Einführung in Datenbanken

Übung 6: Logik in SQL, Join (Verbund)

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 4a

- Geben Sie alle Presidenten (Name und Geburtsjahr) aus, die mindestens 8 Jahre Präsident waren.

  Tipp: Die Anzahl Jahre als Präsident steht in der Spalte `years_serv` der Tabelle `president`.

- `president(`<u>`pres_name`</u>`, birth_year, years_serv, death_age, party, state_born`$\rightarrow$`state)`

- Lösung:

```
select pres_name, birth_year
from   president
where  years_serv >= 8
```

pres_name	birth_year
Jefferson T	1743
⋮	⋮

Hausaufgabe 4b

- Geben Sie alle Staaten aus, deren erster Buchstabe "C" ist.
- `state(state_name, admin_entered, year_entered)`
- Lösung:

```
select state_name
from   state
where  state_name like 'C%'
```

state_name
Connecticut
California
Colorado

- `WHERE SUBSTRING(state_name from 1 for 1) = 'C'`

  Standard-konform, aber nicht besonders portabel.

Hausaufgabe 4c

- Welcher Präsident (`pres_name`) hatte ein Hobby, das irgendwo `ball` in beliebiger Groß- und Kleinschreibung in der Bezeichnung hat (d.h. „`ball`“ oder „`Ball`“ oder noch andere Varianten sind eine Teilzeichenkette des Hobbys)? Nutzen Sie die Tabelle `pres_hobby`.
- `pres_hobby(pres_name→president, hobby)`
- Lösung:

```
select pres_name
from   pres_hobby
where  upper(hobby) like '%BALL%'
```

pres_name
Hoover H C
Kennedy J F

Hausaufgabe 4c: Alternativen

- Die in den Daten vorkommenden Hobbies sind:
  - „`Medicine Ball`“ (Hoover)
  - „`Touch Football`“ (Kennedy)
- Funktioniert im Beispiel-Zustand, gibt aber Punktabzug:

```
SELECT pres_name
FROM   pres_hobby
WHERE  hobby LIKE '%ball%' OR hobby LIKE '%Ball%'
```

  Es ist zwar plausibel, dass das tatsächlich alle vorkommenden Schreibweisen sind, aber die Aufgabe forderte „in beliebiger Groß- und Kleinschreibung“.

- Funktioniert im Beispiel-Zustand, gibt aber Punktabzug:

```
SELECT pres_name
FROM   pres_hobby
WHERE  UPPER(hobby) LIKE UPPER('%ball')
```

  Hier werden nur Hobbies betrachtet, die auf „`ball`“ enden.

Hausaufgabe 4d

- Geben Sie alle Präsidenten aus, deren Alter bei der Eheschließung um mehr als 10 Jahre größer war, als das ihrer Braut („spouse“).

  Nutzen Sie die Tabelle `pres_marriage`. Geben Sie den Namen der Ehepartnerin und die beiden Alter mit aus.

- `pres_marriage(`<u>`pres_name`</u>`→president, `<u>`spouse_name`</u>`, pr_age, sp_age, nr_children, mar_year)`

- Lösung:

```
select pres_name, spouse_name, pr_age, sp_age
from   pres_marriage
where  pr_age > sp_age + 10
```

pres_name	spouse_name	pr_age	sp_age
Madison J	Todd D D P	43	26
⋮	⋮	⋮	⋮

Hausaufgabe 4d: Alternativen

- Alle üblichen mathematischen Umformungen der Bedingung sind möglich:

```
SELECT pres_name, spouse_name, pr_age, sp_age
FROM   pres_marriage
WHERE  pr_age - sp_age > 10
```

- Ebenso:

```
WHERE  sp_age < pres_age - 10
```

- Dagegen führen
  - `pr_age >= sp_age + 10`
  - `pr_age - sp_age >= 10`

  zum Punktabzug.
  In der Aufgabe stand „um mehr als 10 Jahre“.

Hausaufgabe 4e

- Geben Sie alle Bundesstaaten aus, deren Namen aus genau vier Zeichen besteht.

  Nutzen Sie die Tabelle `state`.

- `state(state_name, admin_entered, year_entered)`

- Lösung:

```
select state_name
from   state
where  length(state_name) = 4
```

state_name
Ohio
Iowa
Utah

Hausaufgabe 4e: Bonusaufgabe

- Sie bekommen einen Extrapunkt, wenn Sie erklären können, warum die Bedingung `state_name like '____'` nicht funktioniert.

  Schreiben Sie diese Erklärung bitte als SQL-Kommentar dazu (falls Sie ein Semikolon schreiben, bitte erst nach dem Kommentar).

- `like '___'` funktioniert nicht, da `state_name` als `character(17)` definiert ist, und somit jeder Staatname die Länge 17 hat, mit Leerzeichen aufgefüllt.

- `LIKE` verwendet die „Non-Padded“ Vergleichssemantik, d.h. Leerzeichen am Ende der Zeichenkette sind signifikant.

- Folgende Bedingung würde funktionieren:

```
WHERE  TRIM(state_name) LIKE '____'
```

Inhalt

Präsenzaufgabe: System-Katalog (1)

- Der Systemkatalog eines DBMS enthält „Metadaten“, also „Daten über Daten“, und insbesondere das DB-Schema.

  Das „Information Schema“ ist Teil des SQL-Standards.

  PostgreSQL hat daneben noch seinen eigentlichen „nativen“ Systemkatalog.

- In der Tabelle `information_schema.columns` sind alle Spalten von allen Tabellen der Datenbank eingetragen.

  [https://www.postgresql.org/docs/9.3/infoschema-columns.html]

- Schreiben Sie eine Anfrage, die alle Spalten liefert von einem Typ `numeric(p,s)` mit Nachkommastellen ($s \neq 0$) und höchstens 4 Dezimalstellen ($p \leq 4$).

  Spalten: `data_type` („`numeric`“), `numeric_precision` (p), `numeric_scale` (s).

- Geben Sie `table_schema`, `table_name`, `column_name` und den Datentyp aus, z.B. `NUMERIC(4,1)`.

  Die String-Konkatenation `||` kann auch auf Zahlen angewendet werden.

Präsenzaufgabe: System-Katalog (2)

- Erwartetes Ergebnis:

table_schema	table_name	column_name	datentyp
student...	bewertungen	punkte	NUMERIC(4,1)
sakila_public	film	rental_rate	NUMERIC(4,2)
sakila_public	film_list	price	NUMERIC(4,2)
sakila_public	nicer_...	price	NUMERIC(4,2)

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

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

- Anfragen müssen nicht nur mit dem aktuellen Zustand funktionieren, sondern mit beliebigen DB-Zuständen.
- Sie können sich also nicht darauf verlassen, dass alle Ergebnis-Typen `numeric(`p`,`s`)` die Bedingung $p = 4$ erfüllen. Die Bedingung der Aufgabe ist $p \leq 4$.

Präsenzaufgabe: System-Katalog (3)

- Lösung:

```
SELECT table_schema, table_name, column_name,
       'NUMERIC(' || numeric_precision || ',' ||
           numeric_scale || ')' AS DATENTYP
FROM   information_schema.columns
WHERE  data_type = 'numeric'
AND    numeric_scale >= 1
AND    numeric_precision <= 4;
```

- Alternativen:
  - Die Reihenfolge der drei Bedingungen unter `WHERE` ist egal: `AND` ist kommutativ und assoziativ.
  - . . . (Fortsetzung auf nächsten sechs Folien)

Präsenzaufgabe: System-Katalog (4)

- Bedingung der Musterlösung:

```
WHERE data_type = 'numeric'
AND   numeric_scale >= 1
AND   numeric_precision <= 4;
```

- Alternativen, Forts.:
  - Unter der Voraussetzung, dass `numeric_scale` eine ganze Zahl ist (ist es), ist `numeric_scale > 0` äquivalent zu `numeric_scale >= 1`.
  - Unter der Voraussetzung, dass `numeric_scale` eine nicht-negative ganze Zahl ist (auch gegeben), ist auch `numeric_scale <> 0` äquivalent zu `numeric_scale >= 1`.

    Oracle erlaubt z.B. -3, um die Zahl in Tausendern zu speichern (nicht standard-konform, nicht portabel).

Präsenzaufgabe: System-Katalog (5)

- Bedingung der Musterlösung:

```
WHERE data_type = 'numeric'
AND   numeric_scale >= 1
AND   numeric_precision <= 4;
```

- Alternativen, Forts.:
  - Wenn man wüßte, dass `numeric_scale` nur dann nicht Null ist, wenn `data_type = 'numeric'` ist, könnte man die Bedingung `data_type = 'numeric'` weglassen.

    Die Bedingung `numeric_scale >= 1` würde dann die Bedingung `data_type = 'numeric'` logisch implizieren, d.h. `data_type = 'numeric'` wäre immer wahr, wenn `numeric_scale >= 1` wahr ist. Dann wäre es überflüssig, die Bedingung über den Datentyp explizit zu fordern. Einzig anderer Kandidat wohl `DECIMAL(`p`,`s`)`. PostgreSQL übersetzt das automatisch in `NUMERIC(`p`,`s`)`. Andere Systeme eventuell nicht.

Präsenzaufgabe: System-Katalog (6)

Weitere Alternativen:

- Korrekt, Klammern überflüssig, `NOT` interessant:

```
WHERE (data_type = 'numeric')
AND   (numeric_precision <= 4)
AND   NOT (numeric_scale = 1)
```

- Nicht standard-konform (man muss <> schreiben), aber recht portabel:

```
numeric_scale != 0
```

- Warum Mustervergleich, wenn es ein einfacher Gleichheitstest tut?

```
data_type LIKE 'numeric'
```

In einer Abgabe stand das sogar mit „`SIMILAR TO`“.

Präsenzaufgabe: System-Katalog (7)

Alternativen für Ausgabeterme unter SELECT:

- `CONCAT(UPPER(data_type),'(',numeric_precision, ',',numeric_scale,')') AS DATENTYP`
  - `CONCAT` ist nicht im SQL-Standard

    Das Information Schema soll aber gerade ein Standard sein (DBMS-unabhängig). Dann auch Anfrage standard-konform?
  - PostgreSQL versteht `||` und `CONCAT(...)`.
  - MariaDB 5.5.68 (und auch MySQL) verstehen `||` als logischen Operator (Alternative zu `OR`).

    `SELECT 'a' || 'b'` liefert 0 in MariaDB/MySQL. Die Strings werden automatisch im Typ angepasst. Erst mit `show warnings` (`\W`) bekommt man Warnungen. MariaDB/MySQL hat ein Information Schema, konvertiert `numeric` aber zu `decimal`.

Präsenzaufgabe: System-Katalog (8)

- Interessante Formatierung:

```
SELECT table_schema,
       table_name,
       column_name,
       'NUMERIC'
       || '('
       || numeric_precision
       || ','
       || numeric_scale
       || ')' AS Datentyp
FROM   columns -- siehe nächste Folie
WHERE  data_type = 'numeric'
       AND numeric_precision <= 4
       AND numeric_scale <> 0
```

Präsenzaufgabe: System-Katalog (9)

- In einigen Abgaben wurde „`columns`“ statt „`information_schema.columns`“ geschrieben.
- PostgreSQL hat einen Schema-Suchpfad, den man so setzen könnte, dass auch das `information_schema` durchsucht wird, wenn eine Relation keine explizite Schema-Angabe hat.

  ```
  SET search_path TO information_schema,public;
  ```

  Adminer vergisst das aber gleich wieder (gilt für Sitzung).
- Das ist aber nicht der Default, und man erhält:

  ```
  ERROR: relation "columns" does not exist.
  ```
- Man kann den Suchpfad anzeigen mit:

  ```
  SHOW search_path;
  ```
- Das ist alles ganz PostgreSQL-spezifisch.

Inhalt

Einfache SQL-Anfragen, Stufe 4 (1)

- Einfache SQL-Anfragen, Stufe 4: Mehrere Tabellen („Join“):

  `SELECT` ⟨Wertausdruck⟩, ..., ⟨Wertausdruck⟩
  `FROM` ⟨Tabelle⟩ ⟨Variable⟩, ..., ⟨Tabelle⟩ ⟨Variable⟩
  `WHERE` ⟨Bedingung⟩

  Die Variablen sind „Tupelvariablen“, laufen also über den Zeilen (Tupeln) der zugehörigen Tabelle.

- Die Bedingung enthält normalerweise (`AND`-verknüpft mit dem Rest) eine „Verbundbedingung“:

  ⟨Variable1⟩.⟨Attribut1⟩ = ⟨Variable2⟩.⟨Attribut2⟩

- In den meisten Fällen handelt es sich um Schlüssel und Fremdschlüssel,

  ⟨Variable1⟩.⟨Fremdschlüssel⟩ = ⟨Variable2⟩.⟨Schlüssel⟩

Einfache SQL-Anfragen, Stufe 4 (2)

- Wenn Schlüssel und Fremdschlüssel aus mehreren Spalten bestehen, so müssen sie AND-verknüpft gleichgesetzt werden.
- Beispiel:
  - BEWERTUNGEN(SID→STUDENTEN,
    (ATYP,ANR)→AUFGABEN, PUNKTE)
  - FROM-Klausel:

    ```
    FROM BEWERTUNGEN B, AUFGABEN A
    ```

  - Es gibt also eine Tupelvariable B über BEWERTUNGEN, und eine Tupelvariable A über AUFGABEN.
  - Verbund-Bedingung:

    ```
    WHERE B.ATYP = A.ATYP AND B.ANR = A.ANR
    ```

  - Natürlich kann die WHERE-Bedingung außerdem noch weitere Teilbedingungen enthalten.

Einfache SQL-Anfragen, Stufe 4 (3)

- Wenn man die Join-Bedingung vergisst, erhält man oft viel zu große Antworten.
- Häufig sind auch viele Duplikate dabei.
- Theoretisch geht SQL alle möglichen Variablenbelegungen durch.
  - Wenn man zwei Tabellen unter `FROM` angibt, T_1 mit n Zeilen, und T_2 mit m Zeilen, sind das $n * m$ Variablenbelegungen.
  - Daraus muss man die „interessanten“ Variablenbelegungen über die `WHERE`-Bedingung herausfiltern.
  - Für jede Variablenbelegung, die übrig bleibt, wird eine Ausgabe-Zeile entsprechend der `SELECT`-Klausel erzeugt.

Inhalt

Beispiel-Datenbank

STUDENTEN

SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	...
102	Michael	Grau	NULL
103	Daniel	Sommer	...
104	Iris	Winter	...

BEWERTUNGEN

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

AUFGABEN

ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

Logische Analyse von Anfragen (1)

1. ```
   SELECT VORNAME, NACHNAME
   FROM   STUDENTEN
   WHERE  VORNAME = 'Tim' OR VORNAME = 'Tina'
   ```

2. ```
   SELECT VORNAME, NACHNAME
   FROM   STUDENTEN
   WHERE  NOT(VORNAME <> 'Tim' OR VORNAME <> 'Tina')
   ```

Kreuzen Sie die erste zutreffende Antwort an:

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

B. Bis auf Duplikate die gleiche Antwort.

C. Anfrage 1 liefert immer $\emptyset$ (inkonsistent).

D. Anfrage 2 liefert immer $\emptyset$ (inkonsistent).

E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (2)

1.
```
SELECT S.SID -- Spaltenüberschrift ist SID
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID=B.SID AND B.ATYP='H' AND B.ANR=1
```

2.
```
SELECT SID
FROM   BEWERTUNGEN
WHERE  ANR = 1 AND ATYP = 'H'
```

Kreuzen Sie die erste zutreffende Antwort an:

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

B. Bis auf Duplikate die gleiche Antwort.

C. Anfrage 1 liefert immer $\emptyset$ (inkonsistent).

D. Anfrage 2 liefert immer $\emptyset$ (inkonsistent).

E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (3)

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

2. ```
   SELECT SID
   FROM   BEWERTUNGEN
   WHERE  ANR = 1 AND ATYP = 'H'
   ```

Kreuzen Sie die erste zutreffende Antwort an:

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

B. Bis auf Duplikate die gleiche Antwort.

C. Anfrage 1 liefert immer $\emptyset$ (inkonsistent).

D. Anfrage 2 liefert immer $\emptyset$ (inkonsistent).

E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (4)

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

2.
```
SELECT NACHNAME
FROM   STUDENTEN
WHERE  VORNAME = 'Lisa'
```

Kreuzen Sie die erste zutreffende Antwort an:

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

B. Bis auf Duplikate die gleiche Antwort.

C. Anfrage 1 liefert immer $\emptyset$ (inkonsistent).

D. Anfrage 2 liefert immer $\emptyset$ (inkonsistent).

E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (5)

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

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

Kreuzen Sie die erste zutreffende Antwort an:

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

Logische Analyse von Anfragen (6)

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

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

Kreuzen Sie die erste zutreffende Antwort an:

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

B. Bis auf Duplikate die gleiche Antwort.

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

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

E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (7)

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

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

Kreuzen Sie die erste zutreffende Antwort an:

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

B. Bis auf Duplikate die gleiche Antwort.

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

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

E. Keine der Aussagen trifft zu.

Inhalt

Präsenzaufgabe: System-Katalog (1)

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

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

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

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

- Constraints stehen in `pg_catalog.pg_constraint`.

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

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

    Den Namen dieser Tabelle sollen Sie ausgeben.

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

    Dies soll die Tabelle `studenten` sein.

Präsenzaufgabe: System-Katalog (2)

- Erwartetes Ergebnis:

relname
bewertungen

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

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

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

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

Präsenzaufgabe: System-Katalog (3)

Tipps:

- Wenn Sie

  ```
  SELECT * FROM pg_class
  ```

  eingeben, wird die Spalte `oid` nicht angezeigt.

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

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

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

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

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

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