Mandatory Exercise 3

IN3020&IN4020 (2026)

Deadline: 20 April 2026, 23:59 (Devilry)

Question 1.

Consider the following schedule:

$$\mathtt{r}_1(X), \mathtt{eval}_1(X := X - 10), \mathtt{w}_1(X), \mathtt{r}_2(Y), \mathtt{eval}_2(Y := Y - 20), \mathtt{w}_2(Y),$$
$$\mathtt{r}_1(Y), \mathtt{eval}_1(Y := Y + 10), \mathtt{w}_1(Y), \mathtt{r}_2(X), \mathtt{eval}_2(X := X - Y), \mathtt{w}_2(X)$$

Note: In this schedule, $\mathtt{eval}_i(\cdot)$ is an operation evaluated by transaction T_i internally—that is, without touching the (shared) data; as discussed in lectures, these operations are usually omitted when writing schedules, but in this case they are written explicitly.

Part a.

Is this schedule (conflict-)serialisable? If your answer is yes, give an equivalent serial schedule. If no, indicate the conflicts preventing serialisability.

Part b.

Is this schedule serialisable on the base of semantic equivalence? Here, we say that two schedules are semantic-equivalent if, for every initial database state, the state of the state of the database after the execution of the two schedules on this initial state are the same. If your answer is yes, give a semantically equivalent serial schedule. If not, demonstrate that all serial schedules are not semantically equivalent to this schedule.

Question 2.

Consider the following schedule:

$$\mathtt{w}_1(X), \mathtt{r}_2(X), \mathtt{w}_1(X), \mathtt{r}_1(Z), \mathtt{w}_2(X), \mathtt{w}_3(X), \mathtt{r}_3(Y), \mathtt{r}_1(Y), \mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_3$$

Part a.

Is this schedule recoverable? If not, modify it in a minimal possible way to make it recoverable (your modification here and below should preserve the order of the operations in each of the involved transactions individually, but does not need to preserve conflict-equivalence).

Part b.

Is your recoverable schedule from Part a (i.e., the original or the modified) cascadeless? If not, modify it in a minimal possible way to make it cascadeless.

Part c.

Is your cascadeless schedule from Part b (i.e., the original or the modified) strict? If not, modify it in a minimal possible way to make it strict.

Question 3.

Consider the following three transactions:

$T_1 : \texttt{read}(X), \texttt{write}(Y) \qquad T_2 : \texttt{write}(Z), \texttt{read}(X) \qquad T_3 : \texttt{read}(Y), \texttt{write}(Z)$

Part a.

Add appropriate locks to these transactions so that they can be composed to a schedule following the two-phase locking protocol with shared/exclusive locks and without conversion.

Part b.

Can the transactions (extended with the locks) end up in a deadlock when evaluated under the protocol? If yes, demonstrate it by giving a schedule in a deadlock. If not, modify the transactions by changing reads to writes and/or writes to reads as well as possibly moving locks and unlocks (together with appropriate changing of lock types) in a minimal way so that these transactions can end up in a deadlock, and demonstrate this in the same way.

Part c.

Make a schedule of these transactions (original, without the locks) that

- follows the (basic) timestamp-based protocol,
- has to roll back a transaction due to the protocol rule violation when the Thomas write rule is not used,
- successfully commits all transactions when the Thomas write rule is used.

Question 4.

Consider the following log of an incomplete schedule evaluated under ARIES recovery protocol:

Lsn	Last_lsn	Tran_id	Type	Buffer_id	...
1	0	T_1	write	A	...
2	1	T_1	write	B	...
3	2	T_1	write	C	...
4	`begin_checkpoint`				...
5	`end_checkpoint`				...
6	0	T_2	write	B	...
7	6	T_2	commit		...

where the following transaction and dirty buffer (a.k.a. dirty page) tables are appended to the `end_checkpoint` statement:

TRANSACTION TABLE

Transaction_id	Last_lsn	Status
T_1	3	in progress

DIRTY BUFFER TABLE

Buffer_id	Lsn
B	2
C	3

Assume that a system crash happened after the evaluation of the operations recorded in the log. Which operations will be redone and undone during (successful) recovery and in which order?