Gruppe **A**

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

EXAM AUS		17.09.2020
◯ DATENMODELLIERUNG 2 (**184.790**)	◯ DATENBANKSYSTEME (**184.686**)	**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. [·, ·, ·, X+=d_1, X-=d_2, ·].

Exercise 1: Properties of transactions (12)

Assume that a DBMS implements the following isolation-levels, and each transaction is free to choose in which isolation-level it executes.

- **L1:** Share-Locks are requested immediately before- and released immediately after each read operation (no 2PL); 2PL for Exclusive-Locks (independent of the releases of Share-Locks).
- **L2:** combined 2PL for Share- and Exclusive-Locks, for Exclusive-Locks strict 2PL is used. Lock-Upgrades (of a Share-Lock to an Exclusive-Lock) are possible and do not count as a lock release.

Please answer the following questions with "yes" or "no". When answering *"yes"*, state a schedule (operations b_i, c_i, $r_i(O)$, $w_i(O)$) that is consistent with the stated isolation-level and satisfies the given property. When answering *"no"*, provide a short (1-2 sentences) justification.

Note: You need not state lock requests and releases of locks – just make sure that for your schedule a corresponding valid such sequence exists.

a) Consider the transaction

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

and an arbitrary transaction T_2. Assume that T_1 runs in level L1 and T_2 in level L2.

i) Is it possible that a *Lost-Update* happens in T_2?

Lost-Update in T_2: ◯ yes ◯ no

.....

.....

.....

ii) Is it possible that a *Dirty Read* happens in T_2?

Dirty Read in T_2: ◯ yes ◯ no

.....

.....

.....

b) In addition to T_1 from a), consider the transaction

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

running in isolation level L2.

Is there a valid schedule of these two transactions that does **not** avoid cascading abort?

Schedule that does **not** avoid cascading abort: ◯ yes ◯ no

.....

.....

.....

Exercise 2: Logging and Recovery (12)

Consider the log records of the four transactions T_1, T_2, T_3 and T_4 given below.

a) Perform a recovery following the ARIES algorithm using these log records, and state the log records created during the recovery.

Log records (given)

[#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]

Log records (solution)

.....

.....

.....

.....

.....

.....

.....

.....

b) Based on the given log records, is it possible to determine whether the corresponding schedule was strict (in the sense of the classification of schedules for concurrency control)?
If yes, state whether the corresponding schedule was strict or not (and why).
If no, briefly (1-2 short sentences) justify your answer.

Strict can be decided: ◯ yes ◯ no

.....

.....

.....

c) Assume the replacement strategy used was *force* and *steal*.

For the two pages P_B and P_C, determine as precisely as possible the value of the LSN recorded on the page (LSN of the last log record that recorded a change on the page), that is present at the start of the recovery (i.e. the value that was stored on the page in the persistent memory that was used to perform the recovery).

Page	**lower bound**		**upper bound**
P_B	.....	$\leq$ LSN $\leq$	.....
P_C	.....	$\leq$ LSN $\leq$	.....

Assume that the LSN of both pages initially, i.e. before the log record with LSN #1, was `#0`.

Exercise 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.

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	

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

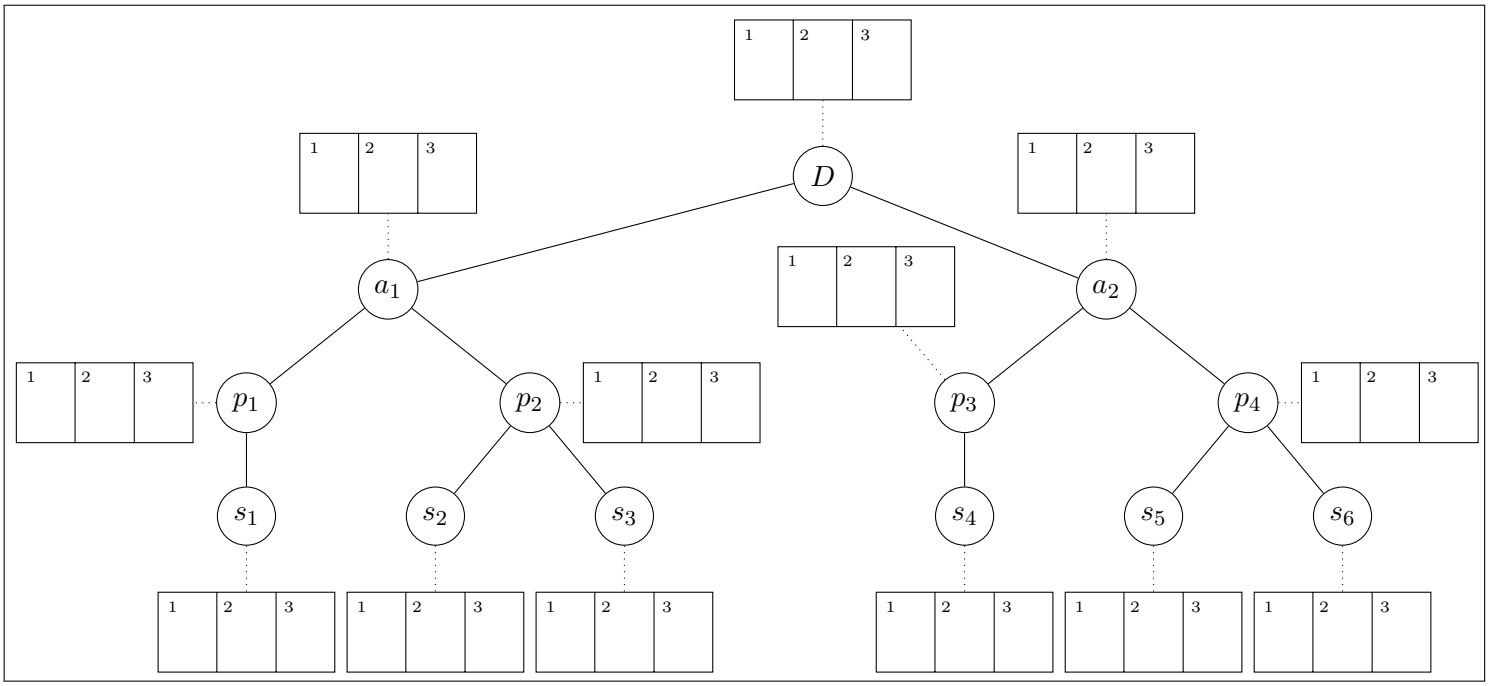
.....

.....

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

b) Multi Granularity Locking:

The following organizational hierarchy of a database is given.



Assume the transactions T_1, T_2 and T_3 want to perform the following operations in the given order:

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

Assume further that the transactions use the MGL protocol for getting the required locks (and apply MGL correctly).

Describe the situation after working through these operations by filling into the above figure which transactions hold which locks at each node. At each node, put S, X, IS and IX into the field with the number of the transaction to indicate that the transaction holds the corresponding lock on that node. If a transaction requests a lock but does not get it (i.e. has to wait), please put WS, WX, WIS or WIX into the corresponding field. If a transaction is blocked, then ignore all further actions of this transaction.

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

Exercise 4: Defining a database schema using SQL and dependencies (11)

a) The following schema is given

`employee`	(id, favoriteN: *fmask.name*, favoriteC: *fmask.color*)
`manufacturer`	(name, sales)
`fmask`	(name, color, worn: *employee.id*, mName: *manufacturer.name*)
`contact`	(from: *employee.id*, with: *employee.id*, duration)

Because of Corona-regulations a company has to make wearing face masks mandatory. To track this, in a database, each employee is uniquely identified by an ID. The ID is implemented as a sequence that starts at 3 and is increased by steps of 2. For better tracking, each face mask is assigned a name, with the combination of the name and its color being unique for each mask. Each face mask is assigned to exactly one employee. Each employee also has a favorite mask. For each face mask, the manufacturer is recorded. Each manufacturer has a unique name and sales. Due to legal glitches, regulations require the sales to be a multiple of 7. This is implemented using a Check. Finally the table `contact` stores, which employee (`from`) was in direct contact with which other employees (`with`), and how long (as hours) the contact was.

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.

b) For this task, you have to consider different key-dependencies.

i) Consider the following schema:

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

Complete the following tables in such a way **that the resulting database instance satisfies the schema.** You must assume that the two tables represent the complete database. To get points, the tables must be filled completely.

building

id	**biggestRoom**
	Audimax
3	
4	
	InfHS

room

id	**name**	**capacity**
	Audimax	400
1		
2		
	InfHS	

ii) Consider the following schema:

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

Complete the following tables in such a way **that the resulting database instance does not satisfy the schema.** You must assume that the two tables represent the complete database. To get points, the tables must be filled completely.

building

id	**name**
1	
	Hörsaal 1
	Hörsaal 2
-7	

passage

from	**to**
Audimax	
	Hörsaal 1
Verschollener Hörsaal	

Exercise 5: Recursive Queries (12)

a) Create the following recursive SQL query:

Each row `(from, with, duration)` in the table `contact` describes a *direct contact* between the two employees `from` and `with`. Define employee A as a *contact person* for an employee B if there either was a direct contact, or there was a direct contact between A and an employee C who is a (direct) contact person of B (and so on).
Attention: Do not extend the relation symmetrically, but keep the order (`from`, `with`) (i.e., assume contacts to be directed).

A *one hour contact* denotes all contact with a duration of at least one hour. A *one hour contact person* is either a direct contact person where the contact took at least one hour, or a contact person where the contact in each "step" took at least one hour. *Two, Three, ... hour contact person* are defined analogously.

Output all two hour contact persons of the employee with ID "1". However, for caution the *direct contacts* of this employee shall be returned already if the duration of the contact was at least one hour. These additional direct contacts are then also included in the recursion, i.e. two hour contact persons of these one hour contact persons of employee "1" shall also be returned.

The result of the query shall contain the ID and the favorite face mask of the employee. Employee that are connected over more than one "path" as a two hour contact person are supposed to occur a corresponding number of times in the list.

Provide the SQL statements required to implement the query described above.

Note: Make sure the query terminates. You may assume that the database does not contain any cyclic contacts.

Exercise 6: PL/SQL Trigger (12)

Assume the **functions and trigges** are defined as stated on **page T** (towards the end of this exam). The following tasks are based on the database instance shown on **page B** (last page of the exam).

Each of the following tasks consists of a SQL statement which is executed every time **on the original database instance** shown own page B. (This means, that the statement in task a) has no effect for the solution of task b), and so forth.) Provide the results of the `SELECT`-statements. **In case an error would occur, state what this error is.**

You are free to use arbitrary shorthand notations in your answers (as long as they can be uniquely identified).

a)

```
UPDATE fmask set worn='7' WHERE name='A';

SELECT * FROM fmask WHERE name='A';
SELECT * FROM contact;
```

---

b)

```
UPDATE fmask SET worn='3' WHERE name='F';

SELECT * FROM fmask WHERE name='F';
SELECT * FROM contact;
```

c)

```
BEGIN;
  INSERT INTO contact VALUES (5, 7, 100);
  UPDATE fmask SET worn='3', color='yellow' WHERE pName='Y';
COMMIT;

SELECT * FROM fmask WHERE pName='Y';
SELECT * FROM contact;
```

Overall: 70 points

Good Luck!

You may separate this page form the exam and keep this page.

Thus, please do not provide any solutions on this page! Solutions written on this sheet will not be graded!

Trigger for Task 6:

```
CREATE FUNCTION maskContact() RETURNS TRIGGER AS $$
BEGIN
    IF OLD.worn = NEW.worn THEN
       RETURN NULL;
    END IF;
    IF NEW.color != 'red' THEN
        INSERT INTO contact VALUES (OLD.worn, NEW.worn, 3);
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;


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




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


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

Sample instance for Task 6:

You may separate this page form the exam and keep this page.

Thus, please do not provide any solutions on this page! Solutions written on this sheet will not be graded!

employee

id	favoriteN	favoriteC
3	A	red
5	E	blue
7	E	blue
9	C	yellow
11	A	red
13	B	orange

contact

from	with	duration
3	5	90
5	9	20

fmask

name	color	worn	mName
A	gray	3	Y
A	red	11	X
B	orange	5	X
C	yellow	3	X
D	green	9	X
E	blue	11	X
F	violet	5	Y
G	red	13	Y

manufacturer

name	sales
X	14
Y	49