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**			12. 01. 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. Für drei Relationen $R(AB)$, $S(AC)$ und $T(BD)$ gilt auf jeden Fall folgende Gleichheit:
   $(R \ltimes S) \cap (R \ltimes T) = \pi_{AB}(R \bowtie S \bowtie T)$ wahr ⊗ falsch ◯
2. Für zwei Relationen $R(AB)$ und $S(AB)$ gilt auf jeden Fall – also insbesondere unabhängig vom konkreten Selektionsprädikat P – folgende Gleichheit (d.h.: Vertauschung von Selektion und Vereinigung):
   $\sigma_P(R \cup S) = \sigma_P(R) \cup \sigma_P(S)$ wahr ⊗ falsch ◯
3. Nehmen Sie an, dass eine Relation R 100.000 Seiten umfasst und dass die Puffergröße 32 beträgt. Es soll ein Hash-Join von R mit einer Relation S durchgeführt werden. Dann hängt es von der Größe von S ab, ob in der Build-Phase ein Re-hashing von R erforderlich ist. wahr ⊗ falsch ◯
4. Es soll ein natürlicher Verbund der Relationen $R(ABC)$ und $S(ADE)$ durchgeführt werden. Dann kann die Realisierung mittels Hybrid Hash Join unter Umständen ein Ergebnis mit weniger Tupeln liefern als die Realisierung mittels Hash Join. wahr ◯ falsch ⊗
5. Die Kostenformel für die Sortierung einer Tabelle T lautet $2b_T(1 + \lceil \log_{m-1}(\lceil b_T/m \rceil) \rceil)$, wobei b_T die Anzahl der Tupel in T und m die Anzahl der verfügbaren Seitenrahmen im Datenbankpuffer bezeichnet. wahr ◯ falsch ⊗
6. Die ACID Eigenschaft von Transaktionen bedeutet: *A*tomicity, *C*onsistency, *I*dentity, und *D*urability. wahr ◯ falsch ⊗
7. Der ISOLATION_LEVEL 'read commited' hat folgenden Effekt auf mögliche Anomalien: 'dirty reads' werden verhindert; es können aber 'unrepeatable reads' auftreten. wahr ⊗ falsch ◯
8. 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, $\text{lockS}_1(\text{A})$, $\text{lockS}_2(\text{B})$, $\text{lockX}_1(\text{B})$, c_2, c_1. Diese Historie ist beim Zeitstempel-Verfahren mit wait-die Strategie möglich und bei wound-wait Strategie unmöglich. wahr ⊗ falsch ◯
9. Die Historie $r_1(A)$, $r_2(A)$, $w_1(B)$, c_1, $w_2(B)$, $w_2(A)$, c_2 ist zwar beim Zwei-Phasen Sperrprotokoll jedoch nicht beim strengen Zwei-Phasen Sperrprotokoll möglich. wahr ◯ falsch ⊗
10. Eine Relation R sei an 5 Netzwerk-Knoten materialisiert mit den Gewichten von jeweils 8. Dann sind $Q_r(R) = 21$ und $Q_w(R) = 20$ gültige Lese- bzw. Schreib-Quoren. 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)

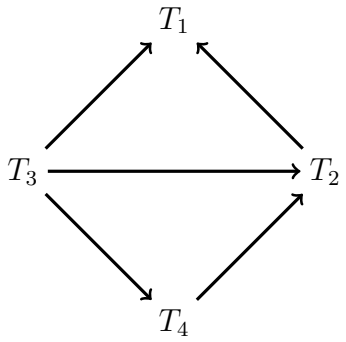
(a) Eine Historie sei durch folgende Folge von Elementaroperationen gegeben: $r_3(D)$, $w_3(C)$, $r_2(C)$, $r_3(A)$, $r_4(B)$, $w_2(D)$, $w_4(A)$, $w_2(B)$, $r_1(B)$, $w_2(A)$, $w_1(C)$, c_1, c_2, c_3, c_4.

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 $T_i \to T_j$ bedeutet, dass in einer äquivalenten seriellen Historie die "Transaktion T_i vor T_j" ausgeführt werden muss (vgl. VO-Folien bzw. Kemper-Buch).

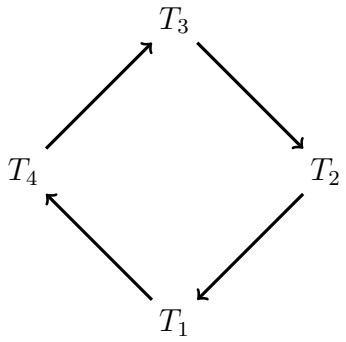
(b) Betrachten Sie die folgende Folge von Sperranforderungen, wobei „lockS$_i(O)$" (bzw. „lockX$_i(O)$") bedeutet, dass die Transaktion T_i eine Lesesperre (bzw. eine Schreibsperre) auf das Datenobjekt O anfordert: lockX$_4(B)$, lockS$_2(A)$, lockS$_3(C)$, lockX$_4(C)$, lockS$_1(D)$, lockX$_3(A)$, lockX$_1(B)$, lockX$_2(D)$.

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 $T_i \to T_j$ bedeutet "Transaktion T_i wartet auf die Freigabe einer Sperre durch T_j" (vgl. VO-Folien bzw. Kemper-Buch).

(a) Serialisierbarkeitsgraph:



(b) Wartegraph:



(c) Ist die Historie aus (a) serialisierbar? ⊗ ja ◯ nein

Wenn ja: in der Reihenfolge T 3 ..... vor T 4 ..... vor T 2 ..... vor T 1 ..... .

Wenn nein: die Historie wird durch das Streichen von zumindest 0 ..... Operationen serialisierbar.

Aufgabe 3: (18)

In einer verteilten Datenbank werden Informationen über SportlerInnen und Schibewerbe verwaltet. In dieser Datenbank sind folgende Tabellen enthalten:

Sportler(SID, name, geburtsdatum, land, jahreseinkommen, anz_titel, saisonen)
Bewerb(BID, sportart, ort, titelverteidiger, preisgeld)

Die Tabelle Sportler wird (vertikal) fragmentiert in folgende 2 Tabellen:
SportlerPersoenlich(SID, name, geburtsdatum, land) und
SportlerSport(SID, jahreseinkommen, anz_titel, saisonen).

Die Tabelle Bewerb wird nach dem Attribut "sportart" (horizontal) in die zwei Tabellen BewerbAlpin und BewerbNordisch fragmentiert, d.h.: sportart = 'alpin' bzw. sportart in ('springen','langlauf','kombination').

Es ist die Anfrage

select s.jahreseinkommen
from Sportler s, Bewerb b
where b.titelverteidiger = s.SID and s.geb > 1996 and b.preisgeld > s.jahreseinkommen/2

auszuführen (d.h. gesucht sind die Jahreseinkommen von jungen SportlerInnen, die einen Bewerb gewonnen haben und dabei mehr als die Hälfte ihres Jahreseinkommens verdient haben).

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

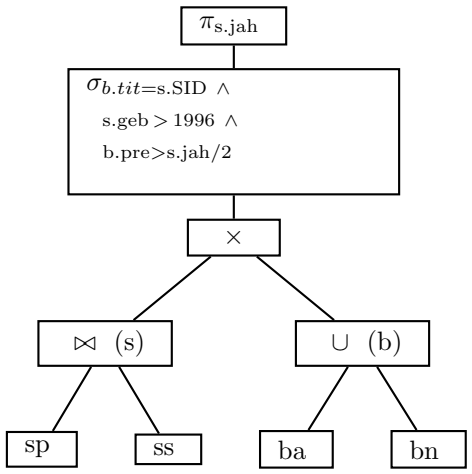
sp (SportlerPersoenlich), **ss** (SportlerSport),
ba (BewerbAlpin) und **bn** (BewerbNordisch).

Außerdem können Sie alle Attributnamen jeweils auf die ersten 3 Buchstaben abkürzen, z.B.: verwenden Sie 'jah' statt 'jahreseinkommen', 'tit' statt 'titelverteidiger', 'pre' statt 'preisgeld', etc.

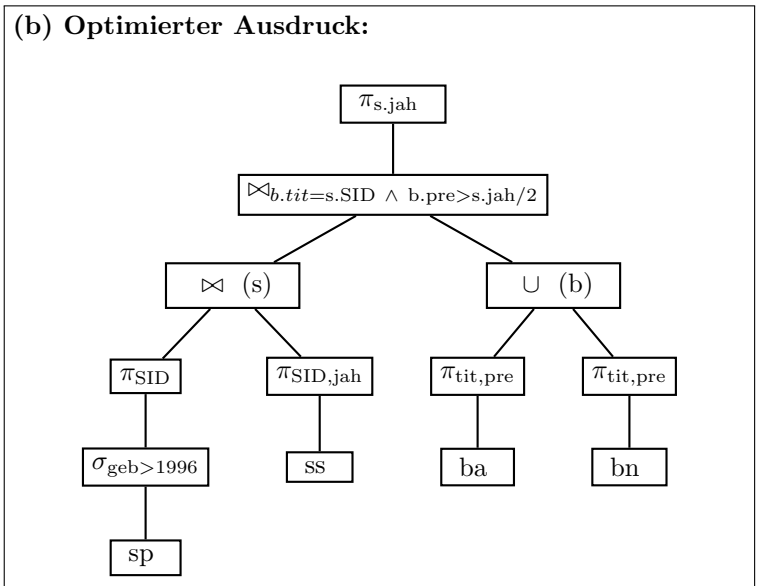
(b) Zeichnen Sie ins zweite Kästchen den Operator-Baum für einen optimierten algebraischen Ausdruck. Wenden Sie für die Optimierung folgende Heuristiken an:

- 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:

Gegeben ist folgendes stark vereinfachtes Datenbankschema, das Flüge eines Kunden und dessen Flugverbindungen repräsentiert:

flight(<u>fid</u>, cno, fno, class, bonus)
connecting_flight(<u>fid</u>: *flight.fid*, <u>cfid</u>: *flight.fid*)

Jeder Flug eines Kunden (`flight`) hat eine eindeutige ID (`fid`), die Kundennummer (`cno`), die Flugnummer (`fno`), die Klasse, für die der Kunde gebucht ist (`class`), und die Bonuspunkte, die der Kunde erhält (`bonus`). Beachten Sie, dass die Flugnummer und die Klasse alphanumerische Werte annehmen können (z.B. "OS 123" bzw. "economy").

Die Tabelle `connecting_flights` enthält für einen Flug (`fid`) die jeweiligen Anschlussflüge (`cfid`).

Aufgabe 4: (8)

Geben Sie nun SQL CREATE Statements an, um die Tabellen `flight` und `connecting_flight` zu erstellen.

```
CREATE TABLE flight (
        fid INTEGER PRIMARY KEY,
        cno INTEGER,
        fno VARCHAR(30),
        class VARCHAR(30),
        bonus INTEGER
);

CREATE TABLE connecting_flight (
        fid INTEGER REFERENCES flight(fid),
        cfid INTEGER REFERENCES flight(fid),
        PRIMARY KEY(fid, cfid)
);
```

Erstellen Sie nun eine Sequence, die für den Primärschlüssel der Tabelle `flight` Werte ab 100 mit Erhöhung um jeweils 10 bis maximal 50000 vergeben soll. Bei Überschreiten dieser Grenze soll wieder bei 100 begonnen werden.

```
CREATE SEQUENCE fid_sequence
        START WITH 100 INCREMENT BY 10
        MAXVALUE 50000 CYCLE;
```

Fügen Sie nun ein Constraint hinzu, das als Werte für `class` nur "economy" oder "business" zulässt.

```
ALTER TABLE flight
        ADD CONSTRAINT class
        CHECK (class IN ('economy', 'business'));
```

Aufgabe 5: (8)

Evaluieren Sie folgendes SQL-Statement mit den angegebenen Tabellen, und geben Sie die Zwischentabelle `temp` und die Ausgabe der gesamten Abfrage an:

flight				
fid	cno	fno	class	bonus
110	2	'OS 1'	'business'	3
120	1	'OS 4'	'economy'	4
130	1	'BA 3'	'economy'	3
140	1	'AA 5'	'economy'	5
150	2	'OS 1'	'business'	2

connecting_flight	
fid	cfid
120	130
130	140

```
WITH RECURSIVE temp(fid, cno, bonus) AS (
    SELECT f.fid, f.cno, f.bonus
    FROM flight f
    WHERE f.fid NOT IN (SELECT cfid FROM connecting_flight)
UNION ALL
    SELECT f.fid, f.cno, f.bonus + t.bonus
    FROM flight f, connecting_flight c, temp t
    WHERE t.fid = c.fid
    AND c.cfid = f.fid
)
SELECT cno, SUM(bonus)
FROM temp
GROUP BY cno;
```

`temp`

fid	cno	bonus
110	2	3
120	1	4
150	2	2
130	1	7
140	1	12

`Ergebnis d. Abfrage`

cno	sum
1	23
2	5

Aufgabe 6: (8)

Nehmen Sie an, dass Funktionen und Trigger wie folgt definiert wurden.

```
CREATE FUNCTION fTrigger1()
RETURNS trigger AS $$
BEGIN
    IF (OLD.bonus <= NEW.bonus) THEN
       RETURN NULL;
    END IF;
    IF (OLD.class = 'business' AND NEW.class = 'business') THEN
       NEW.bonus = NEW.bonus + OLD.bonus;
    END IF;
    IF (NEW.bonus < 6) THEN
       NEW.bonus = OLD.bonus + 20;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tTrigger1
BEFORE UPDATE ON flight
FOR EACH ROW EXECUTE PROCEDURE fTrigger1();
```

Nehmen Sie an, dass folgende Daten in der Tabelle `flight` gespeichert sind:

```
SELECT fid, class, bonus FROM flight ORDER BY fid;
```

fid	class	bonus
110	'economy'	2
120	'business'	5
130	'economy'	3
140	'economy'	7
150	'business'	3

Geben Sie an, wie die Tabelle `flight` nach dem folgenden UPDATE aussieht.

```
UPDATE flight SET bonus = 2, class = 'business';

SELECT fid, class, bonus FROM flight ORDER BY fid;
```

fid	class	bonus
110	'economy	2
120	'business'	7
130	'business'	23
140	'business'	27
150	'business'	23

Aufgabe 7: (6)

Vervollständigen Sie die folgende Java Methode, die für jeden Kunden die Anzahl der Flüge ausgibt. Beachten Sie dazu folgende Anmerkungen:

- Greifen Sie auf eine PostgreSQL Datenbank mit dem Namen `flights` auf dem Server `flynow.test` zu. Verwenden Sie den Datenbankuser `booking` mit dem Passwort `dbpassword` (Methode: `DriverManager.getConnection`).
- Als Ausgabe soll für jeden Kunden eine Zeile mit folgenden beiden Zahlen ausgegeben werden: Die Kundenummer (`cno`), sowie die Anzahl der Flüge (Anzahl der Zeilen in `flight` mit dieser Kundennummer). Eine aufwändige Formatierung der Ausgabe ist nicht nötig!

Um Fehlerbehandlung brauchen Sie sich nicht zu kümmern. Stellen Sie jedoch sicher, dass alle geöffneten Ressourcen auch wieder geschlossen werden!

```
public void printCustomers() throws Exception {
    Class.forName("org.postgresql.Driver");
    Connection c = DriverManager.getConnection(
        "jdbc:postgresql://flynow.test/flights", "booking", "dbpassword");

    Statement s = c.createStatement();
    ResultsSet rs = s.executeQuery(
        "SELECT cno, COUNT(*) FROM flights GROUP BY cno");

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

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

Gesamtpunkte: 75