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)
http://db.in.tum.de/teaching/ss21/ei2/

Blatt Nr. 12
Dieses Blatt wird am Montag, den 12. Juli 2021 besprochen.

Aufgabe 1: 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.

Tipp: Merken Sie sich das Vorgehen bei diesem Aufgabentypen für die Klausur.

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;
```

```
public static void executeQuery2(Database db, List<String> results) {
    HashSet<Integer> hoerendeStudenten = new HashSet<>();
    for (Hoeren h : db.hoeren) hoerendeStudenten.add(h.MatrNr);
    for (Student s : db.studenten)
        if (hoerendeStudenten.contains(s.MatrNr))
            results.add(s.Name);
    }
```

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;
```

```
public static void executeQuery3(Database db, List<Query3Result> results) {
    HashMap<Integer, Integer> vorlesungsBesuche = new HashMap<>();
    for (Vorlesung vorlesung : db.vorlesungen)
        vorlesungsBesuche.put(vorlesung.VorlNr, 0);

    for (Hoeren hoeren : db.hoeren)
        vorlesungsBesuche.merge(hoeren.VorlNr, 1, Integer::sum);

    for (Map.Entry<Integer, Integer> entry : vorlesungsBesuche.entrySet())
        results.add(new Query3Result(entry.getKey(), entry.getValue()));
}
```

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

```
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);
```

```
public static void executeQuery4(Database db, List<Float> results) {
    // Finde persnr von Soktates
    Professor sokrates = null;
    for (Professor prof : db.professoren) {
        if (prof.Name.equals("Sokrates")) {
            sokrates = prof;
            break;
        }
    }
    if (sokrates == null) return;

    // Finde alle Vorlesungen von Sokrates
    HashSet<Integer> vorlesungsNummernSokrates = new HashSet<>();
    for (Vorlesung v : db.vorlesungen)
        if (v.gelesenVon == sokrates.PersNr) vorlesungsNummernSokrates.add(v.VorlNr);

    // Finde Matrikelnummern der Studenten, die eine der Vorlesungen von Sokrates hoeren
    HashSet<Integer> matrikelNummernSokratesHoerer = new HashSet<>();
    for (Hoeren h : db.hoeren)
        if (vorlesungsNummernSokrates.contains(h.VorlNr))
            matrikelNummernSokratesHoerer.add(h.MatrNr);

    if (matrikelNummernSokratesHoerer.size() == 0) return;
```

```
    // Bilde Durschnitt der gefundenen Studenten
    int semesterSumSokratesHoerer = 0;
    for (Student student : db.studenten)
        if (matrikelNummernSokratesHoerer.contains(student.MatrNr))
            semesterSumSokratesHoerer += student.Semester;
    float durchschnitt = (float) semesterSumSokratesHoerer /
                                 matrikelNummernSokratesHoerer.size();
    results.add(durchschnitt);
}
```

Aufgabe 2: 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.

Lösung

```
WITH FakTotal AS (
SELECT FakName,COUNT(*) as total
FROM StudentenGF
GROUP BY FakName),
FakMaenner AS (
SELECT FakName,COUNT(*) as maenner
FROM StudentenGF
WHERE geschlecht='M'
GROUP BY FakName)
SELECT FakTotal.FakName,(CASE WHEN maenner IS NULL THEN 0 ELSE maenner END)/(total*1.0)
FROM FakTotal LEFT JOIN FakMaenner
ON FakTotal.FakName=FakMaenner.FakName
```

Wir müssen beachten, dass nicht jede Fakultät Männer beherbergt, weswegen diese Fakultäten (in der Standardausprägung im SQL Interface ist dies für Theologie der Fall) dann aus dem Ergebnis herausfallen würden. Aus diesem Grund verwenden wir einen `LEFT OUTER`

`JOIN` um die Zahl der Männer und die Zahl der Studenten insgesamt zu verbinden, wodurch auch die Theologie Fakultät im Ergebnis enthalten ist, auch wenn es keine Männer gibt.

Das `CASE`-Konstrukt dient in der oberen Anfrage dazu, den NULL Wert, die durch den Left Join für die Anzahl der Männer entstehen, wenn es keine Männer gibt, durch die Zahl 0 zu ersetzen. Alternativ ist dies möglich, indem man `COALESCE(maenner,0)/(total*1.0)` verwendet.

Alternativ können wir das **case**-Konstrukt verwenden, um die Anzahl der Männer an den jeweiligen Fakultäten zu ermitteln. Den Männeranteil erhalten wir dann, indem wir die Anzahl der Männer durch die Gesamtanzahl der Studenten an der Fakultät teilen.

```
select FakName,
(sum(case when Geschlecht = 'M' then 1 else 0 end)) /
cast (count(*) as float)
from StudentenGF
group by FakName
```

(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.

Lösung

```
select s.*
from StudentenGF s
where not exists (select *
from Vorlesungen v, ProfessorenF p
where v.gelesenVon = p.PersNr
and p.FakName = s.FakName
and not exists
(select *
from hoeren h
where h.VorlNr = v.VorlNr
and h.MatrNr = s.MatrNr));
```

Wir fordern hier, dass es keine Vorlesung an der Fakultät des Studenten (d.h. von einem Professor der gleichen Fakultät gelesen) geben darf, die vom Studenten nicht gehört wird.

Alternativ:

```
select * from studentengf s
where
(select count(*)
from vorlesungen v,
     professorenf p
where v.gelesenvon = p.persnr
  and p.fakname = s.fakname)
=
(select count(*)
from hoeren h, vorlesungen v, professorenf p
where h.matrnr = s.matrnr
```

```
and h.vorlnr = v.vorlnr
and p.persnr = v.gelesenvon
and p.fakname = s.fakname)
```

Aufgabe 3: Outer Join

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

```
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).

Lösung

Ja:

```
select *
from Studenten s, hoeren h
where s.matrnr = h.matrnr
union
select s.*, null, null
from Studenten s
where not exists (select *
from hoeren h
where s.matrnr = h.matrnr)
```

Aufgabe 4: 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.

Lösung

Folgende SQL-Anfrage ermittelt die fleißigen Studenten:

```
SELECT s.*
FROM Studenten s
WHERE s.MatrNr IN
(SELECT h.MatrNr
FROM hoeren h JOIN Vorlesungen v
ON h.VorlNr = v.VorlNr
GROUP BY h.MatrNr
HAVING SUM(SWS) >
(SELECT SUM(cast(SWS as decimal(5,2)))
```

```
/COUNT(DISTINCT(s2.MatrNr))
FROM Studenten s2 LEFT OUTER JOIN hoeren h2
ON h2.MatrNr = s2.MatrNr
LEFT OUTER JOIN Vorlesungen v2
ON v2.VorlNr = h2.VorlNr));
```

Durch die Verwendung temporärer Sichten mit **with** wird die Anfrage übersichtlicher:

```
WITH GesamtSWS AS
(SELECT SUM(cast(SWS as decimal(5,2))) AS AnzSWS
FROM hoeren h2, Vorlesungen v2
WHERE v2.VorlNr = h2.VorlNr),
GesamtStudenten AS
(SELECT count(MatrNr) AS AnzStudenten
FROM Studenten)

SELECT s.*
FROM Studenten s
WHERE s.MatrNr IN (SELECT h.MatrNr
FROM hoeren h JOIN Vorlesungen v ON h.VorlNr = v.VorlNr
GROUP BY h.MatrNr
HAVING SUM(SWS) > (SELECT AnzSWS/AnzStudenten
FROM GesamtSWS, GesamtStudenten));
```

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);
```

Lösung

```
with recursive Baum(persNr, name, parent, len) as (
( select persNr, name, doktorelternteil, 0 from Professoren )
union
( select b.persNr, b.name, p.doktorelternteil, b.len+1
from Professoren p, Baum b where p.persNr = b.parent)
)

select b.persNr, b.name, b.len as generation
from Baum b
where b.len = (select max(t.len)
from Baum t
where t.persNr = b.persNr);
```