Gruppe A

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

PRÜFUNG AUS DATENBANKSYSTEME **VL 181.186**			17. 6. 2010
Kennnr.	Matrikelnr.	Familienname	Vorname

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

Aufgabe 1: (15)

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

1. Es gibt Relationen $R(AB)$ mit 100 Tupeln und $S(AC)$ mit 100 Tupeln, für die der Ausdruck $R \bowtie S$ 10000 Tupeln ergibt. wahr ⊗ falsch ◯
2. Wenn ein Join mittels Block Nested Loop Join realisiert wird, kann die Anzahl der benötigten I/O-Operationen im Idealfall weniger sein, als wenn derselbe Join mittels Nested Loop Join realisiert würde. wahr ⊗ falsch ◯
3. 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 ◯
4. Nehmen Sie an, dass eine Relation R 16384 ($= 128^2$) Seiten umfasst und die Puffergröße 128 beträgt. Dann ist bei einem Hash Join von R mit einer beliebigen Relation S in der Build-Phase auf jeden Fall ein Re-Hashing von R erforderlich. wahr ◯ falsch ⊗
5. Prepared Statements dienen bei JDBC unter Anderem dazu, die Performance der Anfragen zu steigern. wahr ⊗ falsch ◯
6. 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 ⊗
7. Nehmen Sie an, dass in einem verteilten DBMS das globale Relationenschema $R(\underline{A}BCDE)$ (d.h.: A ist der Primärschlüssel) in die vertikalen Fragmente $R(AB)$, $R(ABCD)$, und $R(ADE)$ unterteilt wurde. Diese Fragmentierung ist vollständig und rekonstruierbar. wahr ⊗ falsch ◯
8. Die Historie $r_2(X)$, $w_2(X)$, $r_1(Y)$, c_2, $r_1(X)$, $w_1(X)$, $w_1(Y)$, c_1 ist sowohl beim Zwei-Phasen Sperrprotokoll als auch beim strengen Zwei-Phasen Sperrprotokoll möglich. wahr ⊗ falsch ◯
9. RAID garantiert die Eigenschaften *R*edundancy, *A*tomicity, *I*solation, und *D*urability. wahr ◯ falsch ⊗
10. Falls das Rückrollen von Datenbankänderungen durch einen Systemabsturz unterbrochen wird, muss das Datenbanksystem beim Wiederanlauf in der Lage sein, das Rückrollen 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 2: Mehrbenutzersynchronisation (14)

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 DBMS 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.
Locks sind bis auf Zeilenebene implementiert (row-level locking).

Gegeben sind zwei Transaktionen:

Transaktion 1:

```
SELECT * FROM A;
UPDATE B SET wert = 100 WHERE id = 1;
COMMIT;
```

Transaktion 2:

```
UPDATE B SET wert = 200 WHERE id < 10;
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 Isolation Level Read Committed allerdings nicht. wahr ⊗ falsch ◯

(c) Bei Read Uncommited kann es zu Deadlocks kommen, da die UPDATE-Statements einen Exclusive-Lock auf die selbe Tabelle B anfordern. wahr ◯ falsch ⊗

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

```
UPDATE B SET wert = 100 WHERE id = 1;
SELECT * FROM A;
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;
```

(e) Es kann jetzt bei allen Isolation Levels zu einem Deadlock kommen. wahr ⊗ falsch ◯

(f) Es kann bei allen Isolation Levels zu einem Deadlock kommen, wenn nur Transaktion 2 und 3 ausgeführt werden, Transaktion 1 allerdings nicht. wahr ⊗ falsch ◯

(g) Allein mit Transaktion 2 und 3 können Deadlocks sowohl im Isolation Level Read Committed als auch bei Read Uncommitted auftreten. wahr ⊗ falsch ◯

(h) Bricht Transaktion 2 nach dem zweiten UPDATE-Statement ab und löst einen Rollback aus, so kann es zu kaskadierendem Rücksetzen kommen. wahr ⊗ falsch ◯

(i) Kaskadierendes Rücksetzen kann nur durch zyklische Locks der UPDATE-Statements von Transaktionen 1 und 2 verursacht werden. 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: (14)

Nehmen Sie an, dass Sie ein DBMS hinsichtlich der physischen Datenorganisation optimieren, wozu Sie die vom Betriebssystem angebotene Festplattenverwaltung umgehen, und selbst diverse hardwarespezifische Kenngrößen berechnen müssen.

Gegeben sei eine Festplatte mit folgenden Eigenschaften: Sektorgröße von 512 Bytes, 2000 Spuren pro Plattenoberfläche, 50 Sektoren pro Spur, 5 doppelseitige Platten, und eine durchschnittliche Seek Time von 10 ms (Millisekunden).

(a) Was ist die Kapazität einer Spur in Bytes? Was die Kapazität jeder einzelner Plattenoberfläche? Was die Kapazität der ganzen Festplatte? (jeweils in Kilobyte, wobei 1 kB = 1024 Bytes) [4]

Es gibt $512 \cdot 50 = 25600$ Bytes, d.h. $25600/1024 = 25$ Kilobyte (kB), pro Spur. Jede Plattenoberfläche besitzt $25 \cdot 2000 = 50000$ kB, und somit die ganze Festplatte $50000 \cdot 5 \cdot 2 = 500000$ kB.

(b) Wie viele Zylinder besitzt die Festplatte? [2]

Die Anzahl an Zylindern entspricht der Anzahl der Spuren einer Platte, also 2000.

(c) Sind 256 Bytes eine gültige Blockgröße? 2048? 51200? [3]

Die Blockgröße entspricht mehreren Sektoren. 256 ist somit ungültig, während $2048 = 512 \cdot 4$ und $51200 = 512 \cdot 100$ gültig sind.

(d) Angenommen die Festplatte rotiert mit 5400 rpm (revolutions per minute), also 5400 Umdrehungen pro Minute. Was ist die maximale Latenzzeit (rotational delay)? Was die durchschnittliche Latenzzeit? [3]

Bei 5400 rpm ist die maximale Latenzzeit, die der Zeit für eine ganze Umdrehung entspricht, $1/5400 \cdot 60 \cdot 1000 \approx 11$ Millisekunden. Die durchschnittliche Latenzzeit wird mit einer halben Umdrehung abgeschätzt, also ca. 6 ms.

(e) Angenommen man kann pro Umdrehung genau eine Spur an Daten übertragen. Was ist dann die Transferrate? [2]

Eine Spur umfasst 25 kB Daten. Bei einer Übertragung von einer Spur pro Umdrehung ist somit die Transferrate $25 \cdot (5400/60) = 2250$ kB/s.

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", "U2", "B3", "G4" und "N5" annehmen darf. [3]

```
CREATE TABLE Zeugnis (
  ID INTEGER PRIMARY KEY
  Note CHAR(2) CHECK(Note in ('S1', 'U2', '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 (8)

Dieser Aufgabe liegt folgendes Schema zugrunde:

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

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

produkt	
pid	name
1	'W'
2	'X'
3	'Y'
4	'Z'

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

```
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	2
1	2	6
1	3	2
3	2	8
3	2	8

Ergebnis d. Abfrage

teil	name	sum
1	W	2
2	X	22
3	Y	4

Aufgabe 6: PL/SQL (9)

Definieren Sie einen Cursor "cMA" mit einem numerischen Parameter "`parameter`" und Schreibzugriff für folgende SQL Abfrage:

```
SELECT name, gehalt FROM Mitarbeiter WHERE klasse = parameter;
```

Definieren Sie weiters eine Variable "recMA" die den gleichen Datentyp wie eine Zeile der Tabelle `Mitarbeiter` hat. [4]

```
cMA CURSOR (parameter NUMBER) IS SELECT name, gehalt FROM Mitarbeiter
  WHERE klasse = parameter FOR UPDATE;
recMA Mitarbeiter%ROWTYPE;
```

Schreiben Sie eine Schleife, die über **alle** Zeilen des Cursors "cMA" mit 42 als Parameter iteriert und die Mitarbeiter, deren Gehalt kleiner als 1 ist, löscht.

Stellen Sie sicher dass der Cursor richtig geöffnet und geschlossen wird.

Ein DELETE Statement darf nur innerhalb der Schleife vorkommen und sich auf die aktuelle Cursor-Position beziehen. [5]

```
OPEN cMA(42);
LOOP
  FETCH cMA INTO recMA;
  EXIT WHEN cMA%NOTFOUND;
  IF recMA.gehalt < 1 THEN
    DELETE FROM Mitarbeiter WHERE CURRENT OF cMA;
  END IF;
END LOOP;
CLOSE cMA;
```

Aufgabe 7: Java (9)

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:
`String sql = "SELECT name FROM team WHERE gruendungsjahr = ?"`
Sie können die Variable “`sql`” direkt verwenden.
Führen Sie die Abfrage mit allen Jahren zwischen 0 und 2010 aus. Geben Sie das Attribut `name` zusammen mit dem Jahr auf die Konsole aus. Achten Sie darauf, alle geöffneten Ressourcen wieder zu schließen. Beachten Sie weiters, dass pro Abfrage mehrere Zeilen zurückkommen können.
Beispielausgabe:
`1886: FC Arsenal`
`1899: FC Barcelona`
`1899: AC Mailand` [7]

```
ResultSet rs = null;
PreparedStatement s = c.prepareStatement(sql);
for(int j = 0; j <= 2010; j++) {
  s.setInt(1,j);
  rs = s.executeQuery();
  while(rs.next()) {
    System.out.println(j + ": "+ rs.getString(‘name’));
  }
  rs.close();
}
s.close();
c.close();
```

`Gesamtpunkte: 75`