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

TEST 2 AUS DATENBANKSYSTEME (**184.686**)	**MUSTERLÖSUNG**	21.06.2023	A
Matrikelnr.	Familienname	Vorname	+Blätter

Arbeitszeit: 90 Minuten. Lösen Sie die Aufgaben auf den vorgesehenen Blättern und entfernen Sie nicht die Heftklammern. Lösungen auf Zusatzblättern werden nur gewertet, wenn sie bis spätestens zehn Minuten vor Ende des Tests von einer Aufsichtsperson bestätigt wurden. **Viel Erfolg!**

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 (11 Punkte)

	T_1	T_2	T_3	T_4	
1	b_1	b_2	b_3	b_4	1
2		$w_2(A)$			2
3			$r_3(B)$ X		3
4				$r_4(A)$ 2	4
5			$w_3(C)$		5
6		$w_2(D)$			6
7		$r_2(C)$ 5			7
8				$w_4(A)$	8
9		$w_2(C)$			9
10	$r_1(D)$ 6				10
11		$w_2(A)$			11
12		$abort_2$			12
13			$r_3(C)$ 5		13
14				$w_4(B)$	14
15	$r_1(B)$ 14				15
16	$r_1(A)$ 8				16
17				$r_4(D)$ X	17

Gegeben ist die links dargestellte Historie $\mathcal{H}$ der vier Transaktionen $T_1, \ldots, T_4$.

a) Tragen Sie bei den Leseoperationen die **Zeilennummer** ein, in welcher der gelesene Wert geschrieben wurde. Falls die Historie keine entsprechende Schreiboperation enthält, tragen Sie bitte ein `X` ein.

b) Geben Sie für jede Transaktion an, von welchen anderen Transaktionen sie liest.

T_1 liest von T_2, T_4 .....

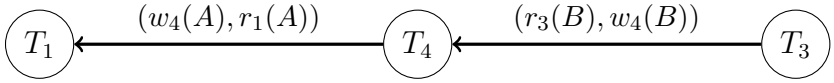
T_2 liest von T_3 .....

T_3 liest von (T_3) .....

T_4 liest von T_2 .....

c) Geben Sie den Serialisierbarkeitsgraph $SG(\mathcal{H})$ der Historie $\mathcal{H}$ an.
Geben Sie außerdem zu jeder Kante des Graphen ein Paar an Operationen an, auf Grund dessen die Kante Teil des Graphen ist. (Annotieren Sie dazu direkt die Kante.)

Serialisierbarkeitsgraph



d) Welche der Transaktionen T_1, T_3 und T_4 dürfen mit einem `commit` abschließen, damit die Historie rücksetzbar bleibt – und warum?

T_3. Für Rücksetzbarkeit dürfen Transaktionen ihr COMMIT erst haben, wenn alle Transaktionen von denen sie gelesen haben ihr COMMIT durchgeführt haben. T_3 ist die einzige Transaktion, die von keiner anderen Transaktion liest.

Achtung: Keine Punkte ohne Begründung.

Aufgabe 2: Protokollierung (Logging) (14 Punkte)

a) Sie finden nach einem Crash Ihrer Datenbank (mit Verlust des Datenbankpuffers, jedoch nicht des Hintergrundspeichers) die folgenden Log-Einträge vor. Im Hintergrundspeicher befinden sich außerdem die unten angegebenen Seiten. Führen Sie anhand dieser Informationen ein Recovery (nach der in der Vorlesung behandelten Variante des ARIES Verfahrens) durch.

Logeinträge (Archiv)

$[\#1, T_2,$ `BOT` $, \#0]$
$[\#2, T_2, P_A, \text{A+=3}, \text{A−=3}, \#1]$
$[\#3, T_1,$ `BOT` $, \#0]$
$[\#4, T_3,$ `BOT` $, \#0]$
$[\#5, T_1, P_B, \text{B+=1}, \text{B−=1}, \#3]$
$[\#6, T_3, P_C, \text{C+=5}, \text{C−=5}, \#4]$
$[\#7, T_1, P_A, \text{A−=2}, \text{A+=2}, \#5]$
$[\#8, T_2, P_A, \text{A+=5}, \text{A−=5}, \#2]$
$[\#9, T_1, P_B, \text{B+=7}, \text{B−=7}, \#7]$
$\langle\#10, T_2, P_A, \text{A−=5}, \#8, \#2\rangle$
$\langle\#11, T_1, P_B, \text{B−=7}, \#9, \#7\rangle$
$[\#12, T_3, P_C, \text{C+=1}, \text{C−=1}, \#6]$
$\langle\#13, T_1, P_A, \text{A+=2}, \#11, \#5\rangle$
$\langle\#14, T_2, P_A, \text{A−=3}, \#10, \#1\rangle$

Seiten im Hintergrundspeicher

P_A	LSN: #10
$A = 25$	

P_B	LSN: #11
$B = 92$	

P_C	LSN: #6
$C = 50$	

i) Bestimmen Sie die Werte für A, B und C nach der *Redo*-Phase.

A: `24` ..... B: `92` ..... C: `51` .....

ii) Führen Sie die *Undo*-Phase aus. Erzeugen Sie die **ersten vier** Compensation Log Records (CLRs), und geben Sie diese auf den untenstehenden Zeilen an. Verwenden Sie das am Beginn beschriebene Format.

1: $\langle$`#15,` T_3`,` P_C`,` C`-=1, #12, #6`$\rangle$ .....

2: $\langle$`#16,` T_3`,` P_C`,` C`-=5, #15, #4`$\rangle$ .....

3: $\langle$`#17,` T_1`,` P_B`,` B`-=1, #13, #3`$\rangle$ .....

4: $\langle$`#18,` T_3`, BOT, #16`$\rangle$ .....

b) (Die Aufgabe ist unabhängig von (a).)
In diesem Beispiel sind die Redo- und Undo- Einträge mittels *physischer Protokollierung* angegeben. Des weiteren wird als Auslagerungsstrategie die Kombination *force* und *no steal* verwendet. Unverändert bleibt, dass das *WAL-Prinzip* eingehalten wird.

Logeinträge

[#4, T_1, `BOT`, #0]
[#5, T_2, `BOT`, #0]
[#6, T_1, P_A, A=20, A=7, #4]
[#7, T_2, P_B, B=150, B=11, #5]
[#8, T_1, P_A, A=42, A=20, #6]
[#9, T_3, `BOT`, #0]
[#10, T_2, P_C, C=70, C=30, #7]
[#11, T_1, `COMMIT`, #8]
[#12, T_2, P_B, B=110, B=150, #10]
[#13, T_3, P_D, D=27, D=13, #9]
[#14, T_2, P_B, B=0, B=110, #12]

Es ist die oben dargestellte Liste an Logeinträgen gegeben. Nehmen Sie an, dass das protokollierte `commit` erfolgreich abgeschlossen wurde.

i) Bestimmen Sie, welche der Logeinträge sich in der dargestellten Situation in der persistenten Log-Datei befinden **müssen**, und welche noch ausschließlich im Log-Puffer stehen könnten.

Geben Sie die LSNs der jeweiligen Einträge **in der richtigen Reihenfolge** an.

Einträge zwingend in Log-Datei: `#4, #5, #6, #7, #8, #9, #10, #11` .....

Einträge möglicherweise in Log-Buffer: `#12, #13, #14` .....

ii) Betrachten Sie die Versionen der Seiten P_A, P_B, P_C und P_D **im Hintergrundspeicher**. Für welche dieser Seiten lassen sich die Werte für die LSN bzw. für A, B, C und D eindeutig bestimmen?

Wo die Werte eindeutig bestimmt sind, geben Sie diese Werte an. Wo dies nicht möglich ist, tragen Sie ein Fragezeichen `?` ein (leere Felder gelten als "keine Antwort").

Seiten im Hintergrundspeicher

P_A	LSN:	`#8`
$A =$	`42`	

P_B	LSN:	`?`
$B =$	`?/11`	

P_C	LSN:	`?`
$C =$	`?/30`	

P_D	LSN:	`?`
$D =$	`?/13`	

Aufgabe 3: Concurrency Control (12 Punkte)

a) Es ist die folgende Historie der Transaktionen T_1 und T_2 gegeben, wobei die Transaktionen mittels ("normalem"/striktem) 2PL synchronisiert werden.

T_1:	b_1 $X_1(A)$ $w_1(A)$ $S_1(B)$ $r_1(B)$ $X_1(D)$ $w_1(D)$ $r_1(D)$ $rel_1(A, B, D)$ c_1
T_2:	b_2 $S_2(B)$ $r_2(B)$ $X_2(C)$ $w_2(C)$ $S_2(D)$ $r_2(D)$ $rel_2(D)$ $r_2(C)$ $rel_2(C)$ $r_2(B)$ $rel_2(B)$ c_2

Kreuzen Sie *alle* untenstehenden Möglichkeiten an, welche es erlauben die gegebene Historie zu erzeugen.

1. ◯ T_1 und T_2 verwenden beide striktes 2PL.
2. ⊗ T_1 verwendet striktes 2PL; T_2 verwendet "normales" 2PL.
3. ◯ T_1 verwendet "normales" 2PL; T_2 verwendet striktes 2PL.
4. ⊗ T_1 und T_2 verwenden beide "normales" 2PL.
5. ◯ Keine der zuvor gelisteten Möglichkeiten.

Achtung: Punkte gibt es nur in Kombination mit einer Antwort auf die nächste Frage, nicht für Ankreuzen alleine:

Geben Sie für Ihre Auswahl mit der *niedrigsten* Nummer entsprechende Sperren und Freigaben an. Verwenden Sie $X_i(O)$ und $S_i(O)$ um das Sperren eines Datensatzes O mittels Exclusive bzw. Shared Lock durch die Transaktion T_i zu notieren, und $rel_i(O)$ um die Freigabe sämtlicher Sperren von Transaktion T_i auf O anzuzeigen. Sie **können** die Sperren und Freigaben **direkt in der Historie** annotieren. Achten Sie, besonders wenn Sie die Lösung woanders angeben, darauf, dass neben der Reihenfolge der Sperren und Freigaben auch klar erkennbar ist, wann die Lese- und Schreibzugriffe geschehen.

Falls Sie sich für Antwort 5. entschieden haben, geben Sie statt der Sperren eine kurze Begründung (**1–2** kurze Sätze!) an:

```
5. ist die falsche Antwort.
```

Hinweis: Das Feld kann leer bleiben, wenn Sie Punkt 5. nicht ausgewählt haben.

b) Wir betrachten Concurrency Control mit der in der Vorlesung vorgestellten Methode der zeitstempelbasierten Synchronisation.

Dazu sind in jeder Unteraufgabe die Lese- und Schreibzeitstempel zweier Datenobjekte A und B gegeben. Zusätzlich ist jeweils eine Operation einer Transaktion T_i sowie der Zeitstempel der Transaktion gegeben.

Entscheiden Sie, ob die Operation ausgeführt werden darf oder ob die Transaktion abgebrochen wird, und geben Sie in jedem Fall die Zeitstempel der Datensätze nach der Operation/dem Abbruch an.

i) T_1 mit Zeitstempel 4:

Operation: $w_1(B)$

Operation erlaubt: ⊗

Transaktion abgebrochen: ◯

	rTS(A)	wTS(A)	rTS(B)	wTS(B)
davor	5	2	4	3
danach	5	2	4	4

ii) T_2 mit Zeitstempel 6:

Operation: $r_2(B)$

Operation erlaubt ⊗

Transaktion abgebrochen: ◯

	rTS(A)	wTS(A)	rTS(B)	wTS(B)
davor	6	4	7	5
danach	6	4	7	5

iii) T_2 mit Zeitstempel 4:

Operation: $w_2(A)$

Operation erlaubt ◯

Transaktion abgebrochen: ⊗

	rTS(A)	wTS(A)	rTS(B)	wTS(B)
davor	5	3	2	6
danach	5	3	2	6

Achtung: Die einzelnen Unteraufgaben sind unabhängig voneinander zu betrachten!

Achtung: Pro richtiger Auswahl (erlaubt/abgebrochen) +1 Punkt, pro falscher Auswahl -1 Punkt, mindestens 0 Punkte auf Unteraufgabe b). Punkte für Zeitstempel nur bei richtiger Auswahl.

Diese Seite ist absichtlich leer. Aufgabe 4 beginnt auf der nächsten Seite.

Aufgabe 4: (6 Punkte)

Es sei das folgende Schema gegeben, welches eine Lieferkette beschreibt.

`item`	(id, description, value)
`company`	(name, industry, totalValue, highestValueId: *item.id*)
`owns`	(id: *item.id*, name: *company.name*, industry: *company.industry*)
`transform`	(old: *item.id*, new: *item.id*)

Ein `item` kann ein Gut, Produkt, Rohstoff, etc. sein, welches von einem Unternehmen verkauft oder weiterverarbeitet werden kann. Jedes `item` wird eindeutig durch eine ID (`id`) gekennzeichnet, hat eine Bezeichnung (`description`) und einen definierten Wert (`value`).
Für ein Unternehmen (`company`) wiederum ist der Namen (`name`) und die Bezeichnung der Industrie (`industry`), in der es tätig ist, notwendig für die eindeutige Identifikation. In `totalValue` ist der Wert (die Summe) aller `item`s gespeichert, welche von dem Unternehmen besessen werden. Außerdem referenziert `highestValueId` die ID des `item`, welches den größten Wert innerhalb dieses Unternehmen hat.
Welches Unternehmen welches `item` besitzt ist aus der Relation `owns` ersichtlich. Hierbei können mehrere Unternehmen gemeinsam ein `item` besitzen. Beachten Sie, dass `name` in Kombination mit `industry` nur zusammen und genau ein Unternehmen referenzieren.
Weiters können Unternehmen neue (`new`) `item` aus anderen (`old`) `item` herstellen (`transform`).
Ergänzen Sie die SQL Statements auf der nächsten Seite, um obiges Schema anzulegen. Wählen Sie passende Typen für jene Attribute, für die noch keine Datentypen vorgegeben sind, und definieren Sie die Primär- und Fremdschlüssel.
Setzten Sie außerdem die folgenden 3 Punkte um:

- Die ID eines `item` soll defaultmäßig mittels einer Sequence vergeben werden. Die Sequence beginnt bei 10 und läuft in 2er Schritten.
- `value` und `totalValue` sollen **nichtnegative** Dezimalzahlen mit 10 Vorkommastellen und 2 Nachkommastellen sein.
- Die Beschreibung (`description`) eines `item` darf nie `NULL` sein. *(Hinweis: Der leerer String '' ist in Ordnung.)*

```
DROP SEQUENCE IF EXISTS seq_id;
CREATE SEQUENCE seq_id INCREMENT BY 2 MINVALUE 10 NO CYCLE;

CREATE TABLE item(
   id          INTEGER DEFAULT nextval('seq_id') PRIMARY KEY,
   description  VARCHAR(100) NOT NULL,
   value       NUMERIC(12,2) NOT NULL CHECK (value >= 0)
);

CREATE TABLE company(
   name          VARCHAR(100),
   industry          VARCHAR(100),
   totalValue        NUMERIC(12,2) NOT NULL CHECK (totalValue >= 0),
   highestValueId    INTEGER REFERENCES item(id),
   PRIMARY KEY (name,industry)
);

CREATE TABLE owns(
   id          INTEGER REFERENCES item(id),
   name        VARCHAR(100),
   industry VARCHAR(100),
   FOREIGN KEY (name, industry) REFERENCES company(name, industry),
   PRIMARY KEY(id, name, industry)
);

CREATE TABLE transform(
   old  INTEGER REFERENCES item(id),
   new  INTEGER REFERENCES item(id),
   PRIMARY KEY(old, new)
);
```

Hinweis: Achten Sie bei den Statements auf die Reihenfolge.

Aufgabe 5: (10 Punkte)

Gegeben ist eine Rekursive Abfrage auf dem Datenbank-Schema von Aufgabe 4. Sie finden die Abfrage auf **Seite 15** (letztes Blatt der Prüfung).

a) Werten Sie diese Abfrage auf der auf der selben Seite dargestellten Datenbank-Instanz aus.

Berechnen Sie zuerst **schrittweise** die Tabelle `tmp`.
Geben Sie hierzu den Inhalt der *working tables* $\texttt{working}_1, \texttt{working}_2, \texttt{working}_3, \ldots$ (entsprechend der Definition für Rekursive SQL Abfragen) als Zwischenschritte an. Markieren Sie Einträge, wo $\texttt{working}_i$ leer ist oder gar nicht mehr berechnet wird, mit $\emptyset$.

Hierbei ist $\texttt{working}_1$ der Inhalt der *working table* während der ersten Auswertung des rekursiven Abschnitts (enthält also das Ergebnis der Auswertung des nicht rekursiven Abschnitts), und ansonsten $\texttt{working}_i$ der Inhalt während der i-ten Auswertung des rekursiven Abschnitts.

$\texttt{working}_1$:

id
16

$\texttt{working}_2$:

id
12
20

$\texttt{working}_3$:

id
10
14

$\texttt{working}_4$:

id

$\texttt{working}_5$:

id

Das Ergebnis der Abfrage ist:

id	description
14	cloth
20	paint

b) Geben Sie **ein** Tupel an, welches zur Tabelle `transform` hinzugefügt werden müsste, damit die Abfrage nicht terminiert. Wäre dies (die Nichttermination) weiterhin der Fall, wenn `UNION ALL` durch `UNION` ersetzt werden würde?

$(16, 12)$ und nein, es würde terminieren.

Aufgabe 6: (17 Punkte)

Betrachten Sie die Beispielinstanz auf **Seite 15** (letztes Blatt der Prüfung). Wir empfehlen das Blatt abzutrennen.

In Teilaufgaben (a) und (b) ist ein SQL Statement gegeben, das **über die Beispielinstanz** ausgeführt wird. Nehmen Sie jeweils nur den Bestand entsprechend des letzten Blattes an, d.h., Inserts in Aufgabe (a) haben keinen Einfluss auf Aufgabe (b) usw. Auch die Funktionen und Trigger beziehen sich nur auf die jeweiligen Teilaufgaben.

Geben Sie die Ausgabe der `SELECT`-Statements an. Falls ein Fehler auftreten würde, geben Sie an welcher Fehler auftritt. Es steht Ihnen frei, in Ihren Antworten auf diese beiden Aufgaben beliebige Kurzbezeichnungen zu verwenden, solange sich diese leicht eindeutig identifizieren lassen.

In Teilaufgabe (c) ist eine Stored Procedure zu schreiben, die das beschriebene Verhalten ausführt.

a)

```
1  CREATE OR REPLACE FUNCTION fInsertOwns() RETURNS TRIGGER AS $$
2  DECLARE
3      val INTEGER;
4  BEGIN
5      SELECT value INTO val
6      FROM item WHERE id = NEW.id;
7
8
9      UPDATE company SET totalValue = totalValue + val
10     WHERE NEW.name = name AND NEW.industry = industry;
11
12     RETURN NEW;
13 END;
14 $$ LANGUAGE plpgsql;
15
16 CREATE TRIGGER tInsertOwns AFTER INSERT ON owns
17     FOR EACH ROW EXECUTE PROCEDURE fInsertOwns();
```

```
1  INSERT INTO owns(id, name, industry)
2      VALUES (16, 'Geddes', 'Fabric Industry'),
3      (18, 'Corolles', 'Paint Industry'),
4      (14, 'Geppetto', 'Timber Industry'),
5      (12, 'Juan', 'Timber Industry');
6
7  SELECT name, totalValue, highestValueId FROM company;
```

```
name | totalvalue | highestvalueid
----------+------------+----------------
Geddes | 40.00 | 14
Corolles | 40.00 | 20
Geppetto | 70.00 | 16
Juan | 45.00 | 18
```

b)

```
1  CREATE OR REPLACE FUNCTION fUpdateNewItems() RETURNS TRIGGER AS $$
2  DECLARE
3      nid INTEGER;
4  BEGIN
5
6      IF NEW.id != OLD.id OR NEW.description != OLD.description THEN
7          RETURN NULL;
8      END IF;
9
10     IF NEW.value != OLD.value THEN
11         FOR nid IN (SELECT t.new FROM transform t
12                     WHERE NEW.id = t.old) LOOP
13             UPDATE item SET value = value + (NEW.value - OLD.value)
14                         WHERE id = nid;
15         END LOOP;
16     END IF;
17
18     RETURN NEW;
19     END;
20 $$ LANGUAGE plpgsql;
21
22 CREATE TRIGGER trUpdateNewItems BEFORE UPDATE ON item
23         FOR EACH ROW EXECUTE PROCEDURE fUpdateNewItems();
```

```
1  UPDATE item SET value = 15 WHERE id = 10;
2  UPDATE item SET description = 'red paint' WHERE id = 20;
3
4  SELECT * FROM item
5  WHERE value > 5;
```

```
id | description | value
----+-------------+-------
20 | paint | 10.00
16 | puppet | 40.00
18 | puppet | 30.00
12 | puppet | 25.00
10 | wood | 15.00
```

---

c)
Erstellen Sie eine Prozedur `fCreateCompany`, die automatisch ein neues Unternehmen (`company`) erstellt. Die Prozedur hat drei Argumente:
Die beiden Argumente *name* und *industry*, welche den `Namen` und die Branche (`industry`) des Unternehmens angeben, sowie ein Argument *descr*, das angibt, welche items dem Unternehmen gehören sollen.

Stellen Sie sicher, dass die folgenden Anforderungen erfüllt sind:

- Der `Name` und die Branche (`industry`) des erstellten Unternehmens erhalten die Werte der Argumente *name* und *industry*.
- Jedes item mit einer `description`, die dem Argument *descr* entspricht, ist Eigentum der erstellten Firma (beschrieben durch Einträge in der Relation `owns`).
- Die `highestValueId` des erstellten Unternehmens (`company`) wird auf die `id` des teuersten (größter `value`) items gesetzt, dessen `description` gleich dem Argument *descr* ist.
- Während und nach der Ausführung der Prozedur werden keine Fremdschlüssel-Bedingungen verletzt.
- Der `totalValue` des erstellten Unternehmens (`company`) soll auf 0 gesetzt werden.

```
CREATE OR REPLACE FUNCTION fCreateCompany(IN name VARCHAR(255), IN industry VARCHAR
    (255), IN descr VARCHAR(255)) RETURNS VOID AS $$
DECLARE
    rec RECORD;
    id INTEGER;
BEGIN
    SELECT i.id INTO id
    FROM item i
    WHERE i.description = descr AND i.value IN
    (SELECT MAX(value) FROM item j WHERE j.description = descr);

    INSERT INTO company(name, industry, totalValue, highestValueId)
    VALUES (name, industry, 0, id);

    FOR rec in (SELECT i.id FROM item i WHERE i.description = descr) LOOP
        INSERT INTO owns(id, name, industry)
        VALUES (rec.id, name, industry);
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

Gesamtpunkte: 70

Viel Erfolg!

Sie können diese Seite abtrennen und brauchen sie nicht abzugeben!

Auf diesem Zettel daher keine Lösungen notieren! Dieser Zettel wird nicht korrigiert! (Lösungen auf diesem Zettel werden nur gewertet, wenn er als Extrazettel bestätigt wurde!)

Die Datenbankinstanz für Aufgabe 5 und Aufgabe 6:

item

id	description	value
10	wood	10
12	puppet	20
14	cloth	5
16	puppet	35
18	puppet	25
20	paint	10
22	iron oxide	5

owns

id	name	industry
10	Geppetto	Timber Industry
12	Geppetto	Timber Industry
14	Geddes	Fabric Industry
16	Geppetto	Timber Industry
18	Juan	Timber Industry
20	Corolles	Paint Industry
22	Corolles	Paint Industry

transform

old	new
10	12
14	12
12	16
12	18
20	16
22	20

company

name	industry	totalValue	highestValueId
Geppetto	Timber Industry	65	16
Geddes	Fabric Industry	5	14
Juan	Timber Industry	25	18
Corolles	Paint Industry	15	20

Die rekursive Abfrage aus Aufgabe 5:

```
WITH RECURSIVE tmp(id) AS
(
 SELECT  i.id
 FROM    item i
 WHERE   i.id = 16
UNION ALL
 SELECT  tr.old
 FROM    tmp, owns o, transform tr
 WHERE   tmp.id = o.id
     AND o.name = 'Geppetto'
     AND o.industry = 'Timber Industry'
     AND tmp.id = tr.new
)
SELECT  tmp.id, i.description
FROM    tmp NATURAL JOIN owns o NATURAL JOIN item i
WHERE   o.name != 'Geppetto' OR o.industry != 'Timber Industry'
```