Einführung in Datenbanken

Übung 7: Duplikate in SQL

Prof. Dr. Stefan Brass
PD Dr. Alexander Hinneburg

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2020/21

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

Inhalt

Hausaufgabe 5a (1)

- Geben Sie alle Stücke von Wolfgang Amadeus Mozart aus der Datenbank aus, die in einer dur-Tonart geschrieben sind.

  D.h. `TONART` endet mit „`dur`“. Stücke mit unbekannter Tonart können dabei ignoriert werden. Drucken Sie `TITEL`, `TONART` und `OPUS`.

- KOMPONIST(KNR, NAME, VORNAME, GEBOREN, GESTORBEN°)
- STUECK(SNR, KNR→KOMPONIST, TITEL, TONART°, OPUS°)
- Die erwartete Antwort ist:

titel	tonart	opus
‘Salzburger Sinfonie’ Nr.1	D-dur	KV 136
‘Salzburger Sinfonie’ Nr.2	B-dur	KV 137
‘Salzburger Sinfonie’ Nr.3	F-dur	KV 138
‘Eine kleine Nachtmusik’	G-dur	KV 525

Hausaufgabe 5a (2)

- Lösung:

```
SELECT titel, tonart, opus
FROM   stueck, komponist
WHERE  name = 'Mozart'
AND    vorname = 'Wolfgang Amadeus'
AND    stueck.knr = komponist.knr
AND    tonart like '%dur'
```

- Dies ist eine Variante ohne explizite Tupelvariablen.

  Die implizit angelegten Tupelvariablen heißen wie die Tabellen, also `stueck` und `komponist`.

- Bei den Attributreferenzen wurde die Tupelvariable nur geschrieben, wenn es unbedingt nötig ist, das Attribut also in beiden Tabellen vorkommt.

  Nur `knr` kann nicht eindeutig einer Tupelvariable zugeordnet werden.

Hausaufgabe 5b (1)

- Geben Sie alle Sinfonien in der Datenbank aus, das sind Stücke, deren Titel „`Sinf`“ oder „`Symph`“ als Teilzeichenkette enthält.

  Drucken Sie Vorname und Name des Komponisten zusammen in einer Spalte (durch Leerzeichen getrennt) sowie den Titel des Stücks.

- Erwartete Antwort:

komponist	titel
Georg Friedrich Händel	Sinfonia e-moll (aus ...)
Leopold Mozart	Sinfonia D-Dur mit ...
⋮	⋮

Hausaufgabe 5b (2)

- Lösung:

```
SELECT k.vorname || ' ' || k.name AS komponist,
       s.titel
FROM   stueck s, komponist k
WHERE  (titel LIKE '%Sinf%' OR
        titel LIKE '%Symph%')
AND    k.knr = s.knr
```

- Die Klammern um die beiden mit „OR“ verknüpften Teilbedingungen sind nötig, da AND stärker bindet als OR.

  Damit würde ohne Klammern die Join-Bedingung nur für die „Symph“ gelten. Eine Sinfonie dagegen würde mit allen Komponisten verbunden, auch wenn der Komponist diese Sinfonie gar nicht geschrieben hat.

Hausaufgabe 5c (1)

- Gesucht sind alle Komponisten aus der Datenbank, die im Jahr 1700 gelebt haben.

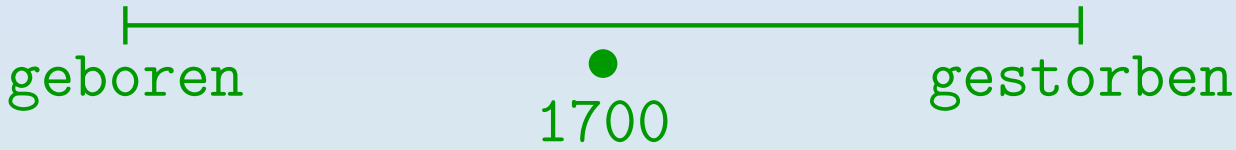
  D.h. das Intervall von Geburtsjahr bis Todesjahr enthält das Jahr 1700, wobei auch Komponisten gemeint sind, die 1700 geboren wurden oder gestorben sind. Drucken Sie Name, Vorname, Geburtsjahr, Todesjahr und das Alter aus (approximiert als Differenz von Todesjahr und Geburtsjahr). Die Spalte mit dem Alter soll `Alter` heissen (mit dieser Groß-/Kleinschreibung).

- Erwartete Antwort:

name	vorname	geboren	gestorben	Alter
Händel	Georg Friedrich	1685	1759	74
Vivaldi	Antonio	1678	1741	63
⋮	⋮	⋮	⋮	⋮

Hausaufgabe 5c (2)

- Lebensintervall:



- Lösung:

```
SELECT name, vorname, geboren, gestorben,
       gestorben-geboren AS "Alter"
FROM   komponist
WHERE  geboren <= 1700 AND gestorben >= 1700
```

- Delimited Identifier "..." für den Spaltennamen „Alter“ ist nötig, da Groß-/Kleinschreibung vorgeschrieben war.

Inhalt

Präsenzaufgabe: System-Katalog (1)

- Es sind alle Tabellen gesucht, die in Fremdschlüsseln die Tabelle `studenten` referenzieren.

  Es soll hier der „native“ Systemkatalog von PostgreSQL genutzt werden.

- Tabellen sind in `pg_catalog.pg_class` eingetragen.
  Wir benötigen nur die Spalten `relname` (Name der Tabelle) und `oid` (Interne Nummer, Object-ID).

  [https://www.postgresql.org/docs/9.1/catalog-pg-class.html]

- Constraints stehen in `pg_catalog.pg_constraint`.

  [https://www.postgresql.org/docs/9.1/catalog-pg-constraint.html]

  - `conrelid`: OID der Tabelle mit diesem Constraint.

    Den Namen dieser Tabelle sollen Sie ausgeben.

  - `contype`: Art des Constraints, z.B. `'f'` für Fremdschlüssel.
  - `confrelid`: OID der referenzierten Tabelle.

    Dies soll die Tabelle `studenten` sein.

Präsenzaufgabe: System-Katalog (2)

- Erwartetes Ergebnis:

relname
bewertungen

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

Die Zugangsdaten unserer Installation stehen in StudIP, Reiter „Adminer“.

- `bewertungen( sid`$\rightarrow$`studenten, (atyp,anr)`$\rightarrow$`aufgaben, punkte)`
- Anfragen müssen nicht nur mit dem aktuellen Zustand funktionieren, sondern mit beliebigen DB-Zuständen.
- Das Schema „`pg_catalog`“ wird immer zuerst durchsucht, wenn es nicht explizit in `search_path` ist.

D.h. man kann sich den Schema-Präfix bei diesen Katalog-Tabellen sparen.

Präsenzaufgabe: System-Katalog (3)

Tipps:

- Wenn Sie

  ```
  SELECT * FROM pg_class
  ```

  eingeben, wird die Spalte `oid` nicht angezeigt.

  Sie ist eine spezielle interne Spalte (ein wenig versteckt).

- Es gibt diese Spalte aber. Z.B. funktioniert Folgendes:

  ```
  SELECT stud.oid
  FROM   pg_class stud
  WHERE  stud.relname = 'studenten'
  ```

- Beim Verbund müssen die Spaltennamen nicht identisch sein, z.B. funktioniert

  ```
  fk.confrelid = stud.oid
  ```

  wenn `fk` eine Tupelvariable über `pg_constraint` ist.

Lösung der Präsenzaufgabe (1)

- Man überlege sich ein Beispiel für Tabellenzeilen, die nötig sind, um eine Ausgabezeile zu produzieren.

	pg_class	
	oid	relname
stud →	16415	studenten
fk_tab →	16426	bewertungen

	pg_constraint		
	contype	conrelid	confrelid
fk →	f	16426	16415

Lösung der Präsenzaufgabe (2)

- Für jede der drei Zeilen führt man eine Tupelvariable ein:

```
SELECT fk_tab.relname
FROM   pg_class fk_tab,
       pg_class stud,
       pg_constraint fk
WHERE  stud.relname = 'studenten'
AND    fk.contype = 'f'
AND    fk.conrelid = fk_tab.oid
AND    fk.confrelid = stud.oid
```

Lösung der Präsenzaufgabe (3)

- Abgabe mit Unteranfrage (erst am Montag in der Vorlesung):

```
SELECT cls.relname
FROM   pg_class cls
WHERE  cls.oid = (SELECT con.conrelid
                  FROM   pg_constraint con,
                         pg_class cls
                  WHERE  con.contype = 'f'
                  AND    con.frelid = cls.oid
                  AND    cls.relname='studenten')
```

- Funktioniert nicht, wenn es Fremdschlüssel nach `STUDENTEN` von mehreren anderen Tabellen gibt.
- Verschattete Tupelvariable `cls` (außen) macht das Verstehen der Anfrage nicht leichter (schlechter Stil).

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`.

Algorithmus bei mehreren Tupelvariablen

- Am Anfang wird $\mathcal{K}$ wie gehabt initialisiert mit
  - Attributreferenzen $X.A$ als Ausgabespalten unter `SELECT`
  - Attributreferenzen, die aufgrund der `WHERE`-Bedingung nur einen Wert annehmen können: $X.A = c$.
- Anschließend wird $\mathcal{K}$ mit den folgenden beiden Regeln erweitert (iteriert, bis sich nichts mehr ändert):
  - Wenn die `WHERE`-Bedingung eine Gleichung $X.A = Y.B$ enthält (als Teil einer `AND`-Liste), und $X.A$ ist schon in $\mathcal{K}$ enthalten, wird $Y.B$ hinzugefügt (und umgekehrt).
  - Wenn ein Schlüssel einer Tupelvariable enthalten ist, werden alle anderen Attribute dieser Tupelvariable auch hinzugefügt.

Beispiel mit mehreren Tupelvariablen (1)

- Kann diese Anfrage Duplikate liefern?

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

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

  Diese Attributreferenzen stehen explizit unter `SELECT`.

- $\mathcal{K} := \mathcal{K} \cup \{\texttt{B.ATYP}, \texttt{B.ANR}\}$

  Wegen der `WHERE`-Bedingungen `B.ATYP = 'H'` und `B.ANR = 1`, die für alle Variablenbelegungen gelten müssen (d.h. nicht durch `OR` ausgehebelt).

- $\mathcal{K} := \mathcal{K} \cup \{\texttt{S.SID}, \texttt{S.EMAIL}\}$

  $\{\texttt{S.VORNAME}, \texttt{S.NACHNAME}\} \subseteq \mathcal{K}$ bilden einen Schlüssel von `S`.

Beispiel mit mehreren Tupelvariablen (2)

- $\mathcal{K} := \mathcal{K} \cup \{\texttt{B.SID}\}$

  Wegen der `WHERE`-Bedingung `S.SID = B.SID` und $\texttt{S.SID} \in \mathcal{K}$.

- $\mathcal{K} := \mathcal{K} \cup \{\texttt{B.PUNKTE}\}$

  $\{\texttt{B.SID}, \texttt{B.ATYP}, \texttt{B.ANR}\} \subseteq \mathcal{K}$ bilden einen Schlüssel von `B`.

- Nun enthält $\mathcal{K}$ alle Attribute von allen Tupelvariablen.
- Damit ist garantiert, dass die Anfrage keine Duplikate liefern kann.
- Der Algorithmus ist nicht prüfungsrelevant.

  Der Algorithmus ist (vermutlich) von Brass/Goldberg.

- Die Fähigkeit, festzustellen, ob eine Anfrage Duplikate liefern kann, schon.

Aufgabe zu Duplikaten (1)

- 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'
```

- Ja/Nein-Abstimmung.
- Falls Nein: Beweisen Sie die Aussage, z.B. mit dem obigen Algorithmus.
- Falls Ja: Geben Sie einen Datenbankzustand an, in dem die Anfrage ein Duplikat liefert.

Aufgabe zu Duplikaten (2)

- Ja! (Student/in mit 10 Punkten für HA 1 und HA 2)

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

- $\mathcal{K} := \{\texttt{S.VORNAME}, \texttt{S.NACHNAME}\}$
- $\mathcal{K} := \mathcal{K} \cup \{\texttt{B.PUNKTE}, \texttt{B.ATYP}\}$
- $\mathcal{K} := \mathcal{K} \cup \{\texttt{S.SID}, \texttt{S.EMAIL}\}$

  Weil $\{\texttt{S.VORNAME}, \texttt{S.NACHNAME}\} \subseteq \mathcal{K}$ ein Schlüssel von S sind.

- $\mathcal{K} := \mathcal{K} \cup \{\texttt{B.SID}\}$
- Kein Schlüssel von B (es fehlt B.ANR).

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

Logische Analyse von Anfragen (5)

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

2.
```
SELECT Y.SID, X.ANR AS NR
FROM   AUFGABEN X, BEWERTUNGEN Y
WHERE  X.MAXPT = Y.PUNKTE
AND    X.ATYP='H' AND Y.ATYP='H' AND X.ANR=Y.ANR
```

Kreuzen Sie die erste zutreffende Antwort an:

A. Die Anfragen liefern immer die gleiche Antwort (äquivalent).
B. Bis auf Duplikate die gleiche Antwort.
C. Anfrage 1 liefert immer $\emptyset$ (inkonsistent).
D. Anfrage 2 liefert immer $\emptyset$ (inkonsistent).
E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (6)

1. ```
   SELECT A.ANR, A.THEMA
   FROM   AUFGABEN A
   WHERE  A.ATYP = 'H'
   ```

2. ```
   SELECT DISTINCT A.ANR, A.THEMA
   FROM   AUFGABEN A, BEWERTUNGEN B
   WHERE  A.ATYP = B.ATYP AND B.ATYP = 'H'
   ```

Kreuzen Sie die erste zutreffende Antwort an:

A. Die Anfragen liefern immer die gleiche Antwort (äquivalent).

B. Bis auf Duplikate die gleiche Antwort.

C. Anfrage 1 liefert immer eine Teilmenge von Anfrage 2 ($\subseteq$).

D. Anfrage 1 liefert immer eine Obermenge von Anfrage 2 ($\supseteq$).

E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (7)

1. ```
   SELECT VORNAME, NACHNAME
   FROM   STUDENTEN
   WHERE  VORNAME = 'Lisa'
   ```

2. ```
   SELECT 'Lisa' AS VORNAME, NACHNAME
   FROM   STUDENTEN
   WHERE  NOT 1=0 AND 'Lisa' = VORNAME
   ```

Kreuzen Sie die erste zutreffende Antwort an:

A. Die Anfragen liefern immer die gleiche Antwort (äquivalent).

B. Bis auf Duplikate die gleiche Antwort.

C. Anfrage 1 liefert immer eine Teilmenge von Anfrage 2 ($\subseteq$).

D. Anfrage 1 liefert immer eine Obermenge von Anfrage 2 ($\supseteq$).

E. Keine der Aussagen trifft zu.

Inhalt

Präsenzaufgabe: Duplikate (1)

- Kann die folgende Anfrage Duplikate liefern?

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN LISA, STUDENTEN S,
       BEWERTUNGEN H1L, BEWERTUNGEN H1S
WHERE  LISA.VORNAME = 'Lisa'
AND    LISA.NACHNAME = 'Weiss'
AND    H1L.SID = LISA.SID
AND    H1S.SID = S.SID
AND    H1L.ATYP = 'H' AND H1S.ATYP = 'H'
AND    H1L.ANR  =  1  AND H1S.ANR  =  1
AND    H1S.PUNKTE > H1L.PUNKTE
```

Übliche Schlüssel: {SID} und {VORNAME, NACHNAME} von STUDENTEN, {SID, ATYP, ANR} von BEWERTUNGEN.

- Ja/Nein reicht (1 Punkt), Bonuspunkt für gute Begründung.

Präsenzaufgabe: Duplikate (2)

- Kann die folgende Anfrage Duplikate liefern?

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN LISA, STUDENTEN S,
       BEWERTUNGEN H1L, BEWERTUNGEN H1S
WHERE  LISA.VORNAME = 'Lisa'
AND    LISA.NACHNAME = 'Weiss'
AND    H1L.SID = LISA.SID
AND    H1S.SID = S.SID
AND    H1L.ATYP = 'H' AND H1S.ATYP = 'H'
AND    H1L.ANR = H1S.ANR
AND    H1S.PUNKTE > H1L.PUNKTE
```

Übliche Schlüssel: {SID} und {VORNAME, NACHNAME} von STUDENTEN, {SID, ATYP, ANR} von BEWERTUNGEN.

- Ja/Nein plus gute Begründung (2 Punkte)

Präsenzaufgabe: Duplikate (3)

- Anfrage 1:

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN LISA, STUDENTEN S,
       BEWERTUNGEN H1L, BEWERTUNGEN H1S
WHERE  LISA.VORNAME = 'Lisa'
AND    LISA.NACHNAME = 'Weiss'
AND    H1L.SID = LISA.SID
AND    H1S.SID = S.SID
AND    H1L.ATYP = 'H' AND H1S.ATYP = 'H'
AND    H1L.ANR  =  1  AND H1S.ANR  =  1
AND    H1S.PUNKTE > H1L.PUNKTE
```

- Anfrage 2: Wie Anfrage 1, aber rote Zeile ersetzt durch:

```
AND    H1L.ANR = H1S.ANR
```