UNIT -IV

ACID PROPERTIES

Transaction: A transaction is a set of actions that forms a single logical unit of task or work.
To maintain the consistency and integrity of the data, there are four properties defined in DBMS known as the ACID properties.

1. Atomicity
2. Consistency
3. Isolation
4. Durability

Atomicity: It means if any operation is performed on the data, either it should be performed completely or should not at all occur. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

Ex: If some money is transferred between A and B, then money debited from A should be credited in B else money should not at all be debited from A. It means atomicity is preserved.

Consistency: It means that the value should remain preserved always i.e. integrity of the data should be maintained. The total sum before and after the transaction should be same.

Ex: If Account A wants to transfer 50$ to account B with initial balances of A and B as 500$ and 300$, then the total sum before the transaction is 800$. Then after the successful transaction T, if the available amount in A becomes 450$ and B becomes $350, then also the total sum is 800$. It means the consistency of database is preserved.

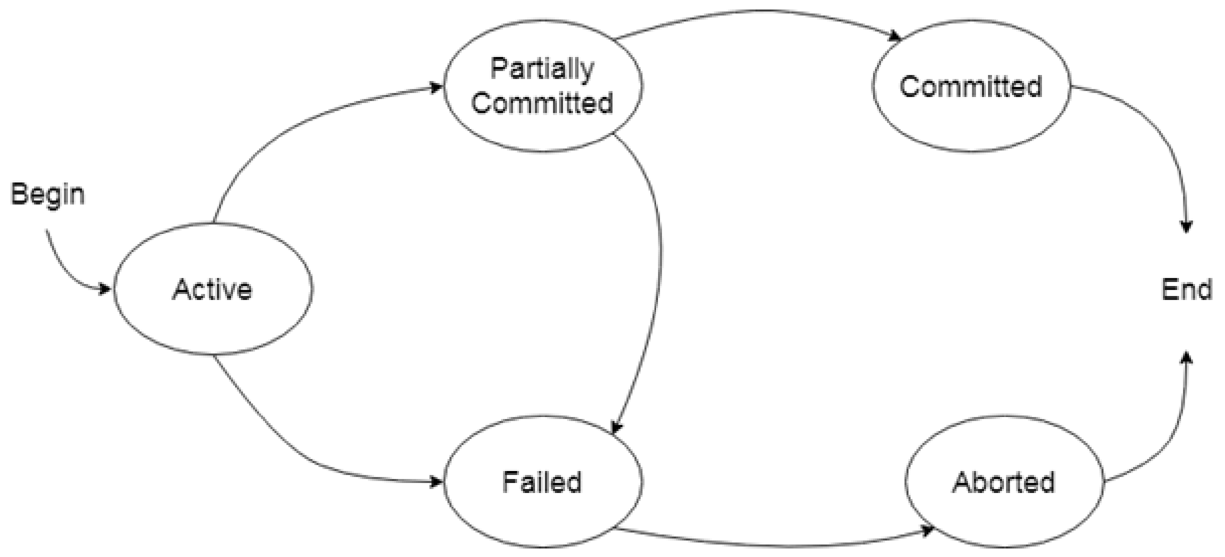
Isolation: It is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete.

Ex: If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected.

Durability: Durability ensures the permanency i.e. even if the system fails or leads to a crash, the database still survives. It becomes the responsibility of the recovery manager for ensuring the durability of the database.

Transaction states:

In a database, the transaction can be in one of the following states -



Active state

- The active state is the first state of every transaction. In this state, the transaction is being executed.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.

Committed

A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

- A transaction is said to be in the failed state if it cannot proceed with normal execution.

Aborted: If a transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.

Concurrent Execution of Transaction

In the transaction process, a system usually allows executing more than one transaction simultaneously. This process is called a concurrent execution.

Advantages of concurrent execution of a transaction

Decrease waiting time or turnaround time.

Improve response time

Increased throughput or resource utilization.

Concurrency Problems in DBMS (Different types of conflicts)-

- When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems.
- Such problems are called as concurrency problems.

The concurrency problems are-

1. Dirty Read Problem/ Reading uncommitted data (Write-Read Conflict)
2. Unrepeatable Read Problem/ Non repeatable read (Read-Write Conflict)
3. Lost Update Problem/ Overwriting Uncommitted data (Write-Write Conflict)
4. Phantom Read Problem

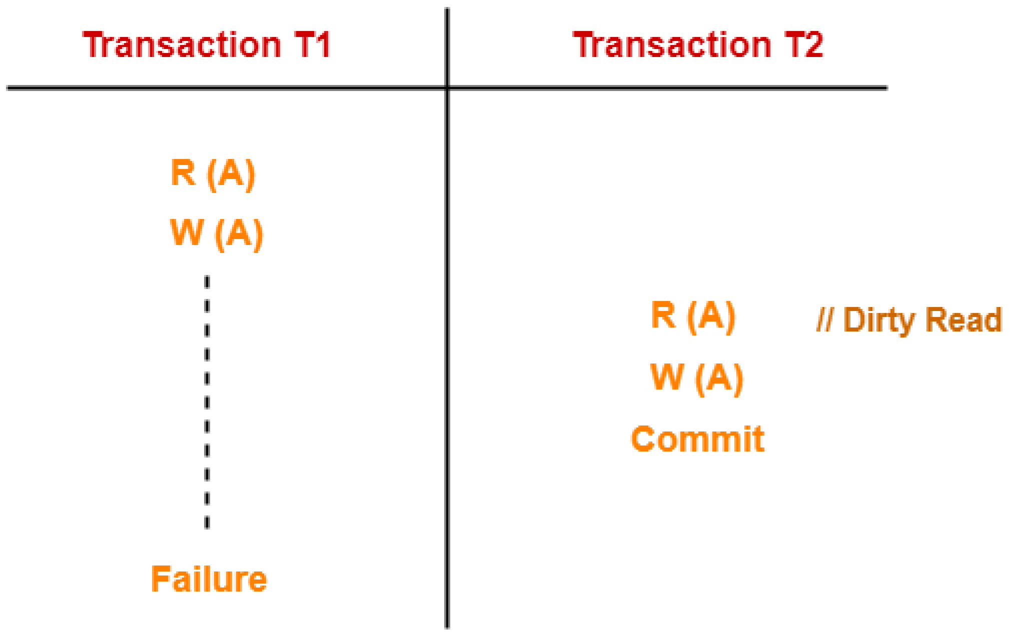
1. Dirty Read Problem-

Reading the data written by an uncommitted transaction is called as dirty read.

This read is called as dirty read because-

- There is always a chance that the uncommitted transaction might roll back later.
- This leads to inconsistency of the database.

Example-



In this example,

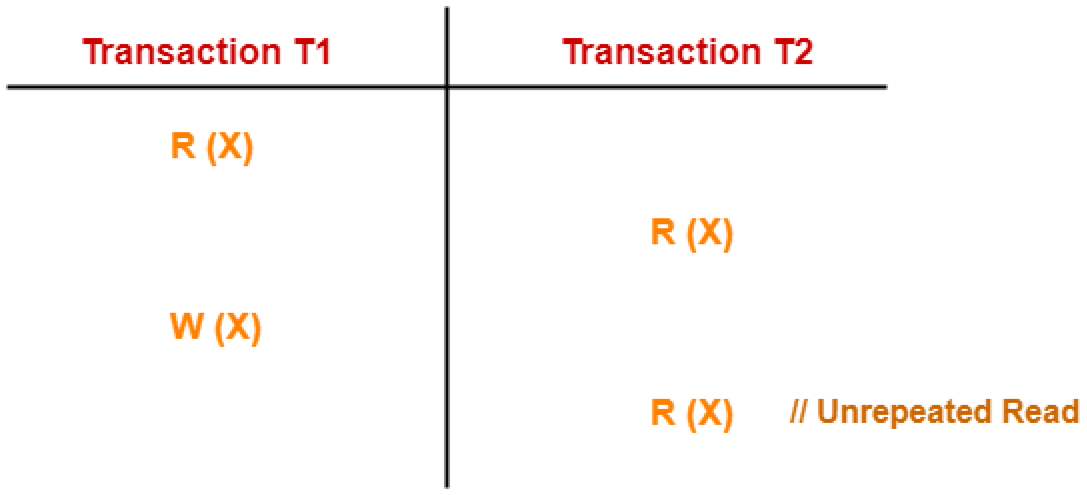
- T2 reads the dirty value of A written by the uncommitted transaction T1.
- T1 fails in later stages and roll backs.

- Thus, the value that T2 read now stands to be incorrect.
- Therefore, database becomes inconsistent.

2. Unrepeatable Read Problem-

This problem occurs when a transaction gets to read unrepeated i.e. different values of the same variable in its different read operations even when it has not updated its value.

Example-



Here,

1. T1 reads the value of X (= 10 say).
2. T2 reads the value of X (= 10).
3. T1 updates the value of X (from 10 to 15 say) in the buffer.
4. T2 again reads the value of X (but = 15).

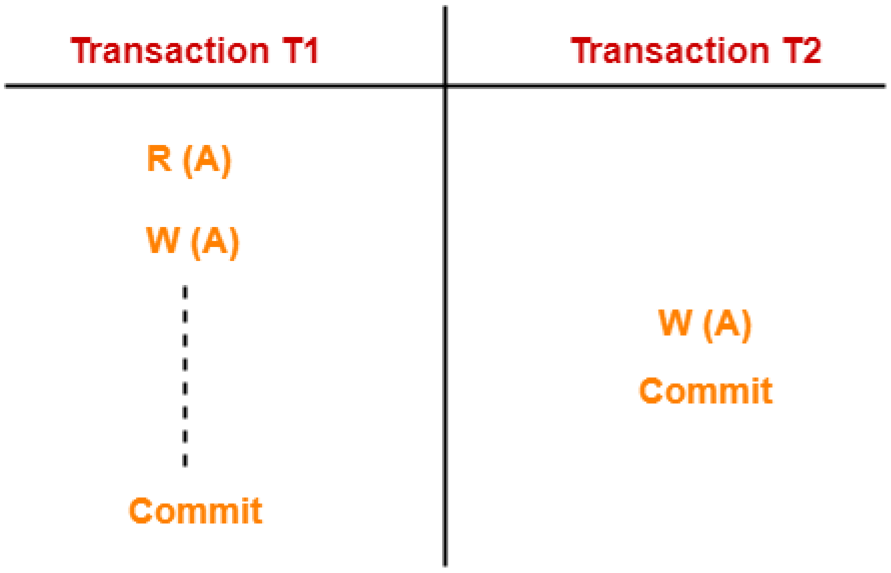
In this example,

- T2 gets to read a different value of X in its second reading.
- T2 wonders how the value of X got changed because according to it, it is running in isolation.

3. Lost Update Problem-

This problem occurs when multiple transactions execute concurrently and updates from one or more transactions get lost.

Example-



Here,

1. T1 reads the value of A (= 10 say).
2. T1 updates the value to A (= 15 say) in the buffer.
3. T2 does blind write A = 25 (write without read) in the buffer.
4. T2 commits.
5. When T1 commits, it writes A = 25 in the database.

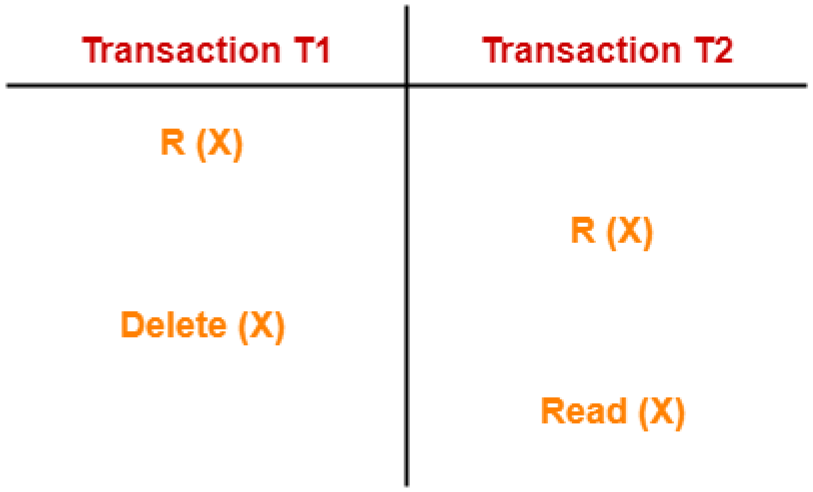
In this example,

- T1 writes the over written value of A in the database.
- Thus, update from T1 gets lost.

4. Phantom Read Problem-

This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later, it finds that the variable does not exist.

Example-



Transaction T1	Transaction T2
R (X)	
	R (X)
Delete (X)	
	Read (X)

Here,

1. T1 reads X.
2. T2 reads X.
3. T1 deletes X.
4. T2 tries reading X but does not find it.

In this example,

- T2 finds that there does not exist any variable X when it tries reading X again.
- T2 wonders who deleted the variable X because according to it, it is running in isolation.

Serializability:

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

There are 3 ways of testing Serializability:

1. Result Serializability
2. Conflict Serializability
3. View Serializability

1.Result Serializability: A non-serial schedule will be result serializable, if its result is equal to the result of its transactions executed serially.

2. Conflict Serializability: The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

Conflicting Operations:

The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

Note: A precedence graph is used to check conflict serializability. If there exists no cycle in the precedence graph then the given schedule S is conflict serializable else it is not conflict serializable.

Example: Check whether the given schedule S is conflict serializable or not-

S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$

Solution-

T1	T2	T3
R(A)		
	R(A)	
R(B)		
	R(B)	
		R(B)
W(A)		
	W(B)	

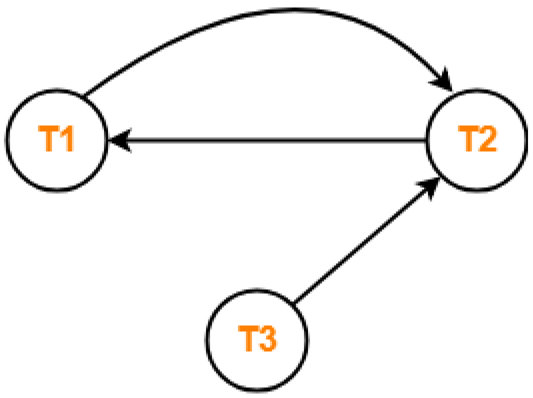
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

-
- $R_1(B)$, $W_2(B)$ ($T_1 \rightarrow T_2$)
- $R_2(A)$, $W_1(A)$ ($T_2 \rightarrow T_1$)
- $R_3(B)$, $W_2(B)$ ($T_3 \rightarrow T_2$)

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.

  Therefore, the given schedule S is not conflict serializable.

3.View Serializability-

If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.

View Equivalent Schedules-

Consider two schedules S1 and S2 each consisting of two transactions T1 and T2.

Schedules S1 and S2 are called view equivalent if the following three conditions hold true for them-

Condition-01:

For each data item X, if transaction T_i reads X from the database initially in schedule S1, then in schedule S2 also, T_i must perform the initial read of X from the database.

Thumb Rule

"Initial readers must be same for all the data items".

Condition-02:

If transaction T_i reads a data item that has been updated by the transaction T_j in schedule S1, then in schedule S2 also, transaction T_i must read the same data item that has been updated by the transaction T_j.

Thumb Rule

"Write-read sequence must be same.".

Condition-03:

For each data item X, if X has been updated at last by transaction T_i in schedule S1, then in schedule S2 also, X must be updated at last by transaction T_i.

Thumb Rule

“Final writers must be same for all the data items”.

RECOVERABILITY:

Recoverable Schedule:

If a Transaction T2 reads a value written by another Transaction T1 and T2 commits after T1 commits, then it is called recoverable schedule because if T1 fails and rolls back , then T2 also can rollback.

Example:

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
COMMIT	
	COMMIT

Irrecoverable Schedule:

If a Transaction T2 reads a value written by another Transaction T1 and T2 commits before T1 commits, then it is called irrecoverable schedule because if T1 fails and rolls back , then T2 cannot rollback.

Example:

T1	T2

R(A)	
W(A)	
	R(A)
	W(A)
	COMMIT
COMMIT	

Lock Based Protocols in DBMS

The protocol utilizes locks, applied by a transaction to data item, which may block other transactions from accessing the same data during the transaction's life.

Types of locking protocols

1. Simple locking
2. 2-Phase Locking
    a. Simple 2-phase locking
    b. Strict 2-phase locking
    c. Rigorous 2-phase locking
    d. Conservative 2-Phase locking

1) Simple locking:

There are two types of lock :

1. Shared Lock
2. Exclusive Lock

Shared Locks are represented by S. The data items can only read without performing modification to it from the database. S – lock is requested using
lock – s instruction.

Exclusive Locks are represented by X. The data items can be read as well as written. X – lock is requested using lock – X instruction.

Lock Compatibility Matrix

- Lock Compatibility Matrix controls whether multiple transactions can acquire locks on the same resource at the same time.

	Shared	Exclusive
Shared	True	False
Exclusive	False	False

<u>LIMITATIONS:</u>

1. If the locks are released too early, it leads to Database Inconsistency.
2. It may not guarantee Serializability.

3. Dead locks are possible.

2) 2-Phase locking protocols:

It is used to overcome the disadvantage of the simple lock. In the 2-phase locking, there are two phases are growing phase and shrinking phase. 2-phase ensures serializability.

Growing phase: A transaction may obtain locks but not release any locks.

Shrinking phase: A transaction may release lock but may not obtain new locks.

A) Simple 2-phase locking: Once a transaction releases a lock it enters in shrinking phase and in shrinking phase it cannot issue any more locks.

Example:

T1	COMMENTS
LOCK-S(A)	GROWING PHASE
READ(A)	
LOCK-X(B)	
READ(B)	
WRITE(B)	
UNLOCK-X(B)	SHRINKING PHASE
UNLOCK-S(A)	

Advantage:

- Less resource utilization.
- Guarantees serializability based on Lock points.

Disadvantage:

- Deadlock present.
- Cascading roll backs are possible

There are three types of 2-phase locking protocols:

B) Strict 2-phase locking:

Transaction does not release any of the exclusive locks until after its commit or abort. It means all the exclusive locks held by a transaction are released only after the transaction commits.

Example:

T1
LOCK-S(A)
READ(A)
LOCK-X(B)
READ(B)
WRITE(B)
UNLOCK-S(A)
COMMIT
UNLOCK-X(B)

Advantage:

- No dirty read problem.
- No Cascading rollbacks.

Disadvantage:

- Deadlock are possible.

C) Rigorous 2-phase locking: Transaction "t" does not release any of its locks until after it commits or aborts. It means both shared and exclusive locks are released only after the transaction commits.

T1

LOCK-S(A)
READ(A)
LOCK-X(B)
READ(B)
WRITE(B)
COMMIT
UNLOCK-S(A)
UNLOCK-X(B)

Advantage:

- No Cascading rollbacks

Disadvantage:

- Deadlock can still occur.

D) **Conservative 2-phase locking**: The transaction will obtain all the locks at the beginning i.e. before the transaction starts and will release all the locks only after it commits.

Example:

T1
LOCK-S(A)
LOCK-X(B)
READ(A)

READ(B)
WRITE(B)
COMMIT
UNLOCK-S(A)
UNLOCK-X(B)

Advantage:

- No waiting for the data item
- No Cascading rollbacks
- No deadlocks

Disadvantage:

- Practically data implementation is difficult
- It leads to starvation.

GRAPH BASED PROTOCOLS

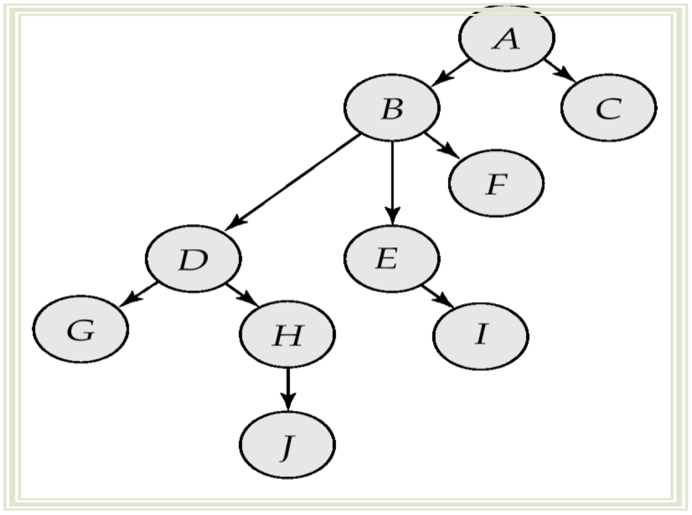
Graph-based protocols are an alternative to two-phase locking.
A Graph based protocol is implemented as a Tree.

Impose a partial ordering on the set D = {d1, d2 ,..., dh} of all data items.

The tree-protocol is a simple kind of graph protocol.

The set of rules used by a Tree Protocol are:

- Only Exclusive locks are allowed.
- The first lock by Ti may be on any data item. Subsequently, a data Q can be locked by Ti only if the parent of Q is currently locked by Ti.
- Data items may be unlocked at any time.
- Data item that is locked and unlocked by Ti cannot be subsequently relocked by the same transaction Ti.



Advantage –

1. Ensures Conflict Serializable Schedule.

2. Ensures Deadlock Free Schedule
3. Shorter waiting time as unlocking can be done anytime

Disadvantage –

1. Unnecessary locking overheads may happen sometimes, like if we want both D and E, then at least we have to lock B to follow the protocol.
2. Cascading Rollbacks is still a problem. We don't follow a rule of when Unlock operation may occur so this problem persists for this protocol.
3. Prior knowledge of data access.

TIME STAMP BASED PROTOCOLS:

Each transaction is issued a timestamp when it enters the system. If an old transaction Ti has time-stamp TS(Ti), a new transaction Tj is assigned time-stamp TS(Tj) such that

TS(Ti) <TS(Tj).

• The protocol manages concurrent execution such that the time-stamps determine the serializability order.

Timestamps can be used in two ways:

1. To determine the outdatedness of a request with respect to the data object it is operating on.

2. To order events with respect to one another.

• In order to assure such behavior, the protocol maintains for each data Q two timestamp values:

• W-timestamp(A) is the largest time-stamp of any transaction that executed write(A) successfully.

R-timestamp(Q) is the largest time-stamp of any transaction that executed read(Q) successfully.

There are two types of Time stamp based protocols:

1. Time stamp ordering protocol
2. Thomas Write Rule.

1. TIMESTAMP ORDERING PROTOCOL:

The timestamp ordering protocol provides an ordering among the transactions so that any conflicting read and write operations are executed in timestamp order.

If not such an operation is rejected and the transaction is rolled back and the rolled back transaction is given a new timestamp.

Timestamp ordering protocol works as follows −

- **If a transaction Ti issues a read(X) operation −**
  - If TS(Ti) < W-timestamp(X)
    - Operation rejected.

T1(9)	T2(10)
	W(X)
R(X)-REJECTED	

  - If TS(Ti) >= W-timestamp(X)
    - Operation executed.

T1(10)	T2(9)
	W(X)
R(X)-ALLOWED	

- **If a transaction Ti issues a write(X) operation −**
  - If TS(Ti) < R-timestamp(X) then the Operation rejected.

T1(9)	T2(10)
	R(X)
W(X)-REJECT	

    - If TS(Ti) < W-timestamp(X) then the Operation rejected and Ti rolled back.

T1(9)	T2(10)
	W(X)
W(X)-REJECT	

2.THOMAS WRITE RULE:

It ignores the outdated write operations.

It improves the Basic Timestamp Ordering Algorithm.

If transaction Ti wants to execute the WRITE operation on X:

The basic Thomas write rules are as follows:

1. If TS(T) < R_TS(X) then transaction T is aborted and rolled back, and operation is rejected.

   Example:

T1(9)	T2(10)
	R(X)
W(X)-REJECT	

2. If TS(T) < W_TS(X) then don't execute the W_item(X) operation of the transaction and continue processing.

T1(9)	T2(10)
	W(X)
	COMMIT
W(X)-OUTDATED	
COMMIT	

3. If neither condition 1 nor condition 2 occurs, then allowed to execute the WRITE operation by transaction Ti and set W_TS(X) to TS(T).

VALIDATION BASED PROTOCOLS:

Validation based protocols are also known as optimistic concurrency control technique or Certification based protocols

It is called optimistic concurrency control protocols because it hopes that all the operations of transaction are correct.

It is used to validate/ check the update operation (read/ write operation) on a data item by a Transaction.

In the validation based protocol, the transaction is executed in the following three phases:

1. Read phase: In this phase, the transaction T reads committed values of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.
2. Validation phase: In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability and everything is correct.
3. Write phase: If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back and restarted.

Here each phase has the following different timestamps:

Start(Ti): It contains the time when Ti started its execution.

Validation (T_i): It contains the time when Ti finishes its read phase and starts its validation phase.

Finish(Ti): It contains the time when Ti finishes its write phase.

- This protocol is used to determine the time stamp for the transaction for serialization using the time stamp of the validation phase, as it is the actual phase which determines if the transaction will commit or rollback.

- Hence TS(T) = validation(T).
- The serializability is determined during the validation process. It can't be decided in advance.
- While executing the transaction, it ensures a greater degree of concurrency and also less number of conflicts.
- Thus it contains transactions which have less number of rollbacks.

Advantages:

- The validation based protocol considers high priority to the greater degree of concurrency resulting in fewer conflict scenarios.
- This protocol is known for less number of rollback while maintaining the concurrency control mechanism.

Disadvantages:

- The validation based protocol may arise in the scenario of transaction starvation. This could be due to the short transaction conflicts associated with this protocol.
- The validation based protocol may not suitable for large transactions as it efficient for maintaining the shorter conflicts in the transactions.

MULTIPLE GRANULARITY

Locking Granularity: It is the size of the data item that can be locked.

There are 2 types of locking granularity:

1. Coarse granularity: It refers to large data item sizes
   Ex: Entire relation or Table
2. Fine granularity: It refers to small data item sizes
   Ex: Tuple or attribute of a relation

Multiple Granularity: It allows the data items of any size to be locked.

- It can be defined as hierarchically breaking up the database into blocks which can be locked.
- The Multiple Granularity protocol enhances concurrency and reduces lock overhead.
- It maintains the track of what to lock and how to lock.
- It makes easy to decide either to lock a data item or to unlock a data item. This type of hierarchy can be graphically represented as a tree.

Note : In multiple-granularity, the locks are acquired in top-down order, and locks must be released in bottom-up order.

There are five lock modes with multiple granularity:

Shared lock: Shared locks support read integrity.

Exclusive lock: Exclusive locks support write integrity.

Intention-shared (IS): Intention-shared (IS) indicates that one or more shared locks will be requested on some descendant node(s) i.e. It contains explicit locking at a lower level of the tree but only with shared locks.

Intention-Exclusive (IX): Intention-exclusive (IX) indicates that one or more exclusive locks will be requested on some descendant node(s) i.e. It contains explicit locking at a lower level with exclusive or shared locks.

Shared & Intention-Exclusive (SIX): Shared-intention-exclusive (SIX) indicates that the current node is locked in shared mode but that one or more exclusive locks will be requested on some descendant node(s). i.e.

MULTIPLE GRANULARITY LOCKING PROTOCOL RULES:

The multiple granularity locking (MGL) protocol consists of the following rules i.e. It requires that if a transaction attempts to lock a node, then that node must follow these rules:

1. Transaction T1 should follow the lock-compatibility matrix.
2. The root of the tree must be locked first, in any mode.
3. A node N can be locked by a transaction T in S or IS mode only if the parent node N is already locked by transaction T in either IS or IX mode.
4. A node N can be locked by a transaction T in X, IX, or SIX mode only if the parent of node N is already locked by transaction T in either IX or SIX mode.
5. A transaction T can lock a node only if it has not unlocked any node (to enforce the 2PL protocol).
6. A transaction T can unlock a node, N, only if none of the children of node N are currently locked by T.

UNIT-V

Recovery and Atomicity

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items.

Transactions are made of various operations, which are atomic in nature.

But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

When a DBMS recovers from a crash, it should maintain the following −

- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction −

- **Log Based Recovery**: Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- **Shadow paging**: Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

<u>LOG BASED RECOVERY</u>

It is used to recover from a system crash or transaction failure using a log.

It is the most commonly used structure for recording database modifications.

- The log is a small file that contains a sequence of records where the records depict the operations performed by Transaction.
- Log of each transaction is maintained in some stable storage (RAM/ shared memory) So that if any failure occurs, then it can be recovered from there.
- Before any operation is performed on the database, it will be recorded in the log.
- The process of storing the logs should be done before the actual transaction is applied in the database.

A log contains:

1. Transaction Identifier
2. Data item identifier
3. Old value
4. New value

   Example: <T_i, X_j,V_1,V_2>
   T_i: Transaction identifier used to uniquely identify a Transaction.
   X_j: Data item on which the operations are performed.
   V_1: Old value of data item
   V_2: New value of data item

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

- When the transaction is initiated, then it writes 'start' log.

  <Tn, Start>

- When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.

  <Tn, City, 'Noida', 'Bangalore' >

- When the transaction is finished, then it writes another log to indicate the end of the transaction.

  <Tn, Commit>

There are two approaches to modify the database using log based recovery:

1. **Deferred database modification**
2. **Immediate database modification.**

1. Deferred database modification:

In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

A Deffered database modification is also called No Undo/ Redo method.

Undo: Restores the old values

Redo: Updates the new values

In this log contains a new value of the data item.

TRANSACTION	LOG	DATABASE
T1		A=3000, B=2000
R(A)	<T1,START>	
A=A-500		
W(A)	<T1,A,2500>	
R(B)		
B=B+500		
W(B)	<T1,B,2500>	
COMMIT	<T1,COMMIT>	A=2500,B=2500
SYSTEM CRASH		
		REDO
		A=2500,B=2500

Example: A=3000, B=2000

The deferred modification technique occurs if the transaction does not modify the database until it has committed.

TRANSACTION	LOG	DATABASE
T1		A=3000, B=2000
R(A)	<T1,START>	
A=A-500		
W(A)	<T1,A,2500>	
R(B)		
B=B+500		
W(B)	<T1,B,2500>	
SYSTEM CRASH		

COMMIT		A=3000,
		
		B =2000

2. Immediate database modification:

- The Immediate modification technique occurs if database modification occurs while the transaction is still active.

- In this technique, the database is modified immediately after every operation. It follows an actual database modification.

TRANSACTION	LOG	DATABASE
T1		A=3000, B=2000
R(A)	<T1,START>	
A=A-500		
W(A)	<T1,A,3000,2500>	A=2500, B=2000
R(B)		
B=B+500		
W(B)	<T1,B,2000,2500>	A=2500, B=2500
COMMIT	<T1, COMMIT>	A=2500,B=2500
SYSTEM CRASH		REDO
		A=2500,B=2500

When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.

Immediate database modification is called Undo/ Redo strategy

UNDO & REDO:

1. If the log contains the record <Ti, Start> and <Ti, Commit> or <Ti, Commit>, then the Transaction Ti needs to be redone.
2. If log contains record<T_n, Start> but does not contain the record either <Ti, commit> or <Ti, abort>, then the Transaction Ti needs to be undone.

Shadow paging in DBMS

This is the method where all the transactions are executed in the primary memory or the shadow copy of database.

Once all the transactions completely executed, it will be updated to the database.

Hence, if there is any failure in the middle of transaction, it will not be reflected in the database.

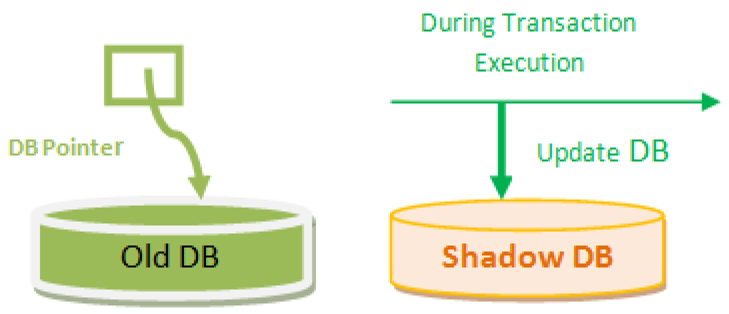
Database will be updated after all the transaction is complete.

A database pointer will be always pointing to the consistent copy of the database, and copy of the database is used by transactions to update.

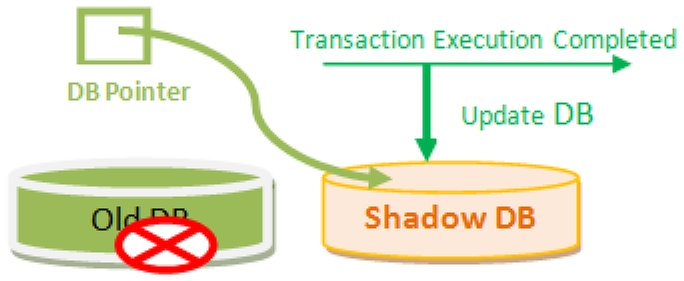
Once all the transactions are complete, the DB pointer is modified to point to new copy of DB, and old copy is deleted.

If there is any failure during the transaction, the pointer will be still pointing to old copy of database, and shadow database will be deleted.

If the transactions are complete then the pointer is changed to point to shadow DB, and old DB is deleted.



During Transaction Execution



As we can see in above diagram, the DB pointer is always pointing to consistent and stable database.

This mechanism assumes that there will not be any disk failure and only one transaction executing at a time so that the shadow DB can hold the data for that transaction.

It is useful if the DB is comparatively small because shadow DB consumes same memory space as the actual DB.

Hence it is not efficient for huge DBs.

In addition, it cannot handle concurrent execution of transactions. It is suitable for one transaction at a time.

Recovery with Concurrent Transactions

When more than one transaction are being executed in parallel, the logs are interleaved.

At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering.

To ease this situation, most modern DBMS use the concept of 'checkpoints'.

Checkpoint

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system.

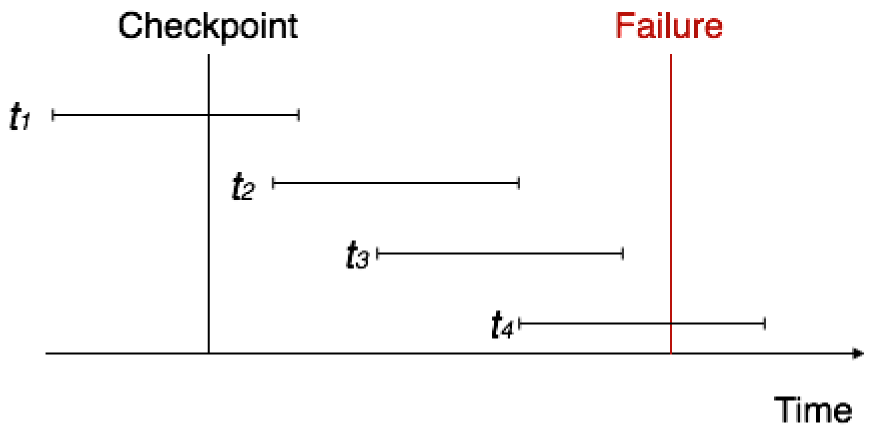
As time passes, the log file may grow too big to be handled at all.

Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk.

Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Recovery

When a system with concurrent transactions crashes and recovers, it behaves in the following manner −



- The recovery system reads log files from the end to start. It reads log files from T4 to T1.
- Recovery system maintains two lists, a redo-list, and an undo-list.
- The transaction is put into redo state if the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs.
- **For example:** In the log file, transaction T2 and T3 will have <Tn, Start> and <Tn, Commit>. The T1 transaction will have only <Tn, commit> in the log file. That's why the transaction is committed after the checkpoint is crossed. Hence it puts T1, T2 and T3 transaction into redo list.
- The transaction is put into undo state if the recovery system sees a log with <Tn, Start> but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed.
- **For example:** Transaction T4 will have <Tn, Start>. So T4 will be put into undo list since this transaction is not yet complete and failed amid.

TRIGGERS:

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.

Trigger is a PL/SQL procedure stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).

A database definition (DDL) statement (CREATE, ALTER, or DROP).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

PL/SQL: Parts of a Trigger

Whenever a trigger is created, it contains the following three sequential parts:

- **Triggering Event or Statement:** The statements due to which a trigger occurs is called triggering event or statement. Such statements can be DDL statements, DML statements or any database operation, executing which gives rise to a trigger.
- **Trigger Restriction:** The condition or any limitation applied on the trigger is called trigger restriction. Thus, if such a condition is **TRUE** then trigger occurs otherwise it does not occur.
- **Trigger Action:** The body containing the executable statements that is to be executed when trigger occurs that is with the execution of Triggering statement and upon evaluation of Trigger restriction as **True** is called Trigger Action.

PL/SQL: Types of Triggers

Triggers can be classified into three categories:

1. Level Triggers
2. Event Triggers
3. Timing Triggers

which are further divided into different parts.

1. Level Triggers

There are 2 different types of level triggers, they are:

ROW LEVEL TRIGGERS:

- It fires for every record that got affected with the execution of DML statements like INSERT, UPDATE, DELETE etc.
- It always use a FOR EACH ROW clause in a triggering statement.

STATEMENT LEVEL TRIGGERS:

- It fires once for each statement that is executed.

2. **Event Triggers**

There are 3 different types of event triggers, they are:

DDL EVENT TRIGGER/ DATABASE LEVEL TRIGGERS

- It fires with the execution of every DDL statement(CREATE, ALTER, DROP, TRUNCATE).

DML EVENT TRIGGER/ TABLE LEVEL TRIGGERS

- It fires with the execution of every DML statement(INSERT, UPDATE, DELETE).

DATABASE EVENT TRIGGER

- It fires with the execution of every database operation which can be LOGON, LOGOFF, SHUTDOWN, SERVERERROR etc.

3. Timing Triggers

There are 2 different types of timing triggers, they are:

- BEFORE TRIGGER
    - It fires before executing DML statement.
    - Triggering statement may or may not executed depending upon the before condition block.
- AFTER TRIGGER
    - It fires after executing DML statement.

Syntax for creating trigger:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

CURSORS:

Cursor is a memory location which is used to run SQL commands.

When an SQL statement is processed, Oracle creates a memory area known as context area.

A cursor is a pointer to this context area.

It contains all information needed for processing the statement.

In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

1) Implicit Cursors: A Cursor declared by Oracle.
2) Explicit Cursors: A Cursor declared by User.

1) Implicit Cursors: These are declared after the execution of DML command.

The name of the cursor is SQL.

All the activities related to cursor like i) Opening the cursor ii) Processing the data in the cursor iii) closing the cursor
are done automatically.
Hence these cursors are called Implicit cursors.

Implicit Cursor Attributes:

There are four Implicit cursor attributes
1) SQL%ISOPEN
2) SQL%FOUND
3) SQL%NOTFOUND
4) SQL%ROWCOUNT

1) SQL%ISOPEN:

It is a boolean attribute. It always returns false. It is not used in programming as it always returns false.

2) SQL%FOUND:

It is a boolean attribute.
Returns TRUE -- if the SQL command effects the data.
Returns FALSE -- if the SQL commands do not effect the data.

3) SQL%NOTFOUND:

It is a boolean attribute
Returns TRUE -- if the SQL command do not effect the data.
Returns FALSE -- if the SQL command effects the data
Note: It is exactly negation to SQL%FOUND

4) SQL%ROWCOUNT:

Returns no of rows effected by the SQL command.

Explicit Cursors:

Explicit cursors are used to run select stmt which returns more than one row in a PL/SQL block

Steps to use Explicit cursors:

Step 1: Declare the cursor
Step 2: Open the cursor
Srep 3: Fetch the data from the cursor to the local variables
Step 4: close the cursor

Syntax of the above four steps:

Step 1: Declaring the cursor

cursor < cursor_name>
is < select stmt >;

Ex: cursor c1 is select * from emp;

step 2: Open the cursor : At this point data is loaded in cursor.

open < cursor_name >;

Ex: open c1;

step 3: Fetch the data(records) from the cursor to the local variables

fetch < cursor_name > into < var1 > , < var2> , ..., < varn >;;

Ex: fetch c1 into x,y,z,…;

step 4: close the cursor

close < cursor_name>;

Ex: close c1;

Explicit cursor attributes:

There are four explicit cursor attributes

1) %ISOPEN
2) %FOUND
3) %NOTFOUND
4) %ROWCOUNT

1) %ISOPEN:

It is a boolean attribute.
Returns TRUE -- if the cursor is open
Returns FALSE -- if the cursor is closed

2) %FOUND:

It is a boolean attribute
Returns TRUE -- if the fetch stmt is successfull
Returns FALSE -- if the fetch stmt fails

3)
%NOTFOUND:

It is boolean attribute
Returns TRUE -- if the fetch stmt fails.
Returns FALSE -- if the fetch stmt is successfull

Note: 1) It is exactly opposite to %FOUND attribute
2) This attribute is used to break the loop of the fetch stmt.

4) %ROWCOUNT:

Returns no of rows fetched by the fetch stmt.

<u>QUERY OPTIMIZATION</u>

Sql Statements are used to retrieve data from the database.

We can get same results by writing different sql queries.

But use of the best query is important when performance is considered. So we need to sql query tuning based on the requirement.

Though a system can create multiple plans for a query, the chosen method should be the best of all. It can be done by comparing each possible plan in terms of their estimated cost. For calculating the net estimated cost of any plan, the cost of each operation within a plan should be determined and combined to get the net estimated cost of the query evaluation plan.

The cost estimation of a query evaluation plan is calculated in terms of various resources that include:

- Number of disk accesses
- Execution time taken by the CPU to execute a query
- Communication costs in distributed or parallel database systems.

To estimate the cost of a query evaluation plan, we use the number of blocks transferred from the disk, and the number of disks seeks. Suppose the disk has an average block access time of t_s seconds and takes an average of t_T seconds to transfer x data blocks. The block access time is the sum of disk seeks time and rotational latency. It performs S seeks than the time taken will be $\mathbf{b*t_T + S*t_S}$ seconds. If t_T=0.1 ms, t_S =4 ms, the block size is 4 KB, and its transfer rate is 40 MB per second. With this, we can easily calculate the estimated cost of the given query evaluation plan.

Generally, for estimating the cost, we consider the worst case that could happen. The users assume that initially, the data is read from the disk only. But there must be a chance that the information is already present in the main memory. However, the users usually ignore this effect, and due to this, the actual cost of execution comes out less than the estimated value.

The response time, i.e., the time required to execute the plan, could be used for estimating the cost of the query evaluation plan. But due to the following reasons, it becomes difficult to calculate the response time without actually executing the query evaluation plan:

- When the query begins its execution, the response time becomes dependent on the contents stored in the buffer. But this information is difficult to retrieve when the query is in optimized mode, or it is not available also.
- When a system with multiple disks is present, the response time depends on an interrogation that in "what way accesses are distributed among the disks?". It is difficult to estimate without having detailed knowledge of the data layout present over the disk.
- Consequently, instead of minimizing the response time for any query evaluation plan, the optimizers finds it better to reduce the total **resource consumption** of the query plan. Thus to estimate the cost of a query evaluation plan, it is good to minimize the resources used for accessing the disk or use of the extra resources.