

Gruppe A

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

EXAM AUS		06.05.2020
<input type="radio"/> DATENMODELLIERUNG 2 (184.790) <input type="radio"/> 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!**

Exercise 1: Locking (0)

The next page shows a sequence of basic operations of the transactions T_1, T_2, T_3 , and T_4 on the database objects A, B, C , and D .

a) The *strict 2-Phase Locking* protocol shall be used to synchronize the concurrent execution of these transactions.

(i) Show the *wait-for graph* at step (*) and state whether it shows a deadlock situation or not.

b) To prevent potential deadlocks, strict 2-Phase Locking is extended by the *wound-wait strategy*.

(i) Assign a suitable and correct timestamp to each transaction. In addition, for each restarted transaction state its timestamp after the restart.

(ii) List the sequence of lock requests and releases created when synchronizing the given sequence using the stated method. Use the notation described on the next page.

Assumptions and conventions (as known from the lecture/exercises):

Assume that locks are only requested when they are actually required, and that they are requested as late as possible. Within the boundaries of the synchronization strategy, you are free to choose when to release locks that are no longer needed.

*If a transaction is blocked, all its further actions are skipped (since the transaction is no longer running). Once a transaction is resumed (because the requested lock is now available), all its skipped actions are performed immediately. If more than once transaction is waiting for the same lock and the lock becomes available, it is granted to the transaction with the **smallest index (!)**.*

Lock Upgrades may be applied whenever the requesting transaction is the only transaction holding a lock on the object. With this exception, all requests for exclusive locks are treated equally, independent of whether the transaction already holds a shared lock on that resource or not. Once the exclusive lock is granted, the transaction holds only the exclusive lock, and no longer the shared lock.

Sequence for a) and b)

T_1	T_2	T_3	T_4
b_1 $r_1(A)$	b_2 $r_2(B)$	b_3 $w_3(C)$ $r_3(A)$ $r_3(B)$	b_4 $r_4(B)$ $w_4(D)$ $w_4(B)$
<i>restart?</i>			
$w_1(D)$	$w_2(A)$ $r_2(A)$ $r_2(B)$	$w_3(C)$	
(*)			
		c_3	
<i>restart?</i>			
$w_1(A)$ c_1	c_2		$r_4(D)$ c_4

Notation:

- $r_i(O)/w_i(O)$: Read-/write operation of T_i on object O .
- b_i/ c_i : Start/commit of T_i .
- *restart?*: Restarting a previously reset/aborted transaction [only b)]

Wait-for graph for task a)

Deadlock: ☐ yes ☐ no

Locks for tasks b)

Events to list and notation:

- Lock requests: $S_i(O)/X_i(O)$ to state that T_i requests a read-/write lock on O .
- Wait/blocking of a transaction: $wait_i$ to state that T_i did not receive a requested lock and is blocked.
- Lock granted: $gS_i(O)/gX_i(O)$ to state that a requested read-/write lock on O was granted to T_i .
Implicit if the request ($S_i(O)$ bzw. $X_i(O)$) is not followed by $wait_i$; need not be stated in such situations.
- Lock release: $rS_i(O)/rX_i(O)$ to state that T_i releases a read-/write lock on O it currently holds. Must always be stated explicitly (e.g. also in case of a *reset_i*).
- Resetting a transaction: *reset_i* to state that T_i was reset/aborted by the synchronization protocol.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

b) Timestamp: T_1 : T_2 : T_3 : T_4 : Restart:

(0)

We write $[LSN, TA, PageID, Redo, Undo, PrevLSN]$ for common log records, and $\langle LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN \rangle$ for compensation log records. For *BOT* and *COMMIT* records, the shortened forms $[LSN, TA, BOT, PrevLSN]$ resp. $[LSN, TA, COMMIT, PrevLSN]$ may be used. For simplicity, assume that the fields A , B , and C are located on the pages P_A , P_B , and P_C , respectively.

a) Assume A , B , and C have the following values at the beginning of the schedule shown below:

$$A = 15, B = 25, C = 10.$$

List all log records created when executing the given schedule. For this, assume that A , B , and C are located at P_A , P_B , and P_C , respectively, and that the **ROLLBACK** in line 14 gets finished before line 15 is executed. The log entries for the **BOT** statements are already given. Extend the log accordingly.

Schedule			
#	T_1	T_2	T_3
1			BOT
2	BOT		
3		BOT	
4		$w(A, 5)$	
5	$r(A, a_1)$		
6	$r(C, c_1)$		
7	$w(A, a_1 + c_1)$		
8	$r(B, b_1)$		
9		$r(A, a_2)$	
10			$r(B, b_3)$
11		$r(C, c_2)$	
12	$w(C, a_1 + 4)$		
13		$w(C, c_2 + a_2)$	
14		ROLLBACK	
15	$w(B, b_1 + 10)$		
16			$w(B, b_3 + b_3)$

Notation:

- $r_i(O, x_i)$: Transaction T_i reads the value of the database object O into the local variable x_i
- $w_i(O, x_i)$: Transaction T_i writes the value of x_i into the database object O
- BOT: Begin of Transaction

Provide the final values of A , B , and C .

A:

$B:$

$C:$

b) Consider the log records and content of the (database) pages P_A , P_B , P_C , and P_D as stated below. Based on these values, conduct a recovery following the ARIES procedure.

List the resulting log records.

Log records (given)	Log records (solution)
[#1, T_1 , BOT, #0]
[#2, T_2 , BOT, #0]
[#3, T_3 , BOT, #0]
[#4, T_4 , BOT, #0]
[#5, T_2 , P_C , C+=5, C-=5, #2]
[#6, T_4 , P_D , D+=1, D-=1, #4]
[#7, T_3 , P_D , D-=6, D+=6, #3]
⟨#8, T_4 , P_D , D-=1, #6, #4⟩
⟨#9, T_4 , BOT, #8⟩
[#10, T_1 , COMMIT, #1]
[#11, T_2 , P_A , A+=10, A-=10, #5]
[#12, T_3 , P_C , C+=1, C-=1, #7]
[#13, T_3 , P_A , A+=4, A-=4, #12]
⟨#14, T_3 , P_A , A-=4, #13, #12⟩
[#15, T_2 , P_A , A-=5, A+=5, #11]
⟨#16, T_3 , P_C , C-=1, #14, #7⟩
⟨#17, T_2 , P_A , A+=5, #15, #11⟩

Pages in persistent storage

P_A	LSN: #14
$A = 50$	

P_B	LSN: #0
$B = 11$	

P_C	LSN: #12
$C = 1701$	

P_D	LSN: #7
$D = 55$	

c) Is it possible, just based on the log records given in task b), to determine whether the corresponding schedule was recoverable (according to the formal classification of “recoverable schedules” in concurrency control)?

If it is possible, state whether the corresponding schedule was recoverable or not (and why).

If it is not possible, briefly justify your answer (1-2 sentences).

Being recoverable can be detected:
 ☐ yes
 ☐ no

.....

.....

.....

(Attention: Just checking yes/no without a suitable argument gives no points!)

Exercise 3: Properties of Transactions

(0)

Consider the schedule given 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 . The notation is as defined in task 1, a_i denotes the abort of transaction T_i .

T_1	T_2	T_3	T_4
b_1	b_2	b_3	b_4
$r_1(B)$			
	$r_2(C)$		
	$w_2(C)$		
		$r_3(C)$	
		$r_3(B)$	
		$w_3(C)$	
			$w_4(D)$
		$r_3(A)$	
		$w_3(A)$	
	$r_2(B)$		
			$r_4(B)$
$w_1(B)$			
$r_1(C)$			
	$w_2(D)$		
$r_1(B)$			
	c_2		
		a_3	
			$r_4(A)$
			c_4
$r_1(C)$			
c_1			

a) Provide the *precedence graph* (*Serialisierbarkeitsgraph*), and state whether the schedule is *conflict serializable*.

If it is *conflict serializable*, provide a conflict equivalent execution order of the transactions.

If it is *not conflict serializable*, state a minimal number of transactions that need to be removed for the schedule becoming conflict serializable. For the remaining transactions, also state a conflict equivalent execution order.

Precedence graph

Schedule is conflict serializable:

☐ yes ☐ no

Transactions to be removed:

Conflict equivalent, serial processing sequence of transactions:

.....

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

T_1 reads from T_2 reads from T_3 reads from T_4 reads from

c) State whether the schedule is recoverable and/or avoids cascading abort. Briefly justify/discuss your answer wrt. the given schedule.

Schedule is recoverable: ☐ yes ☐ no

Justification:

.....

Schedule avoids cascading abort: ☐ yes ☐ no

Justification:

.....

(Attention: Just checking yes/no without a reasonable justification gives no points!)

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 (here we provide an English translation of the schema to give it more meaning; you are free to use either the English or the German names):

author(name, institute, birthdate, bestPaper: *paper.name*)

paper(name, type, year, mainName, mainIns: (*author.name*, *author.institute*))

reviews(revName, revIns: (*author.name*, *author.institute*)), subName, subIns: (*author.name*, *author.institute*))

Each author is uniquely identified by their name and institute. In addition, their birthdate and best paper is stored as well. Each paper has a unique name, and must refer to its main author. Furthermore, a paper has a year and is of a certain type. The year needs to be an uneven number. The type must be an ENUM consisting of one of three values: ‘Journal’, ‘Conference’, or ‘arXiv’. Finally, there is a relation which models that authors can review the work of other authors.

Provide the necessary SQL commands to create database tables according to the provided schema. Make sure to implement all of the described integrity constraints. You may choose appropriate attributes for the columns.

Hint: Take care of the order of your statements.

Exercise 5: Recursive Queries

(14)

Consider the recursive SQL query over the database schema of “Exercise 4” given in “Exercise 5”.

```
WITH RECURSIVE tmp(id,titel) AS
(
SELECT   id, titel
FROM     webinar
WHERE    ('Diotima','Univ-Prof') = (leitName,leitTyp)
UNION ALL
SELECT   w2.id,w2.titel
FROM     tmp NATURAL JOIN webinar, webinar w2
WHERE    w2.nachfolger = tmp.id
           AND w2.id IN (SELECT * FROM popular_webinar)
)
SELECT * FROM tmp
```

Evaluate this query over the database instance given on the last page of the exam:

Exercise 6: PL/SQL Trigger

(14)

Write a PL/pgSQL trigger `trA` and the associated procedure to implement the following behavior:

- If a *paper* P with `type` *conference* or *journal* is inserted, then you should also create the arXiv version A as described below:
 - The `type` of A is 'arXiv'.
 - Year of publication and main author of A is the same as for P .
 - If P was published in 2010 or later, then append “ - Full” to the name of P to create the name of A .
 - If P was published before 2010, then append “ - Archived” to the name of P to create the name of A .
- Make sure that insertion of A will not violate any primary key constraints. If a paper with the same name (as A) already exists, print a warning instead of inserting A . **Do not abort the insertion of P .**

You may separate this page from 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!

Sample instance for Task 5:

paper

author

reviews