# Einführung in Datenbanken

# Übung 11: Aggregationsfunktionen, GROUP BY, CASE

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

Wintersemester 2023/24

http://www.informatik.uni-halle.de/~brass/db23/

# Inhalt

# Haben Sie Fragen/Anmerkungen?

- Die Übung ist dafür da, dass Sie Fragen stellen können.
- Sie soll keineswegs ein Monolog werden.
- Haben Sie Fragen?
  - Organisatorisches?
  - Fragen zum Vorlesungsstoff?
- Haben Sie interessante neue Einsichten, die Sie teilen wollen?

# Inhalt

# 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)

- 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 (2)

Lösung:

- Die Anfrage sollte funktionieren (d.h. „C“ ist korrekt):
  - `COUNT` zählt Duplikate mit. Effektiv werden hier überhaupt nur Duplikate des einen möglichen Wertes gezählt.
  - Da `S.SID` auch nicht Null sein kann, ist `COUNT(S.SID)` hier äquivalent zu `COUNT(*)`.

    Natürlich ist `COUNT(S.SID)` schlechter Stil. Es verschleiert die Intention.
- PostgreSQL 14.0 gibt aber die Fehlermeldung:

  `aggregate functions are not allowed in WHERE`
- Das ist ein Bug.

  Mögliche Erklärung: Der Optimierer schreibt die Anfrage um, weil er glaubt, den Term `COUNT(S.SID)` aus der Unteranfrage herausziehen zu können (weil unabhängig von Tupelvariablen der Unteranfrage). Das ist falsch.

# 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)

Lösung:

- Das war nach der vorangegangenen Diskussion einfach:
  - Es ist natürlich ein Syntaxfehler.
  - Aggregationsfunktionen wie `COUNT` können nicht direkt unter `WHERE` verwendet werden.

    Aber schon in Unteranfragen, die unter `WHERE` geschachtelt sind.
  - Es ist also „A“ richtig.

# Aufgaben/Beispiele zu Aggregationen (5)

- 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 (6)

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 (7)

## 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 (8)

- 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

# Aufgaben/Beispiele zu Aggregationen (9)

Lösung:

- Es ist „B“ richtig.
- Die Gruppierung auch nach der Aufgabennummer (`B.ANR`) führt dazu, dass jede Gruppe effektiv nur aus einer Zeile besteht.

  Es wird nach `S.SID` und `B.ANR` gruppiert, und `B.SID` ist gleich `S.SID` und `B.ATYP` hat einen festen Wert. Damit liegen die Werte von Schlüsseln für `S` und `B` für jede Gruppe fest.
- Außerdem ist `SUM` statt `COUNT` falsch.

# 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

# 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

# Übungsblatt 10, Aufgabe 4: Fehlermeldungen

- Verwenden Sie das Schema „empdept_public“ im Adminer:
  - dept(deptno, dname, loc)
  - emp(empno, ename, job, mgr$^{\circ}\rightarrow$emp, hiredate, sal, comm$^{\circ}$, deptno$^{\circ}\rightarrow$dept)
- Geben Sie drei Beispiele für fehlerhafte Anfragen und die zugehörige Fehlermeldung, die PostgreSQL ausgibt (3 Punkte).
  - Die Fehlermeldungen müssen unterschiedlich sein.

    Maximal ein „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 einer 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

# Bitte beachten!

- Anfragen zählen nur dann als korrekt, wenn sie in allen möglichen Datenbank-Zuständen (die die Integritäts-Bedingungen erfüllen) das richtige Ergebnis liefern.
- Die Anfrage beschreibt ja eine Abbildung/Funktion von Datenbank-Zuständen auf Ergebnis-Relationen.
- Sie dürfen deshalb z.B. nicht IDs von Präparaten in der Datenbank nachschauen und in Ihre Anfrage einsetzen.
- Wenn in der Anfrage nur der Name eines Präparates steht, dürfen Sie nur diesen Namen verwenden.
- In einem anderen Datenbank-Zustand könnte das gleiche Präparat eine andere ID haben.

# Schema der Vitamin-Datenbank

- Schema vit_public im Adminer:
  - Stoff_Kategorie(Kat, Bezeichnung, Sort_Nr)

    Neue Tabelle: Z.B. Vitamin, Mineralstoff, Spurenelement, ...
  - Stoff(Vit, Einheit, Tagesdosis°, Kat → Stoff_Kategorie)
  - Praeparat(Pid, Name, Hersteller, PZN°, Anz°, Einheit°, Tagesdosis°, Gewicht°, Preis°, glutenfrei°, lactosefrei°)
  - Inhalt(Pid → Praeparat, Vit → Stoff, Menge, Prozent°, Anmerkung°)
  - Zutat(Pid → Praeparat, Seq, Name, Anmerkung°)

# Hausaufgabe 10.1: CASE (1)

- `Inhalt(`$\underline{\texttt{Pid}} \to$ `Praeparat, `$\underline{\texttt{Vit}} \to$ `Stoff, Menge, Prozent°, Anmerkung°)`
- Drucken Sie eine Liste der Inhaltsstoffe des Präparats „A-Z Komplett“ von dem Hersteller „Abtei“.
- Falls die Spalte „`Anmerkung`“ nicht den Nullwert enthält, steht dort der Inhaltsstoff, wie er auf der Packung angegeben ist. Drucken Sie dann diesen Namen, und sonst die standardisierte Bezeichnung des Stoffes aus der Spalte „`Vit`“.

  Geben Sie bitte außerdem die Menge und die Einheit aus, sowie die Stoff-Kategorie `kat` und die `sort_nr` der Stoff-Kategorie. Sortieren Sie das Ergebnis mit erster Prioritaet nach der Sortier-Nummer der Stoff-Kategorie und mit zweiter Priorität nach dem Namen des Stoffes (`Anmerkung` bzw. `Vit`).

# Hausaufgabe 10.1: CASE (2)

- Das Ergebnis sollte so aussehen:

| Name | menge | einheit | kat | sort_nr |
|---|---|---|---|---|
| Biotin | 50.0 | ug | V | 1 |
| Folsäure | 200.0 | ug | V | 1 |
| Niacin (NE) | 16.0 | mg | V | 1 |
| Pantothensäure | 6.0 | mg | V | 1 |
| Vitamin A (RE) | 800.0 | ug | V | 1 |
| Vitamin B1 | 1.1 | mg | V | 1 |
| Vitamin B12 | 2.5 | ug | V | 1 |
| Vitamin B2 | 1.4 | mg | V | 1 |
| Vitamin B6 | 1.4 | mg | V | 1 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

Die Ausgabespalten sollen so heißen wie hier gezeigt.

# Hausaufgabe 10.1: CASE (3)

- Stoff_Kategorie(Kat, Bezeichnung, Sort_Nr)
  Stoff(Vit, Einheit, Tagesdosis°,
  Kat → Stoff_Kategorie)
- Mögliche Lösung:

```
SELECT COALESCE(I.Anmerkung,I.Vit) AS "Name",
       I.Menge, S.Einheit, K.Kat, K.Sort_Nr
FROM   Praeparat P, Inhalt I, Stoff S,
       Stoff_Kategorie K
WHERE  P.Name = 'A-Z Komplett'
AND    P.Hersteller = 'Abtei'
AND    P.Pid = I.Pid
AND    I.Vit = S.Vit
AND    S.Kat = K.Kat
ORDER  BY K.Sort_Nr, "Name"
```

# Hausaufgabe 10.1: CASE (4)

- Sie bekommen einen Bonuspunkt, wenn Sie das Problem mit der offensichtlich falschen Einsortierung von Vitamin B12 lösen.

  Sie könnten z.B. in der Sortierungs-Spezifikation den Wert „Vitamin B12“ durch den Wert „Vitamin B92“ ersetzen. Das ist keine schöne Lösung.
  Im Schema fehlt leider eine Sortierungs-Spezifikation für die Stoffe.

- Mögliche Lösung:

```
ORDER BY K.Sort_Nr,
         CASE WHEN I.Vit = 'B12'
              THEN 'Vitamin B92'
              ELSE COALESCE(I.Anmerkung,I.Vit)
         END
```

  Es gibt eine Fehlermeldung, wenn man `"Name"` in einem strukturierten Term verwendet („`Column "Name" does not exist`“).

# Hausaufgabe 10.2: Maximierung (1)

- Welches Präparat bzw. welche Präparate enthalten die maximale Menge an „Zink“?

  Geben Sie die Pid und den Namen des Präparates aus sowie die (maximale) Menge an Zink.

- Das Ergebnis sollte so aussehen:

| pid | name | menge |
|---|---|---|
| 3 | A-Z Komplett | 10.0 |
| 5 | Kardiodrink | 10.0 |

  Die Ausgabespalte soll so heißen wie hier gezeigt.

- Da Aggregationsfunktionen nun in der Vorlesung behandelt wurden, sind Sie aufgefordert, diese hier auch zu verwenden (in einer Unteranfrage).

  Notfalls dürfen Sie wie bisher eine Lösung mit `NOT EXISTS` entwickeln.

# Hausaufgabe 10.2: Maximierung (2)

- Mögliche Lösung:

```
SELECT P.Pid, P.Name, I.Menge
FROM   Praeparat P, Inhalt I
WHERE  P.Pid = I.Pid
AND    I.Vit = 'Zink'
AND    I.Menge = (SELECT MAX(Menge)
                  FROM   Inhalt
                  WHERE  Vit = 'Zink')
```

# Hausaufgabe 10.2: Maximierung (3)

- Alternative:

```
SELECT P.Pid, P.Name, I.Menge
FROM   Praeparat P, Inhalt I
WHERE  P.Pid = I.Pid
AND    I.Vit = 'Zink'
AND    NOT EXISTS(SELECT *
                  FROM   Inhalt Mehr
                  WHERE  Mehr.Vit = 'Zink'
                  AND    Mehr.Menge > I.Menge)
```

# Hausaufgabe 10.2: Maximierung (4)

- Nicht korrekte Lösung:

```
SELECT P.Pid, P.Name, I.Menge
FROM   Praeparat P, Inhalt I
WHERE  P.Pid = I.Pid
AND    I.Vit = 'Zink'
ORDER  BY MENGE DESC
FETCH  FIRST ROW ONLY
```

- „`FETCH FIRST ROW ONLY`“ wird erst in der Vorlesung „Datenbank-Programmierung“ im Sommersemester behandelt.

  Es wird nur die erste Ergebniszeile gedruckt. In MySQL wäre es „`LIMIT 1`“.

- Das Problem ist, dass es mehre Präparate mit der gleichen (maximalen) Menge an Zink geben könnte (und gibt). Diese Lösung liefert nur eins davon.

# Hausaufgabe 10.3: Aggregationsfunktionen (1)

- Geben Sie für jeden Stoff der Kategorien „M“ und „S“ (Mineralstoffe und Spurenelemente) aus,
  - in wie vielen der Nahrungsergänzungsmittel er enthalten ist,
  - was die minimale und maximale Menge ist und
  - was die durchschnittliche Menge ist (gerundet auf eine ganze Zahl).
- Geben Sie außerdem die empfohlene Tagesdosis und die Einheit für diesen Stoff aus.
- Sortieren Sie das Ergebnis alphabetisch nach dem Stoff.

Die Ausgabespalten auf der nächsten Folie sind etwas gekürzt. Eigentlich heißen sie Stoff, Anzahl, Min, Max, Durchschnitt, täglich, Einheit.

# Hausaufgabe 10.3: Aggregationsfunktionen (2)

- Das Ergebnis sollte so aussehen:

| Stoff | Anz. | Min | Max | AVG | tägl. | Einh. |
|---|---|---|---|---|---|---|
| Calcium | 4 | 120.0 | 199.0 | 155 | 800 | mg |
| Chrom | 5 | 25.0 | 60.0 | 41 | 40 | ug |
| Eisen | 4 | 2.1 | 14.0 | 7 | 14 | mg |
| Fluor | 1 | 3.5 | 3.5 | 4 | 4 | mg |
| Jod | 4 | 100.0 | 150.0 | 113 | 150 | ug |
| Kalium | 1 | 499.0 | 499.0 | 499 | 2000 | mg |
| Kupfer | 4 | 500.0 | 1040.0 | 860 | 1000 | ug |
| Magnesium | 7 | 56.0 | 375.0 | 157 | 375 | mg |
| Mangan | 3 | 1.0 | 2.0 | 2 | 2 | mg |
| Molybdän | 4 | 20.0 | 80.0 | 50 | 50 | ug |
| Natrium | 1 | 43.3 | 43.3 | 43 | NULL | mg |
| Phosphor | 3 | 105.0 | 155.0 | 128 | 700 | mg |
| Selen | 6 | 10.0 | 55.0 | 34 | 55 | ug |
| Zink | 6 | 5.0 | 10.0 | 7 | 10 | mg |

# Hausaufgabe 10.3: Aggregationsfunktionen (3)

- Mögliche Lösung:

```
SELECT S.Vit AS "Stoff",
       COUNT(*) as "Anzahl",
       MIN(MENGE) AS "Min",
       MAX(MENGE) AS "Max",
       ROUND(AVG(MENGE)) AS "Durchschnitt",
       S.Tagesdosis AS "täglich",
       S.Einheit AS "Einheit"
FROM   Stoff S, Inhalt I
WHERE  S.Vit = I.Vit
AND    S.KAT IN ('M','S')
GROUP  BY S.Vit, S.Tagesdosis, S.Einheit
ORDER  BY S.Vit
```

# Inhalt

# Präsenzaufgabe: Erste RA-Anfrage

Schreiben Sie folgende Anfrage in relationaler Algebra:

- Geben Sie die SID und die Punktzahl aller Bewertungen von Abgaben für Hausaufgabe 1 mit mindestens 8 Punkten aus.

  Schreiben Sie die Anfragen in ASCII, so dass RelaX sie versteht.
  [http://dbis-uibk.github.io/relax/calc/gist/8dc2652578ee12ae756a234c4cf21b3f]
  Auf die Webadresse wird auch unter „Übung“ auf der Webseite der Vorlesung verlinkt. Sie brauchen die URI nicht abzutippen.

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)