COMP3311 19T3 Database Systems

Exercises 09
Transaction Processing, Concurrency Control

1. [based on Ramakrishnan, ex.17.1]
   Give a brief definition for each of the following terms:

   a. transaction
   b. dirty read
   c. serializable schedule
   d. conflict-serializable schedule
   e. view-serializable schedule
   f. two-phase locking protocol

   [hide answer]

   a. *transaction*

   An execution of a user program that performs some action that is treated as atomic according to the semantics of some database application. The DBMS sees the transaction as a sequence of actions that can include read and write operations on the database, as well as computations.

   b. *dirty read*

   When a transaction reads an object that has been modified by another not-yet-commited transaction.

   c. *serializable schedule*

   A schedule over a set of transactions that produces a result that is the same as some serial execution of the transactions.

   d. *conflict-serializable schedule*

   A schedule is conflict-serializable if it is conflict-equivalent to some serial schedule. Two schedules are conflict-equivalent if they involve the same set of actions and they order every pair of conflicting actions in the same way.

   e. *view-serializable schedule*

   A schedule is view-serializable if it is view-equivalent to some serial schedule. Two schedules are view-equivalent if they satisfy:

   - the initial value of any object is read by the same transaction in both schedules, and
   - the final value of any object is written by the same transaction in both schedules, and
   - any shared object is written-then-read by the same pair of transactions in both schedules.

   f. *two-phase locking protocol*

   The two-phase locking protocol is a way of ensuring that only serializable schedules occur when transactions execute concurrently. Under this protocol each transaction must:

   - acquire a shared lock on an object before reading it
   - acquire an exclusive lock on an object before writing it
   - not acquire any new locks once it has released a lock

2. Draw the precedence graph for the following schedule (where `C` means "commit"):

```
T1:      R(A) W(Z)                C
T2:                R(B) W(Y)        C
T3: W(A)                     W(B)     C
```

[hide answer]

It has an edge from T3 to T1 (because of A) and an edge from T2 to T3 because of B.

This gives: `T2 --> T3 --> T1`

3. [based on Ramakrishnan, ex.17.2]
Consider the following incomplete schedule *S*:

```
T1: R(X) R(Y) W(X)           W(X)
T2:               R(Y)            R(Y)
T3:                    W(Y)
```

a. Determine (by using a precedence graph) whether the schedule is serializable

b. Modify *S* to create a complete schedule that is conflict-serializable

[hide answer]

a. *Determine (by using a precedence graph) whether the schedule is serializable*

The precedence graph has an edge, from *T1* to *T3*, because of the conflict between `T1:R(Y)` and `T3:W(Y)`. It also has an edge, from *T2* to *T3*, because of the conflict between the first `T2:R(Y)` and `T3:W(Y)`. It also has an edge, from *T3* to *T2*, because of the conflict between `T3:W(Y)` and the second `T2:R(Y)`.

b. *Modify S to create a complete schedule that is conflict-serializable*

Trick question. It is not possible. Since the precedence graph is cyclic, we know that it's not conflict-serializable.

If we are allowed to add `abort` actions (which was not mentioned in the question), we could simply abort either `T2` or `T3` and the schedule would become conflict-serializable.

4. [based on Ramakrishnan, ex.17.3]
For each of the following schedules, state whether it is conflict-serializable and/or view-serializable. If you cannot decide whether a schedule belongs to either class, explain briefly. The actions are listed in the order they are scheduled, and prefixed with the transaction name.

a. `T1:R(X) T2:R(X) T1:W(X) T2:W(X)`

b. `T1:W(X) T2:R(Y) T1:R(Y) T2:R(X)`

c. `T1:R(X) T2:R(Y) T3:W(X) T2:R(X) T1:R(Y)`

d. `T1:R(X) T1:R(Y) T1:W(X) T2:R(Y) T3:W(Y) T1:W(X) T2:R(Y)`

e. `T1:R(X) T2:W(X) T1:W(X) T3:W(X)`

[hide answer]

The methods used to determine these solutions:

- for conflict-serializablility, draw precedence graph and look for cycles
- for view-serializablility, apply the definition from lecture notes.

You can short-circuit the view serializability check. As soon as you know that the schedule is conflict-serializable, it must also be view serializable.

Solutions:

a. not conflict-serializable, not view-serializable

b. conflict-serializable, view-serializable

c. conflict-serializable, view-serializable

d. not conflict-serializable, not view-serializable

e. not conflict-serializable, view-serializable (view equivalent to the serial schedule T1, T2, T3)

5. Is the following schedule serializable? Show your working.

```
T1:             R(X)W(X)W(Z)        R(Y)W(Y)
T2: R(Y)W(Y)R(Y)            W(Y)R(X)        W(X)R(V)W(V)
```

[hide answer]

When we talk about serializability and don't specifically say what kind, we usually mean conflict-serializable. As above, the "working" for this question involves constructing a precedence graph, based on conflicting operations, and looking for cycles.

In this case there's a conflict between `T1:R(X)` and `T2:W(X)`, giving a graph edge from `T1` to `T2`. There's also a conflict between `T2:R(Y)` and `T1:W(Y)`, giving a graph edge from `T2` to `T1`. This means the graph has a cycle, so the schedule is not serializable.

6. Consider the following two transactions:

```
     T1              T2
------------    ------------
read(A)         read(B)
A := 10*A+4     B := 2*B+3
write(A)        write(B)
read(B)         read(A)
B := 3*B        A := 100-A
write(B)        write(A)
```

a. Write versions of the above two transactions that use two-phase locking.
b. Is there a non-serial schedule for *T1* and *T2* that is serializable? Why?
c. Can a schedule for *T1* and *T2* result in deadlock? If so, give an example schedule. If not, explain why not.

[hide answer]

a. *Write versions of the above two transactions that use two-phase locking.*

The basic idea behind two-phase locking is that you take out all the locks you need, do the processing, and then release the locks. Thus two-phase implementations of `T1` and `T2` would be:

```
     T1              T2
------------    ------------
write_lock(A)   write_lock(B)
read(A)         read(B)
A := 10*A+4     B := 2*B+3
write(A)        write(B)
write_lock(B)   write_lock(A)
read(B)         read(A)
B := 3*B        A := 100-A
write(B)        write(A)
unlock(A)       unlock(B)
unlock(B)       unlock(A)
```

b. *Is there a non-serial schedule for `T1` and `T2` that is serializable? Why?*

No. There is no such schedule. The last operation in `T1` is `write(B)`, and the last operation in `T2` is `write(A)`. `T1` starts with `read(A)` and `T2` starts with `read(B)`. Therefore, in any serializable schedule, we would require that either `read(A)` in `T1` should be after `write(B)` in `T2` or `read(B)` in `T2` should be after `write(B)` in `T1`.

c. *Can a schedule for T1 and T2 result in deadlock? If so, give an example schedule. If not, explain why not.*

Yes. Consider the following (where `L(X)` denotes taking a lock on object `X`):

```
T1: L(A)R(A)    W(A)L(B) ...
T2:         L(B)        R(B)W(B)L(A) ...
```