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**			23. 06. 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. Für zwei Relationen $R(AB)$ und $S(AC)$ gilt auf jeden Fall die Beziehung $(R \ltimes S) \bowtie (S \ltimes R) \subseteq R \bowtie S$ und unter Umständen auch die Beziehung $(R \ltimes S) \bowtie (S \ltimes R) \neq R \bowtie S$ wahr ◯ falsch ⊗
2. Für zwei Relationen $R(AB)$ und $S(AB)$ gilt auf jeden Fall folgende Gleichheit (d.h.: Vertauschung von Projektion und Durchschnitt): $\pi_A(R \cap S) = \pi_A(R) \cap \pi_A(S)$ wahr ◯ falsch ⊗
3. Nehmen Sie an, dass eine Relation R 20000 Seiten umfasst und dass die Puffergröße 128 beträgt. Dann ist bei einem Hash-Join von R mit S in der Build-Phase auf jeden Fall ein Re-hashing von R erforderlich. wahr ◯ falsch ⊗
4. Nehmen Sie an, dass die zwei Relationen $R(AB)$ und $S(AC)$ bereits nach dem Attribut A sortiert sind. Weiters nehmen Sie an, dass R und S jeweils aus 10.000 Seiten bestehen und dass für die Berechnung des Join $R \bowtie S$ 10 Seiten im Datenbank-Puffer zur Verfügung stehen. Dann gibt es für diesen Join eine Implementierung, die auf jeden Fall mit maximal 20.000 Input Operationen auskommt. wahr ⊗ falsch ◯
5. Betrachten Sie folgende Historie, bei der nur BOT (= Transaktionsbeginn), c_i (= Commit) und Sperranforderungen (aber keine Datenzugriffe oder Informationen über Warten bzw. Wecken einer Transaktion) gegeben sind: BOT_1, BOT_2, $lockX_1(A)$, $lockX_2(B)$, $lockS_1(B)$, c_2, c_1. Diese Historie ist beim Zeitstempel-Verfahren mit wait-die Strategie möglich. wahr ⊗ falsch ◯
6. Ein Ziel des Wiederanlaufs (Recovery) ist es, die Datenbank nach einem Systemabsturz in einen konsistenten Zustand zu bringen. wahr ⊗ falsch ◯
7. Beim Zweiphasen-Commit-Protokoll (2PC-Protokoll) kann der Agent während des Wiederanlaufs aufgrund von den eigenen Log-Einträgen unter Umständen eigentständig (d.h. ohne Kommunikation mit dem Koordinator oder mit anderen Agenten) entscheiden, ob eine Transaktion erfolgreich war oder nicht. wahr ⊗ falsch ◯
8. In einer verteilten Datenbank mit horizontaler Fragmentierung folgt aus der Disjunktheit die Rekonstruierbarkeit der Fragmentierung. wahr ◯ falsch ⊗
9. Die Historie $r_1(A)$, $r_2(A)$, $w_2(B)$, $r_1(B)$, $w_1(A)$, c_1, c_2 ist serialisierbar. wahr ⊗ falsch ◯
10. Die Historie $r_1(A)$, $r_2(A)$, $w_1(B)$, c_1, $r_2(B)$, $w_2(A)$, c_2 ist zwar beim Zwei-Phasen Sperrprotokoll jedoch nicht beim strengen Zwei-Phasen Sperrprotokoll möglich. 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	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	$r(A, a_1)$		
4	$w(B, a_1 - 100)$		[#3, T_1, P_B, B-=110, B+=110, #1] .....
5		$r(A, a_2)$	
6		$r(B, b_2)$	
7		$w(B, a_2 + b_2)$	[#4, T_2, P_B, B+=10, B-=10, #2] .....
8	$w(A, a_1 + 30)$		[#5, T_1, P_A, A+=30, A-=30, #3] .....
9	commit		[#6, T_1, commit, #5] .....
10		$w(C, a_2 - b_2 + 10)$	[#7, T_2, P_C, C+=80, C-=80, #4] .....
11			⟨#8, T_2, P_C, C-=80, #7, #4⟩ .....
12			⟨#9, T_2, P_B, B-=10, #8, #2⟩ .....
13			⟨#10, T_2, (BOT), #9, 0⟩ .....
14			.....

In der obigen Historie bezeichnen A, B und C Daten in der Datenbank, während a_i und b_i lokale Variablen darstellen. Zu Beginn sei der relevante Datenbestand in der Datenbank $A = 10$, $B = 20$, und $C = 30$. Nehmen Sie an, dass unmittelbar nach Zeile 10 ein System-Absturz mit anschließendem Wiederanlauf passiert. Tragen Sie in den Zeilen 4 – 10 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 11ff. 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)

Eine Universitätsdatenbank enthalte folgende Relationen:

Student(sNr, name, alter, wohnort) (kurz s),
Vorlesung(vNr, titel, professor, kategorie) (kurz v) und
Prüfung(sNr, vNr, datum, note) (kurz p). Es ist die Anfrage

```
select s.name, v.professor, p.datum
from Student s, Vorlesung v, Prüfung p
where s.sNr = p.sNr
and v.vNr = p.vNr
and v.kategorie = 'Fort'
and s.alter < 20
and p.note = 1.0;
```

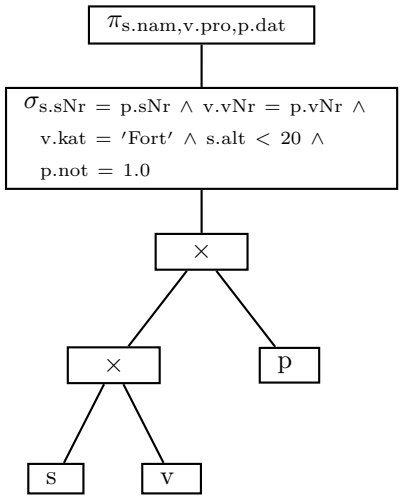
auszuführen (d.h. gesucht sind Informationen über junge StudentInnen, die in einer fortgeschrittenen Vorlesung die Note "Sehr Gut (1.0)" bekommen haben).

(a) Zeichnen Sie ins erste Kästchen den Operator-Baum für die kanonische Übersetzung dieser Anfrage. Verwenden Sie für die Tabellen die Abkürzungen s,v und p. Außerdem dürfen Sie alle Attributnamen auf die ersten 3 Buchstaben abkürzen, z.B.: "nam" statt "name', "alt" statt "alter", etc.

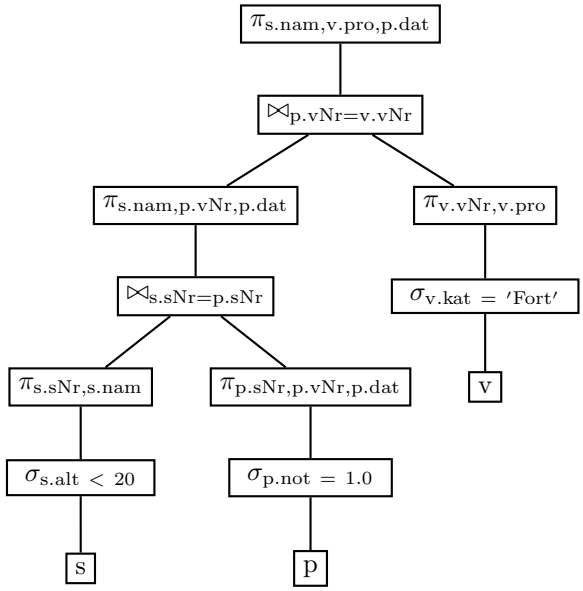
(b) Zeichnen Sie ins zweite Kästchen den Operator-Baum für den optimierten algebraischen Ausdruck. Wenden Sie für die Optimierung folgende Heuristiken an:

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

(a) Kanonische Übersetzung:



(b) Optimierter Ausdruck:



Die folgende Datenbankbeschreibung gilt für die Aufgaben 4 – 7:

Gegeben ist folgendes stark vereinfachtes Datenbankschema, das die Spielergebnisse der laufenden Fußball-EM repräsentiert:

team(<u>code</u>, name)
match(<u>gruppe</u>, <u>team1</u>: *team.code*, <u>team2</u>: *team.code*, tore1, tore2)

Jede Mannschaft (`team`) hat einen eindeutigen Code (`code`, zB `FRA`) und einen vollständigen Namen (`name`, zB `France`). Die Tabelle `match` enthält für jedes Match der Gruppe `gruppe` zwischen den Teams `team1` und `team2` die erzielten Tore des ersten Teams (`tore1`) und des zweiten Teams (`tore2`) als ganzzahlige Werte.

Wählen Sie entsprechende Datentypen für die Attribute.

Aufgabe 4: (6)

Geben Sie nun SQL CREATE Statements an, um die zwei Tabellen zu erstellen.

```
CREATE TABLE team (
        code VARCHAR(3) PRIMARY KEY,
        name VARCHAR(30)
);

CREATE TABLE match (
        gruppe VARCHAR(1),
        team1 VARCHAR(3) REFERENCES team(code),
        team2 VARCHAR(3) REFERENCES team(code),
        tore1 INTEGER,
        tore2 INTEGER,
        PRIMARY KEY (gruppe, team1, team2)
);
```

Das Attribut `gruppe` darf nur die Werte A, B, C, D, E, F für die Gruppenphase, sowie den Wert Z für die Finalspiele enthalten. Erstellen Sie nun hier ein Constraint, das dies sicherstellt.

```
ALTER TABLE match
ADD CONSTRAINT gruppe_constraint
CHECK ("gruppe" IN ('A', 'B', 'C', 'D', 'E', 'F', 'Z' ));
```

Aufgabe 5: SQL Abfragen (8)

Gegeben ist folgender View `sym` und die Tabelle `match`:

```
CREATE OR REPLACE VIEW sym AS (
        SELECT * FROM match
UNION
        SELECT gruppe,
                team2 as team1,
                team1 as team2,
                tore2 as tore1,
                tore1 as tore2
                FROM match);
```

match				
gruppe	team1	team2	tore1	tore 2
'A'	'FRA'	'ROU'	2	1
'A'	'ALB'	'CHF'	0	1
'B'	'WAL'	'SVK'	2	1
'B'	'ENG'	'RUS'	1	1
'A'	'ROU'	'CHF'	1	1
'A'	'FRA'	'ALB'	2	0
'B'	'RUS'	'SVK'	1	2
'B'	'ENG'	'WAL'	2	1
'A'	'CHF'	'FRA'	0	0
'A'	'ROU'	'ALB'	0	1
'B'	'SVK'	'ENG'	0	0
'B'	'RUS'	'WAL'	0	3

Geben Sie die Ergebnisse für die unten folgenden SELECT-Statements bezogen auf `match` und `sym` an.

```
SELECT SUM(tore2) FROM sym WHERE team1='RUS';
```

6

```
SELECT SUM(tore1) FROM sym GROUP BY gruppe ORDER BY gruppe;
```

9 14

```
SELECT COUNT(*) FROM sym WHERE (tore1 - tore2) > 1;
```

2

```
SELECT COUNT(*) FROM sym s1, sym s2 WHERE s1.team1='FRA';
```

72

Aufgabe 6: PL/SQL Funktionen (8)

Erstellen Sie eine PL/pgSQL-Funktion `getPoints`, die als Argumente das Team (`team`), den Gegner (`opponent`) (beides sind Teamcodes, des Typs `team.code`, wie sie auch in der Tabelle `match` vorkommen) und das Kürzel einer Gruppe (`gruppe`) erhält. Durch diese Argumente ist ein eindeutiges Gruppenspiel in der Tabelle `match` bestimmt (sie können der Einfachheit halber davon ausgehen, dass in der Tabelle keine Finalspiele gespeichert sind).

Von Ihrer Funktion soll die Punkteanzahl, die das Team `team` für das Gruppenspiel erhält, zurückgegeben werden. Diese wird wie folgt berechnet:

- Für einen Sieg erhält das Team 3 Punkte. Ein Sieg ist gegeben, wenn das Team mehr Tore erzielt hat als der Gegner.
- Für ein Unentschieden erhält das Team 1 Punkt. Ein Unentschieden ist gegeben, wenn das Team gleich viele Tore erzielt hat als der Gegner.
- Für eine Niederlage erhält das Team 0 Punkte. Eine Niederlage ist gegeben, wenn das Team weniger Tore erzielt hat als der Gegner.

Beachten Sie folgende Hinweise:

- Sie können, müssen aber nicht, den View `sym` verwenden (auf die Tabellen `team` und `match` haben Sie natürlich ebenfalls Zugriff). Die Verwendung des Views empfiehlt sich aber zur Vermeidung von Zusatzaufwand.
- Um Fehlerbehandlung brauchen Sie sich nicht zu kümmern.

```
CREATE OR REPLACE FUNCTION getPoints(team VARCHAR(3), opponent VARCHAR(3),
                                     gruppe VARCHAR(1))
        RETURNS INTEGER AS $$
DECLARE
        tore1 INTEGER;
        tore2 INTEGER;
BEGIN
        SELECT s.tore1, s.tore2 INTO tore1, tore2
                FROM sym s WHERE team=s.team1 AND
                                 opponent=s.team2 AND
                                 gruppe=s.gruppe;

        IF tore1 > tore2 THEN
                RETURN 3;
        ELSEIF tore1 = tore2 THEN
                RETURN 1;
        ELSE
                RETURN 0;
        END IF;
END;
$$ LANGUAGE plpgsql;
```

Aufgabe 7: Java (8)

Vervollständigen Sie die Java Methode `printTable`, die für eine gegebene Gruppe (`gruppe`) die Punktestände der Teams in der Standardausgabe ausgibt.

Sie können, müssen aber nicht, auf folgende Dinge zurückgreifen:

- den in Beispiel 6 definierten View `sym` (auch, wenn Sie dieses Beispiel nicht gelöst haben)
- die in Beispiel 7 definierte Funktion `getPoints`, selbst wenn Sie dieses Beispiel nicht gelöst haben (Sie können davon ausgehen, eine der Spezifikation entsprechende Implementierung zur Verfügung zu haben)

In der Ausgabe soll für jedes Team der angegebenen Gruppe der **Code** des Teams (der vollständige Name ist nicht notwendig), sowie die **Gesamtpunkteanzahl** ausgegeben werden. Dabei ist der Einfachheit halber die Reihenfolge der Teams nicht relevant.

Die Gesamtpunkteanzahl eines Teams ist die Summe der Punkte des Teams über alle in der Tabelle `match` gespeicherten Spiele (Sie können der Einfachheit halber davon ausgehen, dass in der Tabelle `match` keine Finalspiele gespeichert sind). Die Einzelpunkte erhalten Sie am einfachsten durch Aufruf der Funktion `getPoints`.

Beachten Sie dazu folgende Hinweise:

- Um Fehlerbehandlung und die genaue Formatierung der Ausgabe brauchen Sie sich nicht zu kümmern
- Sie können davon ausgehen, dass eine `Connection c` zur Verfügung steht

Vervollständigen Sie nun die Methode `printTable`.

```
public void printTable(String gruppe) throws Exception {
  String q = "SELECT s.team1, SUM(getpoints(s.team1, s.team2))
        FROM sym s WHERE s.gruppe=" + gruppe + " GROUP BY s.team1";

  Statement s = c.createStatement();
  ResultSet rs = s.executeQuery(q);

  while(rs.next()) {
    System.out.println(rs.getString(1) + ":" + rs.getString(2));
  }

  rs.close();
  s.close();
}
```

Gesamtpunkte: 75