Introduction to Data Management CSE 344

Section 10: Transactions Review

Announcements

- HW8 is due tomorrow

- Final Exam
  - Tuesday March 18, 2:30-4:30 pm, in class

ACID Properties

A DBMS guarantees the following four properties of transactions:

- Atomic
  - State shows either all the effects of txn, or none of them
- Consistent
  - Txn moves from a state where integrity holds, to another where integrity holds
- Isolated
  - Effect of txns is the same as txns running one after another (ie looks like batch mode)
- Durable
  - Once a txn has committed, its effects remain in the database

Serial Schedules

A *serial* schedule is one in which transactions are executed one after the other, in serial order

Serializable Schedule

A schedule is *serializable* if it is equivalent to a serial schedule

We want to ensure this instead of serial schedule!

Conflicts

Conflicts: pair of actions (in order) in schedule s.t. if swapped, then behavior changes.

Two actions by same transaction T_i: $r_i(X); w_i(Y)$

Two writes by T_i, T_j to same element $w_i(X); w_j(X)$

Read/write by T_i, T_j to same element $w_i(X); r_j(X)$

$r_i(X); w_j(X)$

Note: any #actions can appear between them

Conflict Serializability

- A schedule is *conflict serializable* if it can be transformed into a serial schedule by a series of swappings of adjacent non-conflicting actions
- Stronger condition than serializability
- How do we check for conflict serializability?
  - Using Precedence Graph

Locks

Major differences between vendors:

- Lock on the entire database
  - SQLite
- Lock on individual records
  - SQL Server, DB2, etc

Two Phase Locking (2PL)

The 2PL rule:

In every transaction, all lock requests must precede all unlock requests

Strict 2PL

The Strict 2PL rule:

All locks are held until the transaction commits or aborts.

Isolation Levels in SQL

1. "Dirty reads"
   SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

2. "Committed reads"
   SET TRANSACTION ISOLATION LEVEL READ COMMITTED

3. "Repeatable reads"
   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

4. Serializable transactions
   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

NOTE: Only valid for that transaction
SQL Server: http://technet.microsoft.com/en-us/library/jj856598.aspx

Consider a database consisting of a single relation

R:

A	B
1	10
2	0

The following two transactions run concurrently on this database

Line	T1	T2
1	begin transaction;	begin transaction;
2	update R set B = B-10 where A=1;	select sum(B) from R;
3	update R set B = B+10 where A=2;	commit;
4	commit;	

Is it ever possible for T2 to see a value of zero in its output?

Answer (Discuss if a schedule where T2 sees a zero value is possible):

Solution: If transaction T2 runs using the READ UNCOMMITTED level of isolation, it may be allowed to read any intermediate value that T1 writes. If it happens that T2 executes its select statement in between the two update statements of T1, it may see the intermediate zero value.

Of course, whether the problem actually arises depends on the DBMS implementation. SQL Server is an example of a DBMS where T2 will read the intermediate, zero value, if it runs using the READ UNCOMMITTED isolation level and executes its select statement in between T1's update statements.

Time	T1	T2	T3
1	begin transaction;		
2	select * from R;		
3		begin transaction;	
4		select * from R where $A = 2$;	
5	update R set $B = 30$ where $A = 2$;		
6		select * from R where $A = 2$;	
7	commit;		
8			begin transaction;
9			select * from R where $A = 2$;
10		commit;	
11			
12			
13			commit;

Consider SQLite and SQL Server.

For each command issued by a transaction, indicate one of the following outcome:

- SUCCESS
- WAIT (indicate resume time)
- ERROR (indicate retry time)

Review: SQLite and SQL Server locking rules from Lectures 20, 21

SQLite

Time	T1	T2	T3
1	begin transaction;		
2	select * from R;		
3		begin transaction;	
4		select * from R where $A = 2$;	
5	update R set $B = 30$ where $A = 2$;		
6		select * from R where $A = 2$;	
7	commit;		
8			begin transaction;
9			select * from R where $A = 2$;
10		commit;	
11			
12			
13			commit;

SUCCESS (1,10),(2,0)

SUCCESS (2,0)

SUCCESS

SUCCESS (2,0)

ERROR

Retry after T2 commit

ERROR

Retry after T1 commit

SUCCESS

Retry commit

SUCCESS

Retry read

SUCCESS (2,30)

Serialization order: T2, T1, T3

SQL Server – Read Committed

Time	T1	T2	T3
1	begin transaction;		
2	select * from R;		
3		begin transaction;	
4		select * from R where $A = 2$;	
5	update R set $B = 30$ where $A = 2$;		
6		select * from R where $A = 2$;	
7	commit;		
8			begin transaction;
9			select * from R where $A = 2$;
10		commit;	
11			
12			
13			commit;

SUCCESS (1,10),(2,0)

SUCCESS (2,0)

SUCCESS

WAIT until T1 commit

SUCCESS

Rsume read SUCCESS (2,30)

SUCCESS (2,30)

SUCCESS

SUCCESS