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

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS21/22

Michael Jungmair, Josef Schmeißer, Moritz Sichert, Lukas Vogel (gdb@in.tum.de)
https://db.in.tum.de/teaching/ws2122/grundlagen/

Blatt Nr. 06

Hausaufgabe 1

Klausuraufgabe aus der Zweitklausur WiSe 2019/20.

Hinweis: Diese Aufgabe wurde erstellt, bevor eine Impfung gegen COVID-19 möglich war. Deswegen werden hier ausschließlich Tests und persönliche Kontakte betrachtet.

Verwenden Sie für diese Aufgabe das **Infektionsschema**. Sie finden die Schemadefinition und eine Beispielausprägung auf der letzten Seite.

In dieser Aufgabe sollen Sie ermitteln, bei welchen ungetesteten Personen ein Virustest **unnötig** ist. Ein Test für Person X ist dann unnötig, wenn:

- X noch nicht getestet wurde und
- in **allen** sozialen Gruppen, in denen X Mitglied ist, keine bestätigte Infektion vorliegt, also alle Gruppenmitglieder entweder ungetestet sind oder negativ getestet wurden.

Beachten Sie, dass eine Person auch Mitglied mehrerer Gruppen oder keiner Gruppe sein kann. Geben Sie `PersonId` und `Name` aus.

Hier finden Sie das erwartete Ergebnis für die Beispielausprägung. Ihre Anfrage muss natürlich auch dann funktionieren, wenn die Ausprägung der Relationen anders ist als die Beispielausprägung.

PersonId	Name
63875	Markus

Lösung:

```
select personid, name
from Person p
where not exists (
    select *
    from Virentest v
    where v.PersonId = p.PersonId
)
and not exists (
    select *
    from MitgliedIn m
    where m.PersonId = p.PersonId
    and exists (
        select *
        from MitgliedIn m2, Virentest v
        where m2.GruppeId = m.GruppeId
        and v.PersonId = m2.PersonId
        and v.Testergebnis = 'positiv'
    )
)
```

Hausaufgabe 2

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
```

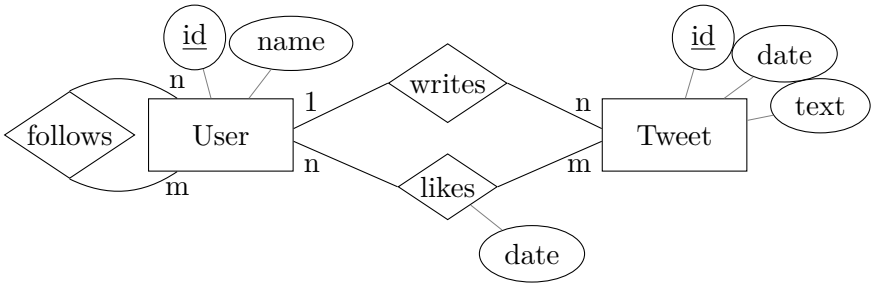
Insgesamt ergibt sich also beispielsweise folgende Anfrage:

```
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
```

Hausaufgabe 3

Gegeben sei folgendes ER-Diagramm, das User, deren Tweets, Likes und Follows modelliert, und das dazugehörige relationale Schema:



$$\begin{aligned}
\text{User}: &\ \{[\underline{\text{id}}, \text{name}]\} \\
\text{Tweet}: &\ \{[\underline{\text{id}}, \text{user_id}, \text{date}, \text{text}]\} \\
\text{follows}: &\ \{[\underline{\text{follower_id}, \text{follows_id}}]\} \\
\text{likes}: &\ \{[\underline{\text{user_id}, \text{tweet_id}}, \text{date}]\}
\end{aligned}$$

a) Geben Sie SQL-Statements zum Erzeugen der Relationen an. Überlegen Sie sich dazu sinnvolle Typen für die Attribute. Verwenden Sie Angaben zu `NULL` und Schlüsseln (`primary key`, `unique`).

b) Ergänzen Sie die SQL-Statements mit referentiellen Integritätsbedingungen. Es soll sichergestellt werden, dass wenn ein User gelöscht wird, auch alle seine Follows, Follower und Likes gelöscht werden. Seine Tweets sollen aber erhalten bleiben, indem die user_id seiner Tweets auf `NULL` gesetzt wird. Wenn ein Tweet gelöscht wird, sollen ebenfalls dessen Likes gelöscht werden.

c) Fügen Sie referenzielle Integritätsbedingungen hinzu, die folgende Eigenschaften garantieren:

- Wenn die user_id eines Tweets `NULL` ist, muss der Text des Tweets „removed“ lauten
- Das Datum eines Likes darf nicht vor dem Datum des Tweets liegen.

Lösung:

a) Geben Sie SQL-Statements zum Erzeugen der Relationen an. Überlegen Sie sich dazu sinnvolle Typen für die Attribute. Verwenden Sie Angaben zu `NULL` und Schlüsseln (`primary key`, `unique`).

Da „User“, „date“ und „text“ von manchen Datenbanken nicht als Namen für Relationen oder Attribute erlaubt sind, sind die betroffenen Namen hier geändert.

Für alle id-Attribute wird der Typ `integer` verwendet. Falls 2^{32} IDs nicht ausreichen sollten, kann stattdessen auch `bigint` verwendet werden. Die Attribute „name“ und „tweet_text“ haben beide den Typen `varchar`, da es sich um einen kurzen Text variabler Länge handelt. Die Datumsattribute haben hier den Typen `timestamp` der ein Datum mit Zeitangabe darstellt statt dem in der Vorlesung vorgestellten Typen `date`, mit dem nur Tage dargestellt werden können.

Generell sollten Attribute so selten wie möglich `NULL` sein, weswegen alle Attribute auf `not null` gesetzt werden. Nur das Attribut „user_id“ der Relation „Tweet“ kann potentiell `NULL` sein, was aber erst in Aufgabe b) verlangt wird.

Die Primärschlüssel sind in dem relationalen Schema schon unterstrichen und müssen in den SQL-Statements beachtet werden. Bei Primärschlüsseln mit nur einem Attribut, kann `primary key` direkt an das Attribute angehängt werden, ansonsten muss der Schlüssel in einer neuen Zeile aufgeführt werden. Zusätzlich sollten Namen eindeutig sein, weswegen das Attribute „name“ den Zusatz `unique` erhält.

```
create table Twitter_User (
   id integer not null primary key,
   name varchar(50) not null unique
);
create table Tweet (
   id integer not null primary key,
   user_id integer null references Twitter_User,
   tweet_date timestamp not null,
   tweet_text varchar(500) not null
);
create table follows (
```

```
    follower_id integer not null references Twitter_User,
    follows_id integer not null references Twitter_User,
    primary key (follower_id, follows_id)
);
create table likes (
    user_id integer not null references Twitter_User,
    tweet_id integer not null references Tweet,
    like_date timestamp not null,
    primary key (user_id, tweet_id)
);
```

b) Ergänzen Sie die SQL-Statements mit referentiellen Integritätsbedingungen. Es soll sichergestellt werden, dass wenn ein User gelöscht wird, auch alle seine Follows, Follower und Likes gelöscht werden. Seine Tweets sollen aber erhalten bleiben, indem die user_id seiner Tweets auf `NULL` gesetzt wird. Wenn ein Tweet gelöscht wird, sollen ebenfalls dessen Likes gelöscht werden.

An allen Stellen, wo User oder Tweets mit `references` referenziert werden, muss entweder `on delete cascade` oder `on delete set null` hinzugefügt werden.

```
create table Twitter_User (
    id integer not null primary key,
    name varchar(50) not null unique
);
create table Tweet (
    id integer not null primary key,
    user_id integer null references Twitter_User on delete set null,
    tweet_date timestamp not null,
    tweet_text varchar(500) not null
);
create table follows (
    follower_id integer not null references Twitter_User on delete cascade,
    follows_id integer not null references Twitter_User on delete cascade,
    primary key (follower_id, follows_id)
);
create table likes (
    user_id integer not null references Twitter_User on delete cascade,
    tweet_id integer not null references Tweet on delete cascade,
    like_date timestamp not null,
    primary key (user_id, tweet_id)
);
```

c) Fügen Sie referenzielle Integritätsbedingungen hinzu, die folgende Eigenschaften garantieren:

- Wenn die user_id eines Tweets `NULL` ist, muss der Text des Tweets „removed“ lauten
- Das Datum eines Likes darf nicht vor dem Datum des Tweets liegen.

Für die erste Eigenschaft muss eine `check`-Bedingung zur Relation Tweet hinzugefügt werden:

```
create table Tweet (
    id integer not null primary key,
    user_id integer null references Twitter_User on delete set null,
    tweet_date timestamp not null,
    tweet_text varchar(500) not null,
    check (user_id is not null or tweet_text = 'removed')
);
```

Für die zweite Eigenschaft wird eine `check`-Bedingung mit einer Unterabfrage in der Relation likes benötigt.

```
create table likes (
    user_id integer not null references Twitter_User on delete cascade,
    tweet_id integer not null references Tweet on delete cascade,
    like_date timestamp not null,
    primary key (user_id, tweet_id),
    check (exists (
        select *
        from Tweet t
        where
            t.id = tweet_id and
            t.tweet_date <= like_date
    ))
);
```

Infektionsschema: Definition und Beispielausprägung

Person : {[PersonId, Name, Geburtsjahr]}
SozialeGruppe : {[GruppeId, Beschreibung]}
MitgliedIn : {[GruppeId, PersonId]}
infiziert : {[WurdeInfiziert, HatInfiziert, GruppeId]}
Labor : {[LaborId, Name]}
Virentest : {[LaborId, PersonId, Testergebnis]}

Person		
PersonId	Name	Geburtsjahr
63061	Noah	1997
63108	Emma	2008
63258	Finn	1981
63376	Ben	1965
63533	Paul	1982
63663	Mia	1976
63748	Sarah	1986
63875	Markus	1957

SozialeGruppe	
GruppeId	Beschreibung
47005	Familie Sichert
47011	Familie Anneser
47012	Lehrstuhl I25
47015	Kindergarten

MitgliedIn	
GruppeId	**PersonId**
47005	63533
47005	63748
47005	63875
47011	63061
47011	63108
47011	63376
47011	63663
47012	63533
47012	63748
47015	63258
47015	63376
47015	63663
47015	63748

infiziert		
WurdeInfiziert	HatInfiziert	GruppeId
63061	63376	47011
63108	63376	47011
63663	63376	47011
63258	63663	47015

Labor	
LaborId	Name
53001	Charité Berlin
53004	Klinikum rechts der Isar

Virentest		
LaborId	**PersonId**	Testergebnis
53001	63061	positiv
53001	63108	positiv
53001	63376	positiv
53001	63533	negativ
53001	63663	positiv
53004	63258	positiv
53004	63376	positiv
53004	63533	negativ
53004	63748	negativ