Gruppe **A**

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

EXAM AUS		19.06.2020
◯ 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.

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.

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, and `PrevLSN` the LSN of the previous log record of the same transaction.

Exercise 1: Properties of transactions (11)

Consider the following recoverable schedule consisting of six transactions $T_1, \dots, T_6$.

	T_1	T_2	T_3	T_4	T_5	T_6	
1	b_1	b_2	b_3	b_4	b_5	b_6	1
2	$r_1(A)$ $\boxed{}$						2
3	$r_1(B)$ $\boxed{}$						3
4			$r_3(A)$ $\boxed{}$				4
5	$w_1(A)$						5
6				$r_4(A)$ $\boxed{}$			6
7		$w_2(C)$					7
8						$r_6(B)$ $\boxed{}$	8
9						$w_6(B)$	9
10				$w_4(A)$			10
11		$r_2(B)$ $\boxed{}$					11
12		$w_2(A)$					12
13						$w_6(C)$	13
14				$r_4(C)$ $\boxed{}$			14
15			$r_3(A)$ $\boxed{}$				15
16				$w_4(C)$			16
17					$r_5(C)$ $\boxed{}$		17
18			$r_3(B)$ $\boxed{}$				18
19	a_1						19

a) At each read operation, insert the line number in which the value read by the operation was written. If the corresponding write is not part of the schedule, please insert `X` for the line number.

b) For each transaction, state the transactions it reads from.

T_1 reads from ....

T_2 reads from ....

T_3 reads from ....

T_4 reads from ....

T_5 reads from ....

T_6 reads from ....

c) Continue the schedule in such a way, that all transactions finish (each transaction may either finish with a_i or c_i). The resulting schedule must remain recoverable, and as many transaction as possible shall finish with c_i.
For this, state for the next 5 steps the corresponding operations (c_i or a_i for $2 \leq i \leq 6$), and very, very briefly justify your choice (justification need not be a complete sentence).

step	**operation**	**justification**
20	....	....
21	....	....
22	....	....
23	....	....
24	....	....

Exercise 2: Logging and Recovery (12)

a) Provide a valid sequence of log-records, satisfying the following properties:

- The log-records shall be created by two transactions T_1 and T_2.
- For both transaction, the `BEGIN OF TRANSACTION` records must be included.
- T_1 must have finished by `COMMIT`, T_2 must still be active/running.
- Undo and Redo information must be recorded using *physical* logging.
- The log-records must be drafted in such a way that a recovery using the ARIES algorithm does **not** produce the same result as if T_2 would have never happened.
- You may use the fields A, B, C, … located on the pages P_A, P_B, P_C, …. Assume that initially all fields store the value 0.

....

....

....

....

b) Independent of a), the following content of the persistent log-file and the volatile (non-persistent) log-buffer are given.

Log-Datei	**Log-Puffer**
[#1, T_1, `BOT`, #0]	[#7, T_3, P_E, ·, ·, #3]
[#2, T_2, `BOT`, #0]	[#8, T_3, P_A, ·, ·, #7]
[#3, T_3, `BOT`, #0]	[#9, T_1, P_C, ·, ·, #6]
[#4, T_1, P_A, ·, ·, #1]	[#10, T_3, `COMMIT`, #8]
[#5, T_2, P_C, ·, ·, #2]	[#11, T_2, P_E, ·, ·, #5]
[#6, T_1, P_E, ·, ·, #4]	

The assumed configuration is *no steal (¬steal)* and *not force (¬force)*, and WAL is followed.

i) Is it possible that a user has been already informed on the successful commit of T_3?

○ yes ○ no: ....

....

ii) Describe a series of actions (updates on log-buffer and log-file, potential new log-records or necessary writing of pages to persistent storage), that allows the page P_A to be moved to persistent storage.

....

....

....

....

Exercise 3: Locking/Concurrency Control (12)

A research institute supervises six experiments A, B, C, D, E, F, which are shared between three projects P_1, P_2, and P_3. Each project can read the values of an experiment E_i ($r(E_i)$), and change the parameters of an experiment E_i ($w(E_i)$). The end of a project is denoted by c.

To make sure, that there is no influence between the projects, access to the experiments is synchronized using Two-Phase Locking with wait-die: Reading requires a shared lock, changes an exclusive lock. Locks are requested as late as possible and released as early as possible. As timestamps, the year when the project was started is used.

a) Consider the following operations on the experiments by the projects:

$P_1(2019)$	b		$w(D)$	$r(A)$			$w(B)$			$w(E)$	$r(D)$	b		$r(D)$	c			
$P_2(2020)$	b	$w(C)$							$r(B)$			o t	$w(E)$				c	
$P_3(2017)$	b				$w(F)$	$w(A)$		$w(E)$				*?*				$r(D)$		c

Due to the locking protocol, actual access to the experiments will not be possible in the given order. Provide the actual order in which the projects access the experiments. When reaching the entry *bot?* for the first time, one project is to be restarted, in case some project had to be reset due to *wait-die* up to this point. If a project has to wait on a lock, all "skipped" operations are caught up once the lock has been granted.

	P_1	P_2	P_3
1		$w(C)$	
2	$w(D)$		
3	$r(A)$		
4			$w(F)$
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			

b) Independent of the concrete schedule given in a), is it ever possible that two projects P and Q both read the values of an experiment changed by the other project without violating the synchronization protocol?
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.

○ ja ○ nein

....

....

....

....

....

....

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

Exercise 4: Defining a database schema using SQL (7)

The following schema is given

instructors	(name, type, teaches: *webinar.id*)
webinar	(id, title, leadName: *instructors.name*, leadType: *instructors.type*)
prerequisites	(before: *webinar.id*, after: *webinar.id*)
participants	(ident, favorite: *webinar.id*)

Each instructor is uniquely identified by name and type. The type must be an Enum, consisting of the values “Univ-Ass”, “Ass-Prof”, and “Univ-Prof”. In addition, for each instructor one webinar she teaches is stored explicitely (`teaches`). Each webinar has a unique ID. Additionaly, it also has a title. For each webinar, one instructor is marked as the leading instructor. As the number 13 is considered to bring bad luck, no webinars must exist with an ID of 13 or a multiple of 13. The relation `prerequisites` stores which webinars (`after`) require which other webinars (`before`). The combination of predecessor and successor is unique. Finally, also the participants, which have a unique identification, are stored. All participant choose one webinar as their favorite.

Provide the necessary SQL commands to create database tables according to the provided schema. Make sure to implement all of the described integrity constraints. Choose appropriate attributes for the columns. You may use VC as an abbreviation for `VARCHAR(100)`.

Hint: Take care of the order of your statements.

Exercise 5: Recursive Queries (14)

a) Create a view `popular_webinar` based on the following query:

We look for the `IDs` of all "popular" webinars. These are webinars which are marked as a favorite from at least 10 participants.

State the necessary SQL statements to create this view.

b) Create the following recursive SQL query:

Each row `(before, after)` in the table `prerequisites` describes a *immediate prerequisite* between webinars (`before` is a direct prerequisite of `after`). We define a webinar to be a *requirement* if it is either a immediate prerequisite, or the immediate prerequisite of an immediate prerequisite, and so on. We denote as $\mathcal{PW}$ (popular webinar) all webinars that are the favorite of at least 10 participants. A "$\mathcal{PW}$-requirement" is a webinar that is both, a requirement and a $\mathcal{PW}$.

We look for all $\mathcal{PW}$-requirements that are lead by an instructor with the name "Diotima" and of type "Univ-Prof". Note that these webinars **need not be popular webinars themselves**.

The query shall return the ID and the title of the requested webinars. Webinars that satisfy these conditions over different "paths" shall occur with the corresponding multiplicity in the result.

Formulate this query in SQL. *It is a good idea to reuse the view from the previous example.*

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

Exercise 6: PL/SQL Trigger (14)

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 `DELETE` statement, which is executed every time **on the original database instance** shown own page B. (This means, that the `DELETE` in task a) has no effect for the solution of task b), and so forth.) Provide the results of the `SELECT`-statements after the `DELETE` has been executed.

In case an error would occur, state what this error is.

a)

```
DELETE FROM webinar WHERE id=6;

SELECT id FROM webinar;
SELECT * FROM prerequisites;
```

b)

```
DELETE FROM webinar WHERE id=1;

SELECT id FROM webinar;
SELECT * FROM prerequisites;
```

c)

```
DELETE FROM prerequisites WHERE after=4 OR after=3;

SELECT id FROM webinar;
SELECT * FROM prerequisites;
```

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 delW() RETURNS TRIGGER AS $$
BEGIN
    IF OLD.leitTyp = 'Univ-Prof' THEN
       RETURN NULL;
    END IF;
    DELETE FROM Bedingungen WHERE vor = OLD.id;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trDw BEFORE delete ON webinar
       FOR EACH ROW EXECUTE PROCEDURE delW();
```

```
CREATE FUNCTION delB() RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT count(*) FROM webinar w
          JOIN teilnehmende t ON w.ID = t.favorit
   WHERE w.id = OLD.nach) < 3
    THEN
      DELETE FROM webinar where ID = OLD.nach;
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trDB after delete ON Bedingungen
       FOR EACH ROW EXECUTE PROCEDURE delB();
```

Sample instance for Task 6:

Webinar

id	titel	leadName	leadTyp
1	Kreise und Bäume	A	Univ-Ass
2	HYPERgraphen	B	Ass-Prof
3	Digital Humanism	C	Univ-Prof
4	Dreiecke und Du	D	Ass-Prof
5	Epistemologie der Moral	C	Univ-Prof
6	Influencen	A	Univ-Ass
7	Unlogik	C	Univ-Ass
8	Logik	D	Ass-Prof

Prerequisites

before	after
1	2
1	3
3	4
2	4
4	5
6	7
7	8

Instructors

name	type	teaches
A	Univ-Ass	8
B	Ass-Prof	2
C	Univ-Prof	3
D	Ass-Prof	4
C	Univ-Ass	7

Participants

ident	favorite
123	8
789	8
032	8
042	7
043	3
456	1
045	1
056	1