

UNIT-4

TRANSACTION MANAGEMENT IN DBMS:

- A **transaction** is a set of logically related operations.
- Now that we understand what is transaction, we should understand what are the problems associated with it.
- The main problem that can happen during a transaction is that the transaction can fail before finishing the all the operations in the set. This can happen due to power failure, system crash etc.
- This is a serious problem that can leave database in an inconsistent state. Assume that transaction fail after third operation (see the example above) then the amount would be deducted from your account but your friend will not receive it.

To solve this problem, we have the following two operations

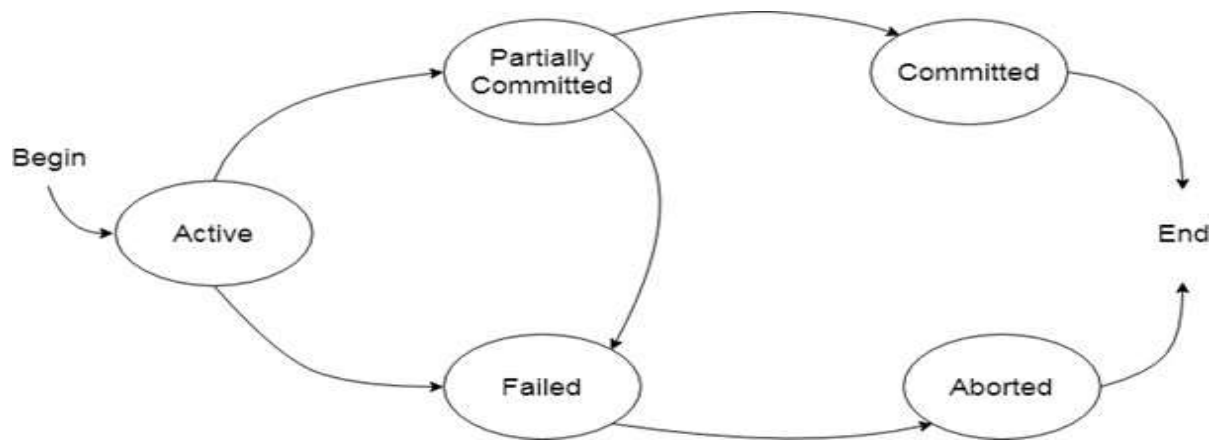
Commit: If all the operations in a transaction are completed successfully then commit those changes to the database permanently.

Rollback: If any of the operation fails then rollback all the changes done by previous operations.

STATES OF TRANSACTION

Transactions can be implemented using SQL queries and Server. In the below-given diagram, you can see how transaction states works.





Active state

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

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

- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
 - In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.
-

Aborted

- If any of the checks fail and the 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.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
 1. Re-start the transaction
 2. Kill the transaction

TRANSACTION PROPERTY

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

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

Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

IMPLEMENTATION OF ATOMICITY AND DURABILITY

The recovery-management component of a database system can support atomicity and durability by a variety of schemes.

E.g. the shadow-database scheme:

Shadow copy:

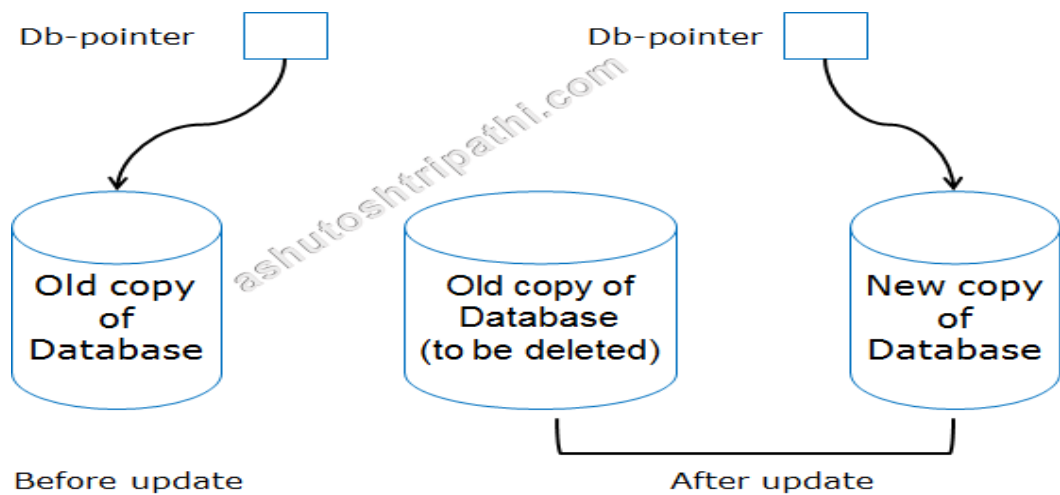
- In the shadow-copy scheme, a transaction that wants to update the database first creates a complete copy of the database.
- All updates are done on the new database copy, leaving the original copy, the shadow copy, untouched. If at any point the transaction has to be aborted, the system merely deletes the new copy. The old copy of the database has not been affected.
- This scheme is based on making copies of the database, called shadow copies, assumes that only one transaction is active at a time.
- The scheme also assumes that the database is simply a file on disk. A pointer called db-pointer is maintained on disk; it points to the current copy of the database.

If the transaction completes, it is committed as follows:

- First, the operating system is asked to make sure that all pages of the new copy of the database have been written out to disk. (Unix systems use the flush command for this purpose.)
- After the operating system has written all the pages to disk, the database system updates the pointer db-pointer to point to the new copy of the database;
- the new copy then becomes the current copy of the database. The old copy of the database is then deleted.

Figure below depicts the scheme, showing the database state before and after the update.

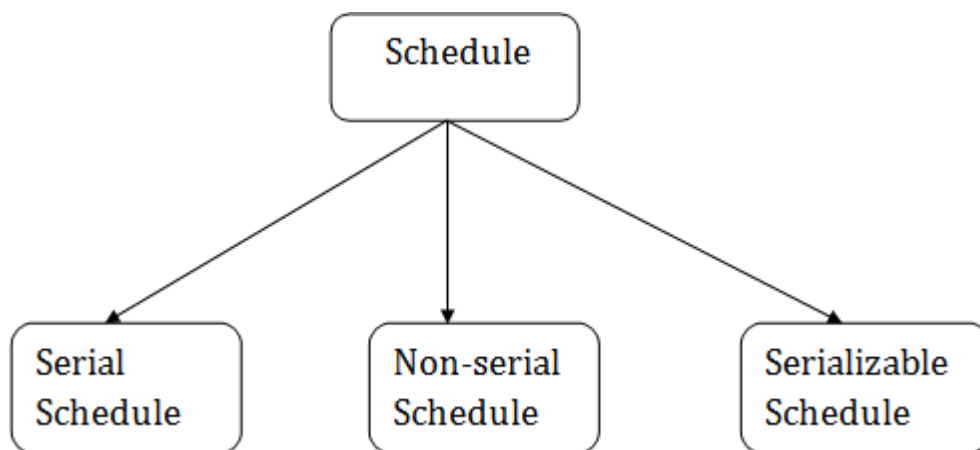




Shadow-copy technique for atomicity and durability

SCHEDULE

A series of operation from one transaction to another transaction is known as schedule. It is used to preserve the order of the operation in each of the individual transaction.



1. SERIAL SCHEDULE

The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

For example: Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:

1. Execute all the operations of T1 which was followed by all the operations of T2.
2. Execute all the operations of T2 which was followed by all the operations of T1.
 - In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.
 - In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

2. NON-SERIAL SCHEDULE

- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. It has interleaving of operations.

3. SERIALIZABLE SCHEDULE

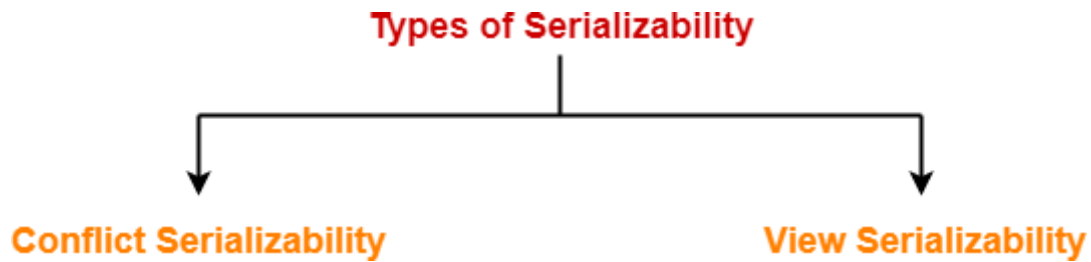
- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule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

SERIALIZABILITY IN DBMS

- Some non-serial schedules may lead to inconsistency of the database.
 - Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.
-

Types of Serializability

Serializability is mainly of two types-



1. Conflict Serializability
2. View Serializability

Conflict Serializability

If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a **conflict serializable schedule**.

Conflicting Operations

Two operations are called as **conflicting operations** if all the following conditions hold true for them-

- Both the operations belong to different transactions
- Both the operations are on the same data item
- At least one of the two operations is a write operation

Example-

Consider the following schedule-

Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

In this schedule,

- W1 (A) and R2 (A) are called as conflicting operations.
- This is because all the above conditions hold true for them.

Checking Whether a Schedule is Conflict Serializable Or Not-

Follow the following steps to check whether a given non-serial schedule is conflict serializable or not-

Follow the following steps to check whether a given non-serial schedule is conflict serializable or not-

Step-01:

Find and list all the conflicting operations.

Step-02:

Start creating a precedence graph by drawing one node for each transaction.

Step-03:

Draw an edge for each conflict pair such that if $X_i(V)$ and $Y_j(V)$ forms a conflict pair then draw an edge from T_i to T_j .

- This ensures that T_i gets executed before T_j .

Step-04:

Check if there is any cycle formed in the graph.

- If there is no cycle found, then the schedule is conflict serializable otherwise not.

VIEW SERIALIZABILITY?

View Serializability is a process to find out that a given schedule is view serializable or not.

To check whether a given schedule is view serializable, we need to check whether the given schedule is **View Equivalent** to its serial schedule. Lets take an example to understand what I mean by that.

View Serializability

- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

Two schedules S_1 and S_2 are said to be view equivalent if they satisfy the following conditions:

1. Initial Read:

An initial read of both schedules must be the same. Suppose two schedule S_1 and S_2 . In schedule S_1 , if a transaction T_1 is reading the data item A , then in S_2 , transaction T_1 should also read A .

T1	T2
Read(A)	Write(A)

Schedule S1

T1	T2
Read(A)	Write(A)

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)	Write(A)	Read(A)

Schedule S1

T1	T2	T3
Write(A)	Write(A)	<u>Read(A)</u>

Schedule S2

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)	Read(A)	Write(A)

Schedule S1

T1	T2	T3
Write(A)	Read(A)	Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

T1	T2	T3
Read(A)	Write(A)	Write(A)
Write(A)		

Recoverability of Schedule

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But

TRANSACTION ISOLATION LEVELS IN DBMS

some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

The SQL standard defines four isolation levels :

1. **Read Uncommitted** – Read Uncommitted is the lowest isolation level. In this level, one transaction may read not yet committed changes made by other transaction, thereby allowing dirty reads. In this level, transactions are not isolated from each other.
2. **Read Committed** – This isolation level guarantees that any data read is committed at the moment it is read. Thus it does not allow dirty read. The transaction holds a read or write lock on the current row, and thus prevent other transactions from reading, updating or deleting it.
3. **Repeatable Read** – This is the most restrictive isolation level. The transaction holds read locks on all rows it references and writes locks on all rows it inserts, updates, or

deletes. Since other transaction cannot read, update or delete these rows, consequently it avoids non-repeatable read.

4. **Serializable** – This is the Highest isolation level. A *serializable* execution is guaranteed to be serializable. Serializable execution is defined to be an execution of operations in which concurrently executing transactions appears to be serially executing.

FAILURE CLASSIFICATION

To find that where the problem has occurred, we generalize a failure into the following categories:

1. Transaction failure
2. System crash
3. Disk failure

1. Transaction failure

The transaction failure occurs when it fails to execute or when it reaches a point from where it can't go any further. If a few transaction or process is hurt, then this is called as transaction failure.

Reasons for a transaction failure could be -

1. **Logical errors:** If a transaction cannot complete due to some code error or an internal error condition, then the logical error occurs.
2. **Syntax error:** It occurs where the DBMS itself terminates an active transaction because the database system is not able to execute it. **For example,** The system aborts an active transaction, in case of deadlock or resource unavailability.

2. System Crash

- System failure can occur due to power failure or other hardware or software failure. **Example:** Operating system error.
-

Fail-stop assumption: In the system crash, non-volatile storage is assumed not to be corrupted.

3. Disk Failure

- It occurs where hard-disk drives or storage drives used to fail frequently. It was a common problem in the early days of technology evolution.
- Disk failure occurs due to the formation of bad sectors, disk head crash, and unreachability to the disk or any other failure, which destroy all or part of disk storage.

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

1. Decrease waiting time or turnaround time.
2. Improve response time
3. Increased throughput or resource utilization.

Problems with Concurrent Execution

In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

- 1: Lost Update Problems (W - W Conflict)
 2. Dirty Read Problems (W-R Conflict)
 3. Unrepeatable Read Problem (W-R Conflict)
-

1. Lost update problem (Write – Write conflict)

This type of problem occurs when two transactions in database access the same data item and have their operations in an interleaved manner that makes the value of some database item incorrect.

If there are two transactions T1 and T2 accessing the same data item value and then update it, then the second record overwrites the first record.

Example: Let's take the value of A is 100

Time	Transaction T1	Transaction T2
t1	Read(A)	
t2	$A = A - 50$	
t3		Read(A)
t4		$A = A + 50$
t5	Write(A)	
t6		Write(A)

Here,

- At t1 time, T1 transaction reads the value of A i.e., 100.
 - At t2 time, T1 transaction deducts the value of A by 50.
 - At t3 time, T2 transactions read the value of A i.e., 100.
 - At t4 time, T2 transaction adds the value of A by 150.
 - At t5 time, T1 transaction writes the value of A data item on the basis of value seen at time t2 i.e., 50.
-

- At t6 time, T2 transaction writes the value of A based on value seen at time t4 i.e., 150.
- So at time T6, the update of Transaction T1 is lost because Transaction T2 overwrites the value of A without looking at its current value.
- Such type of problem is known as the Lost Update Problem.

Dirty read problem (W-R conflict)

This type of problem occurs when one transaction T1 updates a data item of the database, and then that transaction fails due to some reason, but its updates are accessed by some other transaction.

Example: Let's take the value of A is 100

Time	Transaction T1	Transaction T2
t1	Read(A)	
t2	A=A+20	
t3	Write(A)	
t4		Read(A)
t5		A=A+30
t6		Write(A)
t7	Write(B)	

Here,

- At t1 time, T1 transaction reads the value of A i.e., 100.



- At t2 time, T1 transaction adds the value of A by 20.
- At t3 time, T1 transaction writes the value of A (120) in the database.
- At t4 time, T2 transactions read the value of A data item i.e., 120.
- At t5 time, T2 transaction adds the value of A data item by 30.
- At t6 time, T2 transaction writes the value of A (150) in the database.
- At t7 time, a T1 transaction fails due to power failure then it is rollback according to atomicity property of transaction (either all or none).
- So, transaction T2 at t4 time contains a value which has not been committed in the database. The value read by the transaction T2 is known as a dirty read.

Unrepeatable read (R-W Conflict)

It is also known as an inconsistent retrieval problem. If a transaction T_1 reads a value of data item twice and the data item is changed by another transaction T_2 in between the two read operation. Hence T_1 access two different values for its two read operation of the same data item.

Example: Let's take the value of A is 100

Time	Transaction T1	Transaction T2
t1	Read(A)	
t2		Read(A)
t3		$A = A + 30$
t4		Write(A)
t5	Read(A)	

Here,

- At t1 time, T1 transaction reads the value of A i.e., 100.
-

- At t2 time, T2 transaction reads the value of A i.e., 100.
- At t3 time, T2 transaction adds the value of A data item by 30.
- At t4 time, T2 transaction writes the value of A (130) in the database.
- Transaction T2 updates the value of A. Thus, when another read statement is performed by transaction T1, it accesses the new value of A, which was updated by T2. Such type of conflict is known as R-W conflict.

CONCURRENCY CONTROL

Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

The concurrency control protocols ensure the *atomicity*, *consistency*, *isolation*, *durability* and *serializability* of the concurrent execution of the database transactions.

Therefore, these protocols are categorized as:

- Lock Based Concurrency Control Protocol
- Time Stamp Concurrency Control Protocol
- Validation Based Concurrency Control Protocol

Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

1. Shared lock:

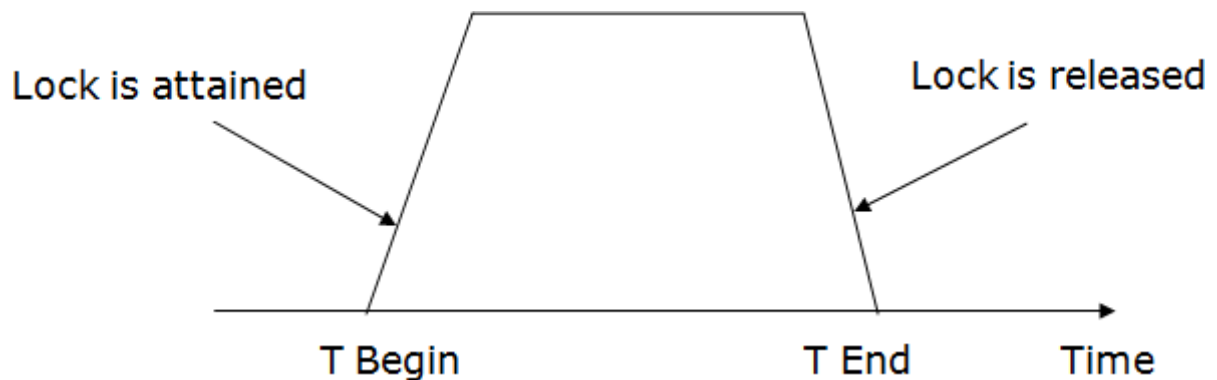
- It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
 - It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.
-

2. Exclusive lock:

- In the exclusive lock, the data item can be both reads as well as written by the transaction.
- This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

TWO-PHASE LOCKING (2PL)

- The two-phase locking protocol divides the execution phase of the transaction into three parts.
- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
- In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



There are two phases of 2PL:

Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

Example:

	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

The following way shows how unlocking and locking work with 2-PL.

Transaction T1:

- **Growing phase:** from step 1-3
 - **Shrinking phase:** from step 5-7
-

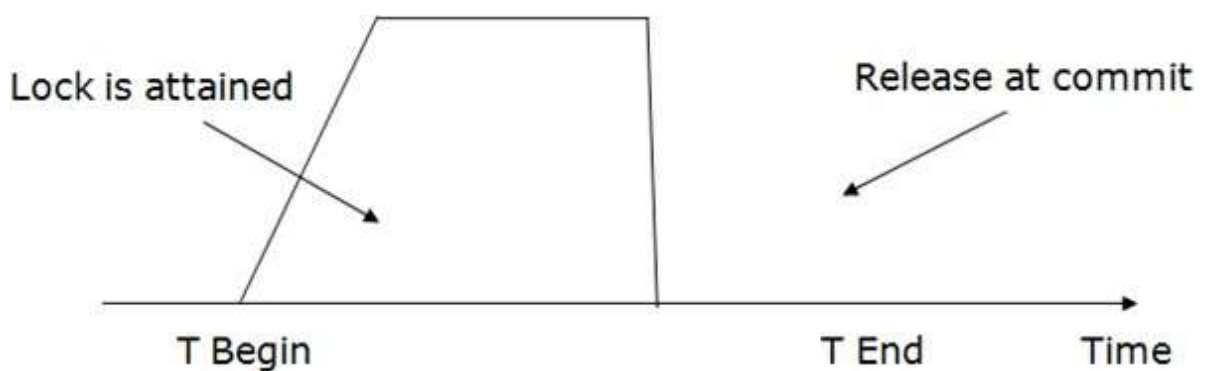
- **Lock point:** at 3

Transaction T2:

- **Growing phase:** from step 2-6
- **Shrinking phase:** from step 8-9
- **Lock point:** at 6

4. Strict Two-phase locking (Strict-2PL)

- The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
- The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
- Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
- Strict-2PL protocol does not have shrinking phase of lock release.



TIMESTAMP ORDERING PROTOCOL

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction T_i issues a **Read (X)** operation:

- If $W_TS(X) > TS(T_i)$ then the operation is rejected.
- If $W_TS(X) \leq TS(T_i)$ then the operation is executed.
- Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction T_i issues a **Write(X)** operation:

- If $TS(T_i) < R_TS(X)$ then the operation is rejected.
- If $TS(T_i) < W_TS(X)$ then the operation is rejected and T_i is rolled back otherwise the operation is executed.

Where,

$TS(T_i)$ denotes the timestamp of the transaction T_i .

$R_TS(X)$ denotes the Read time-stamp of data-item X .

$W_TS(X)$ denotes the Write time-stamp of data-item X .

Validation Based Protocol

Validation phase is also known as optimistic concurrency control technique. In the validation based protocol, the transaction is executed in the following three phases:



1. **Read phase:** In this phase, the transaction T is read and executed. It is used to read the value 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.
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.

Here each phase has the following different timestamps:

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

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

Finish(T_i): It contains the time when T_i 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 $TS(T) = \text{validation}(T)$.
- 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.

THOMAS WRITE RULE

Thomas Write Rule provides the guarantee of serializability order for the protocol. It improves the Basic Timestamp Ordering Algorithm.

The basic Thomas write rules are as follows:

- If $TS(T) < R_TS(X)$ then transaction T is aborted and rolled back, and operation is rejected.
- If $TS(T) < W_TS(X)$ then don't execute the $W_item(X)$ operation of the transaction and continue processing.
- If neither condition 1 nor condition 2 occurs, then allowed to execute the WRITE operation by transaction T_i and set $W_TS(X)$ to $TS(T)$.

MULTIPLE GRANULARITY

Let's start by understanding the meaning of granularity.

Granularity: It is the size of data item allowed to lock.

Multiple Granularity:

- 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.
- The first level or higher level shows the entire database.
- The second level represents a node of type area. The higher level database consists of exactly these areas.
- The area consists of children nodes which are known as files. No file can be present in more than one area.
- Finally, each file contains child nodes known as records. The file has exactly those records that are its child nodes. No records represent in more than one file.
- Hence, the levels of the tree starting from the top level are as follows:
 - Database



- Area
- File
- Record

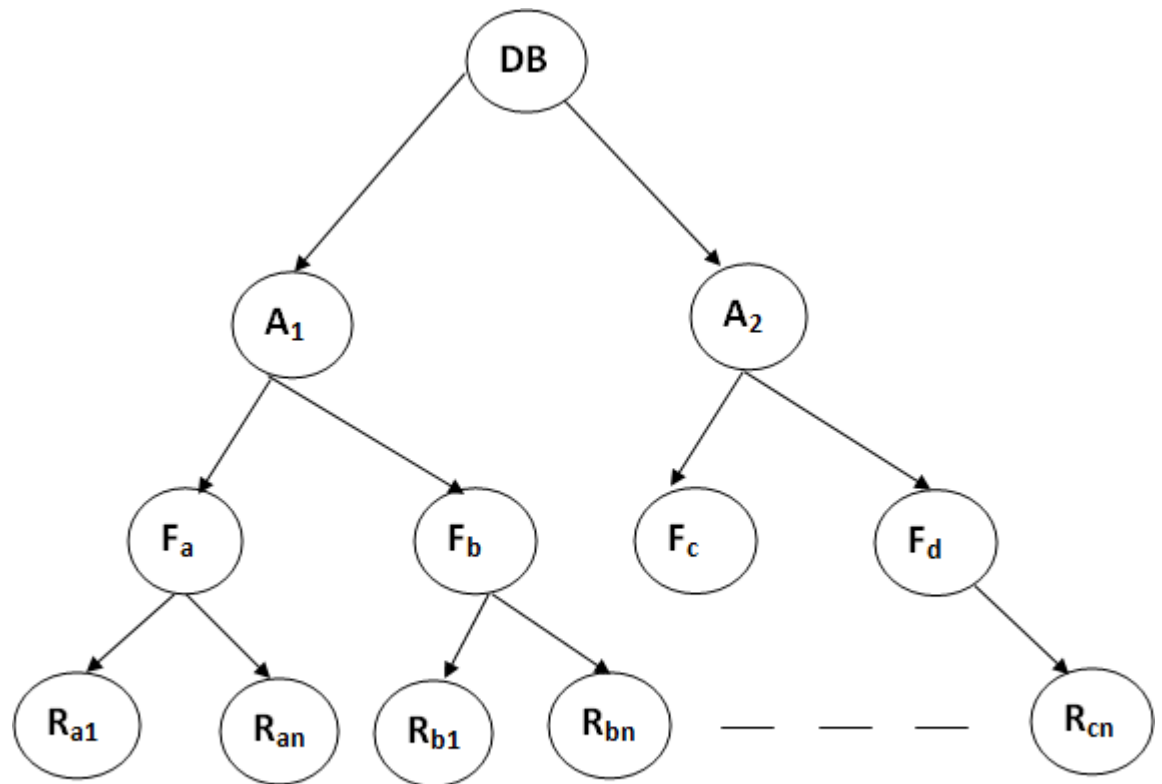


Figure: Multi Granularity tree Hierarchy

