Gruppe **A**

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

PRÜFUNG AUS		08.05.2019
◯ DATENMODELLIERUNG 2 (**184.790**)	◯ DATENBANKSYSTEME (**184.686**)	**GRUPPE A**
Matrikelnr.	Familienname	Vorname

Arbeitszeit: 90 Minuten. Lösen Sie die Aufgaben auf den vorgesehenen Blättern; Lösungen auf Zusatzblättern werden nicht gewertet. **Viel Erfolg!**

Aufgabe 1: Mehrbenutzersynchronisation (13)

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

Read Uncommited (RU): Striktes 2PL für Exclusive-Locks. Keine Share-Locks.
Read Committed (RC): Striktes 2PL für Exclusive-Locks, Share-Locks werden direkt vor jedem Lesezugriff angefordert und nach dem Lesen sofort wieder freigegeben (kein 2PL für Share-Locks).
Repeatable Read (RR): Striktes 2PL ohne Einschränkungen
Serializable (S): Nicht implementiert

a) Gegeben ist die unten links dargestellte Historie der drei Transaktionen T_1, T_2 und T_3, sowie das von der jeweiligen Transaktion verwendete Isolation Level (RU, RC, RR). Dabei bedeutet $r_i(X)$ dass die Transaktion T_i den Wert des Datensatzes X liest, und $w_i(X)$ dass die Transaktion T_i auf den Datensatz X schreibt. Die Aktionen b_i und c_i stehen für den Beginn und das Commit von Transaktion T_i.

Geben Sie die Folge der Sperranforderungen und Freigaben an, welche durch diese Historie erzeugt werden.

T_1	T_2	T_3
RR	RU	RC
	b_2	
b_1		
		b_3
		$r_3(C)$
$w_1(B)$		
	$r_2(B)$	
	$r_2(C)$	
	$w_2(C)$	
		$r_3(B)$
		$r_3(C)$
$r_1(D)$		
$w_1(D)$		
c_1		
	c_2	
		c_3

Treffen Sie dabei folgende **Annahme**: *Während eine Transaktion auf eine Sperre wartet (blockiert ist) werden ihre weiteren Aktionen ausgesetzt. Erhält die Transaktion die Sperre, werden die "übersprungenen" Aktionen sofort nachgeholt – bis entweder alle nachgeholt wurden oder die Transaktion erneut blockiert.*
Benützen Sie bitte die folgende **Notation**:
Verwenden Sie $\mathrm{SL}_i(O)$ und $\mathrm{XL}_i(O)$ um anzuzeigen dass Transaktion T_i einen Share- bzw. Exclusive Lock auf Objekt O anfordert. Verwenden Sie $\mathrm{gSL}_i(O)$ und $\mathrm{gXL}_i(O)$ um zu notieren dass T_i einen angeforderten Share- bzw. Exclusive Lock erhalten hat ("granted"), und $wait_i$ um anzuzeigen dass Transaktion T_i angehalten wurde. Für Freigaben verwenden Sie bitte $\mathrm{rSL}_i(O)$ und $\mathrm{rXL}_i(O)$ ("release").

.....

.....

.....

.....

.....

.....

b) Gegeben sind die beiden Transaktionen T_4 und T_5 durch die Reihenfolgen ihrer Elementaroperationen:

T_4: $b_4, r_4(A), r_4(B), w_4(B), w_4(A), c_4$

T_5: $b_5, w_5(A), r_5(B), w_5(B), r_5(A), c_5$

Geben Sie für jedes Isolation Level an, ob es eine mögliche verzahnte Ausführung von T_4 und T_5 gibt, bei welcher es zu einem Deadlock kommt. **Falls ja**, geben Sie eine entsprechende verzahnte Ausführungsreihenfolge an (bis zu dem Zeitpunkt an dem der Deadlock auftritt). **Falls nein**, begründen Sie dies in ein oder zwei kurzen Sätzen. (*Achtung: Ankreuzen alleine gibt keine Punkte!*)

Isolation Level	Deadlock möglich	Ausführungsreihenfolge/Begründung
Read Uncommited:	◯ ja ◯ nein	.....
Read Commited:	◯ ja ◯ nein	.....
Repeatable Read:	◯ ja ◯ nein	.....

Zusätzlicher Platz für Begründungen:

Aufgabe 2: Eigenschaften von Transaktionen (10)

a) Geben Sie für die unten gezeigte Historie den *Serialisierbarkeitsgraphen* an, und bestimmen Sie, ob die Historie *konfliktserialisierbar* ist. **Falls ja**, geben Sie eine konfliktäquivalente, serielle Ausführungsreihenfolge der Transaktionen an. **Falls nein**, geben Sie eine möglichst geringe Anzahl an Transaktionen an welche man entfernen müsste, damit die Historie konfliktserialisierbar wird. Geben Sie für die verbleibenden Transaktionen ebenfalls eine konfliktäquivalente, serielle Ausführungsreihenfolge an.

T_1	b_1	$r(A)$				$r(B)$			$w(A)$										$r(D)$				c_1	
T_2	b_2				$r(C)$			$r(B)$			$w(B)$											c_2		
T_3	b_3			$r(B)$									$r(C)$		$w(D)$									c_3
T_4	b_4																$r(D)$		$w(C)$				c_4	

Serialisierbarkeitsgraph

Historie ist konfliktserialisierbar:

◯ ja ◯ nein

Transaktionen entfernen: .....

Reihenfolge: .....

b) Gegeben ist die unten dargestellte Transaktion T_1, welche auf den Datensätzen A, B und C arbeitet.

Schritt	T_1
10	`BOT`
20	$w_1(C, 80)$
30	$r_1(B, b_1)$
40	$r_1(C, c_1)$
50	$w_1(C, c_1 + 5)$
60	$w_1(A, b_1 + 10)$
70	`commit`

Schritt	T_2
.....	`BOT`
.....	.....
.....	.....
.....	.....
.....	.....

Schritt	T_3
.....	`BOT`
.....	.....
.....	.....
.....	.....
.....	.....

1) Geben Sie eine Transaktion T_2 an, in welcher auf Grund von T_1 möglicherweise ein *Unrepeatable Read* auftritt.

2) Geben Sie eine Transaktion T_3 an, so dass T_1 und T_3 gemeinsam nicht konfliktserialisierbar sind.

Sie können für die Aufgabe neben Lese- und Schreiboperationen ($r_i(O, x_i)$ und $w_i(O, x_i)$: Transaktion T_i liest den Wert von Feld O in die lokalbe Variable x_i, bzw. schreibt den Wert x_i auf das Feld O; falls nicht relevant kann x_i auch weglassen werden) 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 unabhñagig voneinander. Es müssen *nicht* alle Zeilen in T_2 und T_3 belegt sein!

Aufgabe 3: Logging & Recovery (12)

Es ist die auf "Seite A" (vorletzter Zettel der Prüfung) angeführte Historie für vier Transaktionen T_1, T_2, T_3 und T_4 auf den Datensätzen A, B, C und D gegeben (dabei liegt Datensatz X auf Datenbank-Seite P_X). Des weiteren sind einige zugehörige Logeinträge gegeben.

Wir verwenden für die Logeinträge die selbe Notation wie in der Vorlesung: "Normale" Logeinträge haben das Format [LSN, TA, PageID, Redo, Undo, PrevLSN], und Kompensations Logeinträge (Compensation Log Records) haben das Format ⟨LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN⟩. `BOT` Log-Einträge verwenden das Format [LSN, TA, BOT, PrevLSN], und `COMMIT` Einträge das Format [LSN, TA, COMMIT, PrevLSN].

In Aufgabe a) werden Undo/Redo-Einträge relativ *zum Datenbestand mittels* Addition *bzw.* Subtraktion *angegeben, z.B.: [#i, T_j, P_X, X+=d_1, X-=d_2, #k] bedeutet, dass laut i-tem Eintrag 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 um d_2 verkleinert werden müsste und der vorangegangene Logeintrag dieser Transaktion die Nummer k hat.*

In Aufgabe b) werden Undo/Redo-Einträge mittels *physischer Protokollierung* gespeichert.

Zur Erinnerung, bei dieser Methode werden die Redo- und Undo- Informationen in Form von after- bzw. before-images festgehalten. D.h. die Informationen in einem Logeintrag sind nahezu ident zu Aufgabe a), jedoch bedeudet ein Logeintrag [#i, T_j, P_X, X=d_1, X=d_2, #k] dass der Wert des Feldes X nach der Operation d_1 beträgt, und vor der Operation d_2 betragen hat.

Gehen Sie davon aus, dass die Datensätze am Beginn der Historie die folgende Werte enthalten:

$$A = 5, B = 5, C = 5, D = 5.$$

a) Betrachten Sie die ebenfalls auf "Seite A" angegebenen Logeinträge ("Logeinträge für Aufgabe 3a)"), welche die Aktivitäten der Schritte 1–19 protokollieren.

Führen Sie nun zuerst das durch den `ROLLBACK` in Schritt 20 ausgelöste lokale Undo aus, und führen Sie anschließend die restlichen Schritte der Historie aus. Geben Sie die dabei erzeugten Log-Einträge an.

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

Geben Sie die finalen Werte von A, B, C und D an.

A: ..... B: ..... C: ..... D: .....

b) Betrachten Sie nun die zweite Liste an Logeinträgen auf "Seite A" ("Logeinträge für Aufgabe 3b)"), welche ebenfalls die Aktionen der gegebenen Historie protokollieren, diesmal jedoch mittels *physischer Protokollierung*.

Führen Sie an Hand dieser Log-Einträge ein Recovery (nach dem ARIES Verfahren durch), wobei Sie davon ausgehen sollen, dass keine in den Logeinträgen vermerkten Änderungen auf den Seiten P_A, P_B, P_C und P_D gespeichert ist (d.h. gehen Sie davon aus, dass die auf diesen Seiten vermerkte LSN kleiner als #1 ist). D.h. arbeiten Sie mit den anfänglichen Werten $A = 5, B = 5, C = 5, D = 5$.

Geben Sie die dabei entstehenden Log-Einträge an. Verwenden Sie dazu ebenfalls die *physische Protokollierung*.

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

Geben Sie wiederum die finalen Werte von A, B, C und D an.

A: ..... B: ..... C: ..... D: .....

Für die Aufgaben 4 – 6 gilt die Datenbankbeschreibung auf diesem Blatt.

Aufgabe 4: Erstellen eines Datenbankschemas mittels SQL (7)

Folgendes Schema sei gegeben:

unternehmen(<u>name</u>, ceo, hauptprodukt: *rakete.name*)
rakete(<u>name</u>, typ, anzahl, preis)
kunde(<u>name, bereich</u>, ceo)
iteration(<u>alt</u>: *rakete.name*, <u>neu</u>: *rakete.name*)

Ein Raumfahrt Unternehmen ist eindeutig bestimmt durch seinen Namen, daneben wird auch der oder die Geschäftsführer_in und das Hauptprodukt des Unternehmens gespeichert. Eine Rakete hat einen eindeutigen Namen, ein Typ und es wird die Anzahl der produzierten Raketen mit diesem Namen gespeichert, sowie ihr Preis. Der Typ jedes Produktes fällt in eine von drei Kategorien: 'Schwer', 'Mittel' oder 'Leicht'. Ein Produkt kann Vorgänger (alt) oder Nachfolger (neu) eines anderen Produktes sein, dies wird in der Tabelle 'Iteration' festgehalten.

Geben Sie die nötigen SQL Statements an, um obiges Schema (**ohne die Kunden Tabelle**) (inklusive aller Konsistenzbedingungen) anzulegen. Sie können dabei entsprechende (einfache) Datentypen für die Attribute wählen.

Hinweis: Achten Sie bei den Statements auf die Reihenfolge.

Aufgabe 5: Views und Rekursive Abfragen (14)

a) Erstellen Sie eine View, basierend auf dem Schema der letzten Aufgabe.

Es ist gefragt, dass diese View die Namen aller Raketen beinhaltet, welche von zumindest einem Unternehmen als ihr Hauptprodukt geführt wird und welche den Typ 'Schwer' haben.

Geben Sie die nötigen SQL Statements an, um eine View anzulegen, die obige Bedingungen erfülllt.

b) Erstellen Sie eine Rekursive SQL Abfrage

Realisieren Sie, basierend auf dem Schema der letzten Aufgabe, folgende Abfrage:

Die Tabelle 'Iteration' definiert welche Rakete auf welcher anderen basiert. Wir definieren als "Vorgängerin" einer Rakete, alle Raketen von denen diese über einen oder mehr Iterationsschritte abhängt. Als $\mathcal{SH}$ (schweres Hauptprodukt) verstehen wir alle Raketen, die Hauptprodukt eines Unternehmen sind und als Typ 'Schwer' haben. (Demnach ist "$\mathcal{SH}$-Vorgängerin" eine Rakete die sowohl Vorgängerin als auch ein $\mathcal{SH}$ ist.)

Geben Sie nun die transitive Hülle der Vorgängerinnen aller $\mathcal{SH}$ an. D.h. : Alle $\mathcal{SH}$ die $\mathcal{SH}$-Vorgängerin eines $\mathcal{SH}$ sind, alle $\mathcal{SH}$ die $\mathcal{SH}$-Vorgängerin einer $\mathcal{SH}$-Vorgängerin eines $\mathcal{SH}$ sind und so weiter.

Zusätzlich soll für jede $\mathcal{SH}$-Vorgängerin auch die "Distanz" angegeben werden. Die Distanz sei definiert als die Anzahl der Iterationsschritte die eine $\mathcal{SH}$-Vorgängerin von ihrer Nachfolgerin entfernt ist: zum Beispiel, falls A eine direkte $\mathcal{SH}$-Vorgängerin von B, und B eine direkte $\mathcal{SH}$-Vorgängerin von C ist, dann ist die Distanz von A zu C genau 2, und von B zu C genau 1 (falls es keine kürzeren Verbindungen von C zu A gibt)

Falls ein $\mathcal{SH}$ mehrere $\mathcal{SH}$-Vorgängerinnen hat, dann sollen diese $\mathcal{SH}$-Vorgängerinnen auch mehrfach in der Liste aufscheinen, mit (möglicherweise) unterschiedlichen Distanzen.

Ihre Ausgabe soll aus dem Namen aller $\mathcal{SH}$-Vorgängerinnen und ihrer Distanz (definiert wie oben) bestehen.

Geben Sie die nötigen SQL Statements an, um eine Rekursive Abfrage zu schreiben welche die beschriebene Abfrage implementiert.

Sie dürfen die View aus dem vorherigen Beispiel hier wiederverwenden!

Hinweis: Achten Sie darauf, dass ihre Abfrage terminiert.

Aufgabe 6: PL/SQL Trigger (14)

a) Nehmen Sie an, dass die **Funktion `fRU` und der Trigger `trRU`** wie auf "Seite B" (am vorletzten Blatt dieser Prüfung) definiert wurden.

Es wird nun folgendes `UPDATE` Statement **über die Beispielinstanz** ausgeführt. Geben Sie die Ausgabe der SELECT-Statements an. **Falls beim UPDATE ein Fehler auftreten würde, geben Sie an welcher Fehler auftritt.**

```
UPDATE rakete SET anzahl=0, preis=60 WHERE name='Antares';

SELECT * FROM rakete WHERE name='Antares';
SELECT * FROM unternehmen;
```

Es wird nun folgende Statements **über die Beispielinstanz** ausgeführt. Geben Sie die Ausgabe der SELECT-Statements an. **Falls beim UPDATE ein Fehler auftreten würde, geben Sie an welcher Fehler auftritt.**

```
BEGIN;
  UPDATE rakete SET typ='Schwer' WHERE typ='Mittel';
  UPDATE rakete SET anzahl=anzahl-1;
COMMIT;

SELECT * FROM rakete;
SELECT * FROM unternehmen;
```

---

b) Nehmen Sie an, dass *zusätzlich* zu `fRU` und `trRU` nun auch die Funktion `fUCEO` und der Trigger `trUCEO` wie auf "Seite C" (am letzten Blatt dieser Prüfung) definiert wurde.

Es wird nun folgendes `UPDATE` Statement **über die Beispielinstanz** ausgeführt. Beschreiben Sie was passiert. Dokumentieren Sie alle Einträge die sich in der Datenbank ändern.

```
UPDATE kunde SET ceo = 'Elon' WHERE name='Starlink';
```

Seite A

Sie können diesen und den nächsten Zettel abtrennen und brauchen sie nicht abgeben!

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

Angabe für Aufgabe 3

Historie für Aufgabe 3

Schritt	T_1	T_2	T_3	T_4
1		BOT		
2		$r(A, a_2)$		
3				BOT
4				$r(B, b_4)$
5				$r(A, a_4)$
6				$w(A, a_4 + b_4)$
7			BOT	
8			$r(C, c_3)$	
9		$w(A, a_2 + 3)$		
10	BOT			
11	$r(D, d_1)$			
12	$w(D, d_1 + 8)$			
13		$r(D, d_2)$		
14		$w(D, d_2 - 3)$		
15		COMMIT		
16			$w(C, c_3 + 4)$	
17			COMMIT	
18	$r(C, c_1)$			
19	$w(C, d_1 + c_1)$			
20	ROLLBACK			
21				$r(C, c_4)$
22				$w(C, c_4 + 1)$
23				COMMIT

Notation:

- $r_i(O, x_i)$: Transaktion T_i liest den Wert des Datenbankobjekts O in die lokale Variable x_i
- $w_i(O, x_i)$: Transaktion T_i schreibt den Wert x_i in das Datenbankobjekt O
- BOT: Begin of Transaction

Logeinträge Aufgabe 3a)

[#1, T_2, BOT, #0]
[#2, T_4, BOT, #0]
[#3, T_4, P_A, A+=5; A-=5; #2]
[#4, T_3, BOT, #0]
[#5, T_2, P_A, A-=2, A+=2, #1]
[#6, T_1, BOT, #0]
[#7, T_1, P_D, D+=8, D-=8, #6]
[#8, T_2, P_D, D-=3, D+=3, #5]
[#9, T_2, COMMIT, #8]
[#10, T_3, P_C, C+=4, C-=4, #4]
[#11, T_3, COMMIT, #10]
[#12, T_1, P_C, C+=5, C-=5, #7]

Logeinträge Aufgabe 3b)

[#1, T_2, BOT, #0]
[#2, T_4, BOT, #0]
[#3, T_4, P_A, A=10; A=5; #2]
[#4, T_3, BOT, #0]
[#5, T_2, P_A, A=8, A=10, #1]
[#6, T_1, BOT, #0]
[#7, T_1, P_D, D=13, D=5, #6]
[#8, T_2, P_D, D=10, D=13, #5]
[#9, T_2, COMMIT, #8]
[#10, T_3, P_C, C=9, C=5, #4]
[#11, T_3, COMMIT, #10]
[#12, T_1, P_C, C=14, C=9, #7]
⟨#13, T_1, P_C, C=9, #12, #7⟩
⟨#14, T_1, P_D, D=5, #13, #6⟩
⟨#15, T_1, BOT, #14⟩
[#16, T_4, P_C, C=10, C=9, #3]

Beispielinstanz für Aufgabe 6:

Rakete:

name	typ	anzahl	preis
Serenity	Schwer	1	100
Falcon Heavy	Schwer	1	300
Saturn-V	Schwer	5	500
Delta-IV	Mittel	4	200
Antares	Leicht	24	50
LM-5	Mittel	1	150

Unternehmen:

name	ceo	hauptprodukt
X	Elon	Falcon Heavy
CNSA	Zhang	LM-5
ESA	Joe	Serenity
NASA	Jim	Antares

Kunde:

name	bereich	ceo
Starlink	Telecom	Bob
ESA EO	Public	Sepp

Iteration:

vorgaenger	nachfolger
Delta-IV	Saturn-V
Saturn-V	Falcon Heavy

Trigger für Aufgabe 6a.

```
CREATE FUNCTION fRU() RETURNS TRIGGER AS $$
DECLARE
    ersatz VARCHAR(100);
    ru RECORD;
BEGIN
    IF NEW.anzahl <= 0 THEN
        SELECT name INTO ersatz FROM rakete WHERE
          typ = OLD.typ and name != OLD.name
          ORDER BY preis DESC;
        IF ersatz IS NOT NULL THEN
          FOR ru IN
            (SELECT * FROM unternehmen WHERE hauptprodukt = OLD.name)
          LOOP
              UPDATE unternehmen SET hauptprodukt = ersatz WHERE name = ru.name;
          END LOOP;

          DELETE FROM rakete WHERE name=OLD.name;
          RETURN NULL;
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trRU BEFORE UPDATE ON rakete
       FOR EACH ROW EXECUTE PROCEDURE fRU();
```

Zusätzlicher Trigger für Aufgabe 6b.

```
CREATE FUNCTION fUCEO() RETURNS TRIGGER AS $$
DECLARE
  x RECORD;
BEGIN
  SELECT r.* INTO x FROM rakete r
    JOIN unternehmen u ON r.name = u.hauptprodukt
    WHERE u.ceo = NEW.ceo;

  UPDATE rakete SET anzahl = anzahl-5 WHERE name=x.name;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trUCEO BEFORE UPDATE OF ceo ON kunde
       FOR EACH ROW EXECUTE PROCEDURE fUCEO();
```