufgabe 8a (1)

- In welchen Abteilungen arbeiten weder Analysten (Job „ANALYST“), noch Verkäufer (Job „SALESMAN“)?

  Geben Sie `deptno` (Abteilungsnummer), `dname` (Abteilungsname) und `loc` (Ort der Abteilung) aus. Sortieren Sie die Ausgabe nach der Abteilungsnummer.

deptno	dname	loc
10	ACCOUNTING	NEW YORK
40	OPERATIONS	BOSTON

  2 Datensätze

  Die Abteilung 40 ist tatsächlich leer, erfüllt aber die Bedingung der Anfrage.

- dept(deptno, dname, loc)
- emp(empno, ename, job, mgr°→emp, hiredate, sal, comm°, deptno°→dept)

Hausaufgabe 8a (2)

- Lösung mit `NOT EXISTS`:

```
SELECT deptno, dname, loc
FROM   dept d
WHERE  NOT EXISTS (SELECT *
                   FROM   emp e
                   WHERE  (e.job = 'ANALYST'
                           OR e.job = 'SALESMAN')
                   AND    e.deptno = d.deptno)
ORDER  BY deptno
```

Man könnte auch „`SELECT *`“ schreiben. Für Anfragen in Programmen ist das aber eher nicht empfohlen. Zumindest sollte man darüber nachdenken, was passieren würde, wenn die Tabelle später um zusätzliche Spalten erweitert wird. Für eine ad-hoc Anfrage wäre es natürlich ok.

Hausaufgabe 8a (3)

- Lösung mit `NOT IN`:

```
SELECT deptno, dname, loc
FROM   dept
WHERE  deptno NOT IN (SELECT deptno
                      FROM   emp
                      WHERE  job IN ('ANALYST',
                                     'SALESMAN')
                      AND    deptno IS NOT NULL)
ORDER  BY deptno
```

Bei `NOT IN` ist es wichtig, dass die Unteranfrage keinen Nullwert liefern kann, sonst wäre das Ergebnis leer. Im Beispielzustand ist der einzige Angestellte, der bisher keiner Abteilung zugeordnet ist, ein `CLERK`. Man würde es beim Test also nicht bemerken.

Hausaufgabe 8b (1)

- Welche Jobs gibt es in allen Abteilungen, die nicht leer sind, also mindestens einen Angestellten haben?

  Gesucht ist also die Schnittmenge der Berufe über allen nicht-leeren Abteilungen. Z.B. hat jede Abteilung einen Manager: `CLARK` in Abteilung 10, `JONES` in Abteilung 20 und `BLAKE` in Abteilung 30.

- Die erwartete Antwort ist:

job
CLERK
MANAGER

  2 Datensätze

- Üblicherweise überführt man eine Allaussage in SQL in $\neg\exists\neg$, hier also: „es gibt keine nicht-leere Abteilung, die nicht mindestens einen Angestellten mit dem Job hat“.

Hausaufgabe 8b (2)

- Lösung:

```
SELECT DISTINCT job
FROM   emp j
WHERE  NOT EXISTS
       (SELECT * FROM dept d
        WHERE  EXISTS
               (SELECT * FROM emp x
                WHERE  d.deptno = x.deptno)
        AND    NOT EXISTS
               (SELECT * FROM emp e
                WHERE  e.job = j.job
                AND    e.deptno = d.deptno))
```

Hausaufgabe 8b (3)

- Alternative (kürzer, Abteilung wird durch `emp d` vertreten, damit ist die Abteilung automatisch nicht leer):

```
SELECT DISTINCT job
FROM   emp j
WHERE  NOT EXISTS
       (SELECT * FROM emp d
        WHERE  NOT EXISTS
               (SELECT * FROM emp e
                WHERE  e.job = j.job
                AND    e.deptno = d.deptno))
```

Ich würde aber diese Lösung nicht als absolut besser als die vorangegangene sehen. Sie ist natürlich einfacher. Wenn es aber sehr viele Angestellte und wenig Abteilungen gibt, könnte relevant sein, dass bei der vorigen Lösung die letzte Unteranfrage nur pro Abteilung ausgeführt wird. Leistung ist aber in dieser Vorlesung eigentlich kein Thema.

Hausaufgabe 8b (4)

- Auch möglich:

```
WITH   JOBS AS
       (SELECT DISTINCT job
        FROM   emp)

SELECT job
FROM   jobs j
WHERE  -- Anzahl Abteilungen mit Job j:
       (SELECT COUNT(DISTINCT deptno)
        FROM   emp
        WHERE  emp.job = j.job)
       =
       -- Anzahl aller Abteilungen (nicht-leer):
       (SELECT COUNT(DISTINCT deptno)
        FROM   emp)
```

Hausaufgabe 8c (1)

- Betrachtet seien alle Angestellten mit Untergebenen (in der Spalte `mgr` ist die `empno` des direkten Vorgesetzten eingetragen). Berechnen Sie für diese Angestellten:
  - Anzahl dieser Angestellten (`anz`),
  - das minimale, das maximale und das durchschnittliche Gehalt aus der Spalte `sal` (`min_sal`, `max_sal`, `avg_sal`),
  - die Anzahl verschiedener Jobs (`anz_jobs`).

    Z.B. soll auch dann, wenn zwei Angestelle den Job `MANAGER` haben, der Beruf `MANAGER` nur ein Mal gezählt werden.

  Nennen Sie die Ausgabespalten, wie in Klammern gezeigt. Runden Sie das Durchschnittsgehalt durch Aufruf der Funktion `ROUND(...)`. Die Provision in der Spalte `comm` brauchen Sie nicht zu berücksichtigen. Ihre Anfrage muss genau eine Ergebniszeile mit allen diesen Daten liefern (wie immer bei Aggregationsanfragen ohne `GROUP BY`).

Hausaufgabe 8c (2)

- Die Angestellten mit Untergebenen sind im Beispielzustand:

empno	ename	job	mgr	sal	comm	deptno
7566	JONES	MANAGER	7839	2975	NULL	20
7698	BLAKE	MANAGER	7839	2850	NULL	30
7782	CLARK	MANAGER	7839	2450	NULL	10
7788	SCOTT	ANALYST	7566	3000	NULL	20
7839	KING	PRESIDENT	NULL	5000	NULL	10
7902	FORD	ANALYST	7566	3000	NULL	20

- Die erwartete Antwort ist:

anz	min_sal	max_sal	avg_sal	anz_jobs
6	2450	5000	3213	3

1 Datensatz

Hausaufgabe 8c (3)

- Wenn man die Forderung nach Untergebenen durch einen normalen Verbund implementiert, würden Angestelle mit mehreren Untergebenen mehrfach in den Durchschnitt eingehen. Deswegen hier eine Lösung mit `EXISTS`:

```
SELECT COUNT(*) anz,
       MIN(sal) min_sal,
       MAX(sal) max_sal,
       ROUND(AVG(sal)) avg_sal,
       COUNT(DISTINCT job) anz_jobs
FROM   emp e
WHERE  EXISTS(SELECT *
              FROM   emp u
              WHERE  u.mgr = e.empno)
```

„AS“ vor dem Namen der Ausgabespalte wäre möglich, ist aber optional.

Hausaufgabe 8c (4)

- Man kann auch eine Hilfstabelle nutzen, um Duplikate vor der Durchschnittsberechnung zu eliminieren:

```
WITH   vorgesetzte
AS     (SELECT DISTINCT ueber.empno,
                ueber.sal, ueber.job
         FROM   emp ueber, emp unter
         WHERE  unter.mgr = ueber.empno)
SELECT COUNT(*) anz,
       MIN(sal) min_sal,
       MAX(sal) max_sal,
       ROUND(AVG(sal)) avg_sal,
       COUNT(DISTINCT job) anz_jobs
FROM   vorgesetzte
```

Hausaufgabe 8c (5)

- Natürlich kann man die Unteranfrage genauso gut direkt unter FROM schreiben:

```
SELECT COUNT(*) anz,
       MIN(sal) min_sal,
       MAX(sal) max_sal,
       ROUND(AVG(sal)) avg_sal,
       COUNT(DISTINCT job) anz_jobs
FROM   -- Angestellte mit Untergebenen
       (SELECT DISTINCT ueber.empno,
               ueber.sal, ueber.job, ueber.comm
        FROM   emp ueber, emp unter
        WHERE  unter.mgr = ueber.empno) chef
```

Intern würde das DBMS die WITH-Klausel vermutlich so implementieren.

Hausaufgabe 8c (6)

Zusatzaufgabe:

- Ich hatte überlegt, als zusätzliche Spalte noch die Anzahl der Vorgesetzten auszugeben, die eine Provision haben, für die das Feld „comm“ also nicht Null ist.

  Im Beispiel würde 0 herauskommen. Nur Angestellte mit Job „SALESMAN“ haben eine Provision, diese haben im Beispielzustand aber keine Untergebenen.

- Wie könnte man das berechnen (unter SELECT in obiger Anfrage)? Wählen Sie die erste richtige Antwort.

  A. `COUNT(*)`

  B. `(SELECT COUNT(*) FROM emp WHERE comm IS NOT NULL)`

  C. `COUNT(COMM)`

  D. `SUM(CASE WHEN COMM IS NULL THEN 0 ELSE 1 END)`

Hausaufgabe 8c (7)

- Was halten Sie von diesem Lösungsversuch?

```
SELECT COUNT(*) anz, ...,
       (SELECT COUNT(*)
        FROM   emp e
        WHERE  e.empno = chef.empno
        AND    e.comm IS NOT NULL)
FROM   -- Angestellte mit Untergebenen
       (SELECT DISTINCT ueber.empno,
               ueber.sal, ueber.job, ueber.comm
        FROM   emp ueber, emp unter
        WHERE  unter.mgr = ueber.empno) chef
```

A. Korrekt
B. Syntaktisch korrekt, aber falsches Ergebnis
C. Syntaxfehler

Inhalt

Präsenzaufgabe: Fehlermeldungen

- Verwenden Sie das Schema „empdept_public“ im Adminer:
  - dept(deptno, dname, loc)
  - emp(empno, ename, job, mgr°→emp, hiredate, sal, comm°, deptno°→dept)
- Geben Sie vier Beispiele für fehlerhafte Anfragen und die zugehörige Fehlermeldung, die PostgreSQL ausgibt (4 Punkte).
  - Die Fehlermeldungen müssen unterschiedlich sein.

    Maximal zwei „syntax error at or near“.
  - Die Fehler sollen gefühlt häufig vorkommen.

    Wenn die Fehlermeldung nicht klar ist, müssen Sie den Fehler erläutern.
    Bei der Klausur können Sie wertvolle Zeit sparen, wenn Sie schon mit Fehlermeldungen vertraut sind.
  - Die Beispiel-Anfragen sollten kurz sein.

    Sie sollen nur den Fehler demonstrieren.

Fehlermeldungen (1)

- Wenn man die Hausaufgaben selbst bearbeitet (und nicht abschreibt), sollte man bis zur Klausur eine ganze Anzahl Fehlermeldungen gesehen haben.

  Fehler passieren. Wichtig ist, sie vollständig aufzuklären und daraus zu lernen.

- Das ist nicht ein negativer Unglücksfall, sondern es gehört zum praktischen Umgang mit einem DBMS dazu, dass man
  - häufigere Fehlermeldungen schon mal gesehen hat,
  - versteht, was das Problem ist, und

    Z.B., weil man sich erinnert, was früher das Problem war.

  - den Fehler zügig korrigieren kann.
- Man kann auch mal bewusst falsche Anfragen eingeben.

Fehlermeldungen (1)

- Z.B. Syntaxfehler:

```
SELECT deptno, FROM dept
ERROR: syntax error at or near "from"
LINE 1: select deptno, from dept
                       ^
```

Die Markierung der Position bekommt man in `psql` angezeigt, aber leider nicht im Adminer. Stand der Technik in der Syntaxanalyse ist, dass der Fehler an der ersten Stelle gemeldet wird, an der keine gültige Fortsetzung mehr möglich ist. Natürlich kann der tatsächliche Fehler davor liegen.
Stand der Technik ist eigentlich auch, dass angegeben wird, was an dieser Stelle erwartet wird (was dort legal wäre). Diese Information gibt PostgreSQL leider nicht aus.

- Den Präfix „`Fehler in der SQL-Abfrage (7):`“ gibt der Adminer immer aus. Kann man wohl ignorieren.

Fehlermeldungen (2)

- Z.B. Syntaxfehler:

```
SELECT deptno FROM

ERROR: syntax error at end of input
LINE 2:
```

- Fehler im Tabellen-Namen:

```
SELECT deptno FROM deptx

ERROR: relation "deptx" does not exist
LINE 1: select deptno from deptx
                           ^
```

Fehlermeldungen (3)

- Fehler im Spalten-Namen:

```
SELECT deptnr FROM dept
```

```
ERROR: column "deptnr" does not exist
LINE 1: select deptnr from dept
                ^
HINT: Perhaps you meant to reference the column
      "dept.deptno".
```

Bei meiner alten PostgresSQL-Version (9.2.24) gibt `psql` den Tipp nicht mit aus (auch dann nicht, wenn ich den Konfigurationsparameter `client_min_messages` auf „`info`“ setze).

Fehlermeldungen (4)

- Spaltenreferenz nicht eindeutig:

```
SELECT deptno FROM emp, dept
```

```
ERROR: column reference "deptno" is ambiguous
LINE 1: select deptno from emp, dept
                ^
```

- Tupelvariable nicht deklariert:

```
SELECT e.ename FROM emp
```

```
ERROR: missing FROM-clause entry for table "e"
LINE 1: select e.ename from emp
                ^
```

Beide Fehler auf dieser Folie kamen in abgegebenen Lösungen in der letzten Klausur vor (in der man die Anfragen mit dem Adminer testen konnte). Dabei scheinen die Fehlermeldungen doch sehr klar.

Fehlermeldungen (5)

- Typfehler:

```
SELECT ename / 2 FROM emp
```

```
ERROR: operator does not exist: text / integer
LINE 1: select ename / 2 from emp
                     ^
HINT: No operator matches the given name
      and argument types.
      You might need to add explicit type casts.
```

PostgreSQL erlaubt überladene Funktionen und Operatoren. Deswegen die Aussage „kein Operator passt auf diesen Namen und Argumenttypen“, und nicht einfach: „Das erste Argument muss eine Zahl sein“.

Bei der Version dieser Datenbank im Adminer ist `ename` mit dem Typ `text` (fast unbegrenzte Zeichenketten) deklariert. Das ist nicht sehr portabel.

Weiteres Beispiel: `select round(ename) from emp`.

Fehlermeldung: „`function round(text) does not exist`“.

Fehlermeldungen (6)

- `GROUP BY` fehlt:

```
SELECT deptno, count(*)
FROM   emp
```

```
ERROR: column "emp.deptno"
       must appear in the GROUP BY clause
       or be used in an aggregate function
LINE 1: select deptno, count(*)
                ^
```

In Aggregationsanfragen können unter `SELECT` außerhalb von Aggregationsfunktionen nur `GROUP BY`-Attribute genutzt werden.

- Korrekt wäre:

```
SELECT deptno, count(*)
FROM   emp
GROUP  BY deptno
```

Fehlermeldungen (7)

- Aggregationsfunktion unter `WHERE`:

```
SELECT deptno
FROM   emp
WHERE  count(*) = 3
GROUP  BY deptno
```

```
ERROR: aggregate functions are not allowed
       in WHERE
LINE 3: where count(*) = 3
                ^
```

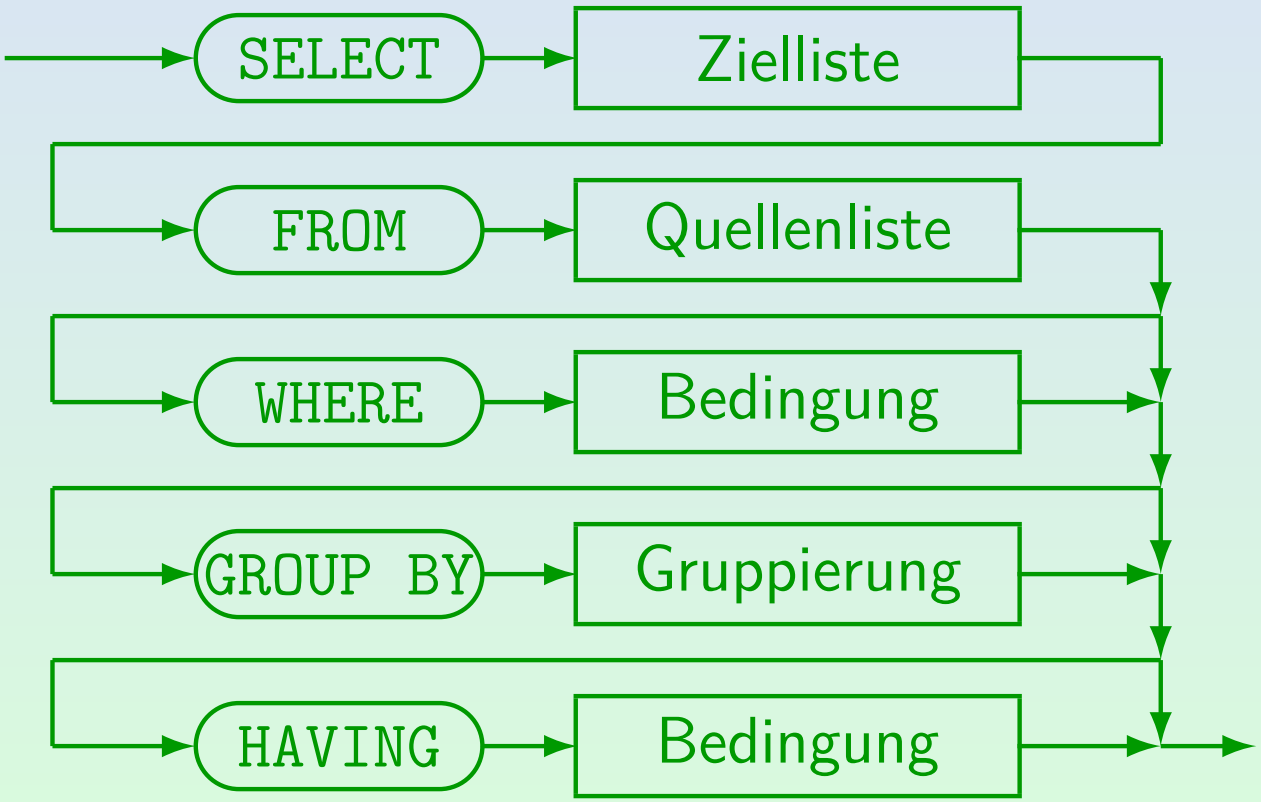
- Korrekt wäre:

```
SELECT deptno
FROM   emp
GROUP  BY deptno
HAVING count(*) = 3
```

Inhalt

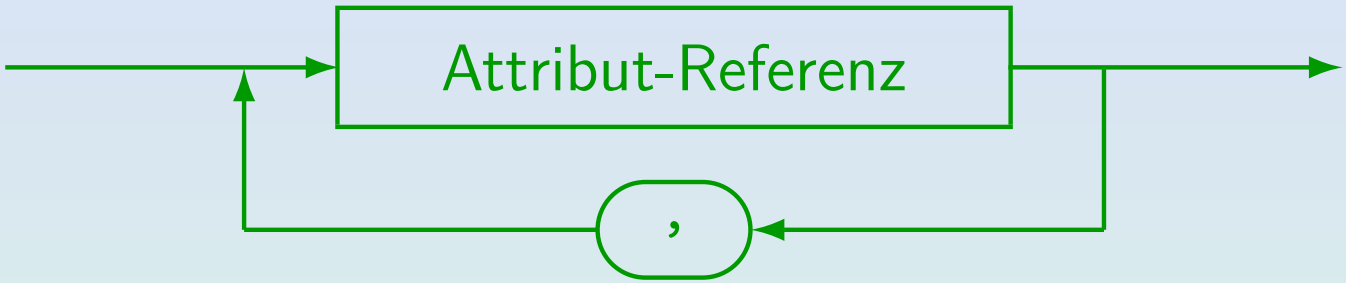
Syntax (1)

SELECT-Ausdruck:



Syntax (2)

Gruppierung:



- Z.B. `GROUP BY VORNAME, NACHNAME, B.SID`
- Oracle, SQL Server, DB2, Access und MySQL unterstützen den allgemeineren "Term" anstatt "Attribut-Referenz". Natürlich sind Aggregationen unter `GROUP BY` nicht gestattet.
- `GROUP BY` wurde in SQL-99 signifikant erweitert.

  `ROLLUP`, `CUBE` und `GROUPING SETS` werden in der Fortsetzungsvorlesung im Sommer besprochen.

Syntax (3)

Restriktionen:

- Eine Aggregation wird ausgeführt, wenn
  - eine Aggregation unter `SELECT` verwendet wird,
  - oder die `GROUP BY` oder `HAVING`-Klausel auftritt.
- Wird eine Aggregation ausgeführt, dann können unter `SELECT` und `HAVING` außerhalb von Aggregationen nur `GROUP BY`-Attribute genutzt werden (siehe aber unten).

  Innerhalb von Aggregationsfunktionen, d.h. als ihre Argumente, sind alle Attribute erlaubt. Man betrachte z.B. `AVG(A)/B`: Das Attribut `A` steht hier innerhalb der Aggregationsfunktion, `B` außerhalb.
- Unter `WHERE` dürfen keine Aggregationsfunktionen benutzt werden.

  Außer in geschachtelten Anfragen, und dort dann unter `SELECT` oder `HAVING`.

Auswertung

1. Alle Kombinationen von Zeilen der Tabellen unter `FROM` (Variablenbelegungen) werden betrachtet.
2. Die `WHERE`-Bedingung filtert eine Teilmenge heraus.
3. Dieses Zwischenergebnis wird in Gruppen aufgespalten, jeweils mit gleichem Wert in den `GROUP BY`-Attributen.
4. Gruppen, die die Bedingung in der `HAVING`-Klausel nicht erfüllen, werden eliminiert.
5. Für jede Gruppe wird durch Auswertung der Terme in der `SELECT`-Klausel eine Ausgabezeile erstellt.

Beispiel-Datenbank

STUDENTEN

SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	...
102	Michael	Grau	NULL
103	Daniel	Sommer	...
104	Iris	Winter	...

BEWERTUNGEN

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

AUFGABEN

ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

Aufgaben/Beispiele zu Aggregationen (1)

- Aufgabe: Gibt es einen echten Unterschied zwischen

```
SELECT THEMA, AVG(PUNKTE*100/MAXPT)
FROM   AUFGABEN A, BEWERTUNGEN B
WHERE  A.ATYP='H' AND B.ATYP='H' AND A.ANR=B.ANR
GROUP BY THEMA
```

und der Anfrage, die zusätzlich nach `A.ANR` gruppiert, die Aufgabennummer aber nicht ausgibt?

```
SELECT THEMA, AVG(PUNKTE*100/MAXPT)
FROM   AUFGABEN A, BEWERTUNGEN B
WHERE  A.ATYP='H' AND B.ATYP='H' AND A.ANR=B.ANR
GROUP BY THEMA, A.ANR
```

Ja/Nein-Umfrage (Ja: Es gibt einen Unterschied, Nein: Es gibt keinen.).

Aufgaben/Beispiele zu Aggregationen (2)

- Die folgende Anfrage soll alle Studenten ausgeben, die mindestens zwei Hausaufgaben gelöst haben:

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN S
WHERE  2 <= (SELECT COUNT(S.SID)
             FROM   BEWERTUNGEN B
             WHERE  B.SID = S.SID
             AND    B.ATYP = 'H')
```

- In der Unteranfrage wird aber `S.SID` gezählt, das für jede (konzeptionelle) Ausführung der Unteranfrage einen festen Wert hat. Funktioniert es trotzdem?

  A. Syntaxfehler
  B. Syntaktisch korrekt, aber falsches Ergebnis
  C. Korrekt

Aufgaben/Beispiele zu Aggregationen (3)

- Was halten Sie von dieser Anfrage?
  Wieder ist die Aufgabe, alle Studenten aufzulisten, die mindestens zwei Hausaufgaben gelöst haben.

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
AND    COUNT(B.ANR) >= 2
```

A. Syntaxfehler

B. Syntaktisch korrekt, aber falsches Ergebnis

C. Korrekt

Aufgaben/Beispiele zu Aggregationen (4)

- Würde dies funktionieren? (Wieder sind alle Studierenden mit mindestens zwei gelösten Hausaufgaben gesucht.)

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
GROUP  BY S.SID
HAVING COUNT(*) >= 2
```

A. Syntaxfehler

B. Syntaktisch korrekt, aber falsches Ergebnis

C. Korrekt

Aufgaben/Beispiele zu Aggregationen (5)

Lösung:

- Nach den Angaben in der Vorlesung ist das letzte Beispiel ein Syntaxfehler, weil `S.VORNAME` und `S.NACHNAME` nicht unter `GROUP BY` stehen.

  Siehe aber die nächste Folie.

- Man könnte sie hinzufügen, ohne die Gruppen zu ändern, und alles wäre gut:

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
GROUP  BY S.SID, S.VORNAME, S.NACHNAME
HAVING COUNT(*) >= 2
```

Aufgaben/Beispiele zu Aggregationen (6)

Lösung, Forts.:

- In PostgreSQL wird es aber auch ohne diese Hinzufügung akzeptiert, weil nach dem Schlüssel `S.SID` von `S` gruppiert wird, und PostgreSQL daher weiss, dass alle anderen Attribute von `S` eindeutig bestimmt sind.

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
GROUP  BY S.SID
HAVING COUNT(*) >= 2
```

- In Oracle und Microsoft SQL Server gibt dies einen Fehler.

  Sie müssen mit Punktabzug für Stil (Portabilität) rechnen. Es ist im Standard (seit SQL-99), aber auch PostgreSQL implementiert nur den einfachsten Fall.

Aufgaben/Beispiele zu Aggregationen (7)

- Und was ist mit dieser Anfrage? Hier ist die Aufgabe, die Anzahl der Hausaufgaben für jeden Studenten aufzulisten.

```
SELECT   S.SID, S.VORNAME, S.NACHNAME, SUM(B.ANR)
FROM     STUDENTEN S, BEWERTUNGEN B
WHERE    S.SID = B.SID
AND      B.ATYP = 'H'
GROUP BY S.SID, S.VORNAME, S.NACHNAME, B.ANR
```

A. Syntaxfehler

B. Syntaktisch korrekt, aber mehrere Zeilen pro Student

C. Synt. korrekt, eine Zeile pro Student, letzte Spalte falsch

D. Korrekt

Inhalt

Operationen der Relationalen Algebra

Die fünf Basisoperationen der relationalen Algebra:

- $\sigma_F(R)$: Selektion

  Dabei ist F eine atomare Formel (bezüglich der Datentyp-Signatur des DBMS und einer Variablenbelegung, bei der die Spalten von R Variablen des entsprechenden Typs sind). Typische Bedingungen sind: $A_i = c$ und $A_i = A_j$.

- $\pi_{A_1,\ldots,A_n}(R)$: Projektion

  Allgemein mit Umbenennung von Spalten und beliebigen Termen: $\pi_{A_1 \leftarrow t_1,\ldots,A_n \leftarrow t_n}(R)$. Falls $t_i = A_i$ kann man statt $A_i \leftarrow A_i$ einfach A_i schreiben.

- $R \times S$: Kartesisches Produkt
- $R \cup S$: Vereinigung
- $R \setminus S$: Mengendifferenz

Aufgabe

- Welche der folgenden Ausdrücke der relationalen Algebra sind syntaktisch korrekt?
  Was bedeuten sie?

  - ☐ $\texttt{STUDENTEN}$.
  - ☐ $\sigma_{\texttt{MAXPT} \neq 10}(\texttt{AUFGABEN})$.
  - ☐ $\pi_{\texttt{VORNAME}}(\pi_{\texttt{NACHNAME}}(\texttt{STUDENTEN}))$.
  - ☐ $\sigma_{\texttt{PUNKTE} \leq 5}(\sigma_{\texttt{PUNKTE} \geq 1}(\texttt{BEWERTUNGEN}))$.
  - ☐ $\sigma_{\texttt{PUNKTE}}(\pi_{\texttt{PUNKTE} = 10}(\texttt{BEWERTUNGEN}))$.

RelaX (1)

- RelaX ist eine Beispiel-Implementierung der relationalen Algebra zu Lehrzwecken:
  [http://dbis-uibk.github.io/relax/calc/local/uibk/local/0]

  Publikation: Johannes Kessler, Michael Tschuggnall and Günther Specht: RelaX: A Webbased Execution and Learning Tool for Relational Algebra, In: Datenbanksysteme für Business, Technologie und Web (BTW), 2019.
  [https://dbis.uibk.ac.at/sites/default/files/2019-03/2019-BTW-Relax.pdf]

- Ich habe einen „GIST“ erstellt mit den Daten der Beispiel-DB:
  [https://gist.github.com/8dc2652578ee12ae756a234c4cf21b3f]

  „A gist is a shareable file that you can edit, clone, and fork on GitHub.“

- Mit folgender URI bekommt man RelaX für die Beispiel-DB:
  [http://dbis-uibk.github.io/relax/calc/gist/8dc2652578ee12ae756a234c4cf21b3f]

RelaX (2)

- Man kann die griechischen Buchstaben durch Anklicken des Symbols in der Palette oben einfügen.
- Man kann die Buchstaben aber auch in ASCII schreiben, z.B. `sigma`, `pi`.

  Vermutlich ist das für die Abgabe der Lösungen und eventuelle Diskussionen per EMail günstiger. Vielleicht ist aber die Unicode-Untersützung gut genug.
- Leider gibt es bei RelaX keine Indexschreibweise, auch `[...]` funktioniert nicht. Man schreibt die Bedingung bzw. die Attributliste einfach nach dem Symbol in der gleichen Zeile weiter:

  ```
  sigma SID = 101 (STUDENTEN)
  ```

  Die runden Klammern um die Eingaberelation sind nicht nötig, sondern nur ein Versuch, die Eingabe optisch vom Index abzusetzen. Wenn man möchte, kann man auch den Index in Klammern setzen.

RelaX (3)

- Wenn man auf „Query ausführen“ klickt, wird die Anfrage unten richtig mit Index-Schreibweise angezeigt.
- Außerdem wird sie als Operator-Baum angezeigt, was bei komplexeren Anfragen mit mehreren Joins nützlich ist.
  - Wenn man mit dem Cursor über einen Knoten im Operatorbaum fährt, wird als „Tooltip“ das Schema der jeweiligen Zwischenrelation angezeigt.
  - Im Knoten steht auch die Anzahl Tupel des Zwischenergebnisses.
  - Wenn man auf den Knoten klickt, wird das Zwischenergebnis angezeigt.

    Anstelle des Ergebnisses des ganzen Ausdrucks.

RelaX (4)

- Beachten Sie, dass RelaX bei Tabellen- und Spaltennamen Wert auf die korrekte Groß-/Kleinschreibung legt.
- Die Attributnamen bei RelaX sind intern immer zweiteilig, bestehend aus Relationenname und dem eigentlichen Spaltennamen, z.B. `STUDENTEN.SID`.
- Wenn der eigentliche Spaltenname (z.B. `SID`) eindeutig ist, reicht er, um eine Spalte anzusprechen.

  Das ist anders als in der Vorlesung. Die Konsequenzen für die Bewertung von Klausuraufgaben sind noch unklar. Halten Sie sich besser an die Vorlesung.
- In der Vorlesung heißt die Spalte einfach `SID`, und es wäre ein Fehler, `STUDENTEN.SID` zu schreiben, wenn man nicht vorher den Operator $\rho_{\texttt{STUDENTEN}}$ verwendet hat, um die Spalte umzubenennen (geht auch in RelaX).

Inhalt

Präsenzaufgabe: Erste RA-Anfragen

Schreiben Sie folgende Anfragen in relationaler Algebra:

- Geben Sie alle Bewertungen für Hausaufgabe 1 aus, bei denen weniger als 10 Punkte erzielt wurden.

  Drucken Sie die komplette Zeilen der Tabelle `BEWERTUNGEN`.

- Geben Sie die E-Mail-Adresse von Lisa Weiss aus.

  Schreiben Sie die Anfragen in ASCII, so dass RelaX sie versteht.
  [http://dbis-uibk.github.io/relax/calc/gist/8dc2652578ee12ae756a234c4cf21b3f]

Dies bezieht sich auf das bekannte Schema:

- STUDENTEN(SID, VORNAME, NACHNAME, EMAIL^o)
- AUFGABEN(ATYP, ANR, THEMA, MAXPT)
- BEWERTUNGEN(SID$\rightarrow$STUDENTEN, (ATYP, ANR)$\rightarrow$AUFGABEN, PUNKTE)