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**			15. 01. 2016
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. Der Index Nested Loop Join kann unter Umständen höhere I/O-Kosten als der Pagewise Nested Loop Join haben.
   wahr ⊗ falsch ◯
2. Wenn der Wartegraph einer Historie zyklisch ist, besteht ein Deadlock, das sich durch das Zurücksetzen einer Transaktion beseitigen lässt. wahr ◯ falsch ⊗
3. Nehmen Sie an, dass beim Zweiphasen-Commit-Protokoll einer der Agenten nach Empfang der PREPARE-Nachricht und vor dem Absenden einer Antwort (also READY bzw. FAILED) abstürzt. Dann muss der Agent beim Wiederanlauf auf jeden Fall auf eine COMMIT- bzw. ABORT-Nachricht des Koordinators warten, um zu entscheiden, ob die Transaktion erfolgreich war. wahr ◯ falsch ⊗
4. Bei den Einstellungen steal/force ist im Falle eines Recovery kein Redo nötig. wahr ⊗ falsch ◯
5. Bei der Historie $r_1(A)$, $r_2(B)$, $w_2(B)$, $r_1(B)$, a_2 führt der Abbruch der Transaktion T_2 zu einem kaskadierenden Rücksetzen von T_1. wahr ⊗ falsch ◯
6. Eine Relation R sei an 5 Netzwerk-Knoten materialisiert mit den Gewichten 1, 1, 1, 1 und 1. Dann entsprechen die Lese- bzw. Schreib-Quoren $Q_r(R) = 3$ bzw. $Q_w(R) = 3$ dem "majority consensus" Verfahren. wahr ⊗ falsch ◯
7. Die Verwendung des objektrelationalen Features "Geschachtelte Relationen" erlaubt bei Anfragen bezüglich n:m-Relationen unter Umständen einen Auswertungsplan mit einer geringeren Anzahl von Joins. wahr ⊗ falsch ◯
8. Betrachten Sie die Relationen $R(AB)$ mit 100 Tupeln und $S(\underline{A}C)$ mit 200 Tupeln, d.h.: Attribut A ist der Primärschlüssel der Relation S. Dann liefert der Ausdruck $R \bowtie S$ auf jeden Fall zwischen 0 und 100 Tupeln. wahr ⊗ falsch ◯
9. Betrachten Sie zwei Relationen $R(AB)$ und $S(AC)$. Dann gilt auf jeden Fall folgende Beziehung: $(R \bowtie S) = (\pi_A(R) \cap \pi_A(S)) \times (\pi_B(R) \times \pi_C(S))$. wahr ◯ falsch ⊗
10. Falls der Wiederanlauf nach einem Systemabsturz durch einen neuerlichen Systemabsturz unterbrochen wird, muss das Datenbanksystem beim zweiten Wiederanlauf auf jeden Fall noch einmal die Redos des ersten Wiederanlaufs abarbeiten.
    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)

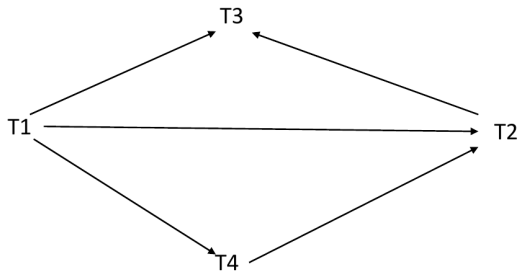
(a) Eine Historie sei gegeben durch folgende Folge von Elementaroperationen: $r_1(A)$, $w_1(D)$, $r_2(D)$, $r_1(C)$, $r_4(B)$, $w_2(A)$, $w_4(C)$, $w_2(B)$, $r_3(B)$, $w_2(C)$, $w_3(D)$, c_1, c_2, c_3, c_4.

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).

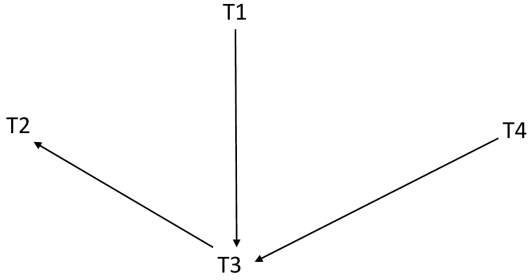
(b) Betrachten Sie die folgende Folge von Sperranforderungen, wobei "lockS_i(O)"(bzw. "lockX_i(O)") bedeutet, dass die Transaktion T_i eine Lesesperre (bzw. eine Schreibsperre) auf das Datenobjekt O anfordert: lockX_1(A), lockS_2(C), lockS_3(B), lockX_1(B), lockX_3(C), lockX_4(B), lockS_4(A).

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) Geben Sie für den Serialisierbarkeitsgraphen aus (a) eine mögliche topologische Sortierung an:

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

Aufgabe 3: (18)

Ein internationales Unternehmen mit Standorten in Wien und Zürich verwaltet eine verteilte Datenbank. In dieser Datenbank sind folgende Tabellen enthalten:

Mitarbeiter(MID, name, geb, wohnort, stelle, gehalt, beschäftigungsdauer)
Abteilung(AID, manager, ort, budget)

Die Tabelle Mitarbeiter wird (vertikal) fragmentiert in folgende 2 Tabellen:
MitarbeiterPrivat(MID, name, geb, wohnort) und
MitarbeiterBeruf(MID, stelle, gehalt, beschäftigungsdauer).

Die Tabelle Abteilung(AID, manager, ort, budget) wird nach dem Attribut "ort" (horizontal) fragmentiert in die zwei Tabellen AbteilungWien und AbteilungZürich.

Es ist die Anfrage

```
select m.gehalt
from Abteilung a, Mitarbeiter m
where a.manager = m.MID and m.geb > 1980 and m.gehalt > a.budget/10
```

auszuführen (d.h. gesucht sind die Gehälter von jungen Managern, die relativ gut verdienen).

(a) Zeichnen Sie ins erste Kästchen den Operator-Baum für die kanonische Übersetzung. Verwenden Sie für die 2 Fragmente der Mitarbeiter-Tabelle sowie für die 2 Fragmente der Abteilung-Tabelle folgende Abkürzungen:

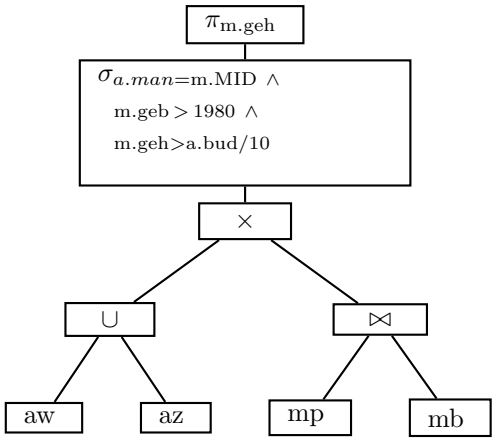
mp (MitarbeiterPrivat), **mb** (MitarbeiterBeruf),
aw (AbteilungWien) und **az** (AbteilungZürich).

Außerdem können Sie alle Attributnamen jeweils auf die ersten 3 Buchstaben abkürzen, z.B.: verwenden Sie 'man' statt 'manager', 'geh' statt 'gehalt', etc.

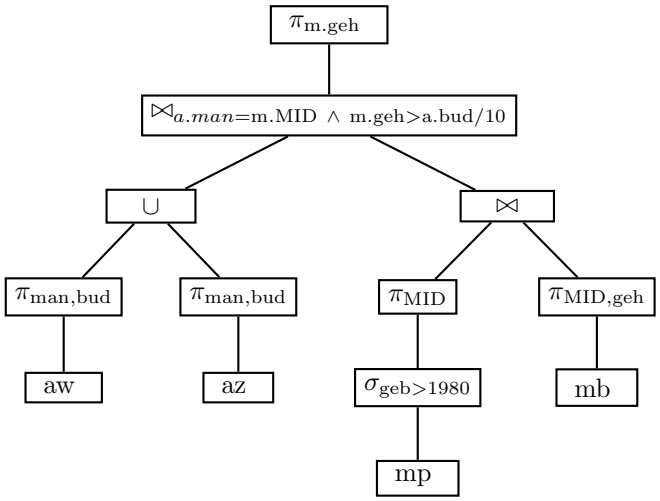
(b) Zeichnen Sie ins zweite Kästchen den Operator-Baum für einen optimierten algebraischen Ausdruck. Wenden Sie für die Optimierung folgende Heuristiken an:

- Selektionen so weit wie möglich nach unten verschieben,
- Kreuzprodukte durch Joins ersetzen,
- 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 ein neues soziales Netzwerk soll folgendes stark vereinfachtes Datenbankschema verwendet werden:

`users`(uid, uname, age)
`groups`(gid, gname, admin: *users.uid*)
`teil`(uid: *users.uid*, gid: *groups.gid*)

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

In der Tabelle `users` werden die Daten der Benutzer gespeichert. Jeder Benutzer wird eindeutig durch die Nummer `uid` identifiziert. Weiters, wird der Name `uname` und das Alter `age` des Benutzers gespeichert. Das Alter `age` muss einen Wert größer als 0 annehmen.

In der Tabelle `groups` werden die Daten von verschiedenen Gruppen gespeichert. Jede Gruppe wird durch die Nummer `gid` eindeutig identifiziert und hat einen Namen `gname`. Zusätzlich wird für jede Gruppe ein Leiter festgelegt. Dessen `uid` wird in der Spalte `admin` gespeichert.

Die Tabelle `teil` bildet die Gruppenmitgliedschaften der einzelnen Benutzer ab. Ein Tupel in der Tabelle `teil` bedeutet, dass der Benutzer mit der Nummer `uid` an der Gruppe mit der Nummer `gid` teilnimmt. Beide Attribute sind Teil des Primärschlüssels.

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

Aufgabe 4: (8)

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

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

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

```
CREATE TABLE users (
    uid INTEGER PRIMARY KEY,
    uname VARCHAR(50) NOT NULL,
    age INTEGER CHECK (age > 0)
);

CREATE TABLE groups (
    gid INTEGER PRIMARY KEY DEFAULT nextval('seq_gids'),
    gname VARCHAR(50) NOT NULL,
    admin INTEGER REFERENCES users(uid)
);

CREATE TABLE teil (
    uid INTEGER REFERENCES users(uid),
    gid INTEGER REFERENCES groups(gid),
    PRIMARY KEY (uid, gid)
);
```

Aufgabe 5: (8)

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

```
CREATE OR REPLACE VIEW view1 AS
   WITH RECURSIVE temp(id1, id2) AS (
      SELECT t1.uid as id1, t2.uid as id2
      FROM teil t1 JOIN teil t2 ON t1.gid = t2.gid
   UNION
      SELECT t1.id1 as id1, t2.uid as id2
      FROM temp t1 JOIN teil t ON t1.id2 = t.uid
                   JOIN teil t2 ON t.gid = t2.gid
   )
SELECT id1, id2
FROM temp t
WHERE id1 <> id2;

SELECT uid, COUNT(id2)
FROM view1 v RIGHT JOIN users u ON v.id1 = u.uid
GROUP BY uid
ORDER BY uid;
```

uid	count
1	1
2	3
3	3
4	3
5	3
6	1

Aufgabe 6: (6)

Erstellen Sie einen PL/pgSQL Trigger `groupAdmin`, der nach dem Einfügen in die `teil`-Tabelle überprüft, ob der erste Gruppenteilnehmer eingefügt wurde. Fall dies der Fall ist, wird der Benutzer Leiter dieser Gruppe (in der Tabelle `groups`).

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

- `INSERT INTO teil VALUES (4,140);`
  Da noch kein Benutzer Teil der Gruppe mit der Nummer "140" ist, wird der Benutzer mit der Nummer "4" dessen Leiter, d.h. der Eintrag in der Tabelle `groups` wird geändert.
- `INSERT INTO teil VALUES (4,100);`
  Da die Gruppe mit der Nummer "100" bereits Teilnehmer besitzt, wird der Leiter nicht verändert.

```
CREATE OR REPLACE FUNCTION groupAdmin() RETURNS TRIGGER AS $$
BEGIN
   IF NOT EXISTS (SELECT * FROM teil WHERE gid = NEW.gid AND uid <> NEW.uid) THEN
      UPDATE groups SET admin=NEW.uid WHERE gid=NEW.gid;
   END IF;

   RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER t_groupAdmin AFTER INSERT ON teil
   FOR EACH ROW EXECUTE PROCEDURE groupAdmin();
```

Aufgabe 7: (8)

Geben Sie die Ausgabe der folgenden Java Methode `printMain` angewendet auf die Datenbankinstanz `network` 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 DISTINCT(u.uname) "
                             + "FROM teil t JOIN users u ON t.uid = u.uid "
                             + "WHERE t.gid IN (SELECT gid FROM teil WHERE uid = ?) AND "
                             + "      u.uid <> ?"
                             + "ORDER BY uname");

    Statement stmt = c.createStatement();
    ResultSet rs1 = stmt.executeQuery("SELECT uid, uname FROM users ORDER BY uname");

    while (rs1.next()) {
        System.out.print(rs1.getString(2));

        pStmt.setInt(1, rs1.getInt(1));
        pStmt.setInt(2, rs1.getInt(1));

        ResultSet rs2 = pStmt.executeQuery();
        while (rs2.next()) {
            if (rs2.isFirst())
                System.out.print(": ");

            System.out.print(rs2.getString(1));

            if (rs2.isLast())
                System.out.println();
            else
                System.out.print(", ");
        }

        rs2.close();
    }

    rs1.close();
    stmt.close();
}
```

```
Anna: Max
Franz: Karl, Maria
Herta: Maria
Karl: Franz
Maria: Franz, Herta
Max: Anna
```

Gesamtpunkte: 75

Sie können diese Seite abtrennen und brauchen sie nicht abzugeben!

Datenbankinstanz **network**:

users		
uid	uname	age
1	Anna	28
2	Franz	25
3	Maria	26
4	Karl	19
5	Herta	23
6	Max	22

groups		
gid	gname	admin
100	Group 1	2
110	Group 2	3
120	Group 4	2
130	Group 5	6
140	Group 6	

teil	
uid	gid
2	100
3	100
3	110
5	110
2	120
4	120
6	130
1	130