Einführung in Datenbanken

Übung 11: Aggregationsanfragen, GROUP BY, HAVING

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. Organisatorisches
2. Präsenzaufg. 9
3. GROUP BY
4. Hausaufgabe 10
5. Hausaufgabe 9
6. Präsenzaufg. 10

Organisatorisches (1)

- Haben Sie Fragen?

  Es ist ein wichtiger Zweck der Übung, Fragen zu beantworten.

- Diese Folien stehen morgen oder übermorgen auf der Webseite (Menupunkt „Übung“):

  [https://users.informatik.uni-halle.de/~brass/db24/uebung.html]

  Es gab keine Beschwerden, dass ich das vor zwei Wochen vergessen hatte.
  Deswegen frage ich mich, ob sich jemand dafür interessiert.

- In den letzten ca. 3 Wochen wurden die Folien um jeweils eine mögliche Lösung der Hausaufgaben erweitert.

Organisatorisches (2)

- Alte Klausuren stehen auf der Webseite (unter „Prüfung“):

  [https://users.informatik.uni-halle.de/~brass/db24/pruefung.html]

- Inzwischen auch mit Link zum Adminer, so dass Sie Ihre Lösungen ausprobieren können.

  Es könnte gelegentlich eine Aufgabe geben, bei der `UNION` benötigt wird, oder ein Outer Join von Vorteil wäre. Beides wird erst noch besprochen. Die meisten Aufgaben sind aber schon lösbar. `UNION` ist ja auch nicht schwer zu verstehen.

- Sollten Sie in der Weihnachtspause also Zeit und Lust haben, könnten Sie vielleicht probehalber eine Klausur durchrechnen.

  Nur die SQL-Anfragen. Aufgaben zum Datenbank-Entwurf und zur relationalen Algebra können Sie noch nicht lösen.

Organisatorisches (3)

- Es wird noch eine Bonus-Aufgaben geben, bei der Sie ein SQL-Lernspiel ausprobieren können, das als Bachelorarbeit in meiner Gruppe entwickelt wurde.

  Eine Web-Schnittstelle ist auch in Arbeit, aber aktuell einsetzbar ist ein Java-Programm mit GUI, das Sie auf Ihrem Rechner installieren müssten.

- Es handelt sich um ein Text-Adventurespiel, bei dem die Daten in Tabellen stehen.

  Vielleicht haben Sie von Spielen wie „Zork“ gehört. Man kann sich in einer Spielwelt bewegen, Objekte nehmen und verwenden und damit Aufgaben/Rätsel lösen. Unser Spiel ist allerdings viel kleiner und einfach zu lösen. Sie haben außerdem Zugriff auf die Daten des Spiels (zumindest soweit, wie Sie die Spielwelt schon erkundet haben). Man kann klassische Adevnture-Befehle oder auch SQL-Anfragen und Updates verwenden.

- Siehe Übungs-Webseite ab ca. Mittwoch abend.

Inhalt

1 Organisatorisches

2 Präsenzaufg. 9

3 GROUP BY

4 Hausaufgabe 10

5 Hausaufgabe 9

6 Präsenzaufg. 10

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.

Präsenzaufgabe: Nullwerte und Aggregationen (3)

Lösung:

- Antwort `D` ist richtig:
  Anfrage 1 und 2 liefern immer denselben Wert.
- Das Argument von `COUNT` ist ein Term, die Konstante 1 ist also zulässig (aber ungewöhnlich).
- `COUNT(1)` liefert das gleiche Ergebnis wie `COUNT(*)`, da `1` niemals Null ist.
- Auch `COUNT(27)` würde das gleiche Ergebnis liefern: Es wird nur gezählt, und nicht aufsummiert. Im Beispiel also die Anzahl der Elemente der Multimenge $\{27, 27, 27, \ldots, 27\}$.

  Das Argument von `COUNT` ist nur wichtig, wenn es Nullwerte enthalten kann (die nicht mitgezählt werden). Ansonsten sollte man `COUNT(*)` verwenden.

Präsenzaufgabe: Nullwerte und Aggregationen (4)

Lösung, Forts.:

- Falls `EMAIL` ein Nullwert ist, liefert

  ```
  EMAIL = EMAIL
  ```

  den dritten Wahrheitswert „unbekannt“, und nicht „true“.

- Dagegen liefert

  ```
  EMAIL IS NOT NULL
  ```

  in diesem Fall den Wahrheitswert „false“.

- Für `SELECT` ist es egal, ob die `WHERE`-Bedingung „unbekannt“ oder „false“ ist: In beiden Fällen erfolgt keine Ausgabe.

  Ein Unterschied ist feststellbar durch Negation (`NOT`): Die Negation von „false“ ist natürlich „true“, die von „unbekannt“ ist „unbekannt“.

  Deswegen wird die „`IS NULL`“-Bedingung wirklich gebraucht.

Inhalt

1 Organisatorisches

2 Präsenzaufg. 9

3 GROUP BY

4 Hausaufgabe 10

5 Hausaufgabe 9

6 Präsenzaufg. 10

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

Aggregationsfunktionen

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.

Aggregations-Anfragen (1)

Aufgabe:

- Ist diese Anfrage syntaktisch korrekt?

```
SELECT ANR, MIN(PUNKTE), MAX(PUNKTE),
       AVG(PUNKTE)
FROM   BEWERTUNGEN
WHERE  ATYP = 'H'
```

A. Korrekt.

B. Syntaxfehler.

Aggregations-Anfragen (2)

Lösung:

- Nein, die Anfrage gibt einen Syntaxfehler:

```
SELECT ANR, MIN(PUNKTE), MAX(PUNKTE),
       AVG(PUNKTE)
FROM   BEWERTUNGEN
WHERE  ATYP = 'H'
```

- Error in query (7):

  ERROR: column "bewertungen.anr" must appear in the GROUP BY clause or be used in an aggregate function

  LINE 1: SELECT ANR, MIN(PUNKTE), MAX(PUNKTE),

Aggregations-Anfragen (3)

Korrektur:

- Man muss `GROUP BY` verwenden und bekommt dann für jede Hausaufgabe eine Ausgabezeile:

```
SELECT ANR, MIN(PUNKTE), MAX(PUNKTE),
       AVG(PUNKTE)
FROM   BEWERTUNGEN
WHERE  ATYP = 'H'
GROUP  BY ANR
```

anr	min	max	avg
1	5.0	10.0	8.0
2	8.0	9.0	8.5

Aggregations-Anfragen (4)

Aufgabe:

- Was halten Sie von dieser Anfrage, bei der zusätzlich nac ATYP gruppiert wird?

```
SELECT ANR, MIN(PUNKTE), MAX(PUNKTE),
       AVG(PUNKTE)
FROM   BEWERTUNGEN
WHERE  ATYP = 'H'
GROUP  BY ANR, ATYP
```

A. Korrekt, aber gleiches Ergebnis.

B. Korrekt, und anderes Ergebnis als letzte Anfrage.

C. Syntaxfehler.

Aggregations-Anfragen (5)

Lösung:

- Es ist syntaktisch zulässig.

  Die Menge der `SELECT`-Attribute außerhalb von Aggregationsfunktionen muss eine Teilmenge (oder gleich) der Menge der `GROUP BY`-Attribute sein.

- Es ist sinnlos. Die Gruppen ändern sich dadurch nicht, weil `ATYP` nur einen möglichen Wert hat.

  Aufgrund der `WHERE`-Bedingung. D.h. Antwort A ist korrekt.

- Ansonsten würden die Gruppen feiner werden (weiter aufgespalten).

  Wenn die `WHERE`-Bedingung nicht wäre, würde man mehrere Ausgaben für die gleiche `ANR` bekommen (eine für den Aufgabentyp „`H`“ und eine für den Aufgabentyp „`Z`“). Wenn man den `ATYP` nicht ausgibt, ist das aber nicht hilfreich.

Aggregations-Anfragen (6)

- Die folgende Anfrage gibt alle Studenten aus, die mindestens 15 Punkte in den Hausaufgaben haben:

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN S
WHERE  15 <= (SELECT SUM(B.PUNKTE)
              FROM   BEWERTUNGEN B
              WHERE  B.SID = S.SID
              AND    B.ATYP = 'H')
```

- Es ist garantiert, dass die Unteranfrage nur eine Zeile liefert, deswegen kann es keinen Laufzeit-Fehler geben.

  Jede Aggregation ohne `GROUP BY` (und `HAVING`) liefert immer genau eine Zeile.

- Man darf die Unteranfrage auch auf die linke Seite des Vergleichs schreiben.

Aggregations-Anfragen (7)

- Was halten Sie von dieser Anfrage?
  Wieder ist die Aufgabe, alle Studenten aufzulisten,
  die mindestens 15 Punkte in den Hausaufgaben haben.

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
AND    SUM(B.PUNKTE) >= 15
```

A. Syntaxfehler

B. Syntaktisch korrekt, aber falsches Ergebnis

C. Korrekt

Aggregations-Anfragen (8)

Lösung:

- Das ist natürlich ein Syntaxfehler.
- Aggregationsfunktionen wie `SUM` können nicht direkt unter `WHERE` verwendet werden.

  Aber schon in Unteranfragen, die unter `WHERE` geschachtelt sind.
- Es ist also „A“ richtig.

Aggregations-Anfragen (9)

Korrektur:

- HAVING ist für Aggregations-Bedingungen gemacht:

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
GROUP  BY VORNAME, NACHNAME
HAVING SUM(B.PUNKTE) >= 15
```

Mindestens wenn man die Punktesumme auch ausgeben will, wäre dies die Standardlösung. Die Variante mit der Unterabfrage würde dann eine Codeduplizierung erfordern. Wenn man dagagen nach Studierenden mit weniger als 15 Punkten sucht, würde die Variante mit der Unterabfrage auch Iris Winter ausgeben, bei dieser Lösung würde sie dagegen durch den Join nicht im Ergebnis erscheinen.

Aggregations-Anfragen (10)

- Würde dies funktionieren? (Wieder sind alle Studierenden mit mindestens 15 Hausaufgaben-Punkten gesucht.)

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
GROUP  BY S.SID
HAVING SUM(B.PUNKTE) >= 15
```

A. Syntaxfehler

B. Syntaktisch korrekt, aber falsches Ergebnis

C. Korrekt

Aggregations-Anfragen (11)

Lösung:

- Nach den Angaben in der Vorlesung ist das letzte Beispiel ein Syntaxfehler, weil `S.VORNAME` und `S.NACHNAME` nicht unter `GROUP BY` stehen.

  Siehe aber die nächste Folie.

- Man könnte sie hinzufügen, ohne die Gruppen zu ändern, und alles wäre gut:

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
GROUP  BY S.SID, S.VORNAME, S.NACHNAME
HAVING SUM(B.PUNKTE) >= 15
```

Aggregations-Anfragen (12)

Lösung, Forts.:

- In PostgreSQL wird es aber auch ohne diese Hinzufügung akzeptiert, weil nach dem Schlüssel `S.SID` von `S` gruppiert wird, und PostgreSQL daher weiss, dass alle anderen Attribute von `S` eindeutig bestimmt sind.

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
GROUP  BY S.SID
HAVING SUM(B.PUNKTE) >= 15
```

- In Oracle und Microsoft SQL Server gibt dies einen Fehler.

  Sie müssen mit Punktabzug für Stil (Portabilität) rechnen. Es ist im Standard (seit SQL-99), aber auch PostgreSQL implementiert nur den einfachsten Fall.

Inhalt

1 Organisatorisches

2 Präsenzaufg. 9

3 GROUP BY

4 Hausaufgabe 10

5 Hausaufgabe 9

6 Präsenzaufg. 10

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 10.1: Abteilungs-Daten inkl. Budget (1)

- Geben Sie für jede Abteilung aus:
  - Nummer und Name der Abteilung,
  - die Anzahl der Angestellten in der Abteilung,
  - die Anzahl Angestellter mit Provision (d.h. kein Nullwert in `comm`),
  - das Gesamt-Budget für die Personalkosten der Abteilung, d.h. die Summe aller Gehälter (`sal`) und Provisionen (`comm`).
- **Sortieren** Sie das Ergebnis absteigend nach dem Budget, bei gleichem Budhet nach der Abteilungsnummer.

Aufgabe 10.1: Abteilungs-Daten inkl. Budget (2)

- Das erwartete Ergebnis ist:

deptno	dname	Angestelle	prov.	Budget
30	SALES	6	4	11600
20	RESEARCH	4	0	10075
10	ACCOUNTING	3	0	8750

- Bitte wählen Sie die Ausgabespalten wie im Beispiel.

  Die Spalte „`prov.`“ heißt eigentlich „`provisionsberechtigt`“.

- Leere Abteilungen sollen nicht in der Ausgabe erscheinen.

  Das vereinfacht die Aufgabe eher.

Aufgabe 10.2: Job-Statistik (1)

- Geben Sie für jeden Job, den mindestens drei Angestellte haben, Folgendes aus:
  - die Anzahl Angestellter mit diesem Job,
  - die Anzahl Abteilungen, in denen der Job vorkommt, sowie
  - das minimale, maximale und durchschnittliche Gehalt aus.
- Runden Sie bitte das durchschnittliche Gehalt auf eine ganze Zahl und **sortieren** Sie die Ausgabe alphabetisch nach dem Job.

Aufgabe 10.2: Job-Statistik (2)

- Das erwartete Ergebnis ist:

job	Ang.	Abt.	von	bis	Durchschnitt
CLERK	4	3	800	1300	1038
MANAGER	3	3	2450	2975	2758
SALESMAN	4	1	1250	1600	1400

- Bitte wählen Sie die Ausgabespalten wie im Beispiel.

  Die Spalte „`Ang.`“ heißt eigentlich „`Angestellte`“ und die Spalte „`Abt.`“ eigentlich „`Abteilungen`“.

Aufgabe 10.3: Gehaltsklassen pro Abteilung (1)

- Geben Sie für jede Abteilung Folgendes aus:
  - Nummer und Name der Abteilung,
  - das durchschnittliche Gehalt (gerundet auf eine ganze Zahl),
  - die Anzahl Angestellter mit Gehalt bis 2000 $,
  - die Anzahl Angestellter mit Gehalt über 2000 $.
- Das erwartete Ergebnis ist:

deptno	dname	AVG	bis 2000	ueber 2000
10	ACCOUNTING	2917	1	2
20	RESEARCH	2519	1	3
30	SALES	1567	5	1

Aufgabe 10.3: Gehaltsklassen pro Abteilung (2)

- Wählen Sie die Spaltenüberschriften bitte wie im Beispiel gezeigt.

  Die Spalte „`AVG`“ heißt eigentlich „`Durchschnitt`“.

- Eine spezielle Sortierung ist nicht verlangt.

  Wenn Sie sich den Vergleich mit der Beispiel-Ausgabe erleichtern wollen, sortieren Sie nach `deptno`.

Aufgabe 10.4: Maximale Anzahl Untergebene

- Geben Sie den oder die Angestellten mit der maximalen Anzahl (direkter) Untergebener aus.

  Die Angestellten-Nummer des direkten Vorgesetzten steht in der Spalte `mgr`.

- Geben Sie Nummer, Name und Job des Vorgesetzten, sowie die Anzahl direkter Untergebener aus.

- Das erwartete Ergebnis ist:

empno	ename	job	Untergebene
7698	BLAKE	MANAGER	5

Aufgabe 10.5: Nummerierung der Angestellten (1)

- Nummerieren Sie die Angestellten durch (1, 2, ...), und zwar nach dem Gehalt (größtes zuerst), und bei gleichem Gehalt nach dem Namen.

  Sie können diese Aufgabe lösen, indem Sie für jeden Angstellten zählen, wie viele Angestellte (A) mehr verdienen als der aktuelle Angestellte, oder (B) genauso viel verdienen wie der aktuelle Angestellte, aber einen Namen haben, der alphabetisch vor dem des aktuellen Angestellten kommt (oder gleich ist). Bei dieser Aufgabe wird angenommen, dass die Angestellten-Namen eindeutig sind.

- Geben Sie die berechnete Nummer, den Namen und das Gehalt für jeden Angestellten aus.
- **Sortieren** Sie die Ausgabe nach der Nummer.
- Die erwartete Antwort steht auf der nächsten Folie.

Aufgabe 10.5: Nummerierung der Angestellten (2)

Nr	ename	sal
1	KING	5000
2	FORD	3000
3	SCOTT	3000
4	JONES	2975
5	BLAKE	2850
6	CLARK	2450
7	ALLEN	1600
8	TURNER	1500
9	MILLER	1300
10	MARTIN	1250
11	WARD	1250
12	ADAMS	1100
13	JAMES	950
14	SMITH	800

Inhalt

1 Organisatorisches
2 Präsenzaufg. 9
3 GROUP BY
4 Hausaufgabe 10
5 Hausaufgabe 9
6 Präsenzaufg. 10

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 9.1: Angestellte ohne Abteilung (3)

Lösung:

- Diese Anfrage benötigt `IS NULL` und einen Selbstverbund:

```
SELECT u.empno, u.ename,
       v.empno "Vorgesetzter Nr.",
       v.ename "Name",
       d.deptno, d.dname
FROM   emp u, emp v, dept d
WHERE  u.deptno IS NULL
AND    u.mgr = v.empno
AND    v.deptno = d.deptno
```

Man braucht nicht extra zu testen, dass `v.deptno` nicht Null ist:
Die Join-Bedingung `v.deptno = d.deptno` wäre dann nicht `true`.

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 9.2: Auszahlungsberechnung (4)

Lösung:

- Diese Anfrage benötigt `COALESCE` (oder `CASE`):

```
SELECT e.ename AS "Angestellter",
       d.dname AS "Abteilung",
       e.sal AS "Gehalt",
       COALESCE(e.comm || '', '(keine)')
           AS "Provision",
       e.comm,
       e.sal+COALESCE(e.comm,0) AS "Auszahlung"
FROM   emp e, dept d
WHERE  e.deptno = d.deptno
ORDER  BY "Abteilung", "Auszahlung" DESC,
          "Angestellter"
```

Aufgabe 9.2: Auszahlungsberechnung (5)

Lösung, Forts.:

- Bei `COALESCE` müssen beide Argumente den gleichen Datentyp haben (zumindest kompatible Typen).
- Eine Zahl (Provision) und eine Zeichenkette (`'keine'`) geht nicht.
- Die Konkatenation einer Zahl mit einer Zeichenkette ist zumindest in PostgreSQL möglich und konvertiert die Zahl in eine Zeichenkette.
- Selbstverständlich wäre auch eine explizite Typumwandlung möglich:

```
COALESCE(CAST(comm AS VARCHAR(10)), '(keine)')
```

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.

Lösung:

- Auf der nächsten Folie steht die erwartete Lösung mit `NOT EXISTS`.
- Auf der übernächsten Folie eine Lösung mit `MIN` und `MAX`.
- Beide Lösungen brauchen außerdem einen Selbstverbund.

Aufgabe 9.3: Gehaltsspanne pro Beruf (4)

```
SELECT e_min.job AS "Beruf",
       e_min.sal AS "von", e_max.sal AS "bis",
       e_min.ename "Min. Gehalt",
       e_max.ename "Max. Gehalt"
FROM   emp e_min, emp e_max
WHERE  e_min.job = e_max.job
AND    e_min.sal < e_max.sal
AND    NOT EXISTS(SELECT * FROM emp weniger
                  WHERE  weniger.job = e_min.job
                  AND    weniger.sal < e_min.sal)
AND    NOT EXISTS(SELECT * FROM emp mehr
                  WHERE  mehr.job = e_max.job
                  AND    mehr.sal > e_max.sal)
ORDER  BY e_min.job, e_min.ename, e_max.ename
```

Aufgabe 9.3: Gehaltsspanne pro Beruf (5)

```
SELECT e_min.job AS "Beruf",
       e_min.sal AS "von", e_max.sal AS "bis",
       e_min.ename "Min. Gehalt",
       e_max.ename "Max. Gehalt"
FROM   emp e_min, emp e_max
WHERE  e_min.job = e_max.job
AND    e_min.sal < e_max.sal
AND    e_min.sal = (SELECT MIN(e.sal)
                    FROM   emp e
                    WHERE  e.job = e_min.job)
AND    e_max.sal = (SELECT MAX(e.sal)
                    FROM   emp e
                    WHERE  e.job = e_max.job)
ORDER  BY e_min.job, e_min.ename, e_max.ename
```

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 9.4: Interessante Sortierung (4)

Lösung:

- Es wird ein CASE-Ausdruck verwendet, um den Sortierwert zu berechnen:

```
SELECT d.dname AS "Abteilung",
       e.job AS "Beruf",
       e.ename AS "Angestellter"
FROM   emp e, dept d
WHERE  e.deptno = d.deptno
ORDER  BY d.dname,
          CASE e.job WHEN 'PRESIDENT' THEN 1
                     WHEN 'MANAGER' THEN 2
                     WHEN 'CLERK' THEN 3
                     ELSE 4 END,
          e.ename
```

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

Aufgabe 9.5: Statistik (3)

Lösung:

- Man braucht `EXISTS`, um zu prüfen, dass der Angestellte ein Vorgesetzter ist:

```
SELECT COUNT(*) anz,
       MIN(sal) min_sal, MAX(sal) max_sal,
       ROUND(AVG(sal)) avg_sal,
       COUNT(DISTINCT job) anz_jobs
FROM   emp e
WHERE  EXISTS(SELECT * FROM emp u
              WHERE  u.mgr = e.empno)
```

Wenn man einen Join ohne `EXISTS` bildet, zählt man jeden Vorgesetzten einmal pro Untergebenem. Das wäre natürlich falsch. Bei der Anzahl würde es sich mit `COUNT(DISTINCT e.empno)` lösen lassen, aber nicht beim durchschnittlichen Gehalt.

Inhalt

1 Organisatorisches

2 Präsenzaufg. 9

3 GROUP BY

4 Hausaufgabe 10

5 Hausaufgabe 9

6 Präsenzaufg. 10

Präsenzaufgabe: Aggregationsanfrage

- Schreiben Sie eine SQL-Anfrage zur Beantwortung der folgenden Frage.
- Von welchen Komponisten stehen mindestens 10 Musikstücke in der Datenbank?

name	vorname	count
Bach	Johann Sebastian	10
Händel	Georg Friedrich	25
Monteverdi	Claudio	10
Prokofiev	Serge	13
Telemann	Georg Philipp	15

- KOMPONIST(KNR, NAME, VORNAME°, GEBOREN°, GESTORBEN°)
- STUECK(SNR, KNR°→KOMPONIST, TITEL, TONART°, OPUS°)