TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Dr. Thomas Neumann

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS19/20

Christoph Anneser, Moritz Sichert, Lukas Vogel (gdb@in.tum.de)
https://db.in.tum.de/teaching/ws1920/grundlagen/

Blatt Nr. 06

Tool zum Üben von SQL-Anfragen: https://hyper-db.com/interface.html.

Hausaufgabe 1

Diese Aufgabe ist die dritte in einer Reihe von Aufgaben, in denen Sie lernen werden, einen gegebenen Sachverhalt zu analysieren, geeignete Modelle zu entwerfen, diese in ein Datenbankschema zu überführen, das Schema in einer Datenbank aufzusetzen und mit Daten zu befüllen. Sie werden ebenfalls lernen, wie Sie die Datenbank für bestimmte Anfragen optimieren können.

Gehen Sie in dieser Aufgabe von dem von Ihnen verfeinerten relationalen Schema des Onlineshops aus der letzten Aufgabe aus. Sollten Sie nicht im Besitz einer zufriedenstellenden Lösung sein, können Sie auch die am Freitag Nachmittag veröffentlichte Beispiellösung als Ausgangspunkt verwenden.

a) Geben Sie DDL-Anweisungen an, die die für den Onlineshop notwendigen Relationen erstellen.

b) Ergänzen Sie Ihre DDL-Anweisungen aus Teilaufgabe a) um die Angabe der Primär- und Fremdschlüsseln.

Zusätzlich zu den in Blatt 4 vorgestellten Anforderungen muss das Datenbankschema nun auch DSGVO-konform werden. Wir nehmen vereinfachend nur folgende Forderung an: *Kunden des Onlineshops müssen die Löschung all ihrer personenbezogenen Daten verlangen können.* Achten Sie außerdem darauf, dass die für die Buchhaltung notwendigen Informationen erhalten bleiben müssen (z.B. aus steuerrechtlichen Gründen).

c) Erweitern Sie Ihre DDL-Anweisungen so, dass die oben genannten Anforderungen erfüllt werden. Das bedeutet also, dass Kunden gelöscht werden können, ihre Bestellungen aber *anonymisiert* im System gespeichert bleiben.

d) Fügen Sie Ihrer Datenbank nun mittels DML-Anweisungen die minimale Anzahl an Tupeln hinzu, sodass es einen Kunden und eine ihm zugeordnete Bestellung gibt. Dabei muss sichergestellt sein, dass keine referentiellen Integritätsbedingungen verletzt werden. Löschen Sie dann den Kunden aus der Datenbank und überprüfen Sie, ob Teilaufgabe c) erfolgreich in der Datenbank umgesetzt ist.

e) *Zusatzaufgabe*: Erstellen Sie ein SQL-Skript, welches alle Tabellen in einer neuen Datenbank anlegt. Sollte die Datenbank bereits Tabellen mit gleichem Namen enthalten, so sollen diese zunächst *restlos* gelöscht werden. Testen Sie das Skript in Ihrer aus dem letzten Blatt aufgesetzten (lokalen) Datenbank.

Lösung:

```
-- Beginn der Zusatzaufgabe e)
DROP TABLE IF EXISTS Bestellpositionen CASCADE;
DROP TABLE IF EXISTS Bestellungen CASCADE;
```

```
DROP TABLE IF EXISTS liefertNach CASCADE;
DROP TABLE IF EXISTS bereitstellen CASCADE;
DROP TABLE IF EXISTS Artikel CASCADE;
DROP TABLE IF EXISTS Kunden CASCADE;
DROP TABLE IF EXISTS Lieferanten CASCADE;
DROP TABLE IF EXISTS Laender CASCADE;
DROP TABLE IF EXISTS Kontinente CASCADE;
-- Ende der Zusatzaufgabe

CREATE TABLE Kontinente (
    KontinentenNr INTEGER PRIMARY KEY,
    Name TEXT NOT NULL
);

CREATE TABLE Laender (
    LaenderCode INTEGER PRIMARY KEY,
    Name TEXT NOT NULL,
    KontinentenNr INTEGER REFERENCES Kontinente
);

CREATE TABLE Lieferanten (
    LieferantenNr INTEGER PRIMARY KEY,
    Name TEXT,
    Adresse TEXT,
    LaenderCode INTEGER REFERENCES Laender
);

CREATE TABLE liefertNach (
    LieferantenNr INTEGER REFERENCES Lieferanten,
    LaenderCode INTEGER REFERENCES Laender,
    PRIMARY KEY (LieferantenNr, LaenderCode)
);

CREATE TABLE Kunden (
    KundenNr INTEGER PRIMARY KEY,
    Name TEXT,
    Adresse TEXT,
    LaenderCode INTEGER REFERENCES Laender,
    Geburtsdatum DATE,
    Geschlecht CHAR(1)
);

CREATE TABLE Artikel (
    ArtikelNr INTEGER PRIMARY KEY,
    Name TEXT,
    Marke TEXT,
    Typ TEXT,
    UVP MONEY
```

```
);

CREATE TABLE bereitstellen (
    ArtikelNr INTEGER REFERENCES Artikel,
    LieferantenNr INTEGER REFERENCES Lieferanten,
    VerfuegbareAnzahl INTEGER,
    Lieferpreis MONEY,
    PRIMARY KEY (ArtikelNr, LieferantenNr)
);

CREATE TABLE Bestellungen (
    BestellNr INTEGER PRIMARY KEY,
    -- Aufgabe c)
    KundenNr INTEGER REFERENCES Kunden ON DELETE SET NULL,
    Gesamtstatus TEXT,
    Gesamtpreis MONEY,
    Datum DATE
);

CREATE TABLE Bestellpositionen (
    BestellNr INTEGER REFERENCES Bestellungen,
    ArtikelNr INTEGER REFERENCES Artikel,
    LieferantenNr INTEGER REFERENCES Lieferanten,
    Position INTEGER,
    Anzahl INTEGER,
    Preis MONEY,
    Steuern MONEY,
    Status TEXT,
    FOREIGN KEY (ArtikelNr, LieferantenNr) REFERENCES bereitstellen,
    PRIMARY KEY(Position, BestellNr)
);
```

d) Um dies sinnvoll zu testen, müssen wir einen neuen Eintrag in der Kundentabelle hinzufügen sowie in Bestellungen. Da Kunden die Tabelle Laender und diese wiederum Kontinente referenziert, müssen wir auch in diese neue Tupel hinzufügen.

```
INSERT INTO Kontinente VALUES (0, 'Europa');
INSERT INTO Laender VALUES (0, 'Deutschland', 0);
INSERT INTO Kunden VALUES (0, 'Max␣Muster', 'Test␣Str.␣39', 0, '04/10/1995', 'm');
INSERT INTO Bestellungen VALUES (0, 0, 'versandt', 99.99, '20/11/2019');

DELETE FROM Kunden WHERE KundenNr = 0;
```

Hausaufgabe 2

„Bekanntheitsgrad“: Formulieren Sie eine SQL-Anfrage, um den Bekanntheitsgrad von Studenten zu ermitteln. Gehen Sie dabei davon aus, dass Studenten sich aus gemeinsam besuchten Vorlesungen kennen. Sortieren Sie das Ergebnis absteigend nach Bekanntheitsgrad!

Lösung:

Zunächst definieren wir eine View, die für jeden Studenten alle seine Bekannten auflistet. Anschließend müssen wir diese Bekannten zählen, um den Bekanntheitsgrad der Studenten zu ermitteln. Damit Studenten ohne Bekannte auch im Ergebnis stehen, verwenden wir einen äußeren Join.

```
with Bekannte as (
  select distinct h1.MatrNr as Student, h2.MatrNr as Bekannter
  from hoeren h1, hoeren h2
  where h1.VorlNr = h2.VorlNr
    and h2.MatrNr <> h1.MatrNr
)
select s.MatrNr, s.Name, count(b.Bekannter) as AnzBekannter
from Studenten s left outer join Bekannte b on (s.MatrNr = b.Student)
group by s.MatrNr, s.Name
order by AnzBekannter desc
```

Ohne View sieht die Anfrage entsprechend komplexer aus:

```
select s.MatrNr, s.Name, count(b.Bekannter) as AnzBekannter
from Studenten s left outer join
  (select distinct h1.MatrNr as Student, h2.MatrNr as Bekannter
   from hoeren h1, hoeren h2
   where h1.VorlNr = h2.VorlNr
     and h2.MatrNr <> h1.MatrNr
   ) b on (s.MatrNr = b.Student)
group by s.MatrNr, s.Name
order by AnzBekannter desc
```

Hausaufgabe 3

Gegeben sei die Relation `Fahrplan`, die strukturell dem folgenden Beispiel gleicht:

Von	Nach	Linie	Abfahrt	Ankunft
Garching, Forschungszentrum	Garching	U6	09:06	09:09
Garching	Garching-Hochbrück	U6	09:09	09:11
Garching-Hochbrück	Fröttmaning	U6	09:11	09:15
...	...			
Fröttmaning	Garching-Hochbrück	U6	09:00	09:04
Garching-Hochbrück	Garching	U6	09:04	09:06
Garching	Garching, Forschungszentrum	U6	09:06	09:09
...	...			
Garching, Forschungszentrum	Technische Universität	690	17:56	17:57

Formulieren Sie die folgenden Anfragen auf dieser Relation in SQL. Sie können dabei den Wert `CURRENT_TIME` für die aktuelle Zeit verwenden. Um einen Zeitwert zu einer Zahl umzuwandeln, verwenden Sie folgende Funktion:

```
EXTRACT(EPOCH FROM x - '00:00:00'::TIME)
```

Diese gibt die Anzahl der Sekunden zurück, die zwischen dem Zeitwert `x` und Mitternacht liegen. Mit dieser Funktion kann auch die Anzahl der Sekunden zwischen zwei beliebigen Zeitwerten berechnet werden.

Fallunterscheidungen können Sie mit dem `case`-Konstrukt ausdrücken:

```
select
    case
        when semester=1 then 'Ersti'
        when semester<=6 then 'Bachelor'
        else 'Master'
    end
from
    Studenten
```

a) Geben Sie alle Einträge des Fahrplans aus, deren Abfahrtshaltestelle das Wort „Garching“ enthält.

b) Geben Sie alle Einträge des Fahrplans mit dem zusätzlichen Attribut „Verkehrsmittel“ aus. Alle Linien, die mit „U“ anfangen, haben das Verkehrsmittel „U-Bahn“, alle die mit „S“ anfangen „S-Bahn“, und alle restlichen „Bus/Tram“.

c) Finden Sie alle Abfahrten ab „Garching, Forschungszentrum“, die Sie heute noch erreichen können. Sortieren Sie das Ergebnis aufsteigend nach Abfahrtszeit.

d) Finden Sie alle Fahrten zwischen zwei Haltestellen (nicht-transitiv), die mindestens drei und höchstens fünf Minuten dauern.

   Bonus: Berücksichtigen Sie, dass Fahrten über Mitternacht möglich sind, z.B. Abfahrt um 23:58 Uhr und Ankunft um 00:02 Uhr.

Laden Sie zum Testen entweder die SQL-Datei von der Übungswebseite in Ihr lokal installiertes Datenbanksystem oder verwenden Sie die Webschnittstelle.

Lösung:

a) Geben Sie alle Einträge des Fahrplans aus, deren Abfahrtshaltestelle das Wort „Garching“ enthält.

```
select * from fahrplan where von like '%Garching%'
```

b) Geben Sie alle Einträge des Fahrplans mit dem zusätzlichen Attribut „Verkehrsmittel“ aus. Alle Linien, die mit „U“ anfangen, haben das Verkehrsmittel „U-Bahn“, alle die mit „S“ anfangen „S-Bahn“, und alle restlichen „Bus/Tram“.

```
select
    *,
    case
        when linie like 'U%' then 'U-Bahn'
        when linie like 'S%' then 'S-Bahn'
        else 'Bus/Tram'
    end as Verkehrsmittel
from fahrplan
```

c) Finden Sie alle Abfahrten ab „Garching, Forschungszentrum“, die Sie heute noch erreichen können. Sortieren Sie das Ergebnis aufsteigend nach Abfahrtszeit.

```
select *
from fahrplan
where
    abfahrt > current_time and
    von = 'Garching,␣Forschungszentrum'
order by abfahrt
```

d) Finden Sie alle Fahrten zwischen zwei Haltestellen (nicht-rekursiv), die mindestens drei und höchstens fünf Minuten dauern.

Wenn Fahrten, die vor 00:00 Uhr abfahren und danach ankommen, ignoriert werden, kann die `extract` direkt wie beschrieben verwendet werden:

```
select *
from fahrplan
where extract(epoch from ankunft - abfahrt) between 3*60 and 5*60
```

Ansonsten muss bei Fahrten über Mitternacht die Differenz zwischen Abfahrt und Ankunft korrigiert werden:

```
with fahrplan_dauer as (
    select
        *,
        case
            when ankunft < abfahrt
            then 60 * 60 * 24 - extract(epoch from abfahrt - ankunft)
            else extract(epoch from ankunft - abfahrt)
        end as dauer_in_sek
    from fahrplan
)
select * from fahrplan_dauer where dauer_in_sek between 3*60 and 5*60
```

Hausaufgabe 4

Formulieren Sie die folgende Anfrage auf dem bekannten Unischema in SQL: Ermitteln Sie für jede Vorlesung, wie viele Studenten diese vorgezogen haben. Ein Student hat eine Vorlesung vorgezogen, wenn er in einem früheren Semester ist als der „Modus“ der Semester der Hörer dieser Vorlesung. Der Modus ist definiert als der Wert, der am häufigsten vorkommt – für diese Anfrage also das Semester, in dem die meisten Hörer dieser Vorlesung sind. Falls es mehrere Semester dieser Art gibt, soll nur das niedrigste zählen.

Beachten Sie, dass auch Vorlesungen ohne Hörer, sowie Vorlesungen deren Hörer alle im gleichen Semester sind, ausgegeben werden sollen.

Geben Sie für jede Vorlesung die Vorlesungsnummer, den Titel und die Anzahl der „Vorzieher“ aus.

Lösung:

Die Aufgabe lässt sich am leichtesten lösen, wenn man sie in mehrere Teile aufbricht:

Zunächst erstellen wir eine Anfrage, welche für jede Vorlesung aufschlüsselt, von wie vielen Stundenten sie pro Semester gehört wird:

```
with vorl_semester_anz as (
    select h.vorlnr, s.semester, count(*) as anzahl
    from hoeren h, Studenten s
    where h.matrnr = s.matrnr
    group by h.vorlnr, s.semester
)
```

Mithilfe dieser Sicht können wir nun für jede Vorlesung den Modus der Hörersemester bestimmen. Der Modus ist dasjenige Semester, dem die meisten Anhörer angehören. Unter diesen Einträgen könnten auch Duplikate sein. Wenn eine Vorlesung z.B. gleich oft von Erst- und Drittsemestern gehört wird, wollen wir sie dem ersten Semester zuordnen. Mit einem einfachen *group by* finden wir das Minimum.

```
with vorl_modus as (
    select v1.vorlnr, min(v1.semester) as modus
    from vorl_semester_anz v1
    where v1.anzahl = (
        select max(v2.anzahl)
        from vorl_semester_anz v2
        where v1.vorlnr = v2.vorlnr
    )
    group by v1.vorlnr
)
```

Nun müssen wir nur noch für jedes dieser Vorlesung-Semester-Paare alle diejenigen Studenten finden und aufsummieren, welche die Vorlesung hören und in einem niedrigeren Semester als der Modus sind. Da wir auch Vorlesungen ohne Hörer ausgeben wollen, müssen wir den *left outer join* verwenden. Wichtig: Die Bedingung `s.semester < vm.semester` muss als Bedingung des outer joins angegeben werden, und *nicht* in der where-Klausel, da ansonsten alle Vorlesungen ohne Studenten aus niedrigeren Semestern wieder herausgefiltert werden würden.

```
select v.vorlnr, v.titel, count(s.matrnr) as anzahl
from
    vorlesungen v left outer join
    vorl_modus vm on v.vorlnr = vm.vorlnr left outer join
    hoeren h on h.vorlnr = v.vorlnr left outer join
    studenten s on s.matrnr = h.matrnr and s.semester < vm.modus
group by v.vorlnr, v.titel
```

Insgesamt ergibt sich also beispielsweise folgende Anfrage:

```
with vorl_semester_anz as (
    select h.vorlnr, s.semester, count(*) as anzahl
    from hoeren h, Studenten s
    where h.matrnr = s.matrnr
    group by h.vorlnr, s.semester
), vorl_modus as (
    select v1.vorlnr, min(v1.semester) as modus
    from vorl_semester_anz v1
    where v1.anzahl = (
```

```
        select max(v2.anzahl)
        from vorl_semester_anz v2
        where v1.vorlnr = v2.vorlnr
    )
    group by v1.vorlnr
)

select v.vorlnr, v.titel, count(s.matrnr) as anzahl
from
    vorlesungen v left outer join
    vorl_modus vm on v.vorlnr = vm.vorlnr left outer join
    hoeren h on h.vorlnr = v.vorlnr left outer join
    studenten s on s.matrnr = h.matrnr and s.semester < vm.modus
group by v.vorlnr, v.titel
```