

# Einführung in Datenbanken

---

## Kapitel 14: Relationale Algebra in SQL

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2022/23

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

# Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- **UNION** und **UNION ALL** in SQL-Anfragen verwenden.

Auch solche mit anschließendem `ORDER BY` oder mit einem Null-Wert in der `SELECT`-Liste. Anfragen mit den Operatoren `INTERSECT` und `EXCEPT` lesen können.

- Den Outer Join in der SQL-92 Join-Syntax unter `FROM` verwenden.

Normale Joins mindestens lesen können. Die Verwendung des Inner Join in dieser Syntax ist Geschmackssache, der Outer Join führt dagegen zu kürzeren Anfragen.

- Einige Probleme/mögliche Fehler bei Outer Join Anfragen aufzählen und erklären.

Und natürlich in eigenen Anfragen diese Fehler vermeiden.

# Inhalt

- ① Einleitung
- ② UNION
- ③ Outer Join

# Beispiel-Datenbank

## STUDENTEN

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

## BEWERTUNGEN

<u>SID</u>	<u>ATYP</u>	<u>ANR</u>	<u>PUNKTE</u>
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

<u>ATYP</u>	<u>ANR</u>	<u>THEMA</u>	<u>MAXPT</u>
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

# Einleitung

- Den ursprünglichen SQL-Kern kann man am einfachsten als Variante des Tupelkalküls verstehen (der wiederum eine Anwendung der Prädikatenlogik ist).

Allerdings müssen in SQL Tupelvariablen direkt bei der Deklaration an eine Relation gebunden werden. Dadurch spart man sich relativ komplizierte Einschränkungen der möglichen Formeln, um die endliche Auswertbarkeit zu garantieren. Der Preis dafür ist, dass UNION unbedingt nötig ist. Damit hatte SQL von Anfang an auch einen Operator der relationalen Algebra.

- Wenn man will, kann man SQL aber auch aus Sicht der relationalen Algebra verstehen.

Allerdings sind EXISTS-Unterabfragen schwierig in relationale Algebra zu übersetzen, man muss dann die ganze logische Formel über Mengenoperationen wie  $\cap$ ,  $\cup$  ausdrücken (statt  $\wedge$ ,  $\vee$ ).

- Mit SQL-92 kamen zu SQL explizite Joins hinzu.

# Von Relationaler Algebra zu SQL (1)

- Ein Ausdruck in relationaler Algebra

$$\pi_{A_1, \dots, A_n} (\sigma_F (R_1 \times \dots \times R_m))$$

wird in SQL geschrieben als

```
SELECT A1, . . . , An
FROM   R1, . . . , Rm
WHERE  F
```

- Das Schlüsselwort **SELECT** entspricht also der Projektion.
- Die Relationen unter **FROM** werden mit  $\times$  verknüpft.
- Unter **WHERE** wird die Selektionsbedingung geschrieben.

Umgekehrt ist aber nicht jede **WHERE**-Bedingung auch als Selektionsbedingung möglich (keine Unteranfragen).

## Von Relationaler Algebra zu SQL (2)

- Haben verschiedene  $R_i$  Attribute mit gleichem Namen  $A$ , schreibt man  $R_i.A$ , um den Bezug eindeutig zu machen.

- Man kann aber auch Umbenennungen verwenden:

$$\pi_{A_1, \dots, A_n}(\sigma_F(\rho_{X_1}(R_1) \times \dots \times \rho_{X_m}(R_m)))$$

wird in SQL geschrieben als

```
SELECT A1, . . . , An
FROM   R1 X1, . . . , Rm Xm
WHERE  F
```

Man kann die  $X_i$  auch als Kopien der Relationen  $R_i$  verstehen.

- SQL arbeitet mit Multimengen von Tupeln, Duplikate sind also möglich.
- Bei Bedarf verwende man **SELECT DISTINCT** statt **SELECT**.

## Von Relationaler Algebra zu SQL (3)

- SQL hat einen **UNION**-Operator, um Ergebnisse mehrerer **SELECT**-Ausdrücke zu kombinieren.

Dieser wird unten eingeführt und ausführlich erklärt.

- Es gibt entsprechend auch andere Mengenoperationen in SQL, diese werden aber selten verwendet:
  - **EXCEPT** für Mengendifferenz,
  - **INTERSECT** für den Schnitt.

Meist verwendet man eher Unteranfragen (z.B. **NOT EXISTS**).

- Unteranfragen unter **FROM** erlauben den typischen Algebra-Stil, dass man Relationen schrittweise verknüpft.

# Inhalt

① Einleitung

② UNION

③ Outer Join

# UNION (1)

- In SQL kann man die Ergebnisse von zwei Anfragen mit **UNION** 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 repräsentiert 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 verknüpft 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  
Es werden alle Duplikate entfernt, nicht nur solche, die durch die Vereinigung entstanden sind.
  - **UNION ALL**: Konkatenation (erhält Duplikate).

Duplikatelimination ist ziemlich teuer.

## 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 Mengenoperationen in SQL

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

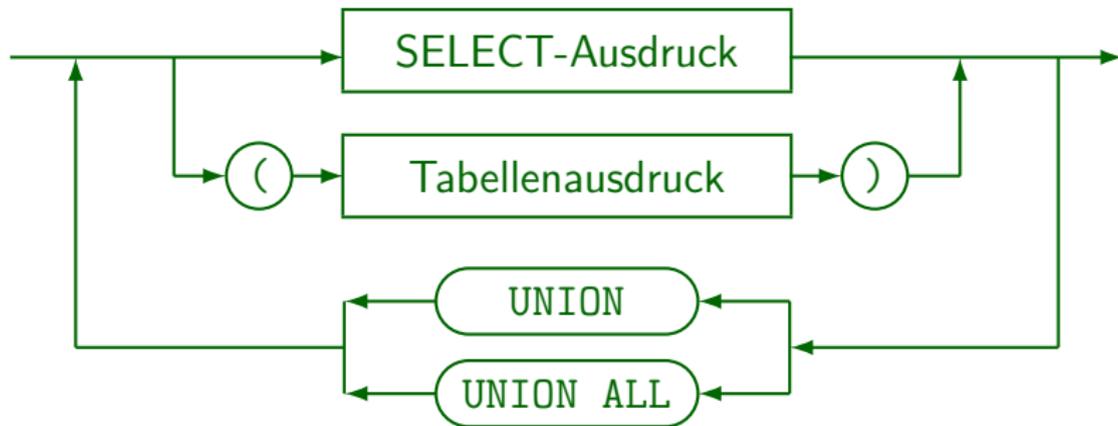
SQL-86, SQL Server, Access und MySQL (ab Ver. 4) unterstützen nur **UNION [ALL]**. Vor Ver. 4 hatte MySQL kein **UNION**. DB2 bietet alle SQL-92 Mengenoperatoren. In Oracle heißt es **MINUS** statt **EXCEPT**. Für **MINUS** und **INTERSECT** wird **ALL** in Oracle nicht unterstützt.

- Diese Operationen tragen nichts zur Ausdruckskraft von SQL bei.

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

## Aufgabe:

- 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 klassische Lösung für Fallunterscheidungen ist, haben neuere DBMS-Versionen auch bedingte Ausdrücke (im SQL-Standard seit SQL-92).
- Z.B.: Ausgabe der Ergebnisse von Lisa Weiss, dabei Aufgabentyp ausgeschrieben:

```
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'
```

## Bedingte Ausdrücke (2)

- Es ist auch folgende Abkürzung möglich:

```
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'
```

## Bedingte Ausdrücke (3)

- Oracle-spezifische Schreibweise:

`DECODE(X, X1, Y1, X2, Y2, ..., Z)`

Dies wird ausgewertet, indem das DBMS zunächst X mit X<sub>1</sub>, dann mit X<sub>2</sub>, usw. vergleicht. Ist X<sub>i</sub> der erste Wert mit X = X<sub>i</sub>, dann wird Y<sub>i</sub> zurückgegeben. Wenn kein X<sub>i</sub> passt, ist Z das Ergebnis.

```
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)
```

# Inhalt

- 1 Einleitung
- 2 UNION
- 3 Outer Join**

# Beispiel-Datenbank

## STUDENTEN

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

## BEWERTUNGEN

<u>SID</u>	<u>ATYP</u>	<u>ANR</u>	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

<u>ATYP</u>	<u>ANR</u>	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

## Verbunde in SQL-92 (1)

- Eine wichtige und nützliche Operation der relationalen Algebra ist der Verbund (mit Varianten).
- In SQL-86 kann man einen Verbund nicht direkt spezifizieren. Man verwendet das kartesische Produkt (FROM) und selektiert dann (WHERE).

Dies ist noch immer der normale Fall.

- Natürlicher Verbund von BEWERTUNGEN und AUFGABEN:

```
SELECT B.ATYP AS ATYP, B.ANR AS ANR, SID,  
       PUNKTE, THEMA, MAXPT  
FROM   BEWERTUNGEN B, AUFGABEN A  
WHERE  B.ATYP = A.ATYP AND B.ANR = A.ANR
```

## Verbunde in SQL-92 (2)

- In SQL-92 kann man z.B. schreiben:

```
SELECT SID, ANR, (PUNKTE/MAXPT)*100
FROM   BEWERTUNGEN B NATURAL JOIN AUFGABEN A
WHERE  ATYP = 'H'
```

- Durch die Schlüsselwörter „NATURAL JOIN“ fügt das System automatisch die Verbundbedingung hinzu:

```
B.ATYP = A.ATYP AND B.ANR = A.ANR
```

- SQL-92 erlaubt Verbunde in der FROM-Klausel, sowie auch auf der äußersten Anfrage-Ebene (wie UNION).

Somit kann man viel im Stil der relationalen Algebra schreiben.

## Verbunde in SQL-92 (3)

- Aktuelle Systeme unterstützen den Standard nur teilweise:
  - SQL-92 Verbunde können nicht in Oracle 8i verwendet werden, in Oracle 9i dagegen fast alle.
  - Einige Verbundtypen werden in DB2, SQL Server und Access unterstützt, aber der „natürliche Verbund“ nicht. Man muss in diesen Systemen die Bedingung explizit aufschreiben:

```
SELECT SID, B.ANR, (PUNKTE/MAXPT)*100
FROM   BEWERTUNGEN B INNER JOIN AUFGABEN A
      ON B.ATYP = A.ATYP AND B.ANR = A.ANR
WHERE  B.ATYP = 'H'
```

## Verbunde in SQL-92 (4)

- Mit der expliziten Verbundbedingung ist die Anfrage nicht kürzer als die äquivalente Anfrage in klassischer Syntax (Verbund-Bedingung unter *WHERE*).
- Die Ausdruckskraft von SQL wird durch die neuen Verbund-Konstrukte nicht vergrößert.
  - Jede Anfrage mit den neuen Verbund-Konstrukten kann in eine äquivalente Anfrage ohne diese Konstrukte überführt werden.
- Der Grund, warum Verbunde zu SQL hinzugefügt wurden, ist wahrscheinlich der „**äußere Verbund**“: Hierfür ist die äquivalente Formulierung in SQL-86 deutlich länger.

## Äußerer Verbund: Wdh.

- Der normale Verbund eliminiert Tupel ohne Verbundpartner:

A	B		B	C				
a <sub>1</sub>	b <sub>1</sub>	⋈	b <sub>2</sub>	c <sub>2</sub>	=	A	B	C
a <sub>2</sub>	b <sub>2</sub>		b <sub>3</sub>	c <sub>3</sub>		a <sub>2</sub>	b <sub>2</sub>	c <sub>2</sub>

- Der linke äußere Verbund stellt sicher, dass Tupel der linken Tabelle auch im Ergebnis vorhanden sind:

A	B		B	C		A	B	C
a <sub>1</sub>	b <sub>1</sub>	⋈	b <sub>2</sub>	c <sub>2</sub>	=	a <sub>1</sub>	b <sub>1</sub>	
a <sub>2</sub>	b <sub>2</sub>		b <sub>3</sub>	c <sub>3</sub>		a <sub>2</sub>	b <sub>2</sub>	c <sub>2</sub>

Zeilen der linken Seite werden, falls notwendig, mit „Null“ aufgefüllt.  
Es gibt auch einen rechten und einen vollen äußeren Verbund.

## Äußerer Verbund in SQL (1)

- Z.B. Anzahl der Abgaben pro Hausaufgabe. Falls es keine Abgabe gibt, soll 0 ausgegeben werden:

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

- Im Ergebnis des linken äußeren Verbunds treten alle Übungen auf. In Übungen ohne Abgaben werden die Attribute SID und PUNKTE mit Nullwerten aufgefüllt.
- COUNT(SID) zählt nur Zeilen, wo SID nicht Null ist.

## Äußerer Verbund in SQL (2)

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

```
SELECT  A.ANR, COUNT(*)
FROM    AUFGABEN A, BEWERTUNGEN B
WHERE   A.ATYP = 'H' AND B.ATYP = 'H'
AND     A.ANR = B.ANR
GROUP BY A.ANR
UNION ALL
SELECT  A.ANR, 0
FROM    AUFGABEN A
WHERE   A.ATYP = 'H'
AND     A.ANR NOT IN (SELECT B.ANR
                       FROM   BEWERTUNGEN B
                       WHERE  B.ATYP = 'H')
```

## Äußerer Verbund in SQL (3)

- 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
```

- Der äußere Verbund wird konstruiert, bevor die WHERE-Bedingung ausgewertet wird.

## Äußerer Verbund in SQL (4)

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

## Äußerer Verbund in SQL (5)

- 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.

## Äußerer Verbund in SQL (6)

- 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
```

- Aufgabe:  
Wird A.ATYP = 'Z' in der Ausgabe auftauchen?  
 ja       nein

## Äußerer Verbund in SQL (7)

- Zum Teil kann NOT EXISTS durch einen äußeren Verbund ersetzt werden (z.B. für MySQL vor Ver. 4.1).
- 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: Übersicht (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.

`UNION JOIN` wurde in SQL-99 gelöscht (ist aber interessante Idee).

- Schlüsselworte in `[...]` sind optional.

## Verbundsyntax: Übersicht (2)

- Mögliche Spezifikationen der Verbundbedingung:
  - Schlüsselwort **NATURAL** vor dem Verbundnamen.
  - „**ON** **<Bedingung>**“ folgt dem Verbund.
  - „**USING** **( $A_1, \dots, A_n$ )**“ folgt dem Verbund.

**USING** listet alle Verbundattribute (z.B. um den natürlichen Verbund zu spezifizieren). Attribute mit den Namen  $A_1, \dots, A_n$  müssen in beiden Tabellen auftauchen. Die Verbundbedingung ist dann  $R.A_1 = S.A_1 \wedge \dots \wedge R.A_n = S.A_n$ . **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: Übersicht (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: Übersicht (4)

- 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.):

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

- Um Join-Ausdrücke kann man Klammern setzen, nicht um normale Tupelvariablen-Deklarationen.

Für den (...) -Ausdruck kann man keine Tupelvariable einführen.

## Verbundsyntax: Details (1)

- 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: Details (2)

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

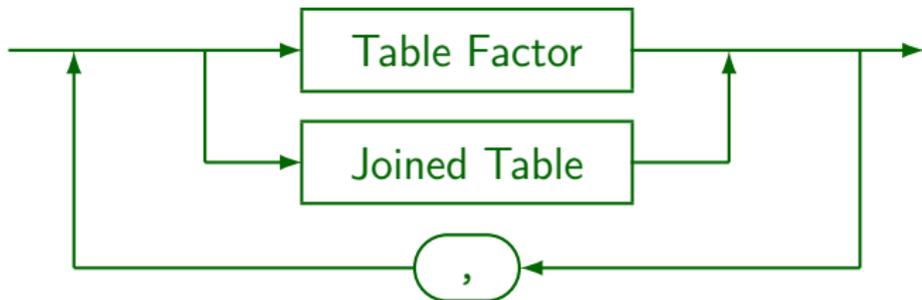
Da man ein Komma für den **CROSS JOIN** schreiben kann, ist dies auch nicht sehr sinnvoll.

- **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: Formal (1)

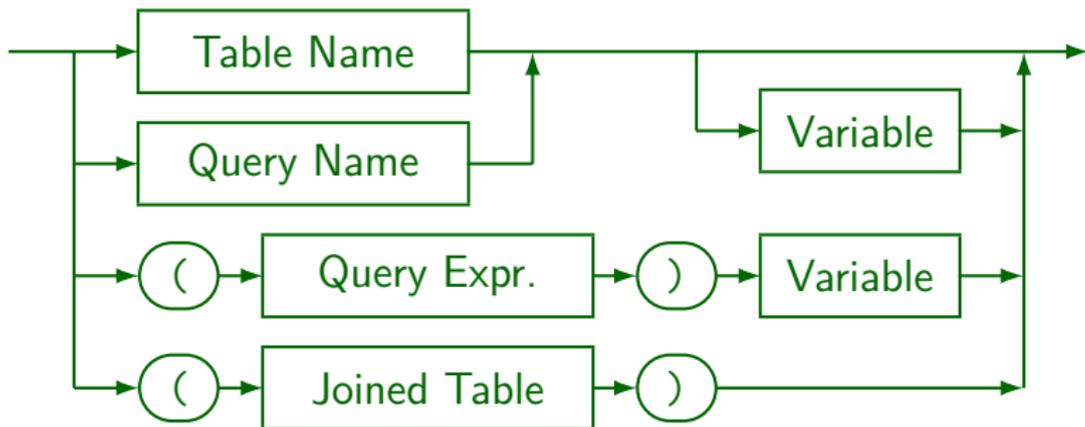
Quell-Liste (nach FROM):



- Die Syntaxgraphen sind eine vereinfachte Version der Grammatik aus dem SQL-2008 Standard. Deswegen sind die Namen der Graphen (syntaktische Kategorien, Nichtterminalsymbole) hier in Englisch. Es wurden allerdings viele kompliziertere Konstrukte weggelassen.
- Der Standard hat eine syntaktische Kategorie „Table Reference“, die (im wesentlichen) der Alternative zwischen „Table Factor“ und „Joined Table“ entspricht.

## Verbundsyntax: Formal (2)

### Table Factor:



- „Query Name“ ist mit der WITH-Klausel definiert.
- „Query Expression“ ist eine Unteranfrage (inkl. ggf. WITH, UNION, ORDER BY).
- Der Standard und viele Systeme erlauben „AS“ vor der Variable.
- Der Standard+einige DBMS erlauben Spalten-Umbenennung:  $R X(A, B)$ .

## Verbundsyntax: Formal (3)

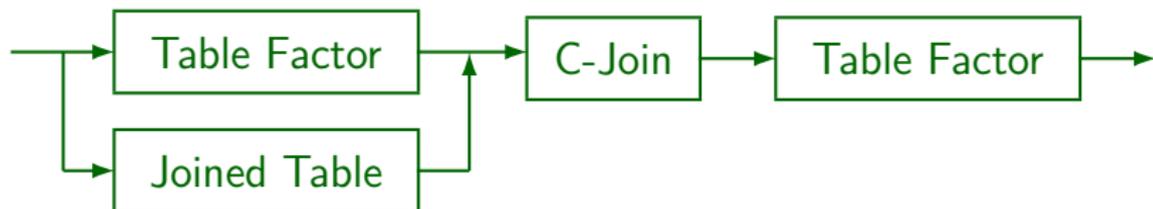
### Joined Table:



- Noch zum „Table Factor“: Man beachte, dass Tupelvariablen bei Tabellennamen optional sind, bei Unteranfragen nötig, und bei geklammerten Join-Ausdrücken nicht erlaubt sind (bei einigen Systemen geht es doch, aber es ist nicht portabel). Tupelvariablen können natürlich für die einzelnen am Join beteiligten Tabellen eingeführt werden.
- „Qualified Join“ ist mit `ON` oder `USING`, s.u.

# Verbundsyntax: Formal (4)

Cross Join:

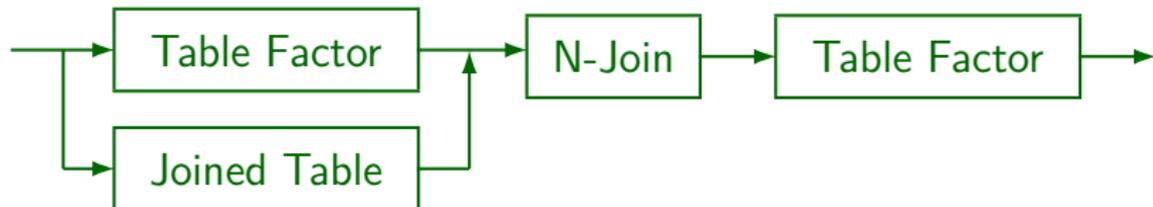


C-Join:



# Verbundsyntax: Formal (5)

Natural Join:

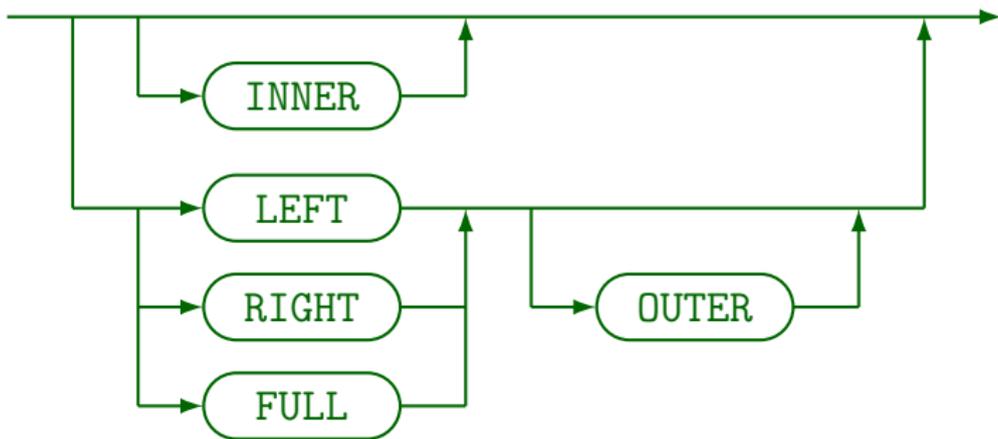


N-Join:



# Verbundsyntax: Formal (6)

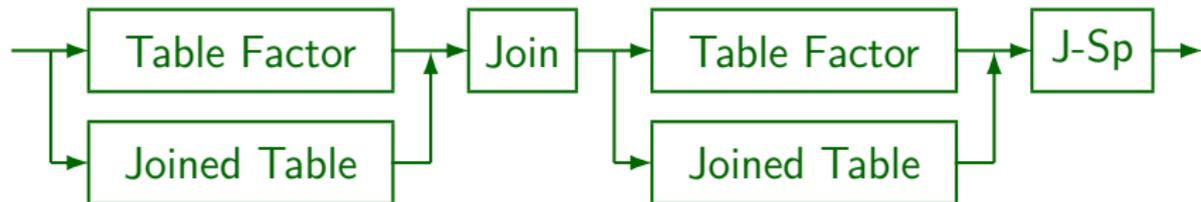
Join Type:



- Der Verbund-Typ ist optional. Wenn kein Typ angegeben ist, wird ein normaler („innerer“) Verbund genommen.

# Verbundsyntax: Formal (7)

## Qualified Join:



## Join:

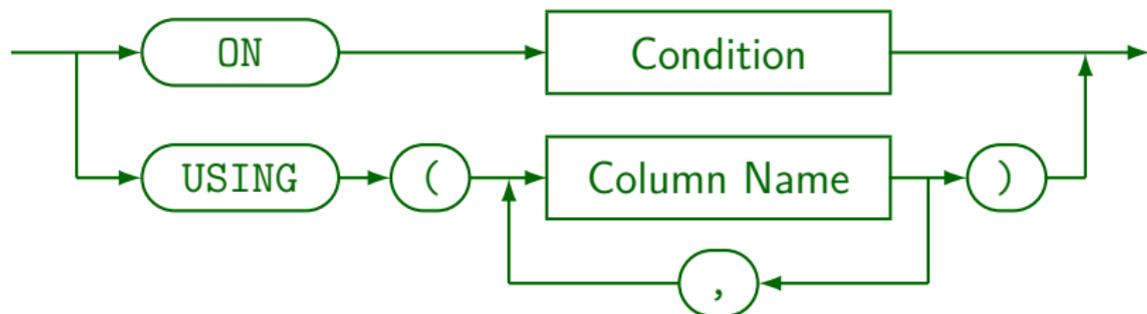


## J-Sp („Join Specification“):

- ON- oder USING-Klausel, siehe nächste Folie.

# Verbundsyntax: Formal (8)

J-Sp („Join Specification“):



## Äußerer Verbund in Oracle (1)

- In Oracle wird der äußere Verbund traditionell unter WHERE spezifiziert (nicht länger nötig ab Version 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 in Oracle (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.