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**			22. 06. 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. Eine Relation R sei an 5 Netzwerk-Knoten materialisiert mit den Gewichten 5, 9, 7, 3, und 5. Dann sind $Q_r(R) = 10$ und $Q_w(R) = 20$ gültige Lese- bzw. Schreib-Quoren. wahr ⊗ falsch ◯
2. Nehmen Sie an, dass in einem verteilten DBMS das globale Relationenschema $R(\underline{A}BCDE)$ (d.h.: A ist der Primärschlüssel) in die vertikalen Fragmente $R(AB)$, $R(ABCD)$ und $R(ADE)$ unterteilt wurde. Diese Fragmentierung ist vollständig und rekonstruierbar. wahr ⊗ falsch ◯
3. Betrachten Sie zwei Relationen $R(ABC)$ und $S(BDE)$. Dann liefert der Ausdruck $(\pi_B(R) \cup \pi_B(S)) \bowtie R$ ausschließlich Tupel, die auch in R enthalten sind. wahr ⊗ falsch ◯
4. Betrachten Sie zwei Relationen $R(\underline{A}B)$ und $S(AC)$. Um den Ausdruck $R \bowtie S$ mittels Index Nested Loop Join auszuwerten, muss A in der Relation S als Fremdschlüssel definiert sein. wahr ◯ falsch ⊗
5. Ein geballter Index kann nur mit Hilfe eines B^+-Baumes erstellt werden, bei Hash-Indizierung ist ein geballter Index nicht möglich. wahr ◯ falsch ⊗
6. Beim Zweiphasen-Commit-Protokoll können die Agenten mittels Timer-Überwachung – durch geeignete Wahl des Timer-Intervalls – verhindern, dass sie bei einem Absturz des Koordinators blockiert werden. wahr ◯ falsch ⊗
7. Die Historie $w_1(A)$, $r_2(A)$, $r_2(B)$, $w_1(B)$, $w_2(C)$, c_1, c_2 ist serialisierbar und nicht strikt. wahr ◯ falsch ⊗
8. Die Historie $r_1(A)$, $w_1(A)$, $r_2(B)$, c_1, $r_2(A)$, $w_2(A)$, $w_2(B)$, c_2 ist sowohl beim Zwei-Phasen Sperrprotokoll als auch beim strengen Zwei-Phasen Sperrprotokoll möglich. wahr ⊗ falsch ◯
9. Bezeichne ST die Menge der strikten Historien, ACA die Menge der Historien ohne kaskadierendes Rücksetzen und RC die Menge der rücksetzbaren Historien. Dann gilt die Beziehung $ST \subseteq ACA \subseteq RC$. wahr ⊗ falsch ◯
10. 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, $lockS_1(A)$, $lockS_2(B)$, $lockX_1(B)$, c_2, c_1. Nehmen Sie an, dass Sperren erst beim Commit frei gegeben werden. Dann ist diese Historie beim Zeitstempel-Verfahren mit wait-die Strategie 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)

Eine Datenbank von Tennisspielern enthält folgende Relationen:

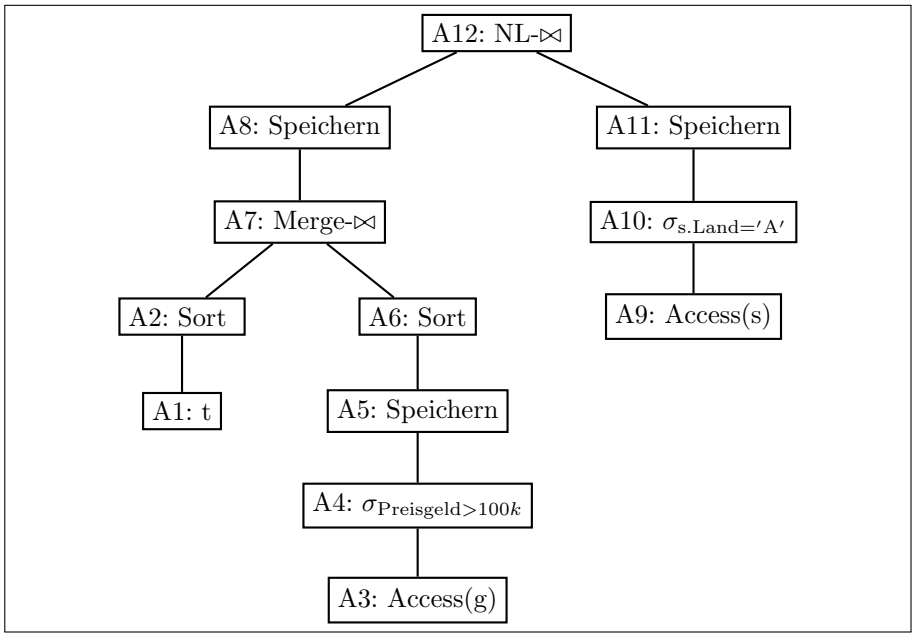
Spieler(SpielerNr, Name, Land, Geburtsdatum, Platzierung) (kurz s),
Turnier(TurnierNr, Name, Jahr, Stadt, Datum) (kurz t) und
gewinnt(SpielerNr, TurnierNr, Platz, Preisgeld) (kurz g).

Nehmen Sie an, dass $|s| = 40000$, $|t| = 50000$, und $|g| = 500000$. Für die durchschnittlichen Tupelgrößen von s, t und g sind die Werte 100, 120 und 40 Bytes anzunehmen. Nehmen Sie weiters an, dass pro Seite 2000 Bytes an Nutzinformation gespeichert werden können und dass die Hauptspeicher-Puffergröße 128 Seiten beträgt. Es ist die Anfrage

```
select *
from Spieler s, Turnier t, gewinnt g
where s.SpielerNr = g.SpielerNr
and t.TurnierNr = g.TurnierNr
and s.Land = 'A'
and g.Preisgeld > 100k;
```

auszuführen (d.h. gesucht sind Informationen über Tennisspieler aus Österreich, die ein hoch-dotiertes Turnier gewonnen haben). Es sind die Selektivitäten $Sel_{s/g} = 1/40000 = 0.000025$, $Sel_{t/g} = 1/50000 = 0.00002$, $Sel_{s.Land='A'} = 0.02$ und $Sel_{g.Preisgeld>100k} = 0.1$ anzunehmen. Für die Primärschlüssel der Relationen s und t sei jeweils ein Hash-Index vorhanden. Nehmen Sie an, dass das Auslesen eines einzelnen Tupels mit einem Hash-Index durchschnittliche Kosten von 1.5 Page I/O erfordert.

Für diese Anfrage ist der Operator-Baum für einen Auswertungsplan gegeben. Bei der Berechnung der benötigten Seiten zum Speichern einer Relation dürfen Sie vereinfachend annehmen, dass die Tupel nicht unbedingt vollständig auf einer Seite Platz haben müssen. Außerdem dürfen Sie annehmen, dass die Tupelgröße beim Join von 2 Relationen gleich der Summe der einzelnen Tupelgrößen ist.



Berechnen Sie für jeden Knoten im Operatorbaum des Auswertungsplans eine Abschätzung für die Anzahl der Tupel im Resultat, die Tupelgröße, die Anzahl der Seiten im Resultat, und die Kosten (Page I/O). Für das Sortieren und für die Joinoperationen ist auch noch die passende Kostenformel anzugeben. Tragen Sie Ihre Berechnungen in folgende Tabelle ein.

Knoten# i	Anzahl Tupel T_i	Tupel-größe g_i	Anzahl Seiten b_i	Kostenformel	Kosten (Page I/O)
A1	50000	120	3000	-	0 .....
A2	50000	120	3000	$2 * b_1 * (1 + I)$ mit ..... $I = \log_{127}(\lceil b_1/128 \rceil) = 1$	12000 .....
A3	500000	40	10000	-	10000 .....
A4	50000	40	1000	-	0 .....
A5	50000	40	1000	-	1000 .....
A6	50000	40	1000	$2 * b_5 * (1 + I)$ mit ..... $I = \log_{127}(\lceil b_5/128 \rceil) = 1$	4000 .....
A7	50000	160	4000	$b_2 + b_6$	4000 .....
A8	50000	160	4000	-	4000 .....
A9	40000	100	2000	-	2000 .....
A10	800	100	40	-	0 .....
A11	800	100	40	-	40 .....
A12	1000	260	130	$b_8 + 1 + \lceil b_8/126 \rceil * (b_{11} - 1)$	5249 .....

Kosten insgesamt (Page I/O):

12000 + 10000 + 1000 + 4000 + 4000 + 4000 + 2000 + 40 + 5249 = 42289 .....

Aufgabe 3: (15)

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-basierenden 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	5	0 .....
8	valueB1 = valueB1 + 10		-	-	-	-
9	UPDATE B = valueB1		10	10	5	5 .....
10		SELECT B INTO valueB2	10	10	10	5 .....
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 Personen und deren Freundschatfsbeziehung beschreibt.

person(pid, name)
friends(a: *person.pid*, b: *person.pid*)

Jede Person `person` hat eine eindeutige Identifikationsnummer `pid` und einen Namen `name`. Die Tabelle `friends` gibt an, dass die Personen mit ID `a` und ID `b` befreundet sind.

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

Aufgabe 4: (6)

Geben Sie nun CREATE-Statements mit allen entsprechenden Constraints für die Tabellen `person` und `friends` an.

```
CREATE TABLE person (
    pid INTEGER PRIMARY KEY,
    name VARCHAR (30)
);

CREATE TABLE friends (
    a INTEGER REFERENCES person(pid),
    b INTEGER REFERENCES person(pid),
        PRIMARY KEY(a,b)
);
```

Geben Sie ein CREATE Statement an, das eine Sequence `pidSequence` erstellt. Diese Sequence soll Werte von 1000 bis 9000 in 10er Schritten vergeben, und bei Erreichen von 9000 wieder bei 1000 fortsetzen.

```
CREATE SEQUENCE pid_sequence
     START WITH 1000
     INCREMENT BY 10
     MAXVALUE 9000
     MINVALUE 1000
     CYCLE;
```

Geben Sie nun ein INSERT-Statement an, das in die person-Tabelle eine Zeile einfügt. Der Wert für `pid` soll dabei mittels der `pidSequence` erzeugt werden.

```
INSERT INTO person VALUES (nextval('pid_sequence'), 'Max');
```

Aufgabe 5: Hierarchische Abfragen (9)

Geben Sie ein SELECT-Statement an, das alle "friends of a friend" der Person mit dem Namen "Max" auswählt. D.h. es sollen die IDs jener Personen ausgegeben werden, die in der `friends`-Tabelle über einen gemeinsamen Freund von Max erreichbar sind.

Hat beispielsweise Max die ID 1, dann soll für folgende `friends`-Tabelle

a	b
1	2
2	3
3	4

ausgegeben werden, dass die Person mit der ID 3 ein "friend of a friend" von Max ist.

Nehmen Sie vereinfachend an, dass die `friends`-Tabelle symmetrisch ist, d.h. wenn a Freund von b ist, so ist auch b Freund von a (ist also wie beim obigen Beispiel das Tupel (1,2) vorhanden, so ist auch das Tupel (2,1) vorhanden usw.).

```
SELECT f2.b
    FROM person p, friends f1, friends f2
    WHERE p.name = "Max"
    AND p.pid = f1.a
    AND f1.b = f2.a
```

Geben Sie nun eine hierarchische Anfrage an, die Max, seine Freunde, Freunde seiner Freunde, ... zurückgibt. D.h. es sollen alle Personen-IDs ausgegeben werden, die durch beliebig viele Schritte in der `friends`-Tabelle von Max aus erreichbar sind.

```
WITH RECURSIVE temp(a) AS (
    SELECT p.pid
    FROM person p
    WHERE p.name = "Max"
UNION ALL
   SELECT f.b
   FROM temp t, friends f
   WHERE t.a = f.a
)
SELECT * FROM temp;
```

Aufgabe 6: PL/SQL Trigger (7)

Erstellen Sie einen Trigger, der immer dann aufgerufen wird, wenn eine `person`-Zeile gelöscht werden soll.

Dieser Trigger soll für jede zu löschende Zeile

- die Anzahl der Freunde der zu löschenden Person bestimmen, d.h. die Anzahl der Zeilen in der friends-Tabelle, die den zu löschenden User betreffen.
- wenn diese Anzahl grösser als 0 ist, eine Exception werfen. In der Meldung der Exception soll die Anzahl der Freunde der zu löschenden Person angegeben werden.

Vergessen Sie nicht, Variablen die Sie benötigen, auch zu deklarieren.

```
CREATE FUNCTION deleteNotification()  RETURNS trigger AS $$
DECLARE
    friendCount INTEGER;
BEGIN
    SELECT count(*) INTO friendCount FROM friends f WHERE OLD.pid=f.a;
    IF friendCount > 0 THEN
        RAISE EXCEPTION 'Zu löschender User hat ' || friendCount || ' Freunde!'
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER deleteTrigger
    BEFORE DELETE ON person FOR EACH ROW
    EXECUTE PROCEDURE deleteNotification();
```

Aufgabe 7: (8)

Erstellen Sie eine Java Methode, die für jede in der Datenbank vorhandene Person die Anzahl der jeweiligen Freunde ausgibt.

- Verwenden Sie für die Bestimmung der Freunde-Anzahl ein `PreparedStatement`.
- Es reicht aus, die IDs der jeweiligen Personen auszugeben.

Um die Behandlung von Exceptions brauchen Sie sich bei dieser Aufgabe nicht zu kümmern.

```
public void printFriends() throws Exception {
    Class.forName("org.postgresql.Driver");
    Connection c = DriverManager.getConnection(
        "jdbc:postgresql://localhost/db", "username", "password");

    PreparedStatement friendPS = c.prepareStatement(
        "SELECT count(*) FROM friends WHERE a=?");

    Statement personS = c.createStatement();
    ResultSet personRS = personS.executeQuery(
        "SELECT pid FROM person");

    while (personRS.next()) {
        int pid = personRS.getInt(1);
        friendPS.setInt(pid, 1);
        ResulSet friendRS = friendPS.executeQuery();
        friendRS.next();
        System.out.println("User " + pid + " hat " +
            friendRS.getInt(1) + " Freunde.");
    }

    friendPS.close();
    personS.close();
    c.close();
}
```

Gesamtpunkte: 75