Einführung in Datenbanken

Übung 8: Duplikate, ORDER BY, Logik

Prof. Dr. Stefan Brass
PD Dr. Alexander Hinneburg
Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2024/25

http://www.informatik.uni-halle.de/~brass/db24/

Inhalt

Duplikate in SQL (1)

- Regeln in der Klausur:
  - Ihre Anfragen dürfen keine Duplikate erzeugen.
  - Im Beispielzustand lässt sich das ja leicht testen, aber die Forderung gilt für alle Zustände, die die Integritätsbedingungen (insbesondere Schlüssel) erfüllen.

    Wenn schon einfaches Ausprobieren Duplikate liefert, könnte man natürlich „`DISTINCT`“ verwenden. Es wäre aber dennoch besser, zuerst die Ursache der Duplikate zu verstehen. Die Ursache könnte z.B. auch eine vergessene Verbund-Bedingung sein.
  - Für ein unnötiges `DISTINCT` wird auch ein Punkt abgezogen.

    Das System muss extra die relativ teure Duplikat-Elimination anschmeißen. Diese braucht z.B. temporären Speicherplatz (die Duplikat-Elimination wird häufig durch Sortierung implementiert).

Duplikate in SQL (2)

- In der Praxis sind gelegentlich einzelne Duplikate interessant:
  - Wenn es z.B. zwei Studierende mit gleichem Vor- und Nachnamen gibt, die die Bedingung erfüllen,
  - möchte man von der Anzahl der Ausgabezeilen wahrscheinlich auf die Anzahl der Studierenden schließen können, und das Duplikat nicht entfernen.

    Man hätte dann wohl besser auch die Matrikelnummer mit ausgegeben.
- In der Klausur versuchen wir das durch zusätzliche Schlüssel eindeutig zu machen (z.B. hier: Vor- und Nachname).
- Wenn Sie in einer Firma bei jeder Anfrage `DISTINCT` schreiben, wird Ihr Chef wahrscheinlich nicht einverstanden sein (der Optimierer eliminiert das eher nicht).

Duplikate in SQL (3)

- Wir erwarten also, dass Sie in der Lage sind, Anfragen daraufhin zu analysieren, ob sie in irgendeinem Zustand Duplikate liefern.

  Zumindest in hinreichend einfachen Fällen. Im allgemeinen ist das Problem unentscheidbar — es gibt also keinen Algorithmus, der das für beliebige Anfragen könnte.

- SQL erzeugt eine Ausgabezeile für jede Belegung der Tupelvariablen unter `FROM`, die die `WHERE`-Bedingung erfüllt.
- Die Frage ist also, ob es zwei verschiedene Belegungen geben kann,
  - die beide die `WHERE`-Bedingung erfüllen, und
  - für die alle unter `SELECT` genannten Terme die gleichen Werte haben.

Duplikate in SQL (4)

- Weil Duplikate so eng mit dem zentralen Begriff der Variablenbelegung zusammenhängen, macht es Sinn, zu fordern, dass Sie in der Lage sind, Duplikate vorherzusagen.
- Dadurch zeigen Sie, dass Sie verstanden haben, wie SQL funktioniert.

Wichtige Klarstellung:

- Duplikate beziehen sich immer auf ganze Ergebniszeilen.

  In den folgenden Beispielen gibt es häufig nur eine Ergebnisspalte, deswegen wird der Unterschied möglicherweise nicht ganz deutlich. Gleiche Werte in einer Spalte wären aber noch kein Duplikat, wenn es noch weitere Ergebnisspalten gibt. Nur wenn zwei Ergebniszeilen in allen Spalten übereinstimmen, ist es ein Duplikat.

Erstes Beispiel zu Duplikaten (1)

Aufgabe:

- Gegeben sei die Tabelle

  STUDENTEN(SID, VORNAME, NACHNAME, EMAIL°)

- Die Kombination VORNAME, NACHNAME sei auch Schlüssel.

  Neben dem Primärschlüssel SID.

- Kann folgende Anfrage Duplikate liefern?

```
SELECT S.NACHNAME
FROM   STUDENTEN S
WHERE  S.VORNAME LIKE 'Alex%'
```

  Ja/Nein-Abstimmung.

  Die Frage ist nicht, ob hier Dupliate sinnvoll wären, sondern nur, ob sie in irgendeinem DB-Zustand herauskommen können.

Erstes Beispiel zu Duplikaten (2)

- Ja, z.B. in folgendem Zustand:

STUDENTEN			
SID	VORNAME	NACHNAME	EMAIL
101	Alexandra	Müller	a_mueller@acm.org
102	Alexander	Müller	alex_m99@gmail.com

- In diesem Zustand liefert die Anfrage folgende Ausgabe:

NACHNAME
Müller
Müller

Die Duplikate entstehen dadurch, dass die anderen Spalten ausgeblendet werden. (In der Relationenalgebra wäre das die Operation „Projektion“.)

Erstes Beispiel zu Duplikaten (3)

- Variablenbelegung $\mathcal{A}_1$ erzeugt Ausgabe „Müller“:

	STUDENTEN			
	SID	VORNAME	NACHNAME	EMAIL
S →	101	Alexandra	Müller	...
	102	Alexander	Müller	...

- Variablenbelegung $\mathcal{A}_2$ erzeugt auch Ausgabe „Müller“:

	STUDENTEN			
	SID	VORNAME	NACHNAME	EMAIL
	101	Alexandra	Müller	...
S →	102	Alexander	Müller	...

Erstes Beispiel zu Duplikaten (4)

- Damit hat man für diesen Zustand zwei Variablenbelegungen,
  - die beide die `WHERE`-Bedingung erfüllen,
  - beide die gleiche Ausgabe (unter `SELECT`) erzeugen.
- Man kann das auch mit dem naiven Auswertungsalgorithmus (Schleife über allen Tabellenzeilen) verstehen:

```
for S in STUDENTEN do
    if like(S.VORNAME, 'Alex%') then
        print S.NACHNAME
```

Mit der Schleife werden gerade die möglichen Variablenbelegungen durchgegangen.

- Im Beispiel wären die Duplikate wohl erwünscht.

Algorithmus für mögliche Duplikate (1)

- In der Vorlesung ist ein Algorithmus angegeben, der eine hinreichende Bedingung für ein überflüssiges `DISTINCT` enthält:
  - Wenn der Algorithmus ausgibt, dass die Anfrage (ohne `DISTINCT`) keine Duplikate liefern kann, ist das sicher.
  - Im anderen Fall weiss man nicht, ob die Anfrage wirklich Duplikate liefern kann, oder der Algorithmus zu schwach ist.

    Da die Frage unentscheidbar ist, kann es keinen Algorithmus geben, der immer eine definitive (und korrekte) Antwort gibt. Man kann das Duplikatsproblem auf das Konsistenzproblem zurückführen und umgekehrt.
- Der Algorithmus sollte für die Klausuraufgaben ausreichen.

  Sie können die Frage nach Duplikaten natürlich auch weniger formal klären. Stellen Sie sich die (geschachtelten) Schleifen über den Tabellen vor (eine für jede Tupelvariable).

Algorithmus für mögliche Duplikate (2)

- Der Algorithmus verwendet eine Menge $\mathcal{K}$
  - von Attributreferenzen, also Einträgen der Form $\langle \text{Tupelvariable} \rangle . \langle \text{Attribut} \rangle$,
  - von denen bekannt ist, dass sie für eine gegebene Ausgabezeile nur einen Wert haben können.
- Das gilt insbesondere für Attribut-Referenzen, die direkt unter `SELECT` stehen, im Beispiel also `S.Nachname`.
- Falls nun die Menge $\mathcal{K}$ einen Schlüssel von jeder Tupelvariable enthält, kann es zu einer gegebenen Ausgabezeile nicht zwei verschiedene Variablenbelegungen geben.

  Die Ausgabezeile bestimmt dann ja die Schlüsselwerte eindeutig. Zu einer gegebenen Ausgabezeile kann es dann nur jeweils eine Tabellenzeile geben, die man der Tupelvariablen zuweisen kann, um diese Ausgabezeile zu erzeugen.

Algorithmus für mögliche Duplikate (3)

- Im Beispiel ist: $\mathcal{K} = \{\texttt{S.Nachname}\}$.
- `Nachname` ist kein Schlüssel der Tabelle, über die `S` läuft (`STUDENTEN`).
- Daher sagt der Algorithmus nichts aus.

  Bzw.: „Die Anfrage könnte eventuell Duplikate liefern“.
- Um ein konkretes Beispiel zu finden, versuche man zwei Variablenbelegungen für die Tupelvariablen zu konstruieren,
  - die die gleichen Werte für alle Elemente von $\mathcal{K}$ haben, und
  - die `WHERE`-Bedingung erfüllen.

    Im allgemeinen greift hier die Unentscheidbarkeit des Konsistenzproblems. Für einfache `WHERE`-Bedingungen wäre das natürlich automatisch möglich. D.h. man könnte den Algorithmus so ausbauen, dass er zumindest manchmal ein klares „Duplikate sind möglich“ ausgibt.

Zweites Beispiel zu Duplikaten

Aufgabe:

- Gegeben sei die Tabelle

  STUDENTEN(SID, VORNAME, NACHNAME, EMAIL°)

- Die Kombination VORNAME, NACHNAME sei auch Schlüssel.

  Neben dem Primärschlüssel SID.

- Kann folgende Anfrage Duplikate liefern?

```
SELECT S.NACHNAME
FROM   STUDENTEN S
WHERE  S.VORNAME = 'Alex'
```

Ja/Nein-Abstimmung.

Zweites Beispiel zum Duplikat-Algorithmus

- Die Menge $\mathcal{K}$ wird wie eben mit Attributreferenzen initialisiert, die direkt unter SELECT als Ausgabespalte stehen:

  $$\mathcal{K} = \{\texttt{S.NACHNAME}\}.$$

  Um eventuelle Alternativen mit OR auszuschließen, wandelt man die Bedingung zunächst in konjunktive Normalform um (eine mit AND-verknüpfte Liste), und schaut dann, ob ein Element dieser Liste die Form $X.A = c$ hat.

- Dann werden alle Attribut-Referenzen hinzugefügt, die aufgrund der WHERE-Klausel einen festen Wert haben:

  $$\mathcal{K} = \{\texttt{S.NACHNAME}, \texttt{S.VORNAME}\}.$$

- Da VORNAME und NACHNAME zusammen einen Schlüssel von STUDENTEN bilden (über dieser Tabelle läuft S), garantiert der Algorithmus, dass diese Anfrage keine Duplikate liefern kann.

Einfache Aufgaben zu Duplikaten (1)

a)

- Kann diese Anfrage Duplikate liefern?

```
SELECT SID
FROM   STUDENTEN
WHERE  NACHNAME LIKE 'M%'
```

- Wie eben sind die Schlüssel von `STUDENTEN`:
  - `SID` (Primärschlüssel)
  - `VORNAME`, `NACHNAME` (Sekundärschlüssel)
- Ja/Nein-Abstimmung.
- Was ist $\mathcal{K}$?

Einfache Aufgaben zu Duplikaten (2)

a)

- Nein (kann keine Duplikate liefern).
- $\mathcal{K} = \{\texttt{STUDENTEN.SID}\}$.

b)

- Kann diese Anfrage Duplikate liefern?

```
SELECT VORNAME
FROM   STUDENTEN
WHERE  SID = 153
```

- Schlüssel wie gehabt.

  {SID} und {VORNAME, NACHNAME}.

- Was ist $\mathcal{K}$?

Einfache Aufgaben zu Duplikaten (3)

b)

- Nein (kann keine Duplikate liefern).
- $\mathcal{K} = \{\texttt{STUDENTEN.VORNAME}, \texttt{STUDENTEN.SID}\}$.

  SID ist Schlüssel von STUDENTEN.

  Es ist hier unwesentlich, dass VORNAME auch in $\mathcal{K}$ enthalten ist.

c)

- Kann diese Anfrage Duplikate liefern?

```
SELECT VORNAME
FROM   STUDENTEN
WHERE  VORNAME = 'Dorothea'
```

- Was ist $\mathcal{K}$?

Einfache Aufgaben zu Duplikaten (4)

c)

- Ja (kann Duplikate liefern).

  Es kann mehrere Studentinnen geben, die Dorothea heißen.

- $\mathcal{K} = \{\texttt{STUDENTEN.VORNAME}\}$.

  `VORNAME` allein ist nicht Schlüssel von Studenten.

d)

- Kann diese Anfrage Duplikate liefern?

```
SELECT SID
FROM   STUDENTEN
WHERE  VORNAME = 'Lisa' OR NACHNAME = 'Weiss'
```

Einfache Aufgaben zu Duplikaten (5)

d)

- Nein (keine Duplikate).

  Obwohl es „Lisa Weiss“ gibt, die beide Teile der Disjunktion erfüllt, wird sie nicht doppelt geliefert: Jede Variablenbelegung wird nur ein Mal getestet.

- $\mathcal{K} = \{\texttt{STUDENTEN.SID}\}$.

  `SID.VORNAME` und `SID.NACHNAME` sind nicht dabei, weil die `WHERE`-Bedingung nicht eine einzelne Konstante erzwingt (durch das `OR`).

e)

- Kann diese Anfrage Duplikate liefern?

```
SELECT VORNAME
FROM   STUDENTEN
WHERE  SID <= 101 AND SID >= 101
OR     SID >  200 AND SID <= 200
```

Einfache Aufgaben zu Duplikaten (6)

e)

- Nein, die Bedingung ist äquivalent zu `SID = 101`.

  Die Anfrage liefert also nur höchstens eine Zeile.

  Der Teil nach dem `OR` ist inkonsistent.

- Der Algorithmus liefert $\mathcal{K} = \{\texttt{STUDENTEN.VORNAME}\}$ und deswegen „ich weiss nicht/Duplikate eventuell möglich“.

  Äquivalenz ist unentscheidbar. Der Algorithmus findet $X.A = c$ nur, wenn es syntaktisch offensichtlich ist.

f)

- Kann diese Anfrage Duplikate liefern?

```
SELECT X.SID
FROM   STUDENTEN X, STUDENTEN Y
```

Einfache Aufgaben zu Duplikaten (7)

f)

- Ja, Duplikate sind möglich: Die Tupelvariable `Y` erzeugt eine zusätzliche Schleife über den `STUDENTEN`.

```
for X in STUDENTEN do
    for Y in STUDENTEN do
        print X.SID
```

- Wenn `STUDENTEN` n Zeilen hat, werden n^2 Ergebniszeilen ausgegeben, und zwar die n verschiedenen `SID`s jeweils n mal.

  D.h. im Fall $n = 0$ und $n = 1$ gibt es keine Duplikate. Die Frage ist aber immer, ob es mindestens einen DB-Zustand gibt, in dem die Anfrage Duplikate erzeugt. Das gilt hier natürlich.

- $\mathcal{K} = \{\texttt{X.SID}\}$.
  Dies enthält zwar einen Schlüssel von `X`, aber nicht von `Y`.

Inhalt

ORDER BY (1)

- Gegeben sei folgende Tabelle R:

R		
A	B	C
a	1	10
b	1	20
c	2	10
d	2	20
e	3	20

- Was gibt diese SQL-Anfrage aus (Folge der A-Werte reicht)?

```
SELECT *
FROM   R
WHERE  B < 3
ORDER  BY B, C DESC
```

ORDER BY (2)

Lösung:

- Die Anfrage war:

```
SELECT *
FROM   R
WHERE  B < 3
ORDER  BY B, C DESC
```

- B ist das Haupt-Sortierkriterium (aufsteigend),
  bei gleichem B-Wert wird nach C sortiert (absteigend):

A	B	C
b	1	20
a	1	10
d	2	20
c	2	10

ORDER BY (3)

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

ORDER BY (4)

Aufgabe:

- Ist diese Anfrage syntaktisch korrekt (man beachte, dass beide Tupelvariablen ein Attribut `SID` haben)?

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

Zum Ausprobieren (Mogeln):

- [Link zum Adminer]

ORDER BY (5)

Aufgabe:

- Ist diese Anfrage syntaktisch korrekt (man beachte, dass beide Tupelvariablen ein Attribut SID haben)?

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

Lösung:

- Ja, die Anfrage ist korrekt.
- Offiziell kann man nur nach Ergebnisspalten sortieren. Die Ergebnisspalte heißt SID.

ORDER BY (6)

Aufgabe:

- Wird diese Anfrage von PostgreSQL und anderen DBMS akzeptiert?

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

ORDER BY (7)

Aufgabe:

- Wird diese Anfrage von PostgreSQL und anderen DBMS akzeptiert?

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

Lösung:

- Ja, man kann nach jedem Term sortieren, der unter SELECT gestanden haben könnte.

ORDER BY (8)

Aufgabe:

- Wird diese Anfrage von PostgreSQL akzeptiert?

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

ORDER BY (9)

Aufgabe:

- Wird diese Anfrage von PostgreSQL akzeptiert?

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

Lösung:

- Nein, Fehlermeldung: „for SELECT DISTINCT, ORDER BY expressions must appear in select list“
- Dagegen wird `ORDER BY S.SID` akzeptiert.

Inhalt

Monotone und nichtmonotone Anfragen (1)

- Definition: Ein DB-Zustand $\mathcal{I}_1$ ist ist kleinergleich einem DB-Zustand $\mathcal{I}_2$, geschrieben $\mathcal{I}_1 \subseteq \mathcal{I}_2$, gdw. $\mathcal{I}_1(R) \subseteq \mathcal{I}_2(R)$ für alle Relationen R im Schema.
- Erläuterung: Das bedeutet, der Zustand $\mathcal{I}_2$ entsteht aus dem Zustand $\mathcal{I}_1$ durch Einfügung von Tabellenzeilen.
- Definition: Eine Anfrage Q heißt monoton gdw. für alle DB-Zustände $\mathcal{I}_1, \mathcal{I}_2$ gilt:

  $$\mathcal{I}_1 \subseteq \mathcal{I}_2 \implies \mathcal{I}_1[Q] \subseteq \mathcal{I}_2[Q].$$

  Anderfalls heißt sie nichtmonoton.
- Erläuterung: Bei einer monotonen Anfrage bekommt man nach der Einfügung also mindestens die gleichen Antworttupel wie vorher (eventuell noch zusätzliche).

Monotone und nichtmonotone Anfragen (2)

- Satz: SQL-Anfragen Q der Form

$$\begin{array}{ll} \texttt{SELECT} & t_1, \ldots, t_k \\ \texttt{FROM} & R_1\ X_1,\ \ldots,\ R_n\ X_n \\ \texttt{WHERE} & F \end{array}$$

  wobei die Bedingung F keine Unteranfragen enthält (also eine quantorenfreie Formel ist), sind monoton.

- Das bedeutet:
  - Sei $\mathcal{I}_1$ ein DB-Zustand, und resultiere $\mathcal{I}_2$ aus $\mathcal{I}_1$ durch Einfügen von einem oder mehreren Tupeln.
  - Dann ist jedes Antworttupel t der Anfrage Q in $\mathcal{I}_1$ auch in der Antwort auf Q in $\mathcal{I}_2$ enthalten.

    D.h. korrekte Antworten bleiben auch nach Einfügungen gültig.

Monotone und nichtmonotone Anfragen (3)

Aufgaben:

- Findet diese Anfrage Studenten ohne eine Hausaufgabe in der DB? Wenn nicht, was berechnet sie?

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

- Bekommt man so Übungen (noch) ohne Abgaben?

```
SELECT DISTINCT A.ATYP, A.ANR
FROM   AUFGABEN A, BEWERTUNGEN B
WHERE  A.ATYP = B.ATYP AND A.ANR = B.ANR
AND    B.SID IS NULL
```

Monotone und nichtmonotone Anfragen (4)

- Gesucht: Studierende ohne abgegebene Hausaufgaben.
- Korrekte Lösung:

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S
WHERE  NOT EXISTS (SELECT *
                   FROM   BEWERTUNGEN B
                   WHERE  S.SID  = B.SID
                   AND    B.ATYP = 'H')
```

Monotone und nichtmonotone Anfragen (5)

Aufgaben:

- Liefert diese Anfrage den Studenten / die Studentin mit den meisten Punkten für Hausaufgabe 1?

```
SELECT DISTINCT S.VORNAME, S.NACHNAME, X.PUNKTE
FROM   STUDENTEN S, BEWERTUNGEN X, BEWERTUNGEN Y
WHERE  S.SID = X.SID
AND    X.ATYP = 'H' AND X.ANR = 1
AND    Y.ATYP = 'H' AND Y.ANR = 1
AND    X.PUNKTE > Y.PUNKTE
```

- Wenn nicht, was berechnet sie?

Monotone und nichtmonotone Anfragen (6)

- Gesucht: Studierende mit maximal erreichter Punktzahl für Hausaufgabe 1.
- Korrekte Lösung:

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H' AND B.ANR = 1
AND    NOT EXISTS (SELECT *
                   FROM   BEWERTUNGEN X
                   WHERE  X.ATYP = 'H'
                   AND    X.ANR = 1
                   AND    X.PUNKTE > B.PUNKTE)
```

Inhalt

EMP-DEPT-Datenbank

- `dept`: „Department“ (Abteilungen einer Firma)

  `dept(deptno, dname, loc)`

- `emp`: „Employee“ (Angestellte der Firma)

  `emp(empno, ename, job, mgr°→emp, hiredate, sal, comm°, deptno°→dept)`

- `salgrade`: „Salary Grade“ (Gehaltsstufen)

  `salgrade(grade, losal°, hisal°)`

- Link zum Adminer (Schema „`empdept_public`“):

  https://dbs.informatik.uni-halle.de/edb?pgsql=db&username=student_gast&db=postgres&ns=empdept_public

Aufgabe 7.1: Gehaltsvergleich (1)

- Wer verdient mehr als sein direkter Vorgesetzter?

  Die Angestelltennummer (`empno`) des Vorgesetzten steht in der Spalte `mgr` („Manager“). Das Gehalt steht in der Spalte `sal` („Salary“).

- Geben Sie Name und Gehalt des Untergebenen und Name und Gehalt des Vorgesetzten aus, und nennen Sie die Ergebnisspalten wie in der Beispiel-Ausgabe gezeigt.

  Die Spalten sollen heißen: (1) Angestellter, (2) Gehalt, (3) Vorgesetzter, (4) Gehalt des Vorgesetzten. In der Beispiel-Ausgabe auf der nächsten Folie ist die letzte Spalte etwas gekürzt.

- **Sortieren** Sie das Ergebnis nach dem Namen des Angestellten.

Aufgabe 7.1: Gehaltsvergleich (2)

- Erwartetes Ergebnis:

Angestellter	Gehalt	Vorgesetzter	Gehalt d.V.
FORD	3000	JONES	2975
SCOTT	3000	JONES	2975

Aufgabe 7.2: Gehaltsspanne in Abteilungen

- In welchen Abteilungen gibt es Angestellte, die um mindestens drei Gehaltsstufen auseinander liegen?

  Die Gehaltsstufen sind in der Tabelle `salgrade` definiert: Wenn ein Gehalt zwischen den Werten `losal` und `hisal` liegt (einschließlich der beiden Grenzen), wird der Angestellte der Gehaltsstufe `grade` zugeordnet. Gesucht sind Abteilungen, in denen es Angestellte mit Gehaltsstufen gibt, deren Differenz wenigstens 3 ist.

- Geben Sie Nummer und Name der jeweiligen Abteilung aus und **sortieren** Sie das Ergebnis nach der Abteilungsnummer.

deptno	dname
10	ACCOUNTING
20	RESEARCH
30	SALES

Aufgabe 7.3: Manager/Clerk pro Abteilung (1)

- Geben Sie für jede Abteilung, die einen „MANAGER“ und einen „CLERK“ (Sachbearbeiter, Sekretär) hat, Nummer und Name der Abteilung und den Namen des Managers und den Namen des Sachbearbeiters aus.

  Gemeint sind natürlich immer männliche und weibliche Personen mit diesen Berufsbezeichnungen.

- Es ist in Ordnung, dass, falls eine Abteilung mehrere Manager und mehrere Sachberarbeiter haben sollte, jede Kombination ausgegeben wird.

  Im Beispielzustand hat jede Abteilung aber nur jeweils einen Angestellten mit diesen Berufsbezeichnungen.

- Verwenden Sie die Spaltennamen wie im Beispiel-Ergebnis und **sortieren** Sie die Ausgabe nach der Abteilungsnummer.

Aufgabe 7.3: Manager/Clerk pro Abteilung (2)

- Erwartetes Ergebnis:

deptno	dname	Manager	Clerk
10	ACCOUNTING	CLARK	MILLER
20	RESEARCH	JONES	ADAMS
30	SALES	BLAKE	JAMES

Aufgabe 7.4: Angestellte ohne Untergebene (1)

- Welche Angestellten haben selbst keinen Untergebenen?

  Gesucht sind also Angestelle `e`, so dass es keinen Angestellten `x` gibt, dessen direkter Vorgesetzter `e` ist. Die Angestellten-Nummer `empno` des direkten Vorgesetzten `e` würde in der Spalte `mgr` des Untergebenen `x` stehen.

- Geben Sie Angestellten-Nummer (`empno`), Name (`ename`), Berufsbezeichung (`job`) und Gehalt (`sal`) der gesuchten Angestellten aus.

- **Sortieren** Sie die Ausgabe absteigend nach dem Gehalt (also größtes Gehalt zuerst) und bei gleichem Gehalt nach der Angestellten-Nummer (normal, d.h. aufsteigend).

Aufgabe 7.4: Angestellte ohne Untergebene (2)

- Das erwartete Ergebnis ist:

empno	ename	job	sal
7499	ALLEN	SALESMAN	1600
7844	TURNER	SALESMAN	1500
7934	MILLER	CLERK	1300
7521	WARD	SALESMAN	1250
7654	MARTIN	SALESMAN	1250
7876	ADAMS	CLERK	1100
7900	JAMES	CLERK	950
7369	SMITH	CLERK	800

Aufgabe 7.5: Obere Gehaltshälfte (1)

- Welche Angestellten verdienen mindestens die Hälfte vom Top-Verdiener in der Firma?

  Sie müssen also einen Angestellten `t` mit maximalem Gehalt finden (d.h. es gibt keinen, der mehr verdient) und dann alle Angestellten `e` ausgeben, die mindestens 50% des Gehalts von `t` haben.

- Drucken Sie Nummer und Namen des Angestellten, sowie Berufsbezeichnung und Gehalt.

- **Sortieren** Sie die Ausgabe absteigend nach dem Gehalt (also größtes Gehalt zuerst) und bei gleichem Gehalt nach dem Angestellten-Namen.

Aufgabe 7.5: Obere Gehaltshälfte (2)

- Die Anfrage soll keine Duplikate liefern (in keinem DB-Zustand), aber Sie sollen auch kein unnötiges `DISTINCT` verwenden.

  Begründen Sie Ihre Entscheidung, ob Sie eine Duplikat-Eliminierung brauchen oder nicht (in einem kurzen Kommentar).

- Das erwartete Ergebnis ist:

empno	ename	job	sal
7839	KING	PRESIDENT	5000
7902	FORD	ANALYST	3000
7788	SCOTT	ANALYST	3000
7566	JONES	MANAGER	2975
7698	BLAKE	MANAGER	2850

Inhalt

Datenbank: Komponisten, Musik-CDs

- Die Aufgaben dieses Übungsblattes beziehen sich auf die Datenbank mit Informationen über klassische Musik-CDs:

  https://dbs.informatik.uni-halle.de/edb?pgsql=db&username=student_gast&db=postgres&ns=komponist_public

- KOMPONIST(KNR, NAME, VORNAME°, GEBOREN°, GESTORBEN°)
- STUECK(SNR, KNR°→KOMPONIST, TITEL, TONART°, OPUS°)
- CD(CDNR, NAME, HERSTELLER°, ANZ_CDS°, GESAMTSPIELZEIT°)
- AUFNAHME(CDNR→CD, SNR→STUECK, ORCHESTER°, LEITUNG°)
- SOLIST((CDNR, SNR)→AUFNAHME, NAME, INSTRUMENT°)

Aufgabe 6.1: Zeitgenossen von Mozart (1)

- Gesucht sind alle Komponisten in der Datenbank, die Zeitgenossen von Wolfgang Amadeus Mozart sind, deren Lebenszeit also mit der von Mozart überlappt.

  Es sollen dabei auch Komponisten ausgegeben werden, die in dem Jahr gestorben sind, in dem Mozart geboren wurde, bzw. die in dem Jahr geboren wurden, in dem Mozart gestorben ist (wenn die Überlappung also etwas grenzwertig ist).

- „Wolfgang Amadeus Mozart“ selbst soll in der Auflistung nicht erscheinen.

  Sie können voraussetzen, dass Name und Vorname des Komponisten zusammen einen Alternativschlüssel darstellen, dass es also nicht zwei verschiedene Einträge mit gleichem Namen und Vornamen gibt.

Aufgabe 6.1: Zeitgenossen von Mozart (2)

- Geben Sie Name, Vorname, Geburts- und Todesjahr der Komponisten aus, sowie das „Alter“ des Komponisten im Geburtsjahr von Mozart.

  Das kann natürlich negativ sein, wenn der andere Komponist erst nach Mozart geboren wurde.

- Die Spalte `Alter` nennen Sie bitte in der Ausgabe genau so (auch mit dieser Groß-/Kleinschreibung).

- Sie dürfen nicht die Lebensdaten von Mozart in die Anfrage einsetzen.

  Zwar werden die sich nicht mehr ändern (1756–1791), aber der Name „Wolfgang Amadeus Mozart“ soll nur ein Beispiel sein. Ihre Anfrage könnte z.B. später in einem Programm verwendet werden, mit Parametern für den tatsächlichen Namen.

Aufgabe 6.1: Zeitgenossen von Mozart (3)

name	vorname	geboren	gestorben	Alter
Telemann	Georg Philipp	1681	1767	75
Händel	Georg Friedrich	1685	1759	71
Scarlatti	Domenico	1685	1757	71
Locatelli	Pietro	1695	1764	61
Leclair	Lean-Marie	1697	1764	59
Mozart	Leopold	1719	1787	37
Hayden	Joseph	1732	1809	24
Beethoven	Ludwig van	1770	1827	-14

Aufgabe 6.1: Zeitgenossen von Mozart (4)

- Verwenden Sie bitte nur Konstrukte, die in der Vorlesung schon behandelt wurden.
- Insbesondere sind Joins unter `FROM` ausgeschlossen.

  In der Klausur gibt es natürlich keine solchen Beschränkungen (sehr ungewöhnliche Konstrukte könnten aber zu Punktabzügen wegen mangelnder Portabilität führen).
- Wenn in der Aufgabenstellung keine spezielle Sortierung verlangt ist (wie hier), dürfen Sie eine beliebige (sinnvolle) Sortierung vornehmen (oder eben nicht sortieren).
- Die obige Tabelle ist mit

  ```
  ORDER BY GEBOREN, NAME
  ```

  (am Ende der Anfrage) erzeugt.

Aufgabe 6.1: Zeitgenossen von Mozart (5)

- Lösung:

```
SELECT K.NAME, K.VORNAME,
       K.GEBOREN, K.GESTORBEN,
       M.GEBOREN - K.GEBOREN AS "ALTER"
FROM   KOMPONIST M, KOMPONIST K
WHERE  M.NAME = 'Mozart'
AND    M.VORNAME = 'Wolfgang Amadeus'
AND    NOT(K.GESTORBEN < M.GEBOREN
           OR M.GESTORBEN < K.GEBOREN)
AND    K.KNR <> M.KNR
ORDER  BY "ALTER"
```

Man kann natürlich das DeMorgan'sche Gesetz anwenden und erhält:

`K.GESTORBEN >= M.GEBOREN AND K.GEBOREN <= M.GESTORBEN`.

Die Sortierung war nicht verlangt.

Aufgabe 6.2: CDs mit mehreren Komponisten (1)

- Gesucht sind alle CDs, die Stücke von mehr als einem Komponisten enthalten.

  Also von mindestens zwei verschiedenen Komponisten.

- Geben Sie jeweils den Namen der CD aus.
- Bei dieser Anfrage werden Sie wahrscheinlich Duplikate erhalten. Verwenden Sie ggf. `SELECT DISTINCT`, um diese zu eliminieren.
- Für die Lösung dieser Aufgabe verwenden Sie bitte kein `GROUP BY` und keine Aggregationsfunktionen wie `COUNT`, selbst wenn Sie das schon kennen sollten.

  In der Vorlesung war es noch nicht dran.

Aufgabe 6.2: CDs mit mehreren Komponisten (2)

- Das erwartete Ergebnis ist:

cdnr	name
106	Leopold Mozart: Sinfonia D-Dur ...
111	Mozart/Beethoven: Klassische Ouvertüren
125	Tschaikowsky/Mendelssohn: Violinkonzerte
139	Oboenkonzerte
140	Corelli,Albinoni,Scarlatti,Manfredini,...
142	Schlager um 1500

Aufgabe 6.2: CDs mit mehreren Komponisten (3)

- Dies ist ein Beispiel für einen Selbstverbund.
- Lösung:

```
SELECT DISTINCT C.NAME
FROM   CD C, AUFNAHME A1, STUECK S1,
       AUFNAHME A2, STUECK S2
WHERE  C.CDNR = A1.CDNR AND C.CDNR = A2.CDNR
AND    A1.SNR = S1.SNR AND A2.SNR = S2.SNR
AND    S1.KNR <> S2.KNR
```

Es ist überflüssig, noch zwei Tupelvariablen über `KOMPONIST` hinzu zu nehmen. Die die unterschiedlichen Komponistennummern ist schon garantiert, dass es sich um unterschiedliche Komponisten handelt. Ohne `DISTINCT` liefert die Anfrage 266 Zeilen, mit `DISTINCT` 6.

Aufgabe 6.3: Logische Analyse (1)

- Im folgenden sollen Sie jeweils zwei Anfragen vergleichen, u.a. hinsichtlich Äquivalenz.
- Anfrage 1:

```
SELECT KNR, SNR
FROM   STUECK
WHERE  KNR = 70
```

- Anfrage 2:

```
SELECT K.KNR, S.SNR
FROM   STUECK S, KOMPONIST K
WHERE  S.KNR = 70 AND K.KNR = 70
```

Aufgabe 6.3: Logische Analyse (2)

- Welche der folgenden Aussagen ist korrekt? Wenn mehrere korrekt sein sollten, wählen Sie die erste korrekte Aussage.

  A. Die beiden Anfragen liefern immer die gleiche Antwort (äquivalent)
  B. Die beiden Anfragen liefern bis auf Duplikate die gleiche Antwort
  C. Das Ergebnis von Anfrage 1 ist immer leer (inkonsistent)
  D. Das Ergebnis von Anfrage 2 ist immer leer (inkonsistent)
  E. Anfrage 1 liefert immer eine Obermenge ($\supseteq$) von Anfrage 2
  F. Anfrage 1 liefert immer eine Teilmenge ($\subseteq$) von Anfrage 2
  G. Keine der Aussagen trifft zu

  Bei „B.“ ist gemeint, dass die Tupelmengen (nach Duplikat-Eliminierung) identisch sind. Die Teile „E.“ und „F.“ beziehen sich auch auf die Tupel-Mengen ohne Berücksichtigung eventueller Duplikate.

Aufgabe 6.3: Logische Analyse (3)

- Geben Sie den Buchstaben der ersten korrekten Aussage ab sowie eine kurze Begründung.

  Maximal 3 Zeilen, die entscheidenen Stichworte reichen. Bitte laden Sie eine `.txt`-Datei mit Ihrer Antwort hoch (kein PDF, kein Word).

- Beachten Sie, dass Sie alle möglichen Datenbank-Zustände in Betracht ziehen müssen, nicht nur den aktuellen Zustand.

  Sie können aber voraussetzen, dass Schlüssel, Fremdschlüssel und die `NOT NULL`-Bedingungen aus dem Schema erfüllt sind (und auch die Datentypen so sind, wie im Adminer angegeben).

- Es ist daher nur begrenzt hilfreich, die Anfrage auszuprobieren.

  In der Klausur könnte es ähnliche Aufgaben auch zu einem Schema geben, das nicht im Adminer zur Verfügung steht. Es sind also „theoretische“ Aufgaben, die durch Nachdenken gelöst werden sollten.

Aufgabe 6.3: Logische Analyse (4)

Lösung:

- A. Die beiden Anfragen liefern immer die gleiche Antwort.
- Begründung:
  - Wenn es mindestens ein Stück `S` mit `S.KNR = 70` gibt, gibt es genau eine Zeile `K` in `KOMPONIST` mit `K.KNR = 70`.

    Aufgrund des Fremdschlüssels `STUECK(...,KNR`$\rightarrow$`KOMPONIST, ...)` muss es mindestens eine solche Zeile geben, aufgrund des Schlüssels `KOMPONIST(KNR, ...)` höchstens eine.
  - Wenn es kein solches Stück gibt, sind beide Anfrage-Ergebnisse leer.

Aufgabe 6.3: Logische Analyse (5)

Lösung, Forts.:

- Ausführlichere Begründung:
  - Jede Variablenbelegung $\mathcal{A}$ für S, die bei der ersten Anfrage eine Ausgabe erzeugt, kann also zu einer Variablenbelegung $\mathcal{A}'$ für K und S erweitert werden, die bei der zweiten Anfrage die gleiche Ausgabe erzeugt.

    Und es gibt genau eine solche Erweiterung. Es kommen keine Duplikate hinzu. Die Abbildung zwischen den Mengen von Variablenbelegungen mit Ausgaben ist bijektiv.
  - Hat man umgekehrt eine Variablenbelegung $\mathcal{A}'$, die bei der zweiten Anfrage eine Ausgabe erzeugt, vergisst man einfach die Belegung der Variablen K und erhält eine Variablenbelegung $\mathcal{A}$, die bei der ersten Anfrage die gleiche Ausgabe erzeugt.

Aufgabe 6.4: Logische Analyse (1)

- Vergleichen Sie nach dem Schema von Aufgabe 3 die folgenden beiden Anfragen:
- Anfrage 1:

```
SELECT *
FROM   KOMPONIST
WHERE  KNR = 70 OR KNR = 80 AND GEBOREN < 1500
```

- Anfrage 2:

```
SELECT *
FROM   KOMPONIST
WHERE  KNR IN (70, 80) AND NOT GEBOREN >= 1500
```

Aufgabe 6.4: Logische Analyse (2)

Lösung:

- E. Anfrage 1 liefert immer eine Obermenge von Anfrage 2.

  Für alle DB-Zustände $\mathcal{I}$ gilt also: $\mathcal{I}$[Anfrage 1] $\supseteq$ $\mathcal{I}$[Anfrage 2].

- Da `OR` schwächer bindet als `AND`, gibt die Anfrage 1 einen Komponisten mit `KNR = 70` auch dann aus, wenn er nicht vor 1500 geboren ist.

- Anfrage 2 ist dagegen äquivalent zu:

```
SELECT *
FROM   KOMPONIST
WHERE  (KNR = 70 OR KNR = 80) AND GEBOREN < 1500
```

- Die Bedingung impliziert die Bedingung von Anfrage 1:

```
WHERE KNR = 70 OR (KNR = 80 AND GEBOREN < 1500)
```

Aufgabe 6.5: Logische Analyse (1)

- Vergleichen Sie entsprechend die folgenden beiden Anfragen:
- Anfrage 1:

```
SELECT *
FROM   STUECK
WHERE  TONART = 'C-dur'
AND    (TONART = 'a-moll'
        OR TITEL LIKE 'Symphon%')
```

- Anfrage 2:

```
SELECT *
FROM   STUECK
WHERE  TONART = 'C-dur'
AND    TITEL LIKE 'Symphon%'
```

Aufgabe 6.5: Logische Analyse (2)

Lösung:

- A. Die beiden Anfragen liefern immer die gleiche Antwort.
- Wenn die Tonart „C-dur“ ist, kann sie nicht gleichzeitig „a-moll“ sein.
- Die erste Hälfte der Disjunktion in Anfrage 1 kann also nie erfüllt sein, wenn die `WHERE`-Bedingung als Ganzes wahr ist.
- Daher kann man sie auch weglassen.
- Genauere Erklärung auf der nächsten Folie.

Aufgabe 6.5: Logische Analyse (3)

Lösung, Forts.:

- Genauer kann man die `WHERE`-Bedingung der Anfrage 1 mit dem Distributiv-Gesetz äquivalent umformen:

```
SELECT *
FROM   STUECK
WHERE  (TONART = 'C-dur'
        AND TONART = 'a-moll')
OR     (TONART = 'C-dur'
        AND TITEL LIKE 'Symphon%')
```

- Der erste Teil der Disjunktion ist äquivalent zu `false`, und `false OR F` ist äquivalent zu `F`.

Inhalt

Präsenzaufgabe 7: Duplikate

- Kann diese Anfrage Duplikate liefern?

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.PUNKTE = 10
AND    B.ATYP = 'H'
```

Es sei angenommen, dass `VORNAME`, `NACHNAME` zusammen ein Alternativschlüssel der Tabelle `STUDENTEN` sind.

- ☐ Ja ☐ Nein
  - Falls Ja: Geben Sie einen DB-Zustand an, in dem die Anfrage ein Duplikat liefert.
  - Falls Nein: Beweisen Sie die Aussage, z.B. mit dem Algorithmus aus der Vorlesung.