

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			17. 1. 2014
Kennnr.	Matrikelnr.	Familienname	Vorname

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

Aufgabe 1:

(15)

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

1. Bei den Einstellungen steal/force ist kein Undo möglich, aber ein Redo nötig. wahr falsch
2. Die Historie $r_1(A)$, $r_2(A)$, $w_1(A)$, $w_2(B)$, $w_1(B)$, c_1 , c_2 ist nicht serialisierbar. wahr falsch
3. Eine Relation R sei an 6 Netzwerk-Knoten materialisiert mit den Gewichten von jeweils 10. Dann sind $Q_r(R) = 40$ und $Q_w(R) = 30$ gültige Lese- bzw. Schreib-Quoren. wahr falsch
4. Betrachten Sie drei Relationen $U(ABC)$, $V(BCD)$ und $W(DE)$. Dann gilt auf jeden Fall folgende Gleichheit:
 $((U \bowtie V) \bowtie W) = ((U \times W) \bowtie V)$ wahr falsch
5. ACID garantiert die Eigenschaften Atomicity, Consistency, Isolation, und Durability. wahr falsch
6. Es gibt Relationen $R(\underline{AB})$ mit 100 Tupeln und $S(AC)$ mit 100 Tupeln, für die der Ausdruck $R \bowtie S$ 10000 Tupeln ergibt. wahr falsch
7. Wenn ein Join mittels Hybrid Hash Join realisiert wird, dann kann das dadurch berechnete Ergebnis im Idealfall weniger Tupel umfassen, als wenn derselbe Join mittels Hash Join realisiert würde. wahr falsch
8. Mit SQL-92 lassen sich Anfragen formulieren, die man mit PL/pgSQL nicht formulieren könnte. wahr falsch
9. Die Anzahl der Zyklen im Wartegraphen entspricht immer der Anzahl der Transaktionen, die zurückgesetzt werden müssen, um einen Deadlock aufzulösen. wahr falsch
10. Die Gesamtkosten für die Sortierung einer Tabelle T sind $2b_T(1 + \lceil \log_{m-1}(\lceil m/b_T \rceil) \rceil)$, mit b_T als Anzahl der Seiten von T am Hintergrundspeicher und m als Anzahl der Seitenrahmen im Datenbankpuffer. wahr falsch

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

Aufgabe 2:

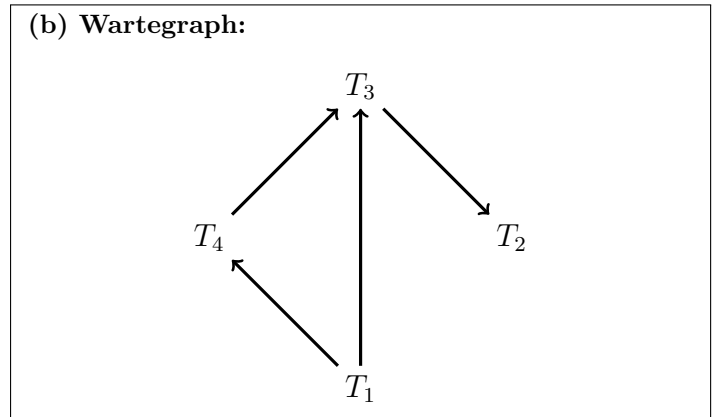
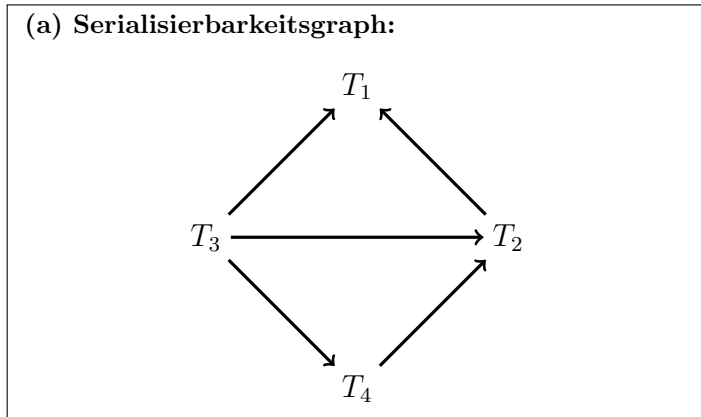
(14)

(a) Eine Historie sei durch folgende Folge von Elementaroperationen gegeben: $r_3(D)$ vor $w_3(C)$ vor $r_2(C)$ vor $r_3(A)$ vor $r_4(B)$ vor $w_2(D)$ vor $w_4(A)$ vor $w_2(B)$ vor $r_1(B)$ vor $w_2(A)$ vor $w_1(C)$ vor c_1 vor c_2 vor c_3 vor c_4 .

Zeichnen Sie ins erste Kästchen den Serialisierbarkeitsgraphen für die Transaktionen T_1, T_2, T_3 und T_4 . Verwenden Sie dabei folgende Konvention: eine Kante $T_i \rightarrow T_j$ bedeutet, dass in einer äquivalenten seriellen Historie die „Transaktion T_i vor T_j “ ausgeführt werden muss (vgl. VO-Folien bzw. Kemper-Buch). [5]

(b) Betrachten Sie die folgende Folge von Sperranforderungen, wobei „lockS $_i(O)$ “ (bzw. „lockX $_i(O)$ “) bedeutet, dass die Transaktion T_i eine Lesesperre (bzw. eine Schreibsperre) auf das Datenobjekt O anfordert: lockX $_4(B)$ vor lockS $_2(A)$ vor lockS $_3(C)$ vor lockX $_4(C)$ vor lockX $_3(A)$ vor lockX $_1(C)$ vor lockS $_1(B)$.

Zeichnen Sie ins zweite Kästchen den Wartegraphen unter der Annahme, dass zum momentanen Zeitpunkt keine der erhaltenen Sperren wieder zurückgegeben wurde. Verwenden Sie dabei folgende Konvention: eine Kante $T_i \rightarrow T_j$ bedeutet „Transaktion T_i wartet auf die Freigabe einer Sperre durch T_j “ (vgl. VO-Folien bzw. Kemper-Buch). [5]



(c) Ist die Historie aus (a) serialisierbar?

ja nein

Wenn ja: in Reihenfolge T_3 vor T_4 vor T_2 vor T_1

Wenn nein: die Historie wird durch das Streichen von zumindest 0 Operationen serialisierbar. [4]

Aufgabe 3:

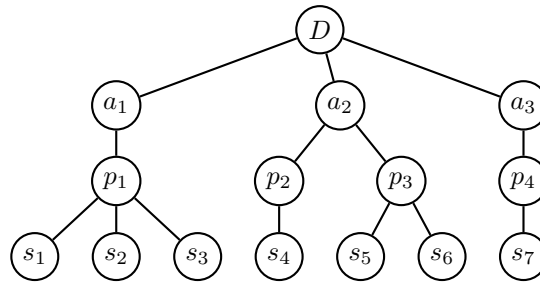
(4)

Nennen und erklären Sie (mindestens) vier Konzepte, die typisch und spezifisch für objekt-relationale Datenbanken sind.

Große Objekte (LOBs: Large Objects), Mengenwertige Attribute, Geschachtelte Relationen, Typdeklarationen (UDTs: User-defined types), Referenzen, Objektidentität, Pfadausdrücke, Vererbung, Operationen

Aufgabe 4:

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. $(T_1, D, IX), (T_2, D, IX), (T_2, a_2, IX), (T_1, a_1, X), (T_2, p_2, X), (T_1, a_2, IX), (T_1, p_3, X)$:
 Blockierung: ja nein Deadlock: ja nein
2. $(T_1, D, IS), (T_2, D, IX), (T_1, a_3, IS), (T_2, a_1, X), (T_1, p_4, S), (T_2, a_3, X), (T_1, a_1, IS), (T_1, p_1, S)$:
 Blockierung: ja nein Deadlock: ja nein
3. $(T_1, D, IS), (T_2, D, IX), (T_1, a_1, IS), (T_1, p_1, IS), (T_2, a_3, X), (T_2, a_1, IX), (T_1, s_2, S), (T_2, p_1, IX), (T_2, s_3, X)$:
 Blockierung: ja nein Deadlock: ja nein
4. $(T_1, D, IX), (T_2, D, IS), (T_2, a_2, IS), (T_1, a_2, IX), (T_2, p_3, S), (T_1, p_2, X), (T_1, p_3, IX), (T_1, s_5, X)$:
 Blockierung: ja nein Deadlock: ja nein

(Pro korrekter Antwort 1,5 Punkte, **pro inkorrektter 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, in dem die Ergebnisse eines Film- und Fernsehpreises (den vergangenen Golden Globe Awards) abgelegt werden sollen. Der Einfachheit halber beschränken wir uns auf TV Serien.

Series(sid, name, genre: *Genre.gid*, wins, nominations, episodes, metacritic)

Genre(gid, name, parent: *Genre.gid*)

Auf der letzten Seite dieser Prüfung finden Sie eine Beispielinstantz dieses Schemas!

In der Tabelle **Series** werden die verschiedenen TV Serien gespeichert, die für den Award nominiert wurden. Das Attribut **wins** speichert die Anzahl der Preise, welche die Serie gewonnen hat und das Attribut **nominations** die Anzahl der Nominierungen. Im Attribut **episodes** ist die Anzahl der Folgen der Serie abgelegt und das Attribut **metacritic** enthält die Bewertung der Serie. Die Tabelle **Genre** enthält die Genres der TV Serien zusammen mit einem möglichen Obergenre **parent**.

Stellen Sie sicher, dass das Attribut **metacritic** nur ganzzahlige Werte von 0 bis 100 annehmen kann!

Treffen Sie plausible Annahmen bezüglich der Datentypen der Attribute.

Aufgabe 5:

(6)

Geben Sie CREATE TABLE Statements mit allen nötigen Constraints für die beiden Tabellen an.

```
CREATE TABLE Genre (  
    gid INTEGER PRIMARY KEY,  
    name VARCHAR(20),  
    parent INTEGER REFERENCES Genre(gid)  
);  
  
CREATE TABLE Series (  
    sid INTEGER PRIMARY KEY,  
    name VARCHAR(40),  
    genre INTEGER REFERENCES Genre(gid),  
    wins INTEGER,  
    nominations INTEGER,  
    episodes INTEGER,  
    metacritic INTEGER CHECK (metacritic >= 0 AND metacritic <= 100)  
);
```

Aufgabe 6:

(7)

Evaluieren Sie das folgende SQL-Statement bezüglich der Datenbankinstanz **awards** (siehe letzte Seite), und geben Sie die Ausgaben der Abfrage an:

```
WITH RECURSIVE Temp(gid,parent,nominations) AS (  
  SELECT g.gid, g.parent, s.nominations FROM Genre g, Series s  
  WHERE s.genre = g.gid  
UNION ALL  
  SELECT g.gid, g.parent, t.nominations FROM Genre g, Temp t  
  WHERE g.gid = t.parent  
)  
SELECT t.gid, SUM(t.nominations) FROM Temp t  
GROUP BY t.gid ORDER BY t.gid DESC;
```

gid	sum
10	2
9	2
8	4
7	3
6	3
5	4
4	3
3	8
2	13
1	21

Betrachten Sie folgende PL/pgSQL Trigger:

```
CREATE OR REPLACE FUNCTION before_update() RETURNS TRIGGER AS $$
BEGIN
  IF (NEW.nominations >= 3) THEN
    NEW.metacritic = NEW.metacritic + 10;
  ELSE
    IF (NEW.episodes >= 100) THEN
      NEW.wins = NEW.wins + 1;
    END IF;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER t_before_update BEFORE UPDATE ON Series FOR EACH ROW
EXECUTE PROCEDURE before_update();
```

```
CREATE OR REPLACE FUNCTION after_update() RETURNS TRIGGER AS $$
BEGIN
  IF (NEW.wins > NEW.nominations) THEN
    NEW.wins = NEW.nominations;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER t_after_update AFTER UPDATE ON Series FOR EACH ROW
EXECUTE PROCEDURE after_update();
```

Geben Sie nun das Resultat der folgenden SQL-Statements, angewandt auf die Datenbankinstanz **awards** (siehe letzte Seite) unter Beachtung der definierten Trigger an:

```
UPDATE Series SET wins = wins * 2 WHERE metacritic > 70;
```

```
SELECT sid, wins, metacritic FROM Series ORDER BY sid;
```

sid	wins	metacritic
1	4	84
2	2	86
3	0	86
4	0	70
5	2	70
6	0	57
7	1	87
8	1	59

Vervollständigen Sie die Java Methode `wins`, der ein Genre `gid` übergeben wird, und die für jede Serie dieses Genres:

- die `sid` und die Anzahl der gewonnenen Preise (`wins`) ausgibt und
- falls die Anzahl der gewonnenen Preise (`wins`) größer als zwei ist, die `metacritic` Bewertung um 10 erhöht.

Verwenden Sie ausschliesslich PreparedStatements (d.h. keine anderen Arten von Statements). Legen Sie die PreparedStatements hier an. Sie können hier auf eine Connection `c` zugreifen.

```
PreparedStatement psSelect = c.prepareStatement(
    "SELECT sid, wins FROM Series WHERE gid = ?");
PreparedStatement psUpdate = c.prepareStatement(
    "UPDATE Series SET metacritic = metacritic + 10 WHERE sid = ?");
```

Vervollständigen Sie nun die Methode `wins`. Sie können die oben angelegten PreparedStatements verwenden. Um die genaue Formatierung der Ausgabe und die Fehlerbehandlung brauchen Sie sich nicht zu kümmern.

```
public void wins(String gid) throws Exception {
    psSelect.setInt(1, gid);
    ResultSet rs = psSelect.executeQuery();

    while (rs.next()) {
        int sid = rs.getInt("sid");
        int wins = rs.getInt("wins");
        System.out.println("Series " + sid + " has " + wins + " wins.");

        if (wins > 2) {
            stmt.setInt(1, sid);
            stmt.executeUpdate();
        }
    }

    rs.close();
}
```

Gesamtpunkte: 75

Sie können diese Seite abtrennen und brauchen sie nicht abgeben!

Datenbankinstanz **awards**:

Series						
sid	name	genre	wins	nom.	epi.	meta.
1	Breaking Bad	4	2	3	62	74
2	House of Cards	5	1	4	26	76
3	The Good Wife	6	0	3	101	76
4	The White Queen	7	0	3	10	70
5	Brooklyn Nine-Nine	8	2	2	12	70
6	The Big Bang Theory	8	0	2	148	57
7	Modern Family	9	0	2	107	87
8	Parks and Recreation	10	1	2	100	59

Genre		
gid	name	parent
1	All	NULL
2	Drama	1
3	Comedy	1
4	Crime Drama	2
5	Political Thriller	2
6	Legal Drama	2
7	Historical Fiction	2
8	Sitcom	3
9	Mockumentary	3
10	Political Satire	3