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

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS22/23

Michael Jungmair, Stefan Lehner, Moritz Sichert, Lukas Vogel (gdb@in.tum.de)
https://db.in.tum.de/teaching/ws2223/grundlagen/

Blatt Nr. 07

Hausaufgabe 1

Gegeben sei die folgende Relation `ZehnkampfD` mit Athletennamen und den von ihnen erreichten Punkten in den jeweiligen Zehnkampfdisziplinen:

ZehnkampfD : {Name, Disziplin, Punkte}

Name	Disziplin	Punkte
Bolt	100m	50
Bolt	Weitsprung	50
Eaton	100m	40
Eaton	Weitsprung	60
Suarez	100m	60
Suarez	Weitsprung	60
Behrenbruch	100m	30
Behrenbruch	Weitsprung	50
...	...	...

Ermitteln Sie die Silbermedaillengewinner. Eine Silbermedaille bekommen alle, für die gilt: es gibt genau eine/n mit mehr Gesamtpunkten. Formulieren Sie die Anfrage in SQL

a) mit korrelierter Unteranfrage

b) basierend auf Zählen

Sie dürfen davon ausgehen, dass jeder Sportler in jeder Disziplin angetreten ist.

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

Lösung:

Mit korrelierter Unteranfrage:

```
with gesamtpunkte(name,punkte) as(
    select z.name, sum(punkte)
    from zehnkampfD z
    group by z.name
)

select name from gesamtpunkte silber
where exists(
select *
from gesamtpunkte gold
where gold.name <> silber.name
and gold.punkte>silber.punkte
and not exists(
  select *
  from gesamtpunkte auchBesserAlsSilber
  where auchBesserAlsSilber.name<> gold.name
  and auchBesserAlsSilber.punkte>silber.punkte
  )
)
```

Basierend auf Zählen

```
with gesamtpunkte(name,punkte) as(
    select z.name, sum(punkte)
    from ZehnkampfD z
    group by z.name
)
select silber.name
from gesamtpunkte silber, gesamtpunkte besser
where silber.punkte<besser.punkte
group by silber.name
having count(besser.name)=1
```

Hausaufgabe 2

„Frühestes Semester“: Formulieren Sie eine SQL-Anfrage, um das Semester zu ermitteln, in dem die Vorlesung „Der Wiener Kreis“ frühestens gehört werden kann. Um eine Vorlesung hören zu können, müssen zu Semesterbeginn alle Voraussetzungen der Vorlesung bereits gehört worden sein. Testen Sie die Anfrage auch mit anderen Vorlesungen, insbesondere mit „Logik“.

Lösung:

```
with recursive voraussetzenRek(vorlnr, counter) as (
(
  select v.vorlnr, 1 as counter
    from vorlesungen v
   where v.titel = 'Der Wiener Kreis'
)
```

```
  union
  (
    select vs.vorgaenger, vsr.counter + 1 as counter
      from voraussetzenRek vsr, voraussetzen vs
     where vsr.vorlnr=vs.nachfolger
  )
)
select max(counter) as fruehestesSemester
  from voraussetzenRek;
```

Hausaufgabe 3

Führen Sie die folgenden Änderungen am Datenbestand des bekannten Universitätsschemas in SQL aus. Stellen Sie sicher, dass Ihre SQL-Statements mit jeder beliebigen Ausprägung des Schemas funktionieren.

a) Alle Professoren, die den Rang C3 haben, werden auf den Rang C4 befördert. Setzen Sie dazu den Rang aller C3-Professoren auf C4.

b) Die Planetenbewegungen sind vollständig erforscht. Löschen Sie alle Assistenten mit diesem Fachgebiet.

c) Eine neue Vorlesung mit dem Namen „Grundlagen: Datenbanken“ mit der Nummer 5278 soll erstellt werden. Die Vorlesung wird von der Professorin Curie gehalten und hat die Vorlesung „Logik“ als Voraussetzung. Sie soll 4 SWS umfassen. Tragen Sie den Studenten mit der Matrikelnummer 28106 als Hörer der Vorlesung ein. Erstellen Sie alle notwendigen SQL-Statements.

Lösung:

a) Alle Professoren, die den Rang C3 haben, werden auf den Rang C4 befördert. Setzen Sie dazu den Rang aller C3-Professoren auf C4.

```
update Professoren set Rang = 'C4' where Rang = 'C3';
```

b) Die Planetenbewegungen sind vollständig erforscht. Löschen Sie alle Assistenten mit diesem Fachgebiet.

```
delete
from Assistenten
where Fachgebiet = 'Planetenbewegung';
```

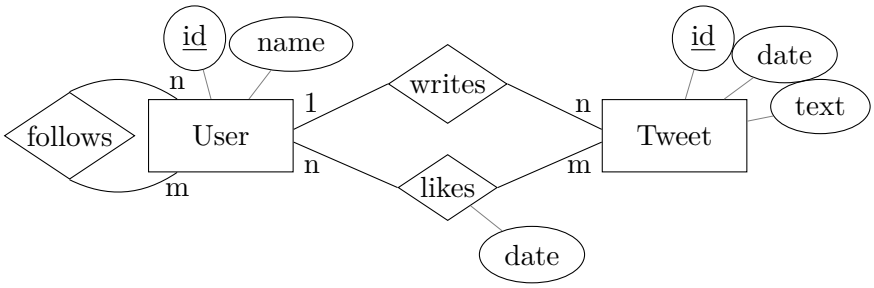
c) Eine neue Vorlesung mit dem Namen „Grundlagen: Datenbanken“ mit der Nummer 5278 soll erstellt werden. Die Vorlesung wird von der Professorin Curie gehalten und hat die Vorlesung „Logik“ als Voraussetzung. Sie soll 4 SWS umfassen. Tragen Sie den Studenten mit der Matrikelnummer 28106 als Hörer der Vorlesung ein. Erstellen Sie alle notwendigen SQL-Statements.

```
insert into Vorlesungen
select 5278, 'Grundlagen: Datenbanken', 4, PersNr
from Professoren
where Name = 'Curie';

insert into voraussetzen
```

```
select VorlNr, 5278
from Vorlesungen
where Titel = 'Logik';

insert into hoeren values (28106, 5278);
```

Hausaufgabe 4

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 statische 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 statische 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
    ))
);
```