Institut für Informatik
Prof. Dr. Oliver Vornberger
Dipl.-Math. Patrick Fox

Universität Osnabrück, 26.05.2009
http://www-lehre.inf.uos.de/~dbs
Abgabe bis 08.06.2009, 12:00 Uhr

Übungen zu Datenbanksysteme

Sommersemester 2009

Blatt 6

Aufgabe 6.1 (30 Punkte)

Um die folgenden Abfragen in SQL an Ihre Datenbank `user_[loginname]_WM2006` stellen zu können, müssen Sie zunächst einen Teil der Abfrage als *Sicht* (`View`) zur Verfügung stellen und mit einer `select`-Abfrage darauf zugreifen. Formulieren Sie anschließend mit Hilfe der Views die folgenden Abfragen in SQL und überprüfen Sie sie anhand Ihrer Datenbank:

1. Bestimmen Sie die Spielbegegnung(en), in denen die meisten Verwarnungen ausgesprochen wurden. Geben Sie dabei auch die Schiedsrichter an, die die entsprechenden Verwarnungen ausgesprochen haben.
2. Erstellen Sie für die die Fußball-WM eine Übersicht der Vorrundenergebnisse. Gruppieren Sie nach den Gruppen der Vorrunde und sortieren Sie die Mannschaften innerhalb der Gruppen absteigend bezüglich der erzielten Punkte.

   Definieren Sie zunächst eine View, die die beiden Mannschaften der Spiele sowie die erzielten Tore auflistet. Beachten Sie dabei, auch die Eigentore einer Mannschaft als Gegentore zu zählen.
3. Welche Mannschaften haben den besten Torhüter? Das sind die Mannschaften, die im Schnitt die wenigsten Gegentore bekommen haben.

   Definieren Sie zunächst eine View, die die Anzahl der Spiele pro Mannschaft auflistet, sowie eine View, die die Anzahl der Gegentore pro Mannschaft auflistet. Beachten Sie dabei wieder, die Eigentore als Gegentore zu zählen.

Musterlösung vom 08.06.2008:

Es ergeben sich folgende Abfragen:

1.

```
-- Eine Sicht, die die Verwarnungen pro Spiel ausgibt

create view VerwJeSpiel as
select SpielID, count(*) as AnzVerw
  from Verwarnung
 group by SpielID;


-- Drei verschiedene Abfragen
```

```
-- SELECT mit 'NOT EXISTS'

select sp.Mannschaft1, sp.Mannschaft2, s.Name, v1.AnzVerw
  from Spiele sp, VerwJeSpiel v1, Schiedsrichter s, leitet l
 where sp.SpielID = v1.SpielID
   and l.SpielID  = v1.SpielID
   and l.SID      = s.SID
   and l.Funktion = 'Referee'
   and not exists
    (select *
       from VerwJeSpiel v2
      where v1.AnzVerw < v2.AnzVerw)


-- SELECT mit 'IN'

select sp.Mannschaft1, sp.Mannschaft2, s.Name, v1.AnzVerw
  from Spiele sp, VerwJeSpiel v1, Schiedsrichter s, leitet l
 where sp.SpielID = v1.SpielID
   and l.SpielID  = v1.SpielID
   and l.SID      = s.SID
   and l.Funktion = 'Referee'
   and v1.AnzVerw in
    (select max(AnzVerw)
       from VerwJeSpiel);

-- SELECT mit 'HAVING'

select sp.Mannschaft1, sp.Mannschaft2, s.Name, v1.AnzVerw
  from Spiele sp, VerwJeSpiel v1, Schiedsrichter s, leitet l
 where sp.SpielID = v1.SpielID
   and l.SpielID  = v1.SpielID
   and l.SID      = s.SID
   and l.Funktion = 'Referee'
 group by sp.Mannschaft1, sp.Mannschaft2, s.Name, v1.AnzVerw
having v1.AnzVerw =
    (select max(AnzVerw)
       from VerwJeSpiel);
```

2.

```
-- Eine View, die die Spiele der Vorrunde mit den jeweils von beiden
-- Mannschaften erzielten Toren auflistet

create view Vorrunde2 as
select sp.*,
       (select count(*)
          from Tore t, Spieler s
         where t.SpielID = sp.SpielID
           and t.SpielerNr = s.SpielerNr
           and ((s.Land = sp.Mannschaft1
           and t.Spielsituation != 'own goal')
            or (s.Land = sp.Mannschaft2
           and t.Spielsituation = 'own goal'))) as Tore1,
       (select count(*)
          from Tore t, Spieler s
         where t.SpielID = sp.SpielID
```

```
           and t.SpielerNr = s.SpielerNr
           and ((s.Land = sp.Mannschaft2
           and t.Spielsituation != 'own goal')
            or (s.Land = sp.Mannschaft1
           and t.Spielsituation = 'own goal'))) as Tore2
  from Spiele sp
 where Runde like 'Group%';


-- Abfrage auf der View
-- Die inneren selects berechnen mit Hilfe des case-Operators die Punkte pro
-- Spiel, jewels fuer beide Mannschaften
-- Das aeussere select summiert die Punkte pro Mannschaft auf, sortiert
-- entsprechend

select t.Runde, t.Land, sum(t.Punkte) as Gesamtpunkte
from
  (select Runde, Mannschaft1 as Land,
    (case when Tore1 < Tore2 then 0
          when Tore1 = Tore2 then 1
          when Tore1 > Tore2 then 3
     end) as Punkte
     from Vorrunde
   union all
   select Runde, Mannschaft2 as Land,
    (case when Tore2 < Tore1 then 0
     when Tore2 = Tore1 then 1
     when Tore2 > Tore1 then 3
     end) as Punkte
     from Vorrunde) t
group by t.Runde, t.Land
order by t.Runde asc, Gesamtpunkte desc;
```

3.

```
-- Eine View, die die Spiele pro Mannschaft zaehlt

  create View SpieleJeMannschaft as
  select m.Land, count(*) as Anzahl
    from Spiele s, Mannschaften m
   where s.Mannschaft1 = m.Land
      or s.Mannschaft2 = m.Land
group by m.Land


-- Das MySQL keine Subselect in der 'from'-clause in Views zulaesst, wird eine
-- Hilfsview definiert, die fuer jedes Gegentor einmal die Mannschaft
-- auflistet, die das Tor kassiert hat.

create view Subselect as
select s.Mannschaft1 as Land
  from Spiele s, Tore t, Spieler sp
 where s.SpielID = t.SpielID
   and s.Mannschaft2 = sp.Land
   and sp.SpielerNr = t.SpielerNr
   and t.Spielsituation != 'own goal'
 union all
```

```
select s.Mannschaft2 as Land
  from Spiele s, Tore t, Spieler sp
 where s.SpielID = t.SpielID
   and s.Mannschaft1 = sp.Land
   and sp.SpielerNr = t.SpielerNr
   and t.Spielsituation != 'own goal'
 union all
select sp.Land as Land
  from Spieler sp, Tore t
 where sp.SpielerNr = t.SpielerNr
   and t.Spielsituation = 'own goal'


-- Select, welches ueber die Hilfsview laeuft und die Eintraege pro Mannschaft,
-- also die Anzahl der Gegentore zaehlt

create view GegentoreJeLand as
  (select Land, count(*) as Gegentore
     from Subselect
 group by Land)


-- Eigentliche Abfrage
-- Das erste innere Select teilt die Anzahl der Gegentore durch die Anzahl der
-- Spiele fuer jede Mannschaft. Das zweite innere select ergaenzt Mannschaften,
-- die keine Gegentore (ausser im Elfmeterschiessen) bekommen haben.
-- Aeusseres select wird benoetigt, um auf der Vereinigungsmenge (union) der
-- beiden inneren Selects sortieren zu koennen

select *
  from (select s.Land, g.Gegentore / s.Anzahl as Rang
          from SpieleJeMannschaft s, GegentoreJeLand g
         where g.Land = s.Land
         union
        select m.Land, 0 as Rang
          from Mannschaften m
         where m.Land not in (select s.Land from GegentoreJeLand s)) tmp
order by Rang
```

Aufgabe 6.2 (10 Punkte)

Alle Studenten sollen ab sofort alle 4-stündigen Vorlesungen von Dozent 'Sokrates' hören. Formulieren Sie eine SQL-Anfrage, die diese Operation durchführt.

Musterlösung vom 08.06.2008:

```
-- Das aeussere select holt alle Studenten und bildet das Kreuzprodukt
-- mit allen relevanten Vorlesungsnummern. Das not exists schliesst dabei die
-- Tupel aus der Menge aus, die schon in hoeren enthalten sind, da das insert
-- sonst eine Fehlermeldung werfen wuerde.

insert into hoeren
select s.MatrNr, v.VorlNr
  from Vorlesungen v, Professoren p, Studenten s
 where v.gelesenVon = p.PersNr
```

```
and SWS = 4
and p.Name = 'Sokrates'
and not exists (select *
                  from hoeren h
                 where h.MatrNr = s.MatrNr
                   and v.VorlNr = h.VorlNr)
```

Aufgabe 6.3 (10 Punkte)

Die Schiedsrichter in Ihrer Datenbank `user_[loginname]_WM2006` sind entweder ein 'Referee' oder ein 'Assistant Referee'. Finden Sie mit Hilfe der MySQL-Dokumentation heraus, wie Sie eine SQL-Anfrage erstellen können, die die Tabelle Schiedsrichter so verändert, dass der Datenbank-Server beim Einfügen von neuen Schiedsrichtern testet, ob es sich bei dem eingefügten Schiedsrichter um einen 'Referee' oder einen 'Assistent Referee' handelt.

Musterlösung vom 08.06.2008:

```
-- Attribut der Tabelle Schiedsrichter einschraenken

alter table Schiedsrichter
modify Funktion ENUM('Referee','Assistant Referee')


-- test, ob's geklappt hat

insert into Schiedsrichter
values ('Schiedsrichter', 'Nobody', 'GER', 30003)
```

Aufgabe 6.4 (25 Punkte)

Gegeben seien die folgenden Relationen:

```
Praktika:            {[PraktNr, Titel, SWS, betreutVon]}
Teilnahme:           {[MatrNr, PraktNr]}
Praktikumsräume :    {[RaumNr, AnzPlaetze]}
Zugangberechtigung:  {[MatrNr, RaumNr]}
Belegung:            {[PersNr, RaumNr]}
```

Diese Relationen sollen Ihre Datenbank `user_[loginname]_Uni` ergänzen. Erstellen Sie für die fünf Relationen `create table`-Anweisungen, die neben den Attributen bereits die Fremdschlüsselangaben mit sinnvollen `cascade`-Angaben enthalten. Erstellen Sie außerdem ein SQL-Skript, um einige Tupel einzufügen. Ändern und löschen Sie einige Tupel in Ihrer Ausprägung, um die Fremdschlüssel-Überprüfung zu testen. Geben Sie zusätzlich an, welche Änderungen sich in Ihrer Datenbank-Ausprägung bei welcher Operation ergeben.

Musterlösung vom 08.06.2008:

```
-- Tabellenstruktur fuer Tabelle 'Laborraeume'

DROP TABLE IF EXISTS Laborraeume;
CREATE TABLE Laborraeume (
RaumNr int(11) NOT NULL,
```

```
AnzPlaetze int(11) NOT NULL,
PRIMARY KEY  (RaumNr)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Tabellenstruktur fuer Tabelle 'Praktika'

DROP TABLE IF EXISTS Praktika;
CREATE TABLE Praktika (
PraktNr int(11) NOT NULL,
Titel varchar(20) default NULL,
SWS int(11) default NULL,
PersNr int(11) default NULL,
PRIMARY KEY  (PraktNr),
FOREIGN KEY (PersNr) REFERENCES Professoren (PersNr)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Tabellenstruktur fuer Tabelle 'Teilnahme'

DROP TABLE IF EXISTS Teilnahme;
CREATE TABLE Teilnahme (
MatrNr int(11) NOT NULL,
PraktNr int(11) NOT NULL,
PRIMARY KEY  (MatrNr,PraktNr)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Tabellenstruktur fuer Tabelle 'Belegung'

DROP TABLE IF EXISTS Belegung;
CREATE TABLE Belegung (
PersNr int(11) NOT NULL,
RaumNr int(11) NOT NULL,
PRIMARY KEY  (PersNr,RaumNr),
FOREIGN KEY (RaumNr) REFERENCES Laborraeume (RaumNr)
  ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (PersNr) REFERENCES Professoren (PersNr)
  ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Tabellenstruktur fuer Tabelle 'Zugangsberechtigung'

DROP TABLE IF EXISTS Zugangsberechtigung;
CREATE TABLE Zugangsberechtigung (
MatrNr int(11) NOT NULL,
RaumNr int(11) NOT NULL,
PRIMARY KEY  (MatrNr,RaumNr),
FOREIGN KEY (RaumNr) REFERENCES Laborraeume (RaumNr)
  ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (MatrNr) REFERENCES Studenten (MatrNr)
  ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Aufgabe 6.5 (25 Punkte)

Wenn ein Student die Teilnahme an einem Praktikum in die Datenbank eingetragen wird, soll auch ein Tupel in die Tabelle `Zugangsberechtigung` eingetragen werden, damit der Student, sofern noch nicht vorhanden, auch eine Freischaltung für seine Zugangskarte zu den entsprechenden Prakti-

kumsräumen erhält.

1. Implementieren Sie eine *stored procedure*, die die notwendigen Tupel in die jeweiligen Tabellen einträgt. Welche Parameter erhält die *stored procedure* beim Aufruf?
2. Erstellen Sie außerdem einen *Trigger*, der die gleiche Aufgabe automatisch löst, wenn ein Student in ein Praktikum eingetragen wird.

Musterlösung vom 08.06.2008:

1.

```
-- Idee: Fuege zunaechst Tupel in die Tabelle 'Teilnahme' ein,
-- sofern noch nicht vorhanden. Anschliessend fuege in die Tabelle
-- 'Zugangsberechtigung' die Tupel ein, die laut den Tabellen 'Teilnahme', 'Belegung'
-- und 'Praktika' in 'Zugangsberechtigung' zu finden sein muessten. Das select
-- findet zunaechst alle Berechtigungen und zieht davon die ab, die schon
-- in der Tabelle 'Zugangsberechtigung' vorhanden sind, da keine doppelten
-- eingetragen werden koennen.

-- DELIMITER $$

CREATE PROCEDURE teilnehmer(mnr integer, pnr integer)
BEGIN
  IF (
    NOT EXISTS (
      SELECT * FROM Teilnahme WHERE MatrNr = mnr AND PraktNr = pnr
    )
  )
  THEN
    INSERT INTO Teilnahme (MatrNr, PraktNr)
    VALUES (mnr, pnr);

END IF ;

  INSERT INTO Zugangsberechtigung
  SELECT t.MatrNr, b.RaumNr
  FROM Teilnahme t, Belegung b, Praktika p
  WHERE t.PraktNr = p.PraktNr and p.PersNr = b.PersNr
  AND NOT EXISTS
    (SELECT * FROM Zugangsberechtigung h
     WHERE t.MatrNr = h.MatrNr and b.RaumNr = h.RaumNr);

END $$

-- DELIMITER ;
```

2.

```
-- Idee: Fuere nach dem Einfuegen in die Tabelle 'Teilnahme' das
-- gleiche INSERT-Statement aus wie in der Stored Procedure. Das traegt
-- einfach die fehlenden Berechtigungen in die Tabelle 'Zugangsber.' ein.

CREATE TRIGGER Zugang AFTER INSERT ON Teilnahme
  FOR EACH ROW
```

```
INSERT INTO Zugangsberechtigung
SELECT t.MatrNr, b.RaumNr
FROM Teilnahme t, Belegung b, Praktika p
WHERE t.PraktNr = p.PraktNr and p.PersNr = b.PersNr
AND NOT EXISTS
  (SELECT * FROM Zugangsberechtigung h
   WHERE t.MatrNr = h.MatrNr and b.RaumNr = h.RaumNr)
```