DBE : UNIT 5

Q1 WHAT IS TRANSACTION? GIVE ACID PROPERTIES OF TRANSACTION.

ANS :

A collection of several operations on the database appears to be a single unit from the point of view of the database user. Collections of operations that form a single logical unit of work are called transactions.

A transaction is a unit of program execution that accesses and possibly updates various data items.

ACID properties of transaction:-

1. Atomicity :-

• All or nothing.

• Either all the statements (operations) of the transaction should execute or nothing should execute.

• If some of the statements of transaction are executed, they need to be undone.

• Bring back to the state before the start of transaction.

2. Consistency :-

• Database should always be in consistent (valid, legal) state.
• There should not be any kind of inconsistency.
• Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

3. Isolation :-

• It must appear for every transaction that, it is the only transaction that is executing currently.
• Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions Ti and Tj, it appears to Ti that either Tj has finished execution before Ti started, or Tj will start execution after Ti finishes.
• Thus, each transaction is unaware of other transactions executing concurrently in the system.

4. Durability :-

• All the changes done by transaction to the database should be permanent in nature.
• After a transaction completes successfully, the changes it has made to the database should persist, even if there are system failures.

Q2 Explain different states of transaction. Draw and explain abstract transaction model.

Ans

-- A transaction may undergo through various states in its lifetime.

1)Aborted-

In the absence of failures, all transactions must complete successfully.

A transaction may not always complete its execution successfully. Such a transaction is termed aborted.

2)**rolled back**-

To ensure the atomicity property, an aborted transaction must have no effect on the state of the database. Any changes that the aborted transaction made to the database must be undone. Once the changes caused by an aborted transaction have been undone, we say that the transaction has been rolled back. It is part of the responsibility of the recovery scheme to manage transaction aborts.

3) committed-

A transaction that completes its execution successfully is said to be committed. A committed transaction that has performed updates transforms the database into a new consistent state, which must persist even if there is a system failure.

4) compensating-

Once a transaction has committed, we cannot undo its effects by aborting it. The only way to undo the effects of a committed transaction is to execute a compensating transaction.

Abstract transaction model:-

A transaction must be in one of the following states::

I. Active

• The initial state

• The transaction stays in this state while it is executing.

II. Partially committed

- After the final statement of the transaction has been executed successfully.

• But the changes are not yet done permanently to the database.

III. <u>Failed</u>

- After the discovery that normal execution can no longer proceed, execution of statements in transaction stops.

IV. <u>Aborted</u>

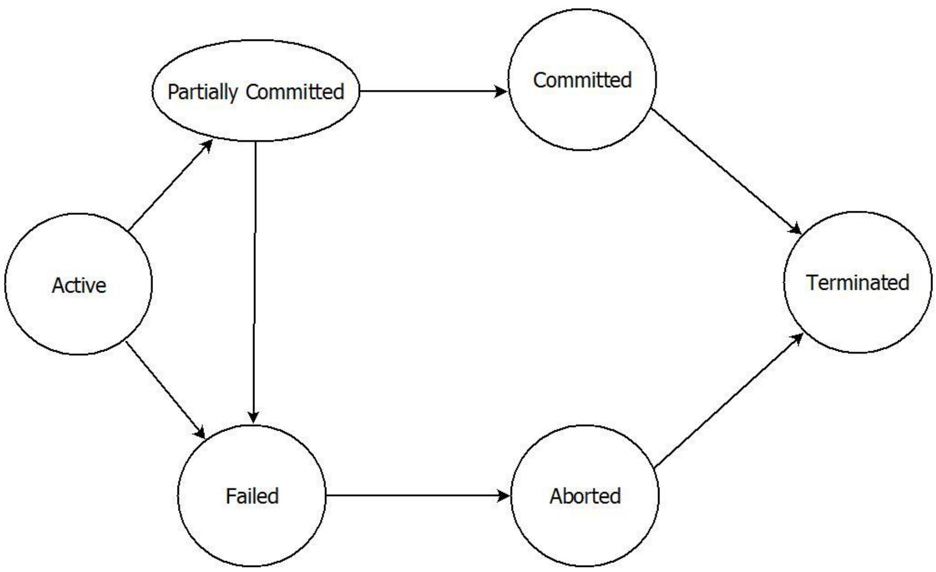
- after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction
- Two options after it has been aborted:
  1> restart the transaction –only if no internal logical error
  2> kill the transaction.

V. <u>Committed</u>

- After successful completion of transaction.
- Changes are also done to the database permanently.

VI. <u>Terminated</u>

- A transaction is said to be terminated, if it is either committed or aborted.



Q3. Give advantages and disadvantages of concurrent execution of transactions.

Ans–

- Advantages :

1. Increased processor and disk utilization leading to better transaction throughput: one transaction can be using the CPU while another is reading from or writing to the disk

2. Reduced average response time for transactions: short transactions need not wait behind long ones. Concurrent Execution of Transactions

3. Concurrency control schemes –

4. mechanisms to achieve isolation, i.e., to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

- Disadvantages :

1. Temporary Update Problem:
   Temporary update or dirty read problem occurs when one transaction updates an item and

fails. But the updated item is used by another transaction before the item is changed or reverted back to its last value.

2. Incorrect Summary Problem:

   Consider a situation, where one transaction is applying the aggregate function on some records while another transaction is updating these records. The aggregate function may calculate some values before the values have been updated and others after they are updated.

3. Lost Update Problem:

   In the lost update problem, update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction.

4. Unrepeatable Read Problem:

   The unrepeatable problem occurs when two or more read operations of the same transaction read different values of the same variable.

5. Phantom Read Problem:

   The phantom read problem occurs when a transaction reads a variable once but when it tries to read that same variable again, an error occurs saying that the variable does not exist.

Q4. Describe the following terms

a. Schedule

b. Serial Schedule

c. Equivalent Schedules

d. Serializable Schedule

e. Recoverable Schedule

Ans–

a. Schedule

• It is a sequence that indicate the chronological order in which instructions of concurrent transactions are executed

– a schedule for a set of transactions must consist of all instructions of those transactions

– must preserve the order in which the instructions appear in each individual transaction.

Example- Let T1 transfer $50 from A to B, and T2 transfer 10% of the balance from A to B. The following is a serial schedule in which T1 is followed by T2 .

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

schedule1

Let T1 and T2 be the transactions defined previously. The following schedule is not a serial schedule, but it is equivalent to Schedule 1. In both Schedule, the sum A + B is preserved.

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

Schedule 2

The following concurrent schedule does not preserve the value of the sum A + B.

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

b. Serial Schedule

c. Equivalent Schedules

Q5 DESCRIBE THE FOLLOWING TERMS

A. CONFLICT EQUIVALENT SCHEDULE

B. CONFLICT SERIALIZABLE SCHEDULE

C. VIEW EQUIVALENT SCHEDULE

D. VIEW SERIALIZABLE SCHEDULE

ANS–

a. Conflict equivalent schedule

b. Conflict Serializable Schedule

Instructions l i and l j of transactions Ti and Tj respectively, conflict if and only if there exists some item Q accessed by both l i and l j , and at least one of these instructions
write Q.

1. l i = read(Q), l j = read(Q). l i and l j don't conflict.
2. l i = read(Q), l j = write(Q). They conflict
3. l i = write(Q), l j = read(Q). They conflict
4. l i = write(Q), l j = write(Q). They conflict

Ii and Ij conflict, if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation. a conflict between l i and l j forces a (logical) temporal order between them. If l i and l j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.

If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, we say that S and S´ are conflict equivalent. We say that a schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

Example of a schedule that is not conflict serializable:

T3	T4
Read(Q)	
WRITE (Q)	WRITE(Q)

We are unable to swap instructions in the above schedule to obtain either the serial schedule < T3 , T4 >, or the serial schedule < T4 , T3 >

Schedule below can be transformed into Schedule 1, a serial schedule where T2 follows T1

,by series of swaps of non-conflicting instructions.
•Therefore Schedule 2 is conflict serializable.

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	
	read(B)
	write(B)

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

c. View Equivalent Schedule

d. View Serializable Schedule

Let S and S´ be two schedules with the same set of transactions. S and S´ are view equivalent if the following three conditions are met:

1. For each data item Q, if transaction Ti reads the initial value of Q in schedule S, then transaction Ti must, in schedule S´, also read the initial value of Q.

2. For each data item Q if transaction Ti executes read(Q) in schedule S, and that value was produced by transaction Tj (if any), then transaction Ti must in schedule S´ also read the value of Q that was produced by transaction Tj

3. For each data item Q, the transaction (if any) that performs the final write(Q) operation in schedule S

must perform the final write(Q) operation in scheduleS´.

Conditions 1 and 2 ensure that each transaction reads the same values in both schedules and,

• therefore, performs the same computation.

• Condition 3, coupled with conditions 1 and 2, ensures that both schedules result in the same final system state.

The concept of view equivalence leads to the concept of view serializability.

• A schedule S is view serializable, if it is view equivalent to a serial schedule.

Every conflict serializable schedule is also view serializable.

• Some schedules which are view-serializable but not conflict serializable.

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Some transactions perform write(Q) operations without having performed a read(Q) operation.

• Writes of this sort are called blind writes.

• Blind writes appear in any view-serializable schedule that is not conflict serializable.

Q7. Explain Conflict Serializability.

Ans–

Consider a schedule S.

• We construct a directed graph, called a precedence graph, from S.

• This graph consists of a pair G=(V, E), where V is a set of vertices and E is a set of edges.

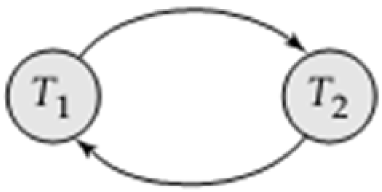
• The set of vertices consists of all the transactions participating in the schedule.

• The set of edges consists of all edges Ti →Tj for which one of three conditions holds:

1. Ti executes write(Q) before Tj executes read(Q).
2. Ti executes read(Q) before Tj executes write(Q).
3. Ti executes write(Q) before Tj executes write(Q).

(a) (b)

If an edge Ti →Tj exists in the precedence graph, then, in any serial schedule S' equivalent to S, Ti must appear before Tj



• The precedence graph appears in Figure.
• It contains the edge T1→T2, because T1 executes read(A) before T2 executes write(A).
• It also contains the edge T2→T1, because T2 executes read(B) before T1 executes write(B).

If the precedence graph for S has a cycle,
• then schedule S is not conflict serializable
• If the graph contains no cycles, then the schedule S is conflict serializable.
Thus, to test for conflict serializability, we need to construct the precedence graph and to invoke a cycle-detection algorithm .

Q8. Explain view serializability.

Ans–

Testing for view serializability is too complicated.

• it has been shown that the problem of testing for view serializability is itself NP-complete.
• Thus, almost certainly there exists no efficient algorithm to test for view serializability.

- However practical algorithms that just check some sufficient conditions for view serializability can still be used.

Q9. COMPARE RECOVERABLE SCHEDULE AND NON-RECOVERABLE SCHEDULE.

ANS–

Q 10. DESCRIBE THE FOLLOWING TERMS

A. CASCADING ROLLBACK

B. CASCADELESS SCHEDULES

ANS–

a. **Cascading Rollback:-**

a single transaction failure leads to a series of transaction rollbacks.

Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

If T10 fails, T11 and T12 must also be rolled back.
Can lead to the undoing of a significant amount of work

b. Cascadeless Schedules:-
cascading rollbacks cannot occur; for each pair of transactions Ti and Tj such that Tj reads a data item previously written by Ti , the commit operation of Ti appears before the read operation of Tj .
Every cascadeless schedule is also recoverable
It is desirable to restrict the schedules to those that are cascadeless.

Q11. GIVE MECHANISM FOR TESTING SERIALIZABILITY

ANS–

Consider a schedule S.

• We construct a directed graph, called a precedence graph, from S.

• This graph consists of a pair G=(V, E), where V is a set of vertices and E is a set of edges

Consider a schedule S.

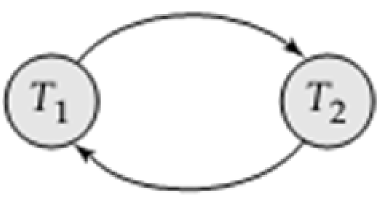
The set of vertices consists of all the transactions participating in the schedule.

• The set of edges consists of all edges Ti →Tj for which one of three conditions holds:

1. Ti executes write(Q) before Tj executes read(Q).
2. Ti executes read(Q) before Tj executes write(Q).
3. Ti executes write(Q) before Tj executes write(Q)

(a) (b)

If an edge Ti →Tj exists in the precedence graph, then, in any serial schedule S' equivalent to S, • Ti must appear before Tj



The precedence graph appears in Figure.

• It contains the edge T1→T2, because T1 executes read(A) before T2 executes write(A).

• It also contains the edge T2→T1, because T2 executes read(B) before T1 executes write(B)

If the precedence graph for S has a cycle, then schedule S is not conflict serializable.

• If the graph contains no cycles, then the schedule S is conflict serializable.

A serializability order of the transactions can be obtained through topological sorting, which determines a linear order consistent with the partial order of the precedence graph.

There are several possible linear orders that can be obtained through a topological sorting.

For example, the graph of Figure a has two acceptable linear orderings shown in Figures b and c.

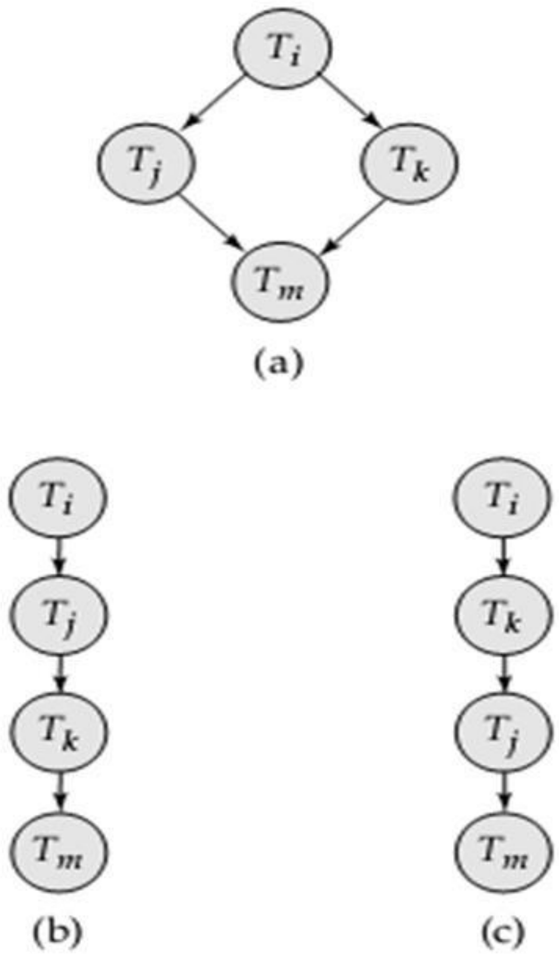


Figure Illustration of topological sorting.

Thus, to test for conflict serializability, we need to construct the precedence graph and to invoke a cycle-detection algorithm.

Q12. WHAT IS PRECEDENCE GRAPH? GIVE THE USES OF PRECEDENCE GRAPH.

ANS–

Q13. Explain Lock-Based Protocols for Concurrency Control.

Ans–

One way to ensure serializability is to require that

• data items be accessed in a mutually exclusive manner;
• while one transaction is accessing a data item,

• no other transaction can modify that data item.

The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item.

A lock is a mechanism to control concurrent access to a data item.

Data items can be locked in two modes :

• **Exclusive (X) Mode.**

Data item can be both read as well as written. X-lock is requested using lock-X instruction.

• **Shared (S) Mode.**

Data item can only be read. S-lock is requested using lock-S instruction.

Lock requests are made to concurrency-control manager.

Transaction can proceed only after request is granted.

Q20. Explain lock conversion and automatic acquisition of locks.

ANS–

A)Lock Conversions

• Two-phase locking with lock conversions:

– Growing Phase / First Phase:

– can acquire a lock-S on item

– can acquire a lock-X on item

– can convert a lock-S to a lock-X (upgrade)

– Shrinking Phase / Second Phase:

– can release a lock-S

– can release a lock-X

– can convert a lock-X to a lock-S (downgrade)

This protocol assures serializability.

B)Automatic Acquisition of Locks

A transaction Ti issues the standard read/write instruction, without explicit locking calls.

- The operation read(D) is processed as:

if Ti has a lock on D

then

read(D)

else

begin

if necessary wait until no other transaction has a lock-X on D grant Ti a lock-S on D;

read(D)

end

write(D) is processed as:

if Ti has a lock-X on D
then
write(D)
else
begin

if necessary wait until no other trans. has any lock on D,
if Ti has a lock-S on D
then
upgrade lock on D to lock-X
else
grant Ti a lock-X on D
write(D)
end;
All locks are released after commit or abort.

Q21. HOW LOCKING CAN BE IMPLEMENTED?

ANS–

• A Lock manager can be implemented as a separate process to which transactions send lock and unlock requests

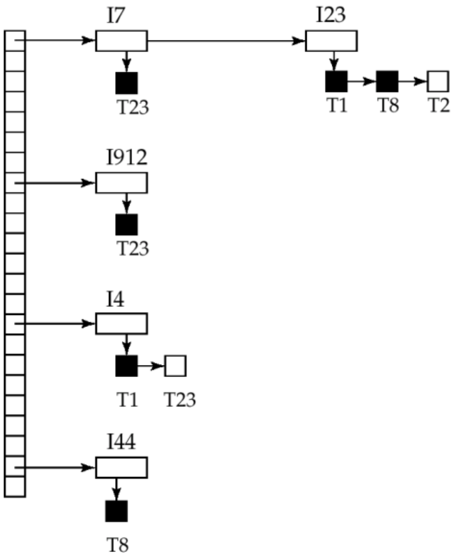
• The lock manager replies to a lock request by sending a lock grant messages (or a message asking the transaction to roll back, in case of a deadlock)

The requesting transaction waits until its request is answered

The lock manager maintains a data structure called a lock table to record granted locks and pending requests.

The lock table is usually implemented as an inmemory hash table indexed on the name of the data item being locked.

Lock Table-



• Black rectangles indicate granted locks, white ones indicate waiting requests

• Lock table also records the type of lock granted or requested

• New request is added to the end of the queue of requests for the data item, and granted if it is compatible with all earlier locks

• Unlock requests result in the request being deleted, and later requests are checked to see if they can now be granted

• If transaction aborts, all waiting or granted requests of the transaction are deleted – lock manager may keep a list of locks held by each transaction, to implement this efficiently

Q22. Explain graph-based protocols for concurrency control.

Ans–

• Two-phase locking protocol is both necessary and sufficient for ensuring serializability in the absence of information concerning the manner in which data items are accessed. If we wish to develop protocols that are not two phase, we need additional information on how each transaction will access the database. Requires that we have prior knowledge about the order in which the database items will be accessed.

Graph-based protocols are an alternative to twophase locking

• Impose a partial ordering → on the set

• D = {d1 , d2 ,..., dh } of all data items. – If di → dj then any transaction accessing both di and dj must access di before accessing dj . – Implies that the set D may now be viewed as a directed acyclic graph, called a database graph.

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

Q23. Explain Timestamp-Based Protocols for Concurrency Control.

ANS–

• 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.

There are two simple methods for implementing this scheme:

• **1. Use the value of the system clock as the timestamp;**

• **2. Use a logical counter that is incremented after a new timestamp has been assigned**

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

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

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

The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.

• Suppose a transaction Ti issues a read(Q)

• 1. If TS(Ti) ≤ W-timestamp(Q), . then Ti needs to read a value of Q that was already overwritten

• Hence, the read operation is rejected, and Ti is rolled back.

• 2. If TS(Ti)≥ W-timestamp(Q), then the read operation is executed, and R-timestamp(Q) is set to the maximum of Rtimestamp(Q) and TS(Ti).

Suppose that transaction Ti issues write(Q).

• If TS(Ti) < R-timestamp(Q), then the value of Q that Ti is producing was needed previously, and the system assumed that that value would never be produced. Hence, the write operation is rejected, and Ti is rolled back.

• If TS(Ti) < W-timestamp(Q), then Ti is attempting to write an obsolete value of Q. Hence, this write operation is rejected, and Ti is rolled back. • Otherwise, the write operation is executed, and Wtimestamp(Q) is set to TS(Ti).

Q24. How Read operation is performed using Timestamp-Based Protocols for concurrency control.

Ans–

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

• The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.

• Suppose a transaction Ti issues a read(Q)

• 1. If TS(Ti) ≤ W-timestamp(Q), . then Ti needs to read a value of Q that was already overwritten

• Hence, the read operation is rejected, and Ti is rolled back.

• 2. If TS(Ti)≥ W-timestamp(Q), then the read operation is executed, and R-timestamp(Q) is set to the maximum of Rtimestamp(Q) and TS(Ti)

Q25. How Write operation is performed using Timestamp-Based Protocols for concurrency control.

Ans–

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

The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.

• Suppose that transaction Ti issues write(Q).

• If TS(Ti) < R-timestamp(Q), then the value of Q that Ti is producing was needed previously, and the system assumed that that value would never be produced.

• Hence, the write operation is rejected, and Ti is rolled back.

• If TS(Ti) < W-timestamp(Q), then Ti is attempting to write an obsolete value of Q.

• Hence, this write operation is rejected, and Ti is rolled back.

• Otherwise, the write operation is executed, and Wtimestamp(Q) is set to TS(Ti).

Q26. Explain Thomas' Write Rule.

Ans–

• The modification to the timestamp-ordering protocol, called Thomas' Write Rule, is this:

• Suppose that transaction Ti issues write(Q).

1.if TS(Ti)<R-timestamp(Q) then the value of Q that Ti is producing was previously needed, and it had been assumed that the value would never be produced. Hence, the system rejects the write operation and rolls Ti back.

2.If TS(Ti)<W-timestamp(Q) Then Ti is attempting to write an obsolete value Of Q.

• Hence, this write operation can be ignored.

3. Otherwise, the system executes the write operation and sets Wtimestamp(Q) to TS(Ti)

Q27. Compare timestamp-based protocols with Thomas' write rule.

ANS---

• The difference between these rules and those of timestamp-ordering protocol lies in the second rule.

• The timestamp-ordering protocol requires that Ti be rolled back if Ti issues write(Q) and TS(Ti)<W-timestamp(Q).

• Timestamp of Current transaction(Ti) is greater than the transaction recently read that data Q

• Thomas' write rule makes use of view serializability by, in effect, deleting obsolete write operations from the transactions that issue them.

• Thomas' Write Rule allows greater potential concurrency.

Q28. Explain Validation-Based Protocol for Concurrency Control. Explain Optimistic Concurrency Control Protocol.

Ans—

• In cases where a majority of transactions are readonly transactions, the rate of conflicts among transactions may be low.

• Thus, many of these transactions, if executed without the supervision of a concurrency-control scheme, would not leave the system in in-consistent state.

• A concurrency-control scheme imposes overhead of code execution and possible delay of transactions.

• It may be better to use an alternative scheme that imposes less overhead.

• A difficulty in reducing the overhead is that we do not know in advance which transactions will be involved in a conflict.

• To gain that knowledge, we need a scheme for monitoring the system.

• We assume that each transaction Ti executes in three different phases in its lifetime, depending on whether it is a read-only or an update transaction. The phases are, in order

1. **Read phase.** –
    - During this phase, the system executes transaction Ti.
    - It reads the values of the various data items and stores them in variables local to Ti.
2. **Validation phase** –
    - Transaction Ti performs a validation test to determine
    - whether it can copy to the database the temporary local variables
    - that hold the results of write operations without causing a violation of serializability.

3. **Write phase.** –
    - If transaction Ti succeeds in validation (step 2),
    - then the system applies the actual updates to the database.
    - Otherwise, the system rolls back Ti.

    - The three phases of concurrently executing transactions can be interleaved, but each transaction must go through the three phases in that order.

• Also called as optimistic concurrency control
• since transaction executes fully in the hope that all will go well during validation.

Q29. What is granularity? What are the different types of granularity?

Ans:

Q30. Explain multi-version timestamp ordering protocol.

Ans–

• The most common transaction ordering technique used by multiversion schemes is timestamping.

• With each transaction Ti in the system, we associate a unique static timestamp, denoted by TS(Ti).

• The database system assigns this timestamp before the transaction starts execution.

• Each data item Q has a sequence of versions . Each version Qk contains three data fields:

– Content -- the value of version Qk .

– W-timestamp(Qk) -- timestamp of the transaction that created (wrote) version Qk

– R-timestamp(Qk) -- largest timestamp of a transaction that successfully read version Qk

• when a transaction Ti creates a new version Qk of Q, Qk 's W-timestamp and R-timestamp are initialized to TS(Ti).

• R-timestamp of Qk is updated whenever a transaction Tj reads Qk , and

• TS(Tj) > R-timestamp(Qk).

• The multiversion timestamp scheme presented ensures serializability.

• Suppose that transaction Ti issues a read(Q) or write(Q) operation.

• Let Qk denote the version of Q whose write timestamp is the largest write timestamp less than or equal to TS(Ti).

1. If transaction Ti issues a read(Q), then the value returned is the content of version Qk .
2. If transaction Ti issues a write(Q), and

• if TS(Ti) < R-timestamp(Qk),

• then transaction Ti is rolled back.

• Otherwise, if TS(Ti) = W-timestamp(Qk), the contents of Qk are overwritten,

• otherwise a new version of Q is created.

• Reads always succeed;

• a write by Ti is rejected if some other transaction Tj that (in the serialization order defined by the timestamp values) should read Ti 's write, has already read a version created by a transaction older than Ti

Q31. Explain Multi-version two-phase locking protocol.

Ans–

• Differentiates between read-only transactions and update transactions

• Read-only transactions are assigned a timestamp by reading the current value of ts-counter before they start execution;

• they follow the multiversion timestamp-ordering protocol for performing reads.

• Update transactions acquire read and write locks and hold all locks up to the end of the transaction.

• That is, update transactions follow rigorous twophase locking. – Each successful write results in the creation of a new version of the data item written. – each version of a data item has a single timestamp whose value is obtained from a counter ts-counter that is incremented during commit processing.

• When an update transaction wants to read a data item, it obtains a shared lock on it, and reads the latest version.

• When it wants to write an item, it obtains X lock on; it then creates a new version of the item and sets this version's timestamp to ∞.

• When update transaction Ti completes, commit processing occurs: – Ti sets timestamp on the versions it has created to tscounter + 1 – Ti increments ts-counter by1

• Read-only transactions that start after Ti increments ts-counter will see the values updated by Ti .

• Read-only transactions that start before Ti increments the ts-counter will see the value before the updates by Ti .

• Only serializable schedules are produced.