Exercise 5

Time tip: Roughly 45sec to 1min per 1pt
Q1) You are given a database that has 2 distinct objects A and B. The following 3 transactions have to be scheduled concurrently.

T1: R(A), W(B), R(A) Commit
T2: W(A), W(B), Commit
T3: R(B), W(A), Commit

For all Q, many correct answers can exist; the schedules I provide are just some correct examples. Verify yourself if yours is correct.

A. [5pts] Produce an “interleaved” schedule (i.e., not serial in the reads/writes) that is not serializable.

T2-W(A), T1-R(A), T1-W(B), T2-W(B), T1-R(A), T1-Commit, T2-Commit, all of T3

The first part is equivalent to neither of T1->T2 nor T2->T1 due to the WW conflict on B and WR conflict on A
Exercise

Q1) You are given a database that has 2 distinct objects A and B. The following 3 transactions have to be scheduled concurrently.

T1: R(A), W(B), R(A) Commit
T2: W(A), W(B), Commit
T3: R(B), W(A), Commit

B. [2 x 5pts] Produce 2 different interleaved schedules that are both serializable.

1) T1-R(A), T1-W(B), T3-R(B), T1-R(A), T1-Commit, T3-W(A), T3-Commit, all of T2
   Equivalent to T1->T3->T2

2) T2-W(A), T1-R(A), T2-W(B), T2-Commit, T1-W(B), T3-R(B), T1-R(A), T1-Commit, T3-W(A), T3-Commit
   Equivalent to T2->T1->T3
Exercise

Q1) You are given a database that has 2 distinct objects A and B. The following 3 transactions have to be scheduled concurrently.

T1: R(A), W(B), R(A) Commit
T2: W(A), W(B), Commit
T3: R(B), W(A), Commit

C. [6pts] Produce an interleaved schedule that is serializable but not recoverable.

T1-R(A), T1-W(B), T3-R(B), T1-R(A), T3-W(A), T3-Commit, T1-Commit, all of T2
Equivalent to T1->T3->T2; so, serializable
T1 supplies dirty B to T3 and T3 commits before T1; so, not recoverable
Exercise

Q1) You are given a database that has 2 distinct objects A and B. The following 3 transactions have to be scheduled concurrently.

T1: R(A), W(B), R(A) Commit
T2: W(A), W(B), Commit
T3: R(B), W(A), Commit

D. [6pts] Produce an interleaved schedule that is both serializable and recoverable.

T2-W(A), T1-R(A), T2-W(B), T2-Commit, T1-W(B), T3-R(B), T1-R(A), T1-Commit, T3-W(A), T3-Commit

Equivalent to T2->T1->T3; so, serializable
T2 supplies dirty A to T1 but commits before T1; T1 supplies dirty B to T3 but commits before; so, recoverable
Exercise

Q2) You are given a database that has 3 distinct objects A, B, and C. The following 3 transactions have to be scheduled concurrently. Suppose T1 arrives just before T2, which arrives just before T3.

- T1: R(A), W(A), R(C) Commit
- T2: R(A), W(B), W(A), Commit
- T3: R(C), W(C), Commit

A. [6pts] Produce a serializable but not recoverable schedule that arises if you use the 2PL protocol and request an appropriate lock on an object only when needed.

  T1 gets X lock on A (since it needs R-W access to A); T2 requests X lock A (needs R-W access) and is made to wait on T1; T3 gets X lock on C (needs R-W access); T1 requests S lock on C and is made to wait on T3; T3 commits and releases lock on C; T1 resumes, gets S lock on C, releases X lock on A; T2 comes in and gets X lock on A, gets X lock on B, goes to commit; T1 resumes, reads C, goes to commit.

So overall: T1-R(A), T3-R(C), T1-W(A), T3-W(C), T3-Commit, T2-R(A), T2-W(B), T2-W(A), T2-Commit, T1-R(C), T1-Commit

Schedule is serializable and equivalent to T3->T1->T2. But it is not recoverable because T2 reads dirty A from T1 and commits before T1.
Exercise

Q2) You are given a database that has 3 distinct objects A, B, and C. The following 3 transactions have to be scheduled concurrently. Suppose T1 arrives just before T2, which arrives just before T3.

T1: R(A), W(A), R(C) Commit
T2: R(A), W(B), W(A), Commit
T3: R(C), W(C), Commit

B. [6pts] Produce a schedule that arises if you use the strict 2PL protocol instead in the above. Is it serializable? Is it recoverable?

T1 gets X lock on A (needs R-W access); T2 requests X lock A (needs R-W access) and is made to wait on T1; T3 gets X lock on C (needs R-W access); T1 requests S lock on C and is made to wait on T3; T3 commits and releases lock on C; T1 resumes, gets S lock on C, and goes to commit and releases its locks on C and A; T2 resumes, gets X lock on A, then gets X lock on B, goes to commit.

So overall: T1-R(A), T3-R(C), T1-W(A), T3-W(C), T3-Commit, T1-R(C), T1-Commit, T2-R(A), T2-W(B), T2-W(A), T2-Commit

Schedule is serializable and equivalent to T3->T1->T2. And it is also recoverable because no txn gets dirty data from another uncommitted txn.
Exercise

Q3) You are given a database that has 2 distinct objects A and B and the following 2 transactions.

$$\begin{align*}
\text{T1: } & R(A), W(A), R(A), W(B), \text{ Commit} \\
\text{T2: } & R(B), W(B), R(A), \text{ Commit}
\end{align*}$$

A. [6pts] Produce an interleaved schedule with at least one WW conflict that is still serializable, if possible.

$$\begin{align*}
\text{T2-R(B), T1-R(A), T2-W(B), T2-R(A), T1-W(A), T1-R(A), T1-W(B), T2-Commit, T1-Commit}
\end{align*}$$

WW conflict on B; but serializable and eq. to T2->T1
Exercise

Q3) You are given a database that has 2 distinct objects A and B and the following 2 transactions.

\[
\begin{align*}
T1: & \text{ R(A), W(A), R(A), W(B), Commit} \\
T2: & \text{ R(B), W(B), R(A), Commit}
\end{align*}
\]

B. [6pts] Produce an interleaved schedule with at least one WR conflict that is still serializable, if possible.

Not possible. On B: T2 writes it but T1 does not read it (only a blind write); T1 also writes it but at the end and T2 reads it but at the beginning—so, the schedule is not “interleaved” any more.

On A: T1 alone writes it and T2 reads it; but T1 also writes B, which means if T1-W(A) goes before T2-R(A), T1-W(B) must also go before T2-R(B) for serializability; once again this makes the schedule not “interleaved” any more.
Exercise

Q3) You are given a database that has 2 distinct objects A and B and the following 2 transactions.

T1: R(A), W(A), R(A), W(B), Commit
T2: R(B), W(B), R(A), Commit

C. [6pts] Produce an interleaved schedule with at least one RW conflict that is still serializable, if possible.

Not possible. On A: Only T1 writes it; T2 reads it but only once at the end, which renders it not a conflict. On B: Only T2 reads it; T1 writes it and putting T1-W(B) in between T2-R(B) and T2-W(B) does lead to an RW conflict but any schedule with such an ordering of these ops will also not be serializable.
Exercise

Q4) You are given a database that has 2 distinct objects A and B and the following 2 transactions.

A. [6pts] Produce an interleaved schedule if possible that arises if you use the READ UNCOMMITTED isolation level of SQL. Is it serializable?

Long X locks and no S locks.
At the start, T1 gets X lock on A (needs RW access) and T2 gets X lock on B (likewise); T1 runs R(A), W(A), R(A); T2 runs R(B), W(B); T1 then requests X lock on B and is made to wait on T2; T2 runs R(A), commits, releases its lock on B; T1 gets X lock on B, runs W(B), commits.
So, schedule is: T1-R(A), T1-W(A), T1-R(A), T2-R(B), T2-W(B), T2-R(A), T2-Commit, T1-W(B), T1-Commit.
Yes, it is serializable and eq. to T2->T1.
Q4) You are given a database that has 2 distinct objects A and B and the following 2 transactions.

   T1: R(A), W(A), R(A), W(B), Commit
   T2: R(B), W(B), R(A), Commit

B. [6pts] Produce an interleaved schedule if possible that arises if you use the READ COMMITTED isolation level of SQL. Is it serializable?
   Long X locks and short S locks.
   At the start, T1 gets X lock on A (needs RW access) and T2 gets X lock on B (likewise); T1 runs R(A), W(A), R(A); T2 runs R(B), W(B); T1 then requests X lock on B and is made to wait on T2; T2 then requests S lock on A but is also made to wait on T1; so, it ends up in a deadlock.

C. [6pts] Produce an interleaved schedule if possible that arises if you use the SERIALIZABLE isolation level of SQL. Is it serializable?
   Long X locks and long S locks.
   Same answer as 4.B. It ends up in a deadlock!
Exercise

Q5) You are given a database that has 3 distinct objects A, B, and C and the following 3 transactions.

T1: R(A), W(A), R(B), W(B), R(C), W(C), Commit
T2: R(B), W(B), R(C), W(C), R(A), W(A), Commit
T3: R(C), W(C), R(A), W(A), R(B), W(B), Commit

A. [3pts] Suppose you follow strict 2PL protocol and request an appropriate lock on an object only when needed. Will it lead to a deadlock situation?
Yes. T1/T2/T3 get long X lock A/B/C respectively at the start; T1 then requests X lock on B but is made to wait on T2; T2 then requests X lock on C but is made to wait on T3; T3 then requests X lock on A but is made to wait on T1. So, there is a cyclic waiting behavior and a deadlock.

B. [3pts] What if you used 2PL protocol instead in the above?
Does not matter; same answer as 5.A because the deadlock occurs during the lock acquisition phase, which is identical for 2PL and strict 2PL.