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**			14. 03. 2017
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. Um einen geballten Index anzulegen, muss als Index-Methode der B^+-Baum verwendet werden. wahr ◯ falsch ⊗
2. Für drei Relationen $R(AB)$, $S(AC)$ und $T(AC)$ gilt auf jeden Fall folgende Gleichheit:
   $R \ltimes (S \cap T) = (R \ltimes S) \cap (R \ltimes T)$ wahr ◯ falsch ⊗
3. Für drei Relationen $R(AB)$, $S(AC)$ und $T(AC)$ gilt auf jeden Fall folgende Gleichheit:
   $R \ltimes (S \cup T) = (R \ltimes S) \cup (R \ltimes T)$ wahr ⊗ falsch ◯
4. Nehmen Sie an, dass ein verteiltes DBMS eine Tabelle R mit dem Schema $R(\underline{AB}CD)$ enthält und dass das Attribut D nur Werte aus der Menge $\{1, 2, 3, 4\}$ annehmen kann. Diese Tabelle werde mit Hilfe der möglichen Werte des Attributs D in 4 Fragmente zerlegt. Dann ist diese Fragmentierung horizontal, vollständig, disjunkt und rekonstruierbar.
   wahr ⊗ falsch ◯
5. Es soll ein natürlicher Verbund der Relationen $R(\underline{A}BC)$ und $S(AD\underline{E})$ durchgeführt werden, wobei das Attribut A in R und das Attribut E in S ein Schlüssel ist. Dann kann das Ergebnis des Verbunds $R \bowtie S$ unter Umständen weniger Tupeln enthalten als die Relation S. wahr ⊗ falsch ◯
6. Die Anzahl $3 \cdot (b_R + b_S)$ der von einem Hash Join benötigten I/O Operationen reduziert sich auf $2 \cdot (b_R + b_S)$, wenn die Ergebnistupeln des Joins nicht mehr zwischengespeichert werden müssen (z.B.: bei Pipelining oder bei direkter Ausgabe auf dem Bildschirm). wahr ◯ falsch ⊗
7. Bei den Einstellungen ¬steal/force ist beim Wiederanlauf niemals ein Undo erforderlich; ein Redo kann aber sehr wohl erforderlich sein. wahr ◯ falsch ⊗
8. Der ISOLATION_LEVEL 'read committed' hat folgenden Effekt auf mögliche Anomalien: 'unrepeatable reads' werden verhindert; es kannn aber das Phantomproblem auftreten. wahr ◯ falsch ⊗
9. Wenn der Serialisierbarkeitsgraph einer Historie H azyklisch ist, dann kann es unter Umständen zwei oder mehr serielle Historien geben, die zu H äquivalent sind. wahr ⊗ falsch ◯
10. Es bezeichne ST, ACA und RC die Menge aller strikten Historien, Historien ohne kaskadierendes Rücksetzen bzw. rücksetzbaren Historien. Dann gilt: $ST \subseteq ACA \subseteq RC$. 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 Datenbank über Musik-Aufführungen enthalte folgende Relationen:

Werk(wNr, Titel, Kategorie, Komponist) (kurz w),
Orchester(oNr, Name, Stadt) (kurz o) und
Auffuehrung(wNr, oNr, Datum, Ort) (kurz a).

Nehmen Sie an, dass $|w| = 4000$, $|o| = 500$ und $|a| = 100000$. Es ist die Anfrage

```
select w.Titel, o.Name, a.Datum
from Werk w, Orchester o, Auffuehrung a
where o.oNr = a.oNr
and w.wNr = a.wNr
and w.Kategorie = 'Oper'
and o.Stadt = 'Wien';
```

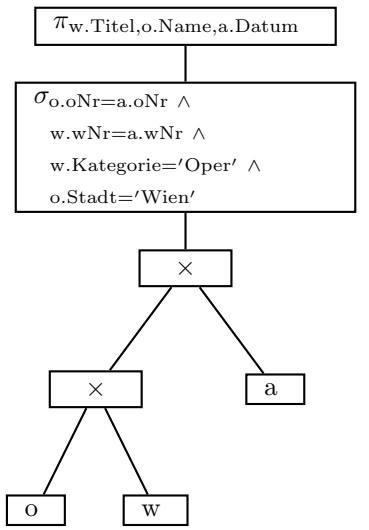
auszuführen (d.h. gesucht sind Informationen über Opern-Aufführungen von Orchestern aus Wien). Es sind die Selektivitäten $Sel_{w/a} = 1/4000 = 0.00025$, $Sel_{o/a} = 1/500 = 0.002$, $Sel_{w.Kategorie='Oper'} = 0.2$ und $Sel_{o.Stadt='Wien'} = 0.1$ anzunehmen.

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

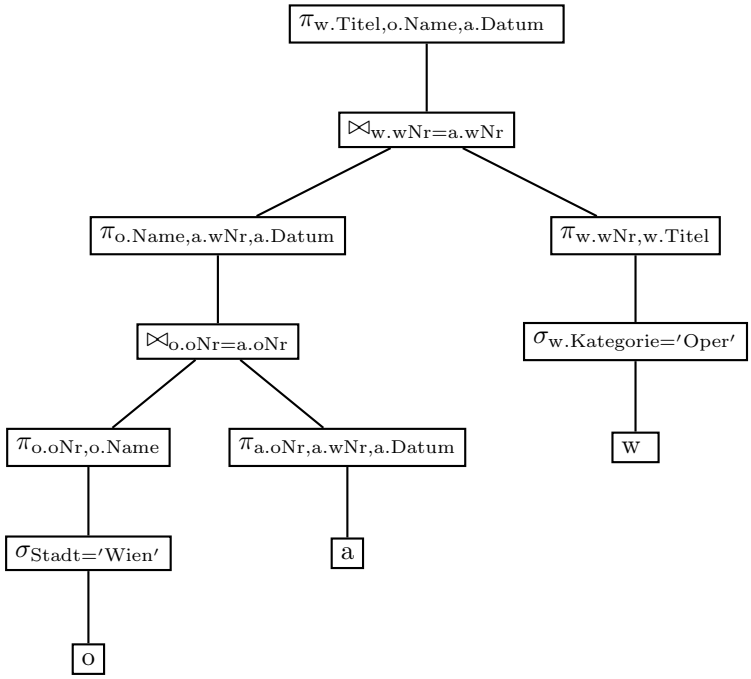
(b) Zeichnen Sie ins zweite Kästchen den Operator-Baum für den optimierten algebraischen Ausdruck. Wenden Sie für die Optimierung folgende Heuristiken an:

- Selektionen so weit wie möglich nach unten verschieben,
- Attribute, die nicht mehr benötigt werden, möglichst früh wegprojizieren,
- Kreuzprodukte wenn möglich durch Joins ersetzen,
- Join-Reihenfolge so, dass der erste Join ein möglichst kleines Zwischenergebnis (d.h.: möglichst wenige Tupel) liefert.

(a) Kanonische Übersetzung:



(b) Optimierter Ausdruck:



Aufgabe 3: (12)

Zeitstempel-basierende Synchronisation. Sie finden in der unten angeführten Tabelle eine Historie von zwei Transaktionen T_1 und T_2. Es gibt zwei Datenobjekte A und B, deren Anfangswerte für readTS und writeTS jeweils 0 sind. Für Lese- und Schreibbefehle auf ein Datum X mit $X \in \{A, B\}$ wird die Notation "SELECT X" bzw. "UPDATE X" verwendet.

Die Zeitstempel für die beiden Transaktionen seien TS(T1) = 5 und TS(T2) = 10. Verwenden Sie die Regeln für Zeitstempel-basierende Synchronisation und geben Sie nun in jeder Zeile in den rechten vier Spalten die aktuellen Werte für readTS(A), writeTS(A), readTS(B) und writeTS(B) nach Ausführung der jeweiligen Anweisung in den ersten zwei Spalten an.

Hinweis: Bei der Zeitstempel-basierende Synchronisation kann es passieren, dass Transaktionen zurückgesetzt werden. Falls in der unten angegebenen Historie das Zurücksetzen einer Transaktion geschieht, müssen Sie alle weiteren Anweisungen dieser Transaktion ignorieren (d.h.: readTS(A), writeTS(A), readTS(B) und writeTS(B) bleiben für alle weiteren Anweisungen dieser Transaktion unverändert).

#	T_1	T_2	readTS(A)	writeTS(A)	readTS(B)	writeTS(B)
1	SELECT A INTO valueA1		5	0	0	0 .....
2	valueA1 = valueA1 + 10		-	-	-	-
3	UPDATE A = valueA1		5	5	0	0 .....
4		SELECT A INTO valueA2	10	5	0	0 .....
5		valueA2 = valueA2 * 2	-	-	-	-
6		UPDATE A = valueA2	10	10	0	0 .....
7		SELECT B INTO valueB1	10	10	10	0 .....
8		valueB = valueB1 + 10	-	-	-	-
9		UPDATE B = valueB1	10	10	10	10 .....
10	SELECT B INTO valueB2		10	10	10	10 .....
11	valueB2 = valueB2 - 10		-	-	-	-
12	UPDATE B = valueB2		10	10	10	10 .....

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 einen Verweis `e` auf das Elternverzeichnis. 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 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: (8)

Evaluieren Sie folgendes SQL-Statement über der angegebenen Tabelle und geben Sie die Ausgabe der gesamten Abfrage an:

eintrag				
id	e	name	typ	groesse
1	NULL	'root'	'V'	19
2	1	'home'	'V'	12
3	1	'usr'	'V'	7
4	2	'admin'	'V'	11
5	4	'config'	'D'	5
6	3	'passwd'	'D'	3
7	2	'README'	'D'	1
8	3	'help.txt'	'D'	4
9	4	'install'	'D'	6

```
WITH RECURSIVE temp(id, e, c) AS (
    SELECT e1.id, e1.e, COUNT(*)
    FROM eintrag e1 JOIN eintrag e2 ON e1.id = e2.e
    GROUP BY e1.id, e1.e
    HAVING e1.typ = 'V'
UNION ALL
    SELECT e1.id, e1.e, t1.c
    FROM eintrag e1 JOIN temp t1 ON e1.id = t1.e
)
SELECT id, SUM(c) as sum
FROM temp
GROUP BY id
ORDER BY sum DESC
```

temp

id	sum
1	8
2	4
3	2
4	2

Aufgabe 6: PL/SQL Trigger (9)

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 eingefügt, geändert oder gelöscht wurde.

Dieser Trigger soll für jede eingefügte, geänderte oder gelöschte Zeile:

- Beim Einfügen und Ändern: Das Attribut `groesse` des neuen Elternknotens erhöhen.
- Beim Ändern und Löschen: Das Attribut `groesse` des alten Elternknotens verringern.

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 OR REPLACE FUNCTION updateGroesse()  RETURNS trigger AS $$
BEGIN
    IF TG_OP = 'DELETE' OR TG_OP = 'UPDATE' THEN
        UPDATE eintrag SET groesse = groesse - OLD.groesse WHERE id = OLD.e;
    END IF;
    IF TG_OP = 'INSERT' OR TG_OP = 'UPDATE' THEN
        UPDATE eintrag SET groesse = groesse + NEW.groesse WHERE id = NEW.e;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER updateTrigger
    AFTER INSERT OR UPDATE OR DELETE 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`.
- Verwenden Sie für die INSERT-Befehle unbedingt ein **PreparedStatement**, das in einer Schleife jeweils mit den Attribut-Werten der aktuellen Datei gefällt wird. Das Attribut `typ` soll den Wert `'D'` erhalten. Achten Sie darauf, die korrekte `execute`... Funktion 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 dateienHinzufuegen(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