Teil 6: SQL III

Literatur:

- Elmasri/Navathe:Fundamentals of Database Systems, 3. Auflage, 1999. Chap. 8, “SQL — The Relational Database Standard” (Sect. 8.2, 8.3.3, part of 8.3.4.)
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3. Auflage. McGraw-Hill, 1999: Chapter 4: “SQL”.
- Kemper/Eickler: Datenbanksysteme , Kap. 4, Oldenbourg, 1997.
- Lipeck: Skript zur Vorlesung Datenbanksysteme, Univ. Hannover, 1996.
- Heuer/Saake: Datenbanken, Konzepte und Sprachen, Thomson, 1995.
- Date/Darwen: A Guide to the SQL Standard, 4. Auflage, Addison-Wesley, 1997.
- Date: A Guide to the SQL Standard, 1. Auflage, Addison-Wesley, 1987.
- van der Lans: SQL, Der ISO-Standard, Hanser, 1990.
- Sunderraman: Oracle Programming, A Primer. Addison-Wesley, 1999.
- Oracle 8i SQL Reference, Release 2 (8.1.6), Dec. 1999, Part No. A76989-01.
- Chamberlin: A Complete Guide to DB2 Universal Database. Morgan Kaufmann, 1998.
- Microsoft SQL Server Books Online: Accessing and Changing Data.
- Microsoft Jet Database Engine Programmer’s Guide, 2. Auflage (Part of MSDN Library Visual Studio 6.0).
- DuBois: MySQL. New Riders Publishing, 2000, ISBN 0-7357-0921-1, 756 pages.

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Fortgeschrittene Anfragen in SQL schreiben, die Aggregationen, Unteranfragen und UNION enthalten.
- Die Teile einer SQL-Anfrage aufzählen/erklären.

  SELECT, FROM, WHERE, GROUP BY, HAVING, ..., ORDER BY
- Verbunde in SQL-92 erklären.
- Eine gegebene Anfrage auf syntaktische Korrektheit prüfen.
- Die Portabilität von Konstrukten beurteilen.

Inhalt

1. Aggregationen I: Aggregationsfunktionen
2. Aggregationen II: GROUP BY, HAVING
3. UNION, Bedingte Ausdrücke
4. Sortieren der Ausgabe: ORDER BY
5. SQL-92 Verbunde, Äußerer Verbund in Oracle

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

Aggregationen (1)

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

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

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

  Aggregationsfunktionen nennt man auch "Mengenfunktionen", "Gruppenfunktionen" 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`.

  Jede kommutative, assoziative Verknüpfung mit neutralem Element kann so erweitert werden, daß sie 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 Duplikatelimination fordern (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`.

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:

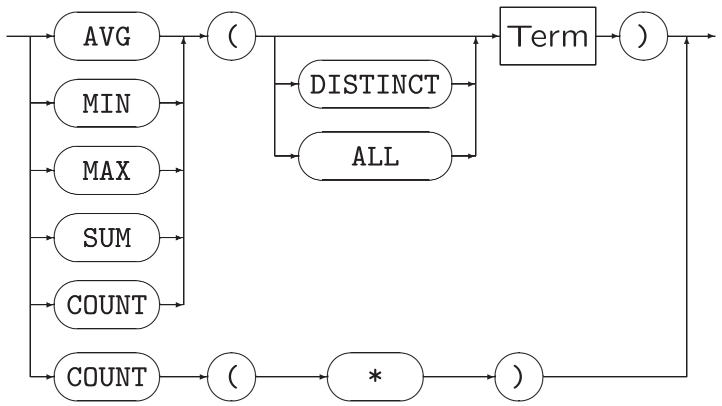
```
SELECT SUM(MAXPT), COUNT(*)
FROM   AUFGABEN A
WHERE  ATYP = 'H'
```

- Dies könnte so ausgewertet werden:

```
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.

  Da kein Attribut außerhalb der Aggregation nur einen einzelnen Ausgabewert hat. Aber man beachte GROUP BY.

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


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

Nullwerte in Aggregationen

- Meist 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, wo EMAIL nicht Null ist, und das zweite alle Zeilen zählt.

  Andererseits ist der Attributwert nicht wichtig für COUNT, und man sollte wahrscheinlich COUNT(*) verwenden. Wenn natürlich Attribute, wie in COUNT(DISTINCT ATYP) eliminiert werden, dann ist der Attributwert 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. (Ein Grund dafür könnte sein, daß die SUM-Aggregationsfunktion keinen Unterschied entdeckt, zwischen der leeren Eingabemenge, weil es keine qualifizierenden Tupel gibt, und der leeren Eingabemenge, weil alle qualifizierenden Tupel Nullwerte in diesem Argument haben.)

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

  Alternativ: Verwende z.B. NVL(SUM(PUNKTE),0) in Oracle, um den Nullwert zu ersetzen.

Inhalt

1. Aggregationen I: Aggregationsfunktionen
2. Aggregationen II: GROUP BY, HAVING
3. UNION, Bedingte Ausdrücke
4. Sortieren der Ausgabe: ORDER BY
5. SQL-92 Verbunde, Äußerer Verbund in Oracle

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)   Falsch!
FROM   AUFGABEN A, BEWERTUNGEN B
WHERE  A.ATYP='H' AND B.ATYP='H' AND A.ANR=B.ANR
GROUP BY A.ANR
```

  `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)

- 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 (6)

- Übung: 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 (7)

- `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 unterstüzen `GROUP BY` mit beliebigen Termen. SQL92 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 (8)

- 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 (9)

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 effizienter anders auswerten.

DISTINCT vs. GROUP BY

- Duplikate sollten mit `DISTINCT` eliminiert werden, obwohl es auch mit `GROUP BY` funktioniert:

```
SELECT   ATYP, ANR
FROM     BEWERTUNGEN
GROUP BY ATYP, ANR
```
Schlechter Stil!

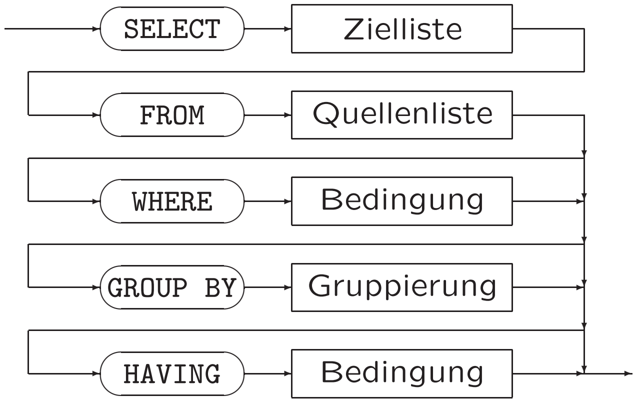
  Dies teilt die Tabelle in Gruppen von Tupeln auf: jede Gruppe enthält Tupel, die in den `GROUP BY`-Attributen `ATYP`, `ANR` übereinstimmen. Für jede Gruppe wird nur ein Tupel ausgegeben. Normalerweise verwendet, um Aggregationsfunktionen (`SUM`, `COUNT`) für jede Gruppe auszuwerten.

- Ich sehe dies als Missbrauch von `GROUP BY` an.

  `GROUP BY` ist jedoch flexibler als `DISTINCT`, wenn man nur manche Duplikate eliminieren möchte. Alte Versionen von MySQL unterstützten kein `DISTINCT`. Dann musste man `GROUP BY` verwenden.

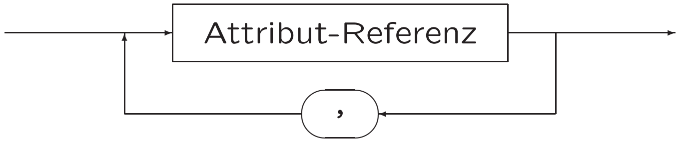
Syntax (1)

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.

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
```


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 SQL92, 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.

Agg. 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
```

Agg. über mehrere Mengen (2)

- Aggregation über verschiedenen Mengen ist auch mit bedingten Ausdrücken möglich, z.B. in Oracle:

```
SELECT VORNAME, NACHNAME,
       SUM(DECODE(B.ATYP, 'H', B.PUNKTE, 0)) HA
       SUM(DECODE(B.ATYP, 'Z', B.PUNKTE, 0)) ZK
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
```

- Z.B. liefert der bedingte Ausdruck

```
DECODE(B.ATYP, 'H', B.PUNKTE, 0)
```

  B.PUNKTE, falls B.ATYP = 'H', und 0 sonst.

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)
```


Übung: 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?

Übung: Mögliche Fehler (2)

- Man beachte, daß Unteranfragen nicht direkt in die Aggregations-Funktion geschrieben werden können:

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

- Das Argument von Funktionen muß ein Term sein.

  Aufgabe: Unter bestimmten Voraussetzungen können Unteranfragen doch als Term verwendet werden, warum macht das aber bei Aggregationsfunktionen keinen Sinn?

Übung: Mögliche Fehler (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
```


Übung: Mögliche Fehler (4)

- 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
```


Inhalt

1. Aggregationen I: Aggregationsfunktionen
2. Aggregationen II: GROUP BY, HAVING
3. UNION, Bedingte Ausdrücke
4. Sortieren der Ausgabe: ORDER BY
5. SQL-92 Verbunde, Äußerer Verbund in Oracle

UNION (1)

- In SQL kann man die Ergebnisse von zwei Anfragen mit UNION (Vereinigung von Mengen) verknüpfen.

  $R \cup S$ ist die Menge aller Tupel, die in R, in S, oder in beiden sind.

- UNION wird benötigt, da es sonst keine Möglichkeit gibt, eine Ergebnisspalte mit Werten aus mehreren Tabellenspalten (Eingabespalten) zu konstruieren.

  Dies wird z.B. benötigt, wenn Subklassen durch verschiedene Tabellen representiert werden. Z.B. könnte es eine Tabelle STUDENTEN und eine andere Tabelle GASTHÖRER geben.

- UNION wird auch für Fallunterscheidungen verwendet (um **if ... then ... else ...** darzustellen).

UNION (2)

- Die Unteranfragen, die durch UNION verbunden werden, müssen Tabellen mit der gleichen Anzahl von Spalten liefern. Die Datentypen der korrespondierenden Spalten müssen kompatibel sein.

  Die Attributnamen müssen nicht übereinstimmen. Oracle und SQL Server verwenden im Ergebnis die Attributnamen der ersten Unteranfrage. DB2 verwendet ggf. künstliche Spaltennamen (1, 2, ...).

- SQL unterscheidet zwischen
  - ◇ UNION: $\cup$ mit Duplikatelimination und
  - ◇ UNION ALL: Konkatenation (erhält Duplikate).

  Duplikatelimination kostet Laufzeit.

UNION (3)

- Geben Sie für jeden Studenten die Gesamtpunktzahl für Hausaufgaben aus (auch für Studierende, die keine Aufgaben abgegeben haben).

```
SELECT   VORNAME, NACHNAME, SUM(PUNKTE) AS SUMME
FROM     STUDENTEN S, BEWERTUNGEN B
WHERE    S.SID = B.SID AND ATYP = 'H'
GROUP BY S.SID, VORNAME, NACHNAME
UNION ALL
SELECT   VORNAME, NACHNAME, 0 AS SUMME
FROM     STUDENTEN S
WHERE    S.SID NOT IN (SELECT SID
                       FROM   BEWERTUNGEN
                       WHERE  ATYP = 'H')
```


UNION (4)

- Erstellen Sie Noten für die Studenten basierend auf Hausaufgabe 1:

```
SELECT S.SID, S.VORNAME, S.NACHNAME, 1 NOTE
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID=B.SID AND B.ATYP='H' AND B.ANR=1
AND    B.PUNKTE >= 9
    UNION ALL
SELECT S.SID, S.VORNAME, S.NACHNAME, 2 NOTE
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID=B.SID AND B.ATYP='H' AND B.ANR=1
AND    B.PUNKTE >= 7 AND B.PUNKTE < 9
    UNION ALL
...
```

Andere Mengenop. in SQL

- SQL-86 enthielt nur `UNION [ALL]`.
- Der SQL-92 Standard enthält zusätzlich `EXCEPT` (Mengendifferenz, $-$) und `INTERSECT` ($\cap$).

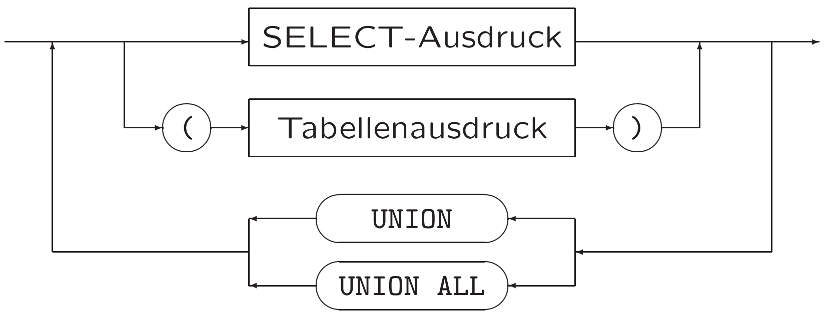
  SQL-86, SQL Server und Access unterstützen nur `UNION [ALL]`. MySQL unterstützt keinen der Operatoren. DB2 bietet alle SQL-92 Mengenoperatoren. In Oracle 8.0, wird `MINUS` statt `EXCEPT` für $-$ verwendet. Für `MINUS` und `INTERSECT` wird `ALL` in Oracle nicht unterstützt.

- Die Operationen `EXCEPT` und `INTERSECT` erweitern die Ausdruckskraft von SQL nicht.

  Anfragen, die `EXCEPT/MINUS` und `INTERSECT` enthalten, können in äquivalente SQL-Anfragen ohne diese Konstrukte umgeformt werden. Anfragen die `UNION` enthalten, können dies im Allgemeinen nicht. Damit ist nur `UNION` wirklich wichtig.

UNION: Syntax

Tabellenausdruck:



- MySQL unterstützt Union nicht. SQL-86 enthält `UNION` und `UNION ALL`.
- SQL-92 und DB2 unterstützen auch `INTERSECT`, `INTERSECT ALL`, `EXCEPT`, und `EXCEPT ALL`. Oracle 8 unterstützt `UNION`, `UNION ALL`, `INTERSECT` und `MINUS`.
- In Access kann man Klammern nicht um eine ganze Anfrage setzen.

Vereinigung vs. Verbund

Übung:

- Zwei Alternativen zur Repräsentation der Bewertungen für Hausaufgaben und Klausuren sind:

Bewertungen_1

STUDENT	H	Z	E
Jim Ford	95	60	75
Ann Lloyd	80	90	95

Bewertungen_2

STUDENT	ATYP	PZT
Jim Ford	H	95
Jim Ford	Z	60
Jim Ford	E	75
Ann Lloyd	H	80
Ann Lloyd	Z	90
Ann Lloyd	E	95

- Übersetzen Sie in beiden Richtungen mittels SQL.

Bedingte Ausdrücke (1)

- Während `UNION` eine portabele Lösung für Fallunterscheidungen ist, kann man manchmal auch einen (effizienteren) bedingten Ausdruck verwenden.

  Bedingte Ausdrücke sind für jedes DBMS verschieden.

- Z.B. hat Oracle Ausdrücke der Form:

  `DECODE(`X, X_1, Y_1, X_2, Y_2, ..., Z`)`

- Dies wird ausgewertet, indem das DBMS zunächst X mit X_1, dann mit X_2, usw. vergleicht. Ist X_i der erste Wert mit $X = X_i$, dann wird Y_i zurückgegeben. Wenn kein X_i passt, wird Z zurückgegeben.

Bedingte Ausdrücke (2)

- Z.B.: Ausgabe der Ergebnisse von Lisa Weiss, dabei Aufgabentyp ausgeschrieben (Oracle Version):

```
SELECT   DECODE(ATYP, 'H', 'Hausaufgabe',
                      'Z', 'Zwischenklausur',
                      'E', 'Endklausur',
                           'Unbekannte Kat.'),
         ANR, PUNKTE
FROM     STUDENTEN S, BEWERTUNGEN B
WHERE    S.SID = B.SID
AND      VORNAME = 'Lisa' AND NACHNAME = 'Weiss'
ORDER BY DECODE(ATYP, 'H', 1, 'Z', 2, 'E', 3, 4)
```


Bedingte Ausdrücke (3)

- Im SQL-Standard (und z.B. in DB2, SQL Server, MySQL) schreibt man dies wie folgt:

```
SELECT CASE WHEN ATYP='H' THEN 'Hausaufgabe'
            WHEN ATYP='Z' THEN 'Zwischenklausur'
            WHEN ATYP='E' THEN 'Endklausur'
            ELSE 'Unbekannte Kat.' END,
       ANR, PUNKTE
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    VORNAME = 'Lisa' AND NACHNAME = 'Weiss'
```

- Oracle 8i (nicht 8.0) erlaubt eine ähnliche Syntax, mit einem Komma zwischen den WHEN-Klauseln.

Bedingte Ausdrücke (4)

- Der SQL-92 Standard (und z.B. DB2, SQL Server, MySQL), nicht Oracle 8i, erlauben auch folgende Abkürzung, die ähnlich zu Oracle's DECODE ist:

```
SELECT CASE ATYP WHEN 'H' THEN 'Hausaufgabe'
                 WHEN 'Z' THEN 'Zwischenklausur'
                 WHEN 'E' THEN 'Endklausur'
                 ELSE 'Unbekannte Kat.' END,
       ANR, PUNKTE
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    VORNAME = 'Lisa' AND NACHNAME = 'Weiss'
```

- In Access: `IIf(Bedingung, FallsJa, FallsNein)`.

Bedingte Ausdrücke (5)

- Bedingte Ausdrücke werden z.B. verwendet, um Null-Werte zu ersetzen.
- In Oracle ist NVL(X, Y) äquivalent zu

  DECODE(X, NULL, Y, X)

  D.h. ist X nicht Null, dann ist X das Ergebnis. Ist X Null, dann ist Y das Ergebnis.
- COALESCE(X, Y) ist das gleiche im SQL-92 Standard. Es ist dort die Abkürzung für

  CASE WHEN X IS NOT NULL THEN X ELSE Y END

Bedingte Ausdrücke (6)

- Z.B.: Ausgabe der E-Mail-Adressen aller Studenten, bzw. Text "keine", wenn die Spalte Null ist:

```
SELECT VORNAME, NACHNAME, NVL(EMAIL, 'keine')
FROM   STUDENTEN
```

- Bedingte Ausdrücke sind normale Terme.
- Daher kann man sie auch als Eingabe für Datentypfunktionen oder Aggregationsfunktionen verwenden.

Inhalt

1. Aggregationen I: Aggregationsfunktionen
2. Aggregationen II: GROUP BY, HAVING
3. UNION, Bedingte Ausdrücke
4. Sortieren der Ausgabe: ORDER BY
5. SQL-92 Verbunde, Äußerer Verbund in Oracle

Sortieren der Ausgabe (1)

- Wenn die Ausgabe länger als einige wenige Zeilen ist, sollte sie übersichtlich sortiert werden.

  Es ist viel einfacher, einen speziellen Wert in einer sortierten Tabelle zu suchen. Ohne "ORDER BY" bedeutet die Reihenfolge der Ausgabezeilen nichts (sie hängt von den verwendeten Algorithmen des DBMS ab).

- Es ist aber wichtig zu verstehen, daß die Entwicklung der Logik einer Anfrage und die Formatierung der Ausgabe zwei verschiedene Dinge sind.

  Während die Sortierung der einzige Formatierungsbefehl im SQL-Standard ist, bieten DBMS meist noch mehr Optionen. Z.B. einen Seitenumbruch zu machen bei Änderung des Wertes einer Spalte, negative Werte in Rot auszudrucken, etc. Die Sortierung kann jedoch auch wichtig sein, wenn ein Anwendungsprogramm die Daten erhält.

Sortieren der Ausgabe (2)

- Beispiel: Geben Sie die Namen der Studenten aus, die Hausaufgabe 1 gelöst haben. Sortieren Sie die Liste alphabetisch nach dem Nachnamen:

```
SELECT   S.VORNAME, S.NACHNAME
FROM     STUDENTEN S, BEWERTUNGEN B
WHERE    S.SID = B.SID
AND      B.ATYP = 'H' AND B.ANR = 1
ORDER BY S.NACHNAME
```

VORNAME	NACHNAME
Michael	Grau
Lisa	Weiss
Daniel	Sommer

Sortieren der Ausgabe (3)

- Man kann eine Liste von Sortierkriterien festlegen.

  Die "ORDER BY"-Liste kann mehrere Spalten enthalten. Die zweite Spalte wird nur zur Sortierung verwendet, wenn zwei Tupel den gleichen Wert in der ersten Spalte haben, usw. Weitere Sortierkriterien sind nur sinnvoll, wenn es Duplikate in den vorherigen Spalten geben kann.

- Z.B.: HA-Ergebnisse, sortiert nach Aufgabennummer, für jede Aufgabe nach Punkten (absteigend), bei gleicher Punktzahl alphabetisch nach Namen:

```
SELECT   ANR, PUNKTE, VORNAME, NACHNAME
FROM     STUDENTEN S, BEWERTUNGEN B
WHERE    S.SID = B.SID AND B.ATYP = 'H'
ORDER BY ANR, PUNKTE DESC, NACHNAME, VORNAME
```


Sortieren der Ausgabe (4)

- Ergebnis der Beispielanfrage der vorherigen Folie:

ANR	PUNKTE	VORNAME	NACHNAME
1	10	Lisa	Weiss
1	9	Michael	Grau
1	5	Daniel	Sommer
2	9	Michael	Grau
2	8	Lisa	Weiss

- Die ersten beiden Tupel haben den gleichen Wert im ersten Sortierkriterium (ANR), das zweite Kriterium (PUNKTE DESC) legt dann ihre Reihenfolge fest.

  Hierbei ist es egal, daß die Reihenfolge nach dem dritten Kriterium (NACHNAME) andersherum wäre.

Sortieren der Ausgabe (5)

- Nach dem SQL-92 Standard kann man nur nach Spalten sortieren, die ausgegeben werden.

  Z.B. ist es nicht möglich, eine Liste von Studierenden sortiert nach Gesamtpunktzahl zu erstellen, ohne diese auszugeben. Werkzeuge wie SQL*Plus können aber Ausgabespalten unterdrücken.

- Man kann aber in allen fünf Systemen (Oracle 8, DB2, SQL Server, Access, MySQL) nach jedem Term sortieren, der unter SELECT stehen könnte.

  In disem Systemen ist es nicht notwendig, daß der Term auch in der SELECT-Liste vorkommt. Z.B. könnte man nach UPPER(NACHNAME) sortieren, aber NACHNAME ausgeben. Bei Angabe von DISTINCT kann man dagegen nur nach Ergebnisspalten sortieren (in Oracle kann man sie noch in Ausdrücken verwenden, und MySQL hat keine Beschränkung).

Sortieren der Ausgabe (6)

- Manchmal muss man Spalten zu einer DB-Tabelle zufügen, um ein Sortierkriterium zu erhalten, z.B.
  - ◇ Die Ergebnisse sollen in der Reihenfolge "HA, Zwischen-, Endklausur" ausgegeben werden.
  - ◇ Die "MLU Halle-Wittenberg" sollte in einer Universitätsliste unter "H" stehen, nicht unter "M".
- Wäre der Studentenname als eine Zeichenkette der Form "Vorname Nachname" gespeichert, wäre es sehr schwierig, nach dem Nachnamen zu sortieren.

  Frage beim DB-Entwurf: Was will ich mit den Daten machen?

Sortieren der Ausgabe (7)

- "DESC" bedeutet descending/absteigend (von hoch zu tief), Default ist "ASC" (ascending/aufsteigend).
- Man kann sich auch durch Nummern auf Spalten beziehen, z.B.: `ORDER BY 2, 4 DESC, 1`

  Spaltennummern beziehen sich auf die Reihenfolge in der SELECT-Liste. Dies war in früheren SQL Versionen wichtig, weil man Ergebnisspalten wie z.B. SUM(PUNKTE) nicht benennen/umbenennen konnte. Heute sollte man Spaltennamen verwenden (übersichtlicher).
- Nullwerte werden alle als erstes oder als letztes aufgelistet (abhängig vom DBMS).

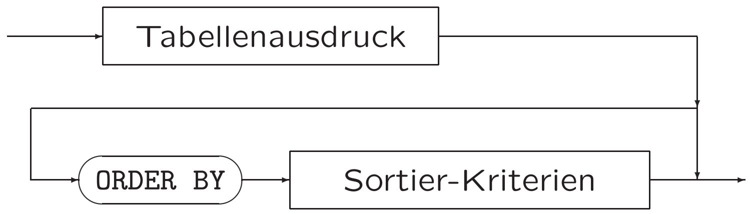
  In Oracle kann man NULLS FIRST oder NULLS LAST festlegen.

Sortieren der Ausgabe (8)

- Der Effekt von "ORDER BY" ist nur kosmetisch: Die Menge der Ausgabetupel wird nicht verändert.
- Deshalb kann "ORDER BY" nur am Ende einer Anfrage angewandt werden. Es kann nicht in Unteranfragen verwendet werden.
- Auch wenn mehrere SELECT-Ausdrücke mit UNION verknüpft werden, kann ORDER BY nur ganz am Ende stehen (es bezieht sich auf alle Ergebnistupel).

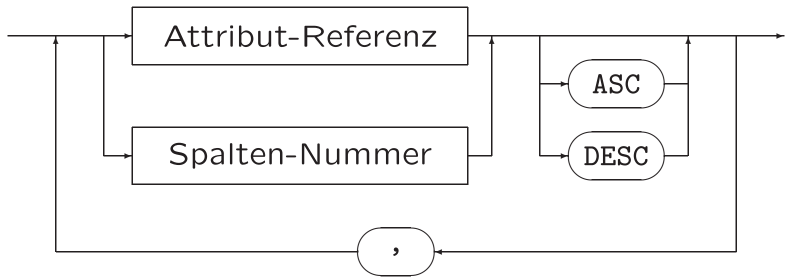
Sortieren der Ausgabe (9)

SQL-Anfrage:



Sortieren der Ausgabe (10)

Sortier-Kriterien:



- Die meisten DBMS lassen "Term" statt "Attribut-Referenz" zu (außer wenn DISTINCT oder UNION etc. spezifiziert wurden). Dann gelten die gleichen Beschränkungen wie für Terme in der SELECT-Liste (es kann weitere Beschränkungen für die Verwendung von Aggregationen geben).

"Erste n" Anfragen (1)

- Gesucht sind die beiden Studenten mit dem besten Ergebnis für Hausaufgabe 1 (allgemein: die ersten n bezüglich einer Ordnung: "Top-N Queries").
- Solche Anfragen sind mit den Möglichkeiten des SQL Standards relativ schwierig zu lösen: Man kann zählen, wie viele ein schlechteres Ergebnis haben (siehe nächste Folie).
- Da solche Anfragen aber relativ häufig vorkommen, haben viele Systeme ein spezielles Konstrukt dafür, das dann aber nicht portabel ist.

"Erste n" Anfragen (2)

- Erste n (hier $n = 2$) Studenten bezüglich der Punkte in Hausaufgabe 1 (bei gleicher Punktzahl: SID):

```
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    2 > (SELECT COUNT(*) FROM BEWERTUNGEN X
            WHERE  X.ATYP = 'H' AND X.ANR = 1
            AND    (X.PUNKTE > B.PUNKTE OR
                    X.PUNKTE = B.PUNKTE AND
                       X.SID < B.SID))
ORDER  BY B.PUNKTE
```


"Erste n" Anfragen (3)

- Wie oben erläutert, kann man bei neueren Datenbanken die Unteranfrage auch links schreiben:

```
(SELECT COUNT(*) ...) < 2
```

- Die Formel

```
(X.PUNKTE > B.PUNKTE OR
 X.PUNKTE = B.PUNKTE AND X.SID < B.SID)
```

entspricht dem Sortierkriterium

```
ORDER BY PUNKTE DESC, SID
```

Es werden also die X gezählt, die mehr Punkte als B haben, oder bei gleicher Punktzahl eine kleinere SID.

"Erste n" Anfragen (4)

- Falls man bei gleicher Punktzahl doch alle auflisten will, also ggf. auch mehr als n (hier $n = 2$):

```
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    2 > (SELECT COUNT(*) FROM BEWERTUNGEN X
            WHERE  X.ATYP = 'H' AND X.ANR = 1
            AND    X.PUNKTE > B.PUNKTE)
ORDER  BY B.PUNKTE
```

“Erste n” Anfragen (5)

- Oracle hat die 0-stellige Funktion `ROWNUM`, die eindeutige Nummern für die Ausgabezeilen erzeugt (beginnend mit 1).

  In der Oracle Dokumentation findet man `ROWNUM` im Abschnitt “Pseudocolumns”, d.h. virtuelle Spalten, die bei Bedarf berechnet werden. In diesem Fall wäre es allerdings eine Spalte der Ausgabetabelle.

- Z.B. kann man damit die Anzahl der Ausgabezeilen begrenzen:

```
SELECT SID, PUNKTE
FROM   BEWERTUNGEN
WHERE  ATYP = 'H' AND ANR = 1
AND    ROWNUM <= 2
```


“Erste n” Anfragen (6)

- Die obige Anfrage gibt zwei Bewertungen für Hausaufgabe 1 aus. Es ist nicht vorhersehbar, welche Bewertungen ausgegeben werden.

  Die Auswahl hängt von internen Algorithmen des Anfrageoptimierers ab und kann sich von einer Oracle-Version zur nächsten ändern.

  Genauer funktioniert `ROWNUM` folgendermaßen: Es wird ein Zähler verwendet, der mit 1 initialisiert ist. `ROWNUM` liefert immer den aktuellen Wert des Zählers. Falls eine Tupelkombination die `WHERE`-Bedingung erfüllt hat, wird der Zähler hochgesetzt.

  `ROWNUM` bezieht sich nicht eigentlich auf Ausgabezeilen. Der Unterschied wird bei “`GROUP BY`”-Anfragen deutlich: Hier kann `ROWNUM` unter `WHERE`, aber nicht unter `SELECT` verwendet werden. Bei Anfragen ohne “`GROUP BY`” kann es natürlich auch unter `SELECT` verwendet werden.

“Erste n” Anfragen (7)

- Da “`ORDER BY`” erst nach der `WHERE`-Bedingung ausgewertet wird, hat es keinen Einfluss auf `ROWNUM`.

  Möglicherweise wählt der Optimierer aber einen anderen Auswertungsplan, was dann die ausgewählten Zeilen verändern könnte.

- Es geht aber mit einer Unteranfrage (wobei aber `ORDER BY` in der Unteranfrage den Standard verletzt):

```
SELECT SID, PUNKTE
FROM   (SELECT SID, PUNKTE
        FROM   BEWERTUNGEN
        WHERE  ATYP = 'H' AND ANR = 1
        ORDER  BY PUNKTE DESC, SID)
WHERE  ROWNUM <= 2
```


“Erste n” Anfragen (8)

- In DB2 würde diese Aufgabe (die beiden Studenten mit bester Punktzahl für Hausaufgabe 1) so gelöst:

```
SELECT SID, PUNKTE
FROM   BEWERTUNGEN
WHERE  ATYP = 'H' AND ANR = 1
ORDER  BY PUNKTE
FETCH  FIRST 2 ROWS ONLY
```

  Falls nur ein Antworttupel: “`FETCH FIRST ROW ONLY`”.

- Im MySQL heißt es “`LIMIT 2`” statt “`FETCH ...`”.

  Man kann auch `LIMIT 0, 2` schreiben: Der erste Parameter ist ein Offset (Anzahl Zeilen, die am Anfang weggelassen werden).

Ranking (1)

- Moderne Weiterentwicklung: "Window Functions".
  Ähnlich zu `ROWNUM`, aber wesentlich allgemeiner:
  - ◇ Man kann Tupel in einer Anfrage nach verschiedenen Kriterien durchnumerieren, und nach einem weiteren Kriterium die Ausgabe sortieren.
  - ◇ Ranking wird nach `GROUP BY` ausgeführt.
  - ◇ Man kann lokale Zähler innerhalb von Partitionen verwenden, nicht nur einen pro Anfrage.
  - ◇ Man kann entscheiden, wie mit Gruppen von Tupeln umgegangen wird, die sich bezüglich des Sortierkriteriums nicht unterscheiden.

Ranking (2)

- Z.B. alle Abgaben für Hausaufgabe 1 mit Position in der Sortierreihenfolge bezüglich Punkten, aber Ausgabe sortiert nach Namen:

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

- `RANK` liefert die Position in der Sortierreihenfolge.

  `RANK` gibt es z.B. in Oracle (ab Version 8i) und DB2. Diese Funktionen heißen "Window Functions" oder auch "Analytic Functions".

Inhalt

1. Aggregationen I: Aggregationsfunktionen
2. Aggregationen II: GROUP BY, HAVING
3. UNION, Bedingte Ausdrücke
4. Sortieren der Ausgabe: ORDER BY
5. SQL-92 Verbunde, Äußerer Verbund in Oracle

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
H	3	SQL	10
Z	1	SQL	14

Hausaufgabe 3 ist neu.
Es gibt noch keine Abgaben.

Verbunde in SQL-92 (1)

- Sobald man sich in einer Anfrage auf mehr als eine Tabelle bezieht, und eine Bedingung zur Verknüpfung dieser Tabellen angibt, verwendet man einen "Verbund" dieser Tabellen (engl. "Join").

  Dies wurde in Kapitel 4 ja schon ausführlich besprochen.

- Z.B. enthält diese Anfrage einen Verbund der Tabellen AUFGABEN und BEWERTUNGEN:

```
SELECT A.ATYP, A.ANR, COUNT(B.SID)
FROM   AUFGABEN A, BEWERTUNGEN B
WHERE  A.ATYP = B.ATYP AND A.ANR = B.ANR
AND    A.THEMA = 'SQL'
GROUP  BY A.ATYP, A.ANR
```


Verbunde in SQL-92 (2)

- In der formalen Anfragesprache "relationale Algebra", die eine wichtige Grundlage der Theorie relationaler Datenbanken ist, gibt es den Verbund als explizite Operation $\bowtie$.
- In SQL-86 mußte man den Verbund dagegen in FROM und WHERE-Klausel zerlegt ausdrücken, wobei die Verbund-Bedingung zwischen eventuell vielen anderen Bedingungen steht (an beliebiger Stelle).
- In SQL-92 wurde eine spezielle Syntax für Verbunde in der FROM-Klausel eingeführt.

Verbunde in SQL-92 (3)

- In SQL-92 kann man die obige Anfrage auch so schreiben ("Inner Join" = normaler Verbund, s.u.):

```
SELECT A.ATYP, A.ANR, COUNT(B.SID)
FROM   AUFGABEN A INNER JOIN BEWERTUNGEN B
       ON A.ATYP = B.ATYP AND A.ANR = B.ANR
WHERE  A.THEMA = 'SQL'
GROUP  BY A.ATYP, A.ANR
```

- Es ist eine Geschmacksfrage, welche Syntax man verwendet.

  Es ist zwar die Verbund-Bedingung jetzt deutlich von anderen Bedingungen getrennt, aber die Anfrage ist nicht kürzer geworden, und es gibt noch einige ältere Systeme, die diese Syntax nicht verstehen.

Verbunde in SQL-92 (4)

- Es hätte sich sicher nicht gelohnt, die SQL Syntax durch ein neues Konstrukt zu verkomplizieren, wenn man damit nur Anfragen etwas anders strukturiert aufschreiben kann, ohne deutliche Verbesserung.
- Es gibt aber einen speziellen Verbund, den äußeren Verbund ("Outer Join"), den man in klassischem SQL zwar ausdrücken kann, aber nur umständlich.
- Dies ist der eigentliche Grund, warum die SQL Syntax erweitert wurde.

  Bestimmte Anfragen können mit den neuen Konstrukten wesentlich kürzer aufgeschrieben werden.

Verbunde in SQL-92 (5)

- Der normale Verbund eliminiert Tupel, die keinen Verbundpartner besitzen.

  Im Beispiel erscheint die neue Hausaufgabe 3, für die es noch keine Abgaben gibt (also keine passenden Zeilen in BEWERTUNGEN), nicht im Ergebnis.

- Beim linken äußeren Verbund ist dagegen garantiert, daß alle Tupel der linken Tabelle auch im Ergebnis des Verbundes auftauchen.

  Sie können natürlich noch durch die WHERE-Klausel eliminiert werden.

- Gibt es kein passendes Tupel der rechten Tabelle, so werden deren Spalten durch Nullwerte ersetzt.

Verbunde in SQL-92 (6)

- Wenn man den linken äußeren Verbund verwendet, erscheint jetzt auch Hausaufgabe 3 im Ergebnis:

```
SELECT A.ATYP, A.ANR, COUNT(B.SID)
FROM   AUFGABEN A LEFT OUTER JOIN BEWERTUNGEN B
       ON A.ATYP = B.ATYP AND A.ANR = B.ANR
WHERE  A.THEMA = 'SQL'
GROUP  BY A.ATYP, A.ANR
```

- Das COUNT liefert auch die korrekte Anzahl von Abgaben, nämlich 0, weil B.SID für Hausaufgabe 3 NULL ist (es gibt ja gerade kein passendes Tupel B), und COUNT Nullwerte nicht mitzählt.

Verbunde in SQL-92 (7)

- Äquivalente Anfrage in SQL-86 (12 vs. 5 Zeilen):

```
SELECT   A.TYP, A.ANR, COUNT(B.SID)
FROM     AUFGABEN A, BEWERTUNGEN B
WHERE    A.ATYP = B.ATYP AND A.ANR = B.ANR
AND      A.THEMA = 'SQL'
GROUP BY A.ATYP, A.ANR
UNION ALL
SELECT   A.ATYP, A.ANR, 0
FROM     AUFGABEN A
WHERE    A.THEMA = 'SQL' AND NOT EXISTS
         (SELECT *
          FROM   BEWERTUNGEN B
          WHERE  A.ATYP=B.ATYP AND A.ANR=B.ANR)
```


Verbunde in SQL-92 (8)

- Natürlich gibt es auch den rechten äußeren Verbund (sowie den beidseitigen "Full Outer Join"):

```
FROM AUFGABEN A RIGHT OUTER JOIN BEWERTUNGEN B
     ON A.ATYP = B.ATYP AND A.ANR = B.ANR
```

- Die Tupel der durch LEFT oder RIGHT markierten Tabelle sind beim Verbund "geschützt", werden also durch den Verbund nicht eliminiert.
- Im Beispiel entspricht das Ergebnis dem normalen Verbund, weil durch den Fremdschlüssel garantiert ist, daß es zu jedem B einen Verbundpartner A gibt.

Verbunde in SQL-92 (9)

- Neben den verschiedenen Typen von Verbunden in SQL-92 gibt es auch noch verschiedene Möglichkeiten, die Verbund-Bedingung anzugeben.
- Statt
```
AUFGABEN A LEFT OUTER JOIN BEWERTUNGEN B
ON A.ATYP = B.ATYP AND A.ANR = B.ANR
```
kann man in einigen Systemen auch schreiben:
```
AUFGABEN A LEFT OUTER JOIN BEWERTUNGEN B
USING (ATYP, ANR)
```
oder sogar noch kürzer:
```
AUFGABEN A NATURAL LEFT OUTER JOIN BEWERTUNGEN B
```


Verbunde in SQL-92 (10)

- `USING` und `NATURAL` setzen voraus, daß die Tabellen über Spalten verknüpft werden sollen, die in beiden Tabellen gleich heißen.

  Dagegen erlaubt `ON` beliebige Bedingungen, z.B. auch Verknüpfungen über Spalten mit verschiedenen Namen, aber fordert mehr Tipparbeit.
- Bei `USING` werden diese Spalten explizit angegeben.

  Man könnte auch nur eine Teilmenge der gleich benannten Spalten wählen. `USING` und `NATURAL` sind in vielen Systemen nicht implementiert.
- Bei `NATURAL` ("natürlicher Verbund") müssen alle gleich benannten Spalten gleiche Werte enthalten.

Mögliche Fehler (1)

- Geben Sie für jeden Studenten die Anzahl der abgegebenen Hausaufgaben aus (einschließlich 0).
- Die folgende Anfrage funktioniert nicht: Studenten ohne Hausaufgaben werden nicht aufgelistet.

```
SELECT VORNAME, NACHNAME, COUNT(ANR)    Falsch!
FROM   STUDENTEN S LEFT OUTER JOIN BEWERTUNGEN B
       ON S.SID = B.SID
WHERE  B.ATYP = 'H'
GROUP BY S.SID, VORNAME, NACHNAME
```

Die `WHERE`-Klausel wird nach dem Verbund ausgewertet und elimiert Studierende ohne HA: Hat er/sie Bewertungen für einen anderen `ATYP`, so greift der äußere Verbund nicht, sonst ist `B.ATYP` unter `WHERE` null.

Mögliche Fehler (2)

- Mögliche Verbundpartner dürfen nicht nach Konstruktion des äußeren Verbundes eliminiert werden.
- Man muss die Hausaufgabenergebnisse selektieren bevor der äußere Verbund gebildet wird:

```
SELECT VORNAME, NACHNAME, COUNT(B.ANR)
FROM   STUDENTEN S LEFT OUTER JOIN
           (SELECT SID, ANR
            FROM   BEWERTUNGEN
            WHERE  ATYP = 'H') B
       ON S.SID = B.SID
GROUP BY S.SID, VORNAME, NACHNAME
```

Mögliche Fehler (3)

- Man kann auch die Bedingung für die rechte Tabelle in die Verbundbedingung integrieren:

```
SELECT VORNAME, NACHNAME, COUNT(B.ANR)
FROM   STUDENTEN S LEFT OUTER JOIN BEWERTUNGEN B
       ON S.SID = B.SID AND B.ATYP = 'H'
GROUP BY S.SID, VORNAME, NACHNAME
```

- SQL-92 lässt jede WHERE-Bedingung, die sich nur auf die rechte oder linke Tabelle bezieht, zu. (Das sollte aber nicht missbraucht werden.)

  Es scheint, daß DB2 und Access keine Unteranfragen in der ON-Klausel zulassen. In Access müssen komplexere Bedingungen in Klammern eingeschlossen werden.

Mögliche Fehler (4)

- Bedingungen für die linke Tabelle machen in einem linken äußeren Verbund wenig Sinn.
- Z.B. betrachte man diese Anfrage:

```
SELECT A.ATYP, A.ANR, B.SID, B.PUNKTE
FROM   AUFGABEN A LEFT OUTER JOIN BEWERTUNGEN B
       ON A.ATYP = 'H' AND B.ATYP = 'H'
          AND A.ANR = B.ANR
```

- Übung:

  Wird A.ATYP = 'Z' in der Ausgabe auftauchen?

  ☐ ja ☐ nein

Outer Join statt Not Exists

- MySQL hat keine Unteranfragen, aber manchmal kann man dafür einen äußeren Verbund verwenden.
- Z.B. Studenten ohne eine gelöste Hausaufgabe:

```
SELECT S.SID. S.VORNAME, S.NACHNAME
FROM   STUDENTEN S LEFT OUTER JOIN BEWERTUNGEN B
       ON S.SID = B.SID AND B.ATYP = 'H'
WHERE  B.ATYP IS NULL
```

- Natürlich kann man statt B.ATYP jedes Attribut von BEWERTUNGEN auf Null testen.

  Der Test auf den Nullwert prüft, ob das aktuelle STUDENTEN-Tupel einen Verbundpartner gefunden hat.

Verbundsyntax in SQL-92 (1)

- SQL-92 hat folgende Verbundtypen:
  - ◇ [INNER] JOIN: Gewöhnlicher Verbund.
  - ◇ LEFT [OUTER] JOIN: Erhält Tupel der linken Tabelle.
  - ◇ RIGHT [OUTER] JOIN: Erhält alle Tupel von rechts.
  - ◇ FULL [OUTER] JOIN: Erhält alle Eingabetupel.
  - ◇ CROSS JOIN: Kartesisches Produkt $\times$.
  - ◇ UNION JOIN: Diese Vereinigung füllt die Spalten der anderen Tabelle mit Nullwerten auf.
- Schlüsselworte in [...] sind optional.

Verbundsyntax in SQL-92 (2)

- Mögliche Spezifikationen der Verbundbedingung:
  - ◇ Schlüsselwort NATURAL vor dem Verbundnamen.
  - ◇ "ON ⟨Bedingung⟩" folgt dem Verbund.
  - ◇ "USING ($A_1, \ldots, A_n$)" folgt dem Verbund.

    USING listet alle Verbundattribute (z.B. um den natürlichen Verbund zu spezifizieren). Attribute mit den Namen $A_1, \ldots, A_n$ müssen in beiden Tabellen auftauchen. Die Verbundbedingung ist dann $R.A_1 = S.A_1 \land \cdots \land R.A_n = S.A_n$. NATURAL ist äquivalent zu USING mit allen gleichen Attributnamen.
- Nur eines der Konstrukte kann verwendet werden.
- CROSS JOIN und UNION JOIN haben keine Verbundbedingung.

Verbundsyntax in SQL-92 (3)

- Nach dem Standard liefern der NATURAL Join und der Join mit USING eine Tabelle mit nur einer Kopie der gemeinsamen Attribute.
- Diese Attribute sind die ersten Spalten im Ergebnis. Man kann sie nicht mit Tupelvariablen ansprechen.

```
SELECT *
FROM   BEWERTUNGEN B NATURAL JOIN AUFGABEN A
```

- Die Ergebnisspalten sind ATYP, ANR, B.SID, B.PUNKTE, A.THEMA, A.MAXPT (in dieser Reihenfolge).

  Es ist unzulässig, sich auf B.ATYP oder A.ATYP zu beziehen: Es kann nur ATYP verwendet werden (das gleiche gilt für ANR).

Verbundsyntax in SQL-92 (4)

- Oracle 9i unterstützt die SQL-92 Verbunde.

  Einschließlich des Natural Join, aber ohne UNION JOIN (der in SQL:1999 entfernt wurde). Oracle 8i unterstützte keinen der SQL-92 Verbunde.
- Innerer und äußerer Verbund mit ON funktionieren auch in DB2, SQL Server, Access und MySQL.

  In Access und MySQL ist das Schlüsselwort INNER nicht optional.
- USING und NATURAL funktionieren nur in Oracle 9i.

  NATURAL existiert auch in MySQL, aber MySQL vereinigt die gleichen Attribute nicht. Das verletzt den SQL-92 Standard.

Verbundsyntax in SQL-92 (5)

- CROSS JOIN wird nur in Oracle 9i, SQL Server und MySQL, aber nicht in Access und DB2 unterstützt.

  Da man ein Komma für den CROSS JOIN schreiben kann, ist dies auch nicht sehr sinnvoll.
- UNION JOIN unterstützt keins der fünf Systeme.

  Man kann aber in SQL-92 (und z.B. Oracle, DB2, SQL Server, nicht Access) eine Unteranfrage unter FROM schreiben, die UNION oder UNION ALL enthält. Mit etwas mehr Aufwand kann also der Union Join simuliert werden. Nebenbei bemerkt, ist es etwas seltsam, daß z.B. "FROM A NATURAL JOIN B" in SQL-92 zulässig ist, aber "FROM A UNION B" nicht. Auch lässt SQL-92 die Schreibweise "FROM (SELECT * FROM A UNION SELECT * FROM B) X" zu, aber das gleiche mit "NATURAL JOIN" statt "UNION" liefert einen Syntaxfehler [Date/Darwen, 1997, S. 148].

Verbundsyntax in SQL-92 (6)

- Man kann in der FROM-Klausel auch sowohl Verbunde verwenden, als auch weitere Tupelvariablen deklarieren (getrennt durch “,”).
- Das Ergebnis eines Verbundes zweier Tabellen kann mit einer dritten Tabelle verbunden werden (usw.). Die Syntax ist:

```
SELECT ...
FROM   R LEFT JOIN S ON R.A=S.B
         LEFT JOIN T ON S.C=T.D
```

- Man kann auch Klammern setzen, aber dann muss man nach (...) eine neue Tupelvariable deklarieren.

Äußerer Verbund: Oracle 8 (1)

- In Oracle wird der äußere Verbund traditionell unter WHERE spezifiziert (nicht länger notwendig in 9i).
- Statt der Bedingung $R.A = S.B$ schreibt man
  - ◇ $R.A = S.B(+)$ für den linken äußeren Verbund
  - ◇ $R.A(+) = S.B$ für den rechten äußeren Verbund

  D.h. das Zeichen “(+)” wird an die Attribute angehängt, die durch Null ersetzt werden können.

  D.h. dies erhält die Tupel der anderen Tabelle (die nicht mit “(+)” markiert sind). Viele syntaktische Restriktionen sichern, daß dies wirklich ein äußerer Verbund ist. Wird der Verbund über mehrere Attribute durchgeführt, müssen alle markiert werden. Man kann auch $S.B(+) = c$ mit einer Konstante c schreiben, oder z.B. $R.A = S.B(+)+1$.

Äußerer Verbund: Oracle 8 (2)

- Z.B. Anzahl der Abgaben pro HA (kann 0 sein):

```
SELECT   A.ATYP, A.ANR, COUNT(SID)
FROM     AUFGABEN A, BEWERTUNGEN B
WHERE    A.ATYP = B.ATYP(+) AND A.ANR = B.ANR(+)
GROUP BY A.ATYP, A.ANR
```

- Wie im Äußeren Verbund von SQL-92, wird der äußere Verbund durchgeführt, bevor irgendeine andere Bedingung der WHERE-Klausel angewandt wird.

  Egal in welcher Reihenfolge die Bedingungen stehen. Aber wie oben gezeigt, kann man eine Unteranfrage unter FROM machen, um vor dem äußeren Verbund zu selektieren.