Einführung in Datenbanken

Übung 10: Programmierstil, NOT EXISTS, NULL

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

1. Präsenzaufg. 8
2. NULL
3. SQL Stil
4. Aggregationen
5. Hausaufgabe 9
6. Hausaufgabe 8
7. Präsenzaufg. 9

Präsenzaufgabe: Nichtmonotone Anfrage (1)

- 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 Verkäufer 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.

Falls es keinen „SALESMAN“ gibt, sollen alle Angestellten herauskommen.

Präsenzaufgabe: Nichtmonotone Anfrage (2)

EMP

EMPNO	ENAME	JOB	MGR	SAL	DEPTNO
7369	SMITH	CLERK	7902	800	20
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

Lösung der Präsenzaufgabe 8 (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, job
FROM   emp e
WHERE  NOT EXISTS (SELECT *
                   FROM   emp x
                   WHERE  x.job = 'SALESMAN'
                   AND    x.mgr <> e.empno)
```

Lösung der Präsenzaufgabe 8 (2)

- Lösung 2:

```
SELECT empno, ename, job
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.

Lösung der Präsenzaufgabe 8 (3)

Bemerkung zu Nullwerten:

- Im Prinzip kann die Spalte `mgr` einen Nullwert enthalten.
- Das kommt in den Daten nur für den `PRESIDENT` der Firma vor, nicht für einen `SALESMAN`.
- Aber das ist Wissen, was nicht in der Aufgabe steht, und nicht über einen `CHECK`-Constraint im Schema abgesichert ist.
- Streng genommen ist Lösung 1 falsch: Die Bedingung wäre auch erfüllt, wenn es einen `SALESMAN` gibt, der keinen Vorgesetzten hat (und es sonst nur einen Vorgesetzten aller Verkäufer gibt).

  Es gibt aber dennoch die volle Punktzahl für diese Lösung.
- Lösung 2 würde in dieser Situation das leere Ergebnis liefern.

Inhalt

1 Präsenzaufg. 8
2 NULL
3 SQL Stil
4 Aggregationen
5 Hausaufgabe 9
6 Hausaufgabe 8
7 Präsenzaufg. 9

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

Nullwerte (1)

- Warum funktioniert diese Anfrage nicht?

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN
WHERE  EMAIL = NULL
```

- Es gibt keine Fehlermeldung.

  Bei PostgreSQL und Oracle und einigen anderen DBMS. Andere Systeme (z.B. DB2) verlangen vielleicht `CAST(NULL AS VARCHAR(20))`, damit `NULL` einen definierten Datentyp hat.

- Das Ergebnis ist leer, obwohl Michael Grau einen Nullwert in der Spalte `EMAIL` hat.

- Wie könnte man die Anfrage korrigieren?

Nullwerte (2)

- Jeder Vergleich mit einem Nullwert gibt den dritten Wahrheitswert „unbekannt“, den SELECT wie „false/falsch“ behandelt.
- Deswegen braucht man einen speziellen Test auf einen Nullwert:

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN
WHERE  EMAIL IS NULL
```

- Aufgabe: Wenn SELECT den dritten Wahrheitswert sowieso wie false behandelt, braucht man dann wirklich eine dreiwertige Logik? Geben Sie ein Beispiel für eine Anfrage, bei der man am Ergebnis sehen kann, dass SQL wirklich eine dreiwertige Logik verwendet.

Nullwerte (3)

- Beispiel:

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN
WHERE  EMAIL = NULL
OR     NOT (EMAIL = NULL)
```

- In der zweiwertigen Logik gilt das Gesetz vom ausgeschlossenen Dritten. Dort ist die Bedingung `F OR NOT F` äquivalent zu `true`, es müssten also alle Studierenden herauskommen.
- Tatsächlich fehlt aber Michael Grau.
- Die Negation des dritten Wahrheitswertes „unbekannt“ ist wieder „unbekannt“.

Und „unbekannt `OR` unbekannt“ ist natürlich „unbekannt“.

Nullwerte (4)

Aufgabe:

- Geben Sie Vorname, Nachname und EMail-Adresse der Studierenden aus.
- Anstelle eines Nullwerts soll „–“ in der Ausgabe erscheinen.

Nullwerte (5)

- Lösung 1:

```
SELECT VORNAME, NACHNAME,
       COALESCE(EMAIL, '-') AS EMAIL
FROM   STUDENTEN
```

- Lösung 2 (etwas länger):

```
SELECT VORNAME, NACHNAME,
       CASE WHEN EMAIL IS NULL THEN '-'
            ELSE EMAIL END AS EMAIL
FROM   STUDENTEN
```

Nullwerte (6)

- Lösung 3 (erst später in der Vorlesung):

```
SELECT VORNAME, NACHNAME, EMAIL
FROM   STUDENTEN
WHERE  EMAIL IS NOT NULL
UNION  ALL
SELECT VORNAME, NACHNAME, '-' AS EMAIL
FROM   STUDENTEN
WHERE  EMAIL IS NULL
```

- Es wäre möglich, dass bei dieser Lösung 0.5 Punkte abgezogen werden wegen der Codeduplizierung.

  Andererseits wäre die Codeduplizierung nur minimal.

- Die Lösung würde auch ohne „ALL“ funktionieren.

  Dann findet eine hier überflüssige Duplikateliminierung statt (-0.5 Punkte).

Inhalt

1 Präsenzaufg. 8
2 NULL
3 SQL Stil
4 Aggregationen
5 Hausaufgabe 9
6 Hausaufgabe 8
7 Präsenzaufg. 9

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

1 Präsenzaufg. 8
2 NULL
3 SQL Stil
4 Aggregationen
5 Hausaufgabe 9
6 Hausaufgabe 8
7 Präsenzaufg. 9

Aggregationsfunktionen (1)

Wichtigste Aggregationsfunktionen in SQL:

- `COUNT(*)`: Anzahl.
- `COUNT(`⟨Term⟩`)`: Nur Nicht-Nullwerte werden gezählt.

  Duplikate werden mitgezählt.
- `COUNT(DISTINCT` ⟨Term⟩`)`: Anzahl nach Eliminierung von Nullwerten und Duplikaten.
- `MIN(`⟨Term⟩`)`: Minimum.
- `MAX(`⟨Term⟩`)`: Maximum.
- `AVG(`⟨Term⟩`)`: Durchschnitt (arithmetisches Mittel).
- `SUM(`⟨Term⟩`)`: Summe.

Aggregationsfunktionen (2)

Syntaxregeln für einfache Aggregationsanfragen:

- Aggregationsfunktionen können direkt nur unter `SELECT` verwendet werden.

  Man kann aber z.B. unter `WHERE` auch Unteranfragen verwenden, die eine Aggregationsfunktion unter `SELECT` enthalten. „Indirekt“ kann man sie also auch in anderen Teilen der Anfrage verwenden.

- Wenn eine Aggregationsfunktion unter `SELECT` verwendet wird (und es kein `GROUP BY` gibt), müssen alle Spalten unter `SELECT` innerhalb von Aggregationsfunktionen stehen.

  Es gibt bei diesen Anfragen immer genau eine Ausgabezeile. Eine Spalte, die nicht als Argument einer Aggregationsfunktion auftritt, hat im allgemeinen mehr als einen Wert. Das DBMS wüsste dann nicht, welcher Wert in der einen Zeile gedruckt werden soll.

- Aggregationsfunktionen können nicht geschachtelt werden.

Inhalt

1. Präsenzaufg. 8
2. NULL
3. SQL Stil
4. Aggregationen
5. Hausaufgabe 9
6. Hausaufgabe 8
7. Präsenzaufg. 9

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 9.1: Angestellte ohne Abteilung (1)

- Gesucht sind Angestellte, die noch keiner Abteilung zugeordnet sind (also einen Nullwert in der Spalte `deptno` haben), die aber einen Vorgesetzten haben. der einer Abteilung zugeordnet ist.

  Man könnte sie dann vielleicht auch dieser Abteilung zuordnen.

- Geben Sie folgende Daten aus:
  - Nummer und Name des Angestellten ohne Abteilung,
  - Nummer und Name des Vorgesetzten,
  - Nummer und Name der Abteilung des Vorgesetzten.

Aufgabe 9.1: Angestellte ohne Abteilung (2)

- Das erwartete Ergebnis ist:

empno	ename	Vorg. Nr.	Name	deptno	dname
7369	SMITH	7902	FORD	20	RESEARCH

- Bitte wählen Sie die Ausgabespalten wie im Beispiel.

  Die dritte Spalte heißt eigentlich „`Vorgesetzter Nr.`“. Sie musste für die Folie gekürzt werden.

Aufgabe 9.2: Auszahlungsberechnung (1)

- Einige Angestellten bekommen eine Provision für Verkäufe (der Wert steht in der Spalte `comm`: „commission“).
- Drucken Sie für jeden Angestellten
  - den Namen,
  - die Abteilung (den Namen der Abteilung),
  - das normale Gehalt (in der Spalte `sal`: „salary“),
  - die Provision und
  - die Auszahlung als Summe des Gehalts und der Provision.

Aufgabe 9.2: Auszahlungsberechnung (2)

- Für Angestellte, die keine Provision haben (Nullwert in der Spalte „`comm`“) geben Sie bitte „`(keine)`“ aus.

  Es könnte dabei das Problem auftreten, dass Zahlen und Zeichenketten in einer Spalte gemischt werden sollen. Sie müssen den Zahlwert also in eine Zeichenkette verwandeln (mittels `CAST` oder durch Konkatenation mit der leeren Zeichenkette).

- **Sortieren** Sie die Ausgabe alphabetisch nach dem Abteilungsnamen und bei gleicher Abteilung absteigend nach der Auszahlung, und, falls dies auch noch gleich ist, alphabetisch nach dem Namen des Angestellten.

- Das erwartete Ergebnis steht auf der nächsten Folie.

Aufgabe 9.2: Auszahlungsberechnung (3)

Angest.	Abteilung	Gehalt	Provision	Auszahlung
KING	ACCOUNTING	5000	(keine)	5000
CLARK	ACCOUNTING	2450	(keine)	2450
MILLER	ACCOUNTING	1300	(keine)	1300
FORD	RESEARCH	3000	(keine)	3000
SCOTT	RESEARCH	3000	(keine)	3000
JONES	RESEARCH	2975	(keine)	2975
ADAMS	RESEARCH	1100	(keine)	1100
BLAKE	SALES	2850	(keine)	2850
MARTIN	SALES	1250	1400	2650
ALLEN	SALES	1600	300	1900
WARD	SALES	1250	500	1750
TURNER	SALES	1500	0	1500
JAMES	SALES	950	(keine)	950

Aufgabe 9.3: Gehaltsspanne pro Beruf (1)

- Geben Sie für jeden Beruf (Spalte `job`) das minimale und das maximale Gehalt aus und den Angestellten mit minimalem und maximalem Gehalt.

  Gehalt ist der Wert in der Spalte `sal` („salary“), die Provision in der Spalte `comm` soll hier nicht berücksichtigt werden.

- Die Daten sollen nur für Berufe ausgegeben werden, bei denen es tatsächlich eine Gehaltsspanne gibt, d.h. das minimale Gehalt muss echt kleiner als das maximale Gehalt sein.

  Falls mehrere Angestellte das gleiche minimale oder maximale Gehalt haben, ist es in Ordnung, dass es mehrere Ausgabezeilen für einen Beruf gibt. Es soll dann jede Kombination eines Angestellten mit minimalem Gehalt und eines Angestellten mit maximalem Gehalt in der Ausgabe erscheinen. Glücklicherweise ist dieser Fall im Beispiel-Zustand selten, es gibt dort nur zwei Verkäufer mit dem gleichen minimalen Gehalt.

Aufgabe 9.3: Gehaltsspanne pro Beruf (2)

- Das erwartete Ergebnis ist:

Beruf	von	bis	Min. Gehalt	Max. Gehalt
CLERK	800	1300	SMITH	MILLER
MANAGER	2450	2975	CLARK	JONES
SALESMAN	1250	1600	MARTIN	ALLEN
SALESMAN	1250	1600	WARD	ALLEN

- Wählen Sie die Spaltenüberschriften bitte wie im Beispiel gezeigt.

  Eine spezielle Sortierung ist nicht verlangt. Wenn Sie sich den Vergleich mit der Beispiel-Ausgabe erleichtern wollen, sortieren Sie nach Beruf, Min. Gehalt und Max. Gehalt.

Aufgabe 9.3: Gehaltsspanne pro Beruf (3)

- **Hinweis:** Erwartet wird eine Lösung unter Verwendung von `NOT EXISTS`. Es steht Ihnen aber frei, auch Aggregationsfunktionen einzusetzen.

Aufgabe 9.4: Interessante Sortierung (1)

- Geben Sie alle Angestellten mit Beruf und Abteilungs-Namen aus.
- Die Abteilungen sollen alphabetisch sortiert sein.
- Innerhalb jeder Abteilung soll
  - zuerst der „`PRESIDENT`“ aufgelistet werden,

    Falls in dieser Abteilung vorhanden.
  - dann der Manager,
  - dann der Clerk und
  - dann alle anderen Angestellten.
- In jedem Abschnitt (Präsident, . . .) sollen die Angestellten alphabetisch nach ihren Namen sortiert werden.

Aufgabe 9.4: Interessante Sortierung (2)

Abteilung	Beruf	Angestellter
ACCOUNTING	PRESIDENT	KING
ACCOUNTING	MANAGER	CLARK
ACCOUNTING	CLERK	MILLER
RESEARCH	MANAGER	JONES
RESEARCH	CLERK	ADAMS
RESEARCH	ANALYST	FORD
RESEARCH	ANALYST	SCOTT
SALES	MANAGER	BLAKE
SALES	CLERK	JAMES
SALES	SALESMAN	ALLEN
SALES	SALESMAN	MARTIN
SALES	SALESMAN	TURNER
SALES	SALESMAN	WARD

Aufgabe 9.4: Interessante Sortierung (3)

- **Hinweis:** Sie können (bei PostgreSQL) unter `ORDER BY` Terme (Wertausdrücke) verwenden, nicht nur einzelne Spalten.
  - Sie können also nach berechneten Werten sortieren.
  - Im Skript werden Wertausdrücke mit Fallunterscheidungen besprochen.
- Es gibt aber auch eine andere Lösung, die wie beim Logikrätsel mit einer `WITH`-Hilfstabelle und `VALUES` arbeitet.
  - Sie dürfen dafür voraussetzen, dass `PRESIDENT`, `MANAGER`, `CLERK`, `ANALYST` und `SALESMAN` die einzigen Berufe in der Firma sind.

Aufgabe 9.5: Statistik (1)

- Betrachtet seien alle Angestellten mit Untergebenen (in der Spalte `mgr` ist die `empno` des direkten Vorgesetzten eingetragen). Berechnen Sie für diese Angestellten:
  - Anzahl dieser Angestellten (`anz`),
  - das minimale, das maximale und das durchschnittliche Gehalt aus der Spalte `sal` (`min_sal`, `max_sal`, `avg_sal`),
  - die Anzahl verschiedener Jobs (`anz_jobs`).

    Z.B. soll auch dann, wenn zwei Angestelle den Job `MANAGER` haben, der Beruf `MANAGER` nur ein Mal gezählt werden.

  Nennen Sie die Ausgabespalten, wie in Klammern gezeigt. Runden Sie das Durchschnittsgehalt durch Aufruf der Funktion `ROUND(...)`. Die Provision in der Spalte `comm` brauchen Sie nicht zu berücksichtigen. Ihre Anfrage muss genau eine Ergebniszeile mit allen diesen Daten liefern (wie immer bei Aggregationsanfragen ohne `GROUP BY`).

Aufgabe 9.5: Statistik (2)

- Die Angestellten mit Untergebenen sind im Beispielzustand:

empno	ename	job	mgr	sal	comm	deptno
7566	JONES	MANAGER	7839	2975	NULL	20
7698	BLAKE	MANAGER	7839	2850	NULL	30
7782	CLARK	MANAGER	7839	2450	NULL	10
7788	SCOTT	ANALYST	7566	3000	NULL	20
7839	KING	PRESIDENT	NULL	5000	NULL	10
7902	FORD	ANALYST	7566	3000	NULL	20

- Die erwartete Antwort ist:

anz	min_sal	max_sal	avg_sal	anz_jobs
6	2450	5000	3213	3

1 Datensatz

Inhalt

1. Präsenzaufg. 8
2. NULL
3. SQL Stil
4. Aggregationen
5. Hausaufgabe 9
6. Hausaufgabe 8
7. Präsenzaufg. 9

Datenbank der US-Präsidenten

- Schema president_public im Adminer:
  - state(state_name, admin_entered, year_entered)
  - president(pres_name, birth_year, years_serv, death_age, party, state_born→state)
  - pres_hobby(pres_name→president, hobby)
  - administration(admin_nr, pres_name→president, year_inaugurated)
  - admin_pr_vp((admin_nr, pres_name)→administration, vice_pres_name)
  - pres_marriage(pres_name→president, spouse_name, pr_age, sp_age, nr_children, mar_year)
  - election(election_year, candidate, votes, winner_loser_indic)

Aufgabe 8.1: Staaten ohne Präsidenten (1)

- Aus welchen Staaten gab es seit 1850 keinen Presidenten mehr?
- Genauer: Es gab keinen Presidenten, der in dem Staat geboren ist (`state_born`) und dessen Administration ab 1850 (einschließlich) eingesetzt wurde (`year_inaugurated`).
- Geben Sie jeweils den Namen der Staaten aus (alphabetisch **sortiert**).
- Das erwartete Ergebnis hat 30 Zeilen, siehe nächste Folie.

Aufgabe 8.1: Staaten ohne Präsidenten (2)

state_name
Alabama
Alaska
Arizona
Colorado
Delawhare
Florida
Idaho
Indiana
Kansas
Louisiana

state_name
Maine
Maryland
Michigan
Minnesota
Mississippi
Montana
Nevada
New Mexico
North Dakota
Oklahoma

state_name
Oregon
Rhode Island
South Carolina
South Dakota
Tennesee
Utah
Washington
West Virginia
Wisconsin
Wyoming

Aufgabe 8.1: Staaten ohne Präsidenten (3)

Lösung:

- Diese Anfrage ist nichtmonoton und braucht `NOT EXISTS` oder `NOT IN`:

```
SELECT state_name
FROM   state
WHERE  state_name NOT IN
       (SELECT p.state_born
        FROM   president p, administration a
        WHERE  p.pres_name = a.pres_name
        AND    a.year_inaugurated >= 1850)
ORDER  BY state_name
```

Im Schema ist die Spalte `state_born` als `NOT NULL` deklariert. Das ist wichtig, damit das `NOT IN` funktioniert. Bei `NOT EXISTS` gibt es dieses Problem nicht.

Aufg. 8.2: Staaten mit genau einem Präsidenten (1)

- Aus welchen Bundesstaaten gab es genau einen Presidenten?
- D.h. es gab einen Presidenten, der in dem Staat geboren ist, aber keinen weiteren Präsidenten, der im gleichen Staat geboren ist.

  Im Gegensatz zu Aufgabe 1 spielt das Jahr der Amtseinführung hier keine Rolle.

- Geben Sie den Namen des Staates und den Namen des Präsidenten aus und **sortieren** Sie die Ausgabe nach dem Staat.
- Die Ausgabespalten sollen `State` und `President` heißen (mit dieser Groß-/Kleinschreibung).

Aufg. 8.2: Staaten mit genau einem Präsidenten (2)

- Das erwartete Ergebnis ist:

State	President
Arkansas	Clinton W J
California	Johnson L B
Connecticut	Bush G W
Hawaii	Obama B H
Illinois	Reagan R
Iowa	Hoover H C
Kentucky	Lincoln A
Missouri	Truman H S
Nebraska	Nixon R M
New Hampshire	Pierce F
New Jersey	Cleveland G
South Carolina	Jackson A

Aufg. 8.2: Staaten mit genau einem Präsidenten (3)

Lösung:

- Man testet mit `NOT EXISTS`, dass es keinen zweiten Präsidenten aus dem gleichen Staat gibt:

```
SELECT p.state_born as "State",
       p.pres_name AS "President"
FROM   president p
WHERE  NOT EXISTS
       (SELECT * FROM president p2
        WHERE  p2.state_born = p.state_born
        AND    p2.pres_name <> p.pres_name)
ORDER  BY "State"
```

Eine Tupelvariable über `state` wäre überflüssig und würde einen halben Punkt kosten wegen eines überflüssigen Joins.

Aufgabe 8.3: Spät verheiratete Präsidenten (1)

- Welche Präsidenten haben im Alter von 40 und mehr erstmals geheiratet, d.h.
  - sie waren verheiratet, wobei `pr_age` mindestens 40 war,
  - aber es gibt keine andere Eheschließung des gleichen Präsidenten, wo er jünger als 40 Jahre alt war.
- Geben Sie den Namen des Präsidenten, sein Alter bei der Hochzeit und das Jahr der Eheschließung aus.
- Das erwartete Ergebnis ist:

pres_name	pr_age	mar_year
Madison J	43	1794
Cleveland G	49	1886

Aufgabe 8.3: Spät verheiratete Präsidenten (2)

Lösung:

- Man braucht `NOT EXISTS` für Eheschließungen vor 40:

```
SELECT m.pres_name, m.pr_age, m.mar_year
FROM   pres_marriage m
WHERE  m.pr_age >= 40
AND    NOT EXISTS
       (SELECT * FROM pres_marriage m2
        WHERE  m.pres_name = m2.pres_name
        AND    m2.pr_age < 40)
ORDER  BY pr_age;
```

Die Bedingung `m.pr_age >= 40` ist eigentlich überflüssig: Die Unteranfrage stellt ja sicher, dass keine Eheschließung dieses Präsidenten vor 40 war.
Es ist aber vielleicht klarer, kostet kaum etwas, eventuell könnte sich durch den Filter vor dem Anti-Join (`NOT EXISTS`) die Performance sogar verbessern.

Aufgabe 8.4: Hobbies von zwei Präsidenten (1)

- Welche Hobbies haben nach der Datenbank genau zwei Präsidenten?
- Geben Sie das Hobby und die Namen der beiden Präsidenten aus, wobei der Name des ersten Präsidenten alphabetisch vor dem Namen des zweiten Präsidenten kommen soll.
- Sie müssen prüfen, dass es außer den beiden Präsidenten keinen weiteren Präsidenten gibt, der das gleiche Hobby hat.
- Nennen Sie die Spalten für die beiden Namen „`President 1`“ und „`President 2`“.

Aufgabe 8.4: Hobbies von zwei Präsidenten (2)

- Das erwartete Ergebnis ist:

hobby	President 1	President 2
Billards	Adams J Q	Garfield J A
Poker	Harding W G	Truman H S
Reading	Bush G W	Clinton W J
Shooting	Hayes R B	Roosevelt T

Aufgabe 8.4: Hobbies von zwei Präsidenten (3)

Lösung:

- Hier braucht man einen Selbstverbund und `NOT EXISTS`:

```
SELECT h1.hobby,
       h1.pres_name AS "President 1",
       h2.pres_name AS "President 2"
FROM   pres_hobby h1, pres_hobby h2
WHERE  h1.hobby = h2.hobby
AND    h1.pres_name < h2.pres_name
AND    NOT EXISTS
       (SELECT * FROM pres_hobby h3
        WHERE  h3.hobby = h1.hobby
        AND    h3.pres_name <> h1.pres_name
        AND    h3.pres_name <> h2.pres_name)
ORDER  BY h1.hobby
```

Aufgabe 8.5: Immer gewonnen (1)

- Welche Präsidenten ab 1980 (`year_inaugurated`) haben jede Wahl, zu der sie angetreten sind, gewonnen?

  Wert „`W`“ in `winner_loser_indic` in allen `election`-Zeilen mit dem Präsidenten als `candidate`.

- Geben Sie außerdem das Jahr der Amtseinführung (`year_inaugurated`) des Präsidenten aus und **sortieren** Sie nach dem Jahr.

- Manche Präsidenten hatten mehrere (zwei) Präsidentschaften. Geben Sie aber jeden Präsidenten nur einmal aus, und zwar jeweils mit dem Jahr seiner ersten Präsidentschaft (das minimale `year_inaugurated`).

  Die Grenze 1980 ist so gemeint, dass die erste Präsidentschaft frühestens 1980 startete. Bei den Wahlen sollen Sie dagegen alle berücksichtigen, ggf. auch solche vor 1980.

Aufgabe 8.5: Immer gewonnen (2)

- Das erwartete Ergebnis ist:

pres_name	year_inaugurated
Reagan R	1981
Clinton W J	1993
Bush G W	2001
Obama B H	2009
Biden J R	2021

Aufgabe 8.5: Immer gewonnen (3)

Lösung:

- „Immer gewonnen“ heißt niemals verloren:

```
SELECT a.pres_name, a.year_inaugurated
FROM   administration a
WHERE  a.year_inaugurated >= 1980
AND    NOT EXISTS
       (SELECT * FROM election e
        WHERE  e.candidate = a.pres_name
        AND    e.winner_loser_indic <> 'W')
AND    NOT EXISTS
       (SELECT * FROM administration b
        WHERE  b.pres_name = a.pres_name
        AND    b.year_inaugurated <
                  a.year_inaugurated)
ORDER  BY a.year_inaugurated
```

Inhalt

1 Präsenzaufg. 8
2 NULL
3 SQL Stil
4 Aggregationen
5 Hausaufgabe 9
6 Hausaufgabe 8
7 Präsenzaufg. 9

Präsenzaufgabe: Nullwerte und Aggregationen (1)

- Vergleichen Sie die folgenden Anfragen. Kreuzen Sie anschließend die erste korrekte Aussage an.
- Anfrage 1:

```
SELECT COUNT(1)
FROM   STUDENTEN
WHERE  EMAIL IS NOT NULL
```

- Anfrage 2:

```
SELECT COUNT(*)
FROM   STUDENTEN
WHERE  EMAIL = EMAIL
```

- Schema: STUDENTEN(SID, VORNAME, NACHNAME, EMAIL°)

Präsenzaufgabe: Nullwerte und Aggregationen (2)

A. Anfrage 1 enthält einen Syntaxfehler.

B. Anfrage 1 liefert immer 1 (sofern `STUDENTEN` nicht leer).

C. Das Ergebnis von Anfrage 2 würde sich nicht ändern, wenn man die `WHERE`-Bedingung weglässt.

D. Anfrage 1 und Anfrage 2 liefern immer denselben Wert.

E. Es gibt DB-Zustände, in denen die beiden Anfragen unterschiedliche Werte liefern.