DB

Unit 6.

Q1. Purpose of checkpoint mechanism. Explain steps for performing a checkpoint.

⇒.

- A checkpoint is point in time in log when a known & consistent state for database system is established.
- Typically, a checkpoint involves recording a certain amount of info. so that, if failure occurs, the database server can restart at that established point.

- Purpose of checkpoint mechanism:

① Efficient recovery preparation:
- If system crash occurs, the recovery involves redoing or undoing transactions based on log records.
- Without checkpoint, system searches entire log. Hence time consuming.
- Checkpoint simplifies this process by marking a point in log where certain actions have been completed, which reduces the search space for recovery.

② Reducing recovery overhead.
- Transactions that need to be redone, may have already written their updates into database by time of crash.
- Checkpoints reduces recovery overhead by ensuring that updates are written to disk before a checkpoint, minimizing the need for redo operations after crash.

- Steps for Performing a checkpoint:

① Output log records to stable storage.
② Output modified buffer blocks to disk
③ Record Active transactions
- A log record from <checkpoint, L> is output into stable storage.
L: List of transactions active at that time

- During checkpoints, transactions are freezed.
- After system crash <checkpoint, L> comes in action.

// Undo & Redo conditions.

- Checkpoint facilitates efficient recovery, minimizes overhead, synchronizes disk storage.

Q2. Various classes of failure in DB system.

⇒

① Transaction Failure:
- Occurs during transaction
- If ACID is not followed, failure occurs
i) Logical Error:
- Encounters internal condition
- Prevents from continuing normal execution.
- eg: bad input, data not found, overflow.
ii) System Error:
- System enters undesirable state like deadlock that prevents from continuing normal execution.
- Transactions can be reexecuted later.

② System Crash

- Occurs due to h/w malfunction / bugs in s/w.
- Loss in volatile storage ∴ transaction halts.
- Non-volatile is ok & not corrupted.
- ↑ is called as fail-stop assumption.
- Recovery involves, system restart, perform recovery actions to restore consistency & transaction atomicity.

③ Disk failure

- Disk block loses contents ∴ head crash or failure during data transfer.
- Data is copied to other disk / archival backup for recovery.
- Algorithms used to recover lost data of block disk from copies / backups.

Q3. Immediate Database Modification with its recovery mechanism.

⇒

- Technique occurs on active transactions
- DB is modified immediately after every operation.
- Follows actual DB modification
- Technique is used for maintaining transaction log files in DBMS
- Also known as UNDO/REDO technique.
- Serves for transaction recovery from power outrages, memory issues, OS failures.
- Upon any transaction, updates are directly applied to DB & log file for old & new val are maintained
- Commited transactions are permanently stored in DB, & previous records are discarded.
- For roll back, old values are restored in DB, all changes made in DB are discarded. This is known as "Undoing" process

transaction

- If system crashes after commit, upon system restart, changes are permanently stored in DB.

Recovery using log records:

- Recovery reads log backwards from end to last checkpoint.
- Maintains undo-list & redo-list.
- If log contains <Tn, Start> & <Tn, commit> or only <Tn, commit> then Tn needs to be redone. Hence it puts in redo-list

- If log contains <Tn, Start> but no commit or abort log found, it needs to be undone. Recovery puts Tn in undo-list.

CO 4.	Deferred DB Modification.	Immediate Database Modification
Timing of Updates	Applied only after transaction commits	Applied as soon as they are issued.
Write-Ahead Logging (WAL)	Changes are logged before applying them.	Changes are logged before each write operation.
Transaction commit	Changes are written to DB upon Tran. commit.	Changes are written to DB immediately but logged first.
Rollback Handling	Simple rollback.	Requires Undo.
Crash Recovery	Simple (no undo needed)	Complex (Both Undo & Redo needed)
Concurrency control	Easier to manage	Complex.
Buffer Management	less frequent buffer flushing.	More frequent buffer flushing.
Performance	May have better performance	Can be less performant.
Data Integrity	Ensures by applying changes only if Tran. commits.	Maintains with help of logging & concurrency control mechanism.
Logging Overhead	Lower	Higher.
Implement.	Simpler	Complex.
Temp. Storage Requirements.	May require more temp. storage.	Less temp storage required.
Usage Scenario.	Environments where tran are short & commit frequently.	Systems where immediate data visibility is crucial.
Example of Use	Used in Batch processing systems.	Common in online transactions
Consistency Model	Strong consistency	Immediate consistency.

Unit 5

Q1. Variants of Two Phase Lock Protocol (2PL).

=>

- 2PL is concurrency control mechanism
- Ensures serializability due to lock & unlock requests in 2 different phases.

① Growing Phase :
- Aquire locks but cannot release locks.
- It is initial stage as locks are needed.

② Shrinking Phase :
- Can release locks but cannot obtain more locks.
- Occurs after Transaction releases a lock, preventing further lock acquisition.

◦ Variants of 2PL :

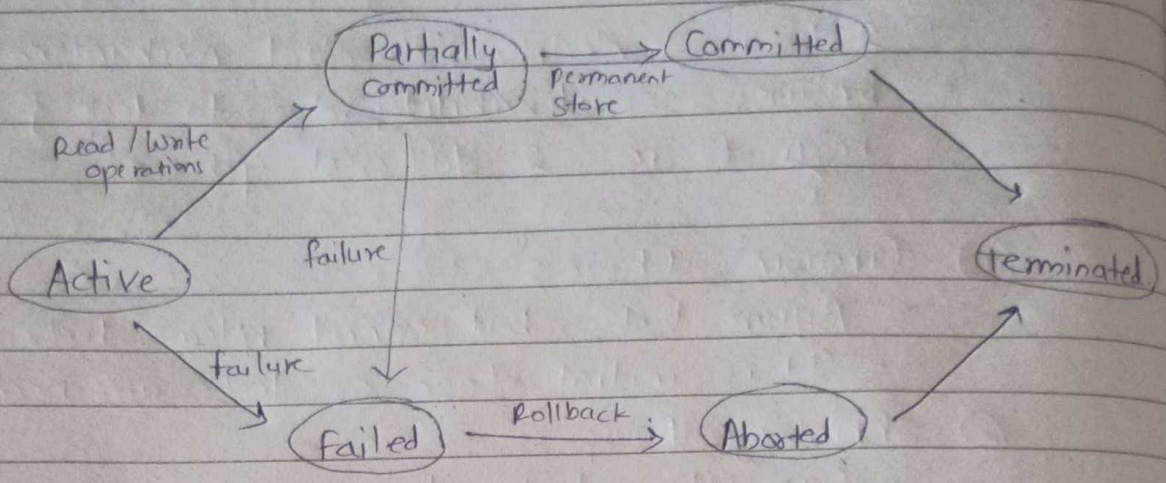
1) Basic 2PL Protocol :
- Guarantees serializability by enforcing lock acquisition & release phase.
- Locks are released only after transaction commits / aboots.

2) Strict 2PL Protocol :
- Requires exclusive locks to held until Tran. commits.
- Data written by uncommitted transaction can't be read by other transaction.

3) Rigorous 2PL Protocol :
- Requires all locks to be held until transaction commit.
- Serialized based on commit order.

4) 2PL with Lock Conversion :
- Allows lock conversions bet^n shared & exclusive during Tran.
- Enable more concurrency (Initially shared, later exclusive)
- Shared → Exclusive occurs in Growing phase.
- Exclusive → Shared occurs in Shring phase.

Q3] Transaction State Diagram



① Active:
- Initial state
- Transaction is in this state while executing
- Transaction's execution = Active
-

② Partially committed:
- Goes from active to P. committed when there are read / write operations
- Operations performed in main memory instead of actual DB.
- This state is used cause Tran. can fail during execution hence changes in main Memory.
- This state helps in rollback made in DB in case of failure.

③ Committed:
- If Tran. executes successfully then, the local memory's changes from partially committed state are permanently stored in DB.
- Goes from P. committed to committed if everything (read / write operation too) is successful.

④ Failed :
- If instruction from transactions fail then it comes in failed state.
- Or read / write operation fail. i.e from Active → failed or P.committed → failed.

⑤ Aborted :
- Local Memory's changes are rolled back.
- Previous consistent state of memory is brought by roll back.
- A failed state goes to aborted state.

⑥ Terminated.
- If System is consistent then it goes to Terminated state. to terminate the transaction.
- i.e whether failure occurs / tran. successfully executes, it terminates at end.
- Only terminates if there isn't any rollback or commits if system is consistent.