

Gruppe A

Bitte tragen Sie **SOFORT** und **LESERLICH** Namen und Matrikelnr. ein, und legen Sie Ihren Studentenausweis bereit.

PRÜFUNG AUS		21.06.2018
<input type="radio"/> DATENMODELLIERUNG 2 (184.790) <input type="radio"/> DATENBANKSYSTEME (184.686)		<b>GRUPPE A</b>
Matrikelnr.	Familiennamen	Vorname

Arbeitszeit: 60 Minuten. Lösen Sie die Aufgaben auf den vorgesehenen Blättern; Lösungen auf Zusatzblättern werden nicht gewertet. **Viel Erfolg!**

**Aufgabe 1:** Eigenschaften von Transaktionen (9)

Gegeben ist die unten dargestellte Transaktion  $T_1$ , welche auf zwei Datensätzen  $A$  und  $B$  arbeitet. Dabei bedeutet  $r_i(X, x_i)$  dass die Transaktion  $T_i$  den Wert des Datensatzes  $X$  in die lokale Variable  $x_i$  liest, und  $w_i(X, x_i)$  dass die Transaktion den Wert  $x_i$  in den Datensatz  $X$  schreibt.

<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;">Schritt</th> <th style="width: 85%;"><math>T_1</math></th> </tr> </thead> <tbody> <tr><td style="text-align: center;">10</td><td style="text-align: center;">BOT</td></tr> <tr><td style="text-align: center;">20</td><td style="text-align: center;"><math>r_1(A, a_1)</math></td></tr> <tr><td style="text-align: center;">30</td><td style="text-align: center;"><math>r_1(B, b_1)</math></td></tr> <tr><td style="text-align: center;">40</td><td style="text-align: center;"><math>w_1(A, a_1 + b_1)</math></td></tr> <tr><td style="text-align: center;">50</td><td style="text-align: center;"><math>r_1(C, c_1)</math></td></tr> <tr><td style="text-align: center;">60</td><td style="text-align: center;"><math>w_1(C, c_1 + 1)</math></td></tr> <tr><td style="text-align: center;">70</td><td style="text-align: center;">commit</td></tr> </tbody> </table>	Schritt	$T_1$	10	BOT	20	$r_1(A, a_1)$	30	$r_1(B, b_1)$	40	$w_1(A, a_1 + b_1)$	50	$r_1(C, c_1)$	60	$w_1(C, c_1 + 1)$	70	commit	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;">Schritt</th> <th style="width: 85%;"><math>T_2</math></th> </tr> </thead> <tbody> <tr><td style="text-align: center;">.....</td><td style="text-align: center;">BOT</td></tr> <tr><td style="text-align: center;">.....</td><td style="text-align: center;">.....</td></tr> <tr><td style="text-align: center;">.....</td><td style="text-align: center;">.....</td></tr> <tr><td style="text-align: center;">.....</td><td style="text-align: center;">.....</td></tr> <tr><td style="text-align: center;">.....</td><td style="text-align: center;">.....</td></tr> <tr><td style="text-align: center;">.....</td><td style="text-align: center;">.....</td></tr> </tbody> </table>	Schritt	$T_2$	.....	BOT	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;">Schritt</th> <th style="width: 85%;"><math>T_3</math></th> </tr> </thead> <tbody> <tr><td style="text-align: center;">.....</td><td style="text-align: center;">BOT</td></tr> <tr><td style="text-align: center;">.....</td><td style="text-align: center;">.....</td></tr> <tr><td style="text-align: center;">.....</td><td style="text-align: center;">.....</td></tr> <tr><td style="text-align: center;">.....</td><td style="text-align: center;">.....</td></tr> <tr><td style="text-align: center;">.....</td><td style="text-align: center;">.....</td></tr> <tr><td style="text-align: center;">.....</td><td style="text-align: center;">.....</td></tr> </tbody> </table>	Schritt	$T_3$	.....	BOT	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
Schritt	$T_1$																																													
10	BOT																																													
20	$r_1(A, a_1)$																																													
30	$r_1(B, b_1)$																																													
40	$w_1(A, a_1 + b_1)$																																													
50	$r_1(C, c_1)$																																													
60	$w_1(C, c_1 + 1)$																																													
70	commit																																													
Schritt	$T_2$																																													
.....	BOT																																													
.....	.....																																													
.....	.....																																													
.....	.....																																													
.....	.....																																													
.....	.....																																													
Schritt	$T_3$																																													
.....	BOT																																													
.....	.....																																													
.....	.....																																													
.....	.....																																													
.....	.....																																													
.....	.....																																													

- a) Geben Sie eine Transaktion  $T_2$  an, so dass diese zu einem *Lost Update* in  $T_1$  führt.
- b) Geben Sie eine Transaktion  $T_3$  an, in welcher auf Grund von  $T_1$  ein *Unrepeatable Read* auftritt.

Für die Aufgaben a) und b) können Sie neben Lese- und Schreiboperationen auch noch **commit** und **abort** verwenden. Achten Sie darauf, dass jede Transaktion entweder mittels **commit** oder **abort** abgeschlossen wird.

Beachten Sie, dass die Aktionen in  $T_1$  in 10er Schritten gezählt werden. Das sollte Ihnen genug Möglichkeiten geben  $T_2$  und  $T_3$  jeweils mit  $T_1$  zu verzahnen.  $T_2$  und  $T_3$  sind unabhängig voneinander. Es müssen *nicht* alle Zeilen in  $T_2$  und  $T_3$  belegt sein!

- c) Betrachten Sie die beiden Transaktionen  $T_1$  und  $T_2$ , auf der nächsten Seite. Ist die Historie bestehend nur aus  $T_1$  und  $T_2$  rücksetzbar? (Ignorieren Sie die Transaktion  $T_3$ .) Begründen Sie Ihre Antwort.

(Achtung: Ankreuzen alleine gibt keine Punkte!)

Historie ist Rücksetzbar: <input type="radio"/> ja <input type="radio"/> nein
Begründung: ..... .....

Historie für Aufgaben 1c) und 2a)			
Schritt	$T_1$	$T_2$	$T_3$
1	BOT		
2		BOT	
3			BOT
4	$r(A, a_1)$		
5			$r(B, b_3)$
6		$w(A, 4)$	
7			$w(B, b_3 + 2)$
8	$w(A, a_1 - 1)$		
9	$w(C, a_1 + 2)$		
10	$r(C, c_1)$		
11		$r(B, b_2)$	
12		$r(A, a_2)$	
13		$w(C, a_2 + b_2)$	
14	$w(C, c_1 - 1)$		
15	commit		

Logeinträge für Aufgabe 2b)
[#1, $T_1$ , BOT, #0]
[#2, $T_2$ , BOT, #0]
[#3, $T_3$ , BOT, #0]
[#4, $T_3$ , $P_B$ , $B+=1$ , $B-=1$ , #3]
[#5, $T_2$ , $P_C$ , $C+=7$ , $C-=7$ , #2]
[#6, $T_3$ , $P_C$ , $C-=6$ , $C+=6$ , #4]
[#7, $T_1$ , $P_A$ , $A+=14$ , $A-=14$ , #1]
[#8, $T_2$ , $P_B$ , $B+=2$ , $B-=2$ , #5]
⟨#9, $T_2$ , $P_B$ , $B-=2$ , #8, #5⟩
⟨#10, $T_2$ , $P_C$ , $C-=7$ , #9, #2⟩
⟨#11, $T_2$ , BOT, #10, #0⟩
[#12, $T_3$ , $P_B$ , $B+=1$ , $B-=1$ , #6]
[#13, $T_1$ , $P_C$ , $C+=14$ , $C-=14$ , #7]
[#14, $T_1$ , COMMIT, #13]
[#15, $T_3$ , $P_A$ , $A-=8$ , $A+=8$ , #12]

Nehmen Sie zur Vereinfachung an, dass die Felder  $A$ ,  $B$  und  $C$  jeweils auf den Seiten  $P_A$ ,  $P_B$  bzw.  $P_C$  liegen. In dieser Aufgabe verwenden wir das Format [LSN, TA, PageID, Redo, Undo, PrevLSN] für "normale" Logeinträge, und das Format ⟨LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN⟩ für Kompensations Logeinträge (Compensation Log Records). Für BOT und COMMIT Log-Einträge kann das Format [LSN, TA, BOT, PrevLSN] bzw. [LSN, TA, COMMIT, PrevLSN] verwendet werden.

Beachten Sie, dass Undo/Redo-Einträge *relativ* zum Datenbestand mittels *Addition* bzw. *Subtraktion* angegeben werden, 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  $k$ .

a) Gegeben ist obige Historie dreier Transaktionen  $T_1$ ,  $T_2$  und  $T_3$ . Nehmen Sie an, dass zu Beginn der Historie der relevante Datenbestand der Datenbank aus folgenden Werten besteht:

$$A = 10, B = 8 \text{ und } C = 12$$

Geben Sie an, welche Log-Einträge beim Abarbeiten der Historie erstellt werden. Verwenden Sie dazu das weiter oben beschriebene Format. Die *begin of transaction* Einträge sind bereits vorgegeben.

[#1,  $T_1$ , BOT, #0], [#2,  $T_2$ , BOT, #0], [#3,  $T_3$ , BOT, #0]


b) Betrachten Sie die auf der vorigen Seite angegebenen Log-Einträge, und nehmen Sie an, dass an Hand dieser Log-Einträge ein Wiederanlauf (Recovery) durchgeführt wird. Geben Sie an, welche Log-Einträge dabei erstellt werden. Verwenden Sie dazu das weiter oben beschriebene Format.

.....	.....
.....	.....
.....	.....
.....	.....

**Aufgabe 3:** Mehrbenutzersynchronisation

(13)

Gehen Sie davon aus, dass in einem DBMS die Isolation Levels wie folgt implementiert sind:

**Read Uncommitted:** Striktes 2PL für Exclusive-Locks. Keine Share-Locks.

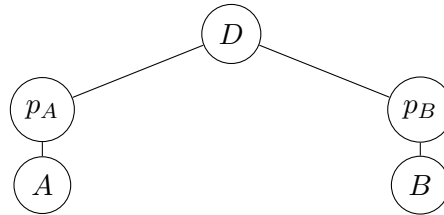
**Read Committed:** Striktes 2PL für Exclusive-Locks, Share-Locks werden sofort nach Erhalt wieder freigegeben.

**Repeatable Read:** Striktes 2PL ohne Einschränkungen.

**Serializable:** Multiple Granularity Locking (Ebenen: Datenbasis, Seite, Datensatz) ohne Einschränkungen.

Gegeben sind zwei Transaktionen  $T_1$  und  $T_2$ , welche auf zwei Datensätzen  $A$  und  $B$  arbeiten (wobei  $A$  und  $B$  auf jeweils getrennten Seiten liegen, die Hierarchie ist unten dargestellt):

$T_1$	$T_2$
$r_1(A, a_1)$	$r_2(B, b_2)$
$w_1(B, 120)$	$w_2(A, 70)$
$r_1(A, b_1)$	$r_2(B, b_2)$



Dabei bedeutet  $r_i(X, x_i)$  dass die Transaktion  $T_i$  den Wert des Datensatzes  $X$  in die lokale Variable  $x_i$  liest, und  $w_i(X, x_i)$  dass die Transaktion den Wert  $x_i$  in den Datensatz  $X$  schreibt.

a) Betrachten Sie folgende verzahnte Ausführung der Transaktionen  $T_1$  und  $T_2$ :

$r_1(A, a_1), r_2(B, b_2), w_2(A, 70), r_2(B, b_2), commit_2, w_1(B, 120), r_1(A, b_1), commit_1$

Geben Sie die Folge von Sperranforderungen und Freigaben an, welche diese Sequenz im Isolation Level *Read Committed* erzeugt. Verwenden Sie  $S_i(A)$  und  $S_i(B)$  falls Transaktion  $T_i$  einen Share Lock auf  $A$  bzw.  $B$  anfordert. Verwenden Sie  $X_i(A)$  und  $X_i(B)$  für die Exclusive Locks, und  $R_i(A)$  und  $R_i(B)$  für die Freigaben.

.....

.....

b) Geben Sie für jedes Isolation Level an, ob es bei der Ausführung von  $T_1$  und  $T_2$  zu einem Deadlock kommen kann (d.h. ob es eine mögliche Ausführungsreihenfolge gibt, in welcher es zu einem Deadlock kommt). Falls ja, geben Sie eine entsprechende Ausführungsreihenfolge an. Falls nein, erweitern Sie  $T_1$  oder  $T_2$  (falls nötig auch beide) um eine Operation, so dass es eine verzahnte Ausführung gibt die zu einem Deadlock führt.

(Achtung: Ankreuzen alleine gibt keine Punkte!)

Isolation Level	Deadlock möglich	zusätzliche Aktion/ Ausführungsreihenfolge
<b>Read Uncommitted:</b>	<input type="radio"/> ja <input type="radio"/> nein	.....
<b>Read Committed:</b>	<input type="radio"/> ja <input type="radio"/> nein	.....
<b>Repeatable Read:</b>	<input type="radio"/> ja <input type="radio"/> nein	.....
<b>Serializable:</b>	<input type="radio"/> ja <input type="radio"/> nein	.....

Für die Aufgaben 4 – 6 gilt die Datenbankbeschreibung am letzten Blatt dieser Prüfung.

**Aufgabe 4:** CREATE Statements

(11)

Die Tabellen `grillplatz` und `person` wurden bereits angelegt.

```
CREATE TABLE grillplatz(  
  gid INTEGER PRIMARY KEY,  
  name VARCHAR(20) NOT NULL,  
  ort VARCHAR(20) NOT NULL  
);
```

```
CREATE TABLE person(  
  pid INTEGER PRIMARY KEY,  
  name VARCHAR(20) NOT NULL,  
  budget NUMERIC(5,2) NOT NULL  
);
```

Geben Sie nun CREATE-Statements mit allen entsprechenden Constraints für die Tabellen `miete` und `grillgruppe` an. Bitte beachten Sie, dass die Spalte `status` in `grillgruppe` nur die Werte 't', 'v' und 'a' annehmen.

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

**Aufgabe 5:** Rekursive Abfragen

(12)

Legen Sie eine VIEW `friends` an, die alle Tupel von zwei Personen ausgibt, die in derselben Grillgruppe sind. Vermeiden Sie die Ausgabe von "trivialen" Freunden (Tupel mit derselben Person in beiden Spalten).

Geben Sie nun eine rekursive Abfrage an, die mit Hilfe der View `friends` alle Bekanntschaften (Freunde, Freunde von Freunden, Freunde von Freunden von Freunden, etc.) ausgibt. Sie können die View auch verwenden, wenn Sie sie nicht angelegt haben!

**Aufgabe 6: PL/SQL Trigger**

(12)

Nehmen Sie an, dass die Funktionen und die Trigger wie am letzten Blatt dieser Prüfung definiert wurde.

Es wird nun folgendes UPDATE-Statement über die Beispielinstantz ausgeführt. Geben Sie die Ausgabe der SELECT-Statements für die Tabelle **person** **vor** und **nach** dem UPDATE an.

```
SELECT pid, budget FROM person WHERE pid IN (7,4);
```

```
UPDATE grillgruppe SET status = 't' WHERE pid IN (7,4);
```

```
SELECT pid, budget FROM person WHERE pid IN (7,4);
```

vor dem UPDATE		nach dem UPDATE	
pid	budget	pid	budget

Über die geänderte Instanz wird nun folgende INSERT-Statements ausgeführt. Geben Sie die Ausgabe der SELECT-Statements für die Tabellen **person** und **miete** **vor** und **nach** den INSERT-Statements an.

```
SELECT budget FROM person WHERE pid = 1; SELECT COUNT(*) FROM miete WHERE mieter = 1;
```

```
INSERT INTO miete VALUES (4,'2018-06-27',90,1); INSERT INTO miete VALUES (4,'2018-06-28',90,1);
```

```
SELECT budget FROM person WHERE pid = 1; SELECT COUNT(*) FROM miete WHERE mieter = 1;
```

Tabelle **person**:

vor den INSERTS	nach den INSERTS
budget	budget

Tabelle **miete**:

vor den INSERTS	nach den INSERTS
COUNT	COUNT

Gesamtpunkte: 70





Sie können diese Seite abtrennen und brauchen ihn nicht abgeben!

Diesen Zettel daher bitte nicht beschriften! (Lösungen auf diesem Zettel werden nicht gewertet!)

Die folgende Datenbankbeschreibung gilt für die Aufgaben 4 – 6:

Gegeben ist folgendes stark vereinfachtes Datenbankschema zum Verwalten von Grillplätzen in Wien.

grillplatz(gid, name, ort)

person(pid, name, budget)

miete(gid: grillplatz.gid, datum, preis, mieter: person.pid)

grillgruppe(gid: miete.gid, datum: miete.datum, pid, person.pid, status)

Jeder Grillplatz hat als Attribute eine eindeutige Identifikationsnummer **gid**, einen Namen **name** und einen **ort**. Jede Person hat als Attribute ebenso eine eindeutige Identifikationsnummer **pid**, einen Namen **name** und ein **budget**.

Ein Grillplatz wird von einer Person gemietet. Diese Miete wird in der Tabelle **miete** gespeichert. Die Miete wird eindeutig durch die Grillplatz **gid** und dem **datum** identifiziert. Jede Miete hat einen anderen Preis **preis** und eine Person als **mieter**.

Der Mieter kann weitere Personen zu einer Grillgruppe einladen. Die Tabelle **grillgruppe** hat als Attribute eine **miete** (**gid** und **datum** aus der **miete** Tabelle), eine Person **pid** und einen **status**. Der Status kann nur die Werte 't' (teilnehmen), 'v' (vielleicht teilnehmen) und 'a' (absagen) annehmen.

Beispielinstanz für Aufgabe 4 – 6:

grillplatz		
gid	name	ort
1	Neue Donau	Wien
2	Alte Donau	Wien
3	Donauinsel	Wien
4	Donaustadtbruecke	Wien

miete			
gid	datum	preis	mieter
1	2018-06-24	160.00	1
2	2018-06-24	120.00	2
3	2018-06-26	90.00	1

person		
pid	name	budget
1	Maria	100.00
2	Max	200.00
3	Elisabeth	300.00
4	Fabian	150.00
5	Anna	200.00
6	Gustav	250.00
7	Lisa	300.00

grillgruppe			
gid	datum	pid	status
1	2018-06-24	1	t
1	2018-06-24	3	t
1	2018-06-24	4	v
1	2018-06-24	5	t
2	2018-06-24	2	t
2	2018-06-24	6	v
2	2018-06-24	3	a
3	2018-06-26	7	v
3	2018-06-26	1	t

## Trigger für Aufgabe 6:

```
CREATE OR REPLACE FUNCTION tfun1() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO grillgruppe VALUES (NEW.gid,NEW.datum,NEW.mieter,'t');
    IF NOT EXISTS (SELECT * FROM grillgruppe
                   WHERE gid=NEW.gid AND
                   datum=NEW.datum AND
                   pid=NEW.mieter) THEN
        DELETE FROM miete WHERE gid = NEW.gid AND datum = NEW.datum;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER t1 AFTER INSERT ON miete FOR EACH ROW EXECUTE PROCEDURE tfun1();

CREATE OR REPLACE FUNCTION tfun2() RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.status = 't' AND (TG_OP='INSERT' OR OLD.status = 'v' OR OLD.status = 'a')) THEN
        IF EXISTS (SELECT * FROM person p, miete m
                   WHERE p.pid = NEW.pid AND
                   m.gid = NEW.gid AND
                   m.datum = NEW.datum AND
                   budget >= preis) THEN
            UPDATE person SET budget = budget - (SELECT preis FROM miete
                                                  WHERE gid = NEW.gid AND datum = NEW.datum)
                WHERE pid = NEW.pid;
        ELSE
            RETURN NULL;
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER t2 BEFORE UPDATE OR INSERT ON grillgruppe FOR EACH ROW EXECUTE PROCEDURE tfun2();
```