Handin 8 - 10 ECTS

Exercise (1)**.**

Let T_1, T_2, T_3 be transactions that operate on objects M, N, O, P and Q. We now have the following schedules: (For brevity we use the notation $r_1(M)$ to mean that transaction T_1 reads object M, $w_1(M)$ to mean transaction T_1 wrote to object M and c_1 to mean that transaction T_1 commits.)
For each these schedules, answer the following:

a. a. What does the precedence graph look like? Please remember to add labels on the edges and to draw legibly.

I have created the following two tables to get a better sense of what is going on, and have chosen to omit the tables provided in the exercise. I have also chosen to omit the commits at the end.

Schedule S_1

T_1	T_2	T_3
$r_1(M)$		
	$r_2(O)$	
		$w_3(P)$
$w_1(M)$		
$r_1(O)$		
	$w_2(M)$	
	$r_2(N)$	
	$r_2(O)$	
	$w_2(N)$	
		$w_3(O)$
	$r_2(M)$	
$w_1(N)$		
$r_1(N)$		
		$r_3(P)$
$w_1(N)$		
		$r_3(N)$

Schedule S_2

T_1	T_2	T_3
$r_1(Q)$		
	$r_2(N)$	
	$r_2(M)$	
	$w_2(N)$	
	$w_2(M)$	
$w_1(N)$		
	$r_2(P)$	
	$r_2(Q)$	
		$r_3(Q)$
	$r_2(M)$	
	$r_2(O)$	
	$w_2(M)$	
	$w_2(P)$	
$r_1(M)$		
	$w_2(O)$	
$w_1(M)$		
$r_1(O)$		
	$r_2(Q)$	
		$r_3(P)$
$r_1(M)$		
		$w_3(P)$
$w_1(M)$		
		$r_3(M)$
$w_1(O)$		
		$r_3(M)$
$w_1(N)$		
		$r_3(O)$
		$r_3(N)$
		$r_3(O)$
		$w_3(M)$

From these tables we can create precedence graphs using the rules as described by Fatemeh in the youtube video,

Precedence Graph

For each of the transactions T_i create a node. Then for each operations in the transac tions,

1. if $R_i(X)$ is above $W_j(X)$ where $i \neq j$ add a directed edge from $i \rightarrow j$
2. if $W_i(X)$ is above $R_j(X)$ where $i \neq j$ add a directed edge from $i \rightarrow j$
3. if $W_i(X)$ is above $W_j(X)$ where $i \neq j$ add a directed edge from $i \rightarrow j$

This leads to the following precedence graphs,

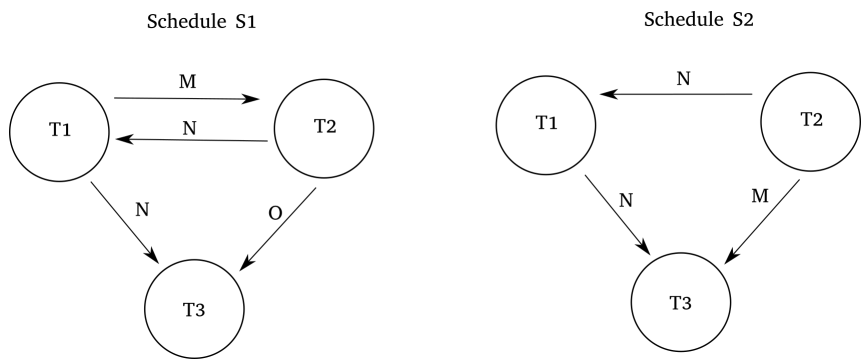

Figure 1.1 Precendence graphs for the schedules S_1 and S_2.

Where, for example, the N_{21} edge in S_2 comes from the read $r_2(N)$ that is then followed by a write $w_1(N)$. We could in principle have added more edges to these graphs, but they would have followed the same paths and are therefore irrelevant.

b. b. Is the schedule serializable? Why so / why not?

We can determine whether a schedule is serializable by checking for loops in the precedence graphs. Since S_1 has a loop, it is not serializable, whereas S_2 is.

c. c. If yes, what is the equivalent serial schedule?

Lets recall, that a serial schedule is one that is ordered by the transactions. That is, each transactions finishes all its operations before the next one begins. Looking at the precende graph, its clear that the order for S_2 should be $T_2 \rightarrow T_1 \rightarrow T_3$

d. d. Is it possible for this schedule to have been produced by 2PL? If so, expand the schedule with the acquisition and release of locks.

In this exercise, i'm a bit unsure whether we are supposed to use the equivalent serial schedule, or the original schedule. Let's first look at the original schedule. In 2PL, the procedure is at follows. A transaction starts by acquiring locks, does all it's operations, and then releases the locks. It is not allowed to acquire - release - acquire. In other words, once it has entered the shrinking phase, it can not begin to grow again. Let's check whether the original S_2 satisfies this. We can take a look at the following sequence,

$$r_2(N), \ldots, W_1(N), r_2(P).$$

In this sequence, T_2 first acquires a read lock on N. Then T_1 acquires a write, which requires T_2 to release it's read lock. This implies that T_1 is now in the shrinking phase. However, T_2 immediately acquires a read lock on P which violates 2PL.

The equivalent serial schedule, will trivially follow 2PL, as we can just acquire all required at the beginning of the transaction and then release them at the end,

Exercise (2)**.**

Assume we have a relation Accounts(id, name, amount) which describes accounts at a bank. Alice now wants to transfer 100kr to Bob's account. This can be done by the following SQL statements.

```
UPDATE Accounts
SET amount = amount - 100
WHERE name = "Alice"
```

```
UPDATE Accounts
SET amount = amount + 100
WHERE name = "Bob"
```

It is furthermore the time of the month where the bank calculates and adds the interest, 2% of the amount, to the account. Therefore, it runs the following update on the database:

```
UPDATE Accounts
SET amount = amount * 1.02
```

Now answer the following:

a. What can be incorrect outcomes if the above SQL is not treated as ACID transactions? Give an example of a schedule of the operations executed by the above SQL statements that leads to an incorrect outcome.

For this to go well, we essentially need to ensure that each of these transactions complte before the next one begins. If they are interleaved we can run into some issues, here is an example of one issue that might arise. Lets call Bob T_2 and the bank T_3. They each have to do the following operations,

$$T_2 : R_2(B) W_2(B)$$
$$T_3 : R_3(B) W_3(B).$$

Bob reads his bank account, adds 100, and then writes this new amount. The bank reads bobs bank account, multiplies by 1.02 and then writes this new amount. The problem arises, if we have the following sequence,

$$R_3(B) R_2(B) W_2(B) W_3(B).$$

In this scenario the bank reads Bob's account before his changes and then overwrites his changes at the end. As a result, the 100kr that Bob added are lost.

b. Which properties of ACID would it break? Why?

The property that is primarily broken here is *isolation*. Isolation ensures that each of the transactions dont interfere with one another, it essentially ensures that the database operations appear to be serial. In our case, the banks operations and Bobs are not isolated, which leads the bank to accidentally overwrite Bobs transfer. Note, that the precedence graph for this schedule has a loop, as we have the following sequence,

$$R_3(B) \to W_2(B) \to W_3(B).$$

The way to fix this, would be to use locks. If we enforced 2PL, the bank would not be able to acquire a write lock on B, after it had released its read-lock when Bob began writing.

Exercise (3)**.**

Consider the following trip description:

When Katrine took the plane to Munich, she checked in her suitcase and got her boarding pass as usual. When at the gate, ready to board the plane, she scans the barcode on her boarding pass. The electronic barrier malfunctions and does not open to let her through, and when she tries to scan at another barrier it does not let her through either as the bar code has already been read by the system. The staff figures that this is simply a flaw in the system, looks at her boarding pass and lets her board the plane. Already seated in the plane, Katrine hears the pilot announce that the departure has been delayed as they need to remove a passenger's luggage from the plane as the passenger did not board the plane. It is only on arrival in Munich, when her luggage is missing, that she realises that it was her luggage which was removed from the airplane.

Now consider the entire trip, from Katrine entering the airport to her realizing her luggage did not make it to Munich, from a database perspective. Think of this trip as a transaction. Which important property is violated by the system or by the staff working with the system?

The major problem seems to be consistency. When Kathrine scanned the barcode on her boarding pass, she is registered as having entered the barrier. However, it seems like the staff don't register this, so her luggage is removed from the plane. One part of the system has her registered as on the plane, while another doesn't and this leads to inconsistency