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**			03. 05. 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. Bei einem nicht perfekt balancierten B+ Baum vom Grad k mit n Elementen kann unter Umständen das Einfügen bzw. Löschen von einem Element länger als $O(\log_2 n)$ dauern. wahr ◯ falsch ◯
2. Es gibt Relationen $R(\underline{A}B)$ mit 100 Tupeln und $S(A\underline{C})$ mit 100 Tupeln, für die der Ausdruck $R \bowtie S$ als Ergebnis 5.000 Tupeln liefert. wahr ◯ falsch ◯
3. Für zwei Relationen $R(ABC)$, $S(ABC)$ gelten auf jeden Fall folgende Gleichheiten: $\pi_{AB}(R \cup S) = \pi_{AB}(R) \cup \pi_{AB}(S)$ und $\pi_{AB}(R \cap S) = \pi_{AB}(R) \cap \pi_{AB}(S)$. wahr ◯ falsch ◯
4. Für zwei Relationen $R(ABC)$, $S(ABC)$ und Selektionsprädikat P gelten auf jeden Fall folgende Gleichheiten: $\sigma_P(R \cup S) = \sigma_P(R) \cup \sigma_P(S)$ und $\sigma_P(R \cap S) = \sigma_P(R) \cap \sigma_P(S)$. wahr ◯ falsch ◯
5. Bei der horizontalen Fragmentierung ergeben sich für n Zerlegungsprädikate bis zu 2^n Fragmente, die durch eine Vereinigung wieder zusammengeführt werden können. wahr ◯ falsch ◯
6. Bei den Einstellungen steal/force ist beim Wiederanlauf möglicherweise ein Undo erforderlich; ein Redo ist aber niemals erforderlich. wahr ◯ falsch ◯
7. 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 ◯
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, $\text{lockX}_1(\text{A})$, $\text{lockX}_2(\text{B})$, $\text{lockS}_1(\text{B})$, c_2, c_1. Diese Historie ist beim Zeitstempel-Verfahren mit wait-die Strategie möglich. wahr ◯ falsch ◯
9. Durch das WAL-Prinzip wird sichergestellt, dass bei einem Systemabsturz die Log-Einträge zu allen Änderungsoperationen der zuletzt aktiven Transaktionen auf der Platte stehen. wahr ◯ falsch ◯
10. Der ISOLATION_LEVEL 'serializable' hat folgenden Effekt auf mögliche Anomalien: das Phantomproblem wird verhindert; es können aber 'unrepeatable reads' auftreten. 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 über Wettkämpfe und Teams, die an den Wettkämpfen teilnehmen, enthält folgende Tabellen:

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

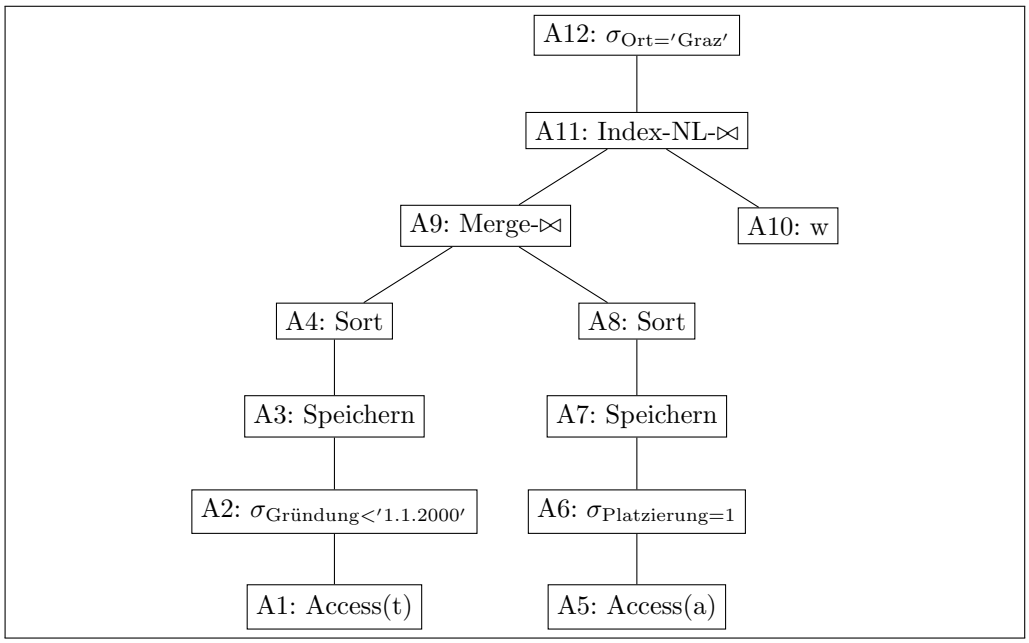
Nehmen Sie an, dass $|t| = 10000$, $|w| = 1000$, und $|a| = 300000$. Für die durchschnittlichen Tupelgrößen von w, t und a sind die Werte 200, 200 und 50 Bytes anzunehmen. Nehmen Sie weiters an, dass pro Seite 1000 Bytes an Nutzinformation gespeichert werden können und dass die Hauptspeicher-Puffergröße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

```
SELECT *
FROM Wettkampf w, Team 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 Teams, die vor dem '01.01.2000' gegründet wurden und die 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 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 a_i	Kostenformel	Kosten (Page I/O)
A1	10000	200	.....	-	.....
A2	.....	.....	.....	-	.....
A3	.....	.....	.....	-	.....
A4	.....	.....	.....	....	.....
A5	300000	50	.....	-	.....
A6	.....	.....	.....	-	.....
A7	.....	.....	.....	-	.....
A8	.....	.....	.....	....	.....
A9	.....	.....	.....	.....	.....
A10	1000	200	.....	-	.....
A11	.....	.....	.....	.....	.....
A12	.....	.....	.....	-	.....

Kosten insgesamt (Page I/O):

.....

Aufgabe 3: (15)
Gegeben ist die folgende Historie von Transaktionen:

Schritt	T_1	T_2	T_3	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			BOT	[#3, T_3, BOT, 0]
4	$r(C, c_1)$			
5		$r(C, c_2)$		
6			$r(A, a_3)$	
7		$w(A, c_2 * 2)$		.....
8	$w(A, c_1 + 200)$			.....
9			$r(B, b_3)$	
10			$w(B, a_3 + b_3)$	.....
11		$w(C, c_2 + 200)$		.....
12		$r(B, b_2)$		
13		$w(B, b_2 + c_2)$		.....
14		commit		.....
15				.....
16				.....
17				.....
18				.....
19				.....
20				.....

In der obigen Historie bezeichnen A, B und C Daten in der Datenbank, während a_i, b_i und c_i lokale Variablen darstellen. Zu Beginn sei der relevante Datenbestand in der Datenbank $A = 100$, $B = 200$ und $C = 100$. Nehmen Sie an, dass unmittelbar nach Zeile 14 ein System-Absturz mit anschließendem Wiederanlauf passiert. Tragen Sie in den Zeilen 7ff. 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 15ff. die Log-Einträge für die Recovery an. Möglicherweise sind nicht alle 20 Zeilen erforderlich. Falls ein Kompensationseintrag Informationen über Undo und/oder Redo enthält, verwenden Sie auch hier die *relative* Schreibweise mittels Addition bzw. Subtraktion.

Die folgende Datenbankbeschreibung gilt für die Aufgaben 4 – 7:

Gegeben ist folgendes stark vereinfachtes Datenbankschema, das ein Installationspaketmanagement für Betriebssysteme repräsentiert:

package(id, name, version, status)
depends_on(id: *package.id*, requires: *package.id*)

Auf der letzten Seite dieser Prüfung finden Sie eine Beispielinstanz dieses Schemas!

Jedes Installationspaket (`package`) hat eine eindeutige ID (`id`), einen Namen (`name`), eine Version (`version`), und den Installationsstatus (`status`) am Rechner. Die Version soll der Einfachheit halber nur ganzzahlige Werte annehmen, der Status ist ein Textstring und kann die Werte "not", "inst", "plan" und "del" annehmen.

Die Tabelle `depends_on` enthält für ein Installationspaket (`id`) die notwendigerweise vorher installierten Pakete (`requires`).

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

Aufgabe 4: (8)

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

Fügen Sie nun ein Constraint hinzu, dass die Kombination der Werte der Spalten `name` und `version` eindeutig sein muss.

Aufgabe 5: Hierarchische Abfragen (8)

Geben Sie ein SQL Statement an, dass zu jedem Paket die Anzahl aller (auch rekursiv) benötigten Pakete ausgibt. Ihr SQL Statement soll, evaluiert über die Datenbankinstanz **packages**, folgende Tabelle ausgeben:

Ausgabe des Statements	
id	cnt
1	0
2	0
3	3
4	1
5	4
6	2

Aufgabe 6: PL/SQL Trigger (8)

Nehmen Sie an, dass folgende Funktionen und Trigger definiert wurden:

```
CREATE FUNCTION fTrigger()
RETURNS trigger AS $$
BEGIN
        IF NEW.version != OLD.version AND
           EXISTS (SELECT * FROM depends_on
                   WHERE requires = OLD.id) THEN
                NEW.version = OLD.version;
        END IF;
        IF OLD.status = 'inst' AND NEW.status = 'not' AND
           NOT EXISTS (SELECT * FROM package p
                       WHERE p.name = OLD.name
                       AND p.version > OLD.version) THEN
                NEW.status = 'del';
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tTrigger
BEFORE UPDATE ON package
FOR EACH ROW EXECUTE PROCEDURE fTrigger();
```

Geben Sie an, wie die Tabelle `package` der Datenbankinstanz **packages** nach dem folgenden UPDATE aussieht:

```
UPDATE package SET status = 'not', version = version + 2;

SELECT id, name, version, status FROM package ORDER BY id;
```

id	name	version	status

Aufgabe 7: Java (6)

Geben Sie die Ausgabe der folgenden Java Methode an. Verwenden Sie dazu die Tabellen der Datenbankinstanz **packages**.

Auf die exakte Formatierung mittels Leerzeichen brauchen Sie nicht zu achten, stellen Sie jedoch zur besseren Lesbarkeit sicher, dass sie für jede Zeile der Ausgabe (`println`) tatsächlich auch eine Zeile verwenden.

```
public void print(Connection c) throws Exception {
    PreparedStatement p = c.prepareStatement(
        "SELECT p.name, p.version FROM package p, depends_on d " +
        "WHERE p.id = d.requires AND d.id = ?" +
        "ORDER BY p.name");

    Statement s = c.createStatement();
    ResultSet r = s.executeQuery(
        "SELECT id, name, version " +
        "FROM package ORDER BY name");

    while(r.next()) {
        p.setInt(1, r.getInt("id"));
        ResultSet r2 = p.executeQuery();

        while(r2.next()) {
            System.out.println(
            r2.getString("name") + " " + r2.getString("version") +
                " > " + r.getString("name") + " " + r.getString("version"));
        }
    }
}
```

Geben Sie nun die Ausgabe an:

Gesamtpunkte: 75

Sie können diese Seite abtrennen und brauchen sie nicht abzugeben!

Datenbankinstanz **packages**:

package			
id	name	version	status
1	'A'	1	'inst'
2	'A'	2	'not'
3	'B'	1	'not'
4	'C'	2	'inst'
5	'B'	3	'inst'
6	'A'	7	'inst'

depends_on	
id	requires
5	3
3	1
3	4
4	2
6	4