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**			27. 6. 2013
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. Der Lock-Manager ist zuständig für die Verwaltung der Log-Files. wahr ◯ falsch ⊗
2. Bei den Einstellungen steal/force ist kein Redo nötig, aber ein Undo sehr wohl möglich. wahr ⊗ falsch ◯
3. Eine Relation R sei an 5 Netzwerk-Knoten materialisiert mit den Gewichten von jeweils 20. Dann sind $Q_r(R) = 50$ und $Q_w(R) = 60$ gültige Lese- bzw. Schreib-Quoren. wahr ⊗ falsch ◯
4. Vertikale Fragmentierung gewährleistet immer Rekonstruierbarkeit. wahr ◯ falsch ⊗
5. Betrachten Sie drei Relationen $U(AB)$, $V(BCD)$ und $W(DE)$. Dann gilt auf jeden Fall folgende Gleichheit: $(U \bowtie V) \bowtie W) = (U \times W) \ltimes V)$ wahr ◯ falsch ⊗
6. 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, $\text{lockX}_1(\text{A})$, $\text{lockX}_2(\text{B})$, $\text{lockS}_1(\text{B})$, c_2, c_1. Diese Historie ist beim Zeitstempel-Verfahren mit wait-die Strategie möglich. wahr ⊗ falsch ◯
7. Durch mehrere aufeinanderfolgende Forwards (auch „Chaining“ genannt) kann es zu hohen Zugriffszeiten kommen, da viele Seiten für nur ein Tupel eingelagert werden müssen. wahr ◯ falsch ⊗
8. Die Historie $r_1(A)$, $r_2(A)$, $w_1(A)$, $w_2(B)$, $w_1(B)$, c_1, c_2 ist serialisierbar. wahr ⊗ falsch ◯
9. Um eine Sort-Merge-Join Operation durchzuführen, ist jeweils ein Hash-Index für die Primärschlüssel der beiden beteiligten Relationen erforderlich. wahr ◯ falsch ⊗
10. Will man in PL/pgSQL eine komplette Zeile einer Tabelle „table“ in einer Variable ablegen, so muss diese entweder vom Typ RECORD oder vom Typ table%ROWTYPE sein. 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: Kostenabschätzungen (20)

Betrachten Sie folgenden Ausschnitt der aus der Übung bekannten Datenbank:

Wettkampf(WID, Ort, von, bis) (kurz w),
Wettkampftruppe(TID, Kategorie, Gründung, Sonderzahlung) (kurz t) und
absolviert(WID, TID, Platzierung) (kurz a).

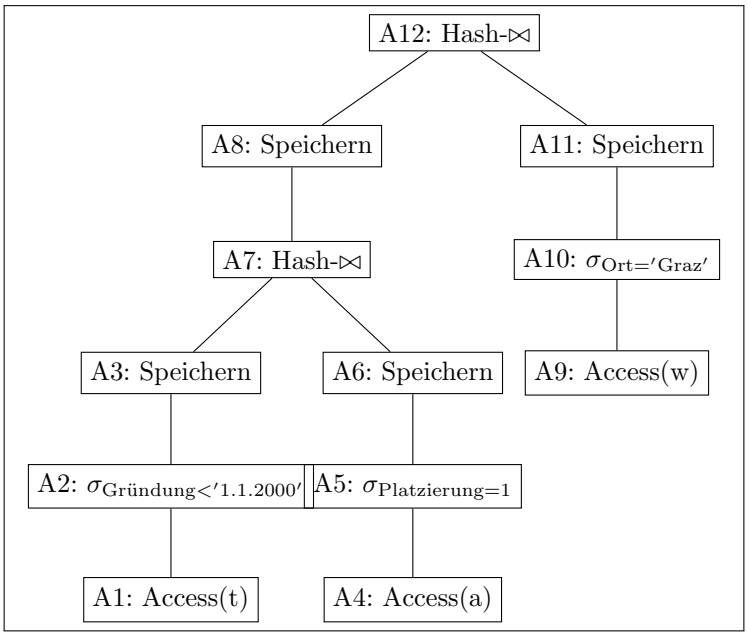
Nehmen Sie an, dass $|t| = 10000$, $|w| = 1000$, und $|a| = 300000$. Für die durchschnittlichen Tupelgrößen von w, t und a sind die Werte 200, 200 und 50 Bytes anzunehmen. Nehmen Sie weiters an, dass pro Seite 1000 Bytes an Nutzinformation gespeichert werden können und dass die Hauptspeicher-Puffergröße 128 Seiten beträgt. Es ist die Anfrage

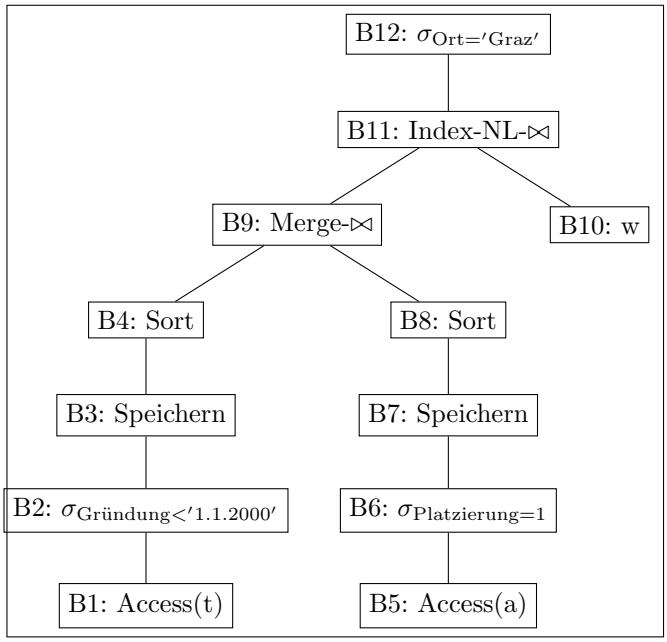
```
SELECT *
FROM Wettkampf w, Wettkampftruppe t, absolviert a
WHERE w.WID = a.WID
AND t.TID = a.TID
AND w.Ort = 'Graz'
AND t.Gründung < '01.01.2000'
AND a.Platzierung = 1;
```

auszuführen (d.h. gesucht sind Informationen über Wettkampftruppen, die vor dem '01.01.2000' gegründet wurden und welche bei einem Wettkampf in 'Graz' den ersten Platz belegt haben).

Es sind folgende Selektivitäten anzunehmen: $Sel_{w/a} = 1/1000 = 0.001$, $Sel_{t/a} = 1/10000 = 0.0001$, $Sel_{w.Ort='\text{Graz}'} = 0.05$, $Sel_{t.Gründung<'01.01.2000'} = 0.4$, und $Sel_{p.Platzierung=1} = 0.05$. Für die Primärschlüssel der Relationen t und w sei jeweils ein Hash-Index vorhanden. Nehmen Sie an, dass das Auslesen eines einzelnen Tupels mit einem Hash-Index durchschnittliche Kosten von 1.4 Page I/O erfordert.

Für diese Anfrage sind die Operator-Bäume für 2 Auswertungspläne gegeben: Plan A im linken Kästchen und Plan B im rechten Kästchen. 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.





(a) Berechnen Sie für jeden Knoten im Operatorbaum des Auswertungsplans A 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 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	10000	200	2000	-	2000 .....
A2	4000	200	800	-	0 .....
A3	4000	200	800	-	800 .....
A4	300000	50	15000	-	15000 .....
A5	15000	50	750	-	0 .....
A6	15000	50	750	-	750 .....
A7	6000	250	1500	$3 * (a_3 + a_6)$	4650 .....
A8	6000	250	1500	-	1500 .....
A9	1000	200	200	-	200 .....
A10	50	200	10	-	0 .....
A11	50	200	10	-	10 .....
A12	300	450	135	$3 * (a_8 + a_{11})$	4530 .....

Kosten insgesamt (Page I/O):

2000 + 800 + 15000 + 750 + 4650 + 1500 + 200 + 10 + 4530 = 29440 .....

(b) Berechnen Sie für jeden Knoten im Operatorbaum des Auswertungsplans B eine Abschätzung für die Anzahl der Tupel im Resultat, die Tupelgröße, die Anzahl der Seiten im Resultat, und die Kosten (Page I/O). 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)
B1	10000	200	2000	-	2000 .....
B2	4000	200	800	-	0 .....
B3	4000	200	800	-	800 .....
B4	4000	200	800	$2 * b_3 * (1 + I)$ mit ..... $I = \lceil \log_{127}(\lceil b_3/128 \rceil) \rceil = 1$	3200 .....
B5	300000	50	15000	-	15000 .....
B6	15000	50	750	-	0 .....
B7	15000	50	750	-	750 .....
B8	15000	50	750	$2 * b_7 * (1 + I)$ mit ..... $I = \lceil \log_{127}(\lceil b_7/128 \rceil) \rceil = 1$	3000 .....
B9	6000	250	1500	$(b_4 + b_8)$	1550 .....
B10	1000	200	200	-	0 .....
B11	6000	450	2700	$b_9 + 1.4 * T_9$	9900 .....
B12	300	450	135	-	0 .....

Kosten insgesamt (Page I/O):

2000 + 800 + 3200 + 15000 + 750 + 3000 + 1550 + 9900 = 36200

Aufgabe 3: Verteilte Datenbanken (10)

Zeigen Sie, dass die *write-all/read-any* Methode zur Synchronisation replizierter Daten einen Spezialfall der *Quorum-Consensus*-Methode darstellt.

(a) Wie müssen Gewichte die Q_w und Q_r vergeben werden, um mit dem *Quorum-Consensus* Verfahren die *write-all/read-any* Methode zu simulieren? [6]

Im einfachsten Fall wird jeder Kopie dasselbe Gewicht zugeordnet, das heißt zum Beispiel ein Gewicht von 1: $w_i = 1$ für $1 \leq i \leq n$. Damit ist das Gesamtgewicht $W(A) = n$.
Damit bei einer Schreiboperation alle Kopien (*write-all*) beschrieben werden, muss nun $Q_w(A) = n$ gesetzt werden. Um beliebiges Lesen (*read-any*) zu erlauben ist folglich $Q_r(A) = 1$ zu setzen.

(b) Zeigen Sie, dass durch die Wahl der Quoren beide Bedingungen des Quorum-Consensus-Verfahrens erfüllt sind. [4]

1. $2 \cdot Q_w(A) > W(A) \Longrightarrow 2n > n$
2. $Q_r(A) + Q_w(A) > W(A) \Longrightarrow 1 + n > n$

Die folgende Datenbankbeschreibung gilt für die Aufgaben 4 – 7:

Gegeben ist folgendes stark vereinfachtes Datenbankschema, in dem die Reservierungen eines Bootsverleihs gespeichert werden.

`Bootstyp`(btid, typ: *Bootstyp.btid*, preis)
`Boot`(bid, typ: *Bootstyp.btid*)
`Reservierung`(von, bis, bid: *Boot.bid*)

In der Tabelle `Bootstyp` werden die verschiedenen Typen von Booten zusammen mit einem Mietpreis (entspricht der Miete pro Tag) gespeichert. Das Attribut `typ` referenziert auf einen möglichen Übertyp (z.B. könnte der Bootstyp "Kajak" den Übertyp "Paddelboot" besitzen).

Die Tabelle `Boot` enthält alle verfügbaren Boote und deren Typ.

In der Tabelle `Reservierung` werden Bootsreservierungen gespeichert. Reservierungen sind nur für ganze Tage möglich. Dabei können Sie der Einfachheit halber (um Datumsfunktionen zu vermeiden) annehmen, dass die Attribute `von` und `bis` ganzzahlige Werte sind, die Tage seit dem Beginn des Jahres 2013 repräsentieren. Beispielsweise kann ein Boot von Tag 204 bis 208 reserviert sein.

Treffen Sie plausible Annahmen bezüglich der Datentypen der Attribute (Bootstypen können Sie der Lesbarkeit halber als Strings annehmen, für `bid` empfiehlt sich ein ganzzahliger Typ).

Aufgabe 4: (6)

Geben Sie CREATE TABLE Statements mit allen nötigen Constraints für die drei Tabellen an. Treffen Sie plausible Annahmen für die Datentypen der Attribute.

```
CREATE TABLE Bootstyp (
   btid VARCHAR(20) PRIMARY KEY,
   typ VARCHAR(20) REFERENCES Bootstyp(btid),
   preis INTEGER
);

CREATE TABLE Boot (
   bid INTEGER PRIMARY KEY,
   typ VARCHAR(20) REFERENCES Bootstyp(btid)
);

CREATE TABLE Reservierung (
   von INTEGER,
   bis INTEGER,
   bid INTEGER REFERENCES Boot(bid),
   PRIMARY KEY (von, bid)
);
```

Legen Sie nun eine SEQUENCE für den Primärschlüssel der Tabelle `Boot` an. Treffen Sie plausible Annahmen für die Eigenschaften dieser Sequence.

```
CREATE SEQUENCE boot_seq;
```

Geben Sie nun ein INSERT Statement für die Tabelle `Boot` an. Verwenden Sie dazu die eben erstellte Sequenz. Vergeben Sie einen plausiblen Wert für das zweite Attribut. Sie können davon ausgehen, dass ein Bootstyp mit dem von Ihnen gewählten Typ existiert.

```
INSERT INTO Boot VALUES (nextval('boot_seq'), 'Kajak');
```

Aufgabe 5: (6)

Evaluieren Sie das folgende SQL-Statement bezüglich der angegebenen Tabelle, und geben Sie die Ausgaben der Abfrage an:

Boot	
bid	typ
1	Boot
2	Boot
3	Muskelb. Boot
4	Paddelboot
5	Paddelboot
6	Kajak
7	Kajak
8	Tretboot
9	Tretboot
10	Tretboot
11	Motorboot

Bootstyp		
btid	typ	preis
Boot	NULL	100
Muskelb. Boot	Boot	10
Paddelboot	Muskelb. Boot	15
Kajak	Paddelboot	25
Tretboot	Muskelb. Boot	20
Motorboot	Boot	40

Betrachten Sie folgendes SQL-Statement:

```
WITH RECURSIVE temp(b,t) AS (
   SELECT bid, typ
   FROM Boot
UNION ALL
   SELECT t.b, bt.typ
   FROM temp t, Bootstyp bt
   WHERE t.t=bt.btid
)
SELECT t, count(b) FROM temp
GROUP BY t;
```

Geben Sie nun das Resultat dieses SQL-Statements an:

t	count(b)
Boot	11
Muskelb. Boot	8
Paddelboot	4
Kajak	2
Tretboot	3
Motorboot	1

Aufgabe 6: (9)

Erstellen Sie einen PL/pgSQL Trigger, der vor dem Löschen eines Bootsdatensatzes prüft, wie viele Reservierungen es für dieses Boot gibt. Falls es mindestens eine Reservierung für dieses Boot gibt, soll das Boot nicht gelöscht werden und eine Meldung ausgegeben werden, wie viele Reservierungen für dieses Boot bestehen.

```
CREATE FUNCTION fCheckRes()
RETURNS trigger AS $$
DECLARE
   res NUMBER;
BEGIN
   SELECT count(r.von) INTO res
   FROM Reservierung r
   WHERE OLD.bid = r.bid
   GROUP BY r.von

   IF (res > 0) THEN
      RAISE NOTICE 'Reservierungen: %', res;
      RETURN NULL;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER checkRes
BEFORE DELETE ON Boot
FOR EACH ROW EXECUTE PROCEDURE fCheckRes();
```

Aufgabe 7: (9)

Vervollständigen Sie die folgende Java Methode `umsatz`, der ein Bootstyp `typ` übergeben wird, und die den Mietpreis pro Tag für ein Boot dieses Typs ausgibt.

Weiters soll für jedes Boot des Typs `typ` dessen `bid` und der zu erwartende Umsatz ausgegeben werden. Der zu erwartende Umsatz ergibt sich aus dem Mietpreis pro Tag des Bootstyps, multipliziert mit der Anzahl der Tage, an denen für dieses Boot Reservierungen bestehen.

Stellen Sie sicher, dass die Methode alle darin geöffneten Ressourcen wieder frei gibt (aber nicht die übergebene `Connection` oder PreparedStatements). Sie brauchen sich nicht um Fehlerbehandlung zu kümmern. Die genaue Formatierung der Ausgabe ist nicht wichtig.

Verwenden Sie zur Lösung ausschliesslich PreparedStatements, die sie hier anlegen (sie können auf eine Connection `c` zugreifen) und in der Methode verwenden können:

```
PreparedStatement psPreis = c.prepareStatement("
   SELECT preis FROM Bootstyp WHERE typ=?");

PreparedStatement psTage = c.prepareStatement("
   SELECT b.bid, sum(r.bis - r.von + 1)
   FROM Reservierung r, Boot b
   WHERE b.typ=? AND b.bid=r.bid
   GROUP BY b.bid");
```

Vervollständigen Sie nun die Methode `umsatz`:

```
public void umsatz(Connection c, String typ) throws Exception {
   psPreis.setInt(1, typ);
   ResultSet rs = psPreis.executeQuery();
   rs.next();
   int preis = rs.getInt(1);
   rs.close();

   psTage.setInt(1, typ);
   ResultSet rs2 = psTage.executeQuery();
   while (rs2.next()) {
      System.out.println(rs2.getString(1) + " " rs2.getInt(2));
   }
   rs2.close();
}
```

Gesamtpunkte: 75