Einführung in Datenbanken

Übung 9: Stilfragen, NOT EXISTS

Prof. Dr. Stefan Brass
PD Dr. Alexander Hinneburg
Martin-Luther-Universität Halle-Wittenberg
Wintersemester 2023/24

http://www.informatik.uni-halle.de/~brass/db23/

Inhalt

Organisatorisches (1)

- Ich bin auf einen ersten Plagiatsfall aufmerksam gemacht worden.
- Es war ein sehr fortschrittliches Konstrukt verwendet worden, das der Tutor nicht kannte, der mich deshalb gefragt hat.
- Zwei der Anfragen mit diesem Konstrukt waren identisch.
- Das zwingt mich, genauer nach Plagiaten zu suchen, auch in zurückliegenden Hausaufgaben.
- Sie dürfen Lösungen kopieren, oder von ChatGPT erstellen lassen, aber *NUR* mit Nennung der Quelle.
- Ansonsten bekommen alle Beteiligten 0 Punkte, im Wiederholungsfall droht Schlimmeres.

Organisatorisches (2)

- Ich hätte die Studierenden ja gerne vorrechnen lassen, um herauszufinden, ob mindestens einer sich mit diesem Konstrukt tatsächlich gut auskennt.
- Leider war auch kein `--Vorrechnen` Kommentar da.
- Es besteht ein gewisses Risiko, dass, wenn die 20% Quote nicht annähernd erreicht wird, wir deshalb die Studienleistung verweigern.

  Die Regelung ist noch etwas experimentell, aber lassen Sie es nicht darauf ankommen, ob wir am Ende wirklich tun, was wir am Anfang gesagt haben.
- Vergessen Sie also nicht, bei Lösungen, die Sie erklären können, mindestens `--Vorrechnen:2` dazu zu schreiben.

  Voraussetzung ist, dass Sie zur Übung kommen werden (sonst `--Vorrechnen:0`).

Organisatorisches (3)

- Erklärungen für hohe Anzahl von `JOIN` in Hausaufgaben (noch immer nicht in Vorlesung behandelt)?
- Wer hat andere Quellen verwendet als Vorlesung und Skript (Mehrfachnennungen möglich)?
  - A. SQL schon in Schule/Ausbildung gelernt.
  - B. Andere (frühere) Vorlesung.
  - C. Früherer Durchlauf dieser Vorlesung oder Skript voraus gelesen.
  - D. Lehrbuch
  - E. Andere Vorlesungen im Internet (Youtube/Skript).
  - F. Sonstige Tutorials im Internet/Youtube-Videos.

Zu den Klausuraufgaben (1)

- Aufgaben in der letzten Klausur (48+3 Punkte):
  - 5 SQL-Anfragen (je 5 Punkte).

    Typische Länge jeweils 10 Zeilen. Nötige Konstrukte u.a.:
    Selbstverbund (mehrere Tupelvariablen über der gleichen Relation),
    `LIKE`, `UPPER`, `OR`, `IS NULL`, `NOT EXISTS`, `SELECT DISTINCT`, `AS "..."`,
    Aggregationsfunktionen `MIN`/`MAX`/`AVG`/`SUM`/`COUNT`/`COUNT(DISTINCT ...)`,
    `GROUP BY`, `HAVING`, `ORDER BY` (inkl. `DESC`), `UNION`/`UNION ALL`, `CASE`,
    Outer Join (`LEFT JOIN`).
  - 6 Ankreuzaufgaben zur logischen Analyse (je 1 Punkt).
  - Anfrage in der Relationalen Algebra (4 Punkte).
  - ER-Entwurf (7 Punkte).
  - Schema-Übersetzung: ER $\rightarrow$ Relationales Modell (6 Punkte).
  - Relationale Normalformen/BCNF (3 Bonuspunkte).

Zu den Klausuraufgaben (2)

- Natürlich könnten sich die Klausuraufgaben leicht ändern.

  Z.B. wären die `CREATE TABLE` Syntax oder das CSV-Format auch Dinge, die in den Hausaufgaben dran waren. Ebenso wie die Frage, ob eine Anfrage Duplikate liefern könnte.

- Es ist aber davon auszugehen, dass es keine dramatischen Änderungen gibt, oder das ggf. vorher angesagt würde.

  Ca. 10–20% der Punkte könnten für neue Aufgabentypen vergeben werden.
  Wenn Sie eine sehr gute Note wollen, sollten Sie sich auf „alles“ vorbereiten.
  Trotz immer gleicher Aufgabentypen waren die Noten nicht toll.

- Beim ersten Termin sind 5 DIN A4 Blätter (10 Seiten) Notizen erlaubt (beim zweiten nur 3 Blätter/6 Seiten).

- Es gibt Beispiele für Klausuren auf der Webseite, eine Probeklausur ist geplant.

Inhalt

Wofür kann man Stilpunkte verlieren? (1)

Schlechte Namen von Tupelvariablen:

- Die Namen der Tupelvariablen sollten einen Bezug zur Bedeutung der Zeile haben, für die sie stehen.
- Dadurch wird die eventuell längliche Bedingung der Anfrage besser lesbar.

  Je länger und komplexer die Anfrage, desto wichtiger sind gute Namen.
- Z.B. schlecht: `x1`, `x2`, `x3`. Auch schlecht: `a`, `b`, `c`.

  Es sei denn, es handelt sich um die X-Werte von drei Punkten, oder z.B. die Einträge von Andreas, Bettina und Christina.
- Auf den Folien haben die Tupelvariablen häufig nur einen Buchstaben, aber man kann (und sollte öfters) längere Bezeichner wie in Programmiersprachen wählen.

  Die Folien haben eine große Schrift und sehr begrenzten Platz.

Wofür kann man Stilpunkte verlieren? (2)

Tupelvariablen mit gleichem Namen:

- In einer Anfrage sollten nicht zwei Tupelvariablen mit gleichem Namen verwendet werden.

  Eventuell mit Ausnahme von parallelen Unteranfragen, wenn einfach die Relationennamen als (implizite) Tupelvariablen verwendet werden.

- Fast immer gibt das einen echten Fehler, aber wenn es funktionieren sollte, ist es dennoch schlechter Stil.

Missverständliche Attributreferenzen ohne Tupelvariablen:

- In einer Unteranfrage sollte man auf Attribute äußerer Anfragen mit expliziter Tupelvariable zugreifen.

  Es kann nichts schaden, in jeder Attributreferenz eine explizite Tupelvariable zu verwenden (es sei denn, es gibt überhaupt nur eine Tupelvariable). Aber dieser Stil ist nicht Pflicht.

Wofür kann man Stilpunkte verlieren? (3)

LIKE statt =:

- Wenn die rechte Seite keine Wildcards „%“ und „_“ enthält, ist `LIKE` äquivalent zu „=“.

  Bis möglicherweise auf das Verhalten bei Leerzeichen am Ende der Zeichenkette: `LIKE` hat immer die `NOPAD SPACE` Semantik (d.h. Leerzeichen am Ende sind signifikant), während = häufig die `PAD SPACE` Semantik hat (abhängig von DBMS, `CHAR` vs. `VARCHAR` und gewählter Collation). Wenn Sie wirklich die `NOPAD SPACE` Semantik brauchen, kann ich den Einsatz von `LIKE` verstehen.

- `LIKE` suggeriert den Mustervergleich. Der Leser muss erst verstehen, dass es tatsächlich ein einfacher Gleichheitstest ist.

- Wenn rechts keine String-Konstante steht, kann der Optimierer keinen Index verwenden, da die rechte Seite ja ein Musterzeichen enthalten könnte.

  Bei Stringkonstanten ohne % und _ verwandeln viele Optimierer `LIKE` in =.

Wofür kann man Stilpunkte verlieren? (4)

Unnötiges `DISTINCT`:

- Wenn es auch ohne `DISTINCT` beweisbar keine Duplikate geben kann, gibt es für das überflüssige `DISTINCT` Punktabzug.

  Die Aussage bezieht sich natürlich auf alle möglichen DB-Zustände, die die Integritätsbedingungen (besonders Schlüssel) erfüllen.

- Der Optimierer kann das meistens nicht verstehen, führt also die Duplikatelimination durch, die doch nichts am Ergebnis ändert.
- Das kostet temporären Speicherplatz und Laufzeit.
- Die Fähigkeit, Duplikate vorhersagen zu können, zeigt auch eine gewisse Reife im Umgang mit SQL.

Wofür kann man Stilpunkte verlieren? (5)

Unnötiger Join:

- Man sollte nicht mehr Tupelvariablen als nötig verwenden.

  Das bezieht sich nur auf Basistabellen, nicht auf Sichten oder `WITH`-Hilfstabellen.

- Typisch ist ein Verbund von Fremdschlüssel zu Schlüssel, wobei man von der Tabelle mit dem Schlüssel nur diesen Schlüsselwert verwendet.

  Dann hätte man auch direkt den Fremdschlüssel verwenden können.

- Auch blöd sind zwei Tupelvariablen, von denen die Schlüsselwerte gleichgesetzt sind.

  Diese zeigen immer auf die gleiche Zeile.

- Der Optimierer wird das eher nicht merken, die Anfrage läuft länger. Außerdem ist sie schwieriger zu verstehen.

Wofür kann man Stilpunkte verlieren? (6)

Allgemein unnötige Teile / unnötige Komplikationen:

- Z.B. kann eine Disjunktion (`OR`),
  - bei der eine Hälfte inkonsistent ist,

  dennoch eine formal korrekte Anfrage ergeben,
  - wenn die andere Hälfte gerade die benötigte Bedingung ist.
- Dennoch würde es Punktabzug geben.

  Man zeigt damit ja, dass man die Logik nicht verstanden hat (oder den Korrekteur ärgern will).
- Wenn eine Anfrage sehr viel länger als nötig ist, muss man mit Punktabzug rechnen.

  „Länge“ wird dabei nicht einfach in Zeichen gemessen, dann würde man ja für gute Namen bei Tupelvariablen bestrafen, was sicher nicht beabsichtigt ist.

Wofür kann man Stilpunkte verlieren? (7)

Auffallend schlechte Formatierung:

- Eine lange Anfrage ganz in einer Zeile wäre sicher schlecht.

  Mit meinem Editor kann ich mir nur Zeilen bis 80 Zeichen gut anschauen.

- Rücken Sie so ein, dass die Struktur der Anfrage klar wird.

  Z.B. Fortsetzungszeilen einer Unteranfrage sollten nicht ganz links beginnen.
  Während `AND` der Hauptanfrage ganz links beginnen könnte, wären Bedingungen ganz links (wenn man `AND` am Ende der letzten Zeile geschrieben hat) schlecht.
  Auch bei einer `FROM`-Klausel, die sich über mehrere Zeilen erstreckt, sollten die Fortsetzungszeilen eingerückt werden, so dass man sofort visuell wahrnehmen kann, dass sie noch zur `FROM`-Klausel gehören.
  Dagegen sollten die Schlüsselworte `SELECT`, `FROM`, `WHERE` am Anfang der Zeile stehen, außer wenn die ganze Anfrage in eine Zeile passt.

- Was in Java schlecht wäre, ist auch in SQL schlecht.

Wofür kann man Stilpunkte verlieren? (8)

Sehr viele Unteranfragen / `WITH`-Hilfstabellen:

- Wenn die Anfrage durch Unteranfragen sehr aufgebläht wird, könnte das zum Punktabzug führen.

  Die `WITH`-Hilfstabellen sollten zumindest so benannt sein, dass sie jeweils für das Verständnis der Lösung förderlich sind.

Code-Duplizierung:

- Code-Duplizierung ist in jeder Programmiersprache schlecht.

  Mindestens, seit es `WITH` gibt, hat man auch in SQL keine Entschuldigung mehr.

- Allerdings sind SQL-Anfragen häufig kurz, und wenn nur ganz kleine Teile dupliziert sind (wie `ATYP = 'H'`) wäre die Variante mit `WITH` tatsächlich länger.

- Wägen Sie die Vor- und Nachteile sinnvoll ab.

Wofür kann man Stilpunkte verlieren? (9)

Portabilitäts-Probleme:

- Schreiben Sie `<>` für „verschieden von“, nicht `!=`.
- Offiziell schreibt sich der Kommentar `--`, nicht `/*...*/`.
- `ILIKE` ist nicht im Standard und nicht portabel.
- `GROUP BY`-Anfragen sollten unter `SELECT` außerhalb von Aggregationsfunktionen nur `GROUP BY` Attribute verwenden.

  Die neuen Regeln für funktional bestimmte Attribute sind kaum implementiert.
- Verlassen Sie sich nicht auf großzügige Typ-Umwandlungen, z.B. von Zeichenketten in Zahlen.
- Einige Systeme (z.B. PostgreSQL) verwenden Integer-Division.

  Wenn beide Argumente `INT` sind. Vielleicht sollten Sie statt `/100` besser `/100.0` schreiben. Versuchen Sie auch, die Division durch 0 zu vermeiden (Exception).

Wofür kann man Stilpunkte verlieren? (10)

Angaben, die völlig irrelevant sind:

- Komplizierte `SELECT`-Listen in `EXISTS`-Unteranfragen.

  Es ist egal, was man da hinschreibt, da nur auf die Existenz der Zeile getestet wird. Nutzen Sie `SELECT *` oder `SELECT 1` oder eventuell auch `SELECT` mit einem Attribut, das charakteristisch ist für die Objekte, die existieren sollen (z.B. `SID`, wenn nach einem Studenten gesucht wird). Ich persönlich würde das aber schon für grenzwertig halten. Der Leser fragt sich dann, warum hat er genau dieses Attribut angegeben? Mehrere Attribute in der `SELECT`-Liste einer `EXISTS`-Unteranfrage würden sicher zum Punktabzug führen.

- Benutzen Sie `COUNT(*)`, wenn es nicht Gründe für die `COUNT`-Varianten mit Argument gibt.

  Wenn das Attribut in `COUNT(`⟨Attribut⟩`)` nicht Null ist, und Sie keine Duplikate im `COUNT` eliminieren (mit `DISTINCT`), können Sie genauso gut `COUNT(*)` verwenden.

Wofür kann man Stilpunkte verlieren? (11)

Missbrauch von Konstrukten:

- Verwenden Sie `DISTINCT` zur Duplikat-Elimination, und nicht `GROUP BY`.

  `GROUP BY`-Anfragen sollten normalerweise Aggregationsfunktionen enthalten. Falls Sie nur bestimmte Duplikate eliminieren wollen, aber nicht alle, könnte man über `GROUP BY` nachdenken, aber wahrscheinlich müsste man dann doch `MIN` oder `MAX` benutzen, damit die Anfrage syntaktisch korrekt wird. `EXISTS`-Unteranfragen wären in diesem Fall klarer.

- Vermeiden Sie extrem trickreiche Anfragen, die man nur verstehen kann, wenn man SQL-Experte ist (sofern die Anfrage dadurch nicht wesentlich kürzer wird).

  Z.B. kann man mit „`(SELECT 1 FROM ... WHERE ...) IS NULL`“ testen, ob die Unteranfrage ein leeres Ergebnis liefert (Voraussetzung ist dabei, dass die Unteranfrage niemals mehr als eine Zeile liefern kann.).
  Für diese Aufgabe ist aber `NOT EXISTS` gedacht.

Wofür kann man Stilpunkte verlieren? (12)

Anfragen, die bei zusätzlichen Spalten falsch werden:

- Es kommt gelegentlich vor, dass Tabellen um zusätzliche Spalten erweitert werden müssen.

  Dann freut man sich, wenn Anfragen in Programmen unverändert weiter funktionieren (und man aufgrund des Stils nicht alles neu testen muss).

- `SELECT *` hätte plötzlich mehr Spalten.
  Programme würden eventuell nicht mehr funktionieren.

  Während man bei einer Tupelvariable noch die Hoffnung haben kann, dass die neuen Spalten am Ende sind, und nicht stören, könnten sich bei mehreren Tupelvariablen Spalten verschieben. Das Gleiche gilt auch bei `SELECT X.*, ...`.
  Bei Anfragen in Programmen ist es besser, die Spalten explizit aufzuzählen.

- Ein `NATURAL JOIN` funktioniert nur so lange, wie die zu vergleichenen Spalten die einzigen mit gleichem Namen sind.

  Verwenden Sie besser `USING(`⟨Spalte⟩`, ...)`

Wofür kann man Stilpunkte verlieren? (13)

Zusammenfassung:

- Denken Sie daran, dass die Tutoren jede Woche gut 100 SQL-Anfragen lesen müssen.

  Wir haben momentan gut 100 Einsendungen pro Aufgabenblatt.
  Das enthält normalerweise drei SQL-Anfragen. Wir haben drei Tutoren.
  Sie bekommen jeweils 25h pro Monat bezahlt (für vier Monate). Für eine SQL-Anfrage bleiben also ca. 3–4 min (dabei sind die Präsenzaufgaben noch nicht eingerechnet). Da freut man sich schon über größtmögliche Lesbarkeit.

- Früher habe ich gedroht, dass, wenn ich Ihre Anfrage in 5 min nicht verstehe, es 0 Punkte gibt.

  Tatsächlich arbeite ich bei der Klausur häufig länger an einer Anfrage, aber Spass macht mir das auch nur selten. (Ein neuer Typ von einem automatisch erkennbarem semantischen Fehler wäre natürlich interessant.)

- Dank an die Tutoren für eine Liste mit Stil-Problemen!

Inhalt

Syntax von Unteranfragen (1)

- `NOT EXISTS (SELECT * FROM ... WHERE ...)`

  Natürlich geht das auch ohne `NOT`, das ist aber selten.
  Statt `SELECT *` kann man eine beliebige `SELECT`-Klausel schreiben. Da es völlig egal ist, was man dort schreibt, wären komplizierte `SELECT`-Klauseln schlechter Stil. In der Logik gibt es beim Quantor auch nur die Variablendeklaration (`FROM`) und die quantifizierte Formel (`WHERE`).
  Man beachte, dass links vom `NOT EXISTS` nichts steht. Manche Studierenden verwechseln es mit `NOT IN`, wo man links einen Wertausdruck schreiben muss.

- ⟨Term⟩ `NOT IN (SELECT` ⟨Term⟩ `FROM ... WHERE ...)`

  Statt „Term“ kann man natürlich auch „Wertausdruck“ sagen.
  Auch hier kann man auch `IN` ohne `NOT` verwenden.

- ⟨Term⟩ `>= ALL (SELECT` ⟨Term⟩ `FROM ... WHERE ...)`

  Statt `ALL` sind auch `ANY` und `SOME` möglich (diese beiden Schlüsselworte sind Synonyme: mindestens eins). Statt >= auch andere Vergleichsoperatoren.

Syntax von Unteranfragen (2)

- ⟨Term⟩ >= (SELECT ⟨Term⟩ FROM ... WHERE ...)

  Man kann Unteranfragen mit einer Ergebnisspalte, und höchstens einer Ergebniszeile (für gegebene Variablenbelegung außen), auch wie einen Term verwenden. Auch als Teil eines größeren Terms. Auch unter SELECT.

- Sollte jemals mehr als eine Zeile geliefert werden, gibt es einen Laufzeitfehler. (Falls Ergebnis leer: Nullwert.)
  - Da SQL üblicherweise interpretiert wird (also direkt nach der Syntaxanalyse auch ausgeführt wird), scheint der Unterschied zu Syntaxfehlern zunächst nicht groß.
  - Aufgabe: Warum sind Laufzeitfehler dennoch besonders problematisch (im Vergleich mit Syntaxfehlern)?
- Unteranfragen können auch unter FROM eingesetzt werden (anstelle von Tabellen). Siehe Kapitel 11 über Sichten, WITH.

Zugriff auf Tupelvariablen/Attribut-Referenzen

- Unteranfragen können Variablen der äußeren Anfrage nutzen.
- Beispiel/Aufgabe: Was ist korrekt, was falsch?

```
SELECT SID[1], B.PUNKTE[2]
FROM   STUDENTEN S, AUFGABEN A
WHERE  A.THEMA = 'SQL' AND A.ATYP = 'H'
AND    EXISTS(SELECT NACHNAME[3]
              FROM   BEWERTUNGEN B
              WHERE  SID[4] = S.SID[5]
              AND    A.ANR = B.ANR
              AND    B.ATYP ='H'
              AND    B.PUNKTE = MAXPT)
```

Abstimmung für jede gekennzeichnete Attributreferenz: Ja: Korrekt. Nein: falsch.

STUDENTEN(SID,VORNAME,NACHNAME,EMAIL°) AUFGABEN(ATYP,ANR,THEMA,MAXPT)
BEWERTUNGEN(SID→STUDENTEN,(ATYP,ANR)→AUFGABEN,PUNKTE)

Korrelierte und unkorrelierte Unteranfragen

- Unteranfragen, die Variablen der äußeren Anfrage verwenden, nennt man „korrelierte Unteranfragen“.

  Korrelierte Unteranfragen kann man sich als parametrisiert mit Tupeln der äußeren Anfrage vorstellen. Konzeptionell werden diese Unteranfragen einmal für jede Belegung der Tupelvariablen der äußeren Anfrage ausgeführt.

- Unteranfragen, die nicht auf Variablen der äußeren Anfrage zugreifen, nennt man „unkorrelierte Unteranfragen“.

  Es genügt, eine unkorrelierte Unteranfrage nur einmal auszuführen (da das Ergebnis nicht von Tupelvariablen der äußeren Anfrage abhängt).

- Unkorrelierte `EXISTS`-Unteranfragen sind fast immer falsch!

  Es entspricht einer vergessenen Verbund-Bedingung. Die Unteranfrage ist wahr oder falsch unabhängig von den Zeilen, die gerade außen betrachtet werden. Unkorrelierte `IN`-Unteranfragen sind dagegen ok (der Verbund geschieht hier durch das `IN`).

Inhalt

Präsenzaufgabe: Unverheiratete Präsidenten (1)

- Schema president_public im Adminer:
  - president(pres_name, birth_year, years_serv, death_age, party, state_born→state)
  - administration(admin_nr, pres_name→president, year_inaugurated)
  - pres_marriage(pres_name→president, spouse_name, pr_age, sp_age, nr_children, mar_year)
- Aufgabe: Schreiben Sie eine Anfrage, die Präsidenten findet, die niemals geheiratet haben.
- Geben Sie den Namen und das Jahr der Amtseinführung aus.

pres_name	year_inaugurated
Buchanan J	1857

Präsenzaufgabe: Unverheiratete Präsidenten (2)

- Die Anfrage verhält sich nichtmonoton:
  - Wenn man eine Ehe für „`Buchanan J`“ in die Tabelle `pres_marriage` einfügen würde, würde er nicht mehr herauskommen.
- Daher ist klar, dass man ein nichtmonotones Konstrukt benötigt, z.B. `NOT EXISTS`.
- Mögliche Lösung:

```
SELECT a.pres_name, a.year_inaugurated
FROM   administration a
WHERE  NOT EXISTS
            (SELECT * FROM pres_marriage m
             WHERE  m.pres_name = a.pres_name)
```

Präsenzaufgabe: Unverheiratete Präsidenten (3)

- Diese Anfrage enthält einen unnötigen Join:

```
SELECT p.pres_name, a.year_inaugurated
FROM   president p, administration a
WHERE  p.pres_name = a.pres_name
AND    NOT EXISTS
           (SELECT * FROM pres_marriage m
            WHERE  m.pres_name = p.pres_name)
```

- Man braucht die Tabelle `president` gar nicht:
  - Es wird überhaupt nur das Schlüssel-Attribut `pres_name` verwendet.
  - Dies ist gleich dem Fremdschlüssel `pres_name` in der Tabelle `administration`.

    Durch den Join werden keine `administration`-Tupel entfernt.

Präsenzaufgabe: Unverheiratete Präsidenten (4)

- Ich habe Sie nicht absichtlich auf's Glatteis geführt durch Nennung der Tabelle `president`.

  Mir war bei der Aufgabenstellung auch nicht aufgefallen, dass die Tabelle eigentlich überflüssig ist. In der Klausur wird nur einmal das ganze Schema angegeben, da müssen Sie für jede Anfrage die notwendigen Tabellen selbst herausfinden. Andererseits wird nicht durch Angabe einer Teilmenge suggeriert, dass man alle diese Tabellen brauchen würde.

- Inzwischen haben wir auch `IN` besprochen (letzten Dienstag war das aber noch nicht bekannt):

```
SELECT pres_name, year_inaugurated
FROM   administration
WHERE  pres_name NOT IN (SELECT pres_name
                         FROM   pres_marriage)
```

Präsenzaufgabe: Unverheiratete Präsidenten (5)

- Auch möglich:

```
SELECT pres_name, year_inaugurated
FROM   administration
WHERE  pres_name <> ALL (SELECT pres_name
                         FROM   pres_marriage)
```

- Unnötiger doppelter Vergleich:

```
SELECT a.pres_name, a.year_inaugurated
FROM   administration a
WHERE  a.pres_name NOT IN
           (SELECT m.pres_name
            FROM   pres_marriage m
            WHERE  a.pres_name = m.pres_name)
```

Korrelierte `IN`-Unteranfragen sind auch nicht schön.

Präsenzaufgabe: Unverheiratete Präsidenten (6)

- Was halten Sie von dieser Lösung?

```
SELECT DISTINCT a.pres_name, a.year_inaugurated
FROM   administration a, pres_marriage m
WHERE  a.pres_name NOT IN (SELECT m.pres_name
                           FROM   pres_marriage m)
```

- Diese Lösung kam so ähnlich tatsächlich vor.

  Es war zusätzlich noch der überflüssige Join mit `president p` enthalten.

- Die folgende Lösung kam nicht vor, aber was ist damit?

```
SELECT DISTINCT a.pres_name, a.year_inaugurated
FROM   administration a, pres_marriage m
WHERE  a.pres_name = m.pres_name
AND    a.pres_name NOT IN (SELECT m.pres_name
                           FROM   pres_marriage m)
```

Präsenzaufgabe: Unverheiratete Präsidenten (7)

- Braucht man eine Duplikateliminierung?

```
SELECT DISTINCT pres_name, year_inaugurated
FROM   administration
WHERE  ...
```

- Das ist schwierig (Forts. siehe nächste Folie):
  - Nein, denn es kann nicht sein, dass der gleiche President im gleichen Jahr zwei Mal in sein Amt eingeführt wird.

    Mindestens ein Student hält es in ganz extremen Sondersituationen für möglich, dass der Präsident mit seinen Ministern zurücktritt und dann nochmals mit neuen Ministern antritt und noch im gleichen Jahr erneut vereidigt wird. Ich kann mir persönlich das nicht vorstellen. Vielleicht wollte man in so einer ganz extremen Situation auch das Duplikat.

Präsenzaufgabe: Unverheiratete Präsidenten (8)

- Braucht man `DISTINCT`? Fortsetzung:
  - Ja, weil die Kombination von `pres_name` und `year_inaugurated` nicht als Schlüssel deklariert ist.

    Es wäre vielleicht ein sinnvoller Schlüssel, aber das ist eine Frage des DB-Entwurfs. Bei der Formulierung von Anfragen muss man sich auf das gegebene Schema beziehen.
- Daher gibt es weder einen Punktabzug für unnötiges `DISTINCT` noch für fehlendes `DISTINCT`.

  Ich hätte die Mehrdeutigkeit vielleicht besser vermieden, indem ich verlangt hätte, die Administrations-Nummer mit auszugeben. Andererseits lernt man wahrscheinlich mehr aus Beispielen, die nicht einfach ganz glatt ablaufen.

Präsenzaufgabe: Unverheiratete Präsidenten (9)

- Die folgende Anfrage kann nicht korrekt sein, weil sie sich monoton verhält:

```
SELECT DISTINCT a.pres_name, a.year_inaugurated
FROM   administration a, pres_marriage m
WHERE  a.pres_name <> m.pres_name
```

- Solange es wenigstens zwei verheiratete Präsidenten gibt, werden alle Präsidenten ausgegeben.

  Es gibt dann ja immer in `pres_marriage` einen Eintrag für einen anderen Präsidenten als den, auf den die Tupelvariable a gerade zeigt.

Inhalt

EMP-DEPT-Datenbank (1)

- Klassische Beispiel-Datenbank von Oracle.
- Schema „empdept_public“ im Adminer:
  - dept(deptno, dname, loc)
  - emp(empno, ename, job, mgr°→emp, hiredate, sal, comm°, deptno°→dept)
- dept: „Department“ (Abteilungen einer Firma)
- emp: „Employee“ (Angestellte der Firma)
- Der Link zum Adminer ist:

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

EMP-DEPT-Datenbank (2)

- Die Tabelle dept hat die Spalten
  - deptno: Abteilungsnummer (Department Number),
  - dname: Name der Abteilung (Department Name),
  - loc: Ort/Sitz der Abteilung (Location):

DEPT		
DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

EMP-DEPT-Datenbank (3)

- Die Tabelle `emp` hat folgende Spalten:
  - `empno`: Angestellten-Nummer (identifiziert den Angestellten)
  - `ename`: Name des Angestellten.
  - `job`: Berufsbezeichnung des Angestellten.
  - `mgr`: Angestellten-Nummer des direkten Vorgesetzten.

    `mgr` von „manager“.
  - `hiredate`: Datum der Einstellung.
  - `sal`: Gehalt des Angestellten („salary“).
  - `comm`: Provision (nur für Verkäufer) („commission“).
  - `deptno`: Abteilung des Angestellten.

EMP-DEPT-Datenbank (4)

EMP					
EMPNO	ENAME	JOB	MGR	SAL	DEPTNO
7369	SMITH	CLERK	7902	800	NULL
7499	ALLEN	SALESMAN	7698	1600	30
7521	WARD	SALESMAN	7698	1250	30
7566	JONES	MANAGER	7839	2975	20
7654	MARTIN	SALESMAN	7698	1250	30
7698	BLAKE	MANAGER	7839	2850	30
7782	CLARK	MANAGER	7839	2450	10
7788	SCOTT	ANALYST	7566	3000	20
7839	KING	PRESIDENT		5000	10
7844	TURNER	SALESMAN	7698	1500	30
7876	ADAMS	CLERK	7788	1100	20
7900	JAMES	CLERK	7698	950	30
7902	FORD	ANALYST	7566	3000	20
7934	MILLER	CLERK	7782	1300	10

Hausaufgabe 8.1: NOT EXISTS (1)

- In welchen Abteilungen arbeiten weder Analysten (Job „ANALYST“), noch Verkäufer (Job „SALESMAN“)?
- Geben Sie `deptno` (Abteilungsnummer), `dname` (Abteilungsname) und `loc` (Ort der Abteilung) aus.
- Sortieren Sie die Ausgabe nach der Abteilungsnummer.
- Die erwartete Antwort ist:

deptno	dname	loc
10	ACCOUNTING	NEW YORK
40	OPERATIONS	BOSTON

- Die Abteilung 40 ist tatsächlich leer, erfüllt aber die Bedingung der Anfrage.

Hausaufgabe 8.1: NOT EXISTS (2)

- Lösung mit `NOT EXISTS`:

```
SELECT deptno, dname, loc
FROM   dept d
WHERE  NOT EXISTS (SELECT *
                   FROM   emp e
                   WHERE  (e.job = 'ANALYST'
                    OR e.job = 'SALESMAN')
                   AND    e.deptno = d.deptno)
ORDER  BY deptno
```

Man könnte auch „`SELECT *`“ schreiben. Für Anfragen in Programmen ist das aber eher nicht empfohlen. Zumindest sollte man darüber nachdenken, was passieren würde, wenn die Tabelle später um zusätzliche Spalten erweitert wird. Für eine ad-hoc Anfrage wäre es natürlich ok.

Hausaufgabe 8.1: NOT EXISTS (3)

- Lösung mit `NOT IN`:

```
SELECT deptno, dname, loc
FROM   dept
WHERE  deptno NOT IN (SELECT deptno
                      FROM   emp
                      WHERE  job IN ('ANALYST',
                                     'SALESMAN')
                      AND    deptno IS NOT NULL)
ORDER  BY deptno
```

Bei `NOT IN` ist es wichtig, dass die Unteranfrage keinen Nullwert liefern kann, sonst wäre das Ergebnis leer. Im Beispielzustand ist der einzige Angestellte, der bisher keiner Abteilung zugeordnet ist, ein `CLERK`. Man würde es beim Test also nicht bemerken.

Hausaufgabe 8.2: Keine Untergebenen (1)

- In der Spalte `mgr` steht die Angestellten-Nummer des direkten Vorgesetzten des jeweiligen Angestellten.
- Geben Sie bitte alle Angestellten aus (`empno`, `ename`, `job`, `sal`) die keine Untergebenen haben,
  - deren Angestelltennummer `empno` also nicht in der Spalte `mgr` vorkommt.
- Sortieren Sie das Ergebnis absteigend nach dem Gehalt `sal` (also größtes Gehalt zuerst).
- Das erwartete Ergebnis im Beispiel-Zustand steht auf der nächsten Folie.

Hausaufgabe 8.2: Keine Untergebenen (2)

- Das erwartete Ergebnis im Beispiel-Zustand ist:

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

Hausaufgabe 8.2: Keine Untergebenen (3)

- Mögliche Lösung:

```
SELECT e.empno, e.ename, e.job, e.sal
FROM   emp e
WHERE  NOT EXISTS(SELECT * FROM emp u
                  WHERE  u.mgr = e.empno)
ORDER  BY sal DESC
```

Hausaufgabe 8.3: Niedrigstes Gehalt (1)

- Welche Angestelle haben das niedrigste Gehalt?
- Geben Sie Angestellten-Nummer (`empno`), Namen (`ename`) und Gehalt (`sal`) des oder der Angestellten aus.
- Verwenden Sie keine Aggregationsfunktionen wie `MIN`, sondern nur Konstrukte, die in der Vorlesung schon eingeführt wurden. Das schließt auch den Outer Join aus.
- Das erwartete Ergebnis im Beispiel-Zustand ist:

empno	ename	sal
7369	SMITH	800

Hausaufgabe 8.3: Niedrigstes Gehalt (2)

- Mögliche Lösung:

```
SELECT empno, ename, sal
FROM   emp e
WHERE  NOT EXISTS (SELECT * FROM emp x
                   WHERE  x.sal < e.sal)
```

Hausaufgabe 8.4: Zweithöchstes Gehalt (1)

- Welche Angestellten haben das zweithöchste Gehalt, d.h.
  - es gibt mindestens einen Angestellten, der mehr verdient,
  - aber alle Angestellten, die mehr verdienen, haben das gleiche Gehalt.
- Man kann auch prüfen, dass niemand echt zwischen dem Top-Verdiener und den gesuchten Angestellten liegt.
- Beachten Sie, dass nicht ausgeschlossen ist, dass es mehrere Angestellte mit dem gleichen maximalen Gehalt geben kann.
- Geben Sie Angestellten-Nummer (`empno`), Namen (`ename`), Beruf (`job`) und Gehalt (`sal`) dieser Angestellten aus.

Hausaufgabe 8.4: Zweithöchstes Gehalt (2)

- Sortieren Sie die Ausgabe nach den Namen der Angestellten.
- Das erwartete Ergebnis im Beispiel-Zustand ist:

empno	ename	job	sal
7902	FORD	ANALYST	3000
7788	SCOTT	ANALYST	3000

Hausaufgabe 8.4: Zweithöchstes Gehalt (3)

- Eine mögliche Lösung ist:

```
SELECT   e.empno, e.ename, e.job, e.sal
FROM     emp e, emp top
WHERE    e.sal < top.sal
AND      NOT EXISTS(SELECT *
                    FROM   emp x
                    WHERE  x.sal > e.sal
                    AND    x.sal <> top.sal)
ORDER BY ename
```

Hausaufgabe 8.5: „Für alle“ (1)

- Welche Jobs gibt es in allen Abteilungen, die nicht leer sind, also mindestens einen Angestellten haben?

  Gesucht ist also die Schnittmenge der Berufe über allen nicht-leeren Abteilungen.

  Z.B. hat jede Abteilung einen Manager: `CLARK` in Abteilung 10, `JONES` in Abteilung 20 und `BLAKE` in Abteilung 30.

- Die erwartete Antwort ist:

job
CLERK
MANAGER

  2 Datensätze

- Üblicherweise überführt man eine Allaussage in SQL in $\neg\exists\neg$, hier also: „es gibt keine nicht-leere Abteilung, die nicht mindestens einen Angestellten mit dem Job hat“.

Hausaufgabe 8.5: „Für alle“ (2)

- Lösung:

```
SELECT DISTINCT job
FROM   emp j
WHERE  NOT EXISTS
       (SELECT * FROM dept d
        WHERE  EXISTS
               (SELECT * FROM emp x
                WHERE  d.deptno = x.deptno)
        AND    NOT EXISTS
               (SELECT * FROM emp e
                WHERE  e.job = j.job
                AND    e.deptno = d.deptno))
```

Hausaufgabe 8.5: „Für alle“ (3)

- Alternative (kürzer, Abteilung wird durch `emp d` vertreten, damit ist die Abteilung automatisch nicht leer):

```
SELECT DISTINCT job
FROM   emp j
WHERE  NOT EXISTS
       (SELECT * FROM emp d
        WHERE  d.deptno IS NOT NULL
        AND    NOT EXISTS
               (SELECT * FROM emp e
                WHERE  e.job = j.job
                AND    e.deptno = d.deptno))
```

Während bei der letzten Lösung leere Abteilungen ausgeschlossen werden müssen, ist hier die „Null“-Abteilung von Herrn `SMITH` ein Problem (ohne den Test auf `IS NOT NULL` wäre das Gesamtergebnis leer).

Hausaufgabe 8.5: „Für alle“ (4)

- Auch möglich:

```
WITH   JOBS AS
       (SELECT DISTINCT job
        FROM   emp)
SELECT job
FROM   jobs j
WHERE  -- Anzahl Abteilungen mit Job j:
       (SELECT COUNT(DISTINCT deptno)
        FROM   emp
        WHERE  emp.job = j.job)
       =
       -- Anzahl aller Abteilungen (nicht-leer):
       (SELECT COUNT(DISTINCT deptno)
        FROM   emp)
```

Inhalt

Präsenzaufgabe: Nichtmonotone Anfrage

- Tabellen des Schemas „empdept_public“ im Adminer:
  - dept(deptno, dname, loc)
  - emp(empno, ename, job, mgr°→emp, hiredate, sal, comm°, deptno°→dept)
- Formulieren Sie die folgende Anfrage in SQL:
  - Gibt es einen Angestellten, der direkter Vorgesetzter (mgr) von allen Angestellten mit Job „SALESMAN“ ist (d.h. haben alle denselben Vorgesetzten)? Drucken Sie ggf. empno, ename, job:

7698	BLAKE	MANAGER

Falls es keinen Angestellten mit dieser Bedingung gibt, soll das Ergebnis der Anfrage leer sein.

- Verwenden Sie nur in der Vorlesung eingeführte Konstrukte.