Gruppe **A**

Please fill in your name and registration number (Matrikelnr.) **immediately**.

EXAM IN ◯ DATA MODELLING 2 (**184.790**)	◯ DATA BASE SYSTEMS (**184.686**)	04.03.2022 **GROUP A**
Matrikelnr.	Last Name	First Name

Duration: 90 minutes. Provide the solutions at the designated pages; solutions on additional sheets of paper are not considered. **Good Luck!**

Attention!

To all questions with a multiple choice option, the following rule applies: Just checking an option gives no points; points are only granted in combination with the required justification/example/...

Notation:

In exercises 1 – 3, the following notation (as known from the lecture slides and exercises) for transactions T_i is used:

- $r_i(O)$ and $w_i(O)$: Read, respectively write operation of transaction T_i on object O.
- b_i, c_i, a_i: begin (`BEGIN OF TRANSACTION`), commit (`COMMIT`) and abort (`ABORT/ROLLBACK`) of T_i.

The indices $_i$ can be omitted if it is clear which transaction an operation belongs to.

In addition, log records also have the same format as used throughout the lecture:

`[LSN, TA, PageID, Redo, Undo, PrevLSN]` for "normal" records,
`[LSN, TA, BOT, PrevLSN]` for `BOT` log-records, and
`[LSN, TA, COMMIT, PrevLSN]` for `COMMIT` records.

Compensation log records (CLRs) follow the format
⟨`LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN`⟩ and
⟨`LSN, TA, BOT, PrevLSN`⟩

In these records, `LSN` denotes the Log-Sequence Number, `TA` the transaction, `PageID` the page that was updated, `Redo` and `Undo` the information needed for the Redo resp. Undo operations, `UndoNextLSN` the LSN of the next log record of the same transaction to be undone, and `PrevLSN` the LSN of the previous log record of the same transaction.

In case of logical logging, the changes to the previous value of the database instance are stated only using *addition* and *subtraction*, e.g. $[\cdot, \cdot, \cdot, X{+}{=}d_1, X{-}{=}d_2, \cdot]$.

Question 1: Properties of transactions (12)

Consider the schedule below, consisting of a sequence of basic operations of four transaction T_1, T_2, T_3 and T_4 on database objects A, B, C and D. *b_i denotes the begin, c_i the commit, and a_i the abort of transaction T_i. $r_i(O)$ and $w_i(O)$ denote the read, respectively write operations of transaction T_i on the object O.*

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

a) Specify the transactions between which there is a read dependency. This means, for each transaction specify from which other transactions this transaction reads (if a transaction does not read from any other transaction, please cross out the corresponding field).

(4 Points)

a) Read dependency:

T_1 reads from .... T_2 reads from ....

T_3 reads from .... T_4 reads from ....

b) Then determine whether the schedule avoids cascading reset or not, as well as whether it is strict or not. Provide a brief justification for each, based on the schedule.
(*Attention:* Checking an option without providing a justification gives no points!)

(4 Points)

b) Properties:

Schedule avoids cascading resets: ◯ yes ◯ no

Reason: ....

....

Schedule is strict: ◯ yes ◯ no

Reason: ....

....

c) Consider the pairs of transactions given below. For each of the two pairs, determine whether the two transactions are conflict serializable or not.
If not, provide a sequence of pairs of conflict operations of the two transactions which exclude the existence of a conflict equivalent, serial schedule.
If the transactions are conflict serializable, provide a conflict equivalent, serial history (= sequence of elementary operations!) of the two transactions.

(4 Points)

Transactions T_2 and T_4: conflict serializable: ◯ yes ◯ no

Answer: ....

....

Transactions T_1 and T_4: conflict serializable: ◯ yes ◯ no

Answer: ....

....

Question 2: Logging and Recovery (12)

a) Given a history (left) of three transactions T_1, T_2 und T_3. Assume that the database consists of the following values at the beginning of the history: (4 Points))

$$A = 70, \qquad B = 40, \qquad C = 20$$

Provide log-entries, which are created when executing the history. Use the usual format from lectures and exercises (reminder: we recall the format on page 1 of this exam). Entries for *begin of transaction* are already provided here.

ID	T_1	T_2	T_3
1	BOT		
2	$r(A, a_1)$		
3			BOT
4			$r(A, a_1)$
5		BOT	
6	$w(A, 50)$		
7	$r(B, b_1)$		
8		$r(B, b_2)$	
9	$w(B, a_1 + b_1)$		
10	COMMIT		
11		$w(B, b_2 + 5)$	
12			$w(A, a_1 + 20)$
13		$w(A, 40)$	
14			COMMIT

Give your solution here:
[#1,T_1,BOT,#0] ....
[#2,T_3,BOT,#0] ....
[#3,T_2,BOT,#0] ....
....
....
....
....
....
....
....

b) Does there occur a **Lost Update** in history of Question 2a? (3 Punkte)
If yes, state the lost update and explain why.
If no, state how the history can be modified/extended such that a lost update occurs.

Does there occur a lost update: ○ yes ○ no

Explain here:
....
....
....
....
....

c) On the left side below, log entries for transactions T_1 and T_2 and pages P_A and P_B are given are given. *Assume that the redo-phase has already been carried out successfully.* Your task is to execute *ONLY* the **undo**-phase.
Therefore, create the compensation log records (CLRs) and provide *values for A and B* after the undo-phase is done. (3 Points)

Log entries (description)

[#1,	T_1,	BOT,			#0]
[#2,	T_2,	BOT,			#0]
[#3,	T_1,	P_A,	A−=50,	A+=50,	#1]
[#4,	T_2,	P_A,	A+=0,	A−=0,	#2]
[#5,	T_2,	P_B,	B+=50,	B−=50,	#4]
[#6,	T_1,	P_A,	A+=25,	A−=25,	#3]
[#7,	T_2,	COMMIT,			#5]
⟨#8,	T_1,	P_A,	A−=25,	#6,	#3⟩

Pages on Background Memory

P_A	LSN: #8
$A = 25$	

P_B	LSN: #5
$B = 120$	

Log entries (solution)

. . . .

. . . .

. . . .

. . . .

A: . . . .

B: . . . .

d) Provide how choosing the strategy for outsourcing impacts recovery. (2 Points)
(0.5 Points each correct marked field; -0.5 Points for each wrong marked field, min. 0 Points):

Strategy: **force** and **steal**

Redo		Undo	
◯ yes	◯ no	◯ yes	◯ no

Strategy: ¬**force** and **steal**

Redo		Undo	
◯ yes	◯ no	◯ yes	◯ no

Question 3: Locking/Concurrency Control (11)

a) Below, a sequence of Exclusive- and Share Locks (XL(O), SL(O)), releases of locks (relXL(O), relSL(O)), as well as read- and write operations is given. For different entries within the same row, an arbitrary order may by assumed.

State for each of the five transactions T_1, T_2, T_3, T_4 and T_5, whether they follow the *2-Phase Locking (2PL)* protocol. If not, describe why 2PL is violated. (5 points)

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

Hint: Do not only look at each transaction in isolation.

b) Let T_6 and T_7 be two arbitrary transactions and let A be a data object with the initial value w. Furthermore, assume that T_6 wants to write the new value w' (with $w' \neq w$) into the data object A and that T_7 wants to read from A at an arbitrary point in time. The value stored in A is not altered otherwise.

Assume both transactions follow the 2-Phase Locking protocol. Does a scenario exist in which T_7 reads the new value w' from A and commits successfully if T_6 is aborted at some arbitrary point in time?

If yes, please state a sequence of operations that would lead to such a case.
If not, please briefly justify why such a situation cannot occur. (6 points)

○ yes ○ no

....

....

....

....

....

Please note that it is not sufficient to solely tick one of the choices to receive points for this question. We only grant points for the correct content of your answer.

Tasks 4–6 are all based on the database schema described on this page.

Question 4: Defining a database schema using SQL and dependencies (8)

The following schema is given

`star`	(<u>id</u>: *planet.star*, class, distance, firstPlanet: *planet.name*, galaxy: *galaxy.name*)
`planet`	(<u>star</u>: *star.id*, <u>name</u>, year, temp)
`galaxy`	(<u>name</u>, type, oldest: *star.id*, magnitude)
`generatedBy`	(<u>parent</u>: *star.id*, <u>child</u>: *star.id*, year)

An astronomer needs a database to catalogue her findings about stars, (exo)planets and galaxies.

A `star` is uniquely determined by its `id`. In addition to this, each star has a `class`.
<u>The class must be implemented as an enum, consisting of the values 'O', 'B', 'A', 'F', and 'G'</u>. The `distance` of the star is also saved. For every star, the first planet that was discovered is also part of the schema (`firstPlanet)`). Lastly, the galaxy to which the star belongs to is noted in the schema as well.

A `planet` is uniquely determined by the combination of its star and `name`. In addition to this, the year in which the star was discovered by the astronomer is saved too (`year`). The average temperature (`temp`) of the planet surface is also part of the schema. <u>The temperature must not be negative, since it is stated in Kelvin.</u>

Every `galaxy` has a unique `name`. The `type` of the galaxy is saved, as is the oldest star (`oldest`) occurring in it. Finally, the recorded brightness of the galaxy (`magnitude`) is part of the schema too.
<u>The value of the brightness must be divisible by 100, as it cannot be measured more precisely</u>

The astronomer also wants to record a kind of genalogy of stars (`generatedBy`). It is saved which stars were created as part of another star going nova (`parent`, `child`), this combination must be unique. The year of creation (`year`) is saved as well.

Provide the necessary SQL statements to create the described schema, with all described integrity constraints. Choose appropriate types for attributes. **You may use VC in place of VARCHAR(100).**

Hint: Take care of the order of your statements.

Question 5: Recursive Queries (15)

a) Create a view based on the schema of Question 4.

This view shall contain the IDs of all 'hot star systems', defined as all stars which have two planets with a temperature of over 1000 Kelvin.

Provide the necessary SQL statements to create a view satisfying the above conditions.

(3 points)

b) Create the following recursive SQL query:

Based on the schema of Question 4, implement the following query:

The table `generatedBy` defines which stars were produced through other stars. We shall define as *successor* of a star A, all other stars which were produced via one or more generation steps from star A. A $\mathcal{HS}$-successor (hot star system successor) is a successor of a star that is also a hot star system, meaning it has at least 2 planets with a temperature of over 1000 Kelvin.

The transitive hull of $\mathcal{HS}$-successors of a star A are all $\mathcal{HS}$-successors of A, all $\mathcal{HS}$-successors of $\mathcal{HS}$-successors of star A, and all $\mathcal{HS}$-successors of $\mathcal{HS}$-successors of $\mathcal{HS}$-successor of A, and so forth.

Now state the transitive hull of $\mathcal{HS}$-successors of the star with the id '4'.

You do not have remove duplicates in the output and you may assume there are no cycles within the table `generatedBy`.

Your output should contain the ids of all stars in this transitive hull.

State the required SQL statements necessary to implement this recursive query.

You are encouraged to reuse the view from the previous subtask a) in your query!

(12 points)

```
WITH RECURSIVE hs_descendants(                          ) AS
(



      UNION ALL




)
```

Note: Make sure the query terminates and produces an output.

Question 6: PL/SQL Trigger (12)

a)

(6 points) Define triggers and the associated PL/pgSQL function(s) in order to implement the following case:

When an entry into `generatedBy` is made, it is checked whether the attribute `year`, i.e. the year of creation of the planet, of the predecessor-planet is before that of the successor-planet. If the year of creation of the predecessor-planet is not known, the INSERT is also valid and should succeed.

When the year of creation of the successor-planet is earlier than that of the predecessor-planet, nothing will be inserted and the error message 'year of child before that of parent' is shown.

The behavior is shown by the following example: In the relation `star` there are three stars with `id` 1, 2 and 3. In the relation `generatedBy`, there is an entry `(1,2,1000)`.

In case that a tuple `(2,3,1001)` is inserted into `generatedBy`, this is successful.

In case that a tuple `(2,3,999)` is inserted into `generatedBy`, this fails with the error message 'year of child before that of parent.

```
CREATE FUNCTION checkYear() RETURNS TRIGGER AS $$



BEGIN











END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trGeneratedByYear BEFORE INSERT ON generatedBy
       FOR EACH ROW EXECUTE PROCEDURE checkYear();
```

b)

(6 points)

Define triggers and the associated PL/pgSQL function(s) in order to implement the following case:

When the first planet (`firstPlanet`) of a star is deleted, the reference of the star to this planet is changed to reference a remaining star with minimal date of discovery (`year`).

It can be assumed that, for each deleted star, there is at least one other star for the same planet. Hence, the case that no planed remains after deletion does not need to be considered.

```
CREATE FUNCTION onPlanetDelete() RETURNS TRIGGER AS $$



BEGIN

















END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trPlanetDelete BEFORE DELETE ON generatedBy
       FOR EACH ROW EXECUTE PROCEDURE onPlanetDelete();
```

Overall: 70 points

Good Luck!