TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Einführung in die Informatik 2 für Ingenieure (MSE)*

Christoph Anneser (anneser@in.tum.de), Simon Ellmann (ellmann@in.tum.de)
http://db.in.tum.de/teaching/ss24/ei2/

Blatt Nr. 12
Dieses Blatt wird am Montag, den 15.07.2024 besprochen.

Aufgabe 1: SQL – Allquantoren

Finden Sie in SQL alle Studenten, die alle Vorlesungen gehört haben. Sie dürfen *keine* Abzählung verwenden.

Aufgabe 2: Queries in Java

In dieser Aufgabe wollen wir SQL zu Java übersetzen. Dafür laden Sie zunächst bitte das Java-Projekt [Link] von der Vorlesungswebsite herunter und öffnen es in Ihrer IDE.

In der Datei `Queries.java` sollen Sie nun die folgenden SQL Anfragen in Java übersetzen und auf der Datenbank möglichst effizient ausführen. Die erste Query wurde hierbei bereits beispielhaft implementiert. Speichern Sie die Ergebnisse der Query in der übergebenen Liste ab.

1. Geben Sie den Semesterdurchschnitt der Studenten aus.

   ```
   select avg(semester) from studenten;  -- als Beispiel bereits übersetzt
   ```

   Eine Möglichkeit dies in Java zu übersetzen wäre:

   ```
   public static void executeQuery1(Database db, List<Float> results) {
       int semesters_sum = 0;
       for (Student student : db.studenten) semesters_sum += student.Semester;
       results.add((float) semesters_sum / db.studenten.size());
   }
   ```

2. Geben Sie die Studenten aus, die mindestens eine Vorlesung hören.

   ```
   select distinct(studenten.name)
   from studenten, hoeren
   where studenten.matrnr = hoeren.matrnr;
   ```

3. Geben Sie die Besucherzahlen der Vorlesungen aus.

   ```
   select count(h.matrnr)
   from Vorlesungen v left outer join hoeren h on (v.vorlnr = h.vorlnr)
   group by v.vorlnr, v.titel;
   ```

4. Geben Sie den Semesterdurchschnitt der Studenten aus, die mindestens eine Vorlesung bei Sokrates hören.

```sql
select avg(semester*1.0)
from studenten s
where exists
        (select *
        from hoeren h, vorlesungen v, professoren p
        where v.gelesenVon = p.persnr
        and p.name = 'Sokrates'
        and h.matrnr = s.matrnr
        and v.vorlnr = h.vorlnr);
```

Aufgabe 3: SQL als DML

Gegeben sei ein erweitertes Universitätsschema mit den folgenden zusätzlichen Relationen *StudentenGF* und *ProfessorenF*:

$$\begin{aligned} \text{StudentenGF} &: \{[\underline{\text{MatrNr} : \text{integer}}, \text{Name} : \text{varchar}(20), \text{Semester} : \text{integer}, \\ &\quad \text{Geschlecht} : \text{char}, \text{FakName} : \text{varchar}(20)]\} \\ \text{ProfessorenF} &: \{[\underline{\text{PersNr} : \text{integer}}, \text{Name} : \text{varchar}(20), \text{Rang} : \text{char}(2), \\ &\quad \text{Raum} : \text{integer}, \text{FakName} : \text{varchar}(20)]\} \end{aligned}$$

Die erweiterten Tabellen sind bereits auf der Webschnittstelle unter

`http://hyper-db.com/interface.html`

angelegt.

(a) Ermitteln Sie den Männeranteil an den verschiedenen Fakultäten in SQL! Beachten Sie dabei, dass es auch Fakultäten ohne Männer geben kann.

(b) Ermitteln Sie in SQL die Studenten, die alle Vorlesungen ihrer Fakultät hören. Geben Sie zwei Lösungen an, höchstens eine davon darf auf Abzählen basieren.

Aufgabe 4: Outer Join

In der Vorlesung haben wir den **left outer join** kennen gelernt:

```sql
select *
from Studenten s left outer join
hoeren h on s.matrNr = h.matrNr;
```

Ist es möglich eine semantisch equivalente Anfrage zu formulieren ohne einen **left outer join** zu benutzen (selbverständlich ist auch der **right** und **full outer join** verboten).

Aufgabe 5: Fleißige Studenten

Formulieren Sie eine SQL-Anfrage, um die Studenten zu ermitteln, die mehr SWS belegt haben als der Durchschnitt. Berücksichtigen Sie dabei auch Totalverweigerer, die gar keine Vorlesungen hören.

Sichten und Rekursion

Für diese Aufgabe fügen wir eine Spalte (Attribut/Column) zu der Professoren Relation hinzu: doktorElternteilPersNr. Diese gibt die Personennummer der Professorin an bei der eine Professorin ihren Doktortitel erhalten hat. Um eine endlose Kette von Professorinnen zu verhindern erlauben wir in dieser Spalte null Werte.

Erstellen Sie eine Sicht mit folgendem Schema: [persNr, name, generationen]. Die Generation einer Professorin ist durch die Anzahl von bekannten Doktoreltern definiert. Hat eine Professorin zum Beispiel einen null Eintrag in doktorElternteilPersNr, dann ist ihre Generation 0. Sind dagegen 2 Doktoreltern bekannt ist die Generation auch 2.

Das Webinterface beinhaltet diese zusätzliche Spalte nicht, falls Sie dennoch ihre Anfragen testen möchten, können Sie dies in einer lokalen Datenbank tun. Der Folgende Code kann in das die Schemadefinition eingearbeitet weden um die Tabelle mit der zusätzlichen Spalte zu erstellen.

```
CREATE TABLE Professoren
(PersNr          INTEGER PRIMARY KEY,
Name            VARCHAR(30) NOT NULL,
Rang            CHAR(2) CHECK (Rang in ('C2', 'C3', 'C4')),
Raum            INTEGER UNIQUE,
DoktorElternteil INTEGER,
FOREIGN KEY     (DoktorElternteil) REFERENCES Professoren);

INSERT INTO Professoren VALUES
(2125, 'Sokrates', 'C4', 226, null);
INSERT INTO Professoren VALUES
(2134, 'Augustinus', 'C3', 309, 2125);
INSERT INTO Professoren VALUES(2137, 'Kant', 'C4', 007, 2134);
INSERT INTO Professoren VALUES(2126, 'Russel', 'C4', 232, 2137);
INSERT INTO Professoren VALUES(2133, 'Popper', 'C3', 052, 2137);
INSERT INTO Professoren VALUES
(2127, 'Kopernikus', 'C3', 310, null);
INSERT INTO Professoren VALUES(2136, 'Curie', 'C4', 036, 2127);
```