Gruppe **A**

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

PRÜFUNG AUS	**MUSTERLÖSUNG**	17.09.2020
◯ 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 gibt es nur in Zusammenhang mit geforderter Erklärung/Beispiel/...!

Notation:

In den Aufgaben 1 – 3 wird die folgende (aus der Vorlesung bekannte) Notation für Transaktionen T_i verwendet:

- $r_i(O)$ und $w_i(O)$: Lese- bzw. Schreibzugriff von T_i auf Objekt O.
- b_i, c_i, a_i: Beginn (`BEGIN OF TRANSACTION`), Commit (`COMMIT`) bzw. Abbruch (`ABORT`/`ROLLBACK`) von T_i.

Die Indizes $_i$ können weggelassen werden, wenn klar ist zu welcher Transaktion eine Operation gehört.

Des weiteren wird das aus der Vorlesung bekannte Format für Logeinträge verwendet:

`[LSN, TA, PageID, Redo, Undo, PrevLSN]` für "normale" Einträge, bzw.
`[LSN, TA, BOT, PrevLSN]` für `BOT` Log-Einträge und
`[LSN, TA, COMMIT, PrevLSN]` für `COMMIT` Einträge.

Kompensations Logeinträge (Compensation Log Records) haben das Format
⟨`LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN`⟩ bzw.
⟨`LSN, TA, BOT, PrevLSN`⟩

Dabei stellt `LSN` die Log-Sequence Nummer dar, `TA` die Transaktion, `PageID` ist die veränderte Seite, `Redo` und `Undo` die für das Redo bzw. Undo benötigten Informationen, `UndoNextLSN` ist die LSN des nächsten Logeintrags der selben Transaktion welcher zurückgesetzt werden soll, und `PrevLSN` die LSN des vorherigen Logeintrags derselben Transaktion.

Im Falle logischer Protokollierung sollen die Änderungen zum aktuellen Datenbestand nur mittels *Addition* bzw. *Subtraktion* angegeben werden, z.B. [·, ·, ·, X+=d_1, X-=d_2, ·].

Aufgabe 1: Eigenschaften von Transaktionen (12)

Gehen Sie davon aus, dass in einem DBMS die folgenden Lock-Level implementiert sind, und jede Transaktion sich aussuchen kann, in welchem dieser Level sie ausgeführt werden soll.

L1: Share-Locks werden vor jeder Leseoperation angefragt und sofort nach der Leseoperation wieder freigegeben (kein 2PL); 2PL für Exclusive-Locks (unabhängig von Freigaben von Share-Locks).

L2: gemeinsames 2PL für Share- und Exclusive-Locks, für Exclusive-Locks wird jedoch striktes 2PL verwendet. Lock-Upgrades (eines Share-Locks in einen Exclusive-Lock) sind möglich und zählen nicht als Freigabe.

Beantworten Sie die folgenden Fragen mit "Ja" oder "Nein". Bei *"Ja"* geben Sie bitte eine Historie an (Operationen b_i, c_i, $r_i(O)$, $w_i(O)$) welche mit dem angenommenen Lock-Level konform ist, und die gesuchte Eigenschaft erfüllt. Bei *"Nein"* begründen Sie Ihre Antwort kurz (1-2 Sätze).

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) Betrachten Sie die Transaktion

$$T_1 \colon b_1, r_1(B), w_1(B), r_1(A), c_1$$

und eine beliebige Transaktion T_2. Nehmen Sie an dass T_1 im Level L1 läuft und T_2 im Level L2.

i) Kann es zu einem *Lost-Update* von T_2 kommen?

Lost-Update von T_2: ⊗ ja ○ nein

Mögliche Lösung: .....

$b_1, b_2, r_1(B), r_2(B), w_2(B), c_2, w_1(B), r_1(A), c_1$ .....

.....

ii) Kann es zu einem *Dirty Read* durch T_2 kommen (T_2 liest)?

Dirty Read durch T_2: ⊗ ja ○ nein

Mögliche Lösung: .....

$b_1, b_2, r_1(B), w_1(B), r_2(B), c_2, r_1(A), c_1$ .....

.....

b) Betrachten Sie zusätzlich zu T_1 aus a) die Transaktion

$$T_3 \colon b_3, w_3(B), r_3(A), w_3(A), w_3(C), c_3$$

welche im Level L2 läuft.

Gibt es eine gültige Historie der beiden Transaktionen welche kaskadierendes Rücksetzen **nicht** vermeidet?

Historie die kaskadierendes Rücksetzen **nicht** vermeidet: ○ ja ⊗ nein

Da T_3 im Level L2 läuft werden alle Exclusive-Locks .....

erst beim Commit freigegeben, somit kann T_1 nicht vorher darauf zugreifen. .....

T_1 wiederum schreibt nur B, welcher nie von T_3 gelesen wird. .....

Aufgabe 2: Protokollierung und Wiederanlauf (Logging und Recovery) (12)

Betrachten Sie die unten angegebenen Logeinträge der vier Transaktionen T_1, T_2, T_3 und T_4.

a) Führen Sie anhand dieser Log-Einträge ein Recovery (nach dem ARIES Verfahren) durch, und geben Sie die dabei entstehenden Log-Einträge an.

Logeinträge (Angabe)

[#1, T_3, BOT, #0]

[#2, T_1, BOT, #0]

[#3, T_4, BOT, #0]

[#4, T_2, BOT, #0]

[#5, T_4, P_A, A+=25, A-=25, #3]

[#6, T_3, P_C, C+=0, C-=0, #1]

[#7, T_4, P_B, B-=5, B+=5, #5]

[#8, T_2, P_A, A-=24, A+=24, #4]

[#9, T_4, COMMIT, #7]

[#10, T_1, P_B, B+=5, B-=5, #2]

⟨#11, T_2, P_A, A+=24, #8, #4⟩

[#12, T_3, P_B, B+=5, B-=5, #6]

[#13, T_1, P_A, A-=35, A+=35, #10]

Logeinträge (Lösung)

⟨#14, T_1, P_A, A+=35, #13, #10⟩ .....

⟨#15, T_3, P_B, B-=5, #12, #6⟩ .....

⟨#16, T_1, P_B, B-=5, #14, #2⟩ .....

⟨#17, T_3, P_C, C-=0, #15, #1⟩ .....

⟨#18, T_2, BOT, #11⟩ .....

⟨#19, T_1, BOT, #16⟩ .....

⟨#20, T_3, BOT, #17⟩ .....

.....

b) Ist es möglich, anhand der gegebenen Logeinträge zu bestimmen, ob die zugehörige Historie strikt (im Sinne der Klassifikation von Historien im Rahmen der Mehrbenutzersynchronisation) ist?
Falls ja, geben Sie an ob die zugehörige Historie strikt war oder nicht (und warum).
Falls nein, begründen Sie kurz (1-2 Sätze) Ihre Antwort.

Strikt kann erkannt werden: ⊗ ja ○ nein

Nicht strikt: T_3 überschreibt in Logeintrag #12 den Wert von B, ...

welchen T_1 im Eintrag #10 geschrieben hat. Da T_1 in der .....

Zwischenzeit nicht abgeschlossen hat, ist die Historie nicht strikt gewesen. .....

c) Nehmen Sie an, als Ersetzungsstrategie würde *force* und *steal* verwendet.

Bestimmen Sie für die beiden Seiten P_B und P_C möglichst genau die möglichen Werte der LSN (größte LSN deren Änderung auf der Seite durchgeführt wurde) welche am Beginn des Wiederanlaufs vermerkt war (d.h. jenen Wert, der auf der Seite im Hintergrundspeicher vermerkt war, mit welcher dass Recovery durchgeführt wurde).

Seite	**untere Schranke**		**obere Schranke**
P_B	#7	≤ LSN ≤	#12 ...
P_C	#0	≤ LSN ≤	#6 .....

Nehmen Sie an dass die LSN beider Seiten ursprünglich #0 war.

Aufgabe 3: Sperrprotokolle (Locking) (11)

a) Gegeben ist die untenstehende Folge von Exclusive- und Share-Sperren (XL(O), SL(O)), Freigaben von Sperren (relXL(O), relSL(O)), sowie Lese- und Schreiboperationen. Bei Einträgen in der selben Zeile spielt die zeitliche Anordnung der betroffenen Operationen keine Rolle, es kann eine beliebe Reihenfolge angenommen werden.

Geben Sie für jede der fünf Transaktionen T_1, T_2, T_3, T_4 und T_5 an, ob sie das *2-Phasen Sperrprotokoll (2PL)* befolgt. Falls nicht, beschreiben Sie wodurch 2PL verletzt wird.

T_1	T_2	T_3	T_4	T_5
BOT	BOT	BOT	BOT	BOT
SL(A)		SL(B)		
				XL(D)
	SL(B)			
$r(A)$		SL(A)		
		$r(B)$		SL(A)
XL(C)	$r(B)$	relSL(B)		$r(D)$
		$r(A)$		
$r(C)$				$w(A)$
relSL(A)			XL(C)	
$w(C)$		XL(B)		
		$w(B)$		relSL(A)
relXL(C)				
			SL(A)	
c_1			$r(A)$	relXL(D)
	XL(D)		$w(C)$	
	$w(D)$	relSL(A)	relSL(A)	c_5
	relXL(D)	relXL(B)	relXL(C)	
	c_2	c_3	c_4	

T_1: korrektes 2PL. .....

T_2 gibt bis zum COMMIT nicht .....

alle Sperren frei (SL(B) fehlt). ....

T_3 sperrt B (XL(B)), nachdem .....

es die SL(B) bereits freigegeben hat.

T_4 sperrt C (XL(C)) obwohl T_1 .....

diese Sperre bereits hat. .....

T_5 schreibt auf A obwohl es nur .....

einen Shared Lock besitzt. .....

.....

.....

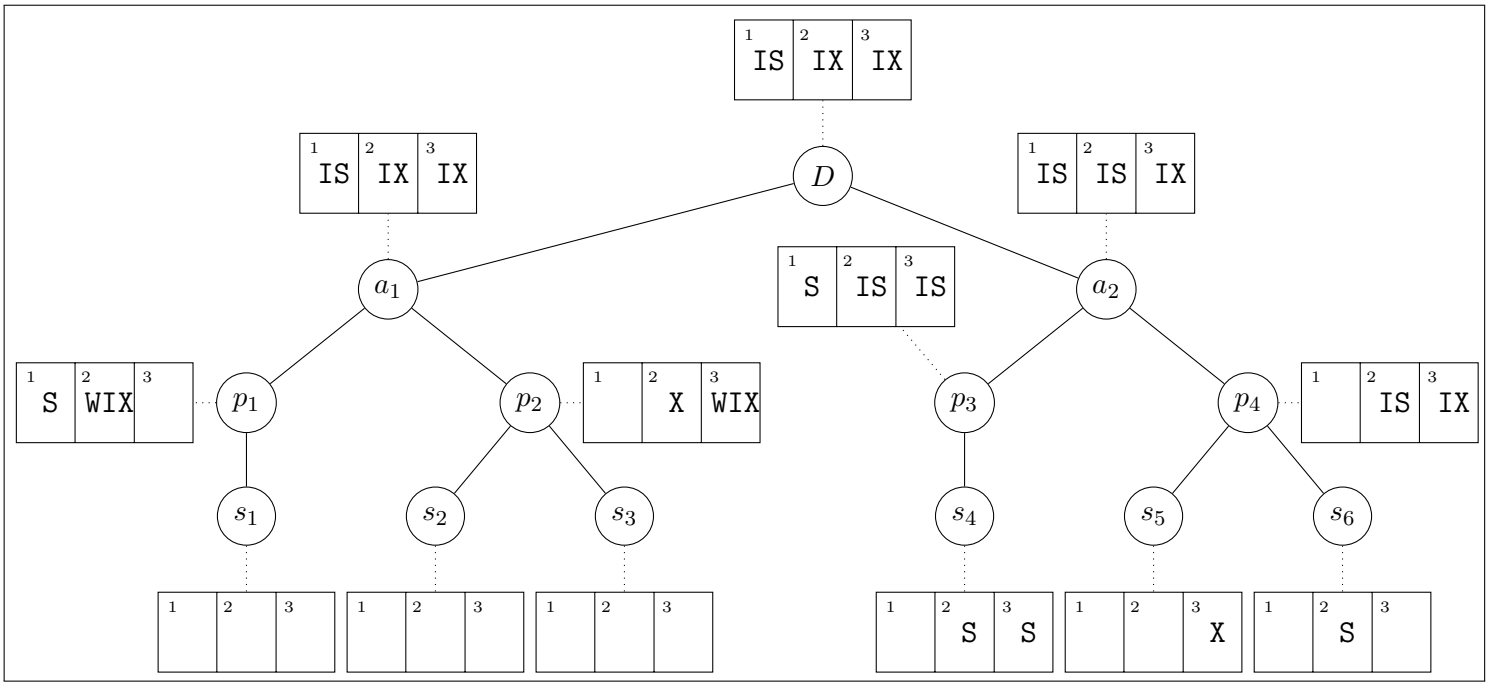
.....

.....

Hinweis: Betrachten Sie nicht nur jede Transaktion für sich.

b) Multi Granularity Locking:

Gegeben ist folgende Datenbasis-Hierarchie.



Nehmen Sie an, die Transaktionen T_1, T_2 und T_3 möchten folgende Operationen in der angegeben Reihenfolge durchführen:

$$w_3(s_5), r_3(s_4), w_2(p_2), r_2(s_6), r_1(p_1), r_1(p_3), w_3(s_2), w_3(s_1), r_2(p_2), r_2(s_4), w_2(s_1)$$

Nehmen Sie weiter an, dass die Transaktionen zum Erhalt dieser Sperren nach dem Sperrprotokoll des MGL vorgehen (und dieses auch korrekt einhalten).

Beschreiben Sie den Zustand nach Abarbeitung dieser Sperranforderungen, indem Sie in der obigen Grafik zu jedem Knoten notieren, welche Transaktionen welche Sperren auf diesem Knoten halten. Tragen Sie dazu S, X, IS und IX in das Feld mit der Transaktionsnummer ein um auszudrücken, dass die Transaktion die entsprechende Sperre hält. Sollte eine Transaktion eine Sperre angefordert aber nicht erhalten haben, tragen Sie bitte WS, WX, WIS oder WIX ein. Sollte eine Transaktion blockieren, ignorieren Sie alle weiteren Anforderungen dieser Transaktion.

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

Aufgabe 4: Erstellen eines Datenbankschemas mittels SQL und Schlüsselabhängigkeiten (11)

a) Folgendes Schema sei gegeben:

`mitarbeitende`	(id, lieblingN: *mns.name*, lieblingF: *mns.farbe*)
`produzent`	(name, umsatz)
`mns`	(name, farbe, tragt: *mitarbeitende.id*, pName: *produzent.name*)
`kontakt`	(von: *mitarbeitende.id*, mit: *mitarbeitende.id*, dauer)

Ein Unternehmen muss aufgrund der Corona-Ampel eine Mund-Nasen-Schutz (MNS) Pflicht einführen. In seiner Datenbank sind alle Mitarbeitende eindeutig durch eine ID bestimmt. Diese ID ist als Sequenz realisiert, die bei 3 startet, und in 2er Schritten erhöht wird. Ein MNS soll zur besseren Unterscheidbarkeit einen Namen erhalten, wobei die Kombination von Farbe und Name hier eindeutig sein soll. Jeder MNS ist genau einer Mitarbeitenden zugeordnet. Mitarbeitende haben auch alle eine Lieblingsmaske. Zuletzt wird auch der Produzent des MNS festgehalten. Jeder Produzent von MNS hat einen eindeutigen Namen, sowie einen Umsatz. Aufgrund gesetzlicher Pannen ist es rechtlich verlangt, dass der Umsatz ein Vielfaches von 7 sein muss. Dies soll über ein Check überprüft werden. Zuletzt hält die Tabelle `kontakt` fest, welche Mitarbeitende (`von`) mit welchen Anderen (`mit`) in Kontakt standen, sowie die Dauer des jeweiligen Kontaktes in Stunden.

Geben Sie die nötigen SQL Statements an, um obiges Schema (mit allen Konsistenzbedingungen) anzulegen. Wählen Sie passende Typen für Attribute. **Die Abkürzung VC statt VARCHAR(100) ist erlaubt.**

```
CREATE SEQUENCE seq_id INCREMENT BY 2 MINVALUE 3 NO CYCLE;

CREATE TABLE mitarbeitende(
   id        INTEGER DEFAULT nextval('seq_id') PRIMARY KEY,
   lieblingN VARCHAR(100),
   lieblingF VARCHAR(100)
);

CREATE TABLE produzent(
   name   VARCHAR(100) PRIMARY KEY,
   umsatz INTEGER NOT NULL CHECK(umsatz % 7 = 0)
);

CREATE TABLE mns(
   name      VARCHAR(100),
   farbe     VARCHAR(100),
   tragt     INTEGER REFERENCES mitarbeitende(id),
   pName VARCHAR(100) REFERENCES produzent(name),
   PRIMARY KEY(name, farbe)
);

CREATE TABLE kontakt(
   von   INTEGER REFERENCES mitarbeitende(id),
   mit   INTEGER REFERENCES mitarbeitende(id),
   dauer INTEGER,
   PRIMARY KEY(von, mit)
);

ALTER TABLE mitarbeitende ADD CONSTRAINT m_maske
   FOREIGN KEY (lieblingN, lieblingF) REFERENCES mns(name, farbe)
   DEFERRABLE INITIALLY DEFERRED;
```

Hinweis: Achten Sie bei den Statements auf die Reihenfolge.

b) Für diese Aufgabe müssen Sie sich mit verschiedenen Schlüsselabhängigkeiten auseinandersetzen.

i) Gegeben ist folgendes Schema:

`building`	(id: *room.id*, biggestRoom: *room.name*)
`room`	(id: *building.id*, name, capacity)

Es ist gefragt, dass Sie folgende Tabellen so erweitern, **dass die resultierende Datenbank-Instanz das Schema erfüllt**. Hierbei stellen die 2 Tabellen den vollständigen Inhalt der Datenbank dar. Um Punkte für diesen Teil zu erhalten, müssen Sie die Tabellen vollständig ausfüllen.

building

id	**biggestRoom**
1	Audimax
3	Audimax
4	InfHS
2	InfHS

room

id	**name**	**capacity**
3	Audimax	400
1	Audimax	300
2	InfHS	200
4	InfHS	100

ii) Gegeben ist folgendes Schema:

`building`	(id, name)
`passage`	(from: *building.id*, to: *building.id*)

Es ist gefragt, dass Sie folgende Tabellen so erweitern, **dass die resultierende Datenbank-Instanz das Schema verletzt**. Hierbei stellen die 2 Tabellen den vollständigen Inhalt der Datenbank dar. Um Punkte für diesen Teil zu erhalten, müssen Sie die Tabellen vollständig ausfüllen.

building

id	**name**
1	Älteste Hörsal
π	Hörsaal 1
μ	Hörsaal 2
-7	Hörsaal 007

passage

from	**to**
Audimax	New York
Atlantis	Hörsaal 1
Verschollener Hörsaal	Ys
Heaven	Hell

Aufgabe 5: Rekursive Abfragen (12)

Erstellen Sie folgende Rekursive SQL Abfrage:

Jede Zeile `(von, mit, dauer)` der Tabelle `kontakt` beschreibt einen *direkten Kontakt* zwischen zwei Mitarbeitenden `von` und `mit`. Wir definieren Mitarbeitende A als *Kontaktperson* für Mitarbeitende B, falls es einen direkten Kontakt gab, oder es einen direkten Kontakt zwischen A und Mitarbeitenden C gab, wobei C selbst direkter Kontakt von B ist, usw.
Achtung: Sie sollen die Tabelle nicht symmetrisch erweitern, sondern immer nur auf (`von`, `mit`) Paare, und ihre Erweiterung in Richtung `mit`, schauen.

Als *einstündiger Kontakt* bezeichnen wir alle Kontakte, deren Dauer zumindest eine Stunde beträgt. Eine "einstündige Kontaktperson" ist eine Kontaktperson, mit der entweder ein direkter Kontakt mit Dauer von 1 Stunde oder mehr besteht, oder bei der alle anderen Kontaktschritte ebenfalls mindestens eine Stunde andauern. Analog definiert sind "zweistündige Kontaktpersonen", "dreistündige Kontaktpersonen", usw.

Gesucht sind vom Unternehmen sämtliche zweistündigen Kontaktpersonen des Mitarbeitenden mit ID "1". Als zusätzliche Sicherheit sollen allerdings die *direkten Kontakte* von Mitarbeitenden mit ID "1" bereits ausgegeben werden, falls der direkte Kontakt mindestens eine Stunde beträgt. Diese direkten Kontakte sollen *danach* regulär in die Rekursion einfließen, das heißt es soll deren zweistündige Kontaktpersonen ermittelt werden.

Die Abfrage soll die ID und den LieblingsMNS der gesuchten Mitarbeitenden ausgeben. Mitarbeitende die über mehrere "Pfade" als zweistündigen Kontaktpersonen auftauchen, sollen auch entsprechend mehrfach in der Liste aufscheinen.

Geben Sie die nötigen SQL Statements an, um die beschriebene Rekursive Abfrage zu implementieren.

```
WITH RECURSIVE tmp(von) AS
(
 SELECT  mit
 FROM    kontakt
 WHERE   von = '1' AND dauer >= 1
UNION ALL
 SELECT  mit
 FROM    tmp NATURAL JOIN kontakt
 WHERE   dauer >= 2
)
SELECT  id, lieblingN,lieblingF
FROM    tmp, mitarbeitende
WHERE   tmp.von = mitarbeitende.id ;
```

Hinweis: Achten Sie darauf, dass ihre Abfrage terminiert. Sie dürfen jedoch davon ausgehen, dass die Datenbank keine zyklischen Kontakte enthält.

Aufgabe 6: PL/SQL Trigger (12)

Nehmen Sie an, dass die **Funktionen und Trigger** wie auf **Seite T** (am vorletzten Blatt dieser Prüfung) definiert wurden. Die Aufgaben beziehen sich auf die Beispielinstanz auf **Seite B** (letztes Blatt der Prüfung).

In jedem der folgenden Aufgaben ist ein SQL Statement gegeben das **über die Beispielinstanz** ausgeführt wird. (Das Statement in Aufgabe a hat keinen Einfluss auf Aufgabe b usw.) Geben Sie die Ausgabe der `SELECT`-Statements an. **Falls ein Fehler auftreten würde, geben Sie an welcher Fehler auftritt.**

a)

```
UPDATE mns set tragt='7' WHERE name='A';

SELECT * FROM mns WHERE name='A';
SELECT * FROM kontakt;
```

mns: (A,grau,7), (A,rot,7)
kontakt: (3 , 5 , 90), (5 , 9 , 20), (3 , 7 , 3), (7 , 3 , 3)

---

b)

```
UPDATE mns SET tragt='3' WHERE name='F';

SELECT * FROM mns WHERE name='F';
SELECT * FROM kontakt;
```

mns: (F, violet,3,Y)
kontakt: (3,5,90), (5,9,20), (5,3,3)

c)

```
BEGIN;
  INSERT INTO kontakt VALUES (5, 7, 100);
  UPDATE mns SET tragt='3', farbe='gelb' WHERE pName='Y';
COMMIT;

SELECT * FROM mns WHERE pName='Y';
SELECT * FROM kontakt;
```

mns: (A,gelb,3), (F,gelb,3), (G,gelb,3)
kontakt: (3 , 5 , 90), (5 , 9 , 20), (5 , 7 , 100), (7 , 5 , 100), (5 , 3 , 3), (13 , 3 , 3), (3 , 13 , 3)

Gesamtpunkte: 70

Viel Erfolg und ein möglichst reibungsloses und erfolgreiches WS 2020!

Sie können diesen Zettel abtrennen und brauchen ihn nicht abgeben!

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

Trigger für Aufgabe 6:

```
CREATE FUNCTION maskContact() RETURNS TRIGGER AS $$
BEGIN
    IF OLD.tragt = NEW.tragt THEN
       RETURN NULL;
    END IF;
    IF NEW.farbe != 'rot' THEN
        INSERT INTO kontakt VALUES (OLD.tragt, NEW.tragt, 3);
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trA AFTER UPDATE ON mns
       FOR EACH ROW EXECUTE PROCEDURE maskContact();




CREATE FUNCTION completeContact() RETURNS TRIGGER AS $$
BEGIN
    IF NOT EXISTS(SELECT 1 FROM kontakt WHERE
       von = NEW.mit AND mit = NEW.von)
    THEN
      INSERT INTO kontakt VALUES (NEW.mit, NEW.von, NEW.dauer);
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trB AFTER INSERT ON kontakt
       FOR EACH ROW EXECUTE PROCEDURE completeContact();
```

Seite B

Beispielinstanz für Aufgabe 6:

Sie können diesen Zettel abtrennen und brauchen ihn nicht abgeben!

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

Mitarbeitende

id	lieblingN	lieblingF
3	A	rot
5	E	blau
7	E	blau
9	C	gelb
11	A	rot
13	B	orange

Kontakt

von	mit	dauer
3	5	90
5	9	20

MNS

name	farbe	tragt	pName
A	grau	3	Y
A	rot	11	X
B	orange	5	X
C	gelb	3	X
D	gruen	9	X
E	blau	11	X
F	violet	5	Y
G	rot	13	Y

Produzent

name	umsatz
X	14
Y	49