Handin 10 - 10 ECTS

We have used Generative Artificial Intelligence tools in doing this assignment, for the following legitimate use cases only: to get background information or understand the topic / problem, to improve writing of own text, to find gaps in our knowledge. The solution of the assignment is entirely our own.

Exercise (1).

What happens for the following sequences of events if a validation-based scheduler is used and why?

a) R1(A), R2(B), R1(B), V1, R3(C), R2(C), R3(D), V3, W1(C), W3(D), V2, W2(A).

b) R1(A), R1(B), V1, R2(B), R2(C), R3(C), V2, V3, W1(A), W2(C), W3(B).

c) R1(A), R2(B), R1(B), R2(C), V1, R3(A), R3(D), V2, W1(A), V3, W3(B), W2(A).

where Ri(X) means that transaction i has X in its read set, Vi means that Ti attempts to validate, and Wi(X) means that Ti enters the write phase, finishes and its write set was X. Ti starts at the first Ri(X) which is performed.

We shall do these in order,

a)

When checking whether transactions will be succesful, it is important to determine which order the transactions validate in (which gives their timestamp) and what their respective read and write sets are. For all of the three schedules, we see that the transactions begin in chronogical order, but they dont validate in the same order, so they will get different time-stamps. The read and write-sets for the transactions are as follows, Now let's analyse the schedule, and check whether there are any

	read_set	write_set
T1	A, B	C
T2	B, C	A
T3	C, D	D

Read and write-sets for schedule 1.

conflicts. The first to attempt to validate is T1, and at this point it is assigned $TS(T_1) = 1$. Since T1 is the youngest, the first one to validate and there aren't any other transactions in either the validation or write phase, its validation is succesful. It then enters the write phase. The next to attempt validation T3, and it is assigned the timestamp $TS(T_3) = 2$. Since T_1 is still in its write-phase, we have to check whether there are any conflicts. Specifically we have to check whether any of the following conditions are met,

Validation

When attempting to validate T_i, then for all other recent T_j in write phase or validation phase with $TS(T_j) < TS(T_i)$, then one of the following three conditions must hold,

1. T_j has finished all three phases before T_i began.

2. T_j finished its write phase before T_i begins its write phase, and T_j didn't write to any items read by T_i. i.e.
$$\text{write\_set}(T_j) \cap \text{read\_set}(T_i) = \emptyset.$$

3. T_j completes its read phase before T_i begins its read phase and,
$$\text{write\_set}(T_j) \cap \text{read\_set}(T_i) = \emptyset$$
$$\text{write\_set}(T_j) \cap \text{write\_set}(T_i) = \emptyset.$$

We see that T_1 does finish its write phase before T_3 begins, as $W_1(C)$ happens before $W_3(D)$, however there is overlap between their read and write sets,
$$\text{write\_set}(T_1) \cap \text{read\_set}(T_2) = C.$$
And since this condition is required for the third condition aswell, T_3 fails and aborts. The third to attempt validation is T_2 which is given the timestamp $TS(T_2) = 3$. This transaction runs into the same issue.
$$\text{write\_set}(T_1) \cap \text{read\_set}(T_3) = C.$$
And is therefore also forced to abort. Since T_1 is still running, it will be allowed to write it's changes to C and commit them.

b)

We start by listing the read and write sets again. The first to validate is T_1, it is given $TS(T_1) = 1$ and

	read_set	write_set
T1	A, B	A
T2	B, C	C
T3	C	B

Read and write-sets for schedule 2.

as before the validation is succesful. Next T_2 attempts to validate and it is given $TS(T_2) = 2$. In this case there is no overlap between the write set of T_1 and read and write sets of T_2,
$$\text{write\_set}(T_1) \cap \text{read\_set}(T_2) = \emptyset$$
$$\text{write\_set}(T_1) \cap \text{write\_set}(T_2) = \emptyset.$$
And furthermore, T_1 completed its read-phase before T_2 began its read phase. This implies that condition 3 is met, and T_2's validation is succesful. The last to attempt validation is T_3, $TS(T_3) = 3$. Since both T_1 and T_2 are still running, we have to check for conflicts with both of them. However, we see that T_2 writes to C, which T_3 reads,
$$\text{write\_set}(T_2) \cap \text{read\_set}(T_3) = C.$$
Which yields a conflict, and therefore T_3 is forced to abort. Since T_1 and T_2 were able to validate, they are allowed to write their changes and commit.

1.0.1 *c)*

We again list the read and write sets, As before T_1 validates first and this is allowed. The next to attempt validation is T_2, since T_1 is still running, we need to check for conflicts. Let's start by checking for overlaps between read and write sets,
$$\text{write\_set}(T_1) \cap \text{read\_set}(T_2) = \emptyset$$
$$\text{write\_set}(T_1) \cap \text{write\_set}(T_2) = A.$$

	read_set	write_set
T1	A, B	A
T2	B, C	A
T3	A, D	B

Read and write-sets for schedule 3.

Note that, T_1 does finish its write phase before T_2 begins its write phase, so condition 2 is met, and T_2 is allowed to validate. Finally T_3 attempts to validate, and as before we have to check for conflicts with T_1 and T_2. We check for overlaps,

$$\begin{aligned}
\text{write_set}(T_1) \cap \text{read_set}(T_3) &= A \\
\text{write_set}(T_1) \cap \text{write_set}(T_3) &= \emptyset \\
\text{write_set}(T_2) \cap \text{read_set}(T_3) &= A \\
\text{write_set}(T_2) \cap \text{write_set}(T_3) &= \emptyset.
\end{aligned}$$

We see that neither T_1 or T_2 finish before T_3 begins, and the overlaps between write and read sets, imply that T_3 will have to abort. As before T_1 and T_2 are allowed to write and commit their changes.

Exercise (2).

Given the granularity hierarchy, where T1 has already acquired the locks in red, answer for each of the transactions below if it will be allowed and what locks will need to be acquired for the transaction to proceed. Each transaction starts on the granularity hierarchy as it is in the figure, i.e. T3 does not need to take locks acquired by T2 into account.

a) T2: Update all records on page p1.

b) T3: Read everything on page p2 and update record r4.

c) T4: Read record r4 and update record r5.

a)

Since T_2 is updating p_1 it will eventually have to acquire an X lock on p_1. According to the multiple granularity locking rules, T_2 is then required to have IX or SIX locks on all ancestor nodes. We obtain the following sequence of locks,

$$SIX\,(db) \rightarrow SIX\,(f_1) \rightarrow X\,(p_1)\,.$$

This sequence is allowed as T_1 only holds IS locks on db and f_1, and these do not conflict with SIX.

b)

This will not be allowed, as T_3 will have to require an X lock on r_4, which is in conflict with the S lock already held by T_1.

c)

Since T_3 will be doing both reading and writing on different nodes beneath p_2, it essentially has different requirements for locks on the ancestor nodes. In this case the most restrictive requirements of the different operations are the ones that should be applied. Since it will be needing an X lock on r_5, it will SIX locks on all the ancestors. These do not conflict with T_1's locks. It will also need to an acquire S lock on r4, which is likewise fine.

$$SIX\,(db) \to SIX\,(f_1) \to SIX\,(p_2) \to S\,(r_4) \to X\,(r_5)\,.$$

Exercise (3)**.**

Consider the log below after a crash.

```
<START T>
<T, A, 10, 11>
<START U>
<U, B, 20, 21>
<T, C, 30, 31>
<U, D, 40, 41>
<COMMIT U>
```

a) Using the UNDO/REDO recovery algorithm, identify which transactions must be redone and which must be undone.

b) Furthermore, list the actions that must be taken for the recovery manager to bring the database back to a consistent state and why.

a)

When using the UNDO/REDO the key is to use the log to check which transaction where able to finish. These should be redone, as the user has commited and expect their changes to persist. Transaction that were underway, but never got to commit should be undone. So in this case T should be undone and U should be redone.

b)

The specific actions that should be taken are.

- Restore the variables A, C to what they were before T ran.
- Set the values of B, D to what U intended.