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**			18. 1. 2013
Kennnr.	Matrikelnr.	Familienname	Vorname

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

Aufgabe 1: (14)
Zeitstempel-basierende Synchronisation. Sie finden in der unten angeführten Tabelle eine Historie 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 Zeitstempel für die beiden Transaktionen sind TS(T1) = 5 und TS(T2) = 10. Verwenden Sie die Regeln für Zeitstempel-basierende Synchronisation und geben Sie nun jeweils in den rechten vier Spalten die aktuellen Werte für readTS(A), writeTS(A), readTS(B) und writeTS(B) in jeder Zeile an.

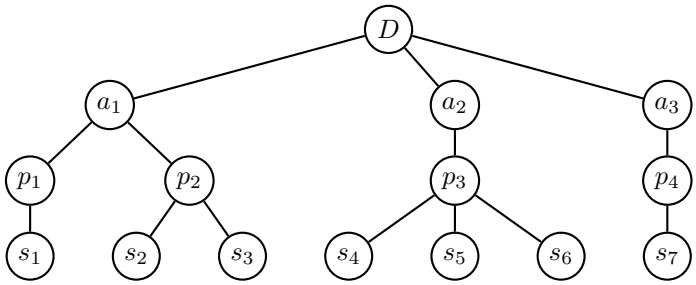
#	T_1	T_2	readTS(A)	writeTS(A)	readTS(B)	writeTS(B)
1	SELECT A INTO valueA1		5	0	0	0 .....
2	valueA = valueA1 + 10		-	-	-	-
3	UPDATE A = valueA1		5	5	0	0 .....
4		SELECT A INTO valueA2	10	5	0	0 .....
5		valueA2 = valueA2 * 2	-	-	-	-
6		UPDATE A = valueA2	10	10	0	0 .....
7		SELECT B INTO valueB1	10	10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

Ist die Historie serialisierbar? ◯ Ja ⊗ Nein

Falls die Historie nicht serialisierbar ist, begründen Sie durch Angabe folgender Kriterien:

Konflikt in Zeile 10 ..... bei Transaktion 1 ..... Verletzte Regel: $\mathrm{TS}(T_1) < \mathrm{readTS}(B)$

Aufgabe 2: (8)
Multi-Granularity Locking. Betrachten Sie folgende Datenbasis-Hierarchie.



Beantworten Sie, welche der folgenden geplanten Sequenzen von Sperr-Anforderungen (bei zwei Transaktionen T_1 und T_2) zu Blockierungen bzw. Deadlocks führen. (Hier bedeutet (T_i, x, L), dass Transaktion T_i versucht, Knoten x in der Hierarchie mit einer Sperre vom Typ L zu belegen.)

Hinweis: Unter Umständen werden nicht alle Sperren dieser Sequenzen auch tatsächlich angefordert, d.h.: Im Falle einer Blockierung einer Transaktion werden die weiter hinten liegenden Sperr-Anforderungen dieser Transaktion gar nicht mehr durchgeführt.

1. (T_1, D, IS), (T_2, D, IX), (T_1, a_1, IS), (T_1, p_1, IS), (T_2, a_1, IX), (T_2, p_2, X), (T_1, s_1, S):
   Blockierung: ja ◯ nein ⊗ Deadlock: ja ◯ nein ⊗
2. (T_1, D, IX), (T_2, D, IX), (T_1, a_3, IX), (T_2, a_2, IX), (T_1, p_4, IX), (T_2, p_3, IX), (T_1, s_7, X), (T_2, s_5, X):
   Blockierung: ja ◯ nein ⊗ Deadlock: ja ◯ nein ⊗
3. (T_1, D, IX), (T_2, D, IX), (T_1, a_1, IX), (T_2, a_2, X), (T_1, p_2, X), (T_2, a_1, IX), (T_2, p_2, IX), (T_2, s_2, X):
   Blockierung: ja ⊗ nein ◯ Deadlock: ja ◯ nein ⊗
4. (T_1, D, IS), (T_2, D, IX), (T_1, a_1, S), (T_2, a_2, IX), (T_2, p_3, X), (T_1, a_2, S), (T_2, a_1, IX), (T_2, p_1, IX), (T_2, s_1, X):
   Blockierung: ja ⊗ nein ◯ Deadlock: ja ⊗ nein ◯

(Pro korrekter Antwort 1 Punkt, **pro inkorrekter Antwort -1 Punkt**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

Aufgabe 3: (8)
Logische Optimierung. Beweisen oder widerlegen Sie folgende Äquivalenzen in der relationalen Algebra:

(a) $\pi_\ell(R_1 \cup R_2) = \pi_\ell(R_1) \cup \pi_\ell(R_2)$ [4]

Bei Schemagleichheit von R_1 und R_2 gilt die Äquivalenz. Sei $t \in \pi_\ell(R_1 \cup R_2)$. Dann existiert ein Tupel $x \in (R_1 \cup R_2)$ mit $x.l = t$. Das wiederum gilt genau dann wenn entweder ein Tupel $y \in R_1$ mit $y.l = t$ oder ein Tupel $z \in R_2$ mit $z.l = t$ vorhanden ist. Der Ausdruck $y \in R_1$ mit $y.l = t$ bedeutet aber auch dass $\exists y' \in \pi_\ell(R_1)$ mit $y' = t$ (analog für z mit R_2). Diese Teilausdrücke kann man wieder rückübersetzen in $t \in \pi_\ell(R_1)$ oder $t \in \pi_\ell(R_2)$, was wiederum genau der Vereinigung ($\cup$) entspricht.

(b) $\pi_\ell(R_1 \setminus R_2) = \pi_\ell(R_1) \setminus \pi_\ell(R_2)$ [4]

Wir konstruieren eine Gegenbeispiel. Seien R_1 und R_2 jeweils eine Tabelle mit den Spalten A und B. Tabelle R_1 sei gefüllt mit dem Tupel (a, b) und Tabelle R_2 mit (b, b). Dann gilt $\pi_B(R_1 \setminus R_2) = \{(b)\}$, aber $\pi_B(R_1) \setminus \pi_B(R_2) = \emptyset$.

Aufgabe 4: (15)

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

1. Ein geballter Index eignet sich vor allem bei Bereichsabfragen. wahr ⊗ falsch ◯
2. 2PL (Two-Phase-Locking) garantiert Rücksetzbarkeit. wahr ◯ falsch ⊗
3. Eine Relation R sei an 5 Netzwerk-Knoten materialisiert mit den Gewichten 20, 50, 30, 70 und 30. Dann sind $Q_r(R) = 90$ und $Q_w(R) = 110$ gültige Lese- bzw. Schreib-Quoren. wahr ◯ falsch ⊗
4. Seien R(AB), S(BC) zwei Relationen. Es gilt auf jeden Fall $R \bowtie S = (R \ltimes_{R.B=S.B} S) \bowtie_{R.B=S.B} S$. wahr ⊗ falsch ◯
5. PL/pgSQL ist eine rein deklarative Sprache. wahr ◯ falsch ⊗
6. Eine Hash-Funktion weist einem Schlüsselwert genau ein bestimmtes Bucket zu. wahr ⊗ falsch ◯
7. B-Bäume garantieren logarithmische Zugriffszeiten. wahr ⊗ falsch ◯
8. Bei den Einstellungen ¬steal/¬force ist kein Undo möglich, aber ein Redo sehr wohl. wahr ⊗ falsch ◯
9. Für die Klassenhierarchie von Historien gilt $ACA \subseteq ST \subseteq RC$. wahr ◯ falsch ⊗
10. Die Historie $w_1(C)$, $r_2(C)$, $r_2(D)$, $w_1(D)$, $w_2(A)$, c_1, c_2 ist serialisierbar und nicht strikt. 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 - 7: Gegeben ist folgendes stark vereinfachtes Datenbankschema für ein Unternehmen:

mitarbeiter(mid, name, gehalt, etage, abteilung: *abteilung.aid*) und
abteilung(aid, name, leiter: *mitarbeiter.mid*)

Jeder Mitarbeiter hat eine eindeutige ID (`mid`), einen vollständigen Namen (`name`), ein Gehalt (`gehalt`), arbeitet in einer bestimmten Etage (`etage`) und in einer bestimmten Abteilung (`abteilung`). Die Tabelle `abteilung` enthält für jede Abteilung eine eindeutige ID (`aid`), einen Namen (`name`) und die ID des Abteilungsleiters (`leiter`).
Der Wertebereich für das Attribut `etage` soll zwischen -2 und 5 liegen. Der Datentyp für das Gehalt soll zwei Nachkommastellen aufweisen. Wählen Sie entsprechende Datentypen (Integer, Varchar) für die restlichen Attribute.

Aufgabe 5: SQL (6)
Geben Sie die Create Table Statements mit allen nötigen Constraints für die Tabellen `mitarbeiter` und `abteilung` an. Beachten Sie eventuelle zyklische Abhängigkeiten zwischen den Fremdschlüsseln.

```
CREATE TABLE mitarbeiter (
  mid INTEGER PRIMARY KEY,
  name VARCHAR(50),
  gehalt NUMERIC(7,2),
  etage INTEGER check(etage between -2 and 5),
  abteilung INTEGER
);

CREATE TABLE abteilung (
  aid INTEGER PRIMARY KEY,
  name VARCHAR(50),
  leiter INTEGER
);

ALTER TABLE mitarbeiter
  ADD CONSTRAINT fk_abteilung
    FOREIGN KEY(abteilung) REFERENCES abteilung (aid) DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE abteilung
  ADD CONSTRAINT fk_leiter
    FOREIGN KEY(leiter) REFERENCES mitarbeiter (mid) DEFERRABLE INITIALLY DEFERRED;
```

Aufgabe 5: SQL Abfragen (6)

Gegeben sind folgende Tabellen:

mitarbeiter				
mid	**name**	**gehalt**	**etage**	**abteilung**
1	Huber	2000.00	1	1
2	Mayer	1500.00	2	3
3	Hofer	1850.00	3	3
4	Gruber	2900.00	1	4
5	Lackner	900.00	-1	2
6	Grabner	4500.00	NULL	2
7	Walder	5500.00	NULL	2
8	Faller	800.00	-1	2

abteilung		
aid	**name**	**leiter**
1	Management	1
2	Entwicklung	7
3	Marketing	3
4	Verkauf	4

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(*) FROM mitarbeiter, abteilung;
```

```
32
```

```
SELECT m.mid FROM mitarbeiter m
WHERE gehalt <= ALL
   (SELECT gehalt FROM mitarbeiter);
```

```
8
```

```
SELECT m.mid FROM mitarbeiter m, abteilung a
WHERE m.abteilung = a.aid
AND a.name ='Entwicklung' AND m.etage >= ALL (
   SELECT m2.etage FROM mitarbeiter m2, abteilung a2
   WHERE m2.abteilung = a2.aid
   AND a2.name = 'Entwicklung' AND m2.etage IS NOT NULL);
```

```
5 8
```

Aufgabe 7: Java (9)

Vervollständigen Sie die Java Methode `printStatistik`, die für jede Abteilung ausgibt, wie viel die Mitarbeiter in der jeweiligen Abteilung in Summe verdienen und zwar aufgegliedert nach den Etagen in denen Sie arbeiten. Zusätzlich ist noch die Gesamtsumme des Gehalts dieser Abteilung auszugeben. Sortieren Sie nach Abteilungsname aufsteigend.

Beachten Sie folgende Beispielausgabe der Abteilung 'Marketing':
`Marketing:`
`> 3: 1850.0`
`> 2: 1500.0`
`Summe: 3350.0`
Die Mitarbeiter der Abteilung 'Marketing', welche sich in der Etage 3 befinden, verdienen in Summe also 1850.0.

Vervollständigen Sie die Methode `printStatistik` in der Form, dass für alle Abteilungen diese Art der Ausgabe erreicht wird. Schließen Sie jene Mitarbeiter von dieser Statistik aus, welche keine zugewiesene Etage haben.

Beachten Sie dazu folgende Hinweise:

- um Fehlerbehandlung und die genaue Formatierung der Ausgabe brauchen Sie sich nicht zu kümmern
- Verwenden Sie zur Ausführung von SQL Ausdrücken ausschliesslich die beiden unten angegebenen PreparedStatements. Diese werden außerhalb der Methode angelegt, und können bei jedem Aufruf der Methode verwendet werden.

Vervollständigen Sie nun die für die Methode `printStatistik` benötigten PreparedStatements.

```
PreparedStatement abteilungen = c.prepareStatement('select aid, name from abteilung
  order by name;');

PreparedStatement etagenSummen = c.prepareStatement('select etage, sum(gehalt) as summe
  from mitarbeiter where abteilung = ? and etage is not null group by etage;');
```

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

```
public void printStatistik() throws Exception {

  ResultSet rs = abteilungen.executeQuery();
  while (rs.next()) {
    int aid = rs.getInt('aid');
    double sum = 0;
    System.out.println(rs.getString('name') + ':');

    stmt2.setInt(1, aid);
    ResultSet rs2 = pstmt2.executeQuery();

    while(rs2.next()) {
      sum += rs2.getDouble('summe');
      System.out.println('> ' + rs2.getInt('etage') + ': ' + rs2.getDouble('summe'));
    }
    System.out.println('Summe: ' + sum);
    rs2.close();
  }

  rs.close();
}
```

Aufgabe 8: PL/SQL Trigger (9)

Nehmen Sie an, dass eine Datenbank wie folgt definiert wurde.

```
CREATE TABLE messwerte (
  id INTEGER PRIMARY KEY,
  typ INTEGER NOT NULL,
  wert INTEGER
);

CREATE FUNCTION fTrigger1()
RETURNS trigger AS $$
BEGIN
  CASE NEW.typ
    WHEN 1 THEN
      DELETE FROM messwerte WHERE wert > 1;
    WHEN 3 THEN
      UPDATE messwerte
       SET wert = wert - 1 WHERE typ < 3;
    WHEN 2 THEN
      UPDATE messwerte
       SET typ = typ - 1,
           wert = wert + 1
       WHERE typ = 2;
    ELSE
      NULL;
  END CASE;
  NEW.wert = 0;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION fTrigger2()
RETURNS trigger AS $$
BEGIN
  IF (NEW.wert != OLD.wert) THEN
    NEW.typ = OLD.typ + 1;
    NEW.wert = OLD.wert + 1;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tTrigger1
AFTER INSERT ON messwerte
FOR EACH ROW EXECUTE PROCEDURE fTrigger1();

CREATE TRIGGER tTrigger2
BEFORE UPDATE ON messwerte
FOR EACH ROW EXECUTE PROCEDURE fTrigger2();
```

Geben Sie an, wie die Tabelle `messwerte` nach jedem der folgenden Befehlsblöcke aussieht.

```
INSERT INTO messwerte VALUES (1, 2, 3);
INSERT INTO messwerte VALUES (2, 4, 2);
INSERT INTO messwerte VALUES (3, 1, 3);
INSERT INTO messwerte VALUES (4, 3, 4);
INSERT INTO messwerte VALUES (5, 2, 3);
```

id	typ	wert
4	3	4
5	3	4

```
UPDATE messwerte SET typ = 2 WHERE id = 3;
UPDATE messwerte SET typ = 4 WHERE id = 5;
```

id	typ	wert
4	3	4
5	4	4

```
INSERT INTO messwerte VALUES (6, 1, 1);
```

id	typ	wert
6	1	1