Einführung in Datenbanken

Kapitel 9: Praktische Aspekte einfacher SQL-Anfragen

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2020/21

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

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Einfache Anfragen in SQL schreiben (mit mehreren Tupelvariablen, auch mehreren über der gleichen Relation).

  Einfach heisst: Ohne Unteranfragen und Aggregationsfunktionen.

- Den Fehler „fehlende Verbund-Bedingungen“ in Anfragen erkennen und die Wirkung beschreiben.
- Anfragen QBE-ähnlich planen. QBE-Konzepte erläutern.
- Erläutern, was „korrekte Anfrage“ in der Klausur bedeutet.
- `BETWEEN`, `IN` mit Aufzählung, `LIKE...ESCAPE` verwenden.
- Duplikateliminierung mit `DISTINCT` mit Bedacht einsetzen.
- `ORDER BY` zur Sortierung in Anfragen verwenden.

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

Verbund (1)

- Der Verbund (engl. „Join“) ist eine Operation der relationalen Algebra (s. Kap. 14), die Tupel zweier Tabellen „zusammenklebt“, z.B. Tupel mit gleichen Werten in gleich benannten Spalten („natürlicher Verbund“).

STUDENTEN ⋈ BEWERTUNGEN						
SID	VORNAME	NACHNAME	EMAIL	ATYP	ANR	PUNKTE
101	Lisa	Weiss	···	H	1	10
101	Lisa	Weiss	···	H	2	8
101	Lisa	Weiss	···	Z	1	12
102	Michael	Grau	NULL	H	1	9
102	Michael	Grau	NULL	H	2	9
102	Michael	Grau	NULL	Z	1	10
103	Daniel	Sommer	···	H	1	5
103	Daniel	Sommer	···	Z	1	7

Verbund (1)

- Im Tupelkalkül (der logischen Basis von SQL) gibt es nicht explizit einen Verbund.
- Dennoch muss man natürlich Anfragen schreiben, die sich auf mehrere Tabellenzeilen (Tupel) gleichzeitig beziehen.
- Dazu deklariert man Variablen über den Tabellen, die man verknüpfen will.
- Die Bedingung zur Verknüpfung (z.B. gleiche `SID` in beiden Tupeln) muss man explizit hinschreiben.
- Sie wird üblicherweise auch „Verbund-Bedingung“ („Join-Bedingung“) genannt.

Fehlende Verbund-Bedingungen (1)

- Beispiel:

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

Falsch!

- SQL betrachtet alle möglichen Variablenbelegungen, d.h. jede Kombination einer Zeile `S` aus `STUDENTEN` und einer Zeile `B` aus `BEWERTUNGEN`.

  Im Beispiel-Zustand hat `STUDENTEN` vier Zeilen und `BEWERTUNGEN` acht, es gibt also $4 * 8 = 32$ mögliche Variablenbelegungen.

- Da die `WHERE`-Bedingung nicht fordert, dass

  `S.SID = B.SID`

  gilt, können auch Variablenbelegungen, die das nicht erfüllen, zu Ausgaben führen.

Fehlende Verbund-Bedingungen (2)

- Wenn es mindestens einen Studenten gibt, der Hausaufgabe 1 gelöst hat, erscheinen alle Studierenden in der Ausgabe, z.B. auch Iris Winter, die Hausaufgabe 1 nicht abgegeben hat.
- Im Beispiel haben drei Studierende Hausaufgabe 1 abgegeben. Daher wird jeder Student drei Mal ausgegeben.

  Die Zeile `S` für diesen Studenten kann ja mit jeder Zeile `B` für einen beliebigen Studenten kombiniert werden, der Hausaufgabe 1 abgegeben hat.
- Es ist typisch, dass man bei fehlenden Verbund-Bedingungen große Anfrage-Ergebnisse mit vielen Duplikaten bekommt.
- Man versuche dann zu verstehen, woher die Duplikate kommen, und löse das Problem nicht einfach mit `SELECT DISTINCT` (s.u.).

Fehlende Verbund-Bedingungen (3)

- Man muss explizit in der WHERE-Bedingung angeben, dass sich S und B auf den gleichen Studenten beziehen sollen:

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

  Die Reihenfolge der Bedingungen unter WHERE ist egal. Der Operator AND ist ja kommutativ, so dass es logisch gesehen keinen Unterschied macht. Der Anfrage-Optimierer sucht sich eine gute Auswertungsreihenfolge, so dass es von der Laufzeit her auch fast immer keinen Unterschied macht („fast“ bezieht sich auf den unwahrscheinlichen Fall, dass sich der Optimierer anders nicht entscheiden kann — dann wäre aber eher FROM-Reihenfolge wichtig).

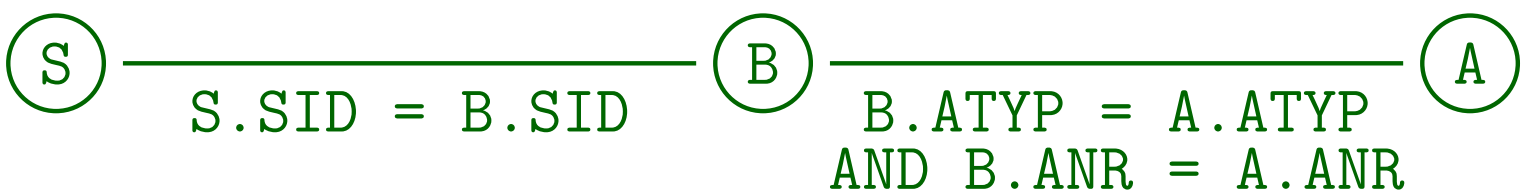
- Seit SQL-92 kann man die Verbund-Bedingung alternativ auch unter FROM angeben (Geschmackssache, s. Kap. 15).

Fehlende Verbund-Bedingungen (4)

- Hier sind drei Tupelvariablen über Verbundbedingungen verknüpft:

```
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'
```

- Die Tupelvariablen sind wie folgt verbunden:



Fehlende Verbund-Bedingungen (5)

- Die obigen Join-Bedingungen sind Gleichheits-Bedingungen zwischen einem Fremdschlüssel und dem Schlüssel der referenzierten Tabelle. Das ist ganz typisch.

  Es gibt aber Ausnahmen.

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

  Man könnte einen Graphen aufbauen mit den Tupelvariablen als Knoten und Kanten für Schlüssel-Fremdschlüssel Join-Bedingungen. Wenn dieser Graph zusammenhängend ist, ist alles in Ordnung. Es gibt aber auch Fälle mit unzusammenhängendem Graphen, die doch kein Problem sind (siehe z.B. nächste Folie). Der Verbund kann eventuell auch in einer Unteranfrage geschehen. Wir haben ein Forschungsprojekt zur Berechnung von Warnungen für SQL-Anfragen, die vermutlich fehlerhaft sind.

Fehlende Verbund-Bedingungen (6)

- Die folgende Anfrage ist in Ordnung:

```
SELECT S.VORNAME, S.NACHNAME,
       S.PUNKTE*100/A.MAXPT AS H1_PROZENT
FROM   STUDENTEN S, BEWERTUNGEN B, AUFGABEN A
WHERE  S.SID = B.SID
AND    B.ATYP = 'H' AND B.ANR = 1
AND    A.ATYP = 'H' AND A.ANR = 1
```

- Natürlich ist z.B.

```
A.ATYP = 'H' AND B.ATYP = 'H'
```

  äquivalent zu

```
A.ATYP = B.ATYP AND B.ATYP = 'H'
```

  Vermutlich ist eine Tupelvariable, deren Schlüssel nur einen möglichen Wert hat, nie ein Problem — selbst wenn nicht wie hier doch eine Verbundbedingung darin verboren ist.

Inhalt

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 erschweren.
- Betrachten Sie z.B. eine Datenbank zur Belegung von Vorlesungen, die auch Tutoren beinhaltet:

Tutoren sind fortgeschrittene Studenten, die bei der Korrektur von Hausaufgaben usw. helfen. In der Relation `TUTOR` könnten Einteilung bzw. Vertragsdaten gespeichert sein.

Hinweis für Klausur-/Hausaufgaben (1)

- Sie dürfen in der Lösung einer Aufgabe nur die Information (z.B. Datenwerte) verwenden, die dort explizit genannt sind.
- Beispiel: „Drucken Sie die Hausaufgabenpunkte der Studentin Lisa Weiss (jeweils Aufgabennummer und Punktzahl)“.
- Folgendes Vorgehen ist unzulässig:
  - Mit einer ersten Anfrage wird die `SID` von Lisa Weiss bestimmt: `SELECT SID FROM STUDENTEN WHERE ...`.

    Eventuell ist in der Klausur oder auf dem Aufgabenblatt auch ein Beispielzustand gezeigt, aus dem sich die `SID` 101 ergibt.
  - Diese wird dann in die eigentliche Anfrage eingesetzt:

```
SELECT ANR, PUNKTE
FROM   BEWERTUNGEN
WHERE  SID = 101 AND ATYP = 'H'
```

Hinweis für Klausur-/Hausaufgaben (2)

- Warum sollte man so nicht vorgehen?
  - Häufig werden Anfragen nicht nur einmal ausgeführt, sondern mehrfach (aus Programmen): Sie müssen auch dann noch funktionieren, wenn sich der Zustand geändert hat.
  - In der Realität wäre „Lisa Weiss“ nur ein Beispiel für Eingabewerte eines Programms.
  - Selbst wenn Sie ein Programm so schreiben, dass es nacheinander die beiden Anfragen ausführt, würden Sie SQL nicht optimal nutzen.

    Die Netzwerk-Kommunikation mit dem Server kostet Zeit. Es ist besser, wenige größere Anfragen zu verwenden, statt viele kleine. Die Optimierung funktioniert auch nur innerhalb einzelner Anfragen, nicht zwischen mehreren Anfragen. Schließlich wäre möglich, dass sich der Zustand zwischen den beiden Anfragen ändert, und es dann gar nicht funktioniert.

Hinweis für Klausur-/Hausaufgaben (3)

- Die Semantik einer Anfrage ist eine Abbildung (Funktion) von Datenbank-Zuständen in Antwort-Relationen.
- Ihre Anfrage ist korrekt, wenn Sie die gewünschte Abbildung beschreibt.

  Ein Punktabzug ist möglich wegen unnötigen Komplikationen oder Stilfehlern.
- Genauer ist die Ausgabe eine endliche Folge von Tupeln.
  - Wenn eine bestimmte Reihenfolge gefordert ist, muss die Liste genau stimmen. Sonst reicht eine beliebige Permutation.
  - Duplikate in der Antwort sind fast immer unerwünscht und führen zum Punktabzug (s.u.).

    Seltene Duplikate sind vielleicht nützlich, wenn z.B. zwei Studierende den gleichen Namen haben. Man möchte aus dem Anfrageergebnis vielleicht die Anzahl der Objekte in der realen Welt schließen.

Hinweis für Klausur-/Hausaufgaben (4)

- Außerdem wollen wir natürlich, dass Sie zeigen, dass Sie Verbundanfragen formulieren können.

  Wenn es nicht nur ein einzelner `SID`-Wert gewesen wäre, sondern mehrere, die sich aus dem Zugriff auf die Studenten-Tabelle ergeben hätten, wäre die Vorgehensweise mit einzelnen Anfragen deutlich aufwändiger.

- Eine gute Anfrage sollte gleich das Informationsbedürfnis erfüllen, und möglichst nicht noch eine manuelle Nachbearbeitung erfordern.

- Weiter Hinweis:

  - Lesen Sie die Aufgaben genau!
  - Im Beispiel ist von „Hausaufgabenpunkten“ die Rede. Daraus ergibt sich die Bedingung `ATYP = 'H'`.

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“.

Selbstverbund (1)

- Man spricht von einem Selbstverbund, wenn mehr als ein Tupel derselben Relation benötigt wird, um ein Ergebnis zu erhalten.

  Die Vorstellung aus der relationalen Algebra ist dann, dass man zwei Kopien der gleichen Relation mit einander verbindet.

- Die Anfrage eben (10 Punkte in Hausaufgabe 1 und mindestens 8 Punkte in HA 2) fällt in diese Kategorie:

```
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 >= 8
```

Selbstverbund (2)

- Studenten, die mindestens 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!

  Sie laufen unabhängig von einander über die Relation.

- 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 (später).

Selbstverbund (3)

Aufgabe:

- 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'
```

Unnötige Joins (1)

- Verknüpfen Sie nicht mehr Tabellen mit dem Verbund (Join) als nötig. D.h. keine unnötigen Tupelvariablen unter `FROM`.

  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
```

Zusammenfassung: Join-Fehler

- Fehlende Join-Bedingungen (sehr häufig)

  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).

- Unerlaubtes Spicken im Beispielzustand: Konstanten der Anfrage sollten im Aufgabentext auftauchen.

- Unnötige Joins (machen Anfrage langsamer)

- Wenn eigentlich mehrere Tupelvariablen über einer Relation benötigt werden, und man „verschmilzt sie“, so erhält man oft inkonsistente Bedingungen.

  Ein leeres Ergebnis sollte ein Warnsignal sein. Wenn man aber z.B. nach Ausnahmen oder fehlerhaften Werten sucht, ist man vielleicht froh, dass das Ergebnis leer ist.

Inhalt

Bedingungen (1)

- Bedingungen entsprechen logischen Formeln und werden in der Hauptsache unter `WHERE` verwendet.
- Eine Bedingung besteht aus „Basisbedingungen“, die mit `AND`, `OR`, `NOT` verknüpft sind.

  „Basisbedingung“ kann natürlich eine atomare Formel sein, aber z.B. auch eine `EXISTS`-Bedingung mit Unteranfrage, die sicher nicht atomar ist.

- Eine `AND`-Bedingung ist wahr, wenn beide Teile wahr sind, eine `OR`-Bedingung 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 (2)

In dieser Vorlesung behandelte „Basis-Bedingungen“:

1. $t_1\ \theta\ t_2$ mit Vergleichsoperator $\theta \in \{=, <>, <, <=, >, >=\}$
2. t_1 `BETWEEN` t_2 `AND` t_3
3. t_1 `LIKE` t_2 bzw. t_1 `LIKE` t_2 `ESCAPE 'c'`
4. t_0 `IN (`t_1`,` t_2`,...,`t_n`)`
5. `EXISTS (`Q`)` mit Unteranfrage Q
6. t `IN (`Q`)`
7. $t\ \theta$ `ALL (`Q`)` bzw. $t\ \theta$ `ANY (`Q`)`
8. t `IS NULL`

Vergleiche (1)

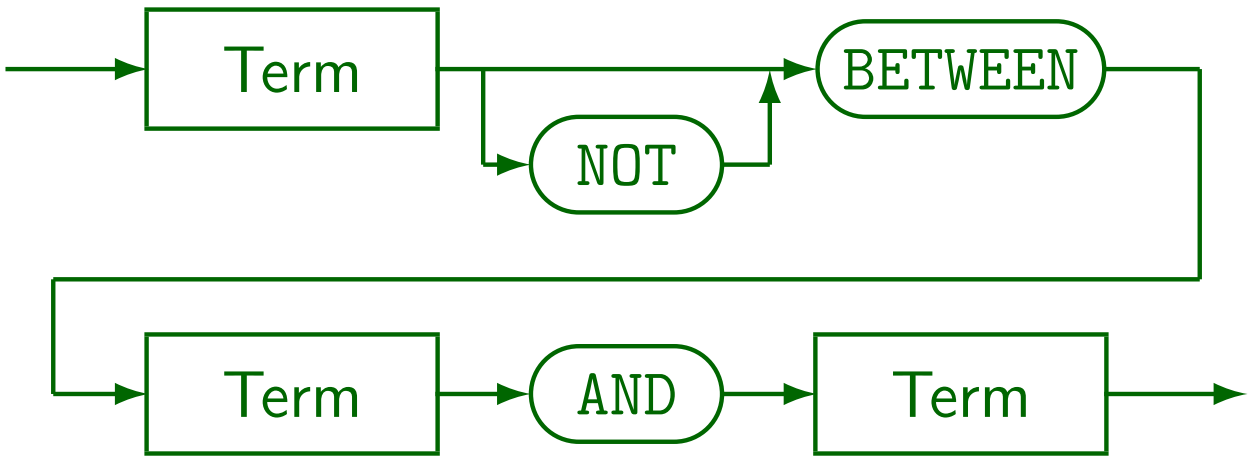
Bedingung (Form 1: Vergleich):

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

  Bei Zeichenketten spielt die gewählte „Collation“ eine Rolle. Je nach Auswahl kann der Vergleich z.B. Groß- und Kleinschreibung unterscheiden oder auch nicht. Außerdem werden Leerzeichen am Ende teils ignoriert (`PAD SPACE`) oder nicht (`NO PAD`). Die Collation kann für die DBMS-Installation, die Datenbank, die Tabelle, die Tabellenspalte, oder auch den einzelnen Vergleich gewählt werden.
- Die Datentypen der beiden Terme müssen kompatibel sein.

BETWEEN-Bedingungen

Bedingung (Form 2: BETWEEN):



- Diese Bedingung ist nur eine Abkürzung:

  ```
  x BETWEEN y AND z
  ```

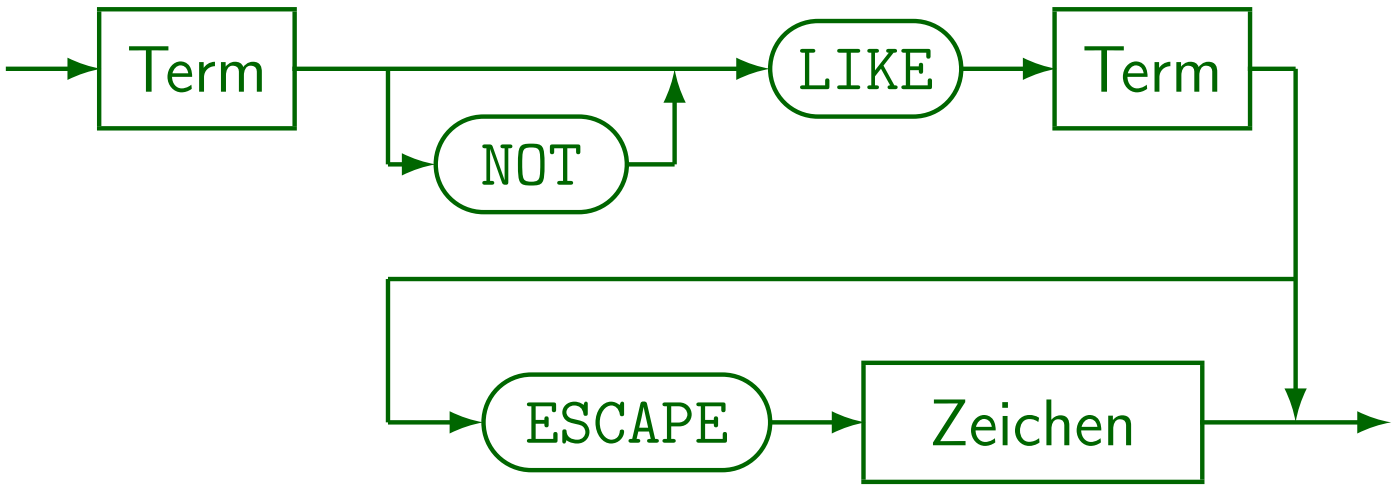
  ist äquivalent zu

  ```
  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 musste das Muster eine String-Konstante sein. Inzwischen kann man einen beliebigen Term verwenden, um das Muster zu berechnen.

  Z.B. kann man eine andere Spalte als Muster verwenden, und ggf. mit der String-Konkatenation || vorn und hinten Wildcards '%' anfügen.

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

  In der UNIX-Shell (Kommando-Interpreter) wird „*“ statt „%“ verwendet.

- „_“ passt auf ein beliebiges einzelnes Zeichen.

  Dies entspricht „?“ in der Shell.

LIKE-Bedingungen (3)

- `LIKE` muss zur Mustersuche verwendet werden.
  Das Gleichheitszeichen überprüft nur die (mehr oder weniger) genaue Gleichheit Zeichen für Zeichen.

  Auch wenn der Vergleichs-String „%“ oder „_“ enthält. Diese Zeichen werden beim Vergleich mit „=“ nicht speziell interpretiert.

- 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!

- Ich finde es stilistisch nicht in Ordnung, `LIKE` zu verwenden, wenn die rechte Seite gar kein „%“ oder „_“ enthält (**!**).

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.

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

- Z.B. könnte man folgende Bedingung in einem CASE-System („Computer Aided Software Engineering“) verwenden, um Variablennamen in der DB zu finden, die mit „_“ beginnen:

```
VAR_NAME LIKE '\_%' ESCAPE '\'
```

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.

- Alle anderen Eigenschaften der Collation wie Case-Sensitivität sind die gleichen 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.

- In Oracle heißt der Operator `REGEXP_LIKE`.

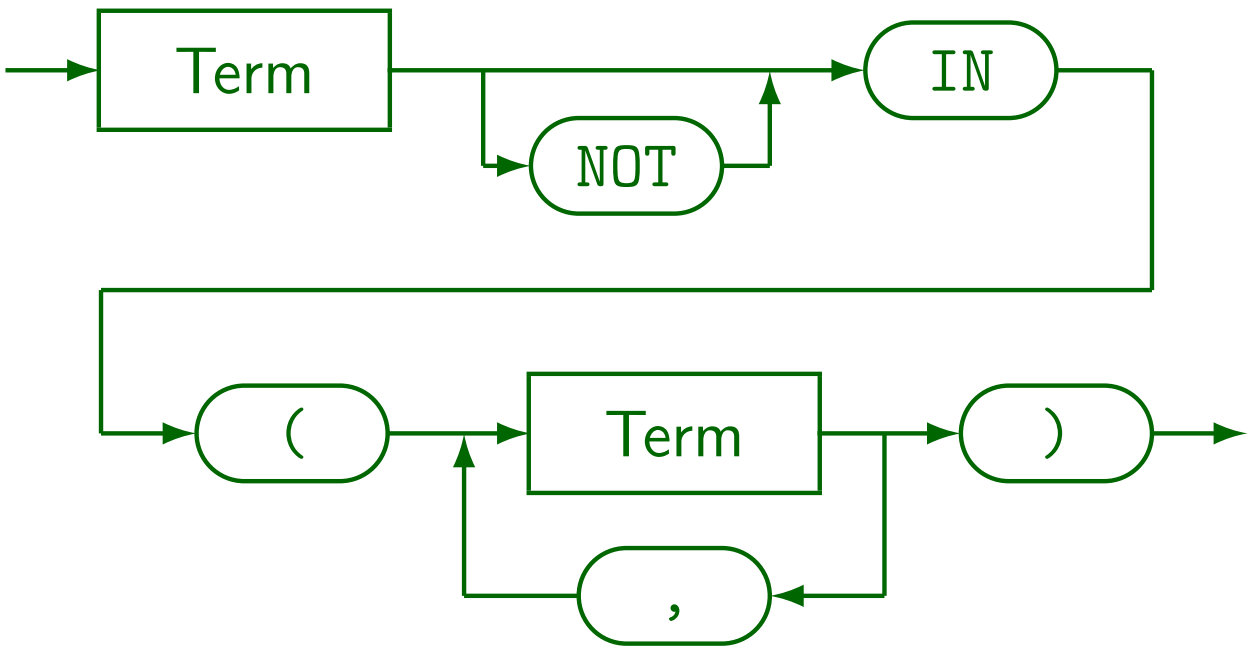
- Der SQL:1999-Standard führte ein Prädikat „`SIMILAR TO`“ ein, das Vergleiche mit regulären Ausdrücken durchführt.

  Die meisten Systeme verstehen diese Schreibweise nicht.

  PostgreSQL ist eine Ausnahme, da kann man `SIMILAR TO` verwenden.

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.

  In der Mathematik wäre $\{\ldots\}$ üblich, und z.B. $(0, 100)$ könnte man auch für ein Intervall halten. Aber SQL funktioniert mit einem relativ kleinen Zeichensatz, die geschweiften Klammern werden dort gar nicht verwendet.

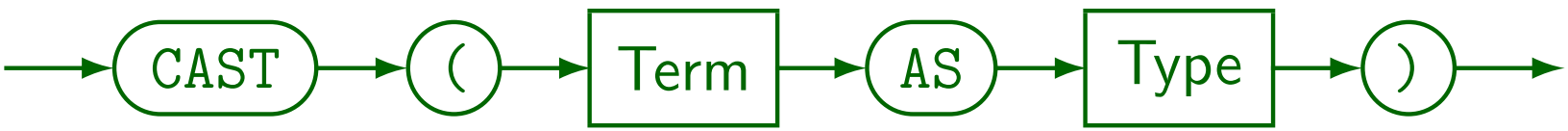
Inhalt

Typ-Umwandlungen (1)

- Wie in Programmiersprachen auch, muss man gelegentlich einen Wert in einen anderen Typ umwandeln (bzw. den Typ für einen Wert explizit angeben).

  In SQL ist das recht selten erforderlich, da es meist sehr großzügige und teils fragwürdige Typ-Umwandlungen automatisch macht. Bei Nullwerten und bei `UNION` braucht man aber öfters eine explizite Typ-Angabe.

- Die Syntax nach dem SQL-Standard ist:



  Dies funktioniert in den meisten modernen Systemen.
  MariaDB/MySQL erlaubt nicht beliebige Typen, z.B. nicht `NUMERIC(1)` und `VARCHAR(15)`, aber schon `INT` und `CHAR(15)`.
  [https://dev.mysql.com/doc/refman/8.0/en/cast-functions.html]

Typ-Umwandlungen (2)

- Oracle versteht die `CAST`-Syntax inzwischen auch, hat aber traditionell Funktionen wie `TO_CHAR(`t`)`, `TO_NUMBER(`t`)` und `TO_DATE(`t`)`.
- Diese Funktionen sind weiter nützlich, da sie als zweites Argument die Angabe eines Formats erlauben, z.B.

```
TO_CHAR((S.PUNKTE*100/A.MAXPT), '999.9')
```

  Das Format `999.9` bedeutet eine Zahl mit drei Stellen vor dem Komma und einer danach. Wenn man `099.9` schreibt, werden führende Nullen gedruckt.
  [https://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements004.htm]

- Auch bei Datumswerten wird häufig eine benutzerdefinierte Formatierung benötigt:

```
TO_CHAR(B.ABGABEDATUM, 'DD.MM.YYYY')
```

Typ-Umwandlungen (3)

- Auch PostgreSQL hat Funktionen `TO_CHAR`, `TO_NUMBER`, `TO_DATE`, `TO_TIMESTAMP` mit einem zweiten Argument für das Format.

  [https://www.postgresql.org/docs/11/functions-formatting.html]

  Im Gegensatz zu Oracle ist das zweite Argument nicht optional.

  Wenn der Eingabestring nicht auf das Format passt, gibt es in PostgreSQL keine Fehlermeldung, aber es kommt ein sinnloser Wert heraus.

  Bei `CAST` sind die möglichen Formate begrenzter, aber dort gibt es ggf. eine Fehlermeldung.

- MySQL/MariaDB haben Funktionen `DATE_FORMAT` und `STR_TO_DATE` zur Konvertierung zwischen Datumswerten und Zeichenketten. Für Zahlen gibt es die Funktion `FORMAT`.

  [https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html]

  [https://dev.mysql.com/doc/refman/8.0/en/string-functions.html]

Binärer Zeichenkettenvergleich

- In Kapitel 7 wurde gezeigt, wie ein Zeichenkettenvergleich mit `LOWER`/`UPPER` case-insensitiv gemacht werden kann, selbst wenn die eingestellte Collation case-sensitiv ist.
- Der umgekehrte 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))
```

Inhalt

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.
- $\mathcal{K} := \{$S.VORNAME, S.NACHNAME, B.ANR, B.PUNKTE$\}$.
- $\mathcal{K} := \mathcal{K} \cup \{$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 13) 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.

Inhalt

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
Daniel	Sommer
Lisa	Weiss

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).

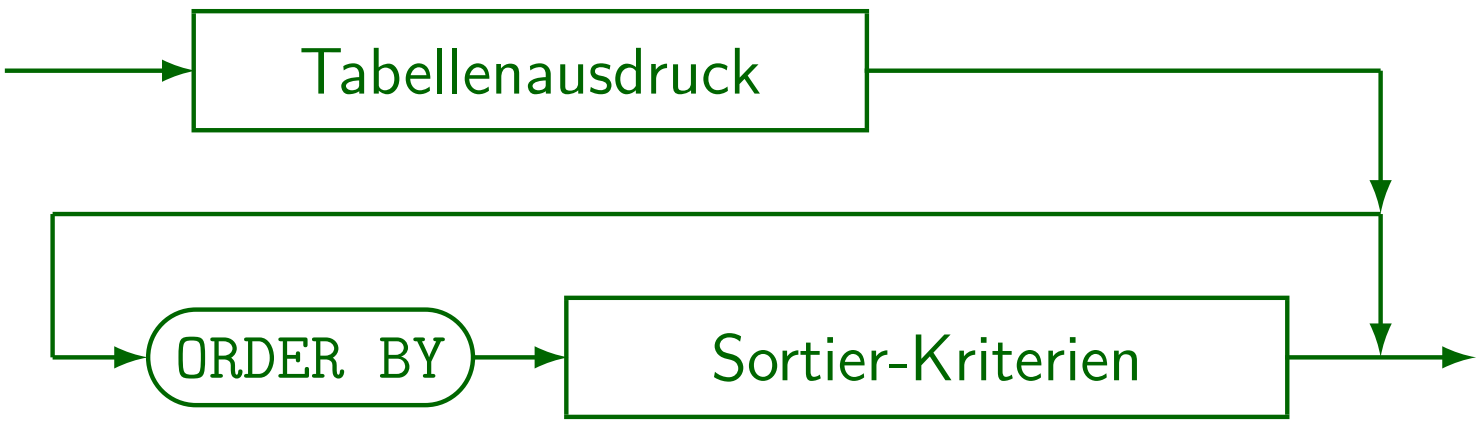
  In Oracle kann man `NULLS FIRST` oder `NULLS LAST` festlegen.

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).

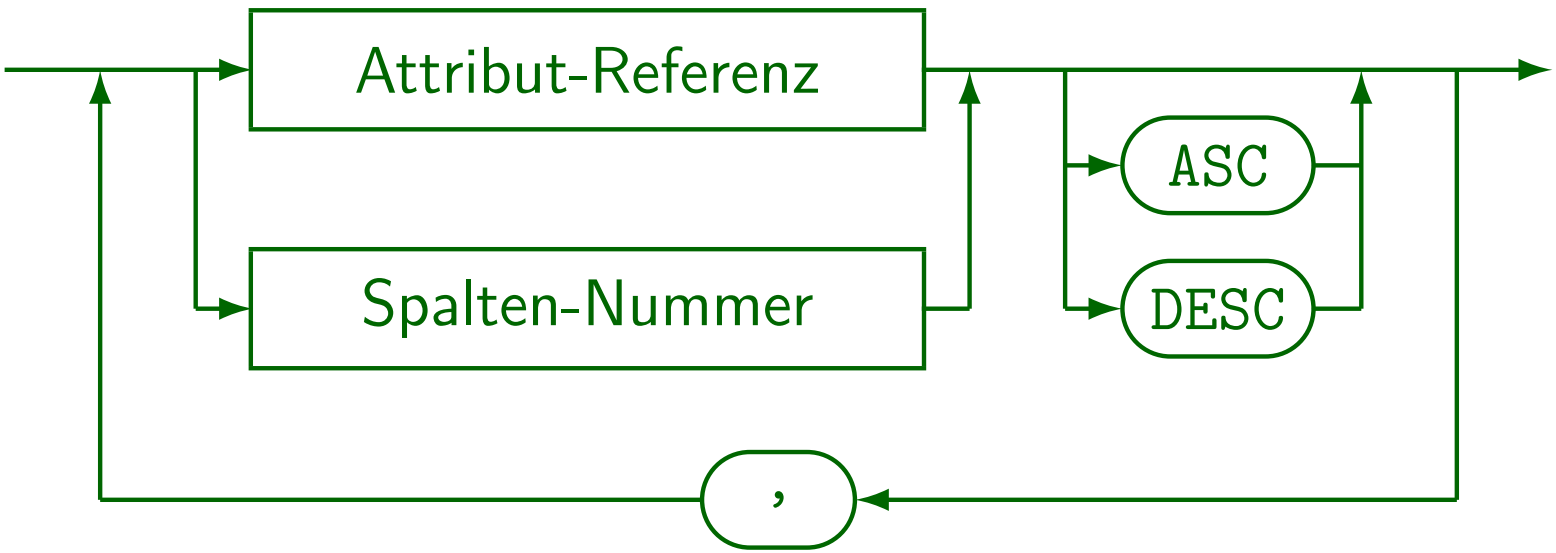
Sortieren der Ausgabe (9)

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).