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**			10. 03. 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. und 2. Angenommen in Zeile 14 der Historie in Beispiel 2 würde an Stelle des commit der Transaktion T_1 ein rollback der Transaktion T_1 erfolgen.
   Dieses rollback würde zu einem kaskadierenden Rücksetzen der Transaktion T_2 führen. wahr ⊗ falsch ◯
   Dieses rollback würde zu einem kaskadierenden Rücksetzen der Transaktion T_3 führen. wahr ⊗ falsch ◯
3. Wenn beim Zweiphasen-Commit-Protokoll der Koordinator abstürzt, müssen die Agenten durch Kommunikation untereinander herausfinden, ob die Transaktion erfolgreich war oder nicht. wahr ◯ falsch ⊗
4. Betrachten Sie zwei Relationen $R(AB)$ und $S(AB)$. Dann gilt auf jeden Fall folgende Gleichheit: $(R \bowtie S) = (R \cap S)$. wahr ⊗ falsch ◯
5. Der Hybrid Hash Join von 2 Relationen mit b_1 bzw. b_2 Seiten benötigt im Idealfall $b_1 + b_2$ Seitenzugriffe. wahr ⊗ falsch ◯
6. Wenn beim Sortieren ein Datenbank-Puffer der Größe m zur Verfügung steht, können bei jedem merge-Schritt bis zu $m + 1$ runs zu einem run kombiniert werden. wahr ◯ falsch ⊗
7. Der Block-Nested-Loop Join erfordert immer eine größere oder bestenfalls gleich große Anzahl an Seitenzugriffen wie der Index-Nested-Loop Join. wahr ◯ falsch ⊗
8. 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 $R(AB)$, $R(BCD)$ und $R(DE)$ unterteilt wurde. Diese Fragmentierung ist rekonstruierbar. wahr ◯ falsch ⊗
9. Bei Erweiterung eines rein relationalen Datenbanksystems um das objektrelationale Feature "Mengenwertige Attribute" lassen sich unter Umständen Hilfstabellen für n:m-Relationen vermeiden. wahr ⊗ falsch ◯
10. Das WAL-Prinzip erfordert, dass vor dem Zurücksetzen einer Transaktion (rollback) alle zu dieser Transaktion gehörenden Log-Eintrage auf die Platte geschrieben werden müssen. 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: (15)

Gegeben ist die folgende Historie von Transaktionen:

Schritt	T_1	T_2	T_3	Log: [LSN, TA, PageID, Redo, Undo, PrevLSN] or ⟨LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN⟩
1	BOT			[#1, T_1, BOT, 0]
2		BOT		[#2, T_2, BOT, 0]
3			BOT	[#3, T_3, BOT, 0]
4	$r(A, a_1)$			
5	$w(A, 2 * a_1)$			[#4, T_1, P_A, A+=50, A-=50, #1] .....
6		$r(A, a_2)$		
7		$w(A, a_2 + 100)$		[#5, T_2, P_A, A+=100, A-=100, #2] ....
8		$w(B, a_2 + 200)$		[#6, T_2, P_B, B+=200, B-=200, #5] ....
9			$r(B, b_3)$	
10	$w(C, a_1)$			[#7, T_1, P_C, C-=100, C+=100, #4] ....
11			$r(C, c_3)$	
12			$w(C, b_3 + c_3)$	[#8, T_3, P_C, C+=300, C-=300, #3] ....
13		$r(A, a_2)$		
14	commit			[#9, T_1, commit, #7] .....
15				⟨#10, T_3, P_C, C-=300, #8, #3⟩ .....
16				⟨#11, T_2, P_B, B-=200, #6, #5⟩ .....
17				⟨#12, T_2, P_A, A-=100, #11, #2⟩ .....
18				⟨#13, T_3, (BOT), #10, 0⟩ .....
19				⟨#14, T_2, (BOT), #12, 0⟩ .....

In der obigen Historie bezeichnen A, B und C Daten in der Datenbank, während a_i, b_i und c_i lokale Variablen darstellen. Zu Beginn sei der relevante Datenbestand in der Datenbank $A = 50$, $B = 100$, und $C = 150$. Nehmen Sie an, dass unmittelbar nach Zeile 14 ein System-Absturz mit anschließendem Wiederanlauf passiert. Tragen Sie in den Zeilen 5ff. die entsprechenden Log-Einträge zu dieser Historie in die rechte Spalte der obigen Tabelle ein.

Dabei sind Undo/Redo-Einträge *relativ* zum Datenbestand mittels Addition bzw. Subtraktion anzugeben, z.B.: [#i, T_j, P_X, X+=d_1, X-=d_2, #k] bedeutet, dass laut i-tem Logeintrag die Transaktion T_j auf ein Datum X auf der Seite P_X schreibend zugreift, so dass beim Redo X um d_1 vergrößert werden müsste und beim Undo X um d_2 verkleinert werden müsste. Außerdem hat der vorangegangene Logeintrag dieser Transaktion die Nummer k.

Geben Sie in den Zeilen 15ff. die Log-Einträge für die Recovery an. Falls ein Kompensationseintrag Informationen über Undo und/oder Redo enthält, verwenden Sie auch hier die *relative* Schreibweise mittels Addition bzw. Subtraktion.

Aufgabe 3: (15)

Die Datenbank einer Autoverleih-Firma enthalte folgende Relationen:

Autos(<u>AutoNr</u>, Marke, Kategorie, PS, Miete) (kurz a),
Verkaeufer(<u>PersNr</u>, Name, Gehalt, Adresse) (kurz v) und
Mieten(<u>RechnungNr</u>, AutoNr, KundenNr, VerkaeuferNr, Datum) (kurz m).

Es ist die Anfrage

```
select v.Name, v.Gehalt, a.Miete
from Autos a, Verkaeufer v, Mieten m
where a.Miete > 100
and v.Gehalt < 1200
and a.AutoNr = m.AutoNr
and v.PersNr = m.VerkaeuferNr
```

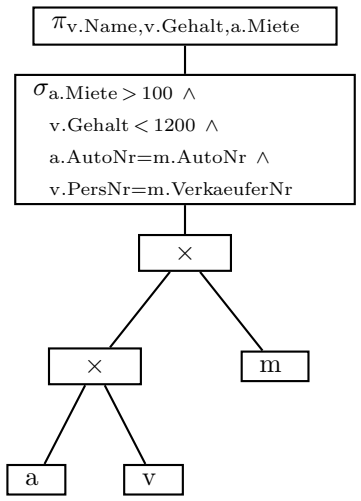
auszuführen (d.h. gesucht sind Informationen über Verkäufer, die teure Autos vermieten, selbst aber eher schlecht verdienen).

(a) Zeichnen Sie ins erste Kästchen den Operator-Baum für die kanonische Übersetzung. Um Platz zu sparen, verwenden Sie für die Relationen Autos, Verkaeufer und Mieten die Abkürzungen a, v und m.

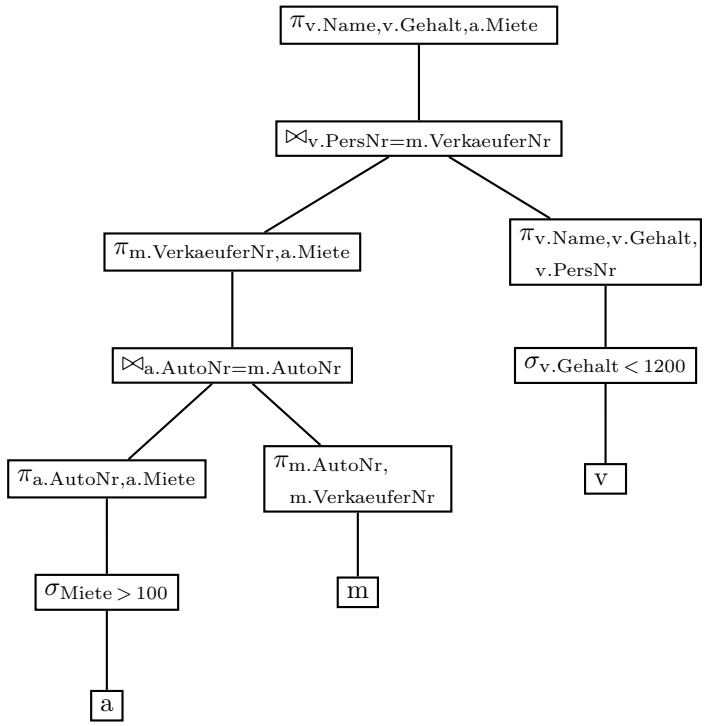
(b) Zeichnen Sie ins zweite Kästchen den Operator-Baum für den optimierten algebraischen Ausdruck. Wenden Sie für die Optimierung folgende Heuristiken an:

- Selektionen so weit wie möglich nach unten verschieben,
- Attribute, die nicht mehr benötigt werden, möglichst früh wegprojizieren,
- alle Kreuzprodukte durch Joins ersetzen.

(a) Kanonische Übersetzung:



(b) Optimierter Ausdruck:



Die folgende Datenbankbeschreibung gilt für die Aufgaben 4 – 7:

Die Stadtverwaltung möchte die einzelnen Objekte der Stadt (Bezirke, Strassen, Häuser, Wohnungen, etc.) in einer Datenbank verwalten. Dazu soll folgendes stark vereinfachtes Datenbankschema verwendet werden.

`Objekt`(oid, teilvon: *Objekt.oid*, typ, wert)
`Haus`(oid: *Objekt.oid*, groesse)

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

In der Tabelle `Objekt` werden die Objekte gespeichert. Jedes Objekt hat eine eindeutige Nummer `oid`. Diese eindeutige Nummer soll mittels der Sequenz `seq_oid` vergeben werden, bei 10 beginnen und in Zehnerschritten erhöht werden. Ein Objekt kann Teil eines anderen Objekts sein (zB.: ein Haus ist Teil einer Strasse), dazu wird die `oid` des übergeordneten Objekts in `teilvon` gespeichert. Weiters wird für jedes Objekt ein `typ` angegeben. Der `typ` muss zwingend einen der folgenden Werte annehmen: `stadt`, `bezirk`, `strasse`, `haus` und `wohnung`. Zusätzlich wird noch der nicht-negative `wert` des Objekts gespeichert.

In der Tabelle `Haus` werden Details zu Objekten des Typs `haus` gespeichert. Dazu muss die `oid` angegeben werden, und es wird die Größe des Hauses in `groesse` gespeichert.

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

Aufgabe 4: (7)

Geben Sie das CREATE SEQUENCE Statement für die Nummer des Objekts und die CREATE TABLE Statements mit allen nötigen Constraints für die zwei Tabellen an.

```
CREATE SEQUENCE seq_oid INCREMENT BY 10 MINVALUE 10 NO CYCLE;

CREATE TABLE Objekt (
        oid INTEGER PRIMARY KEY DEFAULT nextval('seq_oid'),
        teilvon INTEGER REFERENCES Objekt(oid),
        typ VARCHAR(10) CHECK (typ IN ('stadt','bezirk','strasse','haus','wohnung')),
    wert NUMERIC(11,2) CHECK (wert >= 0)
);

CREATE TABLE Haus (
        oid INTEGER PRIMARY KEY REFERENCES Objekt(oid),
        groesse INTEGER
);
```

Aufgabe 5: (5)

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

```
WITH RECURSIVE Temp(oid,teilvon) AS (
      SELECT oid, teilvon FROM Objekt
   UNION
      SELECT o.oid, t.teilvon FROM Objekt o, Temp t
      WHERE t.oid = o.teilvon
)
SELECT t.teilvon AS teilvon, COUNT(t.teilvon) AS cnt FROM Temp t
WHERE t.teilvon IS NOT NULL
GROUP BY t.teilvon
ORDER BY cnt DESC;
```

teilvon	cnt
10	8
20	7
30	5
60	3

Aufgabe 6: (6)

Betrachten Sie folgenden PL/pgSQL Trigger:

```
CREATE OR REPLACE FUNCTION tr() RETURNS TRIGGER AS $$
DECLARE
   change NUMERIC(11,2);
BEGIN
   IF TG_OP = 'INSERT' THEN
      --Bei einem Insert
      change = NEW.wert;
   ELSE
      --Bei einem Update
      change = NEW.wert - OLD.wert;
   END IF;
   UPDATE objekt SET wert = wert + change WHERE oid = NEW.teilvon;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER t_tr BEFORE UPDATE OR INSERT ON Objekt FOR EACH ROW EXECUTE PROCEDURE tr();
```

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

```
INSERT INTO Objekt VALUES (100,50,'wohnung',12000);

UPDATE Objekt SET wert = wert - 5000 WHERE oid = 80;

SELECT oid, wert FROM Objekt ORDER BY oid;
```

oid	wert
10	390000
20	390000
30	390000
40	0
50	212000
60	178000
70	20000
80	20000
90	18000
100	12000

Aufgabe 7: (12)

Vervollständigen Sie die Java Methode `insertHaus`, die ein neues Haus in die Tabellen `Objekt` und `Haus` einfügt und der folgende Parameter übergeben werden:

- `teilvon` – Die `oid` des Objekts, wovon das Haus ein Teil ist.
- `wert` – Der Wert des Hauses.
- `groesse` – Die Größe des Hauses.

Es soll das Haus in der `Objekt`-Tabelle und in der `Haus`-Tabelle angelegt werden. Falls die Größe **100** überschreitet, soll zusätzlich eine Wohnung in diesem Haus mit dem Wert `groesse`*1.5 angelegt werden.

Hinweis: Nach dem Einfügen eines Objekts können Sie mittels `currval('seq_oid')` den letzten vergebenen Wert der Sequenz `seq_oid` ermitteln.

Verwenden Sie ausschließlich PreparedStatements. Für die volle Punktezahl legen Sie maximal **3** PreparedStatements an; für jedes weitere PreparedStatement werden Ihnen **2** Punkte abgezogen. Sie können hier auf eine Connection `c` zugreifen.

```
PreparedStatement psInsHaus = c.prepareStatement(
   "INSERT INTO Haus VALUES ( currval('seq_oid'), ?)");

PreparedStatement psInsObjekt = c.prepareStatement(
   "INSERT INTO Objekt (teilvon,typ,wert) VALUES (?,?,?)");

PreparedStatement psSelOid =  c.prepareStatement(
   "SELECT currval('seq_oid');");
```

Vervollständigen Sie nun die Methode `insertHaus`. Sie können die **oben angelegten** PreparedStatements verwenden. Um die Fehlerbehandlung brauchen Sie sich nicht zu kümmern.

```
public static void insertHaus(int teilvon, int wert, int groesse) throws Exception {
   psInsObjekt.setInt(1,teilvon);
   psInsObjekt.setString(2,"haus");
   psInsObjekt.setInt(3,wert);
   psInsObjekt.executeUpdate();

   ResultSet rs = psSelOid.executeQuery();
   rs.next();
   int iHausOid = rs.getInt(1);
   rs.close();

   psInsHaus.setInt(1,groesse);
   psInsHaus.executeUpdate();

   if (groesse > 100) {
      psInsObjekt.setInt(1, iHausOid);
      psInsObjekt.setString(2,"wohnung");
      psInsObjekt.setDouble(3,groesse*1.5);
      psInsObjekt.executeUpdate();
   }
}
```

Gesamtpunkte: 75

Sie können diese Seite abtrennen und brauchen sie nicht abzugeben!

Datenbankinstanz **objekte**:

Objekt			
oid	teilvon	typ	wert
10	NULL	stadt	383000
20	10	bezirk	383000
30	20	strasse	383000
40	20	strasse	0
50	30	haus	200000
60	30	haus	183000
70	60	wohnung	20000
80	60	wohnung	25000
90	60	wohnung	18000

Haus	
oid	groesse
50	250