Grundlagen Datenbanken: Übung 06

Tanmay Deshpande

Gruppe 20 & 21

ge94vem@mytum.de

QR-Code für die Folien

Wiederholung

Woche 06

Rekursion

- Ein CTE, der sich selbst referenziert heißt recursive
- Mit keyword **recursive** gezeigt
- Bsp: Die folgende Anfrage ermittelt alle (direkten und indirekten) Vorgänger von allen Vorlesungen

```
with recursive allevorg as
(
select * from voraussetzen
union all
select a.vorgaenger, v.nachfolger from voraussetzen v, allevorg a
where a.nachfolger = v.vorgaenger
)

select distinct * from allevorg
```

- UNION entfernt Duplikate, UNION ALL nicht!
- UNION ALL schneller

Schemadefinition und Schemaveränderung in SQL

Erstellen von Tabellen

create table Professoren
(PersNr **integer not null**,
Name **varchar(10) not null**,
Rang **character(2)**);

Einfügen von Tupeln

- insert into hören
  select MatrNr, VorlNr from Studenten, Vorlesungen
  where Titel= `Logik‘ ;
- insert into Studenten (MatrNr, Name)
  values (28121, `Archimedes‘);

Schemadefinition und Schemaveränderung in SQL

Löschen von Tupeln

```
delete Studenten where Semester > 13;
```

Verändern von Tupeln

```
update Studenten set Semester = Semester + 1;
```

Integritätsbedingungen

- Wir wollen manche Bedingungen im Datenbank festlegen:
  - Primär- und Fremdschlüssel kennzeichnen
  - Attribute, die unique sind oder nicht null sein dürfen
  - Referenzielle Integrität
  - Komplexere Bedingungen
- Bsp:

  create Table Book (
  ISBN **integer primary key not null**,
  AuthorID **integer not null references Author on delete cascade**,
  Title **varchar(200) not null**,
  PublisherID **integer references Publisher on delete set null**,
  Pages **integer check (Pages > 0)**
  );

Aufgaben

Woche 06

Aufgabe 01

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

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

- a) Ermitteln Sie den Männeranteil an den verschiedenen Fakultäten in SQL!

- 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.

Lösungsvorschlag 01a

- A) Männeranteil pro Fakultät:

- with FakultaetTotal as
  (select Fakultaet, count(*) as Total from StudentenGF group by Fakultaet),
  FakultaetMaenner as
  (select Fakultaet,count(*) as Maenner from StudentenGF where Geschlecht = 'M' group by Fakultaet)

  select FakultaetTotal.Fakultaet, (case when Maenner is null then 0 else Maenner end)/(total*1.00)
  from FakultaetTotal left outer join FakultaetMaenner on FakultaetTotal.Fakultaet =
  FakultaetMaenner.Fakultaet

- select Fakultaet, (sum(case when Geschlecht = 'M' then 1.00 else 0.00 end)) / count(*) from
  StudentenGF group by Fakultaet

Lösungsvorschlag 01b

- b) Studenten, die alle Vorlesungen ihrer Fakultät hören:

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

- select * from StudentenGF s where
  (select count(*) from Vorlesungen v, ProfessorenF p where
  v.gelesenVon = p.PersNr and p.Fakultaet = s.Fakultaet) =
  (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.Fakultaet = s.Fakultaet)

Aufgabe 02

- Finden Sie alle Vorlesungen (VorlNr und Titel duplikatfrei ausgeben), die nicht vor dem dritten Semester gehört werden sollten.

  Ein Beispiel hierfür ist die Vorlesung Bioethik (5216), da diese die Vorlesung Ethik (5041) voraussetzt, welche wiederum die Vorlesung Grundzüge (5001) als Voraussetzung hat.

Lösungsvorschlag 02

- Ohne Rekursion:
  SELECT DISTINCT v.VorlNr, v.Titel FROM Vorlesungen v, voraussetzen v1, voraussetzen v2
  WHERE v.VorlNr = v1.Nachfolger AND v1.Vorgaenger = v2.Nachfolger
- Mit Rekursion:
  WITH recursive vor as
  (SELECT *, 1 as cnt FROM voraussetzen
  UNION
  SELECT v1.vorgaenger, v2.nachfolger, v1.cnt + 1 FROM
  vor v1, voraussetzen v2 WHERE v1.nachfolger = v2.vorgaenger)

  SELECT DISTINCT nachfolger, v.Titel FROM vor, vorlesungen v WHERE vor.nachfolger = v.VorlNr AND
  vor.cnt >= 2

Aufgabe 03a

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

a) Geben Sie eine Anfrage an, welche für alle Stationen ermittelt, welche anderen Stationen erreicht werden können.

Beachten Sie, dass nur tatsächlich mögliche Verbindungen ausgegeben werden sollen, d.h. die Abfahrt an einer Haltestelle darf nicht vor der Ankunft liegen.

Lösungsvorschlag 03a

- with recursive fahrplan_rec as
  (select von, nach, abfahrt, ankunft from fahrplan
  union all
  select fr.von, f.nach, fr.abfahrt, f.ankunft
  from fahrplan_rec fr, fahrplan f
  where fr.nach = f.von and fr.ankunft <= f.abfahrt and fr.von != f.nach)

  select * from fahrplan_rec

Aufgabe 03b

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

b) Erweitern Sie ihre Anfrage aus Teilaufgabe a), sodass zusätzlich die summierte Fahrtzeit und Wartezeit sowie die gesamte Reisezeit ausgegeben wird.

Lösungsvorschlag 03b

- with recursive fahrplan_rec_linie as
  (select von, nach, abfahrt, ankunft,
  ankunft - abfahrt as fahrtzeit, INTERVAL '00:00:00' as wartezeit
  from fahrplan
  union all
  select fr.von, f.nach, fr.abfahrt, f.ankunft,
  fr.fahrtzeit + (f.ankunft - f.abfahrt), fr.wartezeit + (f.abfahrt - fr.ankunft)
  from fahrplan_rec_linie fr, fahrplan f
  where fr.nach = f.von and fr.ankunft <= f.abfahrt and fr.von != f.nach),

  fahrplan_rec as
  (select von, nach, abfahrt, ankunft, fahrtzeit, wartezeit, fahrtzeit + wartezeit as reisezeit
  from fahrplan_rec_linie)

  select * from fahrplan_rec

Aufgabe 03c

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

c)
Erweitern Sie ihre Anfrage aus Teilaufgabe a) oder b) nochmals und geben Sie die Anzahl der Umstiege für jede Verbindung aus.

Lösungsvorschlag 03c

- with recursive fahrplan_rec_linie as
  (select von, nach, abfahrt, ankunft,
  linie as aktuelle_linie, 0 as umstiege, ankunft - abfahrt as fahrtzeit, INTERVAL '00:00:00' as wartezeit
  from fahrplan
  union all
  select fr.von, f.nach, fr.abfahrt, f.ankunft,
  f.linie, fr.umstiege + case when f.linie != fr.aktuelle_linie or f.abfahrt > fr.ankunft then 1 else 0 end,
  fr.fahrtzeit + (f.ankunft - f.abfahrt), fr.wartezeit + (f.abfahrt - fr.ankunft)
  from fahrplan_rec_linie fr, fahrplan f
  where fr.nach = f.von and fr.ankunft <= f.abfahrt and fr.von != f.nach),

  fahrplan_rec as
  (select von, nach, abfahrt, ankunft, umstiege, fahrtzeit, wartezeit, fahrtzeit + wartezeit as reisezeit from
  fahrplan_rec_linie)

  select * from fahrplan_rec

Aufgabe 03d

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

d) Finden Sie die „guten“ Verbindungen, um von Fröttmaning pünktlich zur Vorlesung „Grundlagen: Datenbanken“ um 10:30 Uhr zu kommen.
Eine Verbindung ist „gut“, wenn sie spätestens um 10:30 in „Garching, Forschungszentrum“ ist und es keine andere Verbindung gibt, die später abfährt aber noch rechtzeitig eintrifft, deren Reisezeit geringer ist und bei der man weniger Umstiege hat.

Lösungsvorschlag 03d

- select * from fahrplan_rec fr
  where
  fr.von = 'Fröttmaning' and fr.nach = 'Garching, Forschungszentrum' and fr.ankunft <= TIME '10:30:00' and
  not exists
  (select * from fahrplan_rec fr2 where fr2.von = fr.von and fr2.nach = fr.nach and
  fr2.ankunft <= TIME '10:30:00’
  and fr2.abfahrt > fr.abfahrt
  and fr2.reisezeit < fr.reisezeit
  and fr2.umstiege < fr.umstiege)

Aufgabe 04a

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

- Geben Sie SQL-Statements zum Erzeugen der Relationen an

Lösungsvorschlag 04a

- 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));

Aufgabe 04b

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

- 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.

Lösungsvorschlag 04b

- 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));

Aufgabe 04c

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

- 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

Lösungsvorschlag 04c

- 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, **check (user_id is not null or tweet_text = 'removed')**);

- 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), **check (exists (select * from Tweet t where t.id = tweet_id and t.tweet_date <= like_date)**);