Gruppe **A**

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

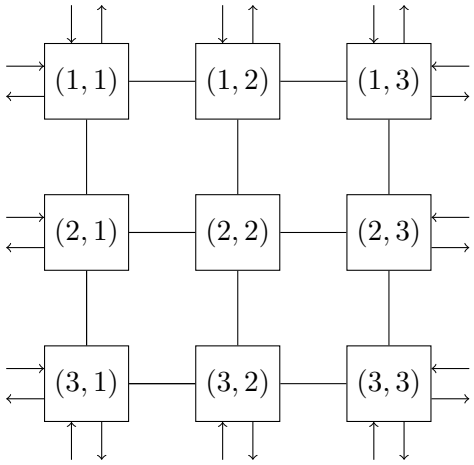
PRÜFUNG AUS	**MUSTERLÖSUNG**	27.06.2019
◯ DATENMODELLIERUNG 2 (**184.790**)	◯ DATENBANKSYSTEME (**184.686**)	**GRUPPE A**

Matrikelnr.	Familienname	Vorname

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

Aufgabe 1: Sperrprotokolle (11)

Gegeben ist die unten (links) dargestellte Kreuzungssituation. Um Unfälle zu vermeiden ist der Kreuzungsbereich in 9 Felder unterteilt, welche wie angegeben mit $(1,1),\ldots,(3,3)$ bezeichnet werden. Fahrzeuge können sich dabei von einem Feld zu einem anderen Feld nur entlang der eingezeichneten Linien bewegen, und die Kreuzung kann entsprechend der Pfeile betreten oder verlassen werden. Um Unfälle zu vermeiden darf sich zu jedem Zeitpunkt auf jedem Feld maximal ein LKW (und kein PKW) oder eine beliebige Anzahl an PKWs (aber kein LKW) aufhalten. Um dies sicherzustellen wird das *conservative 2PL* Protokoll (auch als *striktes 2-Phasen Sperrprotokoll mit Preclaiming* bezeichnet) verwendet. Bevor sie ein Feld befahren dürfen müssen LKWs einen Exclusive Lock dafür besitzen, und PKWs einen Shared Lock.



LKW_1	PKW_2	LKW_3	PKW_4	LKW_5
$(1,1)$				
$(2,1)$	$(1,3)$			
$(2,2)$	$(1,2)$	$(1,3)$		
		$(2,3)$	$(3,2)$	
$(3,2)$	$(2,2)$			
		$(3,3)$	$(2,2)$	$(2,3)$
out	$(1,2)$		$(1,2)$	$(3,3)$
				out
	out		$(1,1)$	
		out		
			out	

Gegeben ist weiters die oben (rechts) gezeigte "Historie" gewünschter Routen verschiedener Fahrzeuge durch die Kreuzung. Diese gibt an, wann welches Fahrzeug versucht auf welchem Feld zu sein. Dabei bezeichnet "out" den Moment an dem ein Fahrzeug die Kreuzung verlässt. Prinzipiell passieren alle Einträge in einer Zeile gleichzeitig. Wenn jedoch zwei Fahrzeuge in der selben Zeile die selbe Sperre anfordern sollten, erhält sie das weiter links stehende Fahrzeug. In einer Zeile freigegebene Sperren stehen erst in der nächsten Zeile wieder zur Verfügung.

Wie bereits erwähnt wird der tatsächliche Verkehr in der Kreuzung mittels des *strikten 2-Phasen Sperrprotokolls mit Preclaiming* koordiniert.

a) Geben Sie an, in welcher Reihenfolge die Fahrzeuge tatsächlich in die Kreuzung einfahren dürfen.

LKW_1, LKW_3, PKW_4, PKW_2, LKW_5 .....

b) Kommt es bei der gegebenen Historie zu einem Deadlock?

◯ ja	⊗ nein

(Richtige Antwort 1 Punkt, Falsche Anwort -1 Punkt, minimal 0 Punkte auf Aufgabe 1)

c) Kann es bei diesem Verfahren generell zu einem Deadlock kommen (für eine beliebige Historie)? *Falls ja,* geben Sie ein (gültig) verzahntes Bewegungsmuster einer entsprechenden Anzahl von Fahrzeugen an. Verwenden Sie dazu die Notation $(x_1, y_1)_{FzK1}, (x_2, y_2)_{FzK2}, \ldots$, wobei (x_i, y_i) Felder der Kreuzung bezeichnen und *FzKi* das Fahrzeug bezeichnet, welches sich auf das Feld bewegt. Geben Sie dabei nicht nur für jedes Fahrzeug die Felder an welche es durchfahren konnte, sondern auch für jedes Fahrzeug jenes Feld für welche es die Sperre nicht erhalten kann. *Falls nein,* geben Sie eine kurze (1 Satz) Begründung an.

Deadlock möglich: ◯ ja ⊗ nein

`Nein, da beim strikten 2-Phasen Sperrprotokoll mit Preclaiming .....`

`keine Deadlock auftreten können. .....`

d) Angenommen, anstelle der Verwendung des strikten 2-Phasen Sperrprotokolls mit Preclaiming wäre es erlaubt, benötigte Felder zu einem beliebigen Zeitpunkt zu sperren, jedoch müssten Fahrzeuge sich den "Rückweg offenhalten", d.h. gesperrte Felder dürften erst wieder freigegeben werden wenn die Kreuzung wieder verlassen wurde.

Welcher Variante des 2-Phasen Sperrprotokolls entspricht diese Regelung?

`Dem strikten 2-Phasen Sperrprotokoll. .....`

Aufgabe 2: Protokollierung (Logging) (12)

Gegeben sind jeweils der Inhalt der (persistenten) Log-Datei, des (flüchtigen) Log-Puffers, sowie manchmal eine Auflistung der Seiten im Hintergrundspeicher. Das verwendete Log-Format ist

`[LSN, Transaktion, PageID, Redo, Undo, PrevLSN]`, bzw. `[LSN, Transaktion, BOT/COMMIT, PrevLSN]`.

Achtung! Für das gesamte Beispiel gilt: Ankreuzen alleine gibt keine Punkte!

a) Bestimmen Sie, ob bei dem dargestellten Zustand das WAL (write ahead log)-Prinzip eingehalten wurde, oder nicht. *Falls es eingehalten wurde*, geben Sie eine konkrete Aktion an, welche das WAL-Prinzip verletzen würde. Verwenden Sie eine der folgenden drei Arten von Aktionen: Ausschreiben eines Log-Eintrags aus dem Puffer (geben Sie die LSN an); Auslagern einer Seite in den Hintergrundspeicher (geben Sie die PageID an); Erstellen eines neuen Log-Eintrags (geben Sie den Eintrag an). *Falls das WAL-Prinzip nicht eingehalten wurde,* beschreiben Sie wie/wodurch es verletzt wurde.

Log-Datei

[#2, T_2, `BOT`, #0]
[#4, T_2, P_a, ·, ·, #2]
[#6, T_2, P_b, ·, ·, #4]
[#8, T_2, P_b, ·, ·, #6]
[#10, T_2, `COMMIT`, #8]

Log-Puffer

[#1, T_1, `BOT`, #0]
[#3, T_3, `BOT`, #0]
[#5, T_1, P_c, ·, ·, #1]
[#7, T_3, P_a, ·, ·, #3]
[#9, T_1, P_c, ·, ·, #5]
[#11, T_1, P_a, ·, ·, #9]

Seiten im Hintergrundspeicher

P_a	LSN: #4

P_b	LSN: #0

P_c	LSN: #0

Das WAL-Prinzip wurde eingehalten: ◯ ja ⊗ nein

Begründung: `Die Reihenfolge der Log-Einträge wurde beim Ausschreiben nicht eingehalten (es wurden nur Einträge von` T_2 `ausgeschrieben).`

b) In diesem Beispiel sind die Redo- und Undo- Einträge mittels *physischer Protokollierung* angegeben. Des weiteren soll als Auslagerungsstrategie die Kombination *force* und *steal* verwendet und das WAL Prinzip eingehalten werden. Ist die dargestellte Situation in so einem Setting möglich/gültig? *Falls ja,* geben Sie eine konkrete, zusätzliche Aktion an, welche in dieser Situation verboten wäre. Sie können aus der selben Art von Aktionen wie in Aufgabe a) wählen. *Falls nein,* beschreiben Sie warum/wo es zu einer Verletzung kam.

Log-Datei

[#1, T_1, `BOT`, #0]
[#2, T_2, `BOT`, #0]
[#4, T_1, P_A, A=10, A=5, #1]
[#5, T_1, P_B, B=15, B=20, #4]
[#6, T_2, P_A, A=5, A=10, #2]
[#7, T_1, `COMMIT`, #5]

Log-Puffer

[#8, T_2, P_B, B=20, B=15, #6]

Seiten im Hintergrundsp.

P_A	LSN: #6
$A = 10$	

P_B	LSN: #5
$B = 15$	

Dargestellte Situation möglich: ◯ ja ⊗ nein

Begründung: `Die Seite` P_A `enthält einen falschen Wert für` A`: Beim COMMIT einer Transaktionen müssen geänderte Seiten ausgeschrieben werden. D.h. beim COMMIT von` T_1 `muss die Seite` P_A `ausgeschrieben werden. Diese muss für` A `aber den Wert` 5 `und nicht` 10 `enthalten, da` A `anschließend nicht mehr geändert wurde.`

c) In diesem Beispiel sind die Redo- und Undo- Einträge mittels *logischer Protokollierung* angegeben. Könnte in der unten dargstellten Situation unter Einhaltung von *not steal* und *force* als Auslagerungsstrategie und des WAL-Prinzips ein `COMMIT` der Transaktion T_2 erfolgen? *Falls ja,* beschreiben Sie welche Schritte (in welcher Reihenfolge) bei so einem `COMMIT` von T_2 unternommen werden müssten. *Falls nein,* begründen Sie kurz warum.

Log-Datei
[#1, T_2, `BOT`, #0]
[#2, T_2, P_B, B+=3, B-=3, #1]
[#3, T_1, `BOT`, #0]
[#4, T_1, P_C, C-=5, C+=5, #3]

Log-Puffer
[#6, T_3, `BOT`, #0]
[#7, T_1, P_B, B+=1, B-=1, #4]
[#8, T_3, P_C, C+=10, C-=10, #6]

Commit unter "not steal" und "force" möglich: ◯ ja ⊗ nein

Begründung: Auf Grund von *force* müsste die Seite P_B bei dem COMMIT ausgelagert ..... werden, da T_2 diese verändert hat. Auf Grund von *not steal* darf aber genau das ..... nicht passieren, da die noch aktive Transaktion T_1 P_B ebenfalls verändert hat. .....

Aufgabe 3: Eigenschaften von Transaktionen (12)

Betrachten Sie die unten angegebene Historie, welche durch eine Abfolge von Elementaroperationen der vier Transaktionen T_1, T_2, T_3 und T_4 auf den Datensätzen A, B, C und D gegeben ist. *Dabei bezeichnen b_i und c_i den Beginn bzw. den erfolgreichen Abschluss (commit) von Transaktion T_i, a_i bezeichnet den Abbruch (abort) von Transaktion T_i, und $r_i(O)$ und $w_i(O)$ Lese- bzw. Schreibzugriffe von Transaktion T_i auf das Objekt O.*

T_1	T_2	T_3	T_4
b_1	b_2	b_3	b_4
$r_1(A)$			
	$r_2(A)$		
		$r_3(C)$	
			$r_4(D)$
$w_1(A)$			
	$r_2(B)$		
		$w_3(B)$	
	$w_2(C)$		
			$w_4(B)$
$w_1(D)$			
c_1			
		$r_3(A)$	
		$w_3(C)$	
			$r_4(A)$
	$r_2(D)$		
		c_3	
			c_4
	$w_2(C)$		
	$r_2(B)$		
	$w_2(B)$		
	c_2		

a) Geben Sie an, zwischen welchen Transaktionen eine Leseabhängigkeit besteht. D.h., geben Sie für jede Transaktion an, von welchen anderen Transaktionen diese Transaktion liest (falls eine Transaktion von keiner anderen Transaktion liest, streichen Sie das entsprechende Feld bitte durch).

a) Leseabhängigkeiten:

T_1 liest von `--` ..... T_2 liest von T_1, T_4 ..

T_3 liest von T_1 ..... T_4 liest von T_1 .....

b) Bestimmen Sie anschließend, ob die Historie kaskadierendes Rücksetzen vermeidet oder nicht, sowie ob sie strikt ist oder nicht. Geben Sie jeweils eine kurze Begründung an Hand der Historie, an.
(*Achtung:* Ankreuzen alleine ohne eine Begründung gibt keine Punkte!)

b) Eigenschaften:

Historie vermeidet kaskadierendes Rücksetzen: ⊗ ja ○ nein

Begründung: `Alle Transaktionen von denen gelesen wird haben ihr COMMIT vor der entsprechenden Leseoperation.`

Historie ist strikt: ○ ja ⊗ nein

Begründung: `z.B. überschreibt` $w_3(C)$ `den Wert von` ..... $w_2(C)$`, ohne dass` T_2 `zuvor beendet wurde.` .....

c) Betrachten Sie jeweils die unten angegebenen Paare von Transaktionen. Bestimmen Sie für jedes der beiden Paare, ob die beiden Transaktionen konfliktserialisierbar sind oder nicht.
Falls nicht, geben Sie eine Sequenz von Paaren von Konfliktoperationen der beiden Transaktionen an, welche die Existenz einer konfliktäquivalenten, seriellen Historie ausschließen.
Falls die Transaktionen konfliktserialisierbar sind, geben Sie eine konfliktäquivalente, serielle Historie (= Sequenz von Elementaroperationen!) der beiden Transaktionen an.

Transaktionen T_1 und T_3: Konfliktserialisierbar: ⊗ ja ○ nein

Antwort: b_1, $r_1(A)$, $w_1(A)$, $w_1(D)$, c_1, b_3, $r_3(C)$, $w_3(B)$, $r_3(A)$, $w_3(C)$, c_3 .....

.....

Transaktionen T_2 und T_3: Konfliktserialisierbar: ○ ja ⊗ nein

Antwort: `mehrere Antworten möglich, z.B.` $(r_2(B), w_3(B))$, $(w_3(B), w_2(B))$, `oder` .....

$(r_3(C), w_2(C))$, $(w_2(C), w_3(C))$, `oder` $(w_2(C), w_3(C))$, $(w_3(C), w_2(C))$ .....

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

Aufgabe 4: Erstellen eines Datenbankschemas mittels SQL (7)

Folgendes Schema sei gegeben:

passagier(<u>name, passnummer</u>, status)
fliegt_in(<u>name, pass</u>: *(passagier.name, passagier.passnummer)*, <u>flug</u>: *flug.code*)
flug(<u>code</u>, von: *flughafen.name*, nach: *flughafen.name*)
flughafen(<u>name</u>, land)

Jeder Passagier ist durch seinen Namen und seine Passnummer eindeutig identifiziert. Darüber hinaus wird auch der Vielfliegerstatus vermerkt. Dieser Status ist ein ENUM bestehend aus einem von vier Werten: 'Kein', 'Vielflieger', 'Senator' oder 'Sapphire'. Darüber hinaus wird gespeichert welcher Passagier für welchen Flug eingecheckt wurde.

Ein Flughafen besteht aus einem eindeutigen Namen und einem Land. Jeder Flug hat einen einzigartigen Code, und es wird vermerkt von welchen Flughafen er startet und zu welchem Flughafen er fliegt.

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

```
CREATE TABLE flughafen (
   name VARCHAR(100) PRIMARY KEY,
   land VARCHAR(100) NOT NULL
);

CREATE TABLE flug (
   code VARCHAR(100) PRIMARY KEY,
   von  VARCHAR(100) REFERENCES flughafen(name),
   nach VARCHAR(100) REFERENCES flughafen(name)
);

create type status as enum('Kein','Vielflieger','Senator','Sapphire');

CREATE TABLE passagier (
   name        VARCHAR(100),
   passnummer  VARCHAR(100),
   status      status,
   PRIMARY KEY (name, passnummer)
);

CREATE TABLE fliegt_in(
   name VARCHAR(100),
   pass VARCHAR(100),
   code VARCHAR(100) REFERENCES flug(code),
   FOREIGN KEY (name,pass) REFERENCES passagier(name,passnummer),
   PRIMARY KEY (name,pass,code)
);
```

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 T(nach,name,pass) AS
(
  SELECT  f.nach,fl.name,fl.pass
  FROM    fliegt_in fl, flug f, passagier p
  WHERE   f.code = 'OS18457' AND fl.flug = f.code   AND
          fl.name = p.name AND fl.pass = p.passnummer AND
          p.status != 'Kein'
UNION ALL
  SELECT  f.nach, T.name, T.pass
  FROM    fliegt_in fl, flug f, passagier p, T
  WHERE   fl.flug = f.code AND f.von = T.nach AND
          fl.pass = T.pass AND fl.name = T.name AND
          fl.name = p.name AND fl.pass = p.passnummer AND
          p.status != 'Kein'
)
SELECT nach FROM T GROUP BY nach;
```

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

```
nach
------
AMS
PEK
CDG
ATL
GRU
LHR
```

Aufgabe 6: PL/SQL Trigger (14)

Erstellen Sie einen PL/pgSQL Trigger `trFl`, der vor dem Einfügen eines Eintrags in die `flug`-Tabelle für jede Zeile die eingefügt werden soll, die Funktion `newFl` aufruft. Diese Funktion soll sicherstellen, dass im Feld `nach` das Foreign Key Constraint nicht verletzt wird. Konkret soll folgendes Verhalten beim Einfügen eines Tupels $(code, von, nach)$ umgesetzt werden:

- Falls der im Feld *nach* angebene Flughafen existiert soll das Tupel unverändert eingefügt werden.
- Existiert der Flughafen nicht, so soll ein neuer Wert $nach_{neu}$ bestimmt werden der **anstatt** *nach* **im einzufügenden Tupel** gesetzt wird. Der Wert für $nach_{neu}$ wird wie folgt bestimmt:
  - Wenn es zumindest einen Flughafen gibt dessen `name` mit dem Wert von *nach* beginnt, soll der `name` einer dieser Flughafens für $nach_{neu}$ gewählt werden.
  - Existiert kein solcher Flughafen soll "VIE" für $nach_{neu}$ gewählt werden.

```
CREATE FUNCTION newFl() RETURNS TRIGGER AS $$
DECLARE
    alt VARCHAR(100);
BEGIN
  IF EXISTS(SELECT 1 FROM flughafen WHERE name = NEW.nach)
  THEN
    RETURN NEW;
  END IF;

  IF EXISTS(SELECT 1 FROM flughafen WHERE name LIKE NEW.nach || '%')
  THEN
    SELECT name INTO alt FROM flughafen WHERE name LIKE NEW.nach || '%';
  ELSE
    alt := 'VIE';
  END IF;

  NEW.nach := alt;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trFl BEFORE Insert ON flug
       FOR EACH ROW EXECUTE PROCEDURE newFl();
```

Gesamtpunkte: 70

Sie können diesen Zettel abtrennen und brauchen ihn nicht abgeben!

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

Beispielinstanz für Aufgabe 6:

flughafen

name	land
VIE	Österreich
AMS	Niederlande
GRU	Brasilien
CDG	Frankreich
LHR	England
PEK	China
ATL	USA
ASU	Paraguay
MAD	Spanien

flug

code	von	nach
OS18457	VIE	CDG
OS15813	ATL	VIE
AF33474	CDG	ATL
AF23514	CDG	GRU
AF48891	PEK	LHR
LA12633	GRU	AMS
KL49152	AMS	PEK
UA88351	ATL	LHR
LA11569	ASU	MAD
AF11489	CDG	MAD

passagier

name	passnummer	status
Dennis Muilenburg	12.12.456	Senator
Alain Bellemare	G81465	Sapphire
Alexis Hoensbroech	P987123	Vielflieger
Guillaume Faury	X56771	Kein

fliegt_in

name	pass	flug
Alain Bellemare	G81465	OS18457
Alain Bellemare	G81465	KL49152
Alain Bellemare	G81465	LA12633
Alain Bellemare	G81465	AF23514
Dennis Muilenburg	12.12.456	OS18457
Dennis Muilenburg	12.12.456	AF23514
Dennis Muilenburg	12.12.456	AF48891
Alexis Hoensbroech	P987123	OS18457
Alexis Hoensbroech	P987123	AF33474
Alexis Hoensbroech	P987123	LA11569
Alexis Hoensbroech	P987123	UA88351
Guillaume Faury	X56771	OS18457
Guillaume Faury	X56771	AF11489

Schöne und erholsame Ferien!