Conflict Equivalent

34

- Outcome of a schedule depends on the order of conflicting operations
- Can interchange non-conflicting ops without changing effect of the schedule

- If two schedules S1 and S2 are conflict equivalent then they have the same effect

  S1 ←→ S2 by swapping non-conflicting ops

Conflict Serializability

35

- Every conflict serializable schedule is serializable

- R1(A); W1(A); R2(A); W2(A); R1(B); W1(B); R2(B); W2(B)

Can we transform into a serial schedule by swapping of adjacent non-conflicting actions?

R1(A); W1(A); R1(B); W1(B); R2(A); W2(A); R2(B); W2(B)

Adapted from M. Balazinska

Precedence Graph Test

36

Is a schedule conflict-serializable ?

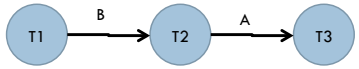
Simple test:

- Build a graph of all transactions Ti
- Edge from Ti to Tj if Ti comes first, and makes an action that conflicts with one of Tj
- The test: if the graph has no cycles, then it is conflict serializable !

Example 1

37

R2(A); R1(B); W2(A); R3(A); W1(B); W3(A); R2(B); W2(B)



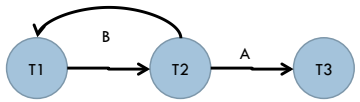
This schedule is conflict serializable

Example 2

38

R2(A); R1(B); W2(A); R3(A); W1(B); W3(A); R2(B); W2(B)

R2(A); R1(B); W2(A); R2(B); R3(A); W1(B); W3(A); W2(B)



This schedule is NOT conflict serializable

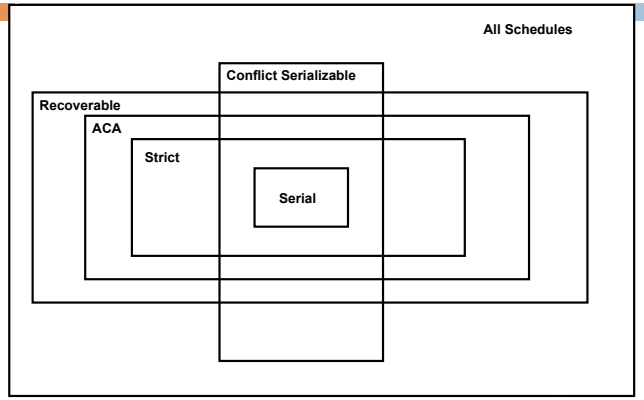
The cycle indicates that the output of T1 depends on T2, and vice-versa.

Strict Schedule

39

- A schedule S is strict if a value written by Ti is not read or overwritten by other Tj until Ti aborts or commits
- Example:
  W1(A); W1(B), C1; W2(A); R2(B); C2;
- Strict schedules are recoverable, and avoid cascading aborts.

Venn Diagram for Schedules



Scheduler

41

- The scheduler is the module that schedules the transaction's actions, ensuring serializability.
- How ?
  - Locks
  - Time stamps

Lock Based Concurrency Control

42

- DBMS aims to allow only recoverable and serializable schedules
- Ensure committed transactions are not un-done while aborting transactions
- Use a *locking protocol*: a set of rules to be followed by each transaction to ensure serializable schedules
- *Lock*: a mechanism to control concurrent access to a data object

Locking Scheduler

43

Simple idea:

- Each element has a unique lock
- Each transaction must first acquire the lock before reading/writing that element
- If the lock is taken by another transaction, then wait
- The transaction must release the lock(s)

Adapted from M. Balazinska

Notation

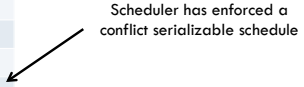
44

- Li(A) = transaction Ti acquires lock for element A
- Ui(A) = transaction Ti releases lock for element A

Example 1

45

T1	T2
L1(A)	
R1(A), W1(A)	
U1(A), L1(B)	
	L2(A)
	R2(A), W2(A)
	U2(A)
	L2(B) , **DENIED…**
R1(B), W1(B)	
U1(B)	
	GRANTED;
	R2(B), W2(B)
	U2(B)



Example 2

46

T1	T2
L1(A)	
R1(A), W1(A)	
U1(A)	
	L2(A)
	R2(A), W2(A)
	U2(A)
	L2(B)
	R2(B), W2(B)
	U2(B)
L1(B)	
R1(B), W1(B)	
U1(B)	

Scheduler has NOT enforced conflict serializability

For A: T1-T2
For B: T2-T1

Types of Locks

47

- Shared lock (for reading)
- Exclusive lock (for writing, and of course, also for reading)
- Notation
  - $S_T(A)$: transaction T requests shared lock on object A
  - $X_T(A)$: transaction T requests exclusive lock on object A

Lock Modes

48

- S = Shared lock (for read)
- X = Exclusive lock (for write)

Lock compatibility matrix

	None	S	X
None	OK	OK	OK
S	OK	OK	Conflict
X	OK	Conflict	Conflict

Strict Two Phase Locking (Strict 2PL)

49

- Most widely used locking protocol
- Two rules:
  1. Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  2. All locks held by a transaction are released when the transaction completes
- If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.
- Strict 2PL allows only schedules whose precedence graph is acyclic (i.e., serializable)
- Recoverable and ACA

Strict 2PL Example

50

T1	T2
L(A);	
R(A), W(A)	
	L(A); **DENIED…**
L(B);	
R(B), W(B)	
U(A), U(B)	
Commit;	
	…GRANTED
	R(A), W(A)
	L(B);
	R(B), W(B)
	U(A), U(B)
	Commit;

All locks held by T1 are released when T1 completes.

Implications

51

- The locking protocol only allows safe interleavings of transactions
- If T1 and T2 access different data objects, then no conflict and each may proceed
- Otherwise, if same object, actions are ordered serially.
  - The Xact who gets the lock first must complete before the other can proceed

Two Phase Locking Protocol (2PL)

52

- Variant of Strict 2PL
- Relaxes the 2nd rule of Strict 2PL to allow Xacts to release locks before the end (commit/abort)
- Two rules:
  - Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  - A transaction cannot request additional locks once it releases any lock.
  - If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.

2PL Example

53

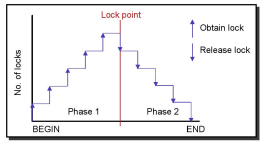
T1	T2
X(A), X(B)	
R(A), W(A)	
U(A)	
	X(A)
	R(A), W(A)
	X(B), **DENIED…**
R(B), W(B)	
U(B)	
	..GRANTED
	R(B), W(B)
	U(A), U(B)

All locks are first acquired, and then released.

2PL Implications

54

- In every transaction, all lock requests must precede all unlock requests.
- This ensures conflict serializability
  - Why? (Think of order Xacts enter their shrinking phase)
  - This induces a sort ordering of the transactions that can be serialized
- Each transaction is executed in two phases:
  - Growing phase: the transaction obtains locks
  - Shrinking phase: the transaction releases locks



Credit: A. Mazeika

Strict 2PL makes 2PL "strict"

55

- Recall: a strict schedule is one where a value written by T is not read/overwritten until T commits/aborts
  - Strict 2PL makes T hold locks until commit/abort
  - No other transaction can see or modify the data object until T is complete

Remarks

56

- What if a transaction releases its locks and then aborts?
- Recall: conflict serializable definition only considers *committed* transactions

Phantom Problem

57

- So far we have assumed the database to be a *static* collection of elements (=tuples)
- If tuples are inserted/deleted then the *phantom problem* appears

Phantom Problem

58

T1	T2
SELECT * FROM Product WHERE color='blue'	
	INSERT INTO Product(name, color) VALUES ('gizmo','blue')
SELECT * FROM Product WHERE color='blue'	

Is this schedule serializable ?

Phantom Problem

59

T1	T2
SELECT * FROM Product WHERE color='blue'	
	INSERT INTO Product(name, color) VALUES ('gizmo','blue')
SELECT * FROM Product WHERE color='blue'	

Suppose there are two blue products, X1, X2:

R1(X1),R1(X2),W2(X3),R1(X1),R1(X2),R1(X3)

This is conflict serializable ! What's wrong ??

Phantom Problem

60

T1	T2
SELECT * FROM Product WHERE color='blue'	
	INSERT INTO Product(name, color) VALUES ('gizmo','blue')
SELECT * FROM Product WHERE color='blue'	

Suppose there are two blue products, X1, X2:

R1(X1),R1(X2),W2(X3),R1(X1),R1(X2),R1(X3)

Not serializable due to ***phantoms***

Phantom Problem

61

- A "phantom" is a tuple that is invisible during part of a transaction execution but not all of it.
- In our example:
  - T1: reads list of products
  - T2: inserts a new product
  - T1: re-reads: a new product appears !

Phantom Problem

62

- In a ***static*** database:
  - Conflict serializability implies serializability
- In a ***dynamic*** database, this may fail due to phantoms

Dealing With Phantoms

63

- Lock the entire table, or
- Lock the index entry for 'blue'
  - If index is available

Dealing with phantoms is expensive !

Transaction Support in SQL

64

- A transaction is initiated implicitly when SQL statement is executed.
  - Each DBMS provides either a commit or a rollback (abort) option
- Isolation level: controls the extent to which a transaction is exposed to actions of other transactions executing concurrently
- Four possible isolation levels
  - Increase concurrency → increasing Xact exposure to uncommitted changes from other Xacts

Isolation Levels (cont'd)

65

- SQL-92 provides different isolation levels that control the degree of concurrency

Isolation Level	In DB2	Dirty Read	Unrepeatable Read
Read Uncommitted	Uncommitted Read	Maybe	Maybe
Read Committed	Cursor Stability (default)	No	Maybe
Repeatable Reads	Repeatable Reads	No	No
Serializable	Read Stability	No	No

Phantoms possible