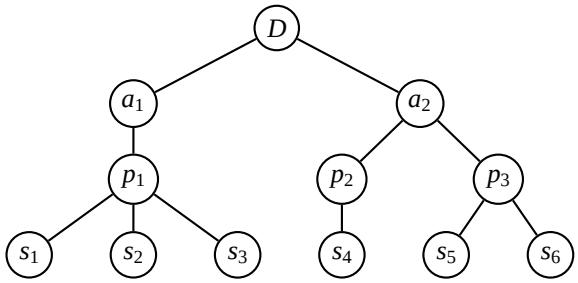
Gruppe A	PRÜFUNG AUS DATENBANKSYSTEME **VL 181.186**		11. 3. 2010
Kennnr.	Matrikelnr.	Familienname	Vorname

Arbeitszeit: 120 Minuten. Aufgaben sind auf den Angabeblättern zu lösen; Zusatzblätter werden nicht gewertet.

Aufgabe 1: (16)
Multi-Granularity Locking. Betrachten Sie folgende Datenbasis-Hierarchie.



Beantworten Sie, welche der folgenden Sequenzen von Anforderungen von Schlössern (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 einem Schloß vom Typ L zu belegen. Nehmen Sie bei dabei an, dass innerhalb dieser Sequenzen keine Freigabe von einmal angeforderten Sperren erfolgt.

1. (T_1, D, IS), (T_2, D, IS), (T_2, a_1, IS), (T_1, a_1, IS), (T_1, p_1, IS), (T_2, p_1, S), (T_1, s_3, S):
   Blockierung: ja ○ nein● Deadlock: ja ○ nein●
2. (T_1, D, IX), (T_2, D, IX), (T_1, a_1, IX), (T_2, a_2, IX), (T_1, p_1, IX), (T_1, s_2, X), (T_2, p_2, X), (T_1, a_2, S), (T_2, a_1, S):
   Blockierung: ja● nein ○ Deadlock: ja● nein ○
3. (T_1, D, IS), (T_2, D, IX), (T_1, a_1, IS), (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_2, X):
   Blockierung: ja● nein ○ Deadlock: ja ○ nein●
4. (T_1, D, IX), (T_2, D, IX), (T_1, a_1, IX), (T_2, a_2, IX), (T_1, p_1, IX), (T_2, p_3, IX), (T_1, s_1, X), (T_2, s_6, X):
   Blockierung: ja ○ nein● Deadlock: ja ○ nein●

(Pro korrekter Antwort 2 Punkte, **pro inkorrekter Antwort -2 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

Aufgabe 2: Mehrbenutzersynchronisation (10)
In einem DBMS ist eine Datenbank mit den Tabellen A und B implementiert, die als Spalten jeweils eine numerische ID ('id', Primary Key) und einen numerischen Wert ('wert') haben.

Gehen Sie davon aus, dass das DMBS alle Isolation Levels wie folgt implementiert:

Read Uncommited: Striktes 2PL für Exclusive-Locks. Keine Share-Locks.
Read Committed: Striktes 2PL für Exclusive-Locks, Share-Locks werden sofort nach Erhalt wieder freigegeben.
Repeatable Read: Striktes 2PL ohne Einschränkungen.
Serializable: Multiple Granularity Locking ohne Einschränkungen.

Gegeben sind zwei Transaktionen:

Transaktion 1:

```
SELECT * FROM A;
UPDATE B SET wert = 100;
COMMIT;
```

Transaktion 2:

```
SELECT * FROM B;
UPDATE A SET wert = 100;
COMMIT;
```

(a) Bei Isolation Level Serializable kann es hier zu einem Deadlock kommen. wahr● falsch ○

(b) Bei Isolation Level Repeatable Read kann es zu einem Deadlock kommen, bei Read Committed und Read Uncommitted allerdings nicht. wahr● falsch ○

(c) Wie müssten Sie Transaktion 2 verändern, damit es bei keinem der vier Isolation Levels zu einem Deadlock kommen kann, das Resultat aber das selbe bleibt? [1]

```
UPDATE A SET wert = 100;
SELECT * FROM B;
COMMIT;
```

Zusätzlich zu den Transaktionen 1 und 2 (in der unveränderten Version) wird nun auch folgende Transaktion 3 ausgeführt:

```
UPDATE A SET wert = 300 WHERE id = 1;
UPDATE B SET wert = 200 WHERE id = 1;
COMMIT;
```

(d) Es kann jetzt bei allen Isolation Levels zu einem Deadlock kommen. wahr ○ falsch ●

(e) Bricht Transaktion 1 nach dem UPDATE-Statement ab und löst einen Rollback aus, so kann es zu kaskadierendem Rücksetzen kommen. wahr ● falsch ○

(f) Zu kaskadierendem Rücksetzen kann es nur kommen, wenn auch Transaktion 3 ausgeführt wird. wahr ○ falsch ●

(g) Allein mit Transaktion 2 (in unveränderter Form) und 3 können Deadlocks im Isolation Level Read Committed und Read Uncommitted auftreten. 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)

Aufgabe 3: (15)

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

1. Bei verletzten Deferred Constraints treten Fehler erst am Ende der Transaktion auf. wahr ● falsch ○
2. PL/(pg)SQL ist eine prozedurale Sprache und spezifiziert daher lediglich was, niemals jedoch wie selektiert werden soll. wahr ○ falsch ●
3. Der Einsatz eines PERFORM-Statements in PL/pgSQL ist nur zulässig, wenn durch die Anfrage keine Seiteneffekte entstehen können. wahr ○ falsch ●
4. Wenn nur eine Leseoperation durchgeführt wird, kann die Datenbank auch direkt von der Festplatte lesen, ohne die komplette Seite in den Hauptspeicher zu laden. wahr ○ falsch ●
5. Durch mehrere aufeinanderfolgende Forwards (auch “Chaining” genannt) kann es zu hohen Zugriffszeiten kommen, da viele Seiten für nur ein Tupel eingelagert werden müssen. wahr ○ falsch ●
6. Eine Relation R sei an 5 Netzwerk-Knoten materialisiert mit den Gewichten 5, 9, 7, 3, und 5. Dann sind $Q_r(R) = 10$ und $Q_w(R) = 20$ gültige Lese- bzw. Schreib-Quoren. wahr ● falsch ○
7. Eine Datenbank enthalte die Relation $R(\underline{A}C)$ mit m Tupeln und die Relation $S(\underline{B}C)$ mit n Tupeln. Nehmen Sie an, dass die beiden Relationen bereits (nach dem Attribut C) sortiert wurden. Dann betragen die I/O-Kosten für den merge join (um den Ausdruck $R \bowtie S$ zu berechnen) $m + n$. wahr ○ falsch ●
8. Wenn in einem Auswertungsplan alle Block Nested Loop Joins durch Hash Joins ersetzt werden, dann kann dies unter Umständen zu einer Verringerung der Anzahl der Tupel in den Zwischenergebnissen führen. wahr ○ falsch ●
9. Nehmen Sie an, dass eine relationale Datenbank um das objektrelationale Feature “Geschachtelte Relationen” erweitert wurde. Dann lassen sich unter Umständen Anfragen bezüglich 1:n-Relationen effizienter auswerten als ohne dieses Feature. wahr ● falsch ○
10. Falls das Rollback von Datenbankänderungen unterbrochen wird, muss das Datenbanksystem beim Wiederanlauf in der Lage sein, das Rollback abzuschließen. 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)

Aufgabe 4: SQL (6)

Definieren Sie eine Tabelle `Zeugnis` mit einer eindeutigen `ID` als Zahl und einer Note als Text, der nur die Werte “S1”, “G2”, “B3”, “G4” und “N5” annehmen darf. [3]

```
CREATE TABLE Zeugnis (
  ID INTEGER PRIMARY KEY
  Note CHAR(2) CHECK(Note in (‘S1’, ‘G2’, ‘B3’, ‘G4’, ‘N5’))
);
```

Definieren Sie weiters eine *Sequence* mit Namen “certKey”, die bei 1 startet und jeweils um 1 erhöht wird. [2]

```
CREATE SEQUENCE certKey
  START WITH 1 INCREMENT BY 1;
```

Fügen Sie zwei Zeilen in die Tabelle `Zeugnis`(ID, Note) ein. Der Wert von `ID` soll von der Sequence “certKey” kommen. [1]

```
INSERT INTO Row VALUES
  (nextval(‘certKey’), ‘S1’);
INSERT INTO Row VALUES
  (nextval(‘certKey’), ‘N5’);
```

Aufgabe 5: Hierarchische Abfragen (10)

Dieser Aufgabe liegt folgendes Schema zugrunde:

`produkt` (pid, name)
`bauplan` (produkt: produkt.pid, teil: produkt.pid, anzahl)

Evaluieren Sie folgendes SQL-Statement auf die angegebenen Tabellen, und geben Sie die Zwischentabelle `temp` und die Ausgabe der gesamten Abfrage an:

produkt	
pid	name
1	'A'
2	'B'
3	'C'
4	'D'

bauplan		
produkt	teil	anzahl
1	2	3
1	3	1
3	2	4
4	1	2
4	3	1

```
WITH RECURSIVE temp(produkt, teil, anzahl) AS (
  SELECT produkt, teil, anzahl
  FROM bauplan
  WHERE produkt NOT IN (SELECT teil from bauplan)
  UNION ALL
  SELECT bp.produkt, bp.teil, bp.anzahl * t.anzahl
  FROM temp t, bauplan bp
  WHERE t.teil = bp.produkt
)
SELECT teil, name, SUM(anzahl)
FROM temp, produkt
where produkt.pid = teil
GROUP BY teil, name;
```

temp

produkt	teil	anzahl
4	1	2
4	3	1
1	2	6
1	3	2
3	2	4
3	2	8

Ergebnis d. Abfrage

teil	name	sum
1	A	2
2	B	18
3	C	3

Aufgabe 6: PL/SQL Trigger (8)

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

```
CREATE TABLE person (
  pid INTEGER PRIMARY KEY,
  name VARCHAR(255),
  salary INTEGER
);

CREATE FUNCTION fTrigger1()
RETURNS trigger AS $$
BEGIN
  IF (NEW.salary < 0) THEN
    RETURN NULL;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION fTrigger2()
RETURNS trigger AS $$
BEGIN
  IF (OLD.salary * 0.8 > NEW.salary) THEN
    NEW.salary = OLD.salary * 0.8;
  END IF;
  IF (OLD.salary * 1.2 < NEW.salary) THEN
    NEW.salary = OLD.salary * 1.2;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tTrigger1
BEFORE INSERT ON person
FOR EACH ROW EXECUTE PROCEDURE fTrigger1();

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

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

```
INSERT INTO person VALUES (1, 'A', 10);
INSERT INTO person VALUES (2, 'B', 100);
INSERT INTO person VALUES (3, 'C', -10);
INSERT INTO person VALUES (4, 'D', 1000);
```

pid	name	salary
1	A	10
2	B	100
4	D	1000

```
UPDATE person SET salary = 100;
```

pid	name	salary
1	A	12
2	B	100
4	D	800

Aufgabe 7: Java (10)

Schreiben Sie ein Java Programm unter der Verwendung von JDBC, das eine Verbindung zu einer lokalen *Postgres Datenbank* öffnet und SQL Statements absetzt.

Um eine Fehlerbehandlung brauchen Sie sich **nicht** kümmern.

Öffnen Sie eine Verbindung zu einer PG Datenbank (Server läuft auf `localhost`, Benutzername `exam`, Passwort `dbs`). [2]

```
Class.forName('org.postgresql.Driver');
Connection c = DriverManager.getConnection('jdbc:postgresql://localhost',
    'exam',
    'dbs');
** oder ***
DriverManager.registerDriver(new Driver());
PGSimpleDataSource ds = new PGSimpleDataSource();
ds.setServerName('localhost') ;
ds.setUser('exam') ;
ds.setPassword('dbs') ;
Connection c = ds.getConnection();
```

Erstellen Sie ein `PreparedStatement` für folgendes SQL statement:
`sql = "SELECT name,note FROM tuwis_zeugnis WHERE matrnr = ?"`
Sie können den String `sql` direkt verwenden.
Führen Sie die Abfrage auf alle Matrikelnummern zwischen 9000000 und 9999999 aus. Geben Sie die Attribute `name` und `note` auf die Konsole aus. Achten Sie darauf alle geöffneten Ressourcen wieder zu schließen. [8]

```
ResultSet rs = null;
PreparedStatement s = c.prepareStatement(sql);
for(int m = 9000000; m <= 9999999; m++) {
  s.setInt(1,m);
  rs = s.executeQuery();
  while(rs.next()) {
    System.out.println(rs.getString('name'));
    System.out.println(rs.getInt('note'));
  }
  rs.close();
}
s.close();
c.close();
```

Gesamtpunkte: 75