Concurrency

Unit 4.2

Locking

A solution to enforcing serialisability?

- read (shareable) lock
- write (exclusive) lock
- coarse granularity
  - easier processing
  - less concurrency
- fine granularity
  - more processing
  - higher concurrency

Locking cont…

Many systems use locking mechanisms for concurrency control. When a transaction needs an assurance that some object will not change in some unpredictable manner, it acquires a lock on that object.

- A transaction holding a read lock is permitted to read an object but not to change it.
- More than one transaction can hold a read lock for the same object.
- Usually, only one transaction may hold a write lock on an object.
- On a transaction schedule, we use 'S' to indicate a shared lock, and 'X' for an exclusive write lock.

Locking – Uncommitted Dependency

Locking solves the uncommitted dependency problem

Time	Transaction A	Transaction B	Lock on R Before => after
t1		write(R)	- => X
t2	read(r) **WAIT**		
t3	…**WAIT…**	**ABORT**	X => -
t4	read(r) **GO**		- -> S

Deadlock

Deadlock can arise when locks are used, and causes all related transactions to WAIT forever

Time	Transaction A	Transaction B	Lock State	
			X	**Y**
t1	write(X)		- => X	-
t2		write(Y)	X	- => X
t3	read(Y) **WAIT**		X	X
t4	…**WAIT…**	read(X) **WAIT**	X	X
t5	…**WAIT…**	…**WAIT…**	X	X

Deadlock cont…

The `lost update’ senario results in deadlock with locks. So does the `inconsistency’ scenario.

Time	**Transaction A**	**Transaction B**	**Lock on R** **Before => after**
t1	read(r)		- => S
t2		read(r)	S => S
t3	write(R)		S
t4	…**WAIT**…	write(R)	S
t5	…**WAIT**…	…**WAIT**…	S

Deadlock Handling

- Deadlock avoidance
  - pre-claim strategy used in operating systems
  - not effective in database environments.
- Deadlock detection
  - whenever a lock requests a wait, or on some perodic basis.
  - if a transaction is blocked due to another transaction, make sure that that transaction is not blocked on the first transaction, either directly or indirectly via another transaction.

Deadlock Resolution

If a set of transactions is considered to be deadlocked:

1. choose a victim (e.g. the shortest-lived transaction)
2. rollback 'victim' transaction and restart it.
   - The rollback terminates the transaction, undoing all its updates and releasing all of its locks.
   - A message is passed to the victim and depending on the system the transaction may or may not be started again automatically.

Two-Phase Locking

The presence of locks does not guarantee serialisability. If a transaction is allowed to release locks before the transaction has completed, and is also allowed to acquire more (or even the same) locks later then the benifit or locking is lost.

If all transactions obey the ‘two-phase locking protocol’, then all possible interleaved executions are guarenteed serialisable.

Two-Phase locking cont...

The two-phase locking protocol:

- Before operating on any item, a transaction must acquire at least a shared lock on that item. Thus no item can be accessed without first obtaining the correct lock.
- After releasing a lock, a transaction must never go on to acquire any more locks.

The technical names for the two phases of the locking protocol are the ‘lock-acquisition phase’ and the ‘lock-release phase’.

Other Database Consistency Methods

Two-phase locking is not the only approach to enforcing database consistency. Another method used in some DMBS is timestamping. With timestamping, there are no locks to prevent transactions seeing uncommitted changes, and all physical updates are deferred to commit time.

- locking synchronises the interleaved execution of a set of transactions in such a way that it is equivalent to some serial execution of those transactions.
- timestamping synchronises that interleaved execution in such a way that it isequivalent to a particular serial order - the order of the timestamps.

Timestamping rules

The following rules are checked when transaction T attempts to change a data item. If the rule indicates ABORT, then transaction T is rolled back and aborted (and perhaps restarted).

- If T attempts to read a data item which has already been written to by a younger transaction then ABORT T.
- If T attempts to write a data item which has been seen or written to by a younger transaction then ABORT T.

If transaction T aborts, then all other transactions which have seen a data item written to by T must also abort. In addition, other aborting transactions can cause further aborts on other transactions. This is a ‘cascading rollback’.