Weekly exercises IN3020&4020 (2026)
Week 9: Transaction Management Concepts and Basic Concurrency Protocols

1 Concepts Review

1. Remind yourself what is a transaction, schedule and conflict, and what is ACID principles.
2. What is a serial schedule? Does two serial schedules of the same transactions always result in the same database state?
3. What is (conflict-)serialisability and how can serialisability be used as a way to ensure isolation?
4. What does it mean for two schedules to be conflict-equivalent and what is the tool to check it?
5. What does it mean for two schedules to be view-equivalent and what is the relations between conflict-equivalent and view-equivalent schedules?
6. What is recoverable, cascadeless and strict schedules, what is their purpose and relations?
7. Discuss the four levels of isolation in SQL.

2 Serialisability

1. Let T_1 be the transaction $r_1(X), w_1(X), r_1(Y), w_1(Y)$ and T_2 the transaction $r_2(X), w_2(X)$. List several (at least 5) possible schedules of T_1 and T_2 and decide which are (conflict-)serialisable.
2. Below are two transactions, described in terms of their effect on the database elements A and B, which we may assume are integers.

T_1
`read`(X);
$X := X - 1$;
`write`(X);
`read`(Y);
$Y := Y + 1$;
`write`(Y);

T_2
`read`(X);
$X := X + 1$;
`write`(X);

Figure 1: Transactions T_1 and T_2

T_1
$r_1(A)$
$A := A + 1$
$w_1(A)$
$r_1(B)$
$B := B * 3$
$w_1(B)$

T_2
$r_2(B)$
$B := B * 2$
$w_2(B)$
$r_2(A)$
$A := A + 3$
$w_2(A)$

(a) Give examples of a serialisable schedule and a non-serialisable schedule of the these transactions

(b) It turns out that both serial schedules have the same effect on any database, that is, (T_1, T_2) and (T_2, T_1) are semantically equivalent. Demonstrate this fact by showing the effect of the two transactions on an arbitrary initial database state (i.e., on a database where the values of items A and B are A_{init} and B_{init}. Modify the transactions so that semantic equivalence does not hold any more. For example, we can replace $B := B * 2$ by $B := B + 2$ to loose equivalence.

3. Consider the following two schedules:

   S$_1$: $w_3(A); r_1(A); w_1(B); r_2(B); w_2(C); r_3(C)$

   S$_2$: $r_1(A); r_2(A); r_3(B); w_1(A); r_2(C); r_2(B); w_2(B); w_1(C)$

   Answer the following questions:

   (a) What is the precedence graph for each of the schedules?

   (b) Is the schedule serialisable? If so, what are the equivalent serial schedules?

4. (*) Does view-equivalence to a serial schedule guarantee no dirty writes? No dirty reads? No non-repeatable reads? Justify your answers.

3 Recoverability concepts

1. For each of the following schedules, classify them recoverable, cascadeless, strict or non-recoverable:
   - $S_1 : r_1(X), w_1(X), c_1, r_2(X), w_2(X), c_2$
   - $S_2 : r_1(X), w_1(X), a_1, r_2(X), w_2(X), c_2$
   - $S_3 : r_1(X), r_2(X), w_1(X), c_1, w_2(X), c_2$
   - $S_4 : r_1(X), r_2(X), w_2(X), c_2, w_1(X), c_1$
   - $S_5 : r_1(X), w_1(X), r_2(X), w_2(X), c_2, c_1$
   - $S_6 : r_1(X), w_1(X), r_2(X), c_1, w_2(X), c_2$

4 Locks

1. What is the two-phase locking protocol?
2. If you are using transactions with binary locks, can you be sure that a schedule is serialisable? What if shared/exclusive locks?
3. If you are using the two-phase locking protocol, can you be sure that a schedule is serialisable?
4. In what cases can we have a deadlock? What are the approaches to this problem?
5. Consider two schedules:
   (a) $w_3(A); r_1(A); w_1(B); r_2(B); w_2(C); r_3(C)$
   (b) $r_1(A); r_2(A); r_3(B); w_1(A); r_2(C); r_2(B); w_2(B); w_1(C)$

   Assume that at transaction takes a (binary) lock on each database element immediately before it reads or writes the element, and that each transaction releases its locks immediately after the last time it accesses an element. Tell what the locking scheduler would do with each of these schedules—that is, transform these schedules to schedules that follow locking rules (i.e., show what request would get delayed, and when would they be allowed to resume)?
6. (*) Prove that two-phase locking always gives serialisable schedules (Hint: if a serialisability graph for a schedule has a cycle, at least one of the transactions in the schedule does not obey the two-phase locking protocol).