Gruppe A

Bitte tragen Sie **sofort** und **leserlich** Namen, Studienkennzahl und Matrikelnummer ein und legen Sie Ihren Studentenausweis bereit.

PRÜFUNG AUS DATENBANKSYSTEME **VL 181.186**			21. 4. 2011
Kennnr.	Matrikelnr.	Familienname	Vorname

Arbeitszeit: 120 Minuten. Aufgaben sind auf den Angabeblättern zu lösen; Zusatzblätter werden nicht gewertet.

Aufgabe 1: (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
```

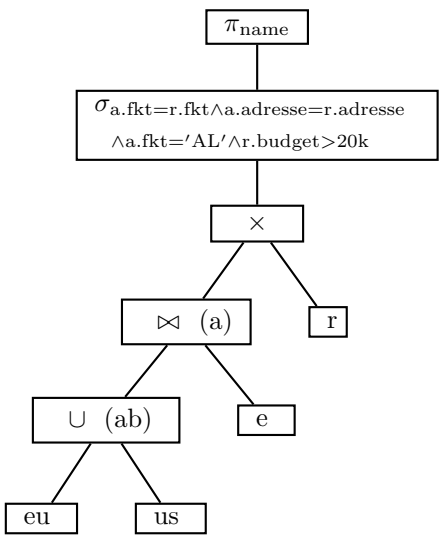
auszuführen (d.h. Informationen über Abteilungsleiter mit höherem Reisebudget).

(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.

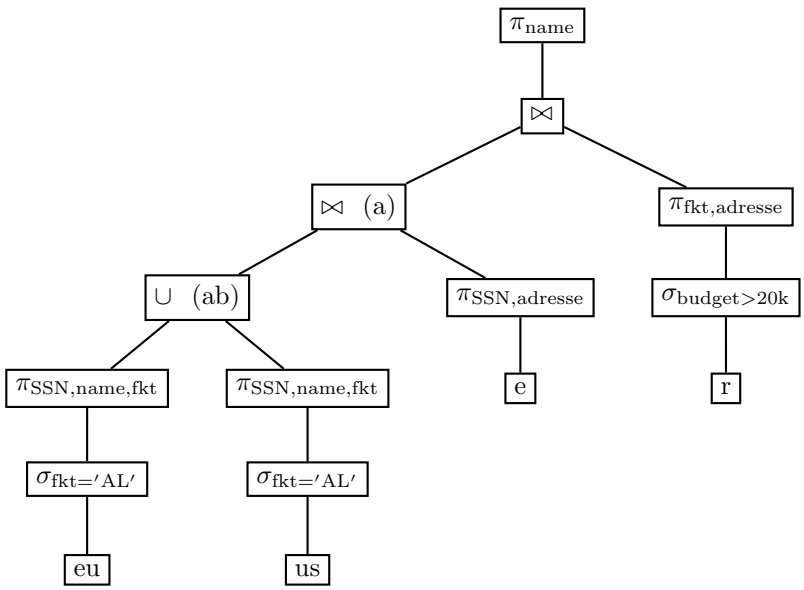
(b) Zeichnen Sie ins zweite Kästchen den Operator-Baum für den optimierten algebraischen Ausdruck. Wenden Sie für die Optimierung folgende Heuristiken an:

- 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 2: (15)

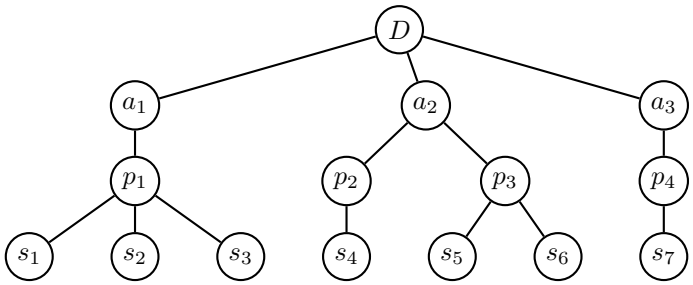
Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind.

1. Nehmen Sie an, dass eine Relation R 65535 ($= 2^{16} - 1$) Seiten umfasst und die Puffergröße 128 beträgt. 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 ⊗
2. Bei einem nicht perfekt balancierten B+ Baum vom Grad k mit n Elementen kann unter Umständen das Einfügen bzw. Löschen von einem Element länger als $O(\log_k n)$ dauern. wahr ◯ falsch ⊗
3. 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 ⊗
4. Die Anzahl der Zyklen im Wartegraphen entspricht immer der Anzahl der Transaktionen, die zurückgesetzt werden müssen, um ein Deadlock aufzulösen. wahr ◯ falsch ⊗
5. Eine Relation R sei an 6 Netzwerk-Knoten materialisiert mit den Gewichten 5, 4, 3, 7, 2, und 9. Dann sind $Q_r(R) = 12$ und $Q_w(R) = 19$ ungültige Lese- bzw. Schreib-Quoren. wahr ◯ falsch ⊗
6. Mit SQL-92 lassen sich Anfragen formulieren, die man mit PL/pgSQL nicht formulieren könnte. wahr ◯ falsch ⊗
7. Betrachten Sie zwei Relationen $R(AB)$ und $S(AB)$. Dann gilt auf jeden Fall folgende Gleichheit: $\pi_A(R-S) = \pi_A R - \pi_A S$ wahr ◯ falsch ⊗
8. Es gibt Relationen $R(\underline{A}B)$ mit 10 Tupeln und $S(AC)$ mit 100 Tupeln, für die der Ausdruck $R \bowtie S$ 1000 Tupeln ergibt. wahr ◯ falsch ⊗
9. In einem verteilten DBMS benötigt ein einzelner Agent unter Umständen eine Kommunikation mit anderen Agenten, um nach einem Absturz die „winner“ und „loser“ Transaktionen zu erkennen. wahr ◯ falsch ⊗
10. Bei Verwendung von RAID Level 6 mit Reed-Solomon Code ist auf jeden Fall die *durability* Eigenschaft von ACID Transaktionen erfüllt. 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 3: (12)

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 inkorrekter Antwort -1,5 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

Die folgende Datenbankbeschreibung gilt für die Aufgaben 4 – 7:

Gegeben ist folgendes stark vereinfachtes Datenbankschema, das Datei- und Verzeichniseinträge in einem Dateisystem repräsentiert.

eintrag(id, e: *eintrag.id*, name, typ, groesse)

Jeder Datei-/Verzeichniseintrag `eintrag` hat eine eindeutige Identifikationsnummer `id` und ein Elternverzeichnis `e`. Weiters gespeichert wird der `name` der Datei oder des Verzeichnisses, der `typ` als Text, und die `groesse` als Zahl.

Wählen Sie entsprechende Datentypen (INTEGER, VARCHAR) für die Attribute.

Aufgabe 4: (6)

Geben Sie nun ein CREATE-Statement mit allen entsprechenden Constraints für die Tabelle `eintrag` an.

```
CREATE TABLE eintrag (
    id INTEGER PRIMARY KEY,
    e INTEGER REFERENCES eintrag(id),
    name VARCHAR(30),
    typ VARCHAR(30),
    groesse INTEGER
);
```

Geben Sie ein UPDATE Statement an, das den Typ aller Einträge auf `'V'` (Verzeichnis) ändert, sofern der derzeitige Typ nicht `'D'` (Datei) ist. Vergessen Sie nicht, dass als derzeitiger Wert auch `null` vorkommen kann.

```
UPDATE eintrag
    SET typ='V'
    WHERE typ != 'D' OR typ IS NULL;
```

Geben Sie nun ein ALTER TABLE-Statement an, welches folgenden Constraint hinzufügt: Der `typ` darf nur den Wert `'V'` (Verzeichnis) oder `'D'` (Datei) annehmen.

```
ALTER TABLE eintrag
    ADD CONSTRAINT typ_constraint
    CHECK (typ IN ('V', 'D'));
```

Aufgabe 5: SQL Abfragen (10)

Gegeben ist folgende Tabelle `eintrag`

id	e	name	typ	groesse
1	null	'a'	'V'	10
2	1	'b'	'V'	2
3	1	'a'	'V'	8
4	2	'd'	'D'	2
5	3	'e'	'D'	3
6	3	'f'	'D'	5

Geben Sie die Ergebnisse für die unten folgenden SELECT-Statements an. Falls mehrere Tupel zurückgegeben werden, geben Sie diese in beliebiger Reihenfolge an.

Beachten Sie dazu folgendes Beispiel:

```
SELECT e.groesse FROM eintrag e;
```

```
10 2 8 2 3 5
```

Geben Sie nun die Ergebnisse der folgenden Anfragen an.

```
SELECT COUNT(*) FROM eintrag e
    WHERE groesse > 3
    GROUP BY e.typ;
```

```
1 2
```

```
SELECT COUNT(*) FROM eintrag v, eintrag d
    WHERE v.typ = 'V' AND d.typ = 'D';
```

```
9
```

```
SELECT COUNT(*) FROM eintrag e
    GROUP BY e.name
    HAVING COUNT(*) < 2;
```

```
1 1 1 1
```

```
SELECT COUNT(*) FROM eintrag e1, eintrag e2
    WHERE e1.e = e2.id;
```

```
5
```

```
SELECT COUNT(*) FROM eintrag e1, eintrag e2
    WHERE e1.id = e2.e
    GROUP BY e1.name;
```

```
1 4
```

Aufgabe 6: PL/SQL Trigger (7)

Das Attribut `groesse` eines Eintrags speichert für

- Dateien die Größe dieser Datei und für
- Verzeichnisse die Summe der Größen aller darin enthaltenen Dateien und Verzeichnisse

Falls die Größe eines Eintrags geändert wird, sollte automatisch die Größe des Elternverzeichnisses angepasst werden. Beträgt beispielsweise die alte Größe eines Eintrags 5 und die neue Größe 3, muss von der Größe des Elternverzeichnisses der Wert 2 abgezogen werden.

Erstellen Sie einen Trigger, der immer dann aufgerufen wird, nachdem eine `eintrag`-Zeile geändert wurde.

Dieser Trigger soll für jede geänderte Zeile:

- Das Attribut `groesse` des Elternknotens auf den korrekten Wert ändern (also um die Differenz zwischen altem und neuem Wert des geänderten Knotens).

Um Überprüfungen des Attributs `typ` (also etwa, ob der geänderte Eintrag vom Typ `'D'` ist) brauchen Sie sich nicht zu kümmern. Beachten Sie weiters, dass Sie sich immer nur um das Elternverzeichnis kümmern müssen. Der Trigger wird automatisch für dessen Elternverzeichnis (und dessen Elternverzeichnis...) aufgerufen, falls Änderungen notwendig sind.

```
CREATE FUNCTION updateGroesse()  RETURNS trigger AS $$
BEGIN
    UPDATE eintrag
        SET groesse = groesse - OLD.groesse + NEW.groesse
        WHERE id = OLD.e;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER updateTrigger
    AFTER UPDATE ON eintrag FOR EACH ROW
    EXECUTE PROCEDURE updateGroesse();
```

Aufgabe 7: Java (7)

Vervollständigen Sie die folgende Java Methode, die eine `Collection` von Dateieinträgen `Datei` als Input erhält und diese Einträge in die Datenbank schreibt. Beachten Sie dazu folgende Anmerkungen:

- Die (stark vereinfachte) Klasse `Datei` hat die sichtbaren Felder `id`, `e`, `name`, `groesse`. Dabei ist `name` vom Typ `String` und die restlichen Felder vom Typ `int`.
- Greifen Sie auf eine PostgreSQL Datenbank mit dem Namen `filesdb` auf dem Server `db.example.com` zu. Verwenden Sie den Datenbankuser `filedbuser` mit dem Passwort `mypassword` (Methode: `DriverManager.getConnection`).
- Verwenden Sie für die INSERT-Befehle unbedingt ein **PreparedStatement**, das in einer Schleife jeweils mit den Attribut-Werten der aktuellen Datei gefüllt wird. Das Attribut `typ` soll den Wert `'D'` erhalten. Achten Sie darauf, die korrekte `execute...` für das PreparedStatement aufzurufen.

Deaktivieren Sie auto-commit (`setAutocommit(false)`). Falls während des Einfügens eines Eintrags der Liste ein **Fehler** auftritt, soll die gesamte Transaktion zurückgerollt (`rollback()`), andernfalls durchgeführt (`commit()`) werden. Ansonsten brauchen Sie sich um Fehlerbehandlung oder das Schliessen von Ressourcen nicht zu kümmern.

```
public void dateienHinzufügen(Collection<Datei> dateien) throws Exception {
    Class.forName("org.postgresql.Driver");
    Connection c = DriverManager.getConnection(
        "jdbc:postgresql://db.example.com/filedb", "filedbuser", "mypassword");

    try {
        c.setAutocommit(false);
        p = c.prepareStatement("INSERT INTO eintrag (id, e, name, type, groesse)
            VALUES (?,?,?,'D',?)");
        for(Datei d : dateien) {
            p.setInt(1, d.id);
            p.setInt(2, d.e);
            p.setString(3, d.name);
            p.setInt(4, d.groesse);
            p.executeUpdate();
        }
        conn.commit();
    } catch(Exception e) {
        conn.rollback();
    }
}
```

Gesamtpunkte: 75