Einführung in Datenbanken

Kapitel 10: Aggregationsfunktionen in SQL

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2018/19

http://www.informatik.uni-halle.de/~brass/db18/

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

Einleitung (1)

- Normalerweise ist die Datenbank sehr groß, so dass man alle Tabellenzeilen einzeln gar nicht in vernünftiger Zeit durchlesen konnte.
- Mit den bisherigen Anfragen werden aus dieser großen Menge von Tupeln eine kleine Anzahl ausgewählt, die die gewünschte Information enthalten.

  Die einzelnen Antwort-Tupel entstehen durch Kombination einer kleinen Anzahl Datenbank-Tupel, ggf. noch Umstrukturierung oder Berechnung abgeleiteter Werte. Bei nichtmonotonen Anfragen kann auch die Nichtexistenz bestimmter Tupel in ein Antworttupel einfließen. Selbst bei monotonen Anfragen kann man aus der Vollständigkeit der Antwortmenge auf die Nichtexistenz weiterer Tupel schließen.

Einleitung (2)

- Mit Aggregationsfunktionen wie Anzahl, Summe oder Durchschnitt fließt nun in jedes Antworttupel eine beliebig große Anzahl von Datenbank-Tupeln ein.
- Dies ist eine andere Art, wie man aus einer großen Menge von Tupeln in der Datenbankbank eine kleine Menge von Antworttupeln generieren kann.

  Klein genug, dass ein Mensch sie in angemessener Zeit lesen und verstehen kann.
- Aggregationen über große Mengen von Tupeln sind besonders typisch für "Data Warehouse" Anwendungen, also z.B. zur Entscheidungsunterstützung für Manager.

  Oft werden die Ergebnisse dann noch graphisch als Diagramm dargestellt.

Inhalt

Aggregationen (1)

- Aggregationsfunktionen sind Funktionen von einer Menge oder Multimenge zu einem einzelnen Wert.

$$\text{E.g.:} \quad min\{41, 57, 19, 23, 27\} = 19$$

- Aggregationsfunktionen fassen eine ganze Menge von Werten zu einem einzelnen Wert zusammen.

  Aggregationsfunktionen nennt man auch "Mengenfunktionen","Grup-penfunktionen" oder "Spaltenfunktionen". Sie haben nicht einen einzelnen Wert als Eingabe, sondern eine ganze Spalte (eine Menge). Die Spalte muss keine Spalte einer gespeicherten Tabelle sein, sie kann auch durch eine Anfrage erstellt werden.

- Aggregationsfunktionen werden oft für statistische Auswertungen verwendet (z.B. Durchschnitt/Avg).

Aggregationen (2)

- SQL-86/92 hat die fünf Aggregationsfunktionen `COUNT`, `SUM`, `AVG`, `MAX`, `MIN`.

  Zusätzliche Aggregationsfunktionen in einigen Systemen:

  Oracle 8i: `CORR` (Korrelation, arbeitet auf einer Menge von Paaren), `COVAR_POP`, `COVAR_SAMP`, lineare Regressionsfunktionen, `STDDEV`, `STDDEV_POP`, `STDEV_SAMP`, `VARIANCE`, `VAR_POP`, `VAR_SAMP`.

  DB2: `CORRELATION`, `COUNT_BIG`, `COVARIANCE`, Regressionsfunktionen, `STDDEV`, `VARIANCE`.

  SQL Server: `VAR`, `VARP`, `STDEV`, `STDEVP`.

  Access: `VAR`, `VARP`, `STDEV`, `STDEVP`, `FIRST`, `LAST`.

  MySQL: `STD`. MySQL unterstützt `DISTINCT` aber nur für `COUNT`.

- Jeder kommutative, assoziative Binäroperator mit neutralem Element kann so erweitert werden, dass er auf Mengen arbeitet. Z.B. ist *sum* die Mengenversion von $+$.

Aggregationen (3)

- Für einige Aggregationsfunktionen sind Duplikate wichtig (z.B. `SUM`), für andere nicht (z.B. `MIN`).

  Z.B. die Summe aller Bestandteile einer Rechnung. Auch wenn zwei Teile das gleiche kosten, müssen trotzdem beide aufsummiert werden.

- In SQL kann man wählen, ob Duplikatelimination
  - vor der Aggregation ausgeführt wird (Eingabe: Menge),
  - oder nicht (Eingabe: Multimenge).

    Eine Multimenge ist eine Menge, in der jedes Element eine Vielfachheit hat, z.B. kann ein Element in einer Multimenge zweimal vorkommen. Im Gegensatz zu einer Liste gibt es keine spezielle Anordnung.

- `SUM(DISTINCT X)` und `AVG(DISTINCT X)`: meist falsch. Einige Studenten verwechseln `SUM` und `COUNT`.

Inhalt

Einfache Aggregationen (1)

- Zunächst werden Aggregationen über alle Ergebniszeilen einer Anfrage erklärt.

  Aggregationen über Gruppen von Zeilen: Siehe nächster Abschnitt.

- Wieviele Studenten gibt es in der Datenbank?

```
SELECT COUNT(*)
FROM   STUDENTEN
```

COUNT(*)
4

- Was ist das beste Ergebnis für Hausaufgabe 1?

```
SELECT MAX(PUNKTE)
FROM   BEWERTUNGEN
WHERE  ATYP = 'H'
AND    ANR = 1
```

MAX(PUNKTE)
10

Einfache Aggregationen (2)

- Wie viele Studierende haben mindestens eine Hausaufgabe abgegeben?

```
SELECT COUNT(DISTINCT SID)
FROM   BEWERTUNGEN
WHERE  ATYP = 'H'
```

COUNT(DISTINCT SID)
3

- Wieviele Punkte hat Studentin 101 insgesamt für Hausaufgaben bekommen?

```
SELECT SUM(PUNKTE) "Gesamtpunkte"
FROM   BEWERTUNGEN
WHERE  SID = 101 AND ATYP = 'H'
```

Gesamtpunkte
18

Einfache Aggregationen (3)

- Wieviel Prozent der Maximalpunktzahl haben die Studenten für HA 1 durchschnittlich bekommen?

```
SELECT AVG((B.PUNKTE/A.MAXPT)*100)
FROM   BEWERTUNGEN B, AUFGABEN A
WHERE  B.ATYP = 'H' AND A.ATYP = 'H'
AND    B.ANR = 1 AND A.ANR = 1
```

- Z.B. Hausaufgabenpunkte von Studentin 101 plus 3 Extrapunkte:

```
SELECT SUM(PUNKTE) + 3 "Gesamte HA-Punkte"
FROM   BEWERTUNGEN
WHERE  SID = 101 AND ATYP = 'H'
```

Einfache Aggregationen (4)

- Man kann auch mehr als eine Aggregation in der SELECT-Liste berechnen, z.B.: Was ist die minimale und maximale Punktzahl für Hausaufgabe 1?

```
SELECT MIN(PUNKTE), MAX(PUNKTE)
FROM   BEWERTUNGEN
WHERE  ATYP = 'H' AND ANR = 1
```

- Die Aggregationen können sich auf verschiedene Spalten beziehen:

```
SELECT COUNT(DISTINCT THEMA), AVG(MAXPT)
FROM   AUFGABEN A
```

Aggregationsanfragen

- Es gibt drei Typen von Anfragen in SQL:
  - Anfragen ohne Aggregationsfunktionen und ohne `GROUP BY` und `HAVING`: siehe oben.
  - Anfragen mit Aggregationsfunktionen, aber ohne `GROUP BY`: Ergebnis ist immer genau eine Zeile.

    Diese wurden oben "einfache Aggregationen" genannt. Die Aggregationsfunktion kann dann nur unter `SELECT` auftauchen (außerdem sind bei jedem Typ Aggregationen in Unteranfragen möglich).
  - Anfragen mit `GROUP BY`.
- Jeder Typ hat verschiedene Syntaxrestriktionen und wird auf verschiedene Weisen ausgewertet.

Auswertung (1)

- Zunächst wird die `FROM`-Klausel ausgewertet.

  Theoretisch werden alle möglichen Tupelkombinationen der unter `FROM` genannten Tabellen konstruiert (mögliche Variablenbelegungen, $\times$).

- Als zweites wird die `WHERE`-Klausel ausgewertet.

  Nur die Tupelkombinationen (Variablenbelegungen), die die Bedingung erfüllen, werden weiter betrachtet (Selektion, Filter). Reale Systeme kombinieren beide Schritte für eine effizientere Auswertung.

- Gibt es keine Aggregation, `GROUP BY`, und `HAVING`, so wird anschließend die `SELECT`-Klausel ausgewertet, indem man die Werte der Terme in der `SELECT`-Liste für die restlichen Tupelkombinationen ausgibt.

Auswertung (2)

- Beim zweiten Anfragetyp (`SELECT` enthält Aggregationsterm, aber es gibt aber kein `GROUP BY`) wird nur eine einzelne Ausgabezeile berechnet.
- Anstatt die Werte der unter `SELECT` genannten Spalten auszugeben, werden sie in eine (Multi-)Menge eingefügt, die dann als Eingabe für die Aggregationsfunktion dient.

  Enthält die `SELECT`-Liste mehrere Aggregationen, müssen mehrere solcher Mengen verwaltet werden. Ist kein `DISTINCT` angegeben (Multimenge), so können die aggregierten Werte inkrementell ohne explizites Speichern der temporären Menge berechnet werden (vgl. nächste Folie).

Auswertung (3)

- Beispiel für inkrementelle Berechnung:

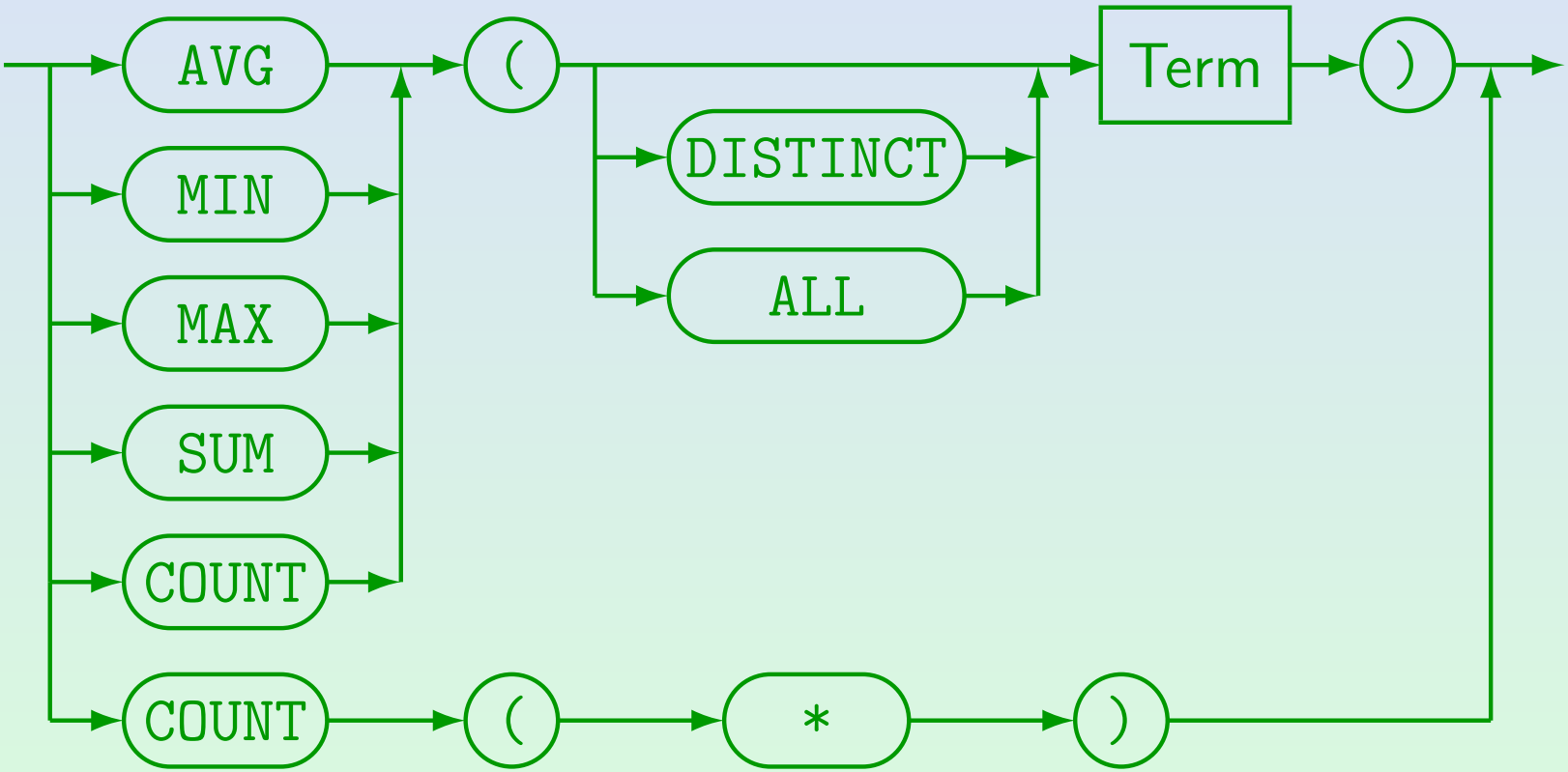
```
SELECT SUM(MAXPT), COUNT(*)
FROM   AUFGABEN A
WHERE  ATYP = 'H'
```

- Dies wird so ausgewertet:

```
ausgabe1 = 0; ausgabe2 = 0;
foreach row A in AUFGABEN do
        if A.ATYP = 'H' then begin
                ausgabe1 = ausgabe1 + A.MAXPT;
                ausgabe2 = ausgabe2 + 1;
        end;
print ausgabe1, ausgabe2;
```

Syntax

Aggregationsterm:



- MySQL unterstützt `DISTINCT` nur für `COUNT` (und versteht `ALL` nicht).

Syntax / Restriktionen (1)

- `SUM` und `AVG` müssen numerische Argumente haben. `COUNT`, `MIN`, und `MAX` akzeptieren jeden Datentyp.
- Aggregationen können nicht verschachtelt werden, z.B. ist folgendes unzulässig:

```
AVG(COUNT(*))
```

Falsch!

`COUNT` liefert einen einzelnen Wert. Damit ist die Anwendung einer zweiten Aggregation sinnlos.

Es ist möglich Aggregationen zunächst auf Gruppen von Zeilen anzuwenden, und dann das Ergebnis als Eingabe für eine andere Aggregation zu verwenden. Z.B. Was ist die durchschnittliche Gesamtpunktzahl, die Studenten für ihre Hausaufgaben erhalten haben? Dazu verwendet man `GROUP BY` und Unteranfragen (siehe unten).

Syntax / Restriktionen (2)

- Die WHERE-Bedingung kann nicht direkt Aggreationsterme enthalten, nur in Unteranfragen.

  Die WHERE-Bedingung wird vor der Berechnung der Aggregation ausgewertet (sie legt fest, welche Tupel in die Aggregation eingehen). Bedingungen an Aggregationen können unter HAVING festgelegt werden.

```
WHERE COUNT(*) > 1
```
Falsch!

- Bei einfachen Aggregationen können keine normalen Attribute in der SELECT-Liste auftauchen.

  Die Attribute haben ja im allgemeinen für verschiedene Variablenbelegungen verschiedene Werte. Dann wäre nicht klar, welcher Wert in der einzelnen Ausgabezeile erscheinen soll. Bei Bedarf verwende man GROUP BY (s.u.).

```
SELECT ATYP, ANR, AVG(PUNKTE)
FROM   BEWERTUNGEN
```
Falsch!

Syntax / Restriktionen (3)

- Jeder Aggregationsoperator benötigt ein Argument (welches die Eingabewerte spezifiziert).

```
SELECT SID
FROM   BEWERTUNGEN
WHERE  ATYP = 'H' AND ANR = 1
AND    PUNKTE = MAX
```

Falsch! Falsch!

Aggregationen sind auch unter `WHERE` nicht erlaubt.

Jedenfalls nicht direkt. Geschachtelt in Unteranfragen schon.

- Es wird eine Unteranfrage benötigt, um den Studenten mit dem besten Ergebnis für Hausaufgabe 1 zu finden (siehe unten).

Nullwerte in Aggregationen

- Normalerweise werden Nullwerte herausgefiltert, bevor die Aggregationsfunktion angewendet wird.
- Nur `COUNT(*)` beinhaltet Nullwerte (da es Zeilen und keine Attributwerte zählt).
- Der einzige Unterschied zwischen `COUNT(EMAIL)` und `COUNT(*)` ist, dass das erste nur die Zeilen zählt, für die `EMAIL` nicht Null ist, und das zweite alle Zeilen zählt.

  Ansonsten ist der Attributwert nicht wichtig für `COUNT`, und man sollte besser `COUNT(*)` verwenden, um dies deutlich zu machen. (Es wäre verwirrend, ein Attribut anzugeben, wenn das tatsächlich irrelevant ist.) Wenn aber Duplikate eliminiert werden, wie z.B. in `COUNT(DISTINCT ATYP)`, ist das Attribut offensichtlich wichtig.

Leere Aggregationen

- Ist die Eingabemenge leer, ergeben die meisten Aggregationen einen Nullwert, nur `COUNT` ergibt 0.

  Dies widerspricht zumindest für `SUM` der Intuition. Man würde erwarten, dass die Summe über die leere Menge `0` ist, aber in SQL erhält man `NULL`. (Möglicher Grund: Wenn alle Nullwerte vor der Aggregation eliminiert werden, bekommt `SUM` auch eine leere Menge als Eingabe, wenn es tatsächlich Nullwerte gab.)

- Da es vorkommen kann, dass keine Zeile die `WHERE`-Bedingung erfüllt, müssen Programme mit dem resultierenden Nullwert umgehen können.

- Mögliche Lösung:

  ```
  COALESCE(SUM(PUNKTE), 0)
  ```

  ersetzt ggf. den Nullwert durch die Zahl 0.

Inhalt

GROUP BY (1)

- Die obigen SQL-Konstrukte können nur eine einzelne aggregierte Ausgabezeile erzeugen.
- Mit der "GROUP BY" Klausel kann man über Gruppen von Tupeln aggregieren (anstatt über alle Tupel).
- Berechnen Sie die durchschnittliche Punktzahl für jede Hausaufgabe:

```
SELECT   ANR, AVG(PUNKTE)
FROM     BEWERTUNGEN
WHERE    ATYP = 'H'
GROUP BY ANR
```

ANR	AVG(PUNKTE)
1	8
2	8.5

GROUP BY (2)

- Das Zwischenergebnis nach Auswertung von `FROM` und `WHERE` wird jetzt in Gruppen aufgespalten, wobei die Tupel einer Gruppe jeweils den gleichen Wert in den `GROUP BY`-Spalten haben.

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

- Man erhält dann eine Ausgabezeile pro Gruppe.

GROUP BY (3)

- Diese Konstruktion kann niemals zu leeren Gruppen führen. Somit ist es unmöglich, dass ein `COUNT(*)` den Wert `0` ergibt.

  Der Wert 0 kann mit `COUNT(A)` entstehen, wenn das Attribut `A` Null ist. Wenn eine Anfrage Gruppen mit count 0 ergeben muss, wird vermutlich ein äußerer Verbund benötigt (siehe unten).

- Einfache Aggregationen (ohne `GROUP BY`) ergeben immer genau eine Ausgabezeile: Wenn die Eingabemenge leer ist, erhält man `COUNT(*) = 0`.

  Dagegen kann eine `GROUP BY`-Anfrage keine, eine oder mehrere Ausgabezeilen liefern.

GROUP BY (4)

- Da `GROUP BY`-Attribute einen eindeutigen Wert für jede Gruppe haben, können sie ausgegeben werden.

  Andere Attribute können unter `SELECT` nur in Aggregationen stehen.

- Z.B. folgendes ist unzulässig:

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

Falsch!

  `A.THEMA` ist kein `GROUP BY`-Attribut, deshalb darf es nicht unter `SELECT` außerhalb von Aggregationsfunktionen verwendet werden. Die SQL-Regel ist rein syntaktisch: Weil `(ATYP, ANR)` Schlüssel von `AUFGABEN` ist, wäre `THEMA` durchaus eindeutig innerhalb der Gruppen.

GROUP BY (5)

- Man kann sich die Situation auch so vorstellen, dass temporär eine geschachtelte Tabelle gebildet wird:

A.ANR	A.THEMA	B.PUNKTE
1	ER	10
	ER	9
	ER	5
2	SQL	8
	SQL	9

Obwohl auch `A.THEMA` in den Gruppen eindeutig wäre, gibt es dafür mehrere Zeilen (mit identischem Wert), während es für das `GROUP BY` Attribut nur einen Wert pro Ausgabezeile gibt.

GROUP BY (6)

- Also muss man nach `A.ANR` und `A.THEMA` gruppieren:

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

A.ANR	A.THEMA	AVG(B.PUNKTE)
1	Rel. Algeb.	8
2	SQL	8.5

- Das Hinzufügen von `A.THEMA` zu `GROUP BY` ändert die Gruppen nicht, aber man kann es nun ausgeben.

GROUP BY (7)

- 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 Anfage, 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
```

GROUP BY (8)

- GROUP BY wird vor SELECT ausgewertet. Deshalb kann man sich nicht auf neue Attributnamen beziehen:

```
SELECT   FLOOR((PUNKTE/MAXPT)*100+0.5) PROZENTE,
         COUNT(*)
FROM     AUFGABEN A, BEWERTUNGEN B
WHERE    A.ATYP = B.ATYP AND A.ANR = B.ANR
GROUP BY PROZENTE
```

Falsch!

- Oracle, SQL Server, DB2, MySQL und Access unterstützen GROUP BY mit beliebigen Termen. Der SQL-92 Standard erlaubt GROUP BY nur mit Spaltennamen.

  D.h. GROUP BY FLOOR(...) funktioniert in diesen Systemen.

  Portabele Alternative: Unteranfrage unter FROM oder Verwendung einer Sicht.

GROUP BY (9)

- Die Reihenfolge der Attribute unter "`GROUP BY`" ist nicht wichtig.

  `GROUP BY A, B` bedeutet, dass zwei Tupel t, u in die gleiche Gruppe gehören, falls $t.\mathtt{A} = u.\mathtt{A}$ und $t.\mathtt{B} = u.\mathtt{B}$.

  `GROUP BY B, A` bedeutet, dass zwei Tupel t, u in die gleiche Gruppe gehören, falls $t.\mathtt{B} = u.\mathtt{B}$ und $t.\mathtt{A} = u.\mathtt{A}$.

- Es macht keinen Sinn, nach einem Schlüssel zu gruppieren (bei nur einer Tabelle unter `FROM`):
  Dann besteht jede Gruppe nur aus einer Zeile.

- Ebenso ist `GROUP BY` nicht sinnvoll, wenn es nur eine einzige Gruppe liefern kann.

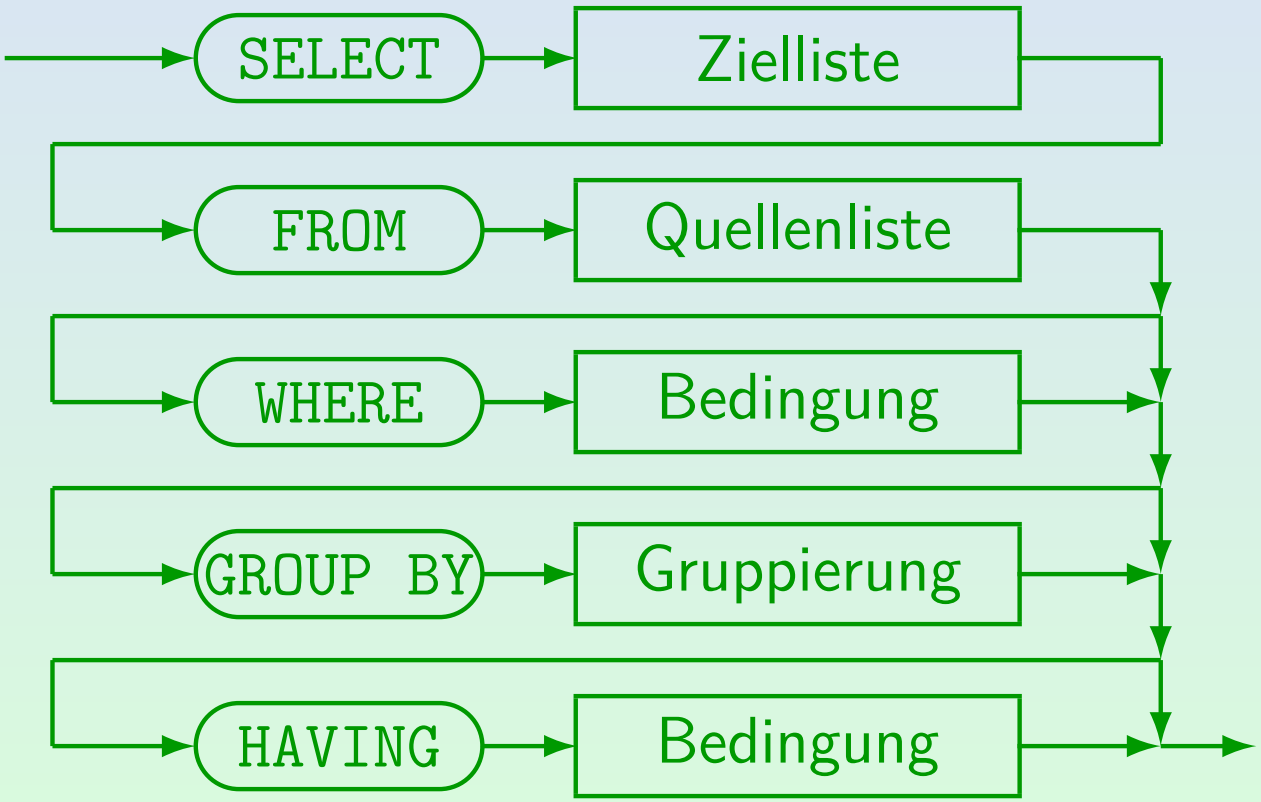
GROUP BY (10)

Warnung:

- "`GROUP BY`" wird öfters mit "`ORDER BY`" verwechselt:
  - `GROUP BY` ist wichtig für das Anfrageergebnis.
  - `ORDER BY` ist nur kosmetisch (schönere Ausgabe).
- "`GROUP BY`" sortiert normalerweise intern die Tupel (so dass Tupel mit gleichem Wert benachbart sind).
- Aber dann führt `GROUP BY` die Gruppierung durch.
- Man kann sich auch nicht darauf verlassen, dass "`GROUP BY`" als Nebeneffekt sortiert: Eventuell kann das DBMS es anders effizienter auswerten.

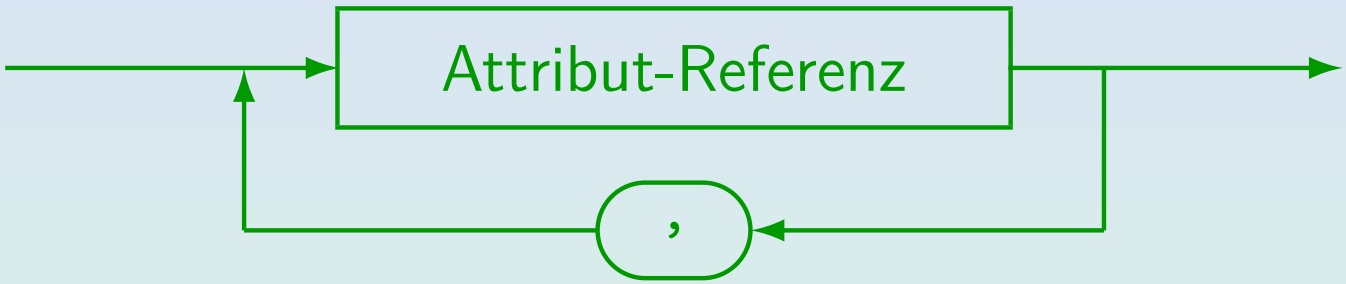
Syntax (1)

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.

Inhalt

HAVING (1)

- Aggregationen sind unter `WHERE` verboten.
- Manchmal benötigt man Aggregationen zur Filterung von Ausgabezeilen.

  Und nicht nur zur Berechnung von Ausgabewerten.
- Deshalb gibt es die `HAVING`-Klausel: Hier kann man eine Bedingung mit Aggregationsfunktionen angeben. So kann man ganze Gruppen eliminieren.
- Wie bei `SELECT` dürfen auch unter `HAVING` außerhalb von Aggregationen nur `GROUP BY`-Attribute stehen.

HAVING (2)

- Wer hat mindestens 18 Hausaufgabenpunkte?

```
SELECT   VORNAME, NACHNAME
FROM     STUDENTEN S, BEWERTUNGEN B
WHERE    S.SID = B.SID AND B.ATYP = 'H'
GROUP BY S.SID, VORNAME, NACHNAME
HAVING   SUM(PUNKTE) >= 18
```

VORNAME	NACHNAME
Lisa	Weiss
Michael	Grau

- Die `WHERE`-Bedingung bezieht sich auf jeweils eine Tupelkombination (Variablenbelegung), die `HAVING`-Bedingung dagegen auf ganze Gruppen.

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.

Syntax: 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.

  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.
- `HAVING` ohne `GROUP BY` ist legal, aber ungewöhnlich. Die Anfrage kann nur 0 oder 1 Ausgabezeile haben.

WHERE vs. HAVING

- Meist legen die Syntaxregeln schon fest, ob eine Bedingung unter `WHERE` oder unter `HAVING` gehört.

  Nur wenn eine Bedingung ausschließlich `GROUP BY`-Attribute, aber keine Aggregationen enthält, wäre sie in beiden Klauseln erlaubt.

- Wenn beides möglich ist, ist es wesentlich effizienter es unter `WHERE` zu stecken. Z.B. diese Anfrage ist zulässig, aber langsam und braucht viel Speicher:

```
SELECT   VORNAME, NACHNAME
FROM     STUDENTEN S, BEWERTUNGEN B
GROUP BY S.SID, B.SID, VORNAME, NACHNAME
HAVING   S.SID = B.SID AND SUM(PUNKTE) >= 18
```

Inhalt

Aggregationsunteranfragen (1)

- Wer hat das beste Ergebnis für Hausaufgabe 1?

```
SELECT S.VORNAME, S.NACHNAME, B.PUNKTE
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID=B.SID AND B.ATYP='H' AND B.ANR=1
AND    B.PUNKTE = (SELECT MAX(PUNKTE)
                   FROM   BEWERTUNGEN
                   WHERE  ATYP='H' AND ANR=1)
```

- Bei der Unteranfrage ist garantiert, dass man genau eine Ergebniszeile erhält (Aggregationsanfrage ohne `GROUP BY`): Daher sind `ANY`/`ALL` nicht erforderlich.

Aggregationsunteranfragen (2)

- In Systemen, in denen Ein-Wert-Unteranfragen wie ein Term verwendet werden können, sind Unteranfragen auch unter SELECT möglich.

  Dies funktioniert in SQL-92, DB2, Oracle 9i, SQL Server und Access.

  SQL-86 und z.B. Oracle 8.0 unterstützten es nicht.

- Das kann GROUP BY ersetzen. Z.B. Geben Sie für jeden Studenten die Summe der HA-Punkte aus:

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

Geschachtelte Aggregation (1)

- Verschachtelte Aggregationen (z.B. Durchschnitt über Summen) benötigen Unteranfragen unter FROM.
- Was ist die durchschnittliche Anzahl HA-Punkte?

```
SELECT AVG(X.HA_PKT)
FROM   (SELECT   SID, SUM(PUNKTE) AS HA_PKT
        FROM     BEWERTUNGEN
        WHERE    ATYP = 'H'
        GROUP BY SID) X
```

X	
SID	HA_PKT
101	18
102	18
103	5

AVG(X.HA_PKT)
13.67

Bemerkung:
Es werden hier nur Studierende gezählt, die mindestens eine Aufgabe gelöst haben.

Geschachtelte Aggregation (2)

- Oracle unterstützt auch folgende Syntax für verschachtelte Aggregationen:

```
SELECT   AVG(SUM(PUNKTE))
FROM     BEWERTUNGEN
WHERE    ATYP = 'H'
GROUP BY SID
```

Nur Oracle!

Das ist aber nicht Standard (wird nicht unterstützt in SQL-92, DB2, SQL Server, Access).

Da es wesentlich kürzer, als die äquivalente Standard-Anfrage ist, kann es bei Ad-hoc-Anfragen bequemer sein. In Anwendungsprogrammen sollte man aber keine unnötigen Übertragbarkeitsprobleme schaffen.

Aggregationen über mehrere Mengen (1)

- Durch Unteranfragen unter FROM kann man in einer Anfrage über verschiedene Mengen aggregieren:

```
SELECT VORNAME, NACHNAME, H.PT AS HA, Z.PT AS ZK
FROM   STUDENTEN S,
       (SELECT   SID, SUM(PUNKTE) AS PT
        FROM     BEWERTUNGEN
        WHERE    ATYP = 'H'
        GROUP BY SID) H,
       (SELECT   SID, SUM(PUNKTE) AS PT
        FROM     BEWERTUNGEN
        WHERE    ATYP = 'Z'
        GROUP BY SID) Z
WHERE  S.SID = H.SID AND S.SID = Z.SID
```

Aggregationen über mehrere Mengen (2)

- Aggregation über verschiedenen Mengen ist auch mit bedingten Ausdrücken möglich:

```
SELECT VORNAME, NACHNAME,
       SUM(CASE B.ATYP WHEN 'H' THEN B.PUNKTE
                                ELSE 0 END) HA,
       SUM(CASE B.ATYP WHEN 'Z' THEN B.PUNKTE
                                ELSE 0 END) ZK,
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
GROUP  BY S.SID, VORNAME, NACHNAME
```

Durch den bedingten Ausdruck fließen in die erste Summe nur die Punkte für Hausaufgaben ein, Punkte für andere Aufgaben werden durch 0 ersetzt.

In Oracle kann der bedingte Ausdruck kürzer geschrieben werden: `DECODE(B.ATYP, 'H', B.PUNKTE, 0)`.

Aggregationen über mehrere Mengen (3)

- Aggregation über verschiedenen Mengen geht auch mit Unteranfragen unter SELECT:

```
SELECT VORNAME, NACHNAME,
       (SELECT SUM(H.PUNKTE) FROM BEWERTUNGEN H
        WHERE H.SID = S.SID AND H.ATYP = 'H')
        AS HAUSAUFGABEN,
       (SELECT SUM(Z.PUNKTE) FROM BEWERTUNGEN
        WHERE Z.SID = S.SID AND Z.ATYP = 'Z')
        AS ZWISCHENKLAUSUR
FROM   STUDENTEN S
```

Die drei Anfragen unterscheiden sich in ihrem Umgang mit Studenten ohne Hausaufgaben, ohne Zwischenklausur-Teilnahme, bzw. ganz ohne Bewertungen (Aufgabe!).

Aggregationen maximieren (1)

- Wer hat das beste Ergebnis in den Hausaufgaben (maximale Summe der Hausaufgabenpunkte)?

```
SELECT   VORNAME, NACHNAME, SUM(PUNKTE) AS SUMME
FROM     STUDENTEN S, BEWERTUNGEN B
WHERE    S.SID = B.SID AND B.ATYP = 'H'
GROUP BY S.SID, VORNAME, NACHNAME
HAVING   SUM(PUNKTE) >= ALL(SELECT   SUM(PUNKTE)
                            FROM     BEWERTUNGEN
                            WHERE    ATYP = 'H'
                            GROUP BY SID)
```

- Alternative Lösung (mit Sicht): siehe nächste Folie.

Aggregationen maximieren (2)

- Gesamtpunktzahl der HA für jeden Studenten:

```
CREATE VIEW HA_SUMMEN AS
   SELECT   SID, SUM(PUNKTE) AS SUMME
   FROM     BEWERTUNGEN
   WHERE    ATYP = 'H'
   GROUP BY SID
```

- Dann kann man dies wie folgt verwenden:

```
SELECT S.VORNAME, S.NACHNAME, H.SUMME
FROM   STUDENTEN S, HA_SUMMEN H
WHERE  S.SID = H.SID
AND    H.SUMME = (SELECT MAX(SUMME)
                  FROM   HA_SUMMEN)
```

Aufgabe: Mögliche Fehler (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?

Aufgabe: Mögliche Fehler (2)

- 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
```

Aufgabe: Mögliche Fehler (3)

- 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
```