

# Protokoll Insy Fußballverein

Ibrahim

[FIRMENNAME] [Firmenadresse]

## Inhalt

Aufgabenstellung.....	2
Erstellen eines ERD und RM.....	4
Installieren von PSQL und erstellen der Datenbank.....	4
<b>Erstellen des Create und Drop-Skriptes.....</b>	<b>6</b>
Inserts .....	7
<b>Was sind Transaktionen überhaupt ? .....</b>	<b>7</b>
<b>ACID .....</b>	<b>8</b>
<b>Befehle um eine Transaktionen zu starten.....</b>	<b>8</b>
<b>Isolationstufen und ihre Probleme .....</b>	<b>8</b>
<b>Non repeatable Read .....</b>	<b>9</b>
1 Transaktion .....	9
2 Transaktion .....	9
Transaktion 1 .....	10
<b>Dirty Read.....</b>	<b>10</b>
1 Transaktion .....	11
2 Transaktion .....	11
1 Transaktion .....	11
2 Transaktion .....	11
Phantom Read.....	12
1 Transaktion .....	12
2 Transaktion .....	12
1 Transaktion .....	12
<b>Lost Update .....</b>	<b>13</b>
<b>Sperren .....</b>	<b>13</b>
<b>Dead Locks.....</b>	<b>13</b>
<b>Live Locks .....</b>	<b>13</b>
Inserts.....	13
Inserts mit Java.....	14

# Aufgabenstellung

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 Beiträge ("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 ("gegruetet") 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 mehrere 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.) 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.

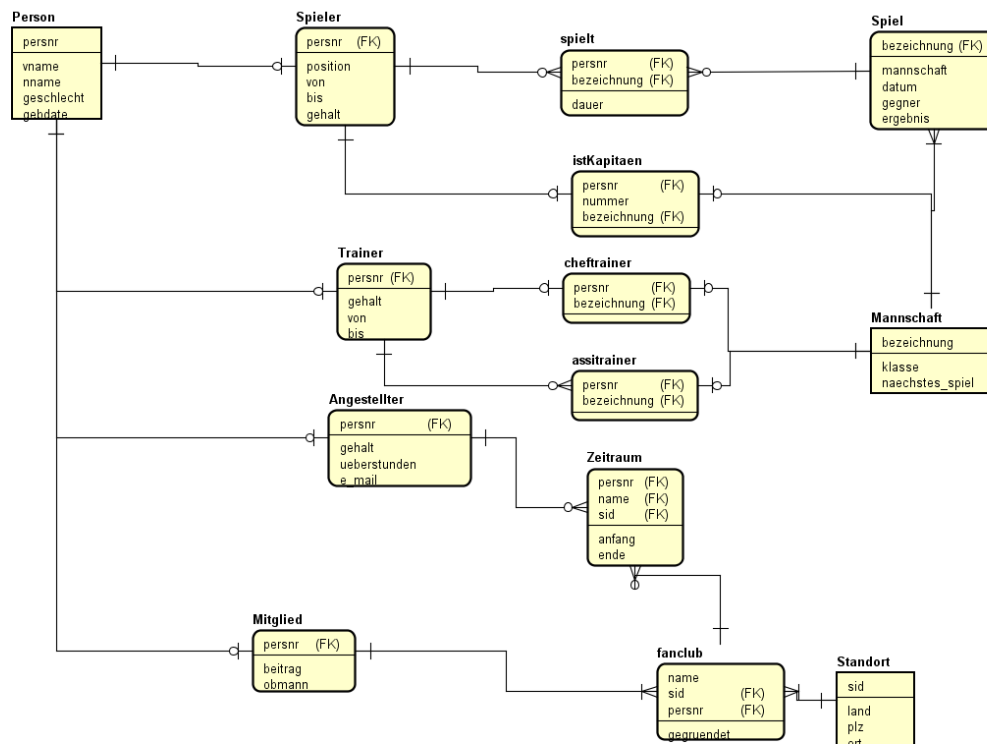
- \* Realisieren Sie folgende Attribute mit fortlaufenden Nummern mit Hilfe von Sequences: "persnr" von Personen und "sid" von Standorten. Für das "persnr"-Attribut sollen nur gerade, 5-stellige Zahlen vergeben werden (d.h.: 10000, 10002, ..., 99998). Für das "sid"-Attribut sollen beliebige positive 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.

## Erstellen eines ERD und RM



Die erste Aufgabe ist das wir ein ERD erstellen. Ich erstelle mein ERD mittels Astah.

Ich hab unabsichtlich das sid irgendwie geklont und ich weiß nicht wie ich es löschen soll.

Nach Diskussionen und Hilfe von Herr Wichert bin ich dazu gekommen die Mannschaft mit dem Spieler zu verbinden .

Grund :Mannschaft hat 11 Spieler. Mein Gedanke war das es schon klar wäre wenn ich nur Mannschaft und spiel verbinde.

## Installieren von PSQL und erstellen der Datenbank

Installieren von postgres und erstellen eines User der die Rechte hat um die Datenbank des Fußballvereines zu bearbeiten

Installieren und runterladen von Postgresql

apt-get install postgresql

```
i@ubuntu: ~  
i@ubuntu:~$ apt-get install postgresql
```

```
Processing triggers for ureadahead (0.100.0-16) ...  
Setting up postgresql-9.3 (9.3.11-0ubuntu0.14.04) ...  
Creating new cluster 9.3/main ...  
    config /etc/postgresql/9.3/main  
    data   /var/lib/postgresql/9.3/main  
    locale en_US.UTF-8  
    port   5432  
update-alternatives: using /usr/share/postgresql/9.3/man/man1/postmaster.1.gz to  
provide /usr/share/man/man1/postmaster.1.gz (postmaster.1.gz) in auto mode  
* Starting PostgreSQL 9.3 database server [ OK ]  
Setting up postgresql (9.3+154ubuntu1) ...  
Processing triggers for libc-bin (2.19-0ubuntu6.7) ...  
i@ubuntu:~$
```

Jetzt müssen wir mit root uns anmelden und dann ein user erstellen der die Datenbank aufrufen muss  
sudo su postgres

```
i@ubuntu: ~  
i@ubuntu:~$ sudo su postgres  
postgres@ubuntu:/home/i$ psql  
psql (9.3.11)  
Type "help" for help.  
  
postgres=#
```

Erstellen eines Users mit einem Passwort

CREATE USER fussUser WITH LOGIN PASSWORD 'fuss';

```
i@ubuntu: ~  
i@ubuntu:~$ sudo su postgres  
postgres@ubuntu:/home/i$ psql  
psql (9.3.11)  
Type "help" for help.  
  
postgres=# CREATE USER fussUser WITH LOGIN PASSWORD 'fuss';  
postgres=#
```

Erstellen der Datenbank und zu weisung des Users  
CREATE DATABASE fussballverein OWNER fussUser;

```
postgres'# CREATE DATABASE fussballverein OWNER fussUser;  
postgres'#
```

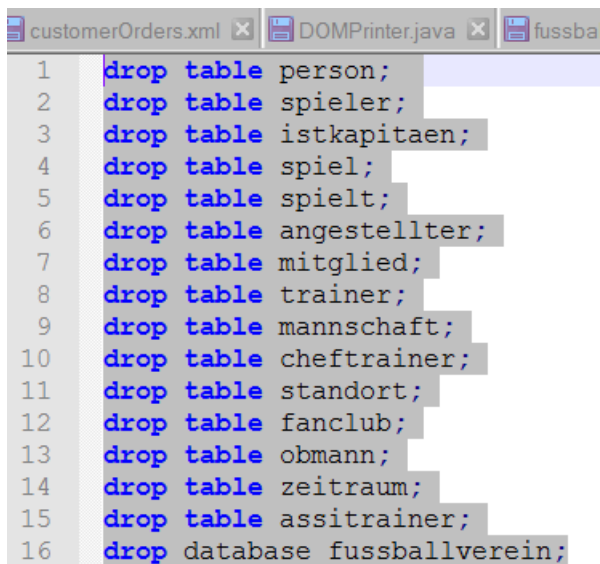
Vergeben von rechten an dem User des Fußballverein  
GRANT ALL PRIVILEGES ON DATABASE fussballverein TO fussUser;

Verbindung der Datenbank

```
psql -h localhost -U fussuser -W -d fussballverein
```

## Erstellen des Create und Drop-Skriptes

### 1)DROP-Befehle



```
customerOrders.xml x DOMPrinter.java x fussba  
1 drop table person;  
2 drop table spieler;  
3 drop table istkapitaen;  
4 drop table spiel;  
5 drop table spielt;  
6 drop table angestellter;  
7 drop table mitglied;  
8 drop table trainer;  
9 drop table mannschaft;  
10 drop table cheftrainer;  
11 drop table standort;  
12 drop table fanclub;  
13 drop table obmann;  
14 drop table zeitraum;  
15 drop table assitrainer;  
16 drop database fussballverein;
```

### 2)Erstellen der datenbank

#### Person

Drop befehle ausführen um zu schauen ob diese überhaupt funktionieren

## Inserts

Nach Absprache mit dem Nachhilfe lehrer wurde eine Lösung gefunden Java Insert generator. Es wurde viel im Internet recherchiert für eine Lösung.

Die Inserts musste in einem File geschrieben und geschrieben werden.

Lösung: Mit einem Bufferdwriter. Verzeichnis angegeben und eine Sql Datei erstellt im gewünschten Verzeichnis.

```
//Erzeugen eines Files
File file = new File("/users/Ibrahim/Desktop/TestJava/inserts_person.sql");

// if file doesnt exists, then create it
if (!file.exists()) {
    file.createNewFile();
}

FileWriter fw = new FileWriter(file.getAbsolutePath());
BufferedWriter bw = new BufferedWriter(fw);
//mit der schleife erzeuge ich zufaellige Geburstdaten und die Insert wobei ich sie in einem File speicher
for(int i=1;i< 100000;i++){
```

Die gewünschten 100.000 Inserts werden mit einer forschleife erzeugt und in das File geschrieben mit der Methode write. Außerdem habe ich durch Herrn Kanyildiz hilfe eine Methode gefunden wie ich ein zufälliges Datum generiere.

```
int dayOfYear = randBetween(1, gc.getActualMaximum(gc.DAY_OF_YEAR));
gc.set(gc.DAY_OF_YEAR, dayOfYear);

bw.write("insert into person(persnr,vname,nname,geschlecht,gebdat) values("
        + i+", " + "'Vorname' + i+", " + "'Nachname'+i+"+" + ",'" + M'+"+" + gc.get(gc.YEAR) + "-" + (gc.get(gc.MONTH) +
        ");" + System.getProperty("line.separator"));
//Test ob wirklich das geburstdatum zwischen 1900 und 2010 ist
System.out.println(gc.get(gc.YEAR) + "-" + (gc.get(gc.MONTH) + 1) + "-" + gc.get(gc.DAY_OF_MONTH));
}
bw.close();

System.out.println("Done");

} catch (IOException e) {
    e.printStackTrace();
}
}
/*
 *
 *
 */
public static int randBetween(int start, int end) {
    return start + (int) Math.round(Math.random() * (end - start));
}
```

## Transaktionen

### Was sind Transaktionen überhaupt ?



Das Wort Transaktion kommt aus dem lateinischen ,wobei man achten sollte das trans übersetzt (hin-)über und agere „treiben,handeln,führen“ . Transaktion nennt man Programmschritte ,die als eine logische Einheit betrachtet werden. Die Daten beziehungsweise die Datenbank ist nach diesem Vorgang konsistent.Transaktionen werden entweder vollständig und fehlerfrei oder garnicht ausgeführt.

Bei fehlerhaften Transaktionen werden diese abgebrochen und auf den letzten Zustand zurückgesetzt . => **Rollback**

## ACID

ACID beschreibt die Eigenschaften einer Transaktion . Diese muss eingehalten werden, damit eine Transaktion konsistent ist und damit es einzigartig und durchführbar sein muss .

**Atomicity:** Eine Transaktion wird entweder ganz oder gar nicht ausgeführt also sind die Transaktion unteilbar.Falls eine Transaktion abgebrochen wurde und es zu einem Rollback kommt ist das System unverändert.

**Consistency:** Nach der Transaktion muss die Datenbank in einer konsistentem Zustand sein wie am Anfang der Transaktion.

**Isolation:** Mehrere gleichzeitig ausgeführte Transaktionen dürfen sich nicht gegenseite beeinflussen

**Durability:**Transaktionen dürfe nicht verloren gehen oder mit der Zeit verblassen. =>dauerhafte Speicherung der Transaktionen auf dem Datenträger.

Diese Regel sollte man befolgen ,falls sich die Daten in einem redudanten Zustand befinden ist es schwer eine Reparatur zu machen .

## Befehle um eine Transaktionen zu starten

```
Begin      Beginn einer Transaktion
SET TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SERIALIZABLE
}
```

Setzte einer Isolationstufe einer Transaktion

Commit

Beenden einer Transaktion

Rollback

laufende Transaktion abbrechen

## Isolationstufen und ihre Probleme

Es wird für eine Transaktion eine Isalationsstufe definiert wird welcher Ausmaß von Ressourcen -oder Datenänderungen isoliert sein muss die von einem anderen Transaktionen druchgeführt werden . Diese Isolationsstufen werden beschrieben und welcher diese Fehlerfälle bei den Isolationsstufen auftreten.

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

## Non repeatable Read

Non repeatable read tritt beim Auslesen einer Datenbank auf. Wie man auf der Tabelle sieht entsteht dieser Problematik wenn man bei der Transaktion die Isolationsstufe auf Readuncommitted setzt oder auf Read committed.

Beginn der Transaktion mit dem Befehl **Begin** als nächstes setzt man die Isolationsstufe auf read uncommitted oder read committed.

Das macht man auch mit der 2. Transaktion. Außerdem wird dann committed mit dem Befehl **Commit** und wieder mit begin gestartet. Auch die Isolationsstufe von der 2. Transaktion.

In diesem konkreten Beispiel habe ich Read Committed benutzt

## 1 Transaktion

```
fussballverein=> Select * from person where persnr =40006 ;
persnr | vname   | nname   | geschlecht | gebdat
-----+-----+-----+-----+-----
  40006 | Clarissa | stefanovic | W          | 2001-02-02
(1 row)

fussballverein=> █
```

Setzen des Isolationslevel;

```
fussballverein=>
fussballverein=> begin
fussballverein-> ;
BEGIN
fussballverein=> Set TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET
fussballverein=> █
```

## 2 Transaktion

```
fussballverein=> begin;
BEGIN
fussballverein=> commit ;
COMMIT
fussballverein=> Set TRANSACTION ISOLATION LEVEL READ COMMITTED;
WARNING: SET TRANSACTION can only be used in transaction blocks
SET
fussballverein=> update person set geschlecht='M' where persnr = 40006;
UPDATE 1
fussballverein=> Select * from person where persnr =40006;
persnr | vname | nname | geschlecht | gebdat
-----+-----+-----+-----+-----
40006 | Clarissa | stefanovic | M | 2005-04-01
(1 row)

fussballverein=> █

fussballverein=> commit;
WARNING: there is no transaction in progress
COMMIT
fussballverein=>
fussballverein=> █
```

## Transaktion 1

```
SET
fussballverein=> Set TRANSACTION ISOLATION LEVEL READ COMMITTED;^C
fussballverein=> Select * from person where persnr =40006;
persnr | vname | nname | geschlecht | gebdat
-----+-----+-----+-----+-----
40006 | Clarissa | stefanovic | M | 2005-04-01
(1 row)

fussballverein=> █
```

Erklärung: Sobald die 1 Transaktion eine Select abfrage macht bekommt man den wert vom Select der 2 Transaktion.

## Dirty Read

Wie man in der Tabelle sieht kommt der Dirty Read nur bei Read uncommitted . Es tritt auf wenn eine Transaktion Daten aus einer Tabelle lesen darf aber eine andere Transaktion die Daten aus der Tabelle ändert die noch nicht committed wurden.

Man setzt eine Transaktion auf Read Uncommitted weil nur dann ist ein Dirty Read möglich . Wie beim vorigen Beispiel muss man die 1.Transaktion mit dem Befehl begin starten und die Isolationsstufe auf Read uncommitted setzen. Als nächstes muss man eine Änderung eines Datensatzes vornehmen mit Update .

Nebenbei sollte man die 2 Transaktion start und bei ihr auch die Isolationsstufe auf Read uncommitted setzen.

Der nächste Schritt ist eine Select abfrage um zu schauen ob die Veränderung übernommen worde .Falls sie nicht übernommen wurde macht man bei dem ersten ein commit und fragt man das gleiche wie oben ab und schaut man ob die Daten verändert worden sind.

## 1 Transaktion

```
fussballverein=> begin;
fussballverein-> ;
BEGIN
fussballverein=> Set TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SET
fussballverein=> Select * from person where persnr = 40006;
persnr | vname | nname | geschlecht | gebdat
-----+-----+-----+-----+-----
40006 | Clarissa | stefanovic | M | 2005-04-01
(1 row)

fussballverein=> update person set nname='pavic' where persnr=40006;
UPDATE 1
fussballverein=> █
```

## 2 Transaktion

```
fussballverein=>
fussballverein=> begin;
BEGIN
fussballverein=> Set TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SET
fussballverein=> Select * from person where persnr =40006;
persnr | vname | nname | geschlecht | gebdat
-----+-----+-----+-----+-----
40006 | Clarissa | stefanovic | M | 2005-04-01
(1 row)

fussballverein=> █
```

## 1 Transaktion

```
UPDATE 1
fussballverein=> commit;
COMMIT
fussballverein=> █
```

## 2 Transaktion

```
fussballverein=> Select * from person where persnr =40006;
persnr | vname | nname | geschlecht | gebdat
-----+-----+-----+-----+-----
40006 | Clarissa | pavic | M | 2005-04-01
(1 row)

fussballverein=> █
```

## Phantom Read

Wie man in der Tabelle sieht kommt der Phantom Read in allen Isolationsstufen außer Serializable vor.

Dies funktioniert erst dann wenn man Aggregationsfunktion bei einem Datensatz benutzt wie zum Beispiel (max,min,sum ..).

Bei diesem konkreten Beispiel startet man die 1 Transaktion und setzt je nach dem wie man will die Isolationsstufe bis auf Serializable. Nach diesen Schritten liest man den größten INT wert hinaus mit der Aggregation max.

Bei der 2 Transaktion startet man die Transaktion und setzt die Isolationslevel auf das gewünschte und macht ein Insert wobei der Int wert im neuen Datensatz höher ist als das vorige. Der nächste Schritt ist das man diese Transaktion committed.

Bei der ersten Transaktion führt man den gleichen Select befehl wie vorher aus und schaut ob diese dann größer ist als die vorige.

### 1 Transaktion

```
fussballverein=> begin;
BEGIN
fussballverein=> Set TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET
fussballverein=> Select max(persnr) from person;
      max
-----
      50000
(1 row)

fussballverein=> █
```

### 2 Transaktion

```
ussballverein=> begin;
EGIN
ussballverein=> Set TRANSACTION ISOLATION LEVEL READ COMMITTED;
ET
ussballverein=> insert into person(persnr,vname,nname,geschlecht,gebdat) values(50002,'Max
,'Musterman','M','1998-03-02');
INSERT 0 1
ussballverein=> commit;
COMMIT
ussballverein=> █
```

### 1. Transaktion

```
fussballverein=> Select max(persnr) from person;
max
-----
50002
(1 row)

fussballverein=> █
```

## Lost Update

Ein Lost update kann vor kommen wenn mehrere Transaktionen die Daten in der Datenbank verändern . Wenn zum Beispiel Transaktion A eine Update ausführt aber zur selben Zeit Transaktion B auch ein Update führt also in der Datenbank eine änderung macht,wird einfach die Änderung von Transaktion A verworfen(überschrieben).

## Sperren

Shared Locks  
Ermöglichen paralleles Lesen  
Verhindern von Änderung  
Exclusive Locks  
Sperrinhaber zu lesen und ändern

## Dead Locks

In einem DeadLock sind mehrer Transaktion involviert welche auf einen Sperre wartet die die andere Transaktion hält.

## Live Locks

Bei einem Livelock gibt es mehrer gesperrte Prozesse oder Transaktionen die nicht im selben Status verharren. Sie ändern ständig ihren Zustand. Also wartet nicht ein Prozesses oder bzw. eine Transaktion im wait-zustand.

Anschaulich kann man sich dazu zwei Personen vorstellen, die sich in einem Gang entgegenkommen und fortwährend versuchen, einander in der gleichen Richtung auszuweichen, und sich dabei trotzdem immer gegenseitig blockieren, während bei einem Deadlock sich die zwei Personen nur gegenüberstehen und jeweils darauf warten, dass die andere beiseite geht.  
=>Sidewalk shuffle

## Inserts

Mein erster Lösungsweg für die Insert wäre der Datafiller . Der Datafiller hat mir aber sehr viele Probleme zubereitet .

```

20
21 CREATE TABLE Person ( --df: mult=4.0
22   persnr Serial NOT NULL, --df: offset=10000 step=2 size=99999
23   vname VARCHAR(55), --df: text=vorname length=1
24   nname VARCHAR(55), --df: text=nachname length=1
25   geschlecht CHAR(1), --df: pattern='(M|W|N)'
26   gebdate DATE, --df: date start=1977-01-01 end=2005-01-01
27   CONSTRAINT PK_Person PRIMARY KEY (persnr)
28
29 );
30
31
32
33 CREATE TABLE Trainer ( --df: mult=1.0
34   persnr Serial NOT NULL REFERENCES Person(persnr), --df: sub=serand
35   gehalt int, --df: offset=10000 step=1 size=99999
36   von DATE, -- df: start=2000-01-01 end=2005-01-01
37   bis DATE, --df: end=2016-01-01
38   CONSTRAINT PK_Trainer PRIMARY KEY (persnr),
39   CONSTRAINT FK_Trainer_0 FOREIGN KEY (persnr) REFERENCES Person (persnr)
40 );
41
42 CREATE TABLE Standort ( --df: mult=1.0
43   sid INT NOT NULL, --df: offset=1
44   land VARCHAR(55), --df: text=land length=1
45   plz VARCHAR(55), --df: text=plz length=1
46   ort VARCHAR(55), --df: text=ort length=1

```

Der Datafiller arbeitet so das er sobald er ein --df: sieht generiert er je nach wunsch gewählten Datentyp Datensätze und so die Daten generiert. Primarykeys und Fremdschlüssel werden vom Datafiller erkannt wenn man eine Reference zu diesen Attributen und Tabellen macht. Der Datafiller kann auch eine Liste benutzen um Varchar extern mit verschiedenen Attributen zu befüllen.

Außerdem kann man Pattern benutzen die Attribut mit dem Bedingungen befüllt.

## Inserts mit Java

Ich hab die Inserts mittels Java generiert. Bei der Aufgabe habe ich enum Objekte verwendet weil es mir leichter fällt. Mittels einer Methode generieren ich eine Random Zahl diese bestimmt eine Zufällige Namen.

```

public static givenname randomGiv() {
    int pick = new Random().nextInt(givenname.values().length);
    return givenname.values()[pick];
}

```

Mit einer forschleife schreibe ich die Inserts in ein File und beachte dabei das nur die gerade Persn bekomme .

Da meine Insert größer ist als 50 mb kann ich diese nicht auf Model hochstellen und auch nicht mal als rar Datei abgeben also gebe ich ihnen die .java dateien ab.

Quelle: Transaktion Schulübung zum Kapitel Transaktion

## Select Abfragen

(Fan-Club Betreuung) Wählen Sie "per Hand" die Personalnummer eines Angestellten aus Ihren Testdaten aus. Schreiben Sie eine SQL-Anfrage, die jene Fan-Clubs ermittelt, die dieser Angestellte im Moment nicht betreut. Geben Sie zu jedem derartigen Fan-Club die Standort-ID und den Namen des Fan-Clubs aus.

Bemerkung: Ein Fan-Club wird von einem Angestellten im Moment nicht betreut, wenn entweder der Angestellte diesen Fan-Club überhaupt nie betreut hat oder wenn das heutige Datum (= sysdate) außerhalb des Betreuungszeitraums liegt. Vergessen Sie nicht, jene Fan-Clubs zu berücksichtigen, die von überhaupt keinem Angestellten betreut werden (dieser Fall sollte zwar laut Datenmodell nicht vorkommen. Die Einhaltung dieser Bedingung wird aber vermutlich vom Datenbanksystem nicht überprüft)!

```
Select Angestellter.persnr, anfang, ende from Zeitraum inner join Angestellter on
Angestellter.persnr=Zeitraum.persnr where anfang< current_date() and ende< current_date()
,(Select name sid from fanclub inner join Standort on Standort.sid=fanclub.sid);
```

Begründung: Der Grund wieso ich mir das gedacht habe ist derjenige, den ich persnr brauche und den Namen von dieser Person außerdem muss ich noch schauen, ob diese Person nicht tätig ist. Das habe ich gemacht, indem ich mit current\_date die Zeit abfrage und diese dann mit einem Vergleichsoperator vergleiche. Außerdem habe ich eine Subquery, die die sid und den Name vom fanclub abfragt.

S2.) (Die eifrigsten Angestellten) Schreiben Sie eine SQL-Anfrage, die den Nachnamen und die Personalnummer jener Angestellten ausgibt, die im Moment sämtliche Fan-Clubs betreuen. Ordnen Sie die Nachnamen alphabetisch.

Bemerkung: Passen Sie die Testdaten so an, dass diese Anfrage zumindest zwei Angestellte liefert.

```
select Person.nname, persnr from Angestellte inner join Person on Person.persnr
=Angestellte.persnr order by nname Asc, (Select persnr from fanclub inner join Angestellte on
Angestellte.persnr=fanclub.persnr);
```

Begründung: Die Begründung zu dieser Abfrage ist, dass die persnr brauche und dabei ein Inner join mit der Tabelle Spieler macht. Außerdem habe ich eine Subquery, mit der mir gelingt, eine join bekomme, in der ich die Daten von fanclub bekomme.

S4.) (Spieler-Ranking) Geben Sie für jeden Spieler den Vornamen und Nachnamen sowie die Gesamtdauer ("gesamtdauer") der von ihm bei Spielen im Jahr 2015 geleisteten Einsätze aus. Vergessen Sie nicht, jene Spieler des Vereins zu berücksichtigen, die im Jahr 2015 bei keinem einzigen Spiel mitgespielt haben (d.h. gesamtdauer = 0). Ordnen Sie die Ausgabe in absteigender Gesamtdauer. Bei Gleichheit der Gesamtdauer sollen die Spieler in alphabetischer Reihenfolge (zuerst des Nachnamen, dann des Vornamen) sortiert werden.

```
4) Select persnr, Person.vname, Person.nname from spielt inner join Person on Person.persnr=
Angestellte.persnr order by dauer Desc, nname Desc, (Select
Datepart(Spiel.datum, yyyy, '2015'), bezeichnung from spiel inner join Spieler on
Spiel.bezeichnung=spielt.bezeichnung);
```

Begründung: Die Begründung zu dieser Abfrage ist, dass die Vname, nname und persnr will. Dies wird mit einem Inner join von der Tabelle Person, außerdem habe nach 2 Sachen geordnet und die absteigen sind. Außerdem habe ich eine Subquery, die eine Datepart hat, in dem man das Jahr 2015 heraus kristallisiert, um die Spieler im Jahr 2015 gespielt haben. Diese Datein kommen von spiel. Wieso kein Outerjoin, weil ich die dauer mit 0 will und nicht mit null.



S5.)(Der fleißigste Spieler) Geben Sie den Vornamen und Nachnamen jenes Spielers aus, von dem die unter b) berechnete Gesamtdauer am größten ist, d.h.: dieser Spieler ist bei Spielen im Jahr 2015 insgesamt am längsten im Einsatz gewesen. Falls sich mehrere Spieler den ersten Platz teilen (d.h. sie kommen auf die gleiche Gesamtdauer), dann sollen diese in alphabetischer Reihenfolge (zuerst des Nachnamen, dann des Vornamen) geordnet werden. Der Fall, dass im Jahr 2015 überhaupt kein Spiel stattfand, darf ignoriert werden.

Bemerkung: Berücksichtigen Sie bei Ihren Testdaten die Situation, dass sich zumindest 2 Spieler den ersten Platz teilen.

```
5)Select Person.nname,Person.vname,person,max(dauer) from spielt inner join Person on
Person.persnr=spielt.persnr order by dauer Desc,(Select
datepart(Spiel.datum,yyyy,'2015'),bezeichnung from spielt inner join Spiel on
Spiel.bezeichnung=spielt.bezeichnung );
```

Begründung: Ich habe mir gedacht das ich die höchste zahl mit einem max bekomme und diese dann mit einer join dann bekomme .Subquerie habe ich gebraucht um das datum zu bekommen

S6.) Schreiben Sie CREATE und DROP Befehle für eine View, die alle Informationen über Trainer aus der Personen- und Trainer-Tabelle zusammenfügt, d.h.: sowohl die allgemeinen Personendaten (Personalnummer, Vorname, Nachname, Geschlecht und Geburtsdatum) als auch die Trainer-spezifischen Informationen (Gehalt sowie Beginn und Ende der Vertragsdauer). In Summe ist also folgende View erforderlich:

Trainer\_view (persnr, vname, nname, geschlecht, gebdat, gehalt, von, bis)

```
6)create view Trainer_view as Select
persnr,Person.vname,Person.nname,gehalt,von,bis,gebdate,geschlecht from Trainer inner join
Person on Person.persnr=Trainer.persnr;
Drop View Trainer_view
```

Quelle: <http://stackoverflow.com/questions/1972392/java-pick-a-random-value-from-an-enum>  
<http://stackoverflow.com/questions/3985392/generate-random-date-of-birth>

Datafiller: <http://blog.coelho.net/database/2013/12/01/datafiller-tutorial/>