Gruppe A

Bitte tragen Sie **sofort** und **leserlich** Namen, Studienkennzahl und Matrikelnummer ein und legen Sie Ihren Studentenausweis bereit.

PRÜFUNG AUS DATENBANKSYSTEME **VU 184.686**			12. 3. 2013
Kennnr.	Matrikelnr.	Familienname	Vorname

Arbeitszeit: 100 Minuten. Aufgaben sind auf den Angabeblättern zu lösen; Zusatzblätter werden nicht gewertet.

Aufgabe 1: Zeitstempel-basierende Synchronisation (10)
Sie finden in der unten angeführten Tabelle Schedules von zwei Transaktionen T_1 und T_2. Es gibt zwei Datenobjekte A und B, deren Anfangswerte für readTS und writeTS jeweils 0 sind. Die beiden Zeitstempel haben die folgenden Werte: TS(T1) = 5 und TS(T2) = 10. Verwenden Sie die Regel für Zeitstempel-basierende Synchronisation wie in der Vorlesung besprochen und geben Sie nun jeweils in den rechten vier Spalte die aktuellen Werte für readTS(A), writeTS(A), readTS(B) und writeTS(B) an.

#	T_1	T_2	readTS(A)	writeTS(A)	readTS(B)	writeTS(B)
1	SELECT A INTO valueA1		5	0	0	0 .....
2	valueA = valueA1 + 10		-	-	-	-
3	UPDATE A = valueA1		5	5	0	0 .....
4		SELECT A INTO valueA2	10	5	0	0 .....
5		valueA2 = valueA2 * 2	-	-	-	-
6		UPDATE A = valueA2	10	10	0	0 .....
7	SELECT B INTO valueB1		10	10	5	0 .....
8	valueB = valueB1 + 10		-	-	-	-
9	UPDATE B = valueB1		10	10	5	5 .....
10		SELECT B INTO valueB2	10	10	10	5 .....
11		valueB2 = valueB2 - 10	-	-	-	-
12		UPDATE B = valueB2	10	10	10	10 .....

Ist der Schedule serialisierbar?
⊗ Ja ◯ Nein

Konflikt in Zeile ..... bei Transaktion ..... Verletzte Regel: .....

Aufgabe 2: Kostenabschätzungen (20)

Eine große Pannendienst-Datenbank enthält folgende Relationen:

Mitarbeiter(MNr, Name, Geburtsjahr, Wohnort, TelNr) (kurz m),
Fahrzeug(FNr, Marke, Type, Baujahr, Zugkraft) (kurz f) und
Einsatz(MNr, FNr, Kategorie, Ort) (kurz e)

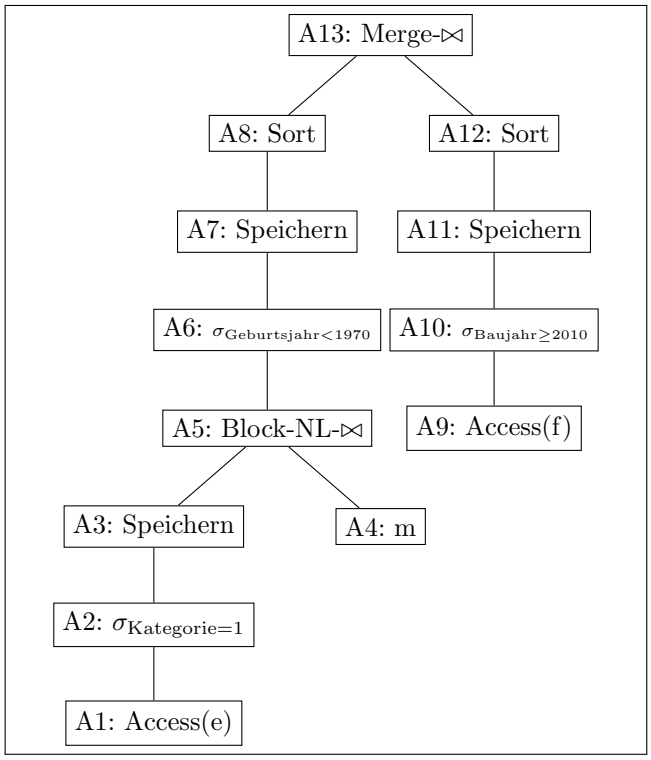
Nehmen Sie an, dass $|m| = 25000$, $|f| = 10000$, und $|e| = 750000$. Für die durchschnittlichen Tupelgrößen von m, f und e sind die Werte 120, 150 und 75 Bytes anzunehmen. Nehmen Sie weiters an, dass pro Seite 2500 Bytes an Nutzinformation gespeichert werden können und dass die Hauptspeicher-Puffergröße 256 Seiten beträgt. Es ist die Anfrage

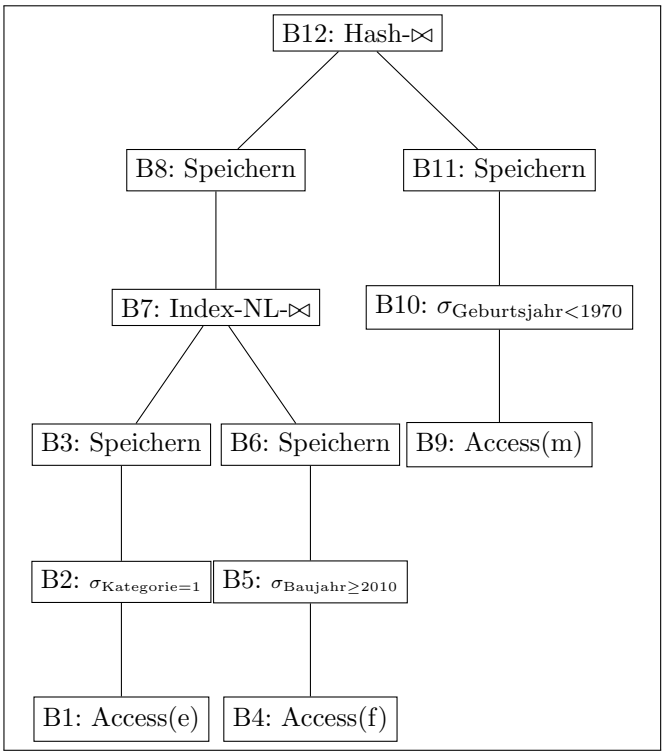
```
SELECT *
FROM Mitarbeiter m, Fahrzeug f, Einsatz e
WHERE e.MNr = m.MNr
AND e.FNr = f.FNr
AND m.Geburtsjahr < 1970
AND f.Baujahr >= 2010
AND e.Kategorie = 1;
```

auszuführen (d.h. gesucht sind Informationen über Mitarbeiter, die vor 1970 geboren wurden und ein Fahrzeug des Baujahres 2010 oder jünger zu einem Einsatz der Kategorie 1 gelenkt haben).

Es sind folgende Selektivitäten anzunehmen: $sel_{m/e} = 1/25000 = 0.00004$, $sel_{f/e} = 1/10000 = 0.0001$, $sel_{m.Geburtsjahr<1970} = 0.55$, $sel_{f.Baujahr>=2010} = 0.4$, und $sel_{e.Kategorie=1} = 0.1$. Für die Primärschlüssel der Relationen m und f sei jeweils ein Hash-Index vorhanden. Nehmen Sie an, dass das Auslesen eines einzelnen Tupels mit einem Hash-Index durchschnittliche Kosten von 1.8 Page I/O erfordert.

Für diese Anfrage sind die Operator-Bäume für 2 Auswertungspläne gegeben: Plan A im linken Kästchen und Plan B im rechten Kästchen. Nehmen Sie bei den Block Nested Loop Joins an, dass $k = 1$ gilt und dass die äußere Relation jeweils links im Baum steht – unabhängig davon, ob dies optimal ist oder nicht. Bei der Berechnung der benötigten Seiten zum Speichern einer Relation dürfen Sie vereinfachend annehmen, dass die Tupel nicht unbedingt vollständig auf einer Seite Platz haben müssen.





(a) Berechnen Sie für jeden Knoten im Operatorbaum des Auswertungsplans A eine Abschätzung für die Anzahl der Tupel im Resultat, die Tupelgröße, die Anzahl der Seiten im Resultat, und die Kosten (Page I/O). Für das Sortieren und für Joinoperationen ist auch noch die passende Kostenformel anzugeben. Tragen Sie Ihre Berechnungen in folgende Tabelle ein.

Knoten# i	Anzahl Tupel T_i	Tupel-größe g_i	Anzahl Seiten b_i	Kostenformel	Kosten (Page I/O)
A1	750000	75	22500	-	22500 .....
A2	75000	75	2250	-	0 .....
A3	75000	75	2250	-	2250 .....
A4	25000	120	1200	-	0 .....
A5	75000	195	5850	$b_3 + k + \lceil b_3/(m-k-1) \rceil * (b_4 - k)$	13042 .....
A6	41250	195	3218	-	0 .....
A7	41250	195	3218	-	3218 .....
A8	41250	195	3218	$2 * b_7 * (1 + I)$ mit ..... $I = \lceil \log_{255}(\lceil b_7/256 \rceil) \rceil = 1$	12872 .....
A9	10000	150	600	-	600 .....
A10	4000	150	240	-	0 .....
A11	4000	150	240	-	240 .....
A12	4000	150	240	$2 * b_{11} * (1 + I)$ mit ..... $I = \lceil \log_{255}(\lceil b_{11}/256 \rceil) \rceil = 0$	480 .....
A13	16500	345	2277	$(b_8 + b_{12})$	3458 .....

Kosten insgesamt (Page I/O):

22500 + 2250 + 13042 + 3218 + 12872 + 600 + 240 + 480 + 3458 = 58660 .....

(b) Berechnen Sie für jeden Knoten im Operatorbaum des Auswertungsplans B eine Abschätzung für die Anzahl der Tupel im Resultat, die Tupelgröße, die Anzahl der Seiten im Resultat, und die Kosten (Page I/O). Für die Joinoperationen ist auch noch die passende Kostenformel anzugeben. Tragen Sie Ihre Berechnungen in folgende Tabelle ein.

Knoten# i	Anzahl Tupel T_i	Tupel-größe g_i	Anzahl Seiten b_i	Kostenformel	Kosten (Page I/O)
B1	750000	75	22500	-	22500 .....
B2	75000	75	2250	-	0 .....
B3	75000	75	2250	-	2250 .....
B4	10000	150	600	-	600 .....
B5	4000	150	240	-	0 .....
B6	4000	150	240	-	240 .....
B7	30000	225	2700	$b_3 + 1.8 * T_3$	137250 ....
B8	30000	225	2700	-	2700 .....
B9	25000	120	1200	-	1200 .....
B10	13750	120	660	-	0 .....
B11	13750	120	660	-	660 .....
B12	16500	345	2277	$3 * (b_8 + b_{11})$	10080 .....

Kosten insgesamt (Page I/O):

22500 + 2250 + 600 + 240 + 137250 + 2700 + 1200 + 660 + 10080 = 177480 .....

Aufgabe 3: (15)

Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind.

1. 2PL (Two-Phase-Locking) garantiert Serialisierbarkeit. wahr ⊗ falsch ◯
2. Bei den Einstellungen ¬steal/force ist kein Undo möglich, aber ein Redo sehr wohl. wahr ◯ falsch ⊗
3. Eine Relation R sei an 6 Netzwerk-Knoten materialisiert mit den Gewichten 10, 50, 40, 70, 30 und 20. Dann sind $Q_r(R) = 120$ und $Q_w(R) = 100$ gültige Lese- bzw. Schreib-Quoren. wahr ◯ falsch ⊗
4. Objektrelationale Datenbanken ermöglichen es Relationen zu verschachteln. wahr ⊗ falsch ◯
5. Für die Klassenhierarchie von Historien gilt $SR \subseteq ST \subseteq ACA \subseteq RC$. wahr ◯ falsch ⊗
6. Vertikale Fragmentierung gewährleistet Rekonstruierbarkeit. wahr ◯ falsch ⊗
7. Geht bei einem VDBMS mit 2PC eine Commit-Nachricht des Koordinators verloren, dann ist der Agent blockiert. wahr ⊗ falsch ◯
8. Betrachten Sie zwei Relationen $R(ABC)$ und $S(ABD)$. Dann gilt auf jeden Fall folgende Gleichheit: $\pi_{AB}(R \bowtie S) = \pi_{AB}(R) \cap \pi_{AB}(S)$. wahr ⊗ falsch ◯
9. Betrachten Sie zwei Relationen $R(\underline{A}B)$ mit 5000 Tupeln und $S(AC)$ mit 3000 Tupeln, wobei A ein Fremdschlüssel von der Relation S auf den Primärschlüssel A in R ist. Dann ergibt der Ausdruck $R \bowtie S$ exakt 3000 Tupel. wahr ⊗ falsch ◯
10. Die topologische Sortierung eines azyklischen, gerichteten Serialisierbarkeitsgraphs ist eindeutig, wenn ein Subgraph einen Baum bildet. wahr ◯ falsch ⊗

(Pro korrekter Antwort 1.5 Punkte, **pro inkorrekter Antwort -1.5 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

Die folgende Datenbankbeschreibung gilt für die Aufgaben 5 - 8:
Gegeben ist folgendes stark vereinfachtes Datenbankschema zur Analyse der Spieler eines Fussballvereins :

spieler(snr, name, stufe)
begegnung(bnr, typ, gegner) und
teilnahme(snr: *spieler.snr*, bnr: *begegnung.bnr*, tore)

Jeder Spieler (`spieler`) hat eine eindeutige Spielernummer (`snr`), einen vollständigen Namen (`name`) und eine Leistungseinstufung (`stufe`). Die Tabelle `teilnahme` enthält für jeden Spieler (`snr`) die Begegnungen, bei denen er teilgenommen hat (`bnr`) und die Summe der Tore (`tore`) die er dabei geschossen hat. Jede Begegnung (`begegnung`) enthält eine eindeutige Nummer (`bnr`), den Spieltyp (`typ`) und den Namen der gegnerischen Mannschaft (`gegner`). Der Spieltyp ist entweder 'Heimspiel' oder 'Auswärtsspiel'. Wählen Sie entsprechende Datentypen (Integer, Varchar) für die Attribute.

Aufgabe 4: SQL (6)
Geben Sie nun SQL Create Statements an, um die drei Tabellen zu erstellen.

```
CREATE TABLE spieler (
  snr INTEGER PRIMARY KEY,
  name VARCHAR(40),
  stufe INTEGER
);

CREATE TABLE begegnung (
  bnr INTEGER PRIMARY KEY,
  typ VARCHAR(20) CHECK (typ IN ('Heimspiel', 'Auswärtsspiel')),
  gegner VARCHAR(50)
);

CREATE TABLE teilnahme (
  snr INTEGER REFERENCES spieler(snr),
  bnr INTEGER REFERENCES begegnung(bnr),
  tore INTEGER,
  PRIMARY KEY (snr, bnr)
);
```

Erstellen Sie nun eine Sequence `begegnung_sequence`, die für den Primärschlüssel der Tabelle `begegnung` Werte ab 2500 mit Erhöhung um jeweils 100 vergeben soll. Bei Überschreiten der Grenze des Datentyps soll wieder bei 2500 begonnen werden.

```
CREATE SEQUENCE begegnung_sequence START WITH 2500 INCREMENT BY 100 CYCLE;
```

Aufgabe 5: SQL Abfragen (6)

Gegeben sind folgende Tabellen:

spieler		
snr	**name**	**stufe**
1	'A'	120
2	'B'	100
3	'C'	140
4	'D'	130

begegnung		
bnr	**typ**	**gegner**
1	'Heimspiel'	'A'
2	'Auswaertsspiel'	'B'
3	'Auswaertsspiel'	'C'
4	'Heimspiel'	'D'

teilnahme		
snr	**bnr**	**tore**
1	1	4
3	1	0
4	1	1
1	2	1
3	2	0
1	3	1
4	3	0
4	4	1

Geben Sie die Ergebnisse für die unten folgenden *select*-Statements an. Falls mehrere Tupel zurückgegeben werden, geben Sie diese in beliebiger Reihenfolge an.

```
SELECT COUNT(snr) FROM teilnahme WHERE tore != 0
```

```
5
```

```
SELECT COUNT(*) FROM spieler, begegnung
```

```
16
```

```
SELECT SUM(tore) FROM teilnahme WHERE snr IN
(SELECT snr FROM spieler WHERE stufe >= 110)
```

```
8
```

```
SELECT SUM(tore) FROM teilnahme GROUP BY snr ORDER BY snr
```

```
6 0 2
```

Aufgabe 6: PL/SQL Trigger (9)

Erstellen Sie einen Trigger, der aufgerufen wird, bevor ein Tupel in die `teilnahme`-Tabelle eingefügt wird. Geben Sie für jeden neuen Eintrag den Ihr Trigger verarbeitet eine `NOTICE` aus, die die Summe der bisherigen Tore des betrachteten Spielers aus allen Begegnungen sowie seinen Namen enthält. Diese Nachricht könnte wie folgt aussehen:

Max Muster hat insgesamt 5 Tore erzielt.

Ist die Summe der Tore kleiner als 0 – der Spieler hat mehr Eigentore als Tore geschossen – soll die Leistungsstufe des Spielers (Attribut `stufe` in der Tabelle `spieler`) durch den Trigger auf 0 zurückgesetzt werden.

```
CREATE FUNCTION leistungsPruefung() RETURNS trigger AS $$
DECLARE
  toreSumme INTEGER;
  spielerName VARCHAR(40);
BEGIN
  SELECT s.name INTO spielerName FROM spieler s WHERE s.snr = NEW.snr;
  SELECT SUM(tore) INTO toreSumme FROM teilnahme t WHERE t.snr = NEW.snr;

  RAISE NOTICE spielerName || ' hat insgesamt ' || toreSumme || ' Tore erzielt.';

  IF toreSumme < 0 THEN
    UPDATE spieler SET stufe = 0 WHERE s.snr = NEW.snr;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER insertTrigger
BEFORE INSERT ON teilnahme FOR EACH ROW
  EXECUTE PROCEDURE leistungsPruefung();
```

Aufgabe 7: Java (9)

Vervollständigen Sie die Java Methode `matchStatistik`, die für einen gegebenen Spieler (`snr`):

- den Namen und die Summe der Tore dieses Spielers aus all seinen unterschiedlichen Begegnungen ausgibt
- die Summe der Tore, die er in diesen Spielen erzielt hat (`tore`) als neue Leistungsstufe (`stufe`) einträgt

Beispielsweise soll eine Nachricht in folgender Form erstellt werden:

Max Muster hat bisher 20 Tore erzielt.

Beachten Sie dazu folgende Hinweise:

- um Fehlerbehandlung und die genaue Formatierung der Ausgabe brauchen Sie sich nicht zu kümmern
- Sie können davon ausgehen, dass eine Connection c zur Verfügung steht
- Verwenden Sie zur Durchführung ausschliesslich PreparedStatements. Diese werden außerhalb der Methode angelegt, und können bei jedem Aufruf der Methode verwendet werden. Verwenden Sie der Übersichtlichkeit halber für die beiden Teile der Aufgabe (Namen und Summe der Tore berechnen, Aktualisieren der Leistungsstufe) separate PreparedStatements.

Legen Sie nun die für die Methode `matchStatistik` benötigten PreparedStatements an. (PreparedStatement ps = ...)

```
PreparedStatement ps1 = c.prepareStatement
  ('SELECT name, SUM(tore) summe FROM spieler
    JOIN teilnahme ON teilnahme.snr = spieler.snr
    WHERE spieler.snr = ? GROUP BY spieler.snr');

PreparedStatement ps2 = c.prepareStatement
  ('UPDATE spieler SET stufe = ? WHERE snr = ?');
```

Vervollständigen Sie nun die Methode `matchStatistik`. Sie können dabei auf alle eben definierten PreparedStatements zugreifen.

```
public void matchStatistik(int snr) throws Exception {
  ps1.setInt(1, snr);

  ResultSet rs = ps1.executeQuery();

  rs.next();

  System.out.println(rs.getString('name') + ' hat bisher ' +
      rs.getInt('summe') + ' Tore erzielt.');

  ps2.setInt(1, rs.getInt('summe'));
  ps2.setInt(2, snr);
  ps2.executeUpdate();

  rs.close();
}
```

Legen Sie zum Abschluss noch ein PreparedStatement an, das als Parameter zwei Strings (Spieltyp und Gegner) erhält und eine neue Begegnung einfügt und dabei die Sequence `begegnung_sequence` für die Vergabe des Primärschlüssels benutzt. (PreparedStatement ps = ...)

```
PreparedStatement ps = c.prepareStatement
  ('INSERT INTO begegnung (bnr, typ, gegner) ' +
    'VALUES (nextval(begegnung_sequence),?,?))');
```

Gesamtpunkte: 75