Transaction Concepts – ACID Properties – Schedules – Serializability – Concurrency Control – Need for Concurrency – Locking Protocols – Two Phase Locking – Deadlock – Transaction Recovery - Save Points – Isolation Levels – SQL Facilities for Concurrency and Recovery

TWO PHASE LOCKING PROTOCOL

- This protocol requires that each transaction issue lock and unlock request in two phases
  - Growing phase
  - Shrinking phase
- **Growing pha**
  - During this phase new locks can be occurred but none can be released
- **Shrinking phase**
  - During which existing locks can be released and no new locks can be occurred
- **Types**
  - Strict two phase locking protocol
  - Rigorous two phase locking protocol
- **Strict two phase locking protocol**
- This protocol requires not only that locking be two phase, but also all exclusive locks taken by a transaction be held until that transaction commits.

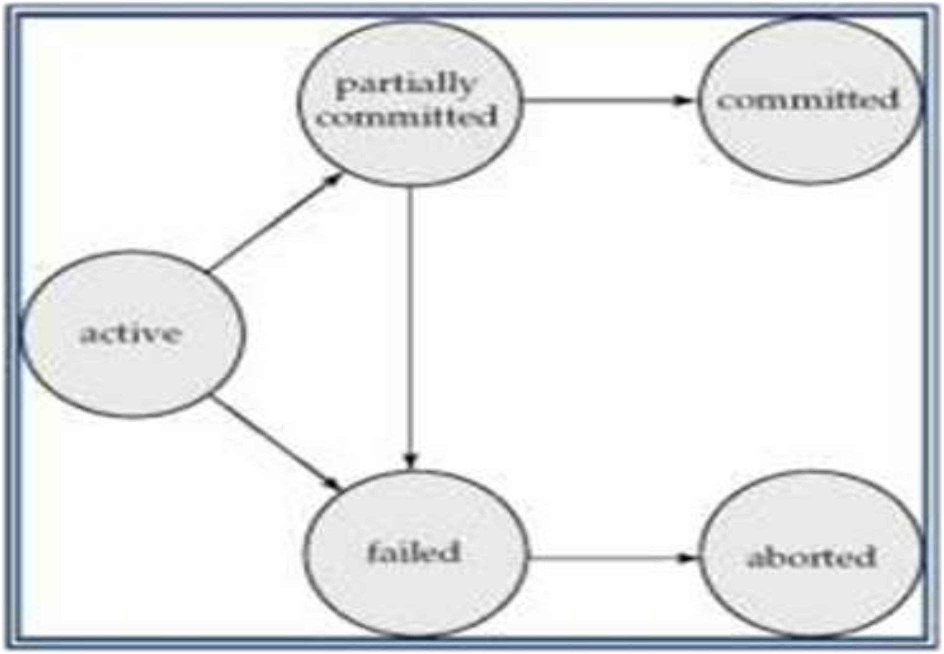
Conn…

- **Rigorous two phase locking protocol**
  - This protocol requires that all locks be held until all transaction commits.
- Consider the two transaction T_1 and T_2 T_1 : read(a_1);
  - read(a_2);
  - ……. read(a_n);
- write(a_1); T_2: read(a_1);
  - read(a_2); display(a_1+a_1);
- **Lock conversion**
  - Lock Upgrade
  - Lock Downgrade
- **Lock upgrade:**
  - Conversion of existing read lock to write lock
  - Take place in only the growing phase
- if Ti has a read-lock (X) and Tj has no read-lock (X) ($i \neq j$) then convert read-lock (X) to write-lock (X)
- else
  - force Ti to wait until Tj unlocks X
- **Lock downgrade:**
  - conversion of existing write lock to read lock
  - Take place in only the shrinking phase
- Ti has a write-lock (X) (*no transaction can have any lock on X*) convert write-lock (X) to read-lock (X)

Conn..

T_1	T_2
Lock-S(a_1)	
	Lock-S(a_1)
Lock-S(a_2)	
	Lock-S(a_1)
Lock-S(a_3)	
Lock-S(a_4)	
	Unlock(a_1)
	Unlock(a_2)
Lock-S(a_1)	
Upgrade(a_1)	

Transaction State



Conn..

- Active – the initial state; the transaction stays in this state while it is executing
- Partially committed – after the final statement has been executed.
- Failed -- after the discovery that normal execution can no longer proceed.
- Aborted – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - restart the transaction
  - kill the transaction
- Committed – after successful completion

- **Log**
  - Log is a history of actions executed by a database management system to guarantee ACID properties over crashes or hardware failures.
  - Physically, a log is a file of updates done to the database, stored in stable storage.
- **Log rule**
  - A log records for a given database update must be physically written to the log, before the update physically written to the database.
  - All other log record for a given transaction must be physically written to the log, before the commit log record for the transaction is physically written to the log.
  - Commit processing for a given transaction must not complete until the commit log record for the transaction is physically written to the log.

Conn..

- **System log**
  - **[Begin transaction ,T]**
  - **[write_item , T, X , oldvalue,newvalue]**
  - **[read_item,T,X]**
  - **[commit,T]**
  - **[abort,T]**

- Assumes fail-stop model – failed sites simply stop working, and do not cause any other harm, such as sending incorrect messages to other sites.
- Execution of the protocol is initiated by the coordinator after the last step of the transaction has been reached.
- The protocol involves all the local sites at which the transaction executed
- Let T be a transaction initiated at site S_i, and let the transaction coordinator at S_i be C_i

- **Phase 1: Obtaining a Decision (prepare)**
- Coordinator asks all participants to *prepare* to commit transaction T_i.

Conn..

- C_i adds the records <prepare *T*> to the log and forces log to stable storage
- sends prepare *T* messages to all sites at which *T* executed

- Upon receiving message, transaction manager at site determines if it can commit the transaction
  - if not, add a record <no *T*> to the log and send abort *T* message to C_i
  - if the transaction can be committed, then:
  - add the record <ready *T*> to the log
  - force *all records* for *T* to stable storage
  - send ready *T* message to C_i
- **Phase 2: Recording the Decision (commit)**
  - *T* can be committed of C_i received a ready *T* message from all the participating sites: otherwise
  - *T* must be aborted.
  - Coordinator adds a decision record, <commit *T*> or <abort *T*>, to the log and forces record onto stable storage. Once the record stable storage it is irrevocable (even if failures occur)
  - Coordinator sends a message to each participant informing it of the decision (commit or abort)
  - Participants take appropriate action locally.

Conn..

- **Handling of Failures - Site Failure**
- When site S_i recovers, it examines its log to determine the fate of transactions active at the time of the failure.
  - Log contain <commit *T*> record: site executes redo (*T*)
  - Log contains <abort *T*> record: site executes undo (*T*)
  - Log contains <ready *T*> record: site must consult C_i to determine the fate of *T*.
    - If *T* committed, redo (*T*)
    - If *T* aborted, undo (*T*)
  - The log contains no control records concerning *T* replies that S_k failed before responding to the prepare *T* message from C_i
    - since the failure of S_k precludes the sending of such a
      - response C_1 must abort *T*
    - S_k must execute undo (*T*)

Conn..

- **Handling of Failures- Coordinator Failure**
  - If coordinator fails while the commit protocol for *T* is executing then participating sites must decide on *T*'s fate:
    1. If an active site contains a <commit *T*> record in its log, then *T* must be committed.
    2. If an active site contains an <abort *T*> record in its log, then *T* must be aborted.
    3. If some active participating site does not contain a <ready *T*> record in its log, then
    - the failed coordinator C_i cannot have decided to commit *T*. Can therefore abort *T*.
    4. If none of the above cases holds, then all active sites must have a <ready *T*> record in their logs, but no additional control records (such as <abort *T*> of <commit *T*>). In this case
    - active sites must wait for C_i to recover, to find decision.
  - Blocking problem : active sites may have to wait for failed coordinator to recover.
- **Handling of Failures - Network Partition**
  - If the coordinator and all its participants remain in one partition, the failure has no effect on the commit protocol.
  - If the coordinator and its participants belong to several partitions:
    - – Sites that are not in the partition containing the coordinator think the coordinator has failed, and execute the protocol to deal with failure of the coordinator.

Conn…

    - No harm results, but sites may still have to wait for decision from coordinator.
- The coordinator and the sites are in the same partition as the coordinator think that the sites in the other partition have failed, and follow the usual commit protocol.
    - Again, no harm results

Conn…

Thank You…….