Einführung in Datenbanken

Kapitel 6: Logik-basiertes Kern-SQL: Praxis, Teil I

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

Tupelvariablen (1)

- In der FROM-Klausel deklariert man Variablen, die über alle Tupel der Relation laufen:

```
SELECT A.ANR, A.THEMA
FROM   AUFGABEN A
WHERE  A.ATYP = 'H'
```

- Diese Anfrage kann wie folgt ausgewertet werden:

```
for A in AUFGABEN do
    if A.ATYP = 'H' then
        print A.ANR, A.THEMA
```

- A steht hier für eine Zeile in AUFGABEN (die Schleife weist der Variablen nacheinander jedes Tupel zu).

Tupelvariablen (2)

- Eine Tupelvariable wird immer erstellt: Ist kein Name angegeben, erhält sie den Namen der Relation:

```
SELECT AUFGABEN.ANR, AUFGABEN.THEMA
FROM   AUFGABEN
WHERE  AUFGABEN.ATYP = 'H'
```

- Wenn man also nur `FROM AUFGABEN` schreibt, wird dies behandelt wie:

```
FROM   AUFGABEN AUFGABEN
```

(Die Tupelvariable namens "`AUFGABEN`" läuft über alle Zeilen der Tabelle "`AUFGABEN`".)

Tupelvariablen (3)

- Wird ein Tupelvariablen-Name angegeben, z.B.

  ```
  FROM  AUFGABEN A
  ```

  so ist es ein Fehler, "`AUFGABEN.ANR`" zu schreiben.

  Die Tupelvariable heißt nun "`A`", nicht "`AUFGABEN`".

- Bezieht man sich auf ein Attribut A einer Tupelvariablen X, ist es möglich, einfach A statt $X.A$ zu schreiben, falls X die einzige Tupelvariable ist, die das Attribut A hat.

  Das wird später näher erklärt. Im Beispiel könnte man einfache "`ANR`" für das Attribut schreiben, egal, ob eine Tupelvariable explizit deklariert wurde oder nicht.

Verbunde/Joins (1)

- Gegeben sei eine Anfrage mit zwei Tupelvariablen:

```
SELECT A_1, ..., A_n
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  C
```

- Dann wird `S` über die 4 Tupel in `STUDENTEN` laufen und `B` über die 8 Tupel in `BEWERTUNGEN`. Im Prinzip werden alle $4 * 8 = 32$ Kombinationen betrachtet:

```
for S in STUDENTEN do
    for B in BEWERTUNGEN do
        if C then print A_1, ..., A_n
```

Verbunde/Joins (2)

- Ein gutes DBMS verwendet evtl. einen besseren Algorithmus zur Auswertung der Anfrage (hängt von Bedingung C ab).

  Wenn z.B. C die Verbundbedingung `S.SID = B.SID` enthält, könnte das DBMS alle Tupel in `BEWERTUNGEN` durchgehen und jeweils das zugehörige Tupel in `STUDENTEN` mit Hilfe eines Indexes über `STUDENTEN.SID` finden (die meisten Systeme erstellen automatisch einen Index über Schlüssel).

- Aber um die Bedeutung der Anfrage zu verstehen, reicht der einfache Algorithmus.

  Der Anfrageoptimierer kann jeden Algorithmus verwenden, der das gleiche Ergebnis hat, eventuell in einer anderen Reihenfolge (SQL legt die Reihenfolge der Ergebnistupel nicht fest).

Verbunde/Joins (3)

- Der Verbund muss explizit in der WHERE-Bedingung angegeben werden:

```
SELECT B.ATYP, B.ANR, B.PUNKTE
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID     -- Verbund-Bedingung
AND    S.VORNAME = 'Lisa'
AND    S.NACHNAME = 'Weiss'
```

- Aufgabe: Was wäre das Ergebnis dieser Anfrage?

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

Falsch!

Verbunde/Joins (4)

- Es ist fast immer ein Fehler, wenn es zwei Tupelvariablen gibt, die nicht durch Verbund-Bedingungen verknüpft sind (eventuell indirekt).

  Oder es sind konstante Werte für die Verbund-Attribute gefordert.

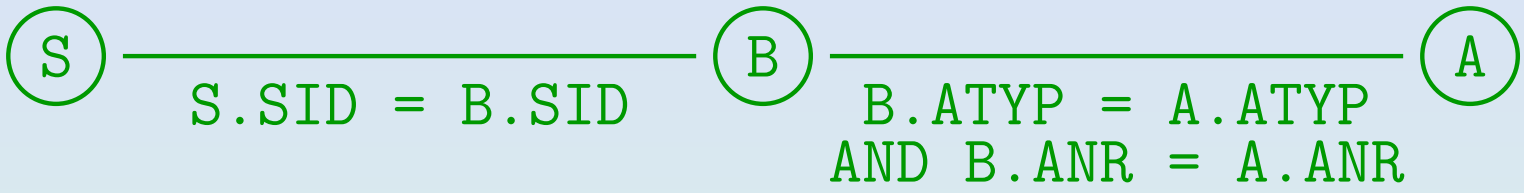
  Der Verbund kann eventuell auch in einer Unteranfrage geschehen.

- Hier sind alle drei Tupelvariablen verbunden:

```
SELECT A.ATYP, A.ANR, B.PUNKTE, A.MAXPT
FROM   STUDENTEN S, BEWERTUNGEN B, AUFGABEN A
WHERE  S.SID = B.SID
AND    B.ATYP = A.ATYP AND B.ANR = A.ANR
AND    S.VORNAME = 'Lisa'
AND    S.NACHNAME = 'Weiss'
```

Verbunde/Joins (5)

- Die Tupelvariablen sind wie folgt verbunden:



- Das entspricht den Schlüssel-Fremdschlüssel-Beziehungen zwischen den Tabellen.

  Die meisten Verbund-Bedingungen sind Gleichheits-Bedingungen zwischen einem Fremdschlüssel und dem Schlüssel der referenzierten Tabelle.

- Wenn man eine Verbund-Bedingung vergisst, wird man oft viele Duplikate erhalten.

  Dann wäre es falsch `DISTINCT` anzuwenden, ohne über den Grund der Duplikate nachzudenken.

Anfrageformulierung (1)

- Aufgabe: "Geben Sie die Themen aller von Lisa Weiss gelösten Aufgaben aus."
- Lisa Weiss ist eine Studentin, daher sind eine Tupelvariable `S` über `STUDENTEN` und folgende Bedingung nötig:
  `S.VORNAME = 'Lisa' AND S.NACHNAME = 'Weiss'`
- Aufgaben-Themen werden verlangt, so dass eine Tupelvariable `A` über `AUFGABEN` benötigt wird.
  Folgender Teil kann bereits erstellt werden:

```
SELECT DISTINCT A.THEMA
```

"`DISTINCT`", da viele Aufgaben das gleiche Thema haben können.

Anfrageformulierung (2)

- Schließlich sind S und A nicht verbunden.
- Es kann helfen, einen Verbindungsgraphen der Tabellen, basierend auf gemeinsamen Spalten (Fremdschlüssel), zu zeichnen:

- Man sieht, dass eine Tupelvariable über BEWERTUNGEN benötigt wird mit folgender Verbund-Bedingung:

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

Anfrageformulierung (3)

- Es ist nicht immer so einfach. Der Verbindungsgraph kann Zyklen enthalten, die die Wahl des richtigen Pfades erschwert.
- Betrachten Sie z.B. eine Datenbank zur Belegung von Vorlesungen, die auch Hiwis beinhaltet.

  Hiwis sind oft fortgeschrittene Studenten, die bei der Korrektur von Hausaufgaben usw. helfen.

QBE als Hilfe zur Anfrageformulierung (1)

- "Query by Example" (QBE) ist oder war eine Anfragesprache, bei der man die Anfrage über Einträge in Tabellen formuliert hat.

  Die Sprache wurde von Moshé M. Zloof bei IBM parallel zu System R entwickelt (1975). Sie basiert auf dem Bereichskalkül, während SQL auf dem Tupelkalkül basiert.

- Es war Ideengeber für viele graphische Schnittstellen zu Datenbanken.

  Z.B. in Microsoft Access.

- Selbst wenn man am Ende eine SQL-Anfrage formulieren muss, kann es hilfreich sein, die Anfrage QBE-ähnlich zu planen.

  Zumindest, wenn man Schwierigkeiten mit SQL hat.

QBE als Hilfe zur Anfrageformulierung (2)

- Beispiel: "Geben Sie alle Studenten aus, die 10 Punkte in Aufgabe 1 und mindestens 8 Punkte in Aufgabe 2 haben."

  Gesucht sind Vorname und Nachname der Studenten.

- Man überlegt sich nun ein Beispiel für Tabellenzeilen, die zu einer Antwort führen würden:

STUDENTEN		
SID	VORNAME	NACHNAME
101	Lisa	Weiss

BEWERTUNGEN			
SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8

Nicht relevante Spalten kann man weglassen, hier z.B. `EMAIL` in `STUDENTEN`.

- Diese drei Zeilen sind nötig, um die Ausgabe "Lisa Weiss" zu erzeugen.

QBE als Hilfe zur Anfrageformulierung (3)

- Nun muss man von dem konkreten Beispiel verallgemeinern:
  - Vorname und Nachname des Studierenden können beliebig sein, und müssen ausgegeben werden.
    In QBE werden auszugebene Einträge mit "P." markiert.
  - Der Wert 101 in den SID-Spalten ist nur ein Beispiel.
    Es kann eine beliebige Zahl sein, aber es muss in allen drei Zeilen der gleiche Wert sein. In QBE werden Variablen mit einem vorangestellten "_" gekennzeichnet.
  - Für Hausaufgabe 2 müssen es mindestens 8 Punkte sein.

STUDENTEN		
SID	VORNAME	NACHNAME
_X	P.	P.

BEWERTUNGEN			
SID	ATYP	ANR	PUNKTE
_X	H	1	10
_X	H	2	>= 8

QBE als Hilfe zur Anfrageformulierung (4)

- Als Variablennamen werden in QBE Beispielwerte mit vorangestelltem "_" empfohlen, z.B. "_101".

  Die Variablen laufen über Datenwerten. QBE-Anfragen lassen sich ganz direkt in den Bereichskalkül übersetzen. Die Übersetzung nach SQL ist mühsamer.

- Für die Übersetzung nach SQL muss man für jede Zeile eine Tupelvariable anlegen, z.B. `S` über Studenten, und `H1` und `H2` über `BEWERTUNGEN`.

- Tabelleneinträge mit der gleichen QBE-Variable müssen über Gleichheitsbedingungen gleichgesetzt werden.

  Kommt die Variable in n Tabelleneinträgen vor, braucht man $n-1$ Gleichheitsbedingungen (jeweils ein Vorkommen muss gleich dem nächsten Vorkommen sein, wobei man irgendeine Reihenfolge der Vorkommen wählen kann). Der Rest folgt über die Transitivität der Gleichheit.

QBE als Hilfe zur Anfrageformulierung (5)

- Für alle Datenwerte in den Beispielzeilen braucht man Gleichheitsbedingungen mit den entsprechenden Konstanten.

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S,
       BEWERTUNGEN H1, BEWERTUNGEN H2
WHERE  S.SID = H1.SID AND H1.SID = H2.SID
AND    H1.ATYP = 'H' AND H1.ANR = 1
AND    H1.PUNKE = 10
AND    H2.ATYP = 'H' AND H2.ANR = 2
AND    H2.PUNKTE >= 8
```

- Für komplexe Bedingungen (z.B. Disjunktionen) hatte QBE noch eine "Condition Box".

Unnötige Joins (1)

- Verbinden Sie nicht mehr Tabellen als nötig.

  Anfragen laufen langsamer: Viele Optimierer entfernen keine Joins.

- Z.B Ergebnisse für Hausaufgabe 1:

```
SELECT B.SID, B.PUNKTE
FROM   BEWERTUNGEN B, AUFGABEN A
WHERE  B.ATYP = A.ATYP AND B.ANR = A.ANR
AND    A.ATYP = 'H' AND A.ANR = 1
```

- Kann diese Anfrage je ein anderes Ergebnis liefern?

```
SELECT B.SID, B.PUNKTE
FROM   BEWERTUNGEN B
WHERE  B.ATYP = 'H' AND B.ANR = 1
```

Unnötige Joins (2)

- Was ist das Ergebnis dieser Anfrage?

```
SELECT B.SID, B.PUNKTE
FROM   BEWERTUNGEN B, AUFGABEN A
WHERE  B.ATYP = 'H' AND B.ANR = 1
```

- Unterscheiden sich die folgenden zwei Anfragen?

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S
```

```
SELECT DISTINCT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
```

Selbstverbund (1)

- Es ist möglich, dass mehr als ein Tupel derselben Relation benötigt wird, um ein bestimmtes Ergebnis zu erhalten.
- Gibt es einen Studenten, der in Hausaufgabe 1 und in Hausaufgabe 2 jeweils 10 Punkte hat?

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN H1, BEWERTUNGEN H2
WHERE  S.SID = H1.SID AND S.SID = H2.SID
AND    H1.ATYP = 'H' AND H1.ANR = 1
AND    H2.ATYP = 'H' AND H2.ANR = 2
AND    H1.PUNKTE = 10 AND H2.PUNKTE = 10
```

Selbstverbund (2)

- Studenten, die mind. zwei Aufgaben gelöst haben:

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN A1, BEWERTUNGEN A2
WHERE  S.SID = A1.SID AND S.SID = A2.SID
```

Falsch!

- Die Tupelvariablen `A1` und `A2` können auf das gleiche Tupel in `BEWERTUNGEN` zeigen.
- Man muss verlangen, dass sie verschieden sind:

```
WHERE S.SID = A1.SID AND S.SID = A2.SID
AND   (A1.ATYP <> A2.ATYP OR A1.ANR <> A2.ANR)
```

- Man kann dies aber auch mit Aggregationen lösen.

Übung

- Erkennen Sie das Problem in dieser Anfrage?
  Ziel ist es, alle Studenten auszugeben, die eine Aufgabe über SQL und eine über ER-Entwurf gelöst haben.

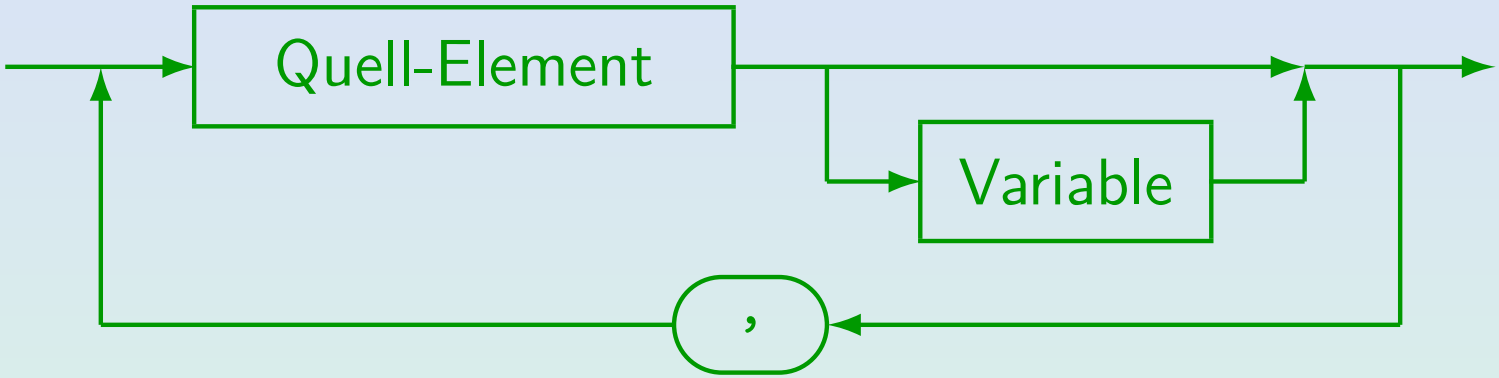
```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B,
       AUFGABEN A1, AUFGABEN A2
WHERE  S.SID = B.SID
AND    B.ATYP = A1.ATYP AND B.ANR = A1.ANR
AND    B.ATYP = A2.ATYP AND B.ANR = A2.ANR
AND    A1.THEMA = 'SQL'
AND    A2.THEMA = 'ER'
```

Join-Fehler

- Fehlende Join-Bedingungen (sehr häufig)
- Unnötige Joins (machen Anfrage langsamer)
- Wenn eigentlich mehrere Tupelvariablen über einer Relation benötigt werden, und man "verschmilzt sie", so erhält man oft inkonsistente Bedingungen.
- Duplikate sind oft ein Zeichen für Fehler: Man sollte die Ursache der Duplikate verstehen und nicht einfach `DISTINCT` anwenden, um das Problem zu vermeiden (eigentlich nur zu verdecken).

FROM-Syntax (1)

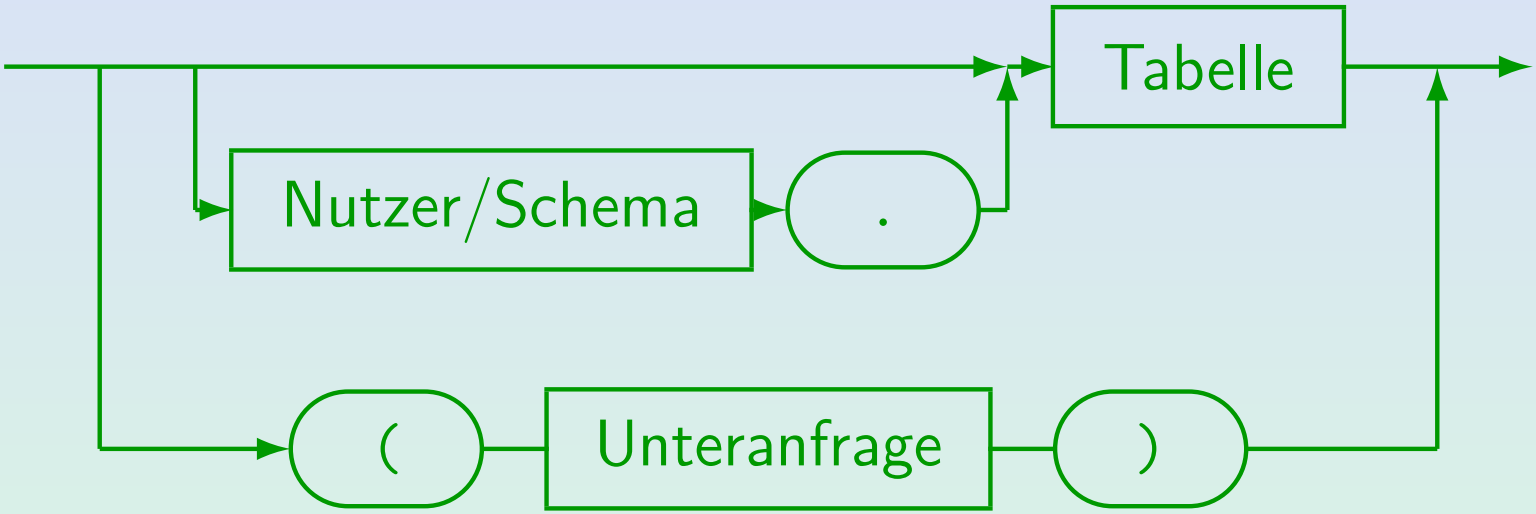
Quell-Liste (nach FROM):



- In SQL-92, SQL Server, Access, DB2 und MySQL (nicht in Oracle 8i) kann man "`AS`" zwischen Quell-Element und Variable schreiben.
- In SQL-92, DB2 (nicht Oracle, SQL Server, Access, MySQL) kann man neue Spaltennamen definieren: "`STUDENTEN AS S(NR,VNAME,NNAME,MAIL)`".
- Ist das "Quell-Element" eine Unteranfrage, wird in SQL-92, SQL Server und DB2 eine Tupelvariable verlangt (aber nicht Oracle, Access). Dann funktioniert obige Spaltenumbenennung plötzlich auch in SQL Server.
- SQL-92, SQL Server, Access, DB2 unterstützen Joins unter `FROM` (später).

FROM-Syntax (2)

Quell-Element:



- SQL-86 erlaubt keine Unteranfragen in der `FROM`-Liste.
- MySQL unterstützt Unteranfragen erst ab Version 4.1.
- Vereinfachte Syntax der `FROM`-Klausel:

```
FROM Tabelle [Variable], ..., Tabelle [Variable]
```

FROM-Syntax (3)

Tabellennamen:

- Man kann sich auf Tabellen anderer Nutzer unter `FROM` beziehen (falls Leserecht erteilt wurde):

```
SELECT * FROM BRASS.AUFGABEN
```

- Der Nutzername ist hier der Name des DB-Schemas (ein DBMS kann mehrere Schemata verwalten).

  In Oracle sind Nutzer und Schema mehr oder weniger das gleiche: Jeder Nutzer hat sein eigenes Schema, jedes Schema gehört genau einem Nutzer. In DB2 kann es mehrere Schemata je Nutzer geben (man kann "Schema.Tabelle" schreiben). In SQL Server hat ein vollständiger Name die Form "Server.DB.Inhaber.Tabelle", aber es gibt viele Abkürzungen, z.B. "Inhaber.Tabelle" oder "Tabelle". In MySQL kann man "DB.Tabelle" schreiben.

Inhalt

Terme (1)

- Ein Term bezeichnet ein Datenelement.

  Der Begriff "Term" wird in der Logik verwendet. In Programmiersprachen sagt man "Ausdruck". Der SQL-Standard verwendet "skalarer Ausdruck", weil es dort auch "Tabellenausdrücke" gibt.

- Terme sind:
  - Attribut-Referenzen, z.B. `STUDENTEN.SID`.
  - Konstanten ("Literale"), z.B. `'Lisa'`, `1`.
  - Zusammengesetzte Terme, z.B. `0.9 * MAXPT`.

    Zusammengesetzte Terme bestehen aus Teil-Termen, die verknüpgt werden mit Datentyp-Operatoren wie `+`, `-`, `*`, `/` (für Zahlen), `||` (String-Konkatenation) und Datentyp-Funktionen wie `SIN`.
  - Aggregations-Terme, z.B. `MAX(PUNKTE)`: siehe Teil 11.

Terme (2)

- Terme verwendet man in Bedingungen, z.B. enthält

  ```
  B.PUNKTE > A.MAXPT * 0.8
  ```

  die Terme "`B.PUNKTE`" und "`A.MAXPT * 0.8`".

- Auch `SELECT`-Liste kann beliebige Terme enthalten:

```
SELECT NACHNAME || ', ' || VORNAME
FROM   STUDENTEN
```

...
Weiss, Lisa
Grau, Michael
Sommer, Daniel
Winter, Iris

Attribut-Referenzen (1)

- Auf Attribute kann man in dieser Form zugreifen:

  `Variable.Attribut`

- Hat nur eine Variable das Attribut, kann der Variablenname fehlen. Z.B. ist diese Anfrage legal:

```
SELECT ATYP, ANR, PUNKTE
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    VORNAME = 'Lisa' AND NACHNAME = 'Weiss'
```

  "VORNAME" und "NACHNAME" gibt es nur in "S", "ATYP", "ANR" und "PUNKTE" nur in "B". "SID" allein wäre jedoch mehrdeutig, da sowohl "S" als auch "B" ein Attribut mit diesem Namen haben.

Attribut-Referenzen (2)

- Gegeben sei diese Anfrage:

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

Falsch!

- SQL verlangt, dass der Nutzer festlegt, ob er `B.ANR` oder `A.ANR` unter `SELECT` auswählt, obwohl beide gleich sind, so dass es eigentlich egal wäre.

  Die Regel ist rein syntaktisch: Hat mehr als eine Tupelvariable in der `FROM`-Klausel das Attribut "`ANR`", darf die Tupelvariable nicht fehlen oder das DBMS (z.B. Oracle) wird den Fehler "ORA-00918: column ambiguously defined" ausgeben. DB2, SQL Server, Access, MySQL sind auch so streng.

Zusammengesetzte Terme (1)

- Der SQL-86-Standard enthielt nur `+`, `-`, `*`, `/`.
- Derzeitige DBMS unterscheiden sich immer noch in anderen Datentyp-Operationen.

  Aber sie haben meist eine große Auswahl an Datentyp-Operationen, z.B. `sin`, `cos`, `substr`. Kapitel 4 enthält Listen von Datentyp-Operationen für verschiedene Systeme.
- Z.B. ist der Operator `||` im SQL-92-Standard enthalten, aber funktioniert z.B. nicht in SQL Server.

  String-Konkatenation wird in SQL Server und Access "+" geschrieben. In MySQL muss man "`concat(`s_1`,` s_2`)`" schreiben (aber es gibt `--ansi`). Andere Datentyp-Funktionen (z.B. `SUBSTR`) sind sogar noch weniger standardisiert.

Zusammengesetzte Terme (2)

- SQL kennt die Standard-Vorrangregeln,
  z.B. bedeutet `A+B*C`:

  `A+(B*C)`,

  und nicht

  `(A+B)*C`.

- Klammern `(...)` können verwendet werden, um eine bestimmte Struktur zu erzwingen.

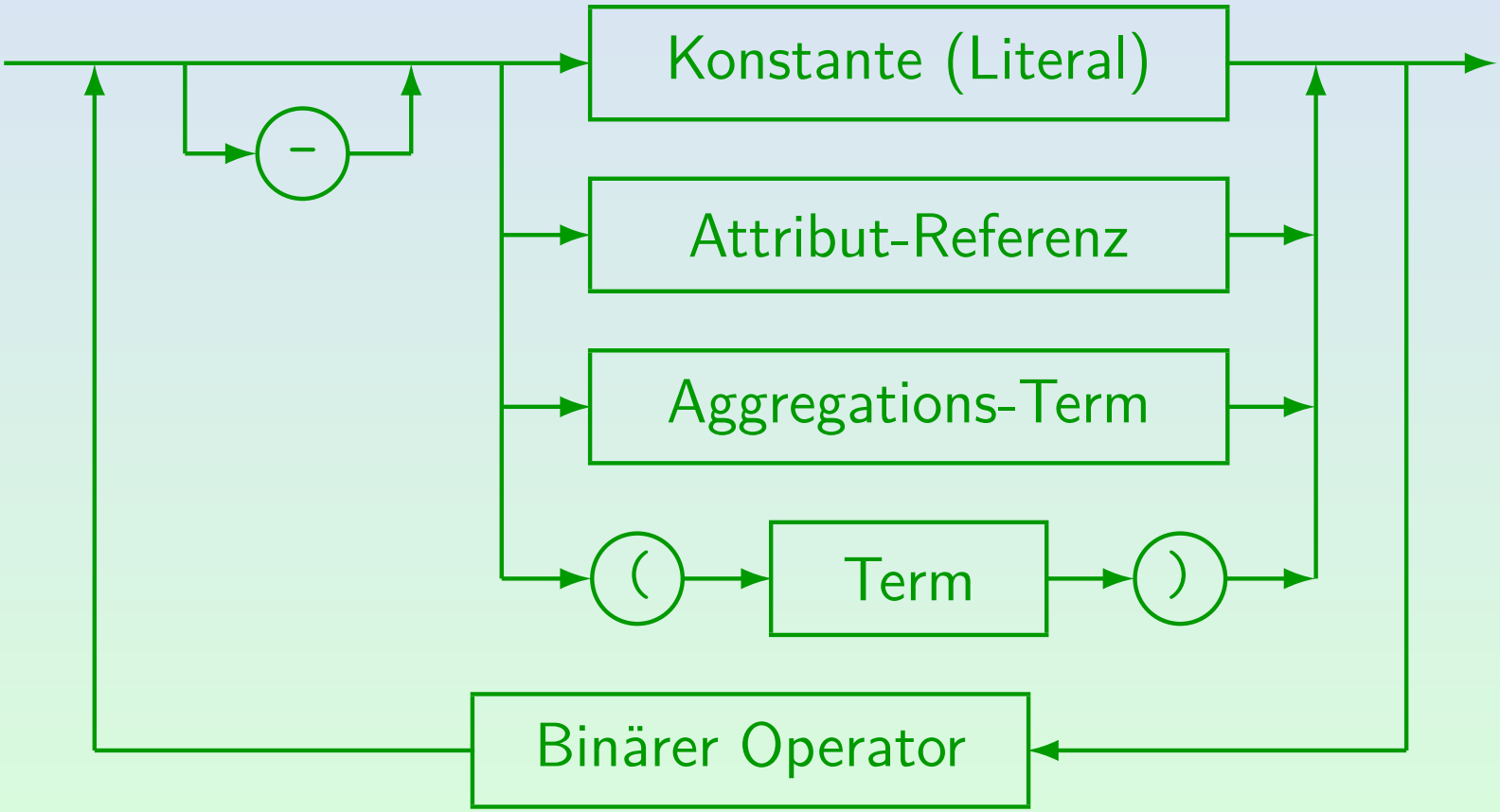
- Übung: Was ist das Ergebnis von `7+3*2-4-1`?

  Es kann nützlich sein, einen Operator-Baum zu zeichnen.

  "`-`" ist links-assoziativ (von links ausgewertet).

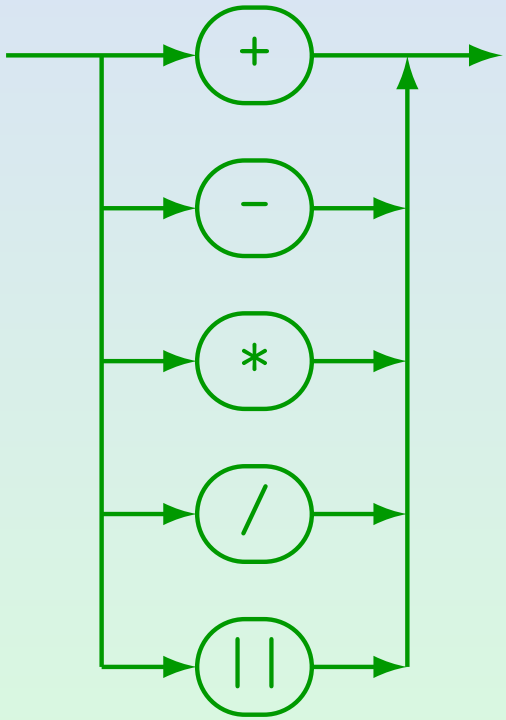
Terme: Syntax (1)

Term (Skalarer Ausdruck, Wert-Ausdruck):



Terme: Syntax (2)

Binärer Operator:



- SQL Server, Access, MySQL verwenden nicht "||" für die Konkatenation.

Inhalt

Bedingungen (1)

- Bedingungen bestehen aus atomaren Formeln, z.B.

  ```
  PUNKTE >= 8,
  ```

  verbunden mit "AND", "OR", "NOT".

- AND bindet stärker als OR, somit wird

  ```
  ATYP = 'H' AND ANR = 1 OR ANR = 2
  ```

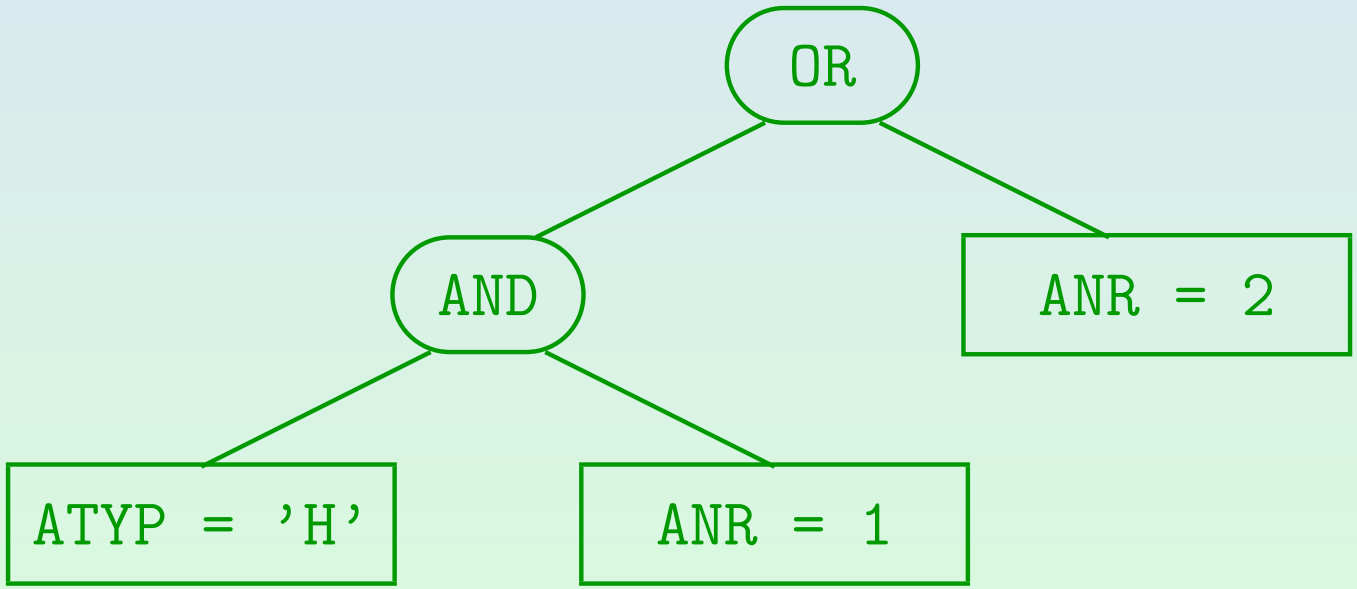
  implizit so geklammert:

  ```
  (ATYP = 'H' AND ANR = 1) OR ANR = 2
  ```

- In diesem Beispiel ist dies jedoch nicht gewünscht.

Bedingungen (2)

- Es kann helfen, komplexe Bedingungen oder Terme als "Operator-Baum" darzustellen:



Bedingungen (3)

- `NOT` bindet am stärksten, d.h. es gilt nur für die direkt folgende Bedingung (atomare Formel).
- Klammern `( ...)` können verwendet werden, um die Bindungsstärken / Prioritäten der Operatoren aufzuheben.
- Manchmal ist es klarer, Klammern zu verwenden, auch wenn sie nicht nötig wären, um die richtige Struktur der Bedingung zu erhalten.

  Anfänger neigen jedoch dazu, viele Klammern zu verwenden (wahrscheinlich weil sie sich über die Bindungsstärken nicht sicher sind). Das macht die Formel nicht verständlicher.

Bedingungen (4)

- Die `WHERE`-Bedingung wird für jede Kombination von Zeilen der unter `FROM` stehenden Tabellen ausgewertet. Ist sie wahr, wird die `SELECT`-Liste ausgegeben.
- Eine `AND`-Bed. ist wahr, wenn beide Teile wahr sind, eine `OR`-Bed. ist wahr, wenn ein Teil wahr ist:

B1	B2	B1 and B2	B1 or B2	not B1
falsch	falsch	falsch	falsch	wahr
falsch	wahr	falsch	wahr	wahr
wahr	falsch	falsch	wahr	falsch
wahr	wahr	wahr	wahr	falsch

Bedingungen (5)

- "Geben Sie die SIDs von Lisa und Iris aus".

```
SELECT SID
FROM   STUDENTEN
WHERE  VORNAME = 'Lisa' AND VORNAME = 'Iris'
```

Falsch!

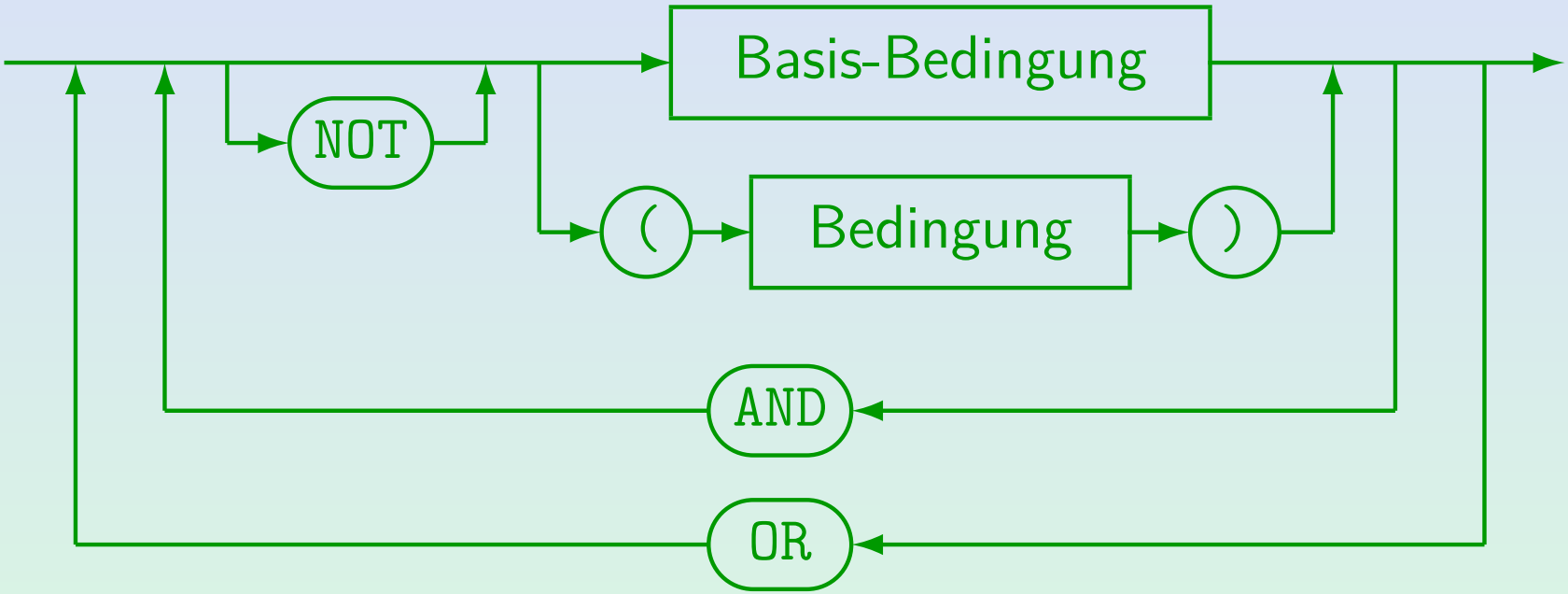
STUDENTEN		
SID	VORNAME	NACHNAME
101	Lisa	Weiss
102	Michael	Grau
103	Daniel	Sommer
104	Iris	Winter

VORN.='Lisa'	VORN.='Iris'	WHERE
wahr	falsch	falsch
falsch	falsch	falsch
falsch	falsch	falsch
falsch	wahr	falsch

- Die obige Bedingung ist inkonsistent.
  Hier muss "OR" verwendet werden, nicht "AND".

Bedingungen (6)

Bedingung:



- SQL-92 erlaubt "`IS NOT TRUE`", "`IS FALSE`" usw. nach Formeln (nicht in Oracle 8.0, SQL Server, DB2, MySQL, Access unterstützt).
- Die syntaktische Kategorie "Basis-Bedingung" enthält natürlich insbesondere die atomaren Formeln, aber auch Bedingungen mit Unteranfragen (entsprechen quantifizierten Formeln).

Bedingungen (7)

- AND und OR müssen auf beiden Seiten vollständige logische Bedingungen haben (etwas, das wahr oder falsch ist).
- Somit ist Folgendes ein Syntaxfehler, obwohl es in der natürlichen Sprache erlaubt wäre:

```
SELECT DISTINCT SID          Falsch!
FROM   BEWERTUNGEN
WHERE  ATYP = 'H' AND PUNKTE >= 9
AND    ANR = 1 OR 2
```

- Ausnahme: `... BETWEEN ... AND ...`

Hier bezeichnet das Wort AND keinen logischen Operator.

Bedingungen (8)

Basis-Bedingung:

Vergleiche (1)

Bedingung (Form 1: Vergleich):

- Vergleichsoperatoren: `=`, `<>`, `<`, `>`, `<=`, `>=`.
- Man kann sie sowohl für Zahlen als auch für Strings verwenden, z.B.: `PUNKTE >= 8`, `NACHNAME < 'M'`.
- "Ungleich" wird in SQL als "`<>`" geschrieben.

  Oracle, SQL Server, DB2 und MySQL verstehen auch "`!=`" (Access nicht). "`^=`" funktioniert in Oracle und DB2, aber nicht in SQL Server, Access oder MySQL.

Vergleiche (2)

- Zahlen werden anders verglichen als Zeichenketten, z.B. `3 < 20`, aber `'3' > '20'`.

  Strings werden Zeichen für Zeichen verglichen, bis das Ergebnis klar ist. In diesem Fall kommt "3" alphabetisch nach "2", daher ist der Rest der Zeichenkette nicht wichtig.

- Nach dem SQL-92-Standard ist es falsch, Zeichenketten mit Zahlen zu vergleichen, z.B. `3 > '20'`.

  Die verglichenen Werte müssen von kompatiblen Datentypen sein: Alle numerischen Typen sind kompatibel und alle String-Typen ebenfalls, aber numerische Typen sind nicht kompatibel mit String-Typen.

Vergleiche (3)

- Vergleiche zwischen Strings und Zahlen sollten vermieden werden (Ergebnis systemabhängig):
  - In SQL-92, DB2 und Access ist es ein Typfehler.
  - Oracle, MySQL, SQL Server konvertieren den String in eine Zahl und vergleichen numerisch.

    Hat der String kein numerisches Format, konvertiert ihn MySQL in 0. Z.B. ist `0 = 'abc'` in MySQL wahr. In Oracle und SQL Server erhält man in diesem Fall jedoch einen Fehler. Das kann ein Laufzeitfehler sein, wenn der String ein Spaltenwert ist.
  - Wird jedoch eine Spalte mit einer Konstanten verglichen, nimmt SQL Server den Spaltentyp.

    Aggregations-Funktionen haben noch höhere Priorität als Spalten.

Zeichenkettenvergleich (1)

- Das Ergebnis eines Vergleichs (=, <>, <, <=, >, >=) zweier Zeichenketten kann vom DBMS abhängen.

  Oder von Einstellungen innerhalb des DBMS.

- Der SQL-92-Standard definiert den Begriff "collation sequences", der Folgendes festlegt:
  - für jedes Paar X und Y von Zeichen, ob $X < Y$, $X = Y$ oder $X > Y$ und
  - ob blank-padded-Semantik (`PAD SPACE`) oder non-padded-Semantik (`NO PAD`) verwendet wird.

Zeichenkettenvergleich (2)

- `'a' < 'b'` usw. und `'A' < 'B'` usw. sollten in jedem System gelten.
- Die Systeme unterscheiden sich schon im Vergleich von Klein- und Großbuchstaben. Die Defaults sind:
  - In Oracle kommen alle Großbuchstaben vor den Kleinbuchstaben (ASCII), z.B. `'Z' < 'a'`.
  - In DB2 liegen die Großbuchstaben zwischen den Kleinbuchstaben, z.B. `'a' < 'A'`, `'A' < 'b'`.
  - SQL Server, MS Access und MySQL sind case-insensitive, z.B. `'a' = 'A'`.

Zeichenkettenvergleich (3)

- Manchmal kann man dies ändern, aber z.B. nur während der Installation (SQL Server) oder während der DB-Erstellung (Oracle, DB2).
- Ist die Reihenfolge (<, =, >) zweier Zeichen bekannt, so ist der Vergleich von Zeichenketten der gleichen Länge klar:
  - Das System vergleicht Zeichen für Zeichen und der erste Vergleich, der nicht "=" ergibt, bestimmt das Ergebnis.

    DB2 macht zwei Schritte: Es vergleicht erst character "weights" und wenn es keinen Unterschied gibt, auch die character codes.

Zeichenkettenvergleich (4)

- Für Zeichenketten veschiedener Länge gibt es
  - Non-Padded Vergleichs-Semantik:
    Z.B. `'a' < 'a '`.

    Strings werden Zeichen für Zeichen verglichen. Endet ein String und es wurde kein Unterschied gefunden, gilt der kürzere String als kleiner.
  - Blank-Padded Vergleichs-Semantik:
    Z.B. `'a' = 'a '`.

    Der kürzere String wird vor dem Vergleich mit `' '` aufgefüllt.

Zeichenkettenvergleich (5)

- DB2, SQL Server, Access und MySQL verwenden blank-padded Semantik (zumindest als Default).
- Oracle hat non-padded Semantik, wenn mindestens ein Operand des Vergleichs den Typ `VARCHAR2` hat.

  Oracle hat einen Typ `VARCHAR2(`n`)` eingeführt. Er ist derzeit äquivalent zu `VARCHAR(`n`)`, aber Oracle beabsichtigt, die Vergleichs-Semantik für `VARCHAR` zu ändern, wobei die Semantik für `VARCHAR2` bleibt wie bisher. String-Konstanten in der Anfrage haben den Typ `CHAR(`n`)`. Z.B. kann ein Vergleich von `CHAR(10)`- und `CHAR(20)`-Spalten möglicherweise wahr sein, sowie ein Vergleich dieser Spalten mit z.B. `'abc'`. Aber `CHAR(10)` und `VARCHAR(20)` können nur gleich sein, wenn der `VARCHAR` zufällig 10 Zeichen hat. Leerzeichen am Stringende in `VARCHAR2`-Spalten sind ein Problem: unsichtbar in der Ausgabe, aber können "=" verhindern.

Zeichenkettenvergleich (6)

- Verwendet das DBMS eine case-sensitive Semantik, bekommt man einen case-insensitiven Vergleich, indem man alles in z.B. Großbuchstaben konvertiert:

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN
WHERE  UPPER(EMAIL) = UPPER('xyz@hotmail.com')
```

- `UPPER` funktioniert in SQL-92, Oracle, SQL Server, DB2, MySQL. In Access nimmt man `UCASE`.

  `UCASE` funktioniert auch in DB2 und MySQL. Das Buch von Chamberlin über DB2 beschreibt nur `UCASE`.

Zeichenkettenvergleich (7)

- Der umgekehrt Fall (case-sensitiver Vergleich mit case-insensitivem DBMS) ist schwieriger.

  Aber auch viel seltener erforderlich.

- Z.B. kann man in MySQL einen String in einen binären String konvertieren, um einen case-sensitiven Vergleich zu machen:

```
BINARY EMAIL = 'xyz@hotmail.com'
```

- Das gleiche funktioniert auch in SQL Server:

```
  CAST(EMAIL AS VARBINARY(255))
= CAST('...' AS VARBINARY(255))
```

Zeichenkettenvergleich (8)

- Vermutet man angehängte Leerzeichen, kann man sie so sichtbar machen:

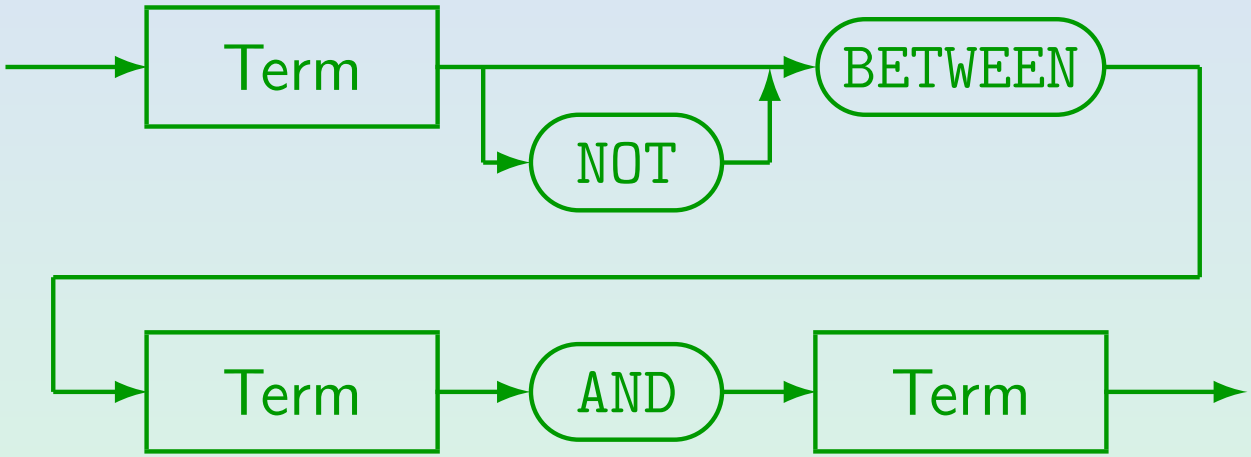
```
SELECT '"' || NACHNAME || '"' AS NACHNAME
FROM   STUDENTEN
```

- Man kann angehängte Leerzeichen auch löschen:
  - `TRIM(TRAILING ' ' FROM NACHNAME)`
    in SQL-92 (funktioniert in MySQL)

    Wird in Oracle, DB2, SQL Server, Access nicht unterstützt.
  - `RTRIM(NACHNAME)`
    in Oracle, DB2, SQL Server, MySQL, Access.

BETWEEN-Bedingungen

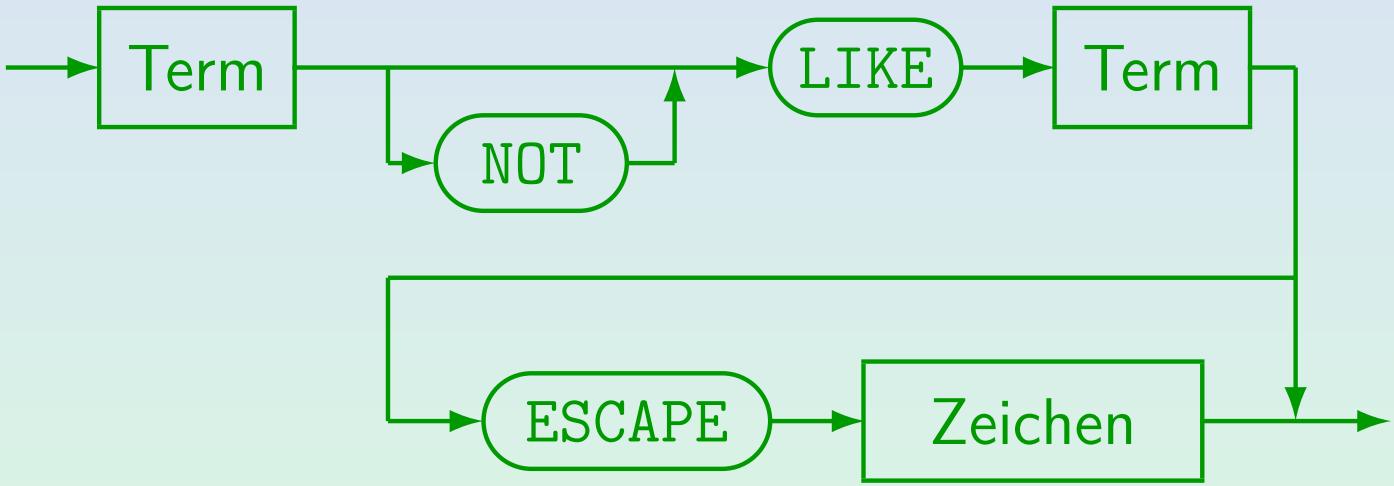
Bedingung (Form 2: BETWEEN):



- `x BETWEEN y AND z` ist äquivalent zu
  `x >= y AND x <= z`.
- Z.B.: `PUNKTE BETWEEN 5 AND 8`

LIKE-Bedingungen (1)

Bedingung (Form 3: LIKE):



- Z.B.: `EMAIL LIKE '%.pitt.edu'`

  Das ist für alle Email-Adressen wahr, die mit "`.pitt.edu`" enden.

LIKE-Bedingungen (2)

- Das rechte Argument wird als Muster interpretiert. In SQL-86 und in DB2 muss dies eine String-Konstante sein.

  In Oracle, SQL Server, Access und MySQL kann man jeden stringwertigen Term als Muster verwenden (z.B. auch eine andere Spalte).

- "%" im Muster ersetzt eine Folge beliebiger Zeichen (den leeren String eingeschlossen).

  Im UNIX-Shell (Kommando-Interpreter) wird "*" statt "%" verwendet.

- "_" passt auf ein beliebiges einzelnes Zeichen.

  Dies entspricht "?" im Shell.

LIKE-Bedingungen (3)

- LIKE muss zur Mustersuche verwendet werden. Das Gleichheitszeichen überprüft nur Zeichengleichheit.

  Auch wenn der Vergleichs-String "%" oder "_" enthält.

- Z.B. ist Folgendes in SQL legal, wird aber das falsche Ergebnis liefern (im Beispiel $\emptyset$):

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN
WHERE  NACHNAME = 'S%'
```

Falsch!

LIKE-Bedingungen (4)

- Um die Zeichen "%" und "_" ohne ihre spezielle Bedeutung im Muster zu verwenden, wird ein "Escape"-Zeichen benötigt.

  Ein Escape-Zeichen löscht die spezielle Bedeutung des darauffolgenden Zeichens. Wenn z.B. "\" das Escape-Zeichen ist, so ist "\%" nur ein Prozentzeichen, und kein beliebiger String.

- Das Escape-Zeichen muss deklariert werden, z.B.:

```
PROZNAME LIKE '\_%' ESCAPE '\'
```

  Dies gibt alle Prozeduren aus, die mit "_" beginnen.

  In MySQL ist "\" der Default, wenn kein Escape-Zeichen deklariert wurde. Dies verletzt jedoch den SQL-92-Standard.

LIKE-Bedingungen(5)

- `LIKE` verwendet die non-padded-Semantik.

  Oracle, DB2, MySQL, Access verwenden die non-padded-Semantik, wie im SQL-92-Standard verlangt. Man beachte, dass MySQL angehängte Leerzeichen entfernt, wenn Strings gespeichert werden. Alle Systeme füllen Werte mit Leerzeichen auf, wenn die Spalte den Typ Zeichenkette mit fester Länge hat.
  In SQL Server stimmt es evtl. auch dann überein, wenn der gespeicherte String mehr Leerzeichen als das Muster hat. Enthält das Muster mehr Leerzeichen, schlägt der Vergleich fehl.

- Z.B. ist `'a' = 'a '` in manchen DBMS wahr, aber `'a' LIKE 'a '` ist mit Sicherheit falsch.

- Die Case-Sensitivität ist die gleiche wie für gewöhnliche Vergleiche.

Reguläre Ausdrücke

- SQL Server und Access unterstützen auch Zeichenbereiche, z.B. `[a-zA-Z]` in `LIKE`-Bedingungen.

  Dies verletzt den Standard.

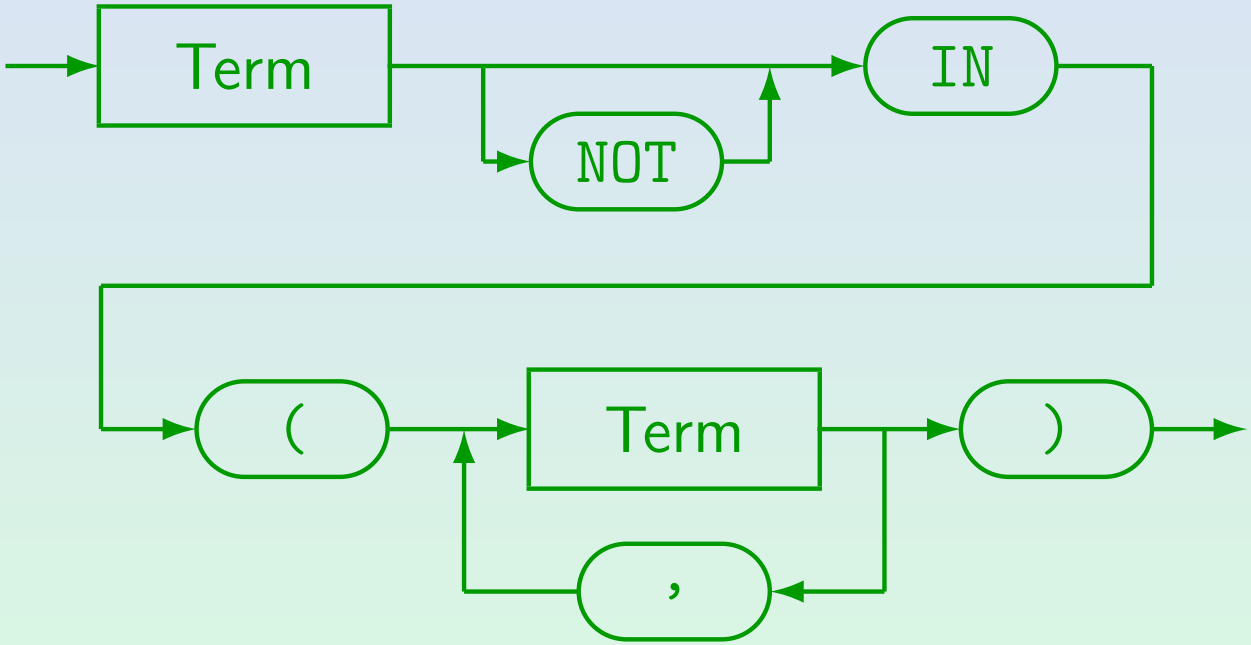
- MySQL hat einen zusätzlichen Operator "`RLIKE/REGEXP`", der beliebige reguläre Ausdrücke als Muster akzeptiert.

- Der SQL:1999-Standard führte ein Prädikat "`SIMILAR TO`" ein, das Vergleiche mit regulären Ausdrücken durchführt.

  In den meisten Systemen, z.B. Oracle 9i, noch nicht implementiert.

IN-Bedingungen (1)

Bedingung (Form 4: IN mit Aufzählung):



IN-Bedingungen (2)

- Z.B. `ATYP IN ('Z', 'E')`
- Dies ist äquivalent zu

```
ATYP = 'Z' OR ATYP = 'E'
```

- Der SQL-86-Standard erlaubt nur Konstanten in der Aufzählung der Werte.

  SQL-92, Oracle, SQL Server und DB2 erlauben beliebige Terme, aber es ist normalerweise besserer Stil, wenn man `OR` verwendet, falls die Menge keine Aufzählung von Konstanten ist.

- Man beachte, dass "`(...)`" hier eine "Menge" ist (obwohl in Mathematik für Intervalle verwendet).

Inhalt

SELECT-Klausel, *

- SELECT legt die Terme fest, die ausgegeben werden, falls die WHERE-Bedingung wahr ist (Ergebnis-Spalten).
- SELECT * kann verwendet werden, um alle Spalten der Tabelle(n) unter FROM auszugeben, z.B. ist

```
SELECT *
FROM   STUDENTEN
```

äquivalent zu

```
SELECT SID, VORNAME, NACHNAME, EMAIL
FROM   STUDENTEN
```

- In Programmen sollte man * vermeiden, da später manchmal Spalten zu Tabellen hinzugefügt werden.

Duplikat-Eliminierung (1)

- Ein Unterschied zwischen SQL und dem Tupelkalkül ist, dass Duplikte in SQL explizit eliminiert werden müssen.
- Z.B.: Welche Aufgaben wurden von mindestens einem Studenten gelöst?

```
SELECT ATYP, ANR
FROM   BEWERTUNGEN
```

ATYP	ANR
H	1
H	2
Z	1
H	1
H	2
Z	1
H	1
Z	1

Duplikat-Eliminierung (2)

- Die Duplikate treten auf, weil die Anfrage mit einer Schleife über die Zeilen in BEWERTUNGEN ausgeführt wird.
- Könnte eine Anfrage Duplikate enthalten und gibt es keinen Grund, diese mit auszugeben, verwendet man "SELECT DISTINCT":

```
SELECT DISTINCT ATYP, ANR
FROM   BEWERTUNGEN
```

ATYP	ANR
H	1
H	2
Z	1

Duplikat-Eliminierung (3)

- Um zu betonen, dass es Duplikate gibt, die auch gewünscht sind, kann man "`SELECT ALL`" schreiben.

  "`ALL`" ist jedoch der Default.

- Man beachte, dass `DISTINCT` immer zu ganzen Zeilen gehört, nicht zu einzelnen Spalten.

  Sonst NF^2-Tabellen nötig. Mit klassischen Relationen z.B. unmöglich: eine Zeile je Student mit all seinen Resultaten. Mit Ausgabe-Formatierung aber ähnliche Ergebnisse: In SQL*Plus kann man Spaltenwerte nur ausgeben lassen, wenn sich der Wert vom vorigen unterscheidet.

- Z.B. ist Folgendes ein Syntax-Fehler:

```
SELECT ATYP, ANR, DISTINCT THEMA
FROM   AUFGABEN
```

Falsch!

Ist DISTINCT nötig? (1)

Hinreichende Bedingung für überflüssiges `DISTINCT`:

- Sei $\mathcal{K}$ die Menge der Attribute, die als Ausgabe-Spalten unter `SELECT` auftreten.

  Die Elemente von $\mathcal{K}$ haben die Form "Tupelvariable.Attribut". $\mathcal{K}$ ist die Menge von Attributen, die einen eindeutigen Wert für eine gegebene Ausgabe-Zeile haben.

- Füge alle Attribute A zu $\mathcal{K}$ hinzu, für die $A = c$ in der `WHERE`-Bedingung auftaucht.

  Natürlich wird eine Bedingung $c = A$ genauso behandelt. Dieser Test nimmt an, dass die Bedingung eine Konjunktion ist. Bedingungen in Unteranfragen zählen nicht (Unteranfragen werden vor dem Test entfernt).

Ist DISTINCT nötig? (2)

Test für überflüssiges `DISTINCT`, fortgesetzt:

- Solange sich etwas ändert, mache Folgendes:
  - Füge zu $\mathcal{K}$ Attribute A hinzu, wobei $A = B$ in der `WHERE`-Bedingung auftaucht und $B \in \mathcal{K}$.
  - Enthält $\mathcal{K}$ einen Schlüssel einer Tupelvariable, füge alle Attribute dieser Tupelvariable hinzu.
- Enthält $\mathcal{K}$ von jeder Tupelvariable unter `FROM` einen Schlüssel, so ist `DISTINCT` überflüssig.

  Enthält die Anfrage `GROUP BY`, prüft man stattdessen, ob alle `GROUP BY`-Spalten in $\mathcal{K}$ enthalten sind.

Ist DISTINCT nötig? (3)

Beispiel:

- Betrachten Sie folgende Anfrage:

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

- Annahme: VORNAME, NACHNAME bilden zusammen einen alternativen Schlüssel für STUDENTEN.
- $\mathcal{K} := \{\texttt{S.VORNAME}, \texttt{S.NACHNAME}, \texttt{B.ANR}, \texttt{B.PUNKTE}\}$.
- $\mathcal{K} := \mathcal{K} \cup \{\texttt{B.ATYP}\}$ wegen Bedingung B.ATYP = 'H'.

Ist DISTINCT nötig? (4)

Beispiel, fortgesetzt:

- $\mathcal{K} := \mathcal{K} \cup \{$`S.SID`, `S.EMAIL`$\}$, da $\mathcal{K}$ einen Schlüssel von `STUDENTEN S` enthält (`S.VORNAME` und `S.NACHNAME`).
- $\mathcal{K} := \mathcal{K} \cup \{$`B.SID`$\}$ wegen `B.SID = S.SID`.
- Nun ist auch ein Schlüssel von `BEWERTUNGEN B` in $\mathcal{K}$ enthalten (`B.SID`, `B.ATYP`, `B.ANR`): `DISTINCT` unnötig.
- Wären `VORNAME`, `NACHNAME` nicht ein Schlüssel von `STUDENTEN`, dann wären Duplikate möglich.

  In diesem Fall könnte es jedoch sinnvoll sein Duplikate auszugeben, da Studenten in der realen Welt durch ihren Namen identifiziert werden.

DISTINCT vs. GROUP BY

- Duplikate sollten mit `DISTINCT` eliminiert werden, obwohl es auch mit `GROUP BY` (siehe Kapitel 11) 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.

Umbenennung von Spalten

- Um Ausgabe-Spalten umzubenennen:

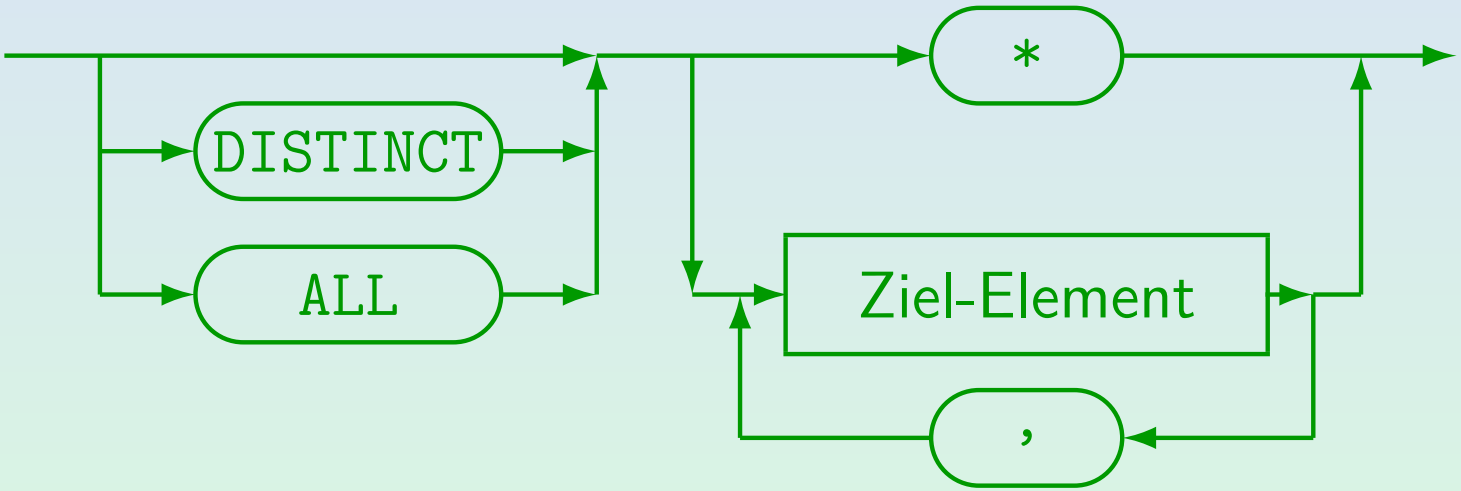
```
SELECT VORNAME AS V_Name, NACHNAME AS "Name"
FROM   STUDENTEN
```

V_NAME	Name
Lisa	Weiss
Michael	Grau
Daniel	Sommer
Iris	Winter

- Dies funktioniert in SQL-92, Oracle, SQL Server, DB2, MySQL, Access, aber nicht in SQL-86.
- "AS" kann in SQL-92 und allen obigen Systemen außer Access weggelassen werden.

SELECT-Syntax (1)

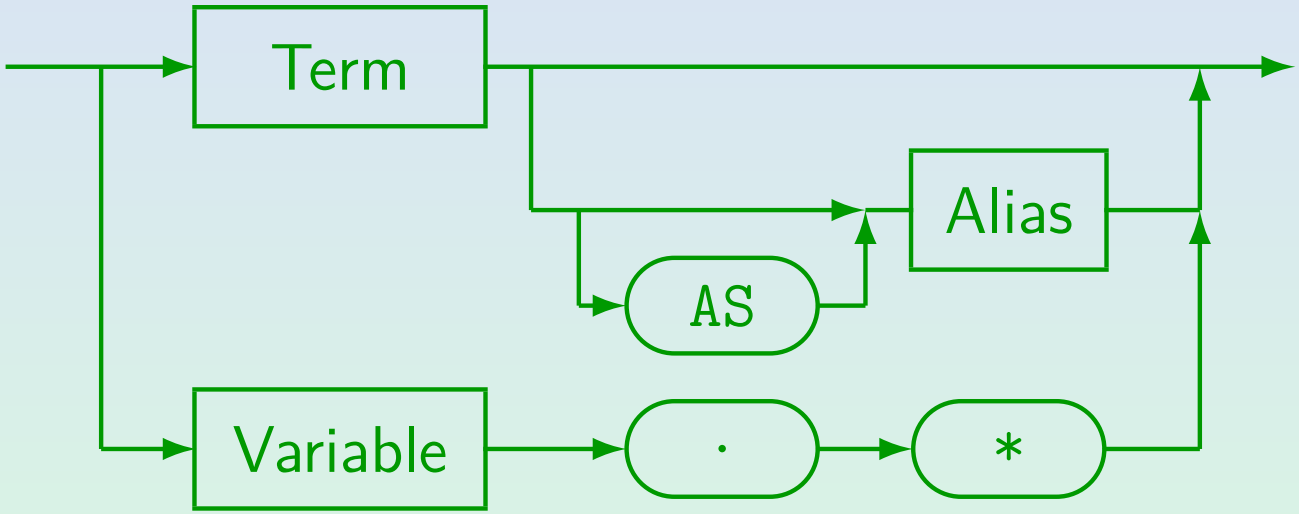
Ziel-Liste (nach `SELECT`):



- `ALL` (keine Duplikat-Elimination) ist der Default.

SELECT-Syntax (2)

Ziel-Element:



- "Variable.*" und "[AS] Alias" funktionieren in SQL-92, Oracle, SQL Server, DB2, MySQL und Access (in Access wird "AS" benötigt). Diese Konstruktionen sind im alten SQL-86-Standard nicht enthalten.