COMP0022: Database and information systems (PG – Term 1)

Transaction management exercises

Part 1: Concurrency

1. Are the following interleavings conflict serializable or not? Explain your answer. If the interleavings are serializable, what is the equivalent serial schedule?

    a. T1: Read(x)
    T2: Read(x)
    T1: Write(x)
    T2: Write(y)

    Conflict serializable. Lines 2 and 3 force a T2 → T1 precedence. However, the last line involves a different piece of data. The graph is therefore acyclic. The equivalent serial schedule is executing T2 in full, then T1.

    b. T1: Read(A)
    T1: Write(A)
    T2: Read(B)
    T2: Write(B)
    T2: Read(A)
    T2: Write(A)
    T1: Read(B)
    T1: Write(B)

    Not conflict serializable. Lines 1 & 6 and 2 & 5 both force a T1 → T2 precedence. Lines 3 & 8 and 4 & 7 both force a T2 → T1 precedence. The graph has a cycle, so we cannot generate a serial schedule.

    c. T1: Read(x)
    T2: Read(y)
    T1: Write(x)
    T3: Read(y)
    T2: Write(y)
    T3: Read(z)
    T3: Write(z)

    Conflict serializable. Note that T1 is the only one that touches x, and T3 is the only one that touches z. However, both T2 and T3 touch y, and lines 4 and 5 force a T3 → T2 precedence. The graph is acyclic and not fully connected. This means that T1 can happen anywhere in equivalent serial orderings, just as long

as T3 precedes T2. So an equivalent serial ordering would be T1, T3, T2. Another would be T3, T1, T2, etc.

2. At what point in the following interleavings (if ever) should a timestamp-based concurrency manager ask a transaction to abort and restart, and which transaction? If the interleavings are serializable, what is the equivalent serial schedule? Note: the numbers of the transactions (e.g. T1) are their timestamps.

    a. T1: Read(x)
    T1: Write(x)
    T2: Read(x)
    T2: Read(y)
    T3: Write(x)
    T1: Read(y)
    T2: Write(x)
    T2: Write(y)

    Using the ignore obsolete write rule, the concurrency manager would not ask any transaction to abort. However, it would also NOT perform line 7. That is to say, the above interleaving without line 7 is equivalent to the serial schedule in timestamp order: T1, T2, T3. Note that this is a view serializable schedule featuring blind writes. If you choose to leave line 7 in, line 7 fails and T2 would be asked to abort and restart. The other lines cause no issues.

    b. T1: Read(A)
    T2: Read(A)
    T2: Write(A)
    T3: Read(A)
    T1: Write(A)
    T3: Write(B)
    T2: Read(B)

    An issue occurs at line 5 when T1 attempts to write A, which has already been read by two later transactions: T2 and T3. T1 would be asked to abort and restart. No equivalent serial schedule.

Part 2: Recovery

3. Based on the contents of the following log, what work needs to be done to restore the database to a consistent state?

Tid	Operation	Object	Before-image	After-image	Op #
T27	INSERT	STAFF SG90		(a, b, c)	1
T29	COMMIT				final
T32	UPDATE	STAFF SG12	(d, e, f)	(d, e, g)	1
T32	UPDATE	PROPERTY P290	(x, y)	(z, w)	2
	CHECKPOINT	T27, T32			
T35	DELETE	CUSTOMER C124	(a, b, c, d)		1
T35	COMMIT				final
T32	UPDATE	PROPLEASE L2903	(e, f, g, h)	(e, f, i, h)	3
T36	INSERT	CUSTOMER C124		(k, l, m, n)	1
T32	COMMIT				final
T27	UPDATE	PROPERTY P309	(a, b)	(c, b)	2
End of log.					

Since the checkpoint, T32 and T35 have committed. Their work has to be redone by applying their changes in forward order: updating STAFF SG12 to (d, e, g), updating PROPERTY P290 to (z, w), deleting CUSTOMER C124, and updating PROPLEASE L2903 to (e, f, i, h) – if those things haven't already happened.

Operations that were active at the time of checkpoint and operations that started after the checkpoint but did not commit have to be undone. Therefore, T27 and T36's changes have to be undone in backwards order: changing PROPERTY P309 back to (a, b), removing CUSTOMER C124, and removing STAFF SG90.

The above changes will result in the database being in a consistent state.