Exercise Sheet 3 (WS 2022)

6.0 VU Datenbanksysteme

About the exercises

General information

In this part of the exercises you apply the theoretical knowledge about transaction management, recovery, and multi-user synchronization that you have gained in the lecture.

We recommend you to solve the problems **on your own** (it is a good preparation for the exam – and also for your possible future job – to carry out the tasks autonomously). Please note that if we detect duplicate submissions or any plagiarism, both the "original" and the "copy" will be awarded 0 points.

Your submission must consist of a single, typeset PDF document (max. 5MB). **We do not accept PDF files with handwritten solutions.**

In total there are 8 tasks and at most 15 points that can be achieved on the entire sheet.

Deadlines

until 19.12.	**12:00 pm**:	Upload your solutions to TUWEL
on 9.01.	20:00 pm:	Evaluation and feedback is provided in TUWEL

Further questions – TUWEL forum

If you have any further questions concerning the contents or organization, do not hesitate to ask them on the TUWEL forum. **Do not post (partial) solutions in the forum!**

Tasks: Recovery

This section is about the use of log records to ensure atomicity and durability of transactions. For log records the format presented in the lecture is used, which is summarized here:
Log records for actions performed by a transaction have the following format

`[LSN, TA, PageID, Redo, Undo, PrevLSN]`,

where `LSN` indicates the LogSequenceNumber of the record, `TA` identifies the transaction performing the action and `PageID` specifies the page that was changed. `Redo` and `Undo` contain the redo/undo information and `PrevLSN` the LSN of the previous log record of the same transaction.

Redo- and *Undo*-information are recorded by describing the necessary changes using addition and subtraction (we will only deal with numerical values):

An example for such a log record is

$[\#i, T_j, P_X, X{+}{=}d_1, X{-}{=}d_2, \#k]$,

which indicates that according to the log record with the number i, the transaction T_j wrote a field X on page P_X. In order to redo this action, X needs to be increased by d_1, whereas to *undo* this step, X must be decreased by d_2. Finally, the previous log record of this transaction has number k.

Log records recording the begin of a transaction (BOT) and commit of transaction only contain the `LSN`, the `TA`, the type of operation (BOT or commit), and the `PrevLSN`. The corresponding log records thus have the following format:

`[LSN, TA, (BOT|Commit), PrevLSN]`.

Compensation Log Records (CLRs) are formatted as follows:

⟨`LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN`⟩,

with `UndoNextLSN` being the LSN of the next log record for the same transactions that needs to be undone. Like for the "standard" log records, also for BOT-CLRs the shortened format

⟨`LSN, TA, BOT, PrevLSN`⟩

may be used.

Note: You can fill the prepared table below to fill solutions. Alternatively, you can create two tables on your own with (a) results and (b) log or CLR entries. Make sure that you distinguish Log / CLR-records properly.

Task 1 (Logging) [1 point]

Consider the schedule in Figure 1, which shows the three transactions T_1, T_2 and T_3.

The capital letters A, B, C and D represent fields in the database and a_i, b_i, c_i and d_i are local variables of the different transactions. Moreover, $r_i(\Gamma, \gamma)$ denotes a read (the value of the field Γ is read from the database and stored in the local variable γ) and $w_i(\Gamma, \gamma)$ indicates a write (the value represented by γ is written to the field Γ in the database). `COMMIT` marks the successful termination of a transaction and `ROLLBACK` the rollback of a transaction. Assume that the rollback of a transaction takes places before the next action according to the schedule is performed, i.e. the rollback starts at the point marked with `ROLLBACK`, and completes successfully before the next action takes place.

Finally, assume that at the start (line 1) relevant part of the database consists of the following values:

A: 47 B: 11 C: 8 D: 15

(a) Lines in the history indicate changes in a field or local variable. Provide for each line of

the history the value of the field/variable after the operation. State the corresponding line number in the history. For stated `abort` operation use the line number 7. The table already contains expected solution for Lines 1–2.

(b) Provide changes to the log-entries of the specified history as a list. Use the order in which entries have been created. State records in the format as suggested in the beginning. Recall that information about *redo-* and *undo* is given via addition and subtraction, respectively. Assume that every field Γ is stored on page P_Γ. For better readability, use notion $\#i$ for `LSN` and `PrevLSN`, respectively. If there is no entry, use 0. Do not forget to include log-records for resetting transaction T_2.

Reminder: Format the log records in a readable way, e.g., in a list with one entry per row or a table with one record per row. Do *not* state the log records as continuous text. If the answer is formatted in a way that is not readable, we may give 0 points for this exercise. (In case you use the LaTeX template provided, you can find a suggestion for a somewhat readable format there. You can also just fill the prepared table below where solutions for Lines 1–2 have already been stated.)

Note: Format the log records in a readable way, e.g. in a real list (with one entry per row) or a table (again, one record per row). Do *not* write the log records as continuous text. If the answer is formatted in a way that is not readable, we may give 0 points for this exercise. (In case you use the LaTeX template provided, you can find a suggestion for a somewhat readable format there.)

	Given History			Sol. 1(a)	Solution Task 1 (b)					
	T_1	T_2	T_3		LSN	TA	Page	Redo	U/PL	PL/UN
1		BOT			[#1	T_2	BOT			#0]
2		$r_2(A, a_2)$		$a_2 = 47$				—		
3		$r_2(B, b_2)$						—		
4	BOT									
5	$r_1(C, c_1)$							—		
6		$w_2(D, 2 \cdot a_2 + b_2)$								
7		abort								
8	$r_1(D, d_1)$							—		
9			BOT							
10			$r_3(A, a_3)$					—		
11	$w_1(A, c_1 + 2 \cdot d)$									
12			$w_3(A, a_3 + 4)$							
13	$r_1(B, b_1)$							—		
14			$w_3(C, a_3 - 7)$							
15	$w_1(D, c_1 - 40)$									
16			$r_3(B, b_3)$					—		
17			$w_3(D, 27)$							
18			$w_3(C, b_3 + a_3)$							
19			commit							
20	commit									

Figure 1: History for Task 1. U... Undo, PL... PrevLSN, PL, UN...UndoNextLSN. Do not forget the brackets for Log/CLR records.

Task 2 (Recovery) [1.5 points]

Suppose that after a crash of some database system you find the situation shown in Figure 2. The left side of the figure shows the content of the recovered logfile. On the right side, the content of the pages P_A, P_C, and P_D is illustrated.

Log-Entries

[#1, T_2, `BOT`, #0]
[#2, T_2, P_B, -4,+4, #1]
[#3, T_2, P_A, +5, -5, #2]
[#4, T_3, `BOT`,#0]
[#5, T_3, P_C, +9, -9, #4]
[#6, T_1, `BOT`, #0]
[#7, T_2, P_B, +7, -7, #3]
[#8, T_2, `COMMIT`, #7]
[#9, T_3, P_A, -5, +5, #5]
[#10, T_1, P_B, -14, +14, #6]
⟨#11, T_3, P_A, +5, #9, #5⟩
[#12, T_1, P_D, -2, +2, #10]
⟨#13, T_3, P_C, -9, #11, #4⟩
[#14, T_1, P_B, -8, +8, #12]
[#15, T_1, P_D, +6, -6, #14]

Pages

P_A	LSN: #3
$A = 27$	

P_B	LSN: #2
$B = 4$	

P_C	LSN: #5
$C = 29$	

P_D	LSN: #0
$D = 18$	

Figure 2: Specification for Task 2: The content of the log archive (left) and the pages of the database (right) after some crash.

Use this information to carry out a recovery of the database.

(a) State the values of A, B, C, and D after the *redo*-step.

(b) State the Compensation Log Records (CLRs) created during the recovery.

(c) State the values of A, B, C, and D once the recovery is completed.

Tasks: Concurrency Control

Task 3 (Properties of transactions) [3 points]

Consider the set $\mathcal{T}_{\dashv}$ and $\mathcal{T}_{\lfloor}$ of transactions and the corresponding schedules $\mathcal{H}_{\dashv}$ and $\mathcal{H}_{\lfloor}$, which is given by a sequence of basic operations:

(a) $\mathcal{T}_a = \{T_1, T_2, T_3, T_4\}$

$\mathcal{H}_a = b_1 \rightarrow w_1(B) \rightarrow b_3 \rightarrow r_1(D) \rightarrow r_3(B) \rightarrow b_2 \rightarrow r_1(A) \rightarrow r_1(C) \rightarrow r_2(A) \rightarrow b_4 \rightarrow w_1(D) \rightarrow c_1 \rightarrow w_3(B) \rightarrow w_2(D) \rightarrow w_2(A) \rightarrow w_4(D) \rightarrow w_3(C) \rightarrow c_3 \rightarrow w_4(B) \rightarrow r_2(C) \rightarrow c_4 \rightarrow c_2$.

(a1) Create the precedence graph (serializability graph) SG($\mathcal{H}$).

(a2) For each edge in the precedence graph, provide at least one pair $p_i \rightarrow p_j$ of operations that justify the existence of this edge.

For the edges $T_i \rightarrow T_j$ (if they are part of the graph) list *all* pairs of operations that justify the existence of these edges.

(a3) If the schedule is conflict serializable, state *one* possible conflict equivalent serial order of the transactions. If the schedule is not conflict serializable, state a minimal set of transactions that must be removed from the schedule to get a conflict serializable schedule. For the remaining schedule, state such a conflict equivalent serial schedule.

(a4) List all pairs of transactions (T_i, T_j) in the schedule $\mathcal{H}$ such that T_j reads from T_i. For each such pair (T_i, T_j) state at least one pair $(w_i(X), r_j(X))$ of operations that make T_j reading from T_i.

(a5) Determine which of the following properties hold for the schedule $\mathcal{H}$:

- Resettable
- Avoids cascading abort
- Strict

Justify your answer.

(b) For the following schedule, determine whether the schedule is conflict serializable, and determine which of the properties (i) resettable, (ii) avoids cascading abort, and (iii) strict; are satisfied.

$\mathcal{T}_b = \{T_1, T_2, T_3, T_4\}$

$\mathcal{H}_b = b_3 \rightarrow r_3(A) \rightarrow b_1 \rightarrow b_2 \rightarrow w_3(A) \rightarrow w_2(A) \rightarrow b_4 \rightarrow r_4(D) \rightarrow r_2(B) \rightarrow w_4(D) \rightarrow w_1(B) \rightarrow r_1(D) \rightarrow w_1(D) \rightarrow r_1(C) \rightarrow w_4(C) \rightarrow r_3(C) \rightarrow r_2(D) \rightarrow w_2(D) \rightarrow c_4 \rightarrow c_1 \rightarrow c_3 \rightarrow c_2$

Again, justify/explain your answer.

Task 4 (Locks and Deadlocks) [2.5 points]

Consider the sequence of lock requests shown below, where "$\texttt{lockS}_i(O)$" (resp. "$\texttt{lockX}_i(O)$") indicates a transaction T_i requesting a shared (resp. an exclusive) lock on an object O, and "$\texttt{rel}_i(O)$" describes a transaction T_i releasing all locks on O:
$\texttt{lockX}_3(A) \rightarrow \texttt{lockS}_1(C) \rightarrow \texttt{lockX}_2(B) \rightarrow \texttt{lockX}_2(C) \rightarrow \texttt{lockS}_4(A) \rightarrow \texttt{lockX}_1(C) \rightarrow \texttt{rel}_4(A) \rightarrow \texttt{lockS}_3(B) \rightarrow \texttt{lockS}_1(D)$.

(a) Assume some DBMS receives the above sequence of lock requests and works through them in the given order for the transactions T_1, T_2, T_3 and T_4. Whenever some request cannot be granted to a transaction, this transaction is blocked (i.e. its subsequent lock requests are postponed until the transaction is revoked).

State the order in which the DBMS works through the lock requests, and state immediately after each request whether the lock is granted or whether the transaction has to wait for it. Use $\texttt{grantS}_i(O)$ and $\texttt{grantX}_i(O)$ to denote that a shared- resp. exclusive lock on an object O is granted, use $\texttt{wait}(i)$ to indicate that a transaction was blocked to wait for a lock, use $\texttt{relS}_i(O)$ and $\texttt{relX}_i(O)$ to show that a shared- resp. exclusive lock on object O was released (as the result of some $\texttt{rel}_i(O)$), and use $\texttt{resume}(i)$ to indicate that some transaction was revoked because the requested lock is now available and is granted.

To determine the correct order, assume that once a blocked transaction is revoked, it catches up all "skipped" operations, i.e. all of its omitted operations are performed, until either the transaction blocks again, or there are no more omitted operations for this transactions. Only if one of these two conditions is reached, proceed with working through the actions on the original release.

Example: Consider the sequence

$\texttt{lockS}_1(A) \rightarrow \texttt{lockS}_2(A) \rightarrow \texttt{lockX}_1(A) \rightarrow \texttt{lockX}_2(B) \rightarrow \texttt{lockS}_1(B)$

of lock requests of two transactions T_1, T_2.

We obtain the following list:

1:	$\texttt{lockS}_1(A)$
2:	$\texttt{grantS}_1(A)$
3:	$\texttt{lockS}_2(A)$
4:	$\texttt{grantS}_2(A)$
5:	$\texttt{lockX}_1(A)$
6:	$\texttt{wait}(1)$
7:	$\texttt{lockX}_2(B)$
8:	$\texttt{grantX}_2(B)$

(b) Sketch the current situation of held locks resp. blocked transactions. Therefor provide a table as shown below. Into each field of the table, insert an X (a S) if the corresponding transaction has an exclusive (resp. a shared) lock on this object. For each blocked transaction fill in a WS (*wait shared*) or WX (*wait exclusive*) into the corresponding field for the lock request that blocked the transaction.

	A	B	C	D
T_1				
T_2				
T_3				
T_4				

(c) Show the wait ("wait-for") graph for the current situation.

(d) State whether there currently exists a deadlock.

(e) State one possible sequence of releases that releases all locks currently held. If releasing such a lock makes it possible to continue a transaction, then all lock- and release requests by this transactions have to be handled before additional releases may be defined. If there currently exists a deadlock, abort transaction T_3 (i.e. all locks of T_3 are released immediately).

(f) Consider the given sequence of lock requests and releases. Does it violate two-phase locking (2PL)? What about the sequence you created in task (e)?

Task 5 (Two-Phase Locking) [1.5 points]

Consider the following three transactions T_1, T_2, and T_3, and the respective sequence of operations ($r_i(O)$ and $w_i(O)$ denote a read- resp. write operation by T_i on O, and c_i marks the commit of T_i).

$$\begin{array}{lccccccccccc} T_1\colon & r_1(D) & \to & r_1(A) & \to & r_1(B) & \to & r_1(A) & \to & r_1(C) & \to & c_1 \\ T_2\colon & r_2(A) & \to & r_2(C) & \to & w_2(A) & \to & r_2(D) & \to & w_2(B) & \to & c_2 \\ T_3\colon & r_3(D) & \to & w_3(D) & \to & r_3(B) & \to & r_3(C) & \to & w_3(B) & \to & c_3 \end{array}$$

Assume that *("normal") 2 Phase Locking* is applied to these transactions. State the resulting schedule (consisting of the lock requests, the read- and write operations, the releases of the locks, and the commits).

Make the following assumptions:

- *Notation:* Please use $\texttt{lockS}_i(O)$ and $\texttt{lockX}_i(O)$ to denote a request for a shared- resp. exclusive lock on object O by transaction T_i. Please also use $\texttt{rel}_i(O)$ to indicate the release of all locks by T_i on O. (*Hint:* You don't need to state explicitly whether a lock is granted or not. This information is given implicitly by the transaction performing a corresponding read resp. write afterwards – or not.)

- *Lock requests and releasing locks:* For each operation (read, write, commit), state the required lock requests (unless the transaction already holds the required locks). In doing so, assume that locks are requested as economical as possible, which means:
  - A lock is only requested if it is really needed.
  - A lock is held as short as possible, i.e. locks are requested at the latest possible time and released at the earliest possible point in time.

  Note: These assumptions are in addition to the rules of 2 Phase Locking.

- *Scheduling of the transactions:* Assume that each transaction performs one operation (read, write, commit) before the next transaction is scheduled. Make sure to begin with transaction 1. In our case the real sequence of actions is $r_1(A) \to r_2(C) \to r_3(B) \to r_1(B) \to \ldots$. Lock requests and releasing locks do not count as operations, i.e. before and after each operation (read, write, commit), the transaction is allowed to request or release an arbitrary number of locks, before it is the next transactions turn.

  Deviations from this sequence are only allowed if one transaction is blocked. In such a case, the transaction is omitted every time it is blocked. This is continued until the block on the transaction is lifted. Once this happens, the transaction again takes her turns in the schedule as before – again only performing one operation (read, write, commit) per turn.

  The following example demonstrates the control flow for two transactions: Assume T_1 consists of actions $\alpha_1, \alpha_2, \ldots$, and T_2 of $\beta_1, \beta_2, \ldots$. The normal sequence is $\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots$. Assuming T_2 requires a lock for β_3 that is held by T_1, i.e. T_2 blocks, then the sequence would continue $\alpha_3, \alpha_4, \alpha_5, \ldots$. If the lock is released after α_5, then the sequence continues with $\alpha_5, \beta_3, \alpha_6, \beta_4, \ldots$.

Task 6 (Multi-Granularity Locking) [3 points]

Consider the hierarchy within a database depicted in Figure 3.

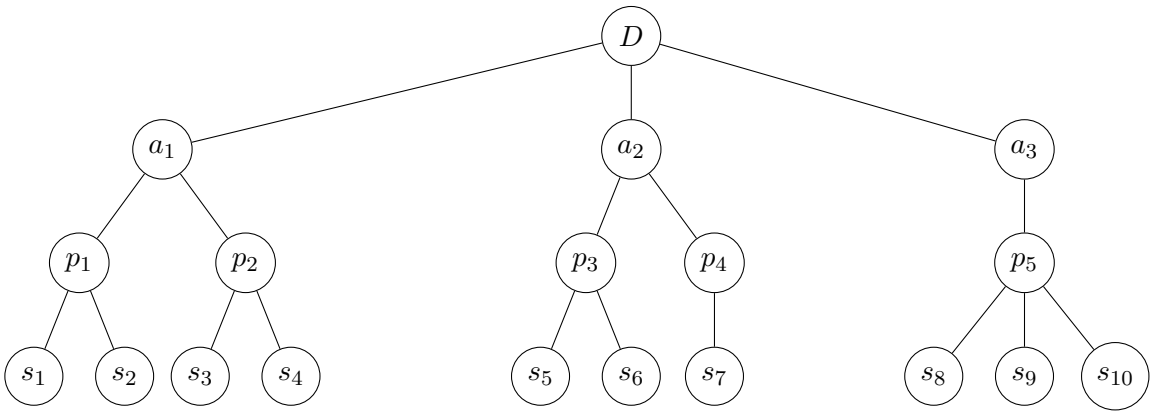

Figure 3: Datenbasis-Hierarchie zu Aufgabe 6

Consider the following sequences of lock requests and releases of locks of four transactions T_1, T_2, T_3, and T_4 on the resources described in Figure 3.

(a) $\texttt{lockS}_1(p_4) \rightarrow \texttt{lockX}_2(s_9) \rightarrow \texttt{lockS}_3(p_2) \rightarrow \texttt{lockX}_1(p_3) \rightarrow \texttt{lockS}_3(s_4) \rightarrow \texttt{lockX}_2(s_6) \rightarrow \texttt{lockS}_4(a_1) \rightarrow \texttt{rel}_1(p_3)$

(b) $\texttt{lockX}_1(p_4) \rightarrow \texttt{lockS}_2(p_5) \rightarrow \texttt{lockX}_3(p_1) \rightarrow \texttt{lockS}_3(s_8) \rightarrow \texttt{lockX}_2(s_5) \rightarrow \texttt{lockX}_1(s_6) \rightarrow \texttt{lockX}_1(p_1) \rightarrow \texttt{lockS}_2(p_3) \rightarrow \texttt{lockS}_4(s_{10}) \rightarrow \texttt{rel}_3(s_8) \rightarrow \texttt{lockX}_3(s_7) \rightarrow \texttt{rel}_1(p_4)$

Within these sequences, $\texttt{lockS}_i(O)$ denotes transaction T_i requesting a shared lock on object O, $\texttt{lockX}_i(O)$ denotes transaction T_i requesting an exclusive lock on O, and $\texttt{rel}_i(O)$ denotes transaction T_i releasing all locks held on O.

Answer the following questions for both sequences:

- Show how one has to proceed to process the lock requests and releases of locks in accordance to the Multi Granularity Locking (MGL) protocol: State the sequence of required lock requests, and in the case of a release state which further locks can be released as a result. *Hint:* Make sure to keep the required order in both cases, when requesting and releasing a lock.

  Please use the following notation: $\texttt{lockIS}_i(O)$, $\texttt{lockIX}_i(O)$, $\texttt{lockS}_i(O)$, and $\texttt{lockX}_i(O)$ for transaction T_i requesting a `IS`-, `IX`-, `S`- resp. `X`-lock on object O; $\texttt{relIS}_i(O)$, $\texttt{relIX}_i(O)$, $\texttt{relS}_i(O)$, and $\texttt{relX}_i(O)$ for transaction T_i releasing a `IS`-, `IS`-, `S`- resp. `X`-lock on object O. Make also sure that a transaction only requires locks it does not already hold.

- Mark lock requests that cannot be granted. Assume that in such a case the corresponding transaction is blocked, i.e. no further actions of the transaction are performed until the lock is granted and the transaction revoked. This may happen because of another transaction releasing the lock (Lock requests and releases of locks of blocked transactions are omitted). Once the transaction is revoked, all omitted actions are made up for before the processing of the sequence is continued at the position where the lock was released.

- For each declined lock, state why this lock is not granted.

- Does a deadlock occur as a result of this sequence? If this is the case, explain why.
- If no deadlock occurs, but there is at least one transaction blocked at the end of the sequence: Provide a minimal number of locks that must be released to continue these transactions. By doing that, note that transactions can only release shared and exclusive locks explicitly; but provide also the IX- and IS-locks which are released implicitly by them. (Don't forget to maintain the right order.)

  Once the blocked transaction is revoked, continue the execution of this transaction. If this makes the transaction block again, state again a minimal number of locks that need to be released in order to continue with processing the transaction.

Hint:

- If the situation occurs that a transaction acquires a lock it already holds on one or more children of this node, you may assume that these locks are automatically removed from the child nodes (you need not state the release of these locks explicitly).

Task 7 (Timestamp based Locking) [2 points]

Given below a schedule of three transactions T_1, T_2, T_3, which access three objects A, B, and C. We employ timestamp-based synchronization as discussed in the lecture. Assume that when using timestamp-based locking the values for `readTS` and `writeTS` are all 0 at the beginning.

- Output the actual schedule of the three transactions, which follows the timestamp-based synchronization process.

  Assume that aborted transaction are repeated in the order in that they have been aborted. Restart begins as soon as all other transactions finished.

- Provide to all read and write operations values of `readTS`(A), `readTS`(b), `readTS`(C), `writeTS`(A), `writeTS`(b), `writeTS`(C). Take as timestamp of the transactions # of `BOT`.

#	T_1	T_2	T_3
1			`BOT`
2	`BOT`		
3		`BOT`	
4			$r_3(A)$
5	$r_1(C)$		
6	$w_1(A)$		
7		$r_2(C)$	
8			$w_3(C)$
9			`commit`
10	$r_1(B)$		
11		$w_2(C)$	
12		`commit`	
13	$w_1(C)$		
14	$w_1(A)$		
15	`commit`		

Task 8 (Transactions in SQL) [0.5 points]

For the following task, you can run parts on a database system. We provide you with access to a database. Alternatively, you can also run the task on a local machine. **Details can be found here:** `https://tuwel.tuwien.ac.at/mod/page/view.php?id=1654198` *Note that the task primarily asks for conceptual questions. In principle, you can answer the questions also without using the database system, but it might be helpful to actually run it on a database especially for task (c).*

Consider the following relational schema of a company, that contains records of performed tasks and charging information on these tasks: (Primary keys are underlined, foreign keys are italic).

```
Tasks    (ticket_id, assigned_to, reviewed_by, description, effort)
done     (ticket_id:  Tasks.ticket_id)
tested   (ticket_id:  Tasks.ticket_id, when_tested)
```

Identify for every scenario described below the *lowest* isolation level that offers the required degree of isolation, respectively. Describe also, whether transactions can run with the given concurrency in their respective isolation level as desired, or whether problems are occurring. Due to lack of space, below we provide only sketches of the queries/transactions. You can find the complete queries in the available SQL files.

(a) *Description:* Sophia and Emma find a conceptual bug in a specific task (FeatureXZ).Each of them thinks that they need about 40 hours for solving the task and need to setup a new task in the task tracking system. However, they also need to update existing tasks – where each tasks will require about 20 hours in addition for implementation and test. Sophia updates the implementation. Emma takes care of the test. *History:*

Sophia	**Emma**
	`BEGIN;` `SET TRANSACTION ISOLATION LEVEL ____`
`BEGIN;` `SET TRANSACTION ISOLATION LEVEL ____`	

`UPDATE Tasks`
`SET effort=effort + 20`
`WHERE description like '%FeatureXZ%';`	

`DELETE FROM done WHERE ticket_id IN`
`(SELECT ticket_id FROM Tasks`
`WHERE description like '%FeatureXZ%');`	

	`UPDATE Tasks`
	`SET effort=effort + 20`
	`WHERE description like '%FeatureXZ%';`

	`DELETE FROM done WHERE ticket_id IN`
	`(SELECT ticket_id FROM Tasks`
	`WHERE description like '%FeatureXZ%');`

`INSERT INTO Tasks VALUES` `(9, 'Emma', 'Sophia', 'Concept', 40);`	
	`INSERT INTO Tasks VALUES` `(8, 'Sophia', 'Emma', 'Concept', 40);`
`COMMIT;`	
	`COMMIT;`

(b) *Description:* Before tasks can be released as feature, they must be approved by a project owner. Approvals are set by transactions into the database. The transactions should run in parallel on the database and unrestrictive. Moreover, we are fine if results of finished parallel transactions show up.

History:

ProjectOwner	**Parallel Queries**
`BEGIN;` `SET TRANSACTION ISOLATION LEVEL ____`	
`SELECT * FROM Tasks WHERE ¬approved`	
	`INSERT INTO Tasks VALUES (12, ...);`
	`INSERT INTO Tasks VALUES (13, ...);`
`SELECT * FROM Tasks NATURAL JOIN tested`	
`SELECT * FROM Tasks NATURAL JOIN tested` `WHERE ¬tested`	
`INSERT INTO approved VALUES (5, ...);`	
`SELECT * FROM Tasks WHERE ¬approved`	
	`INSERT INTO Tasks VALUES (10, ...);`
	`INSERT INTO Tasks VALUES (11, ...);`
`SELECT * FROM Tasks WHERE ¬approved`	
`COMMIT;`	

where `¬approved` and `¬tested` are short hand notations for conditions `ticket_id NOT IN (SELECT ticket_id FROM approved)` and `ticket_id NOT IN (SELECT ticket_id FROM tested)`, respectively.

(c) *Description:* Accounting wants to write an invoice to clients for which features have been successfully implemented, but not yet paid. For this task, it is essential to keep the data stable, meaning data is not allowed to change by potential parallel access.

History:

Accounting	**Parallele Anfragen**
`BEGIN;` `SET TRANSACTION ISOLATION LEVEL ____`	
`SELECT * FROM tasks` `  NATURAL JOIN done WHERE billed IS NULL`	
	`INSERT INTO Tasks VALUES (12, ...);`
	`INSERT INTO Tasks VALUES (13, ...);`
`SELECT sum(amount) FROM open`	
	`INSERT INTO Tasks VALUES (10, ...);`
	`INSERT INTO Tasks VALUES (11, ...);`

`SELECT count(ticket_id) FROM tasks`
`  NATURAL JOIN done WHERE billed IS NULL`
`COMMIT;`	

Please note: You can test these scenarios with the SQL files on a DBMS (the files are written for Postgres, but they can be tested with minor changes on another DBMS). With Postgres, please proceed as follows:

1. Open a database console by the command `psql`.

2. In another terminal, open a second database console by `psql`.
3. In one terminal, load the scenario by `\i Szenario1-a.sql` (where "1" should be replaced by the desired number)
4. In the other terminal, load the parallel scenario by `ıSzenario1-b.sql` (where "1" should be replaced by the desired number)
5. The execution of the SQL commands should be interrupted in both terminals with the message `Press Enter to continue (i)`
6. Press [Enter] in that console where `i` has the smaller value.

If the result at a chosen isolation level differs from your expectations, feel free to document this. In this case please provide the DBMS where you executed the transactions.

You can either use a database on your machine, or you can use our server `bordo`. You can connect to the server via ssh and then use `psql` to get access to a PostgreSQL database. You can find additional information about how you can connect to the server and log in into the database in TUWEL. Your login information are provided in TUWEL.

You can use a docker container or the provided system via ssh. See `https://tuwel.tuwien.ac.at/mod/page/view.php?id=1654198` for more details.