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**			04. 05. 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. Alle Constraints, die per ALTER TABLE Statement angelegt werden, hätten in jedem Fall auch bereits beim CREATE TABLE Statement angelegt werden können. wahr ◯ falsch ⊗
2. Nehmen Sie an, dass eine Relation R aus 20000 Seiten besteht und dass die Puffergröße 100 beträgt. Nehmen Sie weiters an, dass eine Hash-Funktion eine gleichmäßige Verteilung auf die gewünschte Anzahl von Buckets liefert. 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 ⊗
3. In der Kostenformel $b_R + c \cdot M_R$ für den Index Nested Loop Join ist c eine Konstante, die von der Art des Indexes der Relation R abhängt. wahr ◯ falsch ⊗
4. Betrachten Sie zwei Relationen $U(CD)$ und $V(CD)$. Dann gilt auf jeden Fall folgende Gleichheit (d.h.: Vertauschung von Projektion und Mengendifferenz): $\pi_C(U - V) = \pi_C(U) - \pi_C(V)$ wahr ◯ falsch ⊗
5. Sei A ein Attribut der Relation R aber nicht von Relation S und sei B ein Attribut der Relation S aber nicht von Relation R. Dann gilt in jedem Fall folgende Gleichheit (d.h.: Vertauschung von Projektion und kartesischem Produkt): $\pi_{AB}(R \times S) = \pi_A(R) \times \pi_B(S)$ wahr ⊗ falsch ◯
6. In einem verteilten DBMS benötigt ein einzelner Agent niemals eine Kommunikation mit anderen Agenten, um nach seinem eigenen Absturz die „winner“ und „loser“ Transaktionen zu erkennen. wahr ⊗ falsch ◯
7. Die Historie $r_1(C)$, $r_2(C)$, $w_2(D)$, $r_1(D)$, $w_1(C)$, c_1, c_2 ist zwar beim Zwei-Phasen Sperrprotokoll jedoch nicht beim strengen Zwei-Phasen Sperrprotokoll möglich. wahr ⊗ falsch ◯
8. Die Historie $r_1(C)$, $r_2(C)$, $w_2(D)$, $r_1(D)$, $w_1(C)$, c_1, c_2 ist serialisierbar. wahr ⊗ falsch ◯
9. In der Historie $r_1(C)$, $r_2(C)$, $w_2(D)$, $r_1(D)$, $w_1(C)$, a_1 führt der Abbruch der ersten Transaktion zu einem kaskadierenden Rücksetzen der zweiten Transaktion. wahr ◯ falsch ⊗
10. Nehmen Sie an, dass eine relationale Datenbank um das objektrelationale Feature “Geschachtelte Relationen” erweitert wurde. Dann lassen sich $n{:}\,m$ Beziehungen unter Umständen – im Gegensatz zu rein relationalen Datenbanken in 3. Normalform – 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: (18)

Eine international tätige Firma mit Standorten in Europa und Amerika will eine verteilte Personaldatenbank erstellen. In dieser Datenbank sind folgende Tabellen enthalten:

Angestellter(SSN, name, fkt, geschlecht, geburtsdatum, standort, adresse)
Reisebudget(fkt, adresse, budget).

Die Angestelltentabelle wird (vertikal) fragmentiert in folgende zwei Tabellen:

AngestellterBasic(SSN, name, fkt, geschlecht, standort) und
AngestellterExtra(SSN, geburtsdatum, adresse).

Außerdem wird die Tabelle AngestellterBasic(SSN, name, fkt, geschlecht, standort) nach dem Attribut "standort" (horizontal) weiter fragmentiert in die zwei Tabellen AngestellterBasicEuropa und AngestellterBasicAmerika. Es ist die Anfrage

select name
from Angestellter a, Reisebudget r
where a.fkt = 'Abteilungsleiter' and r.budget > 20k and a.fkt = r.fkt and a.adresse = r.adresse

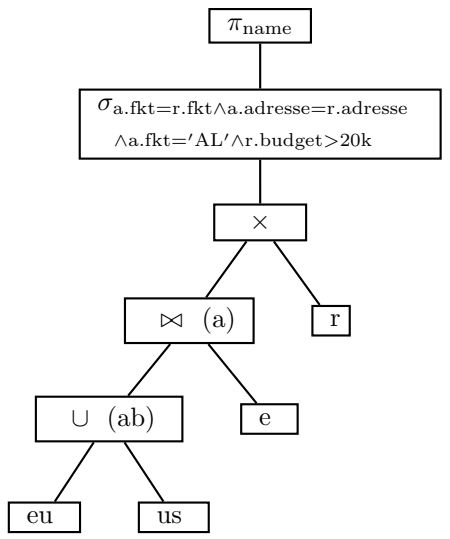
auszuführen (d.h. Informationen über Abteilungsleiter mit einem hohen Reisebudget).

(a) Zeichnen Sie ins erste Kästchen den Operatorbaum für die kanonische Übersetzung. Verwenden Sie für die 3 Fragmente der Tabelle Angestellter sowie für die Tabelle Reisebudget folgende Abkürzungen: **e** (AngestellterExtra), **eu** (AngestellterBasicEuropa), **us** (AngestellterBasicAmerika) und **r** (Reisebudget). Außerdem können Sie den String 'Abteilungsleiter' mit 'AL' abkürzen.

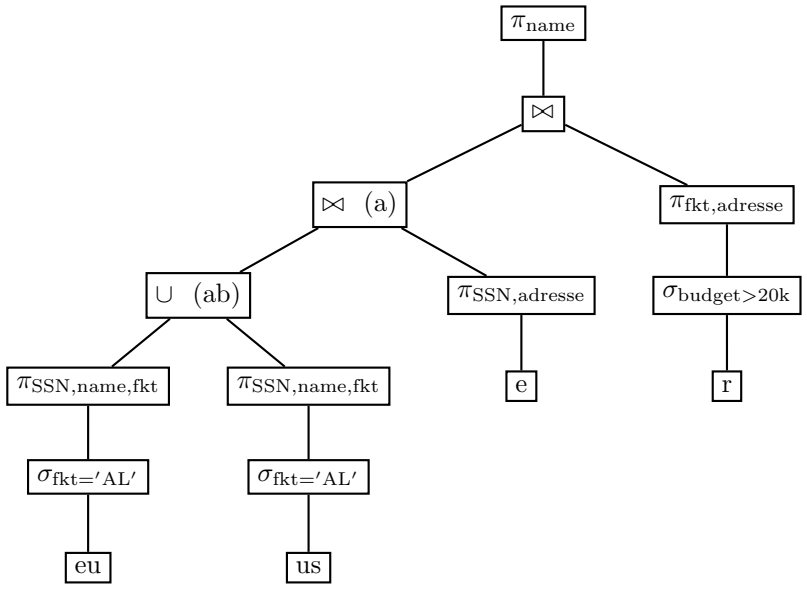
(b) Zeichnen Sie ins zweite Kästchen den Operator-Baum für den optimierten algebraischen Ausdruck. Wenden Sie für die Optimierung folgende Heuristiken an:

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

(a) Kanonische Übersetzung:



(b) Optimierter Ausdruck:



Aufgabe 3: (12)

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 ein striktes Zwei-Phasen-Sperrprotokoll verwendet wird und die 4 Isolation Levels laut SQL Standard (d.h. Read Uncommitted, Read Committed, Repeatable Read, and Serializable) gesondert implementiert sind.

Gegeben sind zwei Transaktionen:

Transaktion 1:

```
SELECT * FROM A;
SELECT * FROM B;
COMMIT;
```

Transaktion 2:

```
UPDATE B SET wert = 200;
UPDATE A SET wert = 100;
COMMIT;
```

(a) Bei welchen Isolation Levels kann es zu einem Deadlock kommen?

○ Read Uncommitted ○ Read Committed ⊗ Repeatable Read ⊗ Serializable

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

```
UPDATE A SET wert = 100;
UPDATE B SET wert = 200;
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;
```

(c) Bei welchen Isolation Levels kann es nun zu einem Deadlock kommen?

⊗ Read Uncommitted ⊗ Read Committed ⊗ Repeatable Read ⊗ Serializable

Die folgende Datenbankbeschreibung gilt für die Aufgaben 4 - 6: Gegeben ist folgendes stark vereinfachtes Datenbankschema für ein Unternehmen:

mitarbeiter(mid, name, gehalt, etage, abteilung: *abteilung.aid*) und
abteilung(aid, name, leiter: *mitarbeiter.mid*)

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

Jeder Mitarbeiter hat eine eindeutige ID (`mid`), einen vollständigen Namen (`name`), ein Gehalt (`gehalt`), arbeitet in einer bestimmten Etage (`etage`) und in einer bestimmten Abteilung (`abteilung`). Die Tabelle `abteilung` enthält für jede Abteilung eine eindeutige ID (`aid`), einen Namen (`name`) und die ID des Abteilungsleiters (`leiter`).
Der Wertebereich für das Attribut `etage` soll zwischen -2 und 5 liegen. Der Datentyp für das Gehalt soll zwei Nachkommastellen aufweisen. Wählen Sie entsprechende Datentypen (Integer, Varchar) für die restlichen Attribute.

Aufgabe 4: SQL (6)
Geben Sie die Create Table Statements mit allen nötigen Constraints für die Tabellen `mitarbeiter` und `abteilung` an. Beachten Sie eventuelle zyklische Abhängigkeiten zwischen den Fremdschlüsseln.

```
CREATE TABLE mitarbeiter (
  mid INTEGER PRIMARY KEY,
  name VARCHAR(50),
  gehalt NUMERIC(7,2),
  etage INTEGER check(etage between -2 and 5),
  abteilung INTEGER
);

CREATE TABLE abteilung (
  aid INTEGER PRIMARY KEY,
  name VARCHAR(50),
  leiter INTEGER
);

ALTER TABLE mitarbeiter
  ADD CONSTRAINT fk_abteilung
    FOREIGN KEY(abteilung) REFERENCES abteilung (aid) DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE abteilung
  ADD CONSTRAINT fk_leiter
    FOREIGN KEY(leiter) REFERENCES mitarbeiter (mid) DEFERRABLE INITIALLY DEFERRED;
```

Aufgabe 5: SQL Abfrage (7)

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

```
WITH RECURSIVE temp(mid1, mid2, etage) AS (
   SELECT m1.mid as mid1, m2.mid as mid2, m2.etage
    FROM mitarbeiter m1 JOIN mitarbeiter m2 ON m1.etage = m2.etage + 1
 UNION
   SELECT t.mid1 as mid1, m2.mid as mid2, m2.etage
    FROM temp t JOIN mitarbeiter m2 ON t.etage = m2.etage + 1
)
SELECT mid1, mid2
FROM temp t
ORDER BY mid1, mid2;;
```

mid1	mid2
2	1
2	4
3	1
3	2
3	4

Aufgabe 6: Java (10)

Vervollständigen Sie die Java Methode `printStatistik`, die für jede Abteilung ausgibt, wie viel die Mitarbeiter in der jeweiligen Abteilung in Summe verdienen und zwar aufgegliedert nach den Etagen in denen Sie arbeiten. Zusätzlich ist noch die Gesamtsumme des Gehalts dieser Abteilung auszugeben. Sortieren Sie nach Abteilungsname aufsteigend.

Beachten Sie folgende Beispielausgabe der Abteilung 'Marketing':

```
Marketing:
> 3: 1850.0
> 2: 1500.0
Summe: 3350.0
```

Die Mitarbeiter der Abteilung 'Marketing', welche sich in der Etage 3 befinden, verdienen in Summe also 1850.0.

Vervollständigen Sie die Methode `printStatistik` in der Form, dass für alle Abteilungen diese Art der Ausgabe erreicht wird. Schließen Sie jene Mitarbeiter von dieser Statistik aus, welche keine zugewiesene Etage haben.

Beachten Sie dazu folgende Hinweise:

- um Fehlerbehandlung und die genaue Formatierung der Ausgabe brauchen Sie sich nicht zu kümmern
- Verwenden Sie zur Ausführung von SQL Ausdrücken ausschliesslich die beiden unten angegebenen PreparedStatements. Diese werden außerhalb der Methode angelegt, und können bei jedem Aufruf der Methode verwendet werden.

Vervollständigen Sie nun die für die Methode `printStatistik` benötigten PreparedStatements.

```
PreparedStatement abteilungen = c.prepareStatement('select aid, name from abteilung
  order by name;');

PreparedStatement etagenSummen = c.prepareStatement('select etage, sum(gehalt) as summe
  from mitarbeiter where abteilung = ? and etage is not null group by etage;');
```

Vervollständigen Sie nun die Methode `printStatistik`. Sie können dabei auf alle eben definierten PreparedStatements zugreifen.

```
public void printStatistik() throws Exception {

  ResultSet rs = abteilungen.executeQuery();
  while (rs.next()) {
    int aid = rs.getInt('aid');
    double sum = 0;
    System.out.println(rs.getString('name') + ':');

    stmt2.setInt(1, aid);
    ResultSet rs2 = pstmt2.executeQuery();

    while(rs2.next()) {
      sum += rs2.getDouble('summe');
      System.out.println('> ' + rs2.getInt('etage') + ': ' + rs2.getDouble('summe'));
    }
    System.out.println('Summe: ' + sum);
    rs2.close();
  }

  rs.close();
}
```

Aufgabe 7: PL/SQL Trigger (7)

Nehmen Sie an, dass eine Datenbank wie folgt definiert wurde.

```
CREATE TABLE messwerte (
  id INTEGER PRIMARY KEY,
  typ INTEGER NOT NULL,
  wert INTEGER
);

CREATE FUNCTION fTrigger1()
RETURNS trigger AS $$
BEGIN
  CASE NEW.typ
    WHEN 1 THEN
      DELETE FROM messwerte WHERE wert > 1;
    WHEN 3 THEN
      NEW.wert = NEW.wert + 1;
    WHEN 2 THEN
      DELETE FROM messwerte WHERE typ=2;
    ELSE
      NULL;
  END CASE;
  NEW.wert = 0;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION fTrigger2()
RETURNS trigger AS $$
BEGIN
  IF (NEW.wert != OLD.wert) THEN
    NEW.typ = OLD.typ + 1;
    NEW.wert = OLD.wert + 1;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tTrigger1
AFTER INSERT ON messwerte
FOR EACH ROW EXECUTE PROCEDURE fTrigger1();

CREATE TRIGGER tTrigger2
BEFORE UPDATE ON messwerte
FOR EACH ROW EXECUTE PROCEDURE fTrigger2();
```

Geben Sie an, wie die Tabelle `messwerte` nach jedem der folgenden Befehlsblöcke aussieht.

```
INSERT INTO messwerte VALUES (1, 2, 3);
INSERT INTO messwerte VALUES (2, 4, 2);
INSERT INTO messwerte VALUES (3, 1, 3);
INSERT INTO messwerte VALUES (4, 3, 4);
INSERT INTO messwerte VALUES (5, 2, 3);
```

id	typ	wert
4	3	4

```
UPDATE messwerte SET typ = 2 WHERE id = 4;
UPDATE messwerte SET wert = wert + 2 WHERE id = 4;
```

id	typ	wert
4	3	5

```
INSERT INTO messwerte VALUES (6, 1, 1);
```

id	typ	wert
6	1	1

Gesamtpunkte: 75

Sie können diese Seite abtrennen und brauchen sie nicht abzugeben!

Datenbankinstanz **abteilungen**:

mitarbeiter				
mid	**name**	**gehalt**	**etage**	**abteilung**
1	Huber	2000.00	1	1
2	Mayer	1500.00	2	3
3	Hofer	1850.00	3	3
4	Gruber	2900.00	1	4
5	Lackner	900.00	-1	2
6	Grabner	4500.00	NULL	2
7	Walder	5500.00	NULL	2
8	Faller	800.00	-1	2

abteilung		
aid	**name**	**leiter**
1	Management	1
2	Entwicklung	7
3	Marketing	3
4	Verkauf	4