Gruppe **A**

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

EXAM IN	**SAMPLE SOLUTION**	21.01.2021
◯ DATA MODELLING 2 (**184.790**)	◯ DATA BASE SYSTEMS (**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, ·].

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
			$r_4(A)$
		$r_3(A)$	
	$r_2(C)$		
$r_1(D)$			
			$w_4(A)$
		$r_3(B)$	
	$w_2(B)$		
		$w_3(C)$	
$w_1(B)$			
			$w_4(D)$
			c_4
	$r_2(A)$		
	$w_2(C)$		
$r_1(A)$			
		$r_3(D)$	
	c_2		
c_1			
		$w_3(C)$	
		$r_3(B)$	
		$w_3(B)$	
		c_3	

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_4 ..... T_2 reads from T_4 .....

T_3 reads from T_1, T_4 ... 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: All transactions that are read from .....

have their commit before the respective read operation.

Schedule is strict: ◯ yes ⊗ no

Reason: for instance, $w_2(C)$ overwrites the value of

$w_3(C)$, without a commit of T_3 before that. .....

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_3: conflict serializable: ◯ yes ⊗ no

Answer: several possible solutions, e.g. $(r_3(B), w_2(B))$, $(w_2(B), w_3(B))$, or .....

$(r_2(C), w_3(C))$, $(w_3(C), w_2(C))$, or $(w_3(C), w_2(C))$, $(w_2(C), w_3(C))$ .....

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

Answer: b_4, $r_4(A)$, $w_4(A)$, $w_4(D)$, c_4, b_2, $r_2(C)$, $w_2(B)$, $r_2(A)$, $w_2(C)$, c_2 .....

.....

Question 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. (6 Points)

Log entries (description)

[#1, T_1, BOT,			#0]
[#2, T_2, BOT,			#0]
[#3, T_3, BOT,			#0]
[#4, T_2, P_C,	C–=6,	C+=6,	#2]
[#5, T_4, BOT,			#0]
[#6, T_2, P_A,	A+=15,	A–=15,	#4]
[#7, T_4, P_D,	D+=17,	D–=17,	#5]
[#8, T_3, P_C,	C+=3,	C–=3,	#3]
[#9, T_4, P_B,	B–=2,	B+=2,	#7]
[#10, T_2, COMMIT,			#6]
⟨#11, T_4, P_B,	B+=2,	#9,	#7⟩
⟨#12, T_4, P_D,	D–=17,	#11,	#5⟩
[#13, T_1, P_C,	C+=1,	C–=1,	#1]
[#14, T_3, P_A,	A+=8,	A–=8,	#8]
[#15, T_1, P_A,	A+=6,	A–=6,	#13]

Log entries (solution)

⟨#16, T_1, P_A, A-=6, #15, #13⟩ .....

⟨#17, T_3, P_A, A-=8, #14, #8⟩ .....

⟨#18, T_1, P_C, C-=1, #16, #1⟩ .....

⟨#19, T_3, P_C, C-=3, #17, #3⟩ .....

⟨#20, T_4, BOT,#12⟩ .....

⟨#21, T_3, BOT, #19⟩ .....

⟨#22, T_1, BOT, #18⟩ .....

.....

.....

.....

4 CLR	+1
3 BOT CLR	+1
LSN korrekt	+0.5
Transaktionen korrekt	+0.5
Pages korrekt	+0.5
Undo korrekt	+0.5
PrevLSN korrekt	+1
UndoNextLSN korrekt	+1
bis 2 Typos	-0
bis 2 Typos	-0
PrevLSN nur aus Log	-0.5

$\sum$ +6

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)? (4 Points)
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 recognized: ⊗ yes ◯ no

Nicht strikt: T_1 überschreibt in Logeintrag #13 den Wert von C, ...

welchen T_3 im Eintrag #8 geschrieben hat. Da T_3 in der .....

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

Alternative .....

Nicht strikt: T_1 überschreibt in Logeintrag #15 den Wert von A, ...

welchen T_3 im Eintrag #14 geschrieben hat. Da T_3 in der .....

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

Ja, kann erkannt werden	+1
Nicht strikt	+1
Begründung	+2
Detail Begründung: pro Eintag	+0.5
Detail Begründung: pro Transaktion	+0.5
∑	+4

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). (2 Points)

page	**lower bound**		**upper bound**
P_A	#6	≤ LSN ≤	#15 ...
P_C	#4	≤ LSN ≤	#13 ...

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

pro richtigem Wert	+0.5
Geschlossene Transaktion bei beiden UB nicht erkannt	-0.5
∑	+2

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(E)			
	SL(B)		XL(D)	
	$r(B)$			SL(B)
				SL(A)
	XL(C)		SL(E)	$r(B)$
	$w(E)$		$r(D)$	relSL(B)
$r(B)$	$r(C)$		$r(E)$	
	relSL(E)			$r(A)$
		XL(A)	relSL(E)	
	$w(C)$	$w(A)$		
	relSL(B)		relXL(D)	XL(E)
	relXL(C)			$w(E)$
		SL(C)		
XL(D)	c_2	$r(C)$		relSL(A)
$w(D)$				
relSL(B)		relXL(A)	c_4	
relXL(D)		relSL(C)		relXL(E)
c_1		c_3		c_5

T_1: correct 2PL. .....

T_2 writes on E although .....

it has solely a shared lock. .....

T_3 writes on A although T_5 .....

still holds a shared lock (SL(A)). .....

T_4: correct 2PL. .....

T_5 locks E (XL(E)), after .....

releasing lock SL(B). .....

.....

.....

.....

.....

.....

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

b) Let T_1 and T_2 be two arbitrary transactions and let A be a data object with the initial value w. Furthermore, assume that T_1 wants to write the new value w' (with $w' \neq w$) into the data object A and that T_2 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 strict 2-Phase Locking protocol. Does a scenario exist in which T_2 reads the new value w' from A and commits successfully if T_1 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

The protocol guarantees strict schedules. .....

The described behaviour corresponds to non-resettable schedules. .....

Resettable schedules are a subset of strict schedules. .....

The strict 2PL guarantees that the resulting schedule is strict .....

and therefore resettable. Thus it is impossible .....

that a transaction that has been commited successfully .....

has read a value that was written by an aborted transcation. .....

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 (11)

a) The following schema is given

`customer`	(name, taxID, country, mostExpensive: *orders.id*)
`orders`	(id, date, fromName: *customer.name*, fromTaxID: *customer.taxID*)
`part`	(id: *orders.id*, number, cost, fromNAME: *supplier.name*)
`supplier`	(name, revenue, country)

A university assistant needs to find a relational schema for a question in a data base systems test and, in the absence of new ideas, decides upon a generic schema consisting of customers, suppliers, orders and parts.

A customer is uniquely identified by `name` and `taxID`. In addition, the `country` of the customer is also saved.

Each of the orders is uniquely identified by an `id`. The ID must be a sequence, starting with 100 and continuing in steps of 300. The `date` on which the order was placed is also recorded. The customer who sent the order is also saved. In addition, the `mostExpensive` order to date is recorded for the respective customer.

There are a number of parts for every order. Each part is uniquely identified by the combination of the `id` of the associated order and a `number`. Each item also has a `cost` and the supplier of the part is saved (`from`). The `cost` must not be negative. `cost` should be implemented as a decimal number with at most 2 decimal places.

Each supplier has a unique `name`. In addition, the `revenue` and the `country` of the supplier are recorded in the database. The `revenue` must lie between 1,000 and 1,000,000 for each entry in the table.

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).**

(7 points)

```
CREATE SEQUENCE seq_id INCREMENT BY 300 MINVALUE 100 NO CYCLE;

CREATE TABLE customer(
   name                 VARCHAR(100),
   taxID                INTEGER,
   country              VARCHAR(100) NOT NULL,
   mostExpensive        INTEGER NOT NULL,
   PRIMARY KEY (name,taxID)
);

CREATE TABLE orders(
   id           INTEGER DEFAULT nextval('seq_id') PRIMARY KEY,
   date         DATE NOT NULL,
   fromName     VARCHAR(100),
   fromTaxID    INTEGER,
   FOREIGN KEY (fromName,fromTaxID) REFERENCES customer(name,taxID)
);

CREATE TABLE supplier(
   name         VARCHAR(100) PRIMARY KEY,
   revenue      INTEGER NOT NULL CHECK(revenue BETWEEN 1000 AND 1000000),
   country      VARCHAR(100) NOT NULL
);

CREATE TABLE part(
   id   INTEGER REFERENCES orders(id),
   number       INTEGER,
   cost         NUMERIC(5,2) NOT NULL CHECK(cost > 0),
   fromNAME     VARCHAR(100) REFERENCES supplier(name),
   PRIMARY KEY (id,number)
);

ALTER TABLE customer ADD CONSTRAINT c_mostExpensive
   FOREIGN KEY (mostExpensive) REFERENCES orders(id)
   DEFERRABLE INITIALLY DEFERRED;
```

Hint: Take care of the order of your statements.

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

(4 points)

i) Consider the following schema:

`shelf` (id: *book.shelf*, newestBook: *book.name*)
`book` (shelf: *shelf.id*, name)

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.

shelf

id	**newestBook**
3	Murder On The Orient Express
2	A Dance With Dragons
4	The Left Hand of Darkness
5	1Q84
1	The Colour Of Magic

book

shelf	**name**
2	A Dance With Dragons
5	1Q84
4	The Left Hand of Darkness
3	Murder On The Orient Express
1	The Colour Of Magic

Note: There are multiple ways of solving this subtask, and only one possible solution is shown here.

ii) Consider the following schema:

`A` (id, B: *B.id* , C: *C.id*)
`B` (id, C: *C.id*)
`C` (id, B: *B.id*)

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

A

id	**B**	**C**
1	2	1
2	3	1
3	1	2

B

id	**C**
1	1
2	2
3	1

C

id	**B**
1	1
2	3
3	2

Note: There are multiple ways of solving this subtask, and only one possible solution is shown here.

Question 5: Recursive Queries (12)

You are given the following recursive query using the same database schema as was used in question 4 a) :

```
WITH RECURSIVE tmp(id) AS
(
 SELECT  o.id
 FROM    orders o
 WHERE   o.fromName = 'Max Mustermann' AND o.fromTaxID = '12'
UNION
 SELECT  o.id
 FROM    tmp t, orders o, part p, supplier s, customer c
 WHERE   t.id = p.orderID AND p.fromNAME = s.name AND s.country = c.country
         AND c.mostExpensive = o.id AND p.cost < 500 AND s.revenue > 50000
)
SELECT  t.id, c.name
FROM    tmp t, customer c, orders o
WHERE   t.id = o.id AND (o.fromName,o.fromTaxID) = (c.name,c.taxID);
```

Evaluate the given query over the database instance, which can be found on page B:

id	name
1	Max Mustermann
2	Erika Mustermann
3	Otto Normalverbraucher
4	Markus Moeglich
5	Lieschen Mueller

Question 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 on 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)

(4 points)

```
INSERT INTO part VALUES (1, 3, 100, 'Rolex');
INSERT INTO part VALUES (1, 4, 200, 'LG');

SELECT name, revenue FROM supplier;
```

```
 name | revenue
------------+---------
InGen | 100000
Acme Corp. | 300000
Yoyodyne | 200000
Alchemax | 400000
Lenovo | 35000
Rolex | 600100
LG | 45200
```

---

b)

(4 points)

```
INSERT INTO part VALUES (8, 0, 800, 'LG'), (9, 0, 200, 'Lenovo');

SELECT name, mostexpensive FROM customer;
```

```
The trigger is not executed, since the query fails with a foreign key violation (there
are no orders with ID 8 and 9).
The SELECT statement outputs the unchainged content of the database.
```

c)

(4 points)

```
INSERT INTO orders VALUES (8, '2021-01-03', 'Otto Normalverbraucher', 14);
INSERT INTO part VALUES (8, 2, 400, 'Lenovo');

SELECT orderid, number, cost FROM part WHERE fromname = 'Lenovo';
```

```
orderid | number | cost
---------+--------+--------
5 | 1 | 400.00
8 | 3 | 200.00
8 | 4 | 200.00
```

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 setMostExpensive() RETURNS TRIGGER AS $$
DECLARE customerName VARCHAR;
DECLARE customerTaxID INTEGER;
DECLARE lastMostExpensive INTEGER;
DECLARE lastMostExpensiveID INTEGER;
DECLARE currentMostExpensive INTEGER;
DECLARE mostExpensiveOrderID INTEGER;
BEGIN
    SELECT name, taxID, mostExpensive
    INTO customerName, customerTaxID, lastMostExpensiveID
    FROM customer, orders
    WHERE customer.name = orders.fromName AND customer.taxID = orders.fromTaxID
    AND orders.id = NEW.orderID;

    SELECT SUM(cost) INTO lastMostExpensive FROM part
    WHERE part.orderID = lastMostExpensiveID;

    SELECT totalCost, orderID INTO currentMostExpensive, mostExpensiveOrderID FROM (
        SELECT SUM(cost) AS totalCost, id AS orderID FROM part, orders
        WHERE orders.fromName = customerName AND orders.fromTaxID = customerTaxID
        AND orders.id = part.orderID
        GROUP BY id
      ) sq
      ORDER BY totalCost DESC
      LIMIT 1;

    IF currentMostExpensive > lastMostExpensive THEN
        UPDATE customer SET mostExpensive = mostExpensiveOrderID
        WHERE customer.name = customerName AND customer.taxID = customerTaxID;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trA AFTER INSERT ON part
       FOR EACH ROW EXECUTE PROCEDURE setMostExpensive();
```

```
CREATE FUNCTION increaseRevenue() RETURNS TRIGGER AS $$
DECLARE rev NUMERIC;
```

```
DECLARE rev_sum NUMERIC;
BEGIN

    SELECT revenue INTO rev FROM supplier WHERE name = NEW.fromName;
    SELECT rev + NEW.cost INTO rev_sum;
    IF rev_sum > 1000000 THEN
        RAISE NOTICE 'revenue␣too␣large';
    ELSE
        UPDATE supplier SET revenue = rev_sum WHERE supplier.name = NEW.fromNAME;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trB AFTER INSERT ON part
       FOR EACH ROW EXECUTE PROCEDURE increaseRevenue();

CREATE FUNCTION splitPart2() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.fromNAME = 'Lenovo' AND NEW.number = 2 THEN
        INSERT INTO part VALUES (NEW.orderID, 3, NEW.cost / 2, NEW.fromNAME), (NEW.orde
        RETURN NULL;
    ELSE
        RETURN NEW;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trC BEFORE INSERT ON part
       FOR EACH ROW EXECUTE PROCEDURE splitPart2();
```

Page B

Sample instance for Task 5 and Task 6:

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

Please do not write on top of this page! Solutions written on this sheet will not be graded!

customer

name	taxID	country	mostExpensive
Max Mustermann	12	Austria	1
Erika Mustermann	13	United States	2
Leon Mustermann	17	Germany	6
Anne Mustermann	18	Germany	7
Otto Normalverbraucher	14	Canada	3
Markus Moeglich	15	Denmark	4
Lieschen Mueller	16	Monaco	5

part

id	number	cost	fromName
1	0	100	InGen
1	1	200	Acme Corp.
1	2	300	Yoyodyne
2	0	600	Rolex
3	0	300	Alchemax
5	0	300	LG
5	1	400	Lenovo

orders

id	date	fromName	fromTaxID
1	2021-11-24	Max Mustermann	12
2	2021-10-12	Erika Mustermann	13
3	2020-09-14	Otto Normalverbraucher	14
4	2021-12-01	Markus Moeglich	15
5	2021-11-15	Lieschen Mueller	16
6	2021-11-27	Leon Mustermann	17
7	2020-12-12	Anne Mustermann	18

supplier

name	revenue	country
InGen	100000	United States
Acme Corp.	300000	Canada
Yoyodyne	200000	Denmark
Alchemax	400000	Monaco
Rolex	600000	Switzerland
Lenovo	35000	China
LG	45000	South Korea