Gruppe **A**

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

PRÜFUNG AUS	**MUSTERLÖSUNG**	10.03.2020
◯ 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!**

Achtung! Für sämtliche Fragen mit Ankreuzmöglichkeiten gilt: Ankreuzen alleine gibt keine Punkte; Punkte nur in Zusammenhang mit geforderter Erklärung/Beispiel/...

Aufgabe 1: Sperrprotokolle (Locking) (14)

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

1. **Read Uncommited (RU):** MGL mit 2PL für Exclusive-Locks, Lesezugriffe ohne Sperre.
2. **Read Committed (RC):** MGL mit 2PL für Exclusive-Locks, MGL ohne 2PL für Share-Locks (Lesesperren werden vor jeder Leseoperation angefragt, und sofort nach der Leseoperation wieder freigegeben).
3. **Repeatable Read (RR):** MGL mit 2PL (ohne Einschränkungen/Sonderregeln).
4. **Serializable (Ser):** MGL mit 2PL und nur einer einzigen Ebene: Datenbasis.

Das Multiple Granularity Locking bei RU, RC und RR arbeitet dabei mit den Ebenen Datenbasis, Bereich (Area), Seite (Page) und Datensatz.

Die Datenbasis DB sei dabei wie auf **"Seite Y"** dargestellt in die beiden Bereiche α und β, mit den Seiten P_A bzw. P_C und P_E mit den zugehörigen Datensätzen unterteilt.

a) Angenommen alle Transaktionen auf dem DBMS laufen im Isolation Level **Serializable**. Ist in diesem Fall ein Deadlock möglich?
Falls ja geben Sie eine Historie an, welche zu einem Deadlock führt. (Notation wie in Aufgabe b) beschrieben.)
Falls nein, geben Sie dafür eine kurze (1-2 Sätze) Begründung an.

Deadlock bei nur Serializable möglich: ⊗ ja ◯ nein

$b_1, b_2, r_1(A), r_2(B), w_1(B), w_2(A)$ .....

`Versuch die Schreibsperren zu erhalten ⇒ Deadlock` .....

b) Betrachten Sie nun die unten angegebene Historie der vier Transaktionen T_1, T_2, T_3 und T_4, wobei zu jeder Transaktion auch das **Isolation Level** gegeben ist in welchem sie läuft.

	T_1 **(RR)**	T_2 **(RC)**	T_3 **(Ser)**	T_4 **(RR)**
1	b_1	b_2	b_3	b_4
2		$r_2(B)$		
3	$r_1(F)$			
4				$r_4(B)$
5		$w_2(A)$		
6			$w_3(F)$	
7			$r_3(C)$	
8	$w_1(E)$			
9		$w_2(P_C)$		
10			$w_3(G)$	
11		$w_2(G)$		
12				$r_4(P_E)$
13		$r_2(E)$		
14				$w_4(\alpha)$
15		$w_2(E)$		
16			$r_3(D)$	
17			$w_3(E)$	
18				$r_4(A)$
19	$w_1(B)$			
20				c_4
⋮	⋮	⋮	⋮	⋮

Notation:

- $r_i(O)$ und $w_i(O)$: Lese- bzw. Schreibzugriff von T_i auf Objekt O.
- b_i, c_i: Beginn bzw. Commit von T_i.

i) Nach welcher Operation darf T_4 *frühestens* die Sperre auf P_E freigeben? (Aus den Operationen von T_4.)

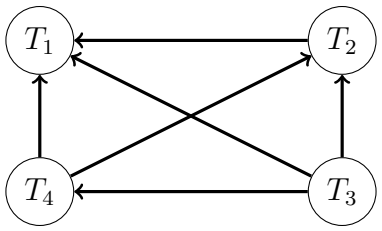
Nach Operation in Zeile `14` .....

ii) Angenommen nach Schritt 20 würde T_2 die Sperre auf G freigeben. Auf welche der Felder A–G kann T_2 im weiteren Verlauf der Transaktion prinzipiell noch lesend bzw. schreibend zugreifen? (Ignorieren Sie mögliche Sperren anderer Transaktionen.)

Lesezugriffe: `A, B, C, D, E, F, G` .....

Schreibzugriffe: `A, C, D, E` .....

iii) Betrachten Sie die Historie und entscheiden Sie, welche Sperrren gewährt werden und welche nicht. Geben Sie den Wartegraphen unmittelbar nach Schritt 14 an.



iv) Geben Sie für folgende Objekte alle unmittelbar nach Schritt 14 gehaltenen Sperren an.
Verwenden Sie S, X, IS und IX falls eine Transaktion eine entsprechende Sperre hält, und WS, WX, WIS und WIX falls sie auf die entsprechende Sperre warten muss.

	T_1	T_2	T_3	T_4
E:	`X`	`WS`		
P_E:	`IX`	`IX`		`WS`

	T_1	T_2	T_3	T_4
α:		`IX`		`IS`
DB:	`IX`	`IX`	`WX`	`IS`

Hinweis zur Musterlösung:

- i) Die Schreibsperre auf α ist die letzte Sperre, welche T_4 anfordern muss; selbst nach Freigabe von G ist lesen auf A mit den restlichen Sperren noch möglich.
- iii) und iv) Nach Schritt 14 sind folgende Transaktionen blockiert:
  - T_3 blockiert in Schritt 6 da sie laut Protokoll eine X-Sperre auf die gesamte DB benötigt, was nicht geht da bereits andere Sperren bestehen.
  - T_4 blockiert in Schritt 12, da eine S-Sperre auf P_E nicht kompatibel mit der IX-Sperre von T_1 ist, welche diese aufgrund von $w_1(E)$ (Schritt 8) hält.
  - T_2 blockiert in Schritt 13 wegen der X-Sperre von T_1 auf E, welche diese seit Schritt 8 hält.

Aufgabe 2: Eigenschaften von Transaktionen (12)

Betrachten Sie das Szenario aus Aufgabe 1 (hypothetisches DBMS und Datenbankstruktur; **nicht** die Historie), und nehmen Sie an, dass Transaktionen ausschließlich im Isolation Level **Read Committed** ablaufen.

Beantworten Sie die folgenden Fragen mit "Ja" oder "Nein". *Bei "Ja"* begründen Sie Ihre Antwort kurz (1–2 Sätze). *Bei "Nein"* geben Sie eine Historie an (Operationen: b_i, c_i, a_i, $r_i(O)$, $w_i(O)$), welche mit der **"Implementierung"** des Isolation Levels konform ist, aber die gefragte Eigenschaft verletzt.
Hinweis: Sie brauchen Sperranforderungen und Freigaben nicht angeben - achten Sie nur darauf, dass es für Ihre Historie eine entsprechende gültige Folge von Sperren und Freigaben gibt.

a) Angenommen es finden nur Schreibzugriffe statt. Erzeugt das DBMS immer *konfliktserialisierbare* Historien?

Nur Schreibzugriffe → Konfliktserialisierbar: ⊗ ja ○ nein

2PL garantiert die Konfliktserialisierbarkeit. Bei ausschließlich Schreibzugriffen wird die Implementierung zu einem reinen 2PL.

b) Angenommen es finden Schreib- und Lesezugriffe statt. Erzeugt das DBMS immer *konfliktserialisierbare* Historien?

Schreib- und Lesezugriffe → Konfliktserialisierbar: ○ ja ⊗ nein

Mögliche Lösung: $b_1, b_2, r_2(A), w_1(A), r_2(A), c_2, c_1$

Mit passenden Sperren und Freigaben (''normales'' 2PL, nicht strikt!): $b_1, b_2,$ .....

$lockS_2(A), r_2(A), relS_2(A), lockX_1(A), w_1(A), relX_1(A), lockS_2(A), r_2(A), relS_2(A), c_2, c_1$ .....

c) Angenommen es finden nur Schreibzugriffe statt. Erzeugt das DBMS immer *strikte* Historien?

Nur Schreibzugriffe → Strikt: ○ ja ⊗ nein

Mögliche Lösung: $b_1, b_2, w_1(A), w_2(A), c_1$

Mit passenden Sperren und Freigaben (''normales'' 2PL, nicht strikt!): .....

$b_1, b_2, lockX_1(A), w_1(A), relX_1(A), lockX_2(A), w_2(A), c_1$

d) Angenommen es finden Schreib- und Lesezugriffe statt. Erzeugt das DBMS immer *rücksetzbare* Historien?

Schreib- und Lesezugriffe → Rücksetzbar: ○ ja ⊗ nein

Mögliche Lösung: $b_1, b_2, w_1(A), r_2(A), c_2$

Mit passenden Sperren und Freigaben (''normales'' 2PL, nicht strikt!): .....

$b_1, b_2, lockX_1(A), w_1(A), relX_1(A), lockS_2(A), r_2(A), relS_2(A), c_2$

Anmerkung Musterlösung: Die Angabe der Sperren und Freigaben nur für Verständnis, kein Teil der Lösung!

Aufgabe 3: Protokollierung und Wiederanlauf (Logging und Recovery) (9)

Führen Sie anhand der unten angegebenen Logeinträge sowie dem dargestellten Inhalt der Seiten einen Wiederanlauf (Recovery) nach dem ARIES Verfahren durch. Das Format der Logeinträge entspricht der in Vorlesung und Übung verwendeten Notation, welche auf auf **"Seite Y"** (vorletzte Seite der Prüfung) wiederholt wird.

a) Geben Sie alle im Rahmen des Wiederanlaufs erstellten *Logeinträge* an.

Logeinträge (Angabe)

[#1, T_2, BOT, #0]
[#2, T_3, BOT, #0]
[#3, T_3, P_C, C+=15, C-=15, #2]
[#4, T_1, BOT, #0]
[#5, T_2, P_A, B+=1, B-=1, #1]
[#6, T_4, BOT, #0]
[#7, T_2, P_E, E+=15, E-=15, #5]
[#8, T_4, P_A, B+=5, B-=5, #6]
[#9, T_1, P_E, E-=16, E+=16, #4]
[#10, T_3, P_A, A+=20, A-=20, #3]
[#11, T_3, P_A, A+=1, A-=1, #10]
⟨#12, T_2, P_E, E-=15, #7, #5⟩
[#13, T_4, P_C, C-=2, C+=2, #8]
[#14, T_4, P_A, A-=5, A+=5, #13]
⟨#15, T_2, P_A, B-=1, #12, #1⟩
[#16, T_1, P_E, E-=10, E+=10, #9]
⟨#17, T_4, P_A, A+=5, #14, #13⟩
⟨#18, T_3, P_A, A-=1, #11, #10⟩
⟨#19, T_2, BOT, #15⟩

Logeinträge (Lösung)

⟨#20, T_1, P_E, E+=10, #16, #9⟩ .....

⟨#21, T_4, P_C, C+=2, #17, #8⟩ .....

⟨#22, T_3, P_A, A-=20, #18, #3⟩ .....

⟨#23, T_1, P_E, E+=16, #20, #4⟩ .....

⟨#24, T_4, P_A, B-=5, #21, #6⟩ .....

⟨#25, T_4, BOT, #24⟩ .....

⟨#26, T_1, BOT, #23⟩ .....

⟨#27, T_3, P_C, C-=15, #22, #2⟩ .....

⟨#28, T_3, BOT, #27⟩ .....

.....

Seiten (DB Inhalt)

P_A	LSN: #10
$A = 12$ $B = 15$	

P_C	LSN: #3
$C = 20$ $D = 25$	

P_E	LSN: #7
$E = 10$ $F = 15$ $G = 11$	

b) Geben Sie die Werte der Felder A, B, C und E *nach* der *Redo*-Phase an:

A: 12 ..... B: 14 ..... C: 18 ..... E: -31 .....

c) Geben Sie die Werte der Felder A, B, C und E *nach* dem erfolgreichen Wiederanlauf an:

A: -8 ..... B: 9 ..... C: 5 ..... E: -5 .....

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

Aufgabe 4: Erstellen eines Datenbankschemas mittels SQL (7)

Folgendes Schema sei gegeben:

importeur(id, land)
produzent(marke, ursprung, name, familie: *(frucht.name, frucht.familie)*)
frucht(name, familie, hauptProduzent: *(produzent.marke)*)
kauft(id: *importeur.id*, produzent: *produzent.marke*, name, familie: *(frucht.name, frucht.familie)*)

Jeder Importeur ist eindeutig durch seine ID gekennzeichnet, daneben wir auch das Land dieses Importeurs vermerkt. Realisieren Sie die fortlaufende Nummerierung der ID mit Hilfe einer Sequence. Die Sequence soll bei 100 beginnen und in Zehnerschritten erhöht werden. Bei Produzenten wird Marke und Ursprungsland vermerkt, wobei jeder Produzent durch die Marke eindeutig gekennzeichnet sein muss. Darüber hinaus wird die am meisten exportierte Frucht dieses Produzenten auch explizit gespeichert. Jede Frucht ist eindeutig durch das Paar von Namen und Familie gekennzeichnet, zusätzlich wird der Hauptproduzent dieser Frucht vermerkt. Die kauft Relation drückt aus, welcher Importeur, von welchem Produzenten welche Frucht gekauft hat.

Geben Sie die nötigen SQL Statements an, um obiges Schema (inklusive aller Konsistenzbedingungen) anzulegen. Sie können dabei entsprechende (einfache) Datentypen für die Attribute wählen.
Die Abkürzung VC statt VARCHAR(100) ist erlaubt.

```
CREATE SEQUENCE seq_id INCREMENT BY 10 MINVALUE 100;
CREATE TABLE importeur (
   id   INTEGER DEFAULT nextval('seq_id') PRIMARY KEY,
   land VARCHAR(100)
);

CREATE TABLE produzent (
   marke    VARCHAR(100) PRIMARY KEY,
   ursprung VARCHAR(100) NOT NULL,
   name     VARCHAR(100),
   familie  VARCHAR(100)
);

CREATE TABLE frucht (
   name            VARCHAR(100),
   familie         VARCHAR(100),
   hauptProduzent  VARCHAR(100),
   FOREIGN KEY (hauptProduzent) REFERENCES produzent(marke),
   PRIMARY KEY (name, familie)
);

CREATE TABLE kauft(
   id           INTEGER REFERENCES importeur(id),
   produzent    VARCHAR(100) REFERENCES produzent(marke),
   name         VARCHAR(100),
   familie      VARCHAR(100),
   FOREIGN KEY (name, familie) REFERENCES frucht(name,familie),
   PRIMARY KEY(id,produzent,name,familie)
);

ALTER TABLE produzent ADD CONSTRAINT p_mainExport
       FOREIGN KEY (name,familie) REFERENCES frucht(name,familie)
        DEFERRABLE INITIALLY DEFERRED;
```

Hinweis: Achten Sie bei den Statements auf die Reihenfolge.

Aufgabe 5: Rekursive Abfragen (14)

Gegeben ist die folgende Rekursive Abfrage auf dem Datenbank-Schema des vorherigen Beispiels:

```
WITH RECURSIVE tmp(name,familie) AS
(
SELECT  name,familie
FROM    produzent
WHERE   marke = 'SanLucar'
UNION ALL
SELECT  k.name,k.familie
FROM    produzent p, kauft k, frucht f NATURAL JOIN tmp t
WHERE   p.marke = f.hauptProduzent AND k.produzent = p.marke AND (k.id / 10) % 2 = 0
)
SELECT name FROM tmp GROUP BY name,familie;
```

Werten Sie diese Abfrage auf der Datenbank-Instanz, die auf der letzten Seite angegeben ist, aus:

name
Pinkerton
McIntosh
Cashew
Dwarf Cavendish
McIntosh
Blue Java

Aufgabe 6: PL/SQL Trigger (14)

Wenn ein Tupel K aus `kauft` gelöscht wird, soll die `frucht` Relation entsprechend angepasst werden. Setzen Sie mittels eines PL/pgSQL Triggers `trF` und zugehöriger Prozedur dazu folgendes Verhalten um:

- Finden Sie die Frucht F auf die der entsprechende Foreign Key in K verweist.
- Falls F.`HauptProduzent` gleich K.`Produzent` ist, soll der `HauptProduzent` von F wie folgt neu gesetzt werden.
  - Wenn F als `Familie` den Wert "Nuss" hat soll aus jenen Produzenten welche die Frucht F verkaufen einer zufällig als neuer `HauptProduzent` von F ausgewählt werden. (Ein Produzent verkauft eine Frucht wenn sie gemeinsam in einem Tupel von `kauft` vorkommen.)
  - Sonst soll der neue `HauptProduzent` von F der alphabetisch erste (nach Feld `Ursprung`) `Produzent` sein der die Frucht F verkauft.
  - Sie können davon ausgehen, dass ein passender Produzent immer existiert.
- Stellen Sie dabei sicher, dass der gelöschte Eintrag K, bei der Neubelegung von `HauptProduzent` nicht mehr in Betracht gezogen wird.

```
CREATE FUNCTION delK() RETURNS TRIGGER AS $$
DECLARE
  f frucht%ROWTYPE;
  newprod VARCHAR(100);
BEGIN
  SELECT * INTO f FROM frucht WHERE OLD.name = name AND OLD.familie = familie;

  IF f.Hauptproduzent != OLD.produzent THEN
     RETURN NULL;
  END IF;

  IF f.familie = 'Nuss' THEN
    SELECT produzent INTO newprod FROM kauft k
      WHERE k.name = f.name AND k.familie = f.familie
      ORDER BY RANDOM() LIMIT 1;
  ELSE
    SELECT produzent INTO newprod FROM kauft k
      JOIN produzent p ON k.produzent = p.marke
      WHERE k.name = f.name AND k.familie = f.familie
      ORDER BY p.ursprung ASC LIMIT 1;
  END IF;

  UPDATE frucht SET hauptproduzent = newprod
    WHERE name=f.name AND familie=f.familie;
  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trF AFTER DELETE ON kauft
       FOR EACH ROW EXECUTE PROCEDURE delK();
```

Gesamtpunkte: 70

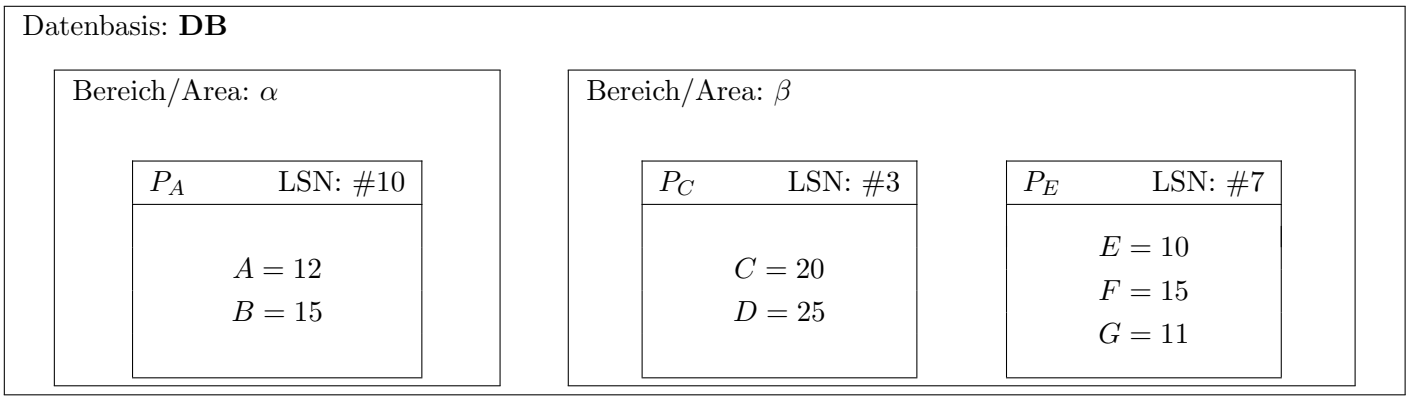
Seite Y

Sie können diesen Zettel abtrennen und brauchen ihn nicht abgeben!

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

(Zusätzliche Informationen/Angaben zu Aufgaben 1–3)

Struktur und Inhalt der Datenbank (Aufgabe 1 – 3):



Beschreibung des Formats für Logeinträge (Aufgabe 3):

"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], das Format entsprechender CLRs ist analog, d.h. ⟨LSN, TA, BOT, PrevLSN⟩ für BOT-CLRs.*

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

Beispielinstanz für Aufgabe 5:

importeur

id	land
100	Deutschland
120	Niederlande
130	Frankreich
140	Österreich
160	Belgien

produzent

marke	ursprung	name	familie
SanLucar	Spanien	Dwarf Cavendish	Banane
TerraSol	Ecuador	Pernambumco	Ananas
Kailas	Indien	Cashew	Nuss
Calavo	USA	Pinkerton	Avocado
EKM	Südafrika	Sanguinello	Orange

frucht

name	familie	hauptProduzent
Dwarf Cavendish	Banane	SanLucar
Blue Java	Banane	TerraSol
Red Dacca	Banane	Kailas
McIntosh	Apfel	Calavo
Akane	Apfel	SanLucar
Pernambumco	Ananas	TerraSol
Sanguinello	Orange	SanLucar
McIntosh	Orange	TerraSol
Pinkerton	Avocado	Calavo
Cashew	Nuss	Kailas

kauft

id	produzent	name	familie
100	SanLucar	Pinkerton	Avocado
100	SanLucar	McIntosh	Orange
120	SanLucar	Blue Java	Banane
140	SanLucar	McIntosh	Apfel
160	SanLucar	Cashew	Nuss
130	Calavo	Sanguinello	Orange
140	TerraSol	Cashew	Nuss
140	EKM	Red Dacca	Banane
100	EKM	Pernambumco	Ananas

Viel Erfolg und einen erfolgreichen Start ins neue Semester!