JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

INSY

SEMESTERAUFGABE - "FUSSBALLVEREIN"

Systemtechnik INSY
4CHIT 2015/16

Maro Otahal

Version 1

Note:

Betreuer: Borko

Begonnen am 15. April 2016

Beendet am 17. April 2016

Inhaltsverzeichnis

1 Einführung

1.1 Ziele

Ein österreichischer Fußballverein braucht eine neue Datenbank, um zumindest die Personenverwaltung auf eine professionelle Basis zu stellen. In dieser Datenbank werden folgende Daten verwaltet:

Jede Person ist identifiziert durch eine eindeutige Personalnummer (kurz "persnr"). Außerdem werden zu jeder Person folgende Informationen verwaltet: Vorname ("vname"), Nachname ("nname"), Geschlecht ("geschlecht") und Geburtsdatum ("gebdat"). Für "geschlecht" sind einzig die Eingaben "W", "M" und "N" gültig.

Jede Person, die am Vereinsleben beteiligt ist, gehört zu genau einer der folgenden 4 Kategorien: Angestellter, Vereinsmitglied (kurz "Mitglied"), Spieler oder Trainer. Für all diese Kategorien von Personen müssen zusätzliche Informationen verwaltet werden:
Von jedem Angestellten werden das Gehalt ("gehalt"), die Überstunden ("ueberstunden") und die E-Mail-Adresse ("e_mail") vermerkt.
Für jedes Vereinsmitglied werden ausständige Beitrage ("beitrag") abgespeichert.
Jeder Spieler hat eine Position ("position") (z.B. Tormann, Stürmer, etc.). Außerdem sind Gehalt ("gehalt") sowie Beginn ("von") und Ende ("bis") der Vertragsdauer abzuspeichern. Jeder Spieler ist zumindest einer Mannschaft zugeordnet. Die Spieler innerhalb einer Mannschaft müssen jeweils eine eindeutige Trikot-Nummer ("nummer") zwischen 1 und 99 haben.
Jeder Trainer hat ein Gehalt ("gehalt") sowie Beginn ("von") und Ende ("bis") der Vertragsdauer. Jeder Trainer ist genau einer Mannschaft zugeordnet.

Der Verein hat mehrere Mannschaften. Für jede Mannschaft sind folgende Informationen zu verwalten: eine eindeutige Bezeichnung ("bezeichnung") (wie 1. Mannschaft, Junioren, A-Jugend, ...), die Klasse ("klasse") sowie das Datum des nächsten Spiels ("naechstes_spiel"). Jede Mannschaft hat mindestens 11 Spieler, genau einen Chef-Trainer und genau einen Trainer-Assistenten. Beide sind als Trainer des Vereins registriert. Außerdem hat jede Mannschaft einen Kapitän (der ein Spieler des Vereins ist).

Der Verein hat mehrere Standorte. Jeder Standort ist identifiziert durch

eine eindeutige ID ("sid"). Außerdem sind zu jedem Standort folgende Informationen verfügbar: Land ("land"), Postleitzahl ("plz") und Ort ("ort").

An jedem Standort gibt es 1 oder mehrere Fan-Clubs. Jeder Fan-Club hat einen für den jeweiligen Standort eindeutigen Namen ("name"), ein Gründungsdatum ("gegruendet") und einen Obmann ("obmann"), der Vereinsmitglied sein muss. Ein Vereinsmitglied kann von höchstens einem Fan-Club der Obmann sein. Die Fan-Clubs werden von Angestellten des Vereins betreut: Und zwar kann jeder Angestellte einen Fan-Club jeweils für einen gewissen Zeitraum betreuen. Dieser Zeitraum ist durch Anfangsdatum ("anfang") und Endedatum ("ende") bestimmt. Jeder Angestellte kann 0 oder mehere Fan-Clubs betreuen, und jeder Fanclub kann von einem oder mehreren Angestellten betreut werden.

Der Verein führt Aufzeichnungen über die absolvierten Spiele. Jedes Spiel ist gekennzeichnet durch die Bezeichnung der Mannschaft ("mannschaft") und das Datum ("datum"). Für jedes Spiel werden der Gegner ("gegner") und das Ergebnis ("ergebnis") eingetragen, das einen der Werte "Sieg", "Unentschieden" oder "Niederlage" haben kann. Außerdem werden zu jedem Spiel die beteiligten Spieler abgespeichert, wobei für jeden Spieler die Dauer Einsatzes ("dauer") als Wert zwischen 1 und 90 vermerkt wird.

1.2 Aufgabenstellung

1.) Schreiben Sie die nötigen CREATE-Befehle, um die vorgestellten Relationen mittels SQL zu realisieren. Dabei sind folgende Punkte zu beachten:

* Die Datenbank soll keine NULL-Werte enthalten.
* Realisieren Sie folgende Attribute mit fortlaufenden Nummern mit Hilfe von Sequences: "persnr" von Personen und "sid" von Standorten. Für das "persnr"-Attribut sollen nur gerade, 5-stellige Zahlen vergeben werden (d.h.: 10000, 10002, ..., 99998). Für das "sid"-Attribut sollen beliebige positive Zahlen (d.h. 1,2, ...) vergeben werden.
* Falls zwischen 2 Tabellen zyklische FOREIGN KEY Beziehungen herrschen, dann sind diese FOREIGN KEYs auf eine Weise zu definieren, dass es möglich ist, immer weitere Datensätze mittels INSERT in diese Tabellen einzufügen.

2.) Schreiben Sie INSERT-Befehle, um Testdaten für die kreierten Tabellen einzurichten. Jede Tabelle soll zumindest 100000 Zeilen enthalten. Sie

dürfen die Wahl der Namen, Bezeichnungen, etc. so einfach wie möglich gestalten, d.h.: Sie müssen nicht "real existierende" Fußballer-Namen, Länder, Städte, etc. wählen. Stattdessen können Sie ruhig 'Spieler 1', 'Spieler 2', 'Land 1', 'Land 2', 'Stadt 1', 'Stadt 2', etc. verwenden. Sie können für die Erstellung der Testdaten auch entsprechende Generatoren verwenden!

3.) Schreiben Sie die nötigen DROP-Befehle, um alle kreierten Datenbankobjekte wieder zu löschen.

Datenbankclient (Java/C++) und DB-Connector (JDBC/libpqxx):

Schreiben Sie einen Client, der eine Datenbank-Verbindung herstellt. Realisieren Sie eine GUI (JavaFX/Qt), die das einfache Ändern (CRUD) der Spieler des Vereins erlaubt. Verwenden Sie dabei auf jeden Fall eine Tabelle (TableView, QTableView), die auch eine grafische Veränderung der Datensätze erlauben soll.

Ermöglichen Sie die gleichzeitige Verbindung von mehreren Clients auf die Datenbasis. Implementieren Sie dabei eine transaktionelle, gesicherte Erstellung und Änderung von Spielen. Beachten Sie dabei, dass der Spielstand und die Spielzeit der einzelnen Spieler laufend und von mehreren Clients gleichzeitig aktualisiert werden könnte. Stellen Sie für die Eingabe der Spielerzeit und Spielstand eine einfache grafische Möglichkeit zur Verfügung. Verwenden Sie dabei Transaktionen bzw. Locks und entsprechende programmtechnische Mittel um Inkonsistenzen zu vermeiden. Definieren Sie dabei für die einzelnen Informationen (Spielerzeit, Spielstand) eigene Threads.

Informationssysteme und Content Management:

Versuchen Sie in einer kurzen schriftlichen Ausführung anzuführen, welche Informationssysteme im betrieblichen Umfeld genutzt werden und wie diese für die vorliegende Aufgabenstellung eingesetzt werden könnten. Im Bereich der Informationssysteme wurde im Unterricht im Speziellen auf die Thematik der Content Management Systeme eingegangen. Versuchen Sie auch hier kurz zu erläutern welche der kennengelernten CMS für die vorliegende Aufgabenstellung besonders gut eingesetzt werden könnten und warum.

Ausgehend von der vorangegangenen Analyse wählt ein entsprechendes Content Management aus, Implementiert ein passendes Template zu unserem Fußballverein und legt entsprechende Benutzer an die in eurem

System erstellen, bearbeiten oder auch nur lesen können sollen. Holen Sie hierbei mittels angelegten Benutzer aus dem Content Management System Datensätze aus der Aufgabe Fußballverein (z.b. Spieler usw.) und zeigt diese entsprechend in eurem CMS an um das Design verdeutlichen sollen.

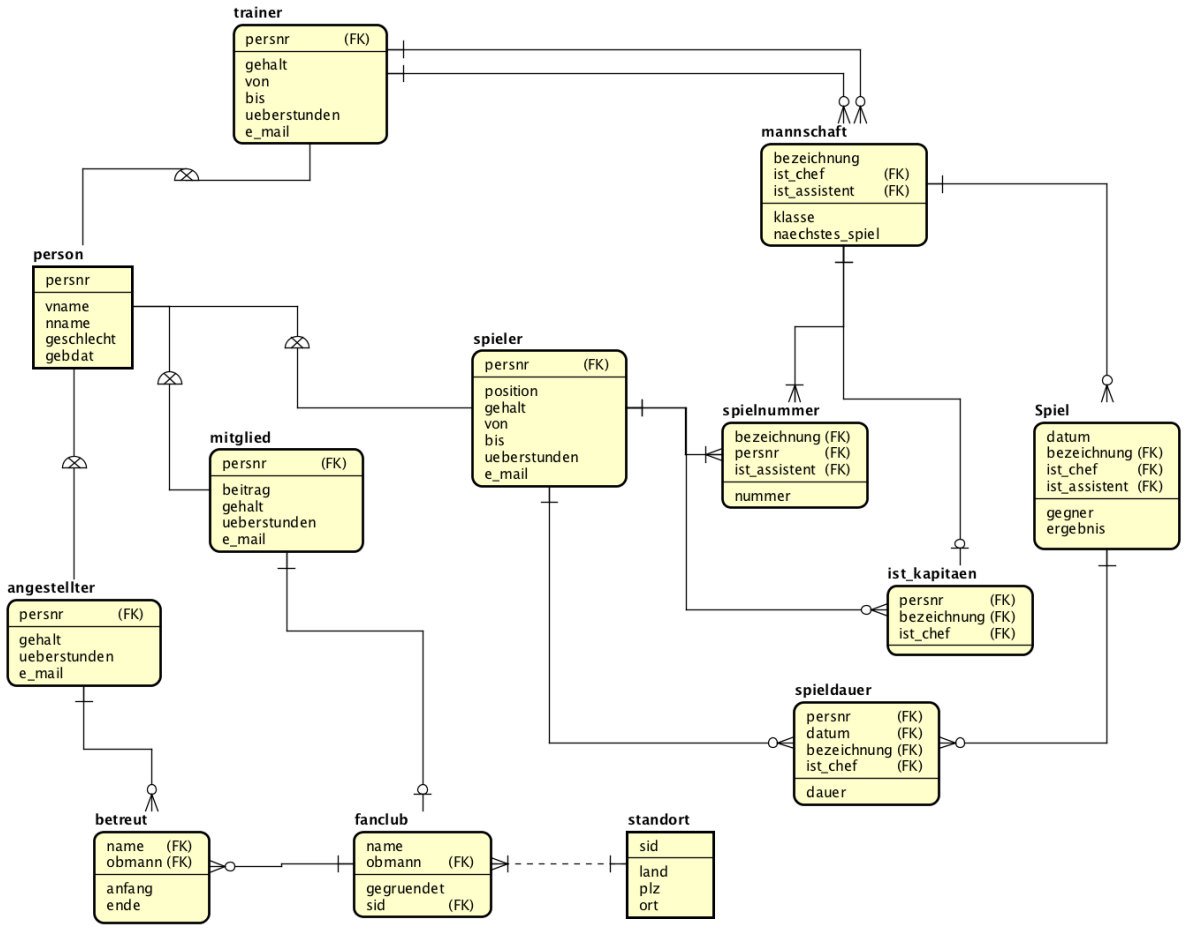
Setzen Sie das entwickelte Rahmenwerk "Fußballverein", das im Zuge der vorliegenden Arbeit generiert wurde, in Form eines Plug-Ins für das von Ihnen gewählte CMS um.

2 Ergebnisse

2.1 ERD in ASTAH

Um in der Datenbank keine NULL-Werte zu haben muss man bei jedem Attribut NOT-NULL ankreuzen.

ER-Diagramm in Astah:



2.2 Create Script / Datafiller einführung

Das Datafiller Script (Python Script) wird unter folgendem Link mit `wget` heruntergeladen: https://www.cri.ensmp.fr/people/coelho/datafiller

Das Script wird mit python ausgeführt, welches davor auf .py umbenannt werden muss!

Folgender Befehl zeigt die nötigen Parameter der Befehlszeile an, um aus dem CREATE script Daten vom Datafiller zu generieren.

```
python datafiller.py ERDiagram-Fussballverein.sql > ERDiagram-Fussballverein-test.sql
```

Wenn man die Tabellen mit 100000 Datensätzen füllen möchte, muss man `--size=100000` angeben: (dies kann man auch einzeln, bei jeder Tabelle durchführen)

```
python datafiller.py --size=100000 ERDiagram-Fussballverein.sql > ERDiagram-Fussballverein-test.sql
```

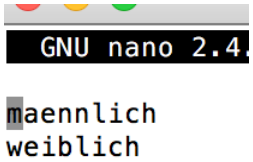
Danach dauert es einige Minuten bis die Datensätze erstellt werden.

Hat man das CREATE Script nach seinen Anforderungen angepasst kann man es mit dem obigen Befehl ausführen.

Wenn man das Script ausführt, überprüft der Datafiller die Kommentare vom CREATE Script und erzeugt daraus die Daten.

Mit `--df` kann man nach jedem Attribut die Funktionen vom Datafiller aufrufen.

2.3 CREATE Script Kommentare



Um bei jeder Person zwischen Männlich und Weiblich zu unterscheiden muss man ein File mit „maennlich" und „weiblich" erstellen.
Dieses wird im CREATE Script `--df fngeschlecht: word=geschlecht` als Funktion deklariert, welche die Daten des Files einliest.
Darauffolgend wird beim benötigten Attribut, in diesem Fall geschlecht die Funktion `--df: use=fngeschlecht` aufgerufen.
Dieser Schritt wird mit den Attributen `Länder, Postion der Spieler` und `Ort` genau so gemacht.

```
--df fncountries: word=countries
```

```
--df fnposition: word=position
```

```
--df fnort: word=ort
```

Bei der E-Mail habe ich die Funktion von der Datafiller Seite kopiert:
`-- df: pattern='[a-z]{3,8}\.[a-z]{3,8}@(gmail|yahoo)\.com'`

Um beim generieren der Daten keine Fehler bei DATE zu bekommen muss man folgenden Kommentar hinzufügen da es über >1900 liegen muss:

`-- df: start=1923-01-01 end=2015-01-01`

[10]

2.3.1 Probleme beim Datafiller

Anfangs war es sehr schwierig sich mit dem Datafiller zurecht zu finden, da auf der Website die Funktionen nicht ausführlich bis gar nicht erklärt wurden.

Da ich nach 3-4h suche nach der Lösung des folgenden Fehlers nicht gekommen bin, habe ich die Tabellen spiel bis betreut aus dem CREATE script rausgelöscht, weil es für meine Teilaufgaben nicht wirklich relevant wäre.

```
# filling table spiel (100000)
ERROR:  insert or update on table "spiel" violates foreign key constraint "fk_spiel_0"
DETAIL:  Key (bezeichnung, ist_chef, ist_assistent)=(bezeichnung_4531, 33753, 74325) is not present in table "mannschaft".
# filling table spieldauer (100000)
ERROR:  current transaction is aborted, commands ignored until end of transaction block
invalid command \.
# filling table spielnummer (100000)
ERROR:  syntax error at or near "21308"
LINE 1: 21308 1991-04-11 bezeichnung_49254_ 79582 8.89497953524
        ^
invalid command \.
# filling table fanclub (100000)
ERROR:  syntax error at or near "bezeichnung_12175_"
LINE 1: bezeichnung_12175_ 99024 5744 57
        ^
invalid command \.
# filling table ist_kapitaen (100000)
ERROR:  syntax error at or near "name_46015"
LINE 1: name_46015 75605 1940-12-27 22235
        ^
```

2.4 Erstellung der Datenbank

Vom root in posgtres wechseln:

`su postgres -> psql`

Anschließend wird in psql eine neue Datenbank erstellt.

`\c fussballverein02`

CREATE Script in die Datenbank laden:
`psql fussballverein02 < ERDiagram-Fussballverein03.sql`

Nachdem das CREATE Script in die Datenbank geladen wurde, werden die INSERTS mit den bereits erstellten Daten in die Tabellen der Datenbank geladen:
`python datafiller.py -f -T ERDiagram-Fussballverein-test.sql | psql fussballverein`

Dies kann auch ein paar Minuten dauern und dann sollte folgender Output kommen:

```
postgres@debian:/root$ python datafiller.py -f -T ERDiagram-Fussballverein-test.sql | psql fussballverein
BEGIN
# filling table person (100000)
COPY 100000
# filling table spieler (100000)
COPY 100000
# filling table standort (100000)
COPY 100000
# filling table trainer (100000)
COPY 100000
# filling table angestellter (100000)
COPY 100000
# filling table mannschaft (100000)
COPY 100000
# filling table mitglied (100000)
COPY 100000
```

Wurden die INSERTS erfolgreich erstellt, kann man die Datenbank einwandfrei benutzen.

Wichtige Shortcuts in postgreSQL:

`psql –d fussballverein02 –U test2`	//man wird mit dem User in die Datenbank eingeloggt
`psql`	// man verbindet sich mit dem Standard user von psql
`\c datenbank`	//man verbindet sich mit der gewünschten DB
`\q`	//man verlässt psql
`\d`	//Tabellen der Datenbank werden angezeigt
`\l`	//Liste der Datenbanken werden angezeigt

3 Transaktionen

3.1 Was sind Transaktionen?

Eine Transaktion ist eine Folge von zusammengehöriger Operationen (SQL Statements), die in einen Transaktionsblock zu packen und zu isolieren.

Wenn ein SQL Statement in einer Transaktion fehlschlägt, dann werden alle Änderungen welche durch die Transaktion verursacht wurden, zurückgesetzt (ROLLBACK).
[2]

3.2 Transaktionseigenschaften

Beschreibt die idealen Eigenschaften von Transaktionen in einem Datenbanksystem

3.2.1 ACID Eigenschaften

Atomarität (A = atomicity)

Eine Transaktion wird entweder komplett oder gar nicht ausgeführt („All or nothing“ – Prinzip). Falls eine Transaktion im halbfertigen Zustand verbleibt (z.B bei Abbruch, Absturz, etc) muss ein ROLLBACK durchgeführt werden. Wenn alle Teilaktionen erfolgreich durchgeführt wurden, wird die Transaktion mit einem COMMIT beendet.

Konsistenz (C = consistency)

Nach Ausführung der Transaktion muss der Datenbestand in einer konsistenten Form sein, wenn er es bereits zu Beginn der Transaktion war.

Isolation (I = isolation)

Mehrere Benutzer können gleichzeitig auf die Datenbank zugreifen. Trotz dieses Mehrbenutzerbetriebs wird garantiert, dass dadurch keine unerwünschten Nebenwirkungen eintreten, wie z. B. das gegenseitige Überschreiben desselben Datenbankobjektes. Vielmehr bietet das DBMS jedem Benutzer und Anwendungsprogramm einen "logischen Einbenutzerbetrieb", so dass parallele Datenbankzugriffe anderer Benutzer unsichtbar bleiben. Diese Isolation bzw. Ablaufintegrität der Transaktionen wird seitens des DBS durch geeignete Synchronisationsmaßnahmen erreicht, z. B. durch bestimmte Sperrverfahren.

Dauerhaftigkeit (D = durability)
Bei erfolgreichem Ende einer Transaktion (COMMIT) müssen die von ihr durchgeführten Änderungen auch nachfolgende Fehlerfälle überleben.

[6] [2] [7]

In PostgreSQL gibt es 4 verschiedene Isolationsstufen:

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

[5]

3.3 Isolation Levels

PostgreSQL unterscheidet zwischen 4 Isolation Levels. In den folgenden Punkten wird näher erläutert, wie genau sich jeweils jeder Isolation Level verhält.

3.3.1 READ UNCOMMITED

Hier ignorieren alle Leseoptionen jegliche Sperren, daher können alle Fehlerfälle auftreten. Jedoch bietet dieses Isolation Level die größte Performance:

3.3.2 READ COMMITED

Bei diesem Isolation Level ist Dirty Read nicht möglich. Daher können Non-Repeatable Read und Phantom Read auftreten, wenn während wiederholten Leseoperationen auf dieselben Daten, zwischen der ersten und der zweiten Leseoperation, eine Schreiboperation einer anderen Transaktion die Daten verändert und committed.

3.3.3 REPEATABLE READ

Bei diesem Isolation Level ist sichergestellt, dass wiederholte Leseoperationen mit den gleichen Parametern auch dieselben Ergebnisse haben. Sowohl bei Lese- als auch bei Schreiboperationen werden für die gesamte Dauer der Transaktion Sperren gesetzt.

Das heißt sobald auf einem Datensatz mit SELECT zugegriffen wird, wird dieser Datensatz bis zum Ende der Transaktion gelockt. Am Ende wird der Lock wieder aufgehoben.

Nonrepeatable Read nicht möglich.

3.3.4 SERIALIZABLE

Die höchste Isolation Level garantiert, dass parallel ablaufende Transaktionen seriell beziehungsweise nacheinander in Folge abgehandelt werden. Das heißt Locks werden auf Tabellenebene gesetzt. z.B. wenn eine Transaktion auf eine Tabelle zugreift, dann wird die ganze Tabelle gesperrt. Somit kann keine andere Transaktion in der Tabelle etwas einfügen, löschen oder verändern.

[5] [8] [9]

3.4 Fehlerfälle

Alle Fehler werden anschließend praktisch dargestellt. Die folgenden Tabellen sollen die zeitliche Abfolge von Befehlen, die einen Fehler hervorrufen, darstellen.

3.4.1 DIRTY READ

Ein Dirty Read ist ein Fehlerfall wobei ein nicht bestätigter (uncommitted) Wert bei einer Transaktion verwendet wird. (Read Uncommitted)

T1	T2
BEGIN	
	BEGIN
	UPDATE X
SELECT X	
	COMMIT
COMMIT	

3.4.2 NONREPEATABLE READ

Beim Nonrepeatable Read wird in einer Transaktion zweimal dasselbe ausgelesen. Allerdings wurde dazwischen dieser Wert verändert. (Read Committed)

T1	T2
BEGIN	
	BEGIN
SELECT X	
	UPDATE X
	COMMIT
SELECT X	
COMMIT	

3.4.3 PHANTOM READ

Beim Phantom Read wird zweimal dieselbe Abfrage durchgeführt (dasselbe Resultset sollte zurückkommen). Dazwischen wird etwas in der entsprechende Tabelle etwas eingefügt/gelöscht
-> das Resultset ist ein anderes

T1	T2
BEGIN	
	BEGIN
SELECT SUM(X) FROM T	
	INSERT T
	COMMIT
SELECT SUM(X) FROM T	
COMMIT	

[Schulübung] [8] [7]

3.5 Praxis

3.4.1 DIRTY READ:

```
begin transaction isolation level READ uncommitted;
```

T1	T2
BEGIN	
	BEGIN
select * from angestellter;	
	UPDATE angestellter set gehalt=100.00 where persnr=1;
select * from angestellter;	
	COMMIT
COMMIT	

3.4.2 NONREPEATABLE READ:

```
begin transaction isolation level READ committed;
```

T1	T2
BEGIN	
	BEGIN
select * from angestellter;	
	UPDATE angestellter set gehalt=100.00 where persnr=2;
	COMMIT
select * from angestellter;	
COMMIT	

3.4.3 Phantom READ

```
begin transaction isolation level REPEATABLE read;
```

T1	T2
BEGIN	
	BEGIN
select sum(gehalt) from angestellter as summe;	
	INSERT into angesteller values (100001,23210.2123,...);
	COMMIT
select sum(gehalt) from angestellter as summe;	
COMMIT	

4 Benutzerverwaltung

4.2 Einleitung

Da in der Aufgabe sehr wenig über die Benutzerverwaltung steht, wollte ich die Aufgabe so ähnlich wie die Aufgabenstellung auf Moodle „SQL Benutzerverwaltung" gestalten.

Daher habe ich mehrere Datenbanken (mit denselben INSERTS) und USER (welche jeweils verschiedene Rechte für jede Datenbank haben) erstellt.

4.3 Durchführung / Praxis

Datenbanken erstellen:

```
Create database fussballverein;

Create database fussballverein02;

Create database fussballverein03;

Create database fussballverein04;
```

Darauffolgend werden jeweils die CREATE Scripts in die Datenbanken geladen und anschließend die INSERTS erstellt.

USER erstellen:

```
create user darfalles with password 'darfalles'; //darf alles

create user nurlesen with password 'nurlesen'; //darf alles lesn

create user test with password 'test';     // bei fussballverein03
                                           schreiben/ lesen, darf bei
                                           fussballverein02 nur lesen

create user test2 with password 'test2';   //fussballverein04 lesen
                                           fussballverein02 schreiben
```

[3]

4.3.1 Rechte vergeben:

`USER darfalles` -> Darf überall schreiben und lesen

```
grant ALL on DATABASE fussballverein to darfalles;

grant ALL on DATABASE fussballverein02 to darfalles;

grant ALL on DATABASE fussballverein03 to darfalles;

grant ALL on DATABASE fussballverein04 to darfalles;
```

[4]

Mit der DB verbinden und für den USER alle Rechte erteilen:

```
\c fussballverein
GRANT ALL ON ALL TABLES IN SCHEMA PUBLIC TO darfalles;

\c fussballverein02
GRANT ALL ON ALL TABLES IN SCHEMA PUBLIC TO darfalles;

\c fussballverein03
GRANT ALL ON ALL TABLES IN SCHEMA PUBLIC TO darfalles;

\c fussballverein04
GRANT ALL ON ALL TABLES IN SCHEMA PUBLIC TO darfalles;
```

[7]

USER `nurlesen` -> Darf überall lesen

```
GRANT CONNECT on DATABASE fussballverein to nurlesen;

GRANT CONNECT on DATABASE fussballverein02 to nurlesen;

GRANT CONNECT on DATABASE fussballverein03 to nurlesen;

GRANT CONNECT on DATABASE fussballverein04 to nurlesen;
```

[4]

Mit den Datenbanken verbinden und für den USER die Rechte SELECT (lesen) erteilen :

```
\c fussballverein

GRANT SELECT ON ALL TABLES IN SCHEMA PUBLIC TO nurlesen;

GRANT

\c fussballverein02

GRANT SELECT ON ALL TABLES IN SCHEMA PUBLIC TO nurlesen;

GRANT

\c fussballverein03

GRANT SELECT ON ALL TABLES IN SCHEMA PUBLIC TO nurlesen;

GRANT

\c fussballverein04

GRANT SELECT ON ALL TABLES IN SCHEMA PUBLIC TO nurlesen;
```

[3]

USER `test`

- Darf bei fussballverein03 schreiben/ lesen
- Darf bei fussballverein02 nur lesen
- Fussballverein und 04 nichts

```
GRANT CONNECT ON DATABASE fussballverein03 to test;
GRANT CONNECT ON DATABASE fussballverein02 to test;
\c fussballverein03
GRANT SELECT ON ALL TABLES IN SCHEMA PUBLIC TO test;
GRANT INSERT ON ALL TABLES IN SCHEMA PUBLIC TO test;
GRANT UPDATE ON ALL TABLES IN SCHEMA PUBLIC TO test;
\c fussballverein02
GRANT SELECT ON ALL TABLES IN SCHEMA PUBLIC TO test;
```

[4]

USER `test2`

- fussballverein04 lesen
- fussballverein02 schreiben
- fussballverein und 03 nichts

```
GRANT CONNECT ON DATABASE fussballverein04 to test2;
GRANT CONNECT ON DATABASE fussballverein02 to test2;
\c fussballverein04
GRANT SELECT ON ALL TABLES IN SCHEMA PUBLIC TO test2;
\c fussballverein02
GRANT INSERT ON ALL TABLES IN SCHEMA PUBLIC TO test2;
GRANT UPDATE ON ALL TABLES IN SCHEMA PUBLIC TO test2;
```

[4]

4.4 Testen der USER

USER `darfalles` mit fussballverein02 verbinden

```
postgres@debian:/root$ psql -d fussballverein02 -U darfalles
Password for user darfalles:
psql (9.5.1, server 9.4.5)
Type "help" for help.

fussballverein02=> \d
                    List of relations
 Schema |             Name              |   Type   |  Owner
--------+-------------------------------+----------+----------
 public | angestellter                  | table    | postgres
 public | angestellter_persnr_seq       | sequence | postgres
```

[1]

Die Tabellen der Datenbank werden angezeigt.

```
fussballverein02=> select * from person limit 10;
 persnr |       vname        |        nname         | geschlecht |   gebdat
--------+--------------------+----------------------+------------+------------
      1 | vname_85908_859    | nname_61407_614      | maennlich  | 1953-09-26
      2 | vname_31399_3139   | nname_26737_267      | weiblich   | 1995-09-30
```

Insert funktioniert

```
postgres@debian:/root$ psql -d fussballverein02 -U darfalles
Password for user darfalles:
psql (9.5.1, server 9.4.5)
Type "help" for help.

fussballverein02=> insert into person values(2000005, 'vname_2','nname_5','maennlich','1955-02-03');
INSERT 0 1
```

Die anderen Datenbanken haben ebenfalls funktioniert.

USER `nurlesen`:

Insert funktionier nicht, da er nur lesen darf:

```
postgres@debian:/root$ psql -d fussballverein02 -U nurlesen
Password for user nurlesen:
psql (9.5.1, server 9.4.5)
Type "help" for help.

fussballverein02=> insert into person values(2000004, 'vname_2','nname_5','maennlich','1955-02-03');
ERROR:  permission denied for relation person
```

Select Abfrage funktioniert:

```
fussballverein02=> insert into person values(2000004, 'vname_2','nname_5','maennli
ERROR:  permission denied for relation person
fussballverein02=> select * from person limit 10;
 persnr |       vname        |        nname         | geschlecht |   gebdat
--------+--------------------+----------------------+------------+------------
      1 | vname_85908_859    | nname_61407_614      | maennlich  | 1953-09-26
```

USER `test`:

- Darf bei fussballverein03 schreiben/ lesen
- Darf bei fussballverein02 nur lesen
- Fussballverein und 04 nichts

Verbunden mit fussballverein02:

INSERT funktioniert nicht: (da es nur lesen darf)

```
postgres@debian:/root$ psql -d fussballverein02 -U test
Password for user test:
psql (9.5.1, server 9.4.5)
Type "help" for help.

fussballverein02=> insert into person values(2000006, 'vname_2','nname_5','maennlich','1955-02-03');
ERROR:  permission denied for relation person
```

SELECT funktioniert:

```
fussballverein02=> insert into person values(2000006, 'vname_2','nname_5','maennl:
ERROR:  permission denied for relation person
fussballverein02=> select * from person limit 10;
 persnr |       vname        |        nname         | geschlecht |   gebdat
--------+--------------------+----------------------+------------+------------
      1 | vname_85908_859    | nname_61407_614      | maennlich  | 1953-09-26
```

Verbunden mit fussballverein03:
Insert funktioniert:

```
postgres@debian:/root$ psql -d fussballverein03 -U test
Password for user test:
psql (9.5.1, server 9.4.5)
Type "help" for help.

fussballverein03=> insert into person values(2000008, 'vname_2','nname_5','maennlich','1955-02-03');
INSERT 0 1
fussballverein03=>
```

SELECT funktioniert:

```
fussballverein03=> insert into person values(2000008, 'vname_2','nname_5','maennlich
INSERT 0 1
fussballverein03=> select * from person limit 10;
 persnr |       vname        |        nname         | geschlecht |   gebdat
--------+--------------------+----------------------+------------+------------
      1 | vname_85908_859    | nname_61407_614      | maennlich  | 1953-09-26
```

USER `test2`:

- fussballverein04 lesen
- fussballverein02 schreiben
- fussballverein und 03 nichts

Verbunden mit fussballverein02

`test2` kann nur schreiben:

```
postgres@debian:/root$ psql -d fussballverein02 -U test2
Password for user test2:
psql (9.5.1, server 9.4.5)
Type "help" for help.

fussballverein02=> insert into person values(1000009, 'vname_2','nname_5','maennlich','1955-02-03');
INSERT 0 1
fussballverein02=> select * from angestellter limit 10;
ERROR:  permission denied for relation angestellter
fussballverein02=>
```

Verbunden mit fussballverein04:

`test2` kann nur lesen:

```
postgres@debian:/root$ psql -d fussballverein04 -U test2
Password for user test2:
psql (9.5.1, server 9.4.5)
Type "help" for help.

fussballverein04=> insert into person values(10000012, 'vname_2','nname_5','maennlich','1955-02-03');
ERROR:  permission denied for relation person
fussballverein04=> select * from person limit 10;
 persnr |       vname        |        nname         | geschlecht |   gebdat
--------+--------------------+----------------------+------------+------------
      1 | vname_85908_859    | nname_61407_614      | maennlich  | 1953-09-26
      2 | vname_31399_3139   | nname_26737_267      | weiblich   | 1995-09-30
```

5 Literaturverzeichnis

[1] http://www.cyberciti.biz/faq/psql-fatal-ident-authentication-failed-for-user/

[2] http://www.informatik.uni-jena.de/dbis/lehre/ss2008/dbs/dbs_lehrer_12.pdf

[3] http://www.postgresql.org/docs/8.0/static/user-manag.html

[4] http://www.postgresql.org/docs/9.0/static/sql-grant.html

[5] http://www.postgresql.org/docs/9.1/static/transaction-iso.html

[6] https://de.wikipedia.org/wiki/Transaktion_(Informatik)

[7] http://dbs.uni-leipzig.de/buecher/DBSI-Buch/HTML/kap13-2.html

[8] https://de.wikipedia.org/wiki/Isolation_(Datenbank)

[9] https://en.wikipedia.org/wiki/Isolation_(database_systems)

[10] https://www.cri.ensmp.fr/people/coelho/datafiller

GITHUB REPOSITORY:
https://github.com/motahal/SEMESTERAUFGABE---FUSSBALLVEREIN--OTAHAL.git

6 Zeitaufzeichnung

Aufgabe	**Dauer**
ER-Diagramm	4 h
CREATE-Script / Datafiller	6 h
Benutzerverwaltung	2 h
Transaktionen	3 h
Protokoll	5 h
Gesamt:	**20 h**