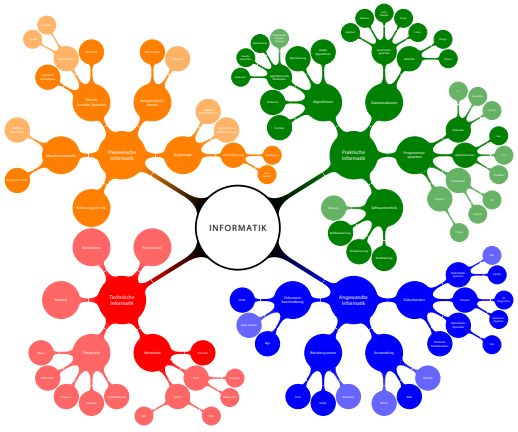
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 39

SQL – Fortgeschrittene Anfragen

Wie viele?

Vorlesung Einführung in die Informatik 2 vom 10. Juni 2014 von Till Tantau



Lernziele von Kapitel 39

1. SQL-Aggregate verstehen und benutzen können
2. SQL-Unteranfragen verstehen und benutzen können

Gliederung von Kapitel 39

Sind die Ställe groß genug?

Das Ziel

- Die Firma *Molecular Sheep* möchte, dass ihre Schafen glücklich sind.
- Dazu möchte sie garantieren, dass nicht zu viele Schafe in einem Stall schlafen, jedes Schaf soll mindestens vier Quadratmeter Platz haben.

Das Problem

- Es gibt keine Zeile in der Datenbank, in der steht, wie viele Schafe in einem Stall liegen.
- Auch mit einem Join lässt sich das Problem nicht lösen, da auch so keine *Zeile* erzeugt werden kann, die diese Zahl enthält.
- Wir brauchen eine Methode, »*etwas mit Spalten zu machen*«.

Selektion versus Aggregate.

Die Liegt-In-Tabelle:

Schafname	Stallnummer	Seit
Dolly	1	2012
Peter	1	2013
Max	2	2012
Ferdinand	2	2012
Flauschi	1	2013

Bei einer *Selektion* (in SQL: `where` . . .) haben wir Zugriff auf

- alle Spalten *einer Zeile*.
- Da die Spalten feste Namen haben, können wir uns direkt auf diese beziehen.

Bei einer *Aggregation* wollen wir Zugriff auf

- alle Zeilen *einer Spalte*.
- Die Anzahl an Zeilen variabel ist, könne wir und nicht auf einzelnen Zeilen beziehen.

Die Syntax der einfachen Aggregation.


```
select Aggregatsfunktion(Spaltenauswahl)
  from ...
 where ...
```

Hierbei ist die `Aggregatsfunktion` eine der folgenden möglichen Funktion, die »Dinge aggregieren«:

- `avg(a)` liefert den Durchschnitt des Attributs `a`.
- `count(a)` liefert die Anzahl an Zeilen.
- `count(distinct a)` liefert die Anzahl an unterschiedlichen Zeilen (in Bezug auf `a`).
- `max(a)` und `min(a)` liefern das Maximum und das Minimum.
- `std(a)` und `stddev(a)` liefern beide die Standardabweichung.
- `sum(a)` liefert die Summe.

(Es gibt noch mehr Funktionen, man findet sie in der SQL-Referenz.)

Semantik (=Bedeutung) der einfachen Aggregation.

Ein Befehl der Form

```
select Aggregatsfunktion(Spaltenauswahl)
  from ...
 where ...
```

hat folgenden Effekt:

- Zunächst wird der `from ... where ...` Teil ganz normal ausgewertet. Hier kann es beliebig komplizierte Joins und Selektionen geben.
- Ebenso wird die `Spaltenauswahl` »ganz normal zeilenweise« ausgewertet. Normalerweise steht hier nur ein Attribut (ein Spaltenname), es können aber auch Ausdrücke wie `length(name)` hier stehen.
- Das Resultat ist dann eine neue Tabelle.
- In dieser Tabelle wird dann die Spalte `Spaltenauswahl` betrachtet und auf ihre Zeilen die angegebene Aggregatsfunktion angewandt.

Beispiele der einfachen Aggregation.

Beispiel (Der größte Stall)

```
select max(groesse) from stall;
```

Beispiel (Die durchschnittliche Stallgröße)

```
select sum(groesse) / count(groesse) from stall;
```

oder

```
select avg(groesse) from stall;
```

Beispiel (Die durchschnittliche Länge der Name von Schafen)

```
select avg(length(name)) from schaf;
```

5-Minuten-Aufgabe

Zur Übung

Nehmen wir an, die folgenden Befehle wurden ausgeführt und die Tabellen gefüllt.

```
create table schaf     (name char(20),
                        farbe char(20),
                        fell char(20));
create table liegt_in  (name char(20), nummer integer);
create table stall     (nummer integer, groesse float);
insert into ...;
```

Geben Sie einen SQL-Befehl an, der die maximale Größe von Ställen berechnet, in denen schwarze Schafe liegen.

Aggregation von Teilen von Spalten.

Zurück zu unserem Ausgangsproblem:

- Jedes Schaf soll mindestens vier Quadratmeter Platz heben.

Für *einen konkrete Stallnummer* (wie »Stall 1«) können wir dies nun wie folgt überprüfen:

```
mysql> select count(name) * 4, max(groesse)
         from liegt_in natural join stall
        where nummer = 1;

+-----------------+--------------+
| count(name) * 4 | max(groesse) |
+-----------------+--------------+
|              12 |           50 |
+-----------------+--------------+
```

- Wir wollen diese Daten aber für *alle* Ställe!
- Wir wollen also viele mit »`where nummer = ...`« erstellte Tabellen in eine Tabelle zusammenfassen.
- Dies nennt man *gruppieren*.

Die gruppierte Aggregation.

Syntax

```
select      Spalte1, Spalte2, ..., Aggregatsf.(Spalte)
      from ...
     where ...
  group by Spalte1, Spalte2, ...
```

Semantik

- Zunächst wird wieder `from ... where ...` normal ausgeführt, was eine Tabelle T ergibt.
- In T kann es nun mehrere Zeilen geben, die in Bezug auf die Werte in den Spalten `Spalte1, Spalte2, ...` gleich sind.
- (Nur) diese Zeilen werden nun gemäß der Aggregatsfunktion zusammengefasst.
- Das Ergebnis bildet dann zusammen mit der Werten aus den ersten Spalte *eine* Zeile in der Ergebnistabelle.

Beispiele gruppierter Anfragen

Wie viele Schafe schlafen in einem Stall?

```
mysql> select nummer, count(name)
         from liegt_in
     group by nummer;

+--------+-------------+
| nummer | count(name) |
+--------+-------------+
|      1 |           3 |
|      2 |           2 |
+--------+-------------+
```

Größe versus Anzahl an Schafen in einem Stall?

```
mysql> select nummer, count(name)*4, max(groesse)
         from liegt_in natural join stall
     group by nummer;

+--------+---------------+--------------+
| nummer | count(name)*4 | max(groesse) |
+--------+---------------+--------------+
|      1 |            12 |           50 |
|      2 |             8 |            6 |
+--------+---------------+--------------+
```

Einschränkung der Ergebnisse einer Gruppierung.

- Wir können mittels Gruppierung nun eine Tabelle erzeugen, die alle Informationen enthält, die wir benötigen.
- Nun wollen wir hieraus wiederum die *Zeilen* auswählen, bei denen die Größe der Ställe zu klein ist.
- Hierfür können wir aber *nicht* ein `where ...` verwenden:
- Ein `where ...` wird immer ausgewertet, *bevor* gruppiert wird –
- wir wollen aber *nach* der Gruppierung Zeilen streichen.

Syntax

```
select      Spalte1, Spalte2, ..., Aggregatsf.(Spalte)
      from ...
     where ...
  group by Spalte1, Spalte2, ...
    having ...;
```

Der Test nach `having` wird erst *nach* der Gruppierung ausgeführt und darf deshalb *aggregierte Werte* nutzen.

Das komplette Beispiel.


```
mysql> select nummer
         from liegt_in natural join stall
     group by nummer
       having count(name) * 4 > max(groesse);

+--------+
| nummer |
+--------+
|      2 |
+--------+
```

5-Minuten-Aufgabe

Zur Übung

Geben Sie einen SQL-Befehl an, der alle Ställe ausgibt, in denen mehr als drei schwarze Schafe schlafen.

Wie viele problematische Ställe gibt es?

- Wir können mittlerweile eine Tabelle erstellen, die alle Ställe enthält, in denen zu viele Schafe liegen.
- Wir können aber noch *nicht* zählen, wie viele Zeilen diese Tabelle hat.
- Der Grund ist, dass wir dazu nach der Gruppierung und nach der Filterung mittels »having« *erneut* aggregieren wollen.
- Statt einem zweiten Group-by sieht SQL hierfür ein allgemeineres Konzept vor: Die *Unteranfrage* (»subquery«).

Unteranfragen als Tabellen

- Wählt man mit `from ...` aus Tabellen etwas aus, so darf statt eines Tabellenamen auch stehen

  ```
  (select ... ) as ein_name
  ```

- Hierbei darf der `select`-Befehl beliebig kompliziert sein.
- Der *Effekt* ist, als ob man das Ergebnis des `select`-Befehls zuerst in eine temporäre Tabelle namens `ein_name` eingefügt hätte und diese dann normal genutzt hätte.

Beispiele von Unteranfragen als Tabellen.

Beispiel (Zwei äquivalente Anfragen)

```
select name from (select *
                    from schafe
                   where farbe = "schwarz") as
                       schwarze_schafe
            where fell = "lockig";
```

liefert dasselbe wie

```
select name from schafe
            where farbe = "schwarz" and
                  fell = "lockig";
```

Beispiel (Wie viele problematische Ställe gibt es?)

```
select count(*)
  from (select nummer
          from liegt_in natural join stall
      group by nummer
        having count(name) * 4 > max(groesse)
        ) as problematisch;
```

5-Minuten-Aufgabe

Zur Übung

Geben Sie eine SQL-Befehl an, um die *durchschnittliche Anzahl an Schafen pro Stall* zu bestimmen.

Unteranfragen als Werte

- Eine besondere Art an Unteranfragen sind solche, die als Ergebnis ein Tabelle mit *genau einer Zeile und genau einer Spalte* liefern.
- In diesem Fall erlaubt es SQL, diesen einen Eintrag »wie einen normalen Wert« praktisch überall zu nutzen, wo man eben Werte nutzt.

Syntax von Werten aus Unteranfragen

- An (fast) allen Stellen, wo in SQL ein Wert benötigt wird, kann statt dessen stehen

```
(select ...)
```

- Voraussetzung ist, dass die Unteranfrage eine Tabelle mit genau einer Zeile und genau einer Spalte liefert.

Beispiel (Liste alle überdurchschnittlich großen Ställe)

```
select * from stall
        where groesse > (select avg(groesse) from stall);
```

Unteranfragen mit Quantoren

- Unteranfragen werden häufig wie eben benutzt, um ein Attribut mit »dem Wert« zu vergleichen, den eine Unteranfrage liefert:

```
select name from liegt_in
           where seit = (select max(jahr)
                         from krank
                        where name = "Dolly");
```

  Dies findet alle Schafe, die in einem Stall liegen genau seit dem letzten Jahr, in dem Dolly krank war. (Die Krankheitstabelle habe die Spalten »krank« und »name«.)

- Wir wollen nun die Schafe finden, die in einem Stall liegen seit *irgendeinem* Jahr (und nicht dem letzten), in dem Dolly krank war.
- Hierzu bietet SQL eine besondere Syntax:

```
select name from liegt_in
           where seit = any (select jahr
                             from krank
                            where name = "Dolly");
```

Unteranfragen mit Quantoren

Allgemein gilt:

- Schreibt man `... = any (select ...)`, so darf die Unteranfrage eine Spalte mit *mehreren* Zeilen zurückliefern.
- Der Ausdruck ist dann wahr, wenn *wenigstens eine Zeile* in der Tabelle gleich der linken Seite vom Gleichheitszeichen ist.

Allgemeine Syntax von Unteranfragen mit Quantoren

Any-Unteranfragen

- `... = any (select ...)` ist wahr, wenn die linke Seite gleich *wenigstens einer* Zeile der Unteranfrage ist.
- Statt = kann man auch <, <=, >, >= und <> (ungleich) schreiben mit der entsprechenden Bedeutung.

All-Unteranfragen

- `... = all (select ...)` ist wahr, wenn die linke Seite gleich *allen* Zeilen der Unteranfrage ist.
- Statt = kann man wieder auch <, <=, >, >= und <> schreiben.

Exists-Unteranfragen

`exists (select ...)` ist wahr, wenn die Unterfrage wenigstens eine Zeile zurückliefert.

Beispiele von Unteranfragen mit Quantoren

Beispiel (Welche Schafe haben keinen Stall?)

```
select name
  from schaf
 where not exists (select * from liegt_in
                           where liegt_in.name =
                                 schaf.name);
```

Beispiel (Welche Schafe schlafen in einem Stall, in dem kein schwarzes Schaf schläft?)

```
select name
  from liegt_in
 where nummer <> all (select nummer
                        from liegt_in natural join schaf
                       where farbe = "schwarz");
```

Zusammenfassung

Aggregate und Gruppierungen

```
select      Spalte1, Spalte2, ...,
    Aggregatsfunktion(Spalte)
      from ...
     where ...
  group by Spalte1, Spalte2, ...
    having ...;
```

- Alle Zeilen, die die *gleichen Werte* haben in den bei **`group by`** angegebenen Spalten werden zusammengefasst.
- *Aggregatsfunktionen* beziehen sich dann gerade immer auf die Mengen von Zeilen, die zusammengefasst werden.
- **`having`** wählt Zeilen *nach* der Aggregation aus, **`where`** hingegen Zeilen *vor* der Aggregation.

Zusammenfassung

Unteranfragen als Tabellen

Bei `from ...` kann statt einem Spaltennamen auch stehen

```
(select ...) as name
```

Dies produziert eine temporäre Tabelle namens `name`.

Unteranfragen als Wert

Statt einem Wert kann überall

```
(select ...)
```

stehen, wenn dies genau eine Zeile und Spalte produziert.

Zusammenfassung

Unteranfragen mit Quantoren

Vergleiche der Form

```
... = any (select ...)
```

sind wahr, wenn der Wert auf der linken Seite gleich irgendeiner Zeile in der rechten Tabelle ist.

- Statt `any` ist auch `all` möglich.
- Statt der Gleichheit sind auch andere Vergleiche erlaubt.
- Will man nur testen, ob die Unteranfrage nicht leer ist, so schreibt man einfach nur `exists` vor die Anfrage.