
Transactions in SQL, View and Conflict Serializability

28 February 2011
Lecture 2

Topics for Today

- Transaction Support in SQL
- 2PL, Serializability, Recoverability
 - Conflict Equivalence
 - Conflict Serializability
 - View Equivalence
 - View Serializability
- Review of Transaction Properties
- Source: R&G 16.6, 17.1

Topics for Today

- Transaction Support in SQL
- 2PL, Serializability, Recoverability
 - Conflict Equivalence
 - Conflict Serializability
 - View Equivalence
 - View Serializability
- Review of Transaction Properties
- Source: R&G 16.6, 17.1

Phantoms

T1:

```
SELECT S.rating,  
AVG(S.age) FROM Sailors  
S WHERE S.rating = 8
```

```
SELECT S.rating,  
AVG(S.age) FROM Sailors  
S WHERE S.rating = 8
```

T2:

```
INSERT INTO Sailors  
  (sid, sname, rating,  
  age) VALUES (128,  
  'Norbert', 8, 10)  
  
COMMIT;
```

- Row level locking won't solve this issue - Phantom rows
- What **should** we do?
 - Should T1 see Norbert?
 - Should T2 be allowed to perform the write?

Dealing with Phantoms

One option: Lock all rows which might satisfy the constraint

- Meaning that T2 not be able to create the Norbert row
- Called “Index Locking”

A second option: Lock the entire table for T_1

- Bad performance

A third option: Allow T1 to see Norbert

- Before T2 commits?
- After T2 commits?

The user can **choose** what to do here using SQL commands

Setting Transaction Characteristics

SQL enables definition of transaction characteristics per transaction

- **Access Mode**
 - Determines whether the transaction will read or read/write
 - Two parameters are
 - READ ONLY – can't update, delete, create, insert.
 - Can't get exclusive locks
 - READ WRITE
- **Diagnostic Size**
 - The number of error conditions to be recorded
- **Isolation Level**
 - More on that...


Isolation Level

- How the transaction is shielded from the effects of other transactions

Level	Dirty Read	Unrepeatable Read	Phantom
READ UNCOMMITTED	Maybe	Maybe	Maybe
READ COMMITTED	No	Maybe	Maybe
REPEATABLE READ	No	No	Maybe
SERIALIZABLE	No	No	No

Level: Serializable


- Most conservative:
 - Obtains locks before reading or writing
 - Ensures the transaction only reads from committed transactions
 - Nothing read or written by it can be modified
 - Any search conditions which the transaction uses cannot be modified by another transaction (no phantoms)

Level	Dirty Read	Unrepeatable Read	Phantom
READ UNCOMMITTED	Maybe	Maybe	Maybe
READ COMMITTED	No	Maybe	Maybe
REPEATABLE READ	No	No	Maybe
 SERIALIZABLE	No	No	No

Level: Repeatable Read


- Second most conservative:
 - Obtains locks before reading or writing
 - Ensures the transaction only reads from committed transactions
 - Nothing read or written by it can be modified
 - Other transactions **may add phantoms** which violate search conditions

Example: T1 searched for all Sailors where rating = 8. T2 may add Norbert with rating = 8. T1 won't see Norbert until T2 commits.

Level	Dirty Read	Unrepeatable Read	Phantom
READ UNCOMMITTED	Maybe	Maybe	Maybe
READ COMMITTED	No	Maybe	Maybe
 REPEATABLE READ	No	No	Maybe
SERIALIZABLE	No	No	No

Level: Read Committed

- A bit less conservative:
 - Obtains locks before reading or writing
 - Ensures that transaction only reads from committed transactions
 - Nothing written by it can be modified
 - Values **read by it may be modified**
 - Transaction acquires shared locks to read, but **immediately releases them**
 - Other transactions may add phantoms which violate search conditions

Level	Dirty Read	Unrepeatable Read	Phantom
READ UNCOMMITTED	Maybe	Maybe	Maybe
 READ COMMITTED	No	Maybe	Maybe
REPEATABLE READ	No	No	Maybe
SERIALIZABLE	No	No	No

Level: Read Uncommitted

- Most liberal:
 - Does **not obtain shared locks**
 - Reads objects written by **uncommitted** transactions
 - Objects may change
 - Never makes lock requests
- So liberal, only **READ ONLY** transactions can be declared with it

Level	Dirty Read	Unrepeatable Read	Phantom
→ READ UNCOMMITTED	Maybe	Maybe	Maybe
READ COMMITTED	No	Maybe	Maybe
REPEATABLE READ	No	No	Maybe
SERIALIZABLE	No	No	No

Setting the Levels

- By default all transactions start with SERIALIZABLE READ WRITE
 - Most conservative, can do anything
- Can be modified using

SET TRANSACTION ISOLATION LEVEL {SERIALIZABLE |
REPEATABLE READ | READ COMMITTED | READ
UNCOMMITTED} [READ ONLY | READ WRITE]

- Example: SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
READ ONLY
 - Example: SET TRANSACTION ISOLATION LEVEL REPEATABLE
READ READ ONLY
 - You can change them during the course of the transaction
- All Isolation levels supported by MS SQL.

Transactions and Vendors

- Varies by the database engine used
- In MySQL:
 - MyISAM engine does not support transaction
 - The InnoDB engine supports full ACID transaction management
- MS SQL Server: Full transaction support
 - Read Uncommitted can be declared by READ WRITE transactions
 - Additional Isolation level: SNAPSHOT
- Oracle: Full transaction support

So Far

- Transaction Support in SQL
- 2PL, Serializability, Recoverability
 - Conflict Equivalence
 - Conflict Serializability
 - View Equivalence
 - View Serializability
- Review of Transaction Properties

Conflict Equivalence

- Two properties
 - Serializability
 - Recoverability
- Two schedules are conflict equivalent if
 - They involve the same sets of actions of same transactions
 - They order every pair of conflicting actions of two committed transactions in the same way

Conflict Equivalence

- Two actions conflict if
 - They operate on the same object
 - At least one is a write
- Outcome of a schedule depends on order of conflicting actions
 - Non-conflicting actions can be rearranged at will without changing the outcome
- Two schedules which are conflict equivalent will have the same outcome on the database
 - We can also derive one from the other by swapping around non-conflicting operations

Conflict Serializability

- A schedule is conflict serializable if it's conflict equivalent to some serial schedule

- Every conflict serializable schedule is serializable
 - not every serializable schedule is conflict serializable
 - Provided that the objects in the database aren't added or deleted

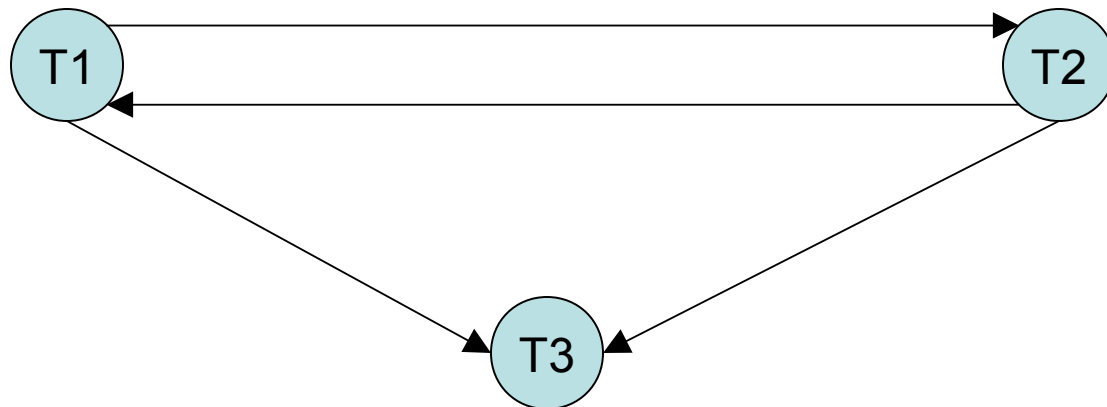
Conflict Serializability

- Example: The schedule below is serializable but not conflict serializable

T_1	T_2	T_3
R(A)	W(A) Commit	
W(A) Commit		
		W(A) Commit

Precedence Graphs

- We can represent conflicts between transactions using a precedence graph
 - Also called a serializability graph
 - Each node is a committed transaction
 - An arc goes from T_i to T_j if an action of T_i precedes and conflicts with one action of T_j
- The precedence graph for previous example



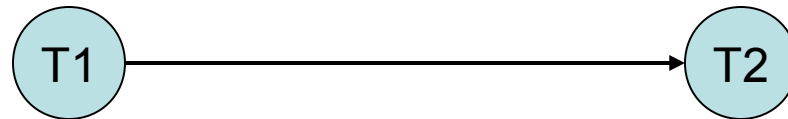
Precedence Graphs

- Example: Another example of a transaction schedule

T_1	T_2
X(A)	
R(A)	
W(A)	
X(B)	
R(B)	
W(B)	
Commit	
	X(A)
	R(A)
	W(A)
	X(B)
	R(B)
	W(B)
	Commit

Precedence Graphs

- Precedence graph
 - T1 uses the same objects A and B that T2 uses



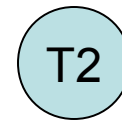
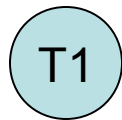
Precedence Graphs

- Example: Interleaving with Strict 2PL Locks

T_1	T_2
S(A)	
R(A)	
	S(A)
	R(A)
	X(B)
	R(B)
	W(B)
	Commit
X(C)	
R(C)	
W(C)	
Commit	

Precedence Graphs

- Precedence graph
 - T1 and T2 both read A, but since neither is a write, there is no conflict



Precedence Graphs & Conflict Serializability

- We can use precedence graphs to determine conflict serializability
- **Any precedence graph which is acyclic is conflict serializable**
- **A serial, conflict equivalent schedule is given by any topological sort of the graph**

Strict Two Phase Locking

- Strict 2PL has two rules:
 - If a transaction T wishes to read (respectively, modify) an object, it first requests and a shared (respectively, exclusive) lock on the object.
 - All the locks held by a transaction are released when the transaction is completed
- We can prove that any schedule allowed by Strict 2PL is conflict serializable because:
 - A schedule S is conflict serializable if and only if its precedence graph is acyclic
 - Strict 2PL ensures that the precedence graph for any schedule that it allows is acyclic.

Two Phase Locking

- A variation of Strict 2PL is Two Phase Locking (2PL)
- 2PL replaces the second rule with:
 - A transaction cannot request additional locks once it releases *any* lock
- Transactions have a *growing phase* and a *shrinking phase*
- We can prove that 2PL also permits only acyclic precedence graphs
 - *Every schedule allowed by 2PL is conflict serializable*
- Intuitively, transactions are ordered by the order in which they enter their shrinking phase
 - We still need to show there aren't any cycles

Strict Schedules

- A **strict** schedule is one where any object written by T is not read or overwritten until T completes or aborts
- Strict schedules are
 - Recoverable
 - Prevent Cascading Aborts
 - Aborted transactions can be undone by just restoring the old values

Strict Schedules

- Strict 2PL adds on to 2PL the properties of being strict.
 - Hence the name

Examples

T_1	T_2
$R(A)$	
$W(A)$	
	$R(A)$
	$W(A)$
	$R(B)$
	$W(B)$
	Commit
$R(B)$	
$W(B)$	
Commit	

T_1	T_2
$R(A)$	
$W(A)$	
	$R(A)$
	$W(A)$
	$R(B)$
	$W(B)$
	Commit
Abort	

View Serializability

- Conflict serializability is sufficient but not necessary for serializability
 - There are serializable schedules which are not conflict serializable

View Equivalence

1. Two schedules S_1 and S_2 are view equivalent if they have the *same transactions* and the following properties are true
 - a) If T_i reads the initial value of object A in S_1 , it must also read the initial value of A in S_2 .
 - b) If T_i reads a value A written by T_j in S_1 , it must also read the value of A written by T_j in S_2
 - c) For each data object A , the transaction (if any) that performs the final write on A in S_1 must also perform the final write on A in S_2

View Serializability

- A schedule is view serializable if it is view equivalent to some serial schedule
- Every conflict serializable schedule is also view serializable.

View Serializability

- Example: The schedule below is view serializable but not conflict serializable

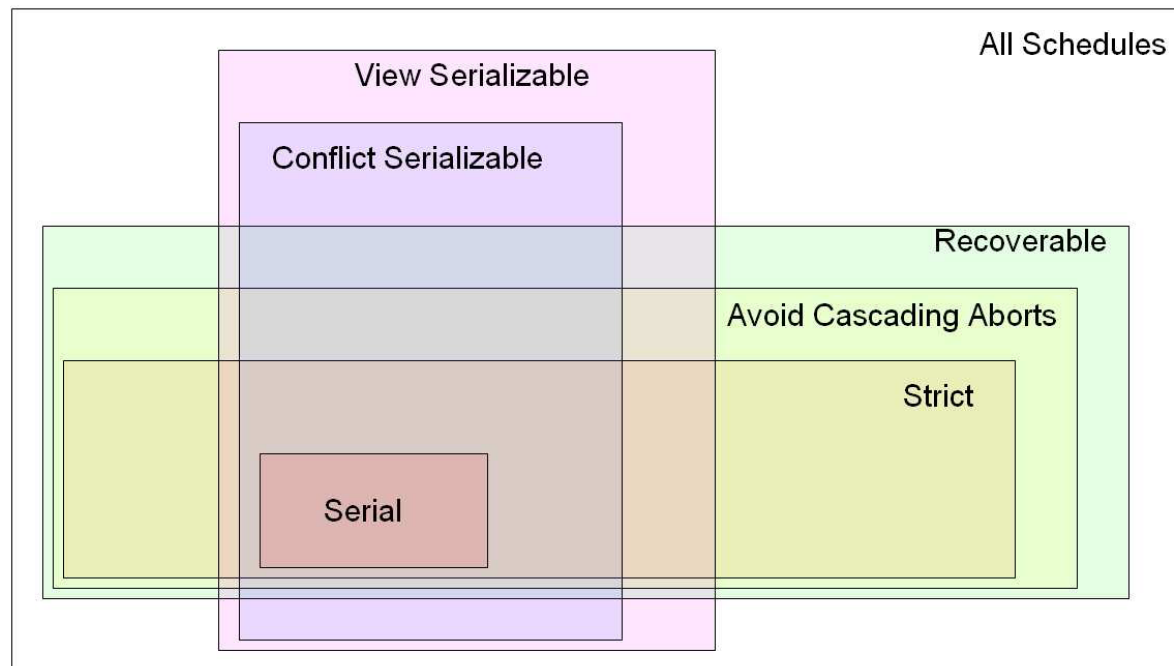
T_1	T_2	T_3
R(A)	W(A) Commit	
W(A) Commit		W(A) Commit

View Serializability

- Any schedule which is view serializable but not conflict serializable has a blind write
- Enforcing or testing for view serializability is more complex than for conflict serializability
 - The problem is NP Complete in general
 - It's nice to work out, but no one actually uses it for that reason

Review

- Picture summarizes how they all relate or overlap using a Venn diagram
 - Overlaps show the presence of overlap, not the percentage of how much they do



Conclusion

- Transaction Support in SQL
- 2PL, Serializability, Recoverability
 - Conflict Equivalence
 - Conflict Serializability
 - View Equivalence
 - View Serializability
- Review of Transaction Properties