Einführung in Datenbanken

Übung 9: Aggregationsfunktionen, WITH (lokale Sichten)

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 7a (1)

- Wer hat eine Wahl verloren und später noch eine Wahl gewonnen?

candidate
Adams J

Die Daten sind sehr unvollständig, aber wir arbeiten daran.

- `election(election_year, candidate, votes, winner_loser_indic)`

  Verloren: Wert `'L'` für `winner_loser_indic` in `election`, gewonnen: `'W'`.

- Beispiel für einen Selbstverbund: Man braucht zwei Zeilen der Tabelle `election` für den gleichen Kandidaten:
  - Eine frühere, die verloren wurde.
  - Eine spätere, die gewonnen wurde.

Hausaufgabe 7a (2)

- Lösung:

```
SELECT DISTINCT frueher.candidate
FROM   election frueher, election spaeter
WHERE  frueher.candidate = spaeter.candidate
AND    frueher.year < spaeter.year
AND    frueher.winner_loser_indic = 'L'
AND    spaeter.winner_loser_indic = 'W'
```

  Natürlich kann man genausogut `spaeter.candidate` ausgeben.

- `DISTINCT` ist nötig, weil der Kandidat auch mehrfach verloren haben kann, bevor er gewonnen hat, oder auch mehrfach gewonnen haben kann.

  D.h. es könnte mehrere Belegungen für die beiden Tupelvariablen geben, die den gleichen Kandidaten ausgeben.

Hausaufgabe 7a (3)

Kommentar:

- Die Erläuterung zur Notwendigkeit von `DISTINCT` sollte als syntaktisch korrekter Kommentar geschrieben werden.
- Ein Kommentar in SQL beginnt mit „`--`“ und erstreckt sich bis zum Ende der Zeile.
- Der Kommentar `/* ... */` funktioniert in vielen Systemen, ist aber nicht standard-konform.
- Java-Kommentare `// ...` sind in PostgreSQL Syntaxfehler!
- Probieren Sie Ihre Anfragen so aus, wie Sie sie abgeben!
- Wir arbeiten an einer halbautomatischen Korrektur, da machen Syntaxfehler zusätzliche Arbeit.

Hausaufgabe 7b (1)

- Welche Präsidenten haben die Hobbies „Riding“ und „Golf“?

  Gesucht sind nur Präsidenten die beide Hobbies haben, nicht etwa Präsidenten, die das eine oder das andere Hobby haben. Geben Sie das Geburtsjahr des Präsidenten mit aus und sortieren Sie die Ausgabe nach diesem Geburtsjahr.

- Die erwartete Antwort ist:

pres_name	birth_year
Wilson W	1856
Taft W H	1857
Harding W G	1865

Hausaufgabe 7b (2)

- pres_hobby(pres_name→president, hobby)
- president(pres_name, birth_year, years_serv, death_age, party, state_born→state)
- Lösung:

```
SELECT riding.pres_name, president.birth_year
FROM   pres_hobby riding, pres_hobby golf,
       president
WHERE  riding.pres_name = golf.pres_name
AND    riding.hobby = 'Riding'
AND    golf.hobby = 'Golf'
AND    president.pres_name = riding.pres_name
ORDER  BY president.birth_year
```

Hausaufgabe 7b (3)

- `DISTINCT` ist nicht nötig.
- Die Ausgabe enthält `riding.pres_name`.
- Durch die Join-Bedingungen sind damit auch `golf.pres_name` und `president.pres_name` eindeutig bestimmt für eine Ausgabezeile.
- `riding.hobby` und `golf.hobby` sind durch Konstanten in der `WHERE`-Bedingung eindeutig bestimmt.
- Damit hat man von jeder Tupelvariable einen Schlüssel:
  - `president`: `pres_name`
  - `riding`: `pres_name`, `hobby`
  - `golf`: `pres_name`, `hobby`

Hausaufgabe 7c (1)

- Welche Präsidenten (bzw. Präsidentschaftskandidaten) haben keine Wahl verloren, bevor Sie eine Wahl gewonnen haben?

  Geben Sie das Jahr der gewonnenen Wahl mit aus und sortieren Sie die Ausgabe danach.

- Die erwartete Antwort ist:

candidate	election_year
Washington G	1789
Washington G	1792
⋮	⋮
Clinton W J	1992

- Dies ist ein Beispiel für eine `NOT EXISTS` Unteranfrage (nichtmonotones Verhalten).

Hausaufgabe 7c (2)

- election(election_year, candidate, votes, winner_loser_indic)
- Lösung:

```
SELECT won.candidate, won.election_year
FROM   election won
WHERE  won.winner_loser_indic = 'W'
AND    NOT EXISTS
       (SELECT *
        FROM   election lost
        WHERE  lost.winner_loser_indic = 'L'
        AND    lost.election_year <
                        won.election_year
        AND    lost.candidate = won.candidate)
ORDER  by won.election_year
```

Hausaufgabe 7c (3)

- `DISTINCT` ist nicht nötig.
- Die äußere Anfrage enthält nur die Tupelvariable `won`.
- Davon wird ein Schlüssel (`election_year`, `candidate`) unter `SELECT` ausgegeben.
- Also kann jede Ausgabezeile nur von einer einzigen Variablenbelegung erzeugt werden.
- Die `NOT EXISTS`-Bedingung ist wahr oder falsch, erzeugt aber selbst keine Duplikate.

  Sie kann nicht mehrfach wahr bzw. mehrfach falsch sein. Duplikate werden nur durch mehrere Variablenbelegungen der äußeren Anfrage erzeugt, die die `WHERE`-Bedingung erfüllen und die gleichen Werte für die `SELECT`-Terme haben.

Inhalt

Präsenzaufgabe: Nichtmonotone Anfragen

- Tabellen des Schemas „empdept_public“ im Adminer:
  - dept(deptno, dname, loc)
  - emp(empno, ename, job, mgr°→emp, hiredate, sal, comm°, deptno°→dept)
- Formulieren Sie folgende Anfragen in SQL (je zwei Punkte):
  a) Wer (empno, ename) von allen Angestellten mit Job „CLERK“ verdient am meisten (sal)?

7934	MILLER	1300

  b) 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

- Bitte keine Aggregationsfunktionen (count, max) verwenden!

Lösung der Präsenzaufgabe 8a (1)

- Wer (empno, ename) von allen Angestellten mit Job „CLERK“ verdient am meisten (sal)?

  Dies ist natürlich eine nichtmonotone Anfrage. Wenn ich einen neuen Angestellten mit noch höherem Gehalt einfüge, kommt der bisherige Spitzenreiter nicht mehr heraus.

- Lösung 1:

```
SELECT empno, ename, sal
FROM   emp e
WHERE  job = 'CLERK'
AND    NOT EXISTS (SELECT *
                   FROM   emp x
                   WHERE  x.job = 'CLERK'
                   AND    x.sal > e.sal)
```

Lösung der Präsenzaufgabe 8a (2)

- Lösung 2:

```
WITH   clerk AS
       (SELECT empno, ename, sal
        FROM   emp
        WHERE  job = 'CLERK')

SELECT empno, ename, sal
FROM   clerk
WHERE  sal >= ALL (SELECT x.sal
                   FROM   clerk x)
```

- So hätte man den Job nur an einer Stelle spezifiziert.

  Der Name der Hilfstabelle (Sicht) spielt ja keine Rolle. Wenn man wirklich plant, die Bedingung dort zu ändern, sollte man es vielleicht „`emp_subset`“ nennen. DIe Teilmengen können auch anders als über den Job definiert sein.

Lösung der Präsenzaufgabe 8b (1)

- Gibt es einen Angestellten, der direkter Vorgesetzter (`mgr`) von allen Angestellten mit Job „`SALESMAN`“ ist (d.h. haben alle denselben Vorgesetzten)?

  Auch nichtmonoton: Wenn ich einen Salesman einfüge, dessen Vorgesetzter nicht Blake ist, kommt Blake nicht mehr heraus.

- Lösung 1:

```
SELECT empno, ename
FROM   emp e
WHERE  NOT EXISTS (SELECT *
                   FROM   emp x
                   WHERE  x.job = 'SALESMAN'
                   AND    x.mgr <> e.empno)
```

Lösung der Präsenzaufgabe 8b (2)

- Lösung 2:

```
SELECT empno, ename
FROM   emp
WHERE  empno = ALL (SELECT mgr
                    FROM   emp
                    WHERE  job = 'SALESMAN')
```

Hier gibt es nun zwei Tupelvariablen, die beide `emp` heißen. Wenn man das stilistisch schlecht findet, kann man eine oder beide umbenennen.
Da die Unteranfrage nicht korreliert ist, kann man sie aber getrennt entwickeln und dann nur in die äußere Anfrage „hineinstecken“.
Insofern macht es durchaus Sinn, wie es ist.

- `ALL`-Bedingungen sind erfüllt, wenn die Unteranfrage ein leeres Ergebnis liefert.

Inhalt

Aggregationsfunktionen (1)

Die fünf klassischen Aggregationsfunktionen von SQL:

- `COUNT` zur Anzahlbestimmung. Dabei gibt es drei Fälle:
  - `COUNT(*)`: Einfache Anzahl (Anzahl Zeilen).
  - `COUNT(`⟨Term⟩`)`: Nur zählen, wenn ⟨Term⟩ nicht Null.
  - `COUNT(DISTINCT` ⟨Term⟩`)`: Nur verschiedene Werte für ⟨Term⟩ zählen (ohne Null).
- `MIN(`⟨Term⟩`)`: Kleinster Wert von ⟨Term⟩.
- `MAX(`⟨Term⟩`)`: Größter Wert von ⟨Term⟩.
- `SUM(`⟨Term⟩`)`: Summe aller Werte von ⟨Term⟩.
- `AVG(`⟨Term⟩`)`: Durchschnitt aller Werte von ⟨Term⟩.

  Das ist das arithmetische Mittel: Summe aller Werte geteilt durch Anzahl.

Aggregationsfunktionen (2)

- Beispiel-Tabelle:

TEST	
ID	WERT
1	10
2	10
3	40
4	NULL

- Was liefert „`SELECT COUNT(WERT) FROM TEST`“?
  - A. 2
  - B. 3
  - C. 4
  - D. 60
  - E. Null

Aggregationsfunktionen (3)

- Im Adminer gibt es die Tabelle `TEST` leider nicht, aber man kann es so ausprobieren:

```
WITH   TEST(ID, WERT) AS
       (VALUES (1,10),
               (2,10),
               (3,40),
               (4,NULL))

SELECT COUNT(WERT)
FROM   TEST
```

Sehr portabel ist das nicht, aber es war schon in SQL-92, und funktioniert in PostgreSQL. Die `VALUES`-Klausel („table value constructor“) ist wie der `SELECT`-Ausdruck („query specification“) und `TABLE` ⟨Tabelle⟩ („explicit table“) eine „simple table“. Die `WITH`-clause kam erst in SQL-99 („query expression“).

Aggregationsfunktionen (4)

- Beispiel-Tabelle:

TEST	
ID	WERT
1	10
2	10
3	40
4	NULL

- Was liefert „`SELECT SUM(WERT) FROM TEST`“?
  - A. 3
  - B. 4
  - C. 50
  - D. 60
  - E. Null

Aggregationsfunktionen (5)

- Beispiel-Tabelle:

TEST	
ID	WERT
1	10
2	10
3	40
4	NULL

- Was liefert folgende Anfrage?

```
SELECT SUM(WERT) FROM TEST WHERE ID > 5
```

A. 0
B. Null
C. Eine Fehlermeldung

Aggregationsfunktionen (6)

- Beispiel-Tabelle:

TEST	
ID	WERT
1	10
2	10
3	40
4	NULL

- Was liefert „`SELECT AVG(WERT) FROM TEST`“?
  - A. 15
  - B. 20
  - C. 25
  - D. Null
  - E. Eine Fehlermeldung

Aggregationsfunktionen (7)

- Beispiel-Tabelle:

TEST	
ID	WERT
1	10
2	10
3	40
4	NULL

- Was liefert „`SELECT COUNT(DISTINCT WERT) FROM TEST`“?
  - A. 2
  - B. 3
  - C. 4
  - D. Null
  - E. Eine Fehlermeldung

Aggregationsfunktionen (8)

- Beispiel-Tabelle:

TEST	
ID	WERT
1	10
2	10
3	40
4	NULL

- Gibt es einen Unterschied zwischen den folgenden Anfragen?
  - `SELECT COUNT(*) FROM TEST`
  - `SELECT COUNT(ID) FROM TEST`

    `ID` ist Primärschlüssel und `NOT NULL`.
- Ja/Nein-Umfrage.

Aggregationsfunktionen (9)

- Beispiel-Tabelle:

TEST	
ID	WERT
1	10
2	10
3	40
4	NULL

- Was liefert „`SELECT COUNT(1) * 2 FROM TEST`“?
  - A. 1
  - B. 2
  - C. 4
  - D. 8
  - E. Das ist ein Syntaxfehler.

Aggregationsfunktionen (10)

- Beispiel-Tabelle:

TEST	
ID	WERT
1	10
2	10
3	40
4	NULL

- Was liefert folgende Anfrage?

```
SELECT WERT, COUNT(*) FROM TEST WHERE WERT < 20
```

A. (10, 2)

B. Das ist ein Syntaxfehler.

Aggregationsfunktionen (11)

- Diese Anfrage ist ein Beispiel für `GROUP BY`:

```
SELECT   WERT, COUNT(*)
FROM     TEST
WHERE    WERT < 20
GROUP BY WERT
```

- Möglich ist aber auch eine Unteranfrage unter `SELECT`:

```
SELECT DISTINCT WERT, (SELECT COUNT(*)
                       FROM   TEST innen
                       WHERE  innen.WERT =
                              aussen.WERT)
FROM   TEST aussen
WHERE  WERT < 20
```

Aggregationsfunktionen (12)

- Um die mehrfache Berechnung der Unteranfrage für gleiche Werte zu vermeiden, könnte man es auch so formulieren:

```
WITH   werte AS
       (SELECT DISTINCT WERT
        FROM   TEST
        WHERE  WERT < 20)

SELECT WERT, (SELECT COUNT(*)
              FROM   TEST
              WHERE  test.WERT = werte.WERT
FROM   werte
```

Vielleicht erkennt der Optimierer aber auch schon bei der Formulierung von der letzten Folie, dass er die Duplikateliminierung vorziehen kann.

Aggregationsfunktionen (13)

- Beispiel-Tabelle:

TEST	
ID	WERT
1	10
2	10
3	40
4	NULL

- Was liefert folgende Anfrage?

```
SELECT ID FROM TEST WHERE WERT = MAX(WERT)
```

A. 3
B. Die leere Menge.
C. Das ist ein Syntaxfehler.

Aggregationsfunktionen (14)

- Lösung mit Unteranfrage:

```
SELECT ID
FROM   TEST
WHERE  WERT = (SELECT MAX(WERT)
               FROM   TEST)
```

- Diese Anfrage liefert 3.

Inhalt

Aufgaben (1)

- Diese Aufgaben sind hier und jetzt einzeln zu bearbeiten, aber nicht abzugeben.
- Tabellen des Schemas „empdept_public“ im Adminer:
  - dept(deptno, dname, loc)
  - emp(empno, ename, job, mgr°→emp, hiredate, sal, comm°, deptno°→dept)
- Was ist die Summe der Gehaelter der Abteilung „RESEARCH“?

  Posten Sie Ihre Lösung im Chat, aber geben Sie den anderen wenigstens drei Minuten. Sie können natürlich jederzeit Fragen stellen.
- Erwartete Antwort:

sum
10075

Aufgaben (2)

- Tabellen des Schemas „empdept_public“ im Adminer:
  - dept(deptno, dname, loc)
  - emp(empno, ename, job, mgr°→emp, hiredate, sal, comm°, deptno°→dept)
- Wer verdient mehr als das durchschnittliche Gehalt in der Abteilung „RESEARCH“?

  Der Durchschnitt aus auch nur auf diese Abteilung bezogen.
- Verwenden Sie eine WITH-Hilfstabelle für alle Angestellten der RESEARCH-Abteilung.
- Sie benötigen eine Unteranfrage für die Berechnung des durchschnittlichen Gehalts.

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

Präsenzaufgabe: Fehlermeldungen

- Verwenden Sie das Schema „empdept_public“ im Adminer:
  - dept(deptno, dname, loc)
  - emp(empno, ename, job, mgr°→emp, hiredate, sal, comm°, deptno°→dept)
- Geben Sie vier Beispiele für fehlerhafte Anfragen und die zugehörige Fehlermeldung, die PostgreSQL ausgibt (4 Punkte).
  - Die Fehlermeldungen müssen unterschiedlich sein.

    Maximal zwei „syntax error at or near“.
  - Die Fehler sollen gefühlt häufig vorkommen.

    Wenn die Fehlermeldung nicht klar ist, müssen Sie den Fehler erläutern.
    Bei der Klausur können Sie wertvolle Zeit sparen, wenn Sie schon mit Fehlermeldungen vertraut sind.
  - Die Beispiel-Anfragen sollten kurz sein.

    Sie sollen nur den Fehler demonstrieren.