



Transaction

DBMS-5

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

Example: Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

1. $\text{Open_Account}(X)$
2. $\text{Old_Balance} = X.\text{balance}$
3. $\text{New_Balance} = \text{Old_Balance} - 800$
4. $X.\text{balance} = \text{New_Balance}$
5. $\text{Close_Account}(X)$

Y's Account

1. $\text{Open_Account}(Y)$
2. $\text{Old_Balance} = Y.\text{balance}$
3. $\text{New_Balance} = \text{Old_Balance} + 800$
4. $Y.\text{balance} = \text{New_Balance}$
5. $\text{Close_Account}(Y)$

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

1. $R(X);$
2. $X = X - 500;$

3. 3. W(X);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

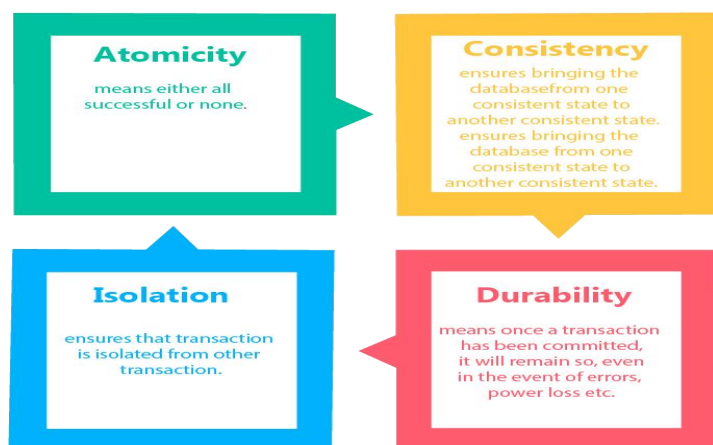
Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.

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



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.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2
Read(A) A:= A-100 Write(A)	Read(B) Y:= Y+100 Write(B)

After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.

If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

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.

For example: The total amount must be maintained before or after the transaction.

1. Total before T occurs = $600 + 300 = 900$
2. Total after T occurs = $500 + 400 = 900$

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

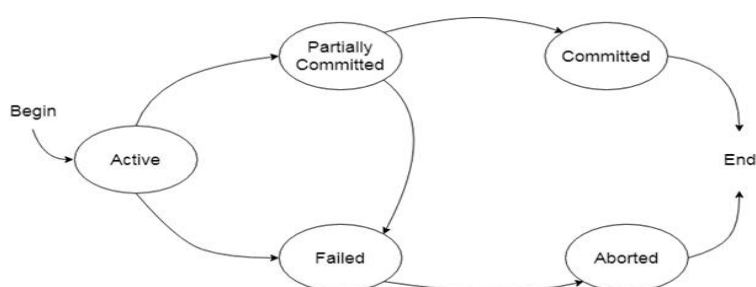
- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction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

States of Transaction

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.
- 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

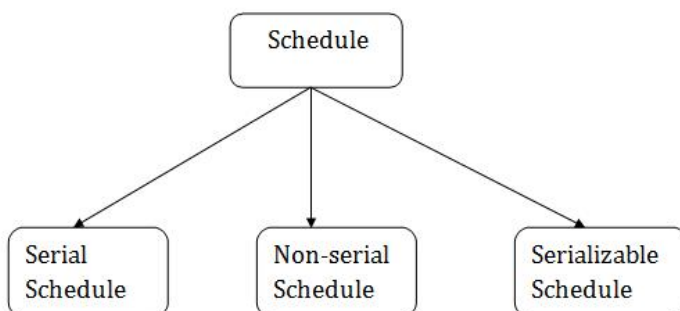
- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

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

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.

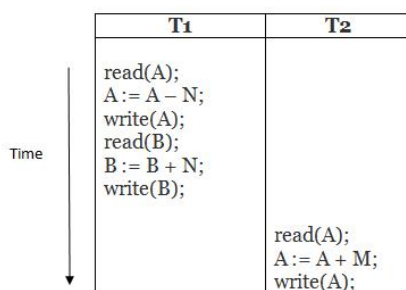
2. Non-serial Schedule

- 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

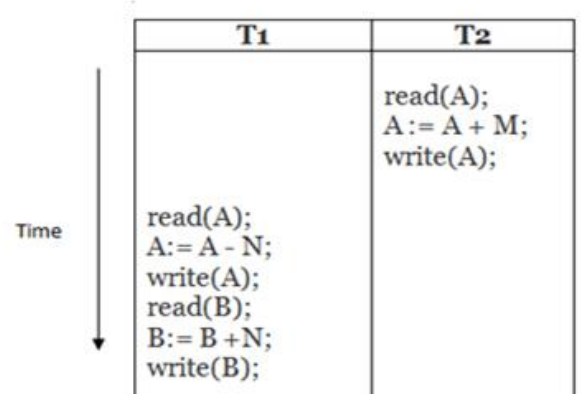
- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule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

(a)



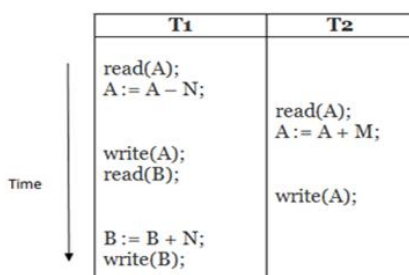
Schedule A

(b)



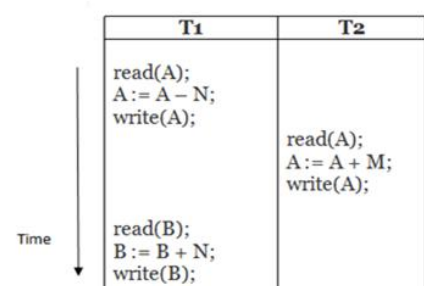
Schedule B

(c)



Schedule C

(d)



Schedule D

Here,
Schedule A and
Schedule B are
serial schedule.

Schedule C and Schedule D are Non-serial schedule.

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

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 S1 and S2 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 S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 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

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

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.

Example:

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

Schedule S

With 3 transactions, the total number of possible schedule

1. $= 3! = 6$
2. $S1 = \langle T1 \ T2 \ T3 \rangle$
3. $S2 = \langle T1 \ T3 \ T2 \rangle$
4. $S3 = \langle T2 \ T3 \ T1 \rangle$
5. $S4 = \langle T2 \ T1 \ T3 \rangle$
6. $S5 = \langle T3 \ T1 \ T2 \rangle$
7. $S6 = \langle T3 \ T2 \ T1 \rangle$

Taking first schedule S1:

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

Schedule S1

Step 1: final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Hence, view equivalent serial schedule is:

1. $T1 \rightarrow T2 \rightarrow T3$

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

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A = A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

The above table 1 shows a schedule which has two transactions. T1 reads and writes the value of A and that value is read and written by T2. T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1. T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed. So this type of schedule is known as irrecoverable schedule.

Irrecoverable schedule: The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A = A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

The above table 2 shows a schedule with two transactions. Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have to rollback T1. T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is recoverable with cascade rollback.

Recoverable with cascading rollback: The schedule will be recoverable with cascading rollback if Tj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A = A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

The above Table 3 shows a schedule with two transactions. Transaction T1 reads and write A and commits, and that value is read and written by T2. So this is a cascade less recoverable schedule.

Deadlock in DBMS

A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.

For example: In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.

Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a

standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

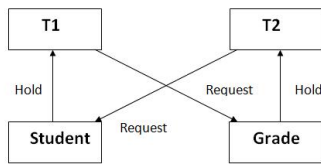


Figure: Deadlock in DBMS

DBMS Concurrency Control

Concurrency Control is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.

But before knowing about concurrency control, we should know about concurrent execution.

Concurrent Execution in DBMS

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.
- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

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:

Problem 1: Lost Update Problems (W – W Conflict)

The problem occurs when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent.

For example:

Consider the below diagram where two transactions T_x and T_y , are performed on the same account A where the balance of account A is \$300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	$A = A - 50$	—
t_3	—	READ (A)
t_4	—	$A = A + 100$
t_5	—	—
t_6	WRITE (A)	—
t_7	—	WRITE (A)

LOST UPDATE PROBLEM

- At time t_1 , transaction T_x reads the value of account A, i.e., \$300 (only read).
- At time t_2 , transaction T_x deducts \$50 from account A that becomes \$250 (only deducted and not updated/write).
- Alternately, at time t_3 , transaction T_y reads the value of account A that will be \$300 only because T_x didn't update the value yet.
- At time t_4 , transaction T_y adds \$100 to account A that becomes \$400 (only added but not updated/write).
- At time t_6 , transaction T_x writes the value of account A that will be updated as \$250 only, as T_y didn't update the value yet.
- Similarly, at time t_7 , transaction T_y writes the values of account A, so it will write as done at time t_4 that will be \$400. It means the value written by T_x is lost, i.e., \$250 is lost.

Hence data becomes incorrect, and database sets to inconsistent.

Dirty Read Problems (W-R Conflict)

The dirty read problem occurs when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.

For example:

Consider two transactions T_x and T_y in the below diagram performing read/write operations on account A where the available balance in account A is \$300:

Time	T_x	T_y
t_1	READ (A)	—
t_2	$A = A + 50$	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

- At time t_1 , transaction T_x reads the value of account A, i.e., \$300.
- At time t_2 , transaction T_x adds \$50 to account A that becomes \$350.
- At time t_3 , transaction T_x writes the updated value in account A, i.e., \$350.
- Then at time t_4 , transaction T_y reads account A that will be read as \$350.
- Then at time t_5 , transaction T_x rollbacks due to server problem, and the value changes back to \$300 (as initially).

- But the value for account A remains \$350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Unrepeatable Read Problem (W-R Conflict)

Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.

For example:

Consider two transactions, T_X and T_Y , performing the read/write operations on account A, having an available balance = \$300. The diagram is shown below:

Time	T_X	T_Y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	$A = A + 100$
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

- At time t_1 , transaction T_X reads the value from account A, i.e., \$300.
- At time t_2 , transaction T_Y reads the value from account A, i.e., \$300.
- At time t_3 , transaction T_Y updates the value of account A by adding \$100 to the available balance, and then it becomes \$400.
- At time t_4 , transaction T_Y writes the updated value, i.e., \$400.
- After that, at time t_5 , transaction T_X reads the available value of account A, and that will be read as \$400.
- It means that within the same transaction T_X , it reads two different values of account A, i.e., \$ 300 initially, and after updation made by transaction T_Y , it reads \$400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Thus, in order to maintain consistency in the database and avoid such problems that take place in concurrent execution, management is needed, and that is where the concept of Concurrency Control comes into role.

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.

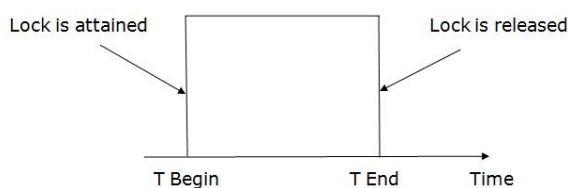
There are four types of lock protocols available:

1. Simplistic lock protocol

It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

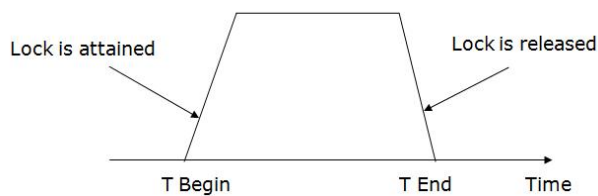
2. Pre-claiming Lock Protocol

- Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
- Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.
- If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
- If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.



3. 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:

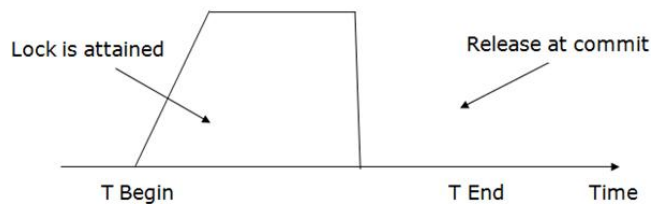
- **Growing phase:** from step 1-3
- **Shrinking phase:** from step 5-7
- **Lock point:** at 3

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.



It does not have cascading abort as 2PL does.

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.
- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

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 .

Advantages and Disadvantages of TO protocol:

- TO protocol ensures serializability since the precedence graph is as follows:



Image: Precedence Graph for TS ordering

- TS protocol ensures freedom from deadlock that means no transaction ever waits.
- But the schedule may not be recoverable and may not even be cascade-free.

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 $TS(T)$.

If we use the Thomas write rule then some serializable schedule can be permitted that does not conflict serializable as illustrate by the schedule in a given figure:

T1	T2
R(A)	W(A) Commit
W(A) Commit	

Figure: A Serializable Schedule that is not Conflict Serializable

In the above figure, T_1 's read and precedes T_1 's write of the same data item. This schedule does not conflict serializable.

Thomas write rule checks that T2's write is never seen by any transaction. If we delete the write operation in transaction T2, then conflict serializable schedule can be obtained which is shown in below figure.

T1	T2
R(A)	Commit
W(A) Commit	

Figure: A Conflict Serializable Schedule

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.

For example: Consider a tree which has four levels of nodes.

- 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:

1. Database
2. Area
3. File
4. Record

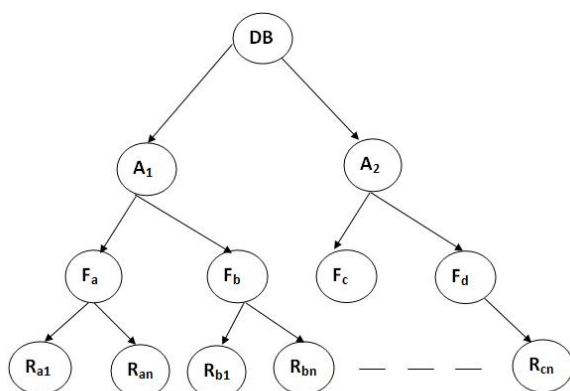


Figure: Multi Granularity tree Hierarchy

In this example, the highest level shows the entire database. The levels below are file, record, and fields. There are three additional lock modes with multiple granularity:

Checkpoint

- The checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk.
- The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of the transaction, the log files will be created.
- When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on.
- The checkpoint is used to declare a point before which the DBMS was in the consistent state, and all transactions were committed.

Recovery with Concurrent Transaction

- Whenever more than one transaction is being executed, then the interleaved of logs occur. During recovery, it would become difficult for the recovery system to backtrack all logs and then start recovering.
- To ease this situation, 'checkpoint' concept is used by most DBMS.

As we have discussed **checkpoint** in Transaction Processing Concept of this tutorial, so you can go through the concepts again to make things more clear.