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**			25. 06. 2015
Kennnr.	Matrikelnr.	Familienname	Vorname

Arbeitszeit: 100 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. Betrachten Sie die Kostenformel $2 * b_R * (1 + I)$ mit $I = \lceil \log_{m-1}(\lceil b_R/m \rceil) \rceil$ für das externe Sortieren. Dabei steht b_R für die Anzahl der Seiten der Relation R und I für die Anzahl der Level-0 Runs. wahr ◯ falsch ⊗
2. Wenn der Serialisierbarkeitsgraph einer Historie azyklisch ist, gibt es immer eine eindeutige äquivalente serielle Historie. wahr ◯ falsch ⊗
3. Nehmen Sie an, dass beim Zweiphasen-Commit-Protokoll einer der Agenten abstürzt. Dann kann dieser Agent beim Wiederanlauf mit Hilfe der eigenen Log-Datei eigenständig (also ohne Rückfrage bei einem anderen Agenten oder beim Koordinator) entscheiden, ob eine Transaktion erfolgreich war. wahr ◯ falsch ⊗
4. Falls das Rückrollen von Datenbank-Änderungen durch einen Systemabsturz unterbrochen wird, muss das Datenbanksystem beim Wiederanlauf in der Lage sein, das Rückrollen abzuschließen. wahr ⊗ falsch ◯
5. Bei den Einstellungen ¬ steal/force ist im Falle eines Recovery kein Undo nötig. wahr ⊗ falsch ◯
6. Aus der Rekonstruierbarkeit der Fragmentierung in einem verteilten Datenbankmanagementsystem folgt die Vollständigkeit der Fragmentierung. wahr ⊗ falsch ◯
7. Betrachten Sie die Relationen $R(\underline{A}B)$ mit 3000 Tupeln und $S(AC)$ mit 5000 Tupeln, d.h.: Attribut A ist ein Primärschlüssel der Relation R. Dann liefert der Ausdruck $R \bowtie S$ nie mehr als 3000 Tupel. wahr ◯ falsch ⊗
8. Betrachten Sie zwei Relationen $R(AB)$ und $S(AB)$. Dann gilt auf jeden Fall folgende Gleichheit: $(\pi_A(R) \cap \pi_A(S)) \times (\pi_B(R) \cap \pi_B(S)) = (R \bowtie S)$. wahr ◯ falsch ⊗
9. Bei der Historie $r_1(A)$, $r_2(A)$, $w_2(B)$, $r_1(B)$, a_1 führt der Abbruch der Transaktion T_1 zu einem kaskadierenden Rücksetzen von T_2. wahr ◯ falsch ⊗
10. Nehmen Sie an, dass eine relationale Datenbank um das objektrelationale Feature "mengewertige Attribute"erweitert wurde. Dann lassen sich $n{:}\,m$ Beziehungen unter Umständen ohne Hilfstabelle realisieren. 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: (12)

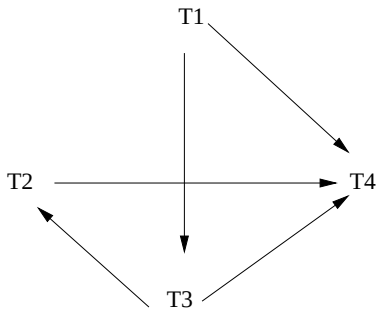
(a) (5 Punkte) Eine Historie sei gegeben durch folgende Folge von Elementaroperationen: $r_1(D)$, $r_2(B)$, $r_3(A)$, $w_2(B)$, $w_2(A)$, $w_3(D)$, $r_4(B)$, $r_1(C)$, $r_4(C)$, $r_3(C)$, $w_4(D)$, c_1, c_2, c_3, c_4.

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).

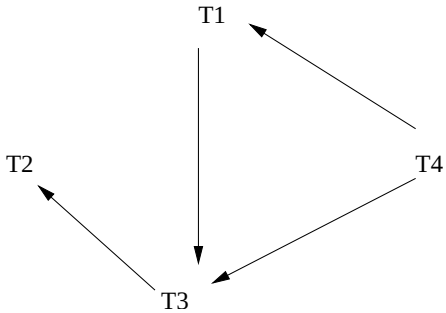
(b) (5 Punkte) Betrachten Sie die folgende Folge von Sperranforderungen, wobei "lockS_i(O)"(bzw. "lockX_i(O)") bedeutet, dass die Transaktion T_i eine Lesesperre (bzw. eine Schreibsperre) auf das Datenobjekt O anfordert: lockS_1(A) vor lockX_2(D) vor lockX_3(C) vor lockS_1(C) vor lockS_3(D) vor lockS_4(C) vor lockX_4(A).

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).

(a) Serialisierbarkeitsgraph:



(b) Wartegraph:



(c) (2 Punkte) Geben Sie für den Serialisierbarkeitsgraphen aus (a) eine mögliche Reihenfolge der Serialisierung an:

T 1 ..... vor T 3 ..... vor T 2 ..... vor T 4 .....

Aufgabe 3: (18)

Der Schönbrunner Zoo verwendet eine Datenbank mit folgenden Relationen:

Tier(Name, Art, Geb, Gewicht) (kurz t)
Tierart(Art, Gattung, Platz, Futter) (kurz ta)
Mitarbeiter(mNr, Name, Geb, Adresse) (kurz m)
pflege(mNr, tierName, Aufgabe) (kurz p).

Nehmen Sie an, dass $|t| = 4000$, $|ta| = 500$, $|m| = 100$ und $|p| = 10000$. Es ist die Anfrage

```
select t.Name, t.Geb
from t Tier, m Mitarbeiter, p pflege, ta Tierart
where t.Name = p.tierName and m.mNr = p.mNr and t.Art = ta.Art and ta.Gattung = 'Raubkatze' and m.Geb > 1995;
```

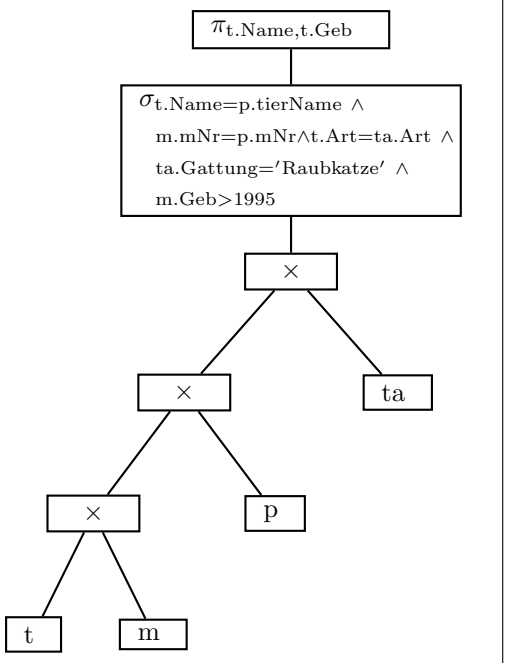
auszuführen (d.h. gesucht sind Informationen über Raubkatzen, die von einem jungen Mitarbeiter gepflegt werden). Es sind die Selektivitäten $Sel_{t/p} = 1/4000 = 0.00025$, $Sel_{m/p} = 1/100 = 0.01$, $Sel_{t/ta} = 1/500 = 0.002$, $Sel_{ta.Gattung='Raubkatze'} = 0.2$ und $Sel_{m.Geb>1995} = 0.1$ anzunehmen.

(a) Zeichnen Sie ins erste Kästchen den Operator-Baum für die kanonische Übersetzung.

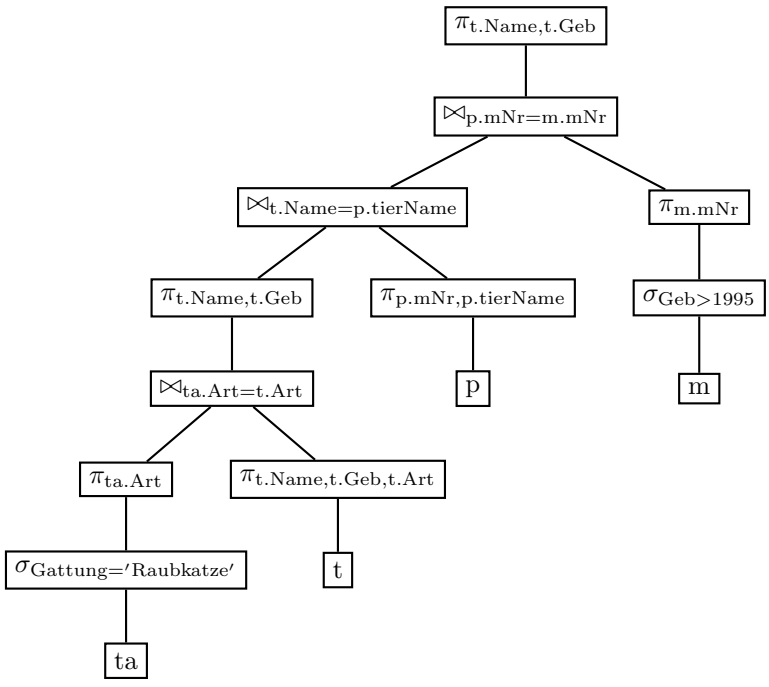
(b) Zeichnen Sie ins zweite Kästchen den Operator-Baum für den optimierten algebraischen Ausdruck. Wenden Sie für die Optimierung folgende Heuristiken an:

- Kreuzprodukte durch Joins ersetzen,
- Join-Reihenfolge so, dass der erste Join möglichst wenige Tupel als Zwischenergebnis liefert,
- Selektionen so weit wie möglich nach unten verschieben,
- Attribute, die nicht mehr benötigt werden, möglichst früh wegprojizieren.

(a) Kanonische Übersetzung:



(b) Optimierter Ausdruck:



Die folgende Datenbankbeschreibung gilt für die Aufgaben 4 – 7:

Für die Verwaltung der Universitätskurse soll folgendes stark vereinfachtes Datenbankschema verwendet werden:

`course`(id, name, points, ctype)
`depends`(id: *course.id*, requires: *course.id*)

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

In der Tabelle `course` werden die verschiedenen Kurse gespeichert. Diese werden eindeutig durch die Nummer `id` identifiziert. Die Nummer `id` soll automatisch mittels der Sequenz `seq_ids` vergeben werden. Jeder Kurs hat einen Namen `name`. Die Anzahl der ECTS wird im Attribut `points` und der Kurstyp im Attribut `ctype` gespeichert. Die Anzahl der `points` muss grösser als 0 sein. Der Kurstyp soll nur folgende Werte annehmen können: "VO", "VU", "UE" oder "SE".

Die Tabelle `depends` stellt Kursabhängigkeiten dar. In beiden Attributen (`id` und `requires`) werden `id`s der Tabelle `course` gespeichert. Ein Tupel in der Tabelle bedeutet, dass der Kurs mit der ID `requires` vor dem Kurs mit der ID `id` absolviert werden muss. Beide Attribute sind Teil des Primärschlüssels.

Treffen Sie plausible Annahmen bezüglich der Datentypen der Attribute, sofern nicht angegeben.

Aufgabe 4: (7)

Die Kurs-IDs sollen automatisch vergeben werden. Legen Sie dazu eine Sequenz `seq_ids` an, welche bei 100 beginnt und in 10er Schritten erhöht wird.

```
CREATE SEQUENCE seq_ids INCREMENT BY 10 MINVALUE 100 NO CYCLE;
```

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

```
CREATE TABLE course (
        id INTEGER PRIMARY KEY DEFAULT nextval('seq_ids'),
        name VARCHAR(50) NOT NULL,
        points INTEGER CHECK (points > 0),
        ctype VARCHAR(2) CHECK (ctype IN ('VO','VU','UE','SE'))
);

CREATE TABLE depends (
        id INTEGER REFERENCES course(id),
        requires INTEGER REFERENCES course(id),
        PRIMARY KEY (id, requires)
);
```

Aufgabe 5: (9)

Evaluieren Sie das folgendes SQL-Statement bezüglich der Datenbankinstanz **courses** (siehe letzte Seite), und geben Sie die Ausgabe der Abfrage an:

```
WITH RECURSIVE temp(id, requires, points) AS (
      SELECT d.id, d.requires, c.points
      FROM depends d JOIN course c ON d.requires = c.id
   UNION
      SELECT c.id, NULL, NULL
      FROM course c
      WHERE c.id NOT IN (SELECT id FROM depends)
   UNION
      SELECT t.id, d.requires, c.points
      FROM temp t, depends d, course c
      WHERE t.requires = d.id AND d.requires = c.id
)
SELECT id, SUM(points) AS sum, COUNT(points) AS count
FROM temp t
GROUP BY id
ORDER BY id;
```

id	sum	count
100	0	0
110	0	0
120	6	1
130	14	3
140	20	4

Aufgabe 6: (6)

Erstellen Sie einen PL/pgSQL Trigger `checkCycles`, der beim Einfügen in die `depends`-Tabelle überprüft, ob eine zyklische Abhängigkeit besteht. Falls eine solche besteht, soll ein Fehler ausgegeben werden.

Wenn folgende (exemplarischen) SQL Befehle über der Instanz `courses` ausgeführt werden, soll folgendes Verhalten gezeigt werden.

- `INSERT INTO depends VALUES (100,140);`
  Ein Fehler soll ausgegeben werden, da 140 über 130 und 120 von 100 abhängig ist.
- `INSERT INTO depends VALUES (140,120);`
  Es soll kein Fehler ausgegeben und das Tuple eingefügt werden, obwohl bereits implizit 140 von 120 abhängig ist.

Sie können folgenden View verwenden, welcher alle expliziten und impliziten Abhängigkeiten liefert:

```
CREATE OR REPLACE VIEW requires AS
   WITH RECURSIVE temp(id, requires) AS (
         SELECT d.id, d.requires
         FROM depends d
      UNION
         SELECT t.id, d.requires
         FROM temp t JOIN depends d ON t.requires = d.id
)
SELECT *
FROM temp t;
```

```
CREATE OR REPLACE FUNCTION checkCycles() RETURNS TRIGGER AS $$
BEGIN
   IF EXISTS (SELECT *
              FROM requires r
              WHERE r.id = NEW.requires AND r.requires = NEW.id) THEN
      RAISE EXCEPTION 'Cycle!';
   END IF;

   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER t_checkCycles BEFORE INSERT ON depends
FOR EACH ROW EXECUTE PROCEDURE checkCycles();
```

Aufgabe 7: (8)

Geben Sie die Ausgabe der folgenden Java Methode `printMain` angewendet auf die Datenbankinstanz `courses` an.

Auf die exakte Formatierung mittels Leerzeichen brauchen Sie nicht zu achten, stellen Sie jedoch zur besseren Lesbarkeit sicher, dass sie für jede Zeile der Ausgabe (`println`) tatsächlich auch eine Zeile verwenden.

```
Connection c; PreparedStatement pStmt;

public void printMain( ) throws Exception{
   pStmt = c.prepareStatement("SELECT count(*) FROM depends WHERE id = ?");
   Statement stmt = c.createStatement();
   ResultSet rs = stmt.executeQuery("SELECT id FROM course ORDER BY id");

   while (rs.next()) {
      print(rs.getInt(1)); System.out.println();
   }

   rs.close(); stmt.close();
}

public void print(int i) throws Exception {
   int cnt = count(i);
   Statement stmt = c.createStatement();
   ResultSet rs = stmt.executeQuery("SELECT requires FROM depends WHERE id = " + i);

   System.out.print(i);
   while (rs.next()) {
      if (rs.isFirst()) {
         System.out.print(" > ");
         if (cnt > 1) System.out.print("(");
      } else
         System.out.print(" , ");
      print(rs.getInt(1));
      if (rs.isLast() && cnt > 1) System.out.print(")");
   }
   rs.close(); stmt.close();
}

private int count(int i) throws Exception {
   pStmt.setInt(1, i);
   ResultSet rs = pStmt.executeQuery();
   rs.next();
   return rs.getInt(1);
}
```

```
 100
 110
 120 > 100
 130 > (110 , 120 > 100)
 140 > 130 > (110 , 120 > 100)
```

Gesamtpunkte: 75

Sie können diese Seite abtrennen und brauchen sie nicht abzugeben!

Datenbankinstanz **courses**:

course			
id	name	points	ctype
100	FMod	6	VO
110	OS	2	VO
120	DM	6	VU
130	DB	6	VU
140	SSD	3	VU

depends	
id	requires
120	100
130	110
130	120
140	130