Chapter 14: Transactions

Edited by Radhika Sukapuram

Database System Concepts, 6^{th} Ed.

Conflicting Instructions

- Let I_i and I_j be two Instructions of transactions T_i and T_j respectively.
- Instructions I_i and I_j **conflict** if and only if
  - there exists some item Q accessed by both I_i and I_j,
  - and at least one of these instructions wrote Q.

  1. I_i = **read**(Q), I_j = **read**(Q). I_i and I_j don't conflict.
  2. I_i = **read**(Q), I_j = **write**(Q). They conflict.
  3. I_i = **write**(Q), I_j = **read**(Q). They conflict
  4. I_i = **write**(Q), I_j = **write**(Q). They conflict
- Intuitively, a conflict between I_i and I_j forces a (logical) temporal order between them.
  - If I_i and I_j are consecutive in a schedule and they do not conflict,
    - their results would remain the same even if they had been interchanged in the schedule.

Conflict Serializability

- If a schedule *S* can be transformed into a schedule *S'*
  - by a series of swaps of non-conflicting instructions,
  - we say that *S* and *S'* are **conflict equivalent**.
- We say that a schedule *S* is **conflict serializable** if
  - it is conflict equivalent to a serial schedule

Conflict Serializability (Cont.)

- Schedule 3 can be transformed into Schedule 6 –
  - a serial schedule where T_2 follows T_1,
    - by a series of swaps of non-conflicting instructions.
  - Therefore, Schedule 3 is conflict serializable.

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Schedule 3

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Schedule 6

Are schedules 1 and 2 conflict equivalent?

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

Schedule 1

T_1	T_2
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	

Schedule 2

Conflict Serializability (Cont.)

- Example of a schedule that is not conflict serializable:

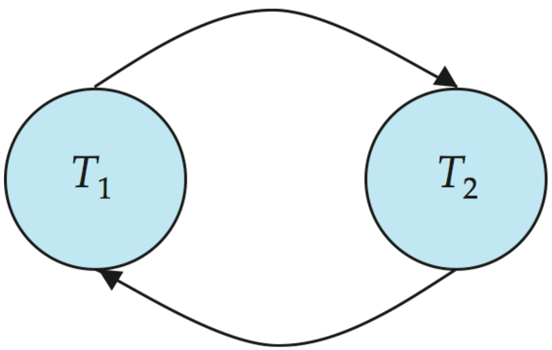
T_3	T_4
read (Q)	
	write (Q)
write (Q)	

- We are unable to swap instructions in the above schedule to obtain
  - either the serial schedule < T_3, T_4 >,
  - or the serial schedule < T_4, T_3 >.
- How to efficiently determine conflict serializability of a schedule ?

Precedence Graph

- Consider some schedule of a set of transactions T_1, T_2, ..., T_n
- **Precedence graph** — a directed graph where the vertices are the transactions (names).
- We draw an arc from T_i to T_j if the two transactions conflict, and T_i accesses the data item first on which the conflict arose (RW, WR, WW)
- We may label the arc by the item that was accessed.

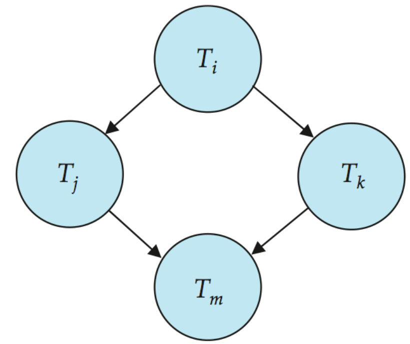
Precedence graph cont.



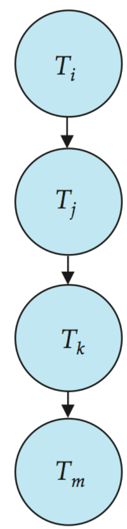
T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

Testing for Conflict Serializability

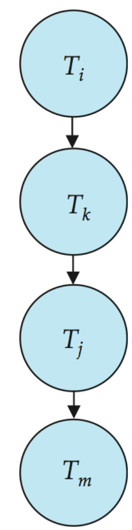
- A schedule is conflict serializable if and only if its precedence graph is acyclic.
- Cycle-detection algorithms exist
  - (order (*n* + *e*) where *e* is the number of edges, n the number of vertices.)
- If precedence graph is acyclic, the **serializability order** can be obtained by a *topological sorting* of the graph.
  - For example, a serializability order for the schedule (a) would be one of either (b) or (c)



(a)



(b)



(c)

Transaction failures

- The discussion on serializability assumes that there are no transaction failures
- Suppose there are transaction failures
  - The effect must be undone
  - A dependent transaction must also be aborted
- What is the impact of this ?

Recoverable Schedules

- The following schedule is not recoverable if T_9 commits immediately after the read(A) operation.

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

- If T_8 should abort,
  - T_9 would have read (and possibly shown to the user) an inconsistent database state.
  - Hence, a database must ensure that schedules are recoverable.
- **Recoverable schedule** — if a transaction T_j reads a data item previously written by a transaction T_i ,
  - then the commit operation of T_i **must** appear before the commit operation of T_j

Cascading Rollbacks

- **Cascading rollback** – a single transaction failure leads to a series of transaction rollbacks.
- Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

  - If T_{10} fails, T_{11} and T_{12} must also be rolled back.
- Can lead to the undoing of a significant amount of work

Cascadeless Schedules

- **Cascadeless schedule** — one where for each pair of transactions T_i and T_j such that
  - T_j reads a data item previously written by T_i,
    - the commit operation of T_i appears before the read operation of T_j.
- Every cascadeless schedule is also recoverable
- It is desirable to restrict the schedules to those that are cascadeless
- Example of a schedule that is NOT cascadeless

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

Concurrency Control

- A database must provide a mechanism that will ensure that all possible schedules are both:
  - Conflict serializable.
  - Recoverable and preferably cascadeless
- A policy in which only one transaction can execute at a time generates serial schedules, but provides a poor degree of concurrency
- Concurrency-control schemes tradeoff between the amount of concurrency they allow and the amount of overhead that they incur
- Tests for serializability help us understand why a concurrency control protocol is correct
- **Goal** – to develop concurrency control protocols that will ensure serializability.

Weak Levels of Consistency

- Some applications are willing to live with weak levels of consistency, allowing schedules that are not serializable
  - E.g., a read-only transaction that wants to get an approximate total balance of all accounts
  - Such transactions need not be serializable with respect to other transactions
- Tradeoff accuracy for performance

Weak levels of consistency cont.

- Dirty read : A transaction T_j reads an uncommitted data item previously written by T_i
- Dirty write : A transaction T_j writes an uncommitted data item previously written by T_i

Weak levels of consistency cont.

- Phantom read: A transaction T_j reads a data item (a phantom) that was inserted into the database by T_i
  - T_{30}: **select count**(*)
    from *instructor*
    where *dept_name* = 'Physics';
  - T_{31} : **insert into** *instructor*
    values (1111, 'Feynman', 'Physics', 94000);
  - If T_{30} reads the value written by T_{31}, in a serializable schedule, T_{30} must come after T_{31}. Else T_{30} must come before T_{31}
  - Conflict is on predicates, not on the data item itself
    - Information used to find tuples must also be considered for concurrency control
  - If concurrency is performed at tuple granularity, the inserted tuple may go undetected – this is called the phantom phenomenon

Levels of Consistency in SQL-92

- **Serializable** — default [may result in non-serializable executions in some DBMSs]
- **Repeatable read** —
  - only committed records to be read, (no dirty reads)
  - repeated reads of same record must return same value. (no non-repeatable reads)
  - However, a transaction may not be serializable – it may find some records inserted by a transaction but not find others (phantoms allowed).
    - T1 may see some records inserted by T2, but may not see others inserted by T2
- **Read committed** — only committed records can be read (no dirty reads), but successive reads of record may return different (but committed) values (non-repeatable reads and phantom reads allowed).
- **Read uncommitted** — even uncommitted records may be read.
- **All the levels above additionally disallow dirty writes**
- Syntax: **set transaction isolation level <isolation_level >;**

Transaction Definition in SQL

- Data manipulation languages must include a construct for specifying the set of actions that comprise a transaction.
- In SQL99, a transaction begins with **begin**
- A transaction in SQL ends by:
  - **commit work** commits current transaction and begins a new one.
  - **rollback work** causes current transaction to abort.
- In almost all database systems, by default, every SQL statement also commits implicitly if it executes successfully
  - Implicit commit can be turned off by a database directive
    - E.g. in JDBC, connection.setAutoCommit(false);

Other Notions of Serializability

View Serializability

- Let S and $S´$ be two schedules with the same set of transactions. S and $S´$ are **view equivalent** if the following three conditions are met, for each data item Q,
  1. If in schedule S, transaction T_i reads the initial value of Q, then in schedule S' also transaction T_i must read the initial value of Q.
  2. If in schedule S transaction T_i executes **read**(Q), and that value was produced by transaction T_j (if any), then in schedule S' also transaction T_i must read the value of Q that was produced by the same **write**(Q) operation of transaction T_j .
  3. The transaction (if any) that performs the final **write**(Q) operation in schedule S must also perform the final **write**(Q) operation in schedule S'.
- As can be seen, view equivalence is also based purely on **reads** and **writes** alone.

View Serializability (Cont.)

- A schedule *S* is **view serializable** if it is view equivalent to a serial schedule.
- Every conflict serializable schedule is also view serializable.
- Below is a schedule which is view-serializable but *not* conflict serializable.

T_{27}	T_{28}	T_{29}
read (Q)		
	write (Q)	
write (Q)		
		write (Q)

- What serial schedule is above equivalent to? <T27, T28, T29>
- Every view serializable schedule that is not conflict serializable has **blind writes (writes without prior reads)**
- The problem of checking if a schedule is view serializable falls in the class of *NP*-complete problems.

More Complex Notions of Serializability

- The schedule below produces the same outcome as the serial schedule $< T_1, T_5 >$, yet is not conflict equivalent or view equivalent to it.

T_1	T_5
read (A) $A := A - 50$ write (A)	
	read (B) $B := B - 10$ write (B)
read (B) $B := B + 50$ write (B)	
	read (A) $A := A + 10$ write (A)

- If we start with A = 1000 and B = 2000, the final result is 960 and 2040
- Determining such equivalence requires analysis of operations other than read and write, ie analysis of what computations are performed.
- Conclusion : There are less stringent definitions of schedule equivalence than conflict/view equivalence

End of Chapter 14

Edited by Radhika Sukapuram

Database System Concepts, 6th Ed.