Unit 4
Query Processing And Transaction Management

-SUSHMA VANKHEDE

Introduction to Transaction in DBMS

A **transaction** is a set of logically related operations. For example, you are transferring money from your bank account to your friend’s account, the set of operations would be like this:

Simple Transaction Example

1. Read your account balance

2. Deduct the amount from your balance

3. Write the remaining balance to your account

4. Read your friend’s account balance

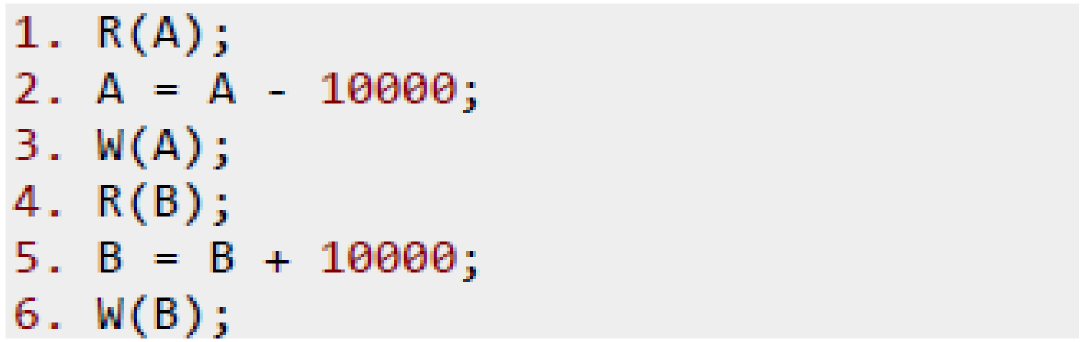
5. Add the amount to his account balance

6. Write the new updated balance to his account

Example

In DBMS, we write the above 6 steps transaction like this:

If your account is A and your friend’s account is B, you are transferring 10000 from A to B, the steps of the transaction are



```
1. R(A);
2. A = A - 10000;
3. W(A);
4. R(B);
5. B = B + 10000;
6. W(B);
```

What are the problems associated with transaction?

Transaction failure in between the operations due to power failure, system crash etc

This is a serious problem that can leave database in an inconsistent state.

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.

But they are not sufficient in concurrent execution. To handle those problems we need to understand database ACID properties.

ACID Properties

Atomicity: This property ensures that either all the operations of a transaction reflect in database or none.

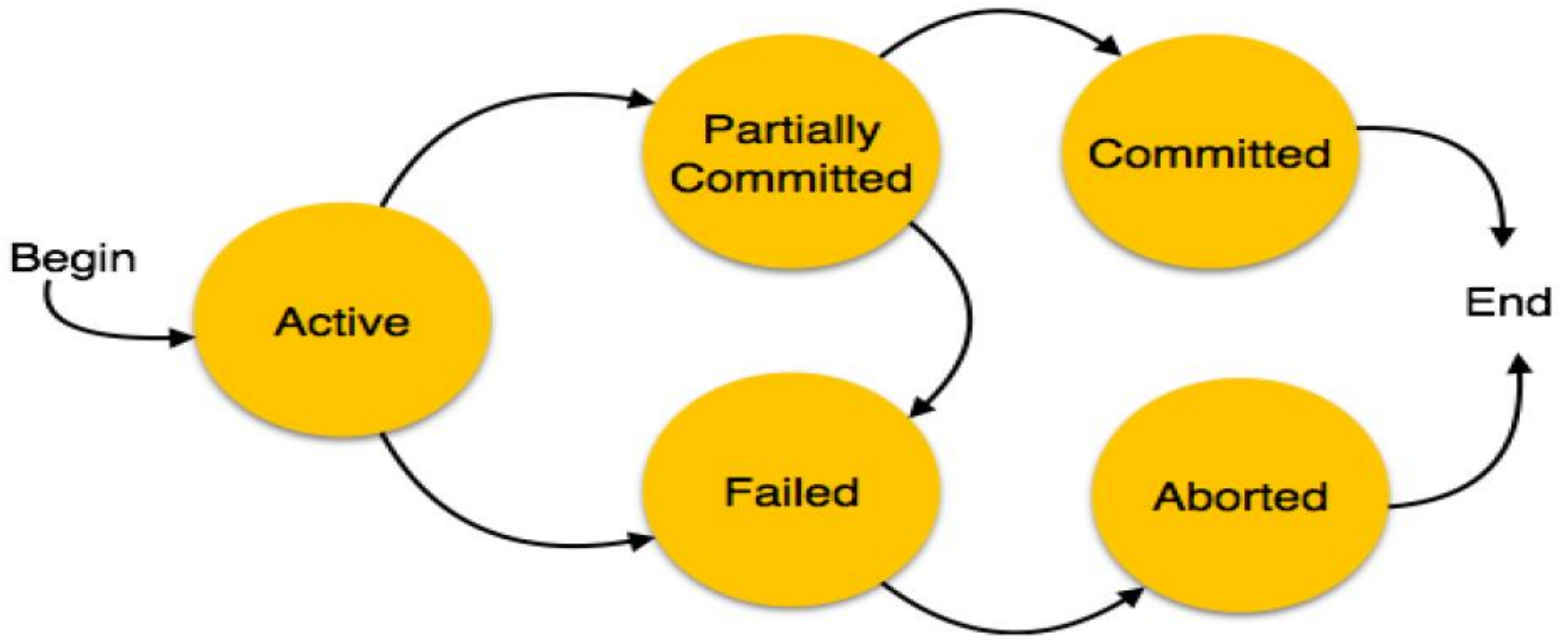
Consistency: To preserve the consistency of database, the execution of transaction should take place in isolation (that means no other transaction should run concurrently when there is a transaction already running).

ACID Properties…cont

Isolation: For every pair of transactions, one transaction should start execution only when the other finished execution.

Durability: Once a transaction completes successfully, the changes it has made into the database should be permanent even if there is a system failure. The recovery-management component of database systems ensures the durability of transaction.

Transaction States



Transaction States …Cont

Active: The initial state when the transaction has just started execution.

Partially Committed: At any given point of time if the transaction is executing properly, then it is going towards it COMMIT POINT. The values generated during the execution are all stored in volatile storage.

Failed: If the transaction fails for some reason. The temporary values are no longer required, and the transaction is set to ROLLBACK. It means that any change made to the database by this transaction up to the point of the failure must be undone. If the failed transaction has withdrawn Rs. 100/- from account A, then the ROLLBACK operation should add Rs 100/- to account A.

Transaction States …Cont

Aborted: When the ROLLBACK operation is over, the database reaches the BFIM. The transaction is now said to have been aborted.

Committed: If no failure occurs then the transaction reaches the COMMIT POINT. All the temporary values are written to the stable storage and the transaction is said to have been committed.

Terminated: Either committed or aborted, the transaction finally reaches this state.

Concurrent Execution

Concurrent execution means running side by side or parallel of transactions.

Advantages of Concurrent execution are:

Improved throughput & Resource utilization – i.e. no. of transactions executed increases in a given amount of time & the processor is utilized properly.

Reduced Waiting time – The unpredictable delays in running transactions as well as the average response time is reduced.

Schedule

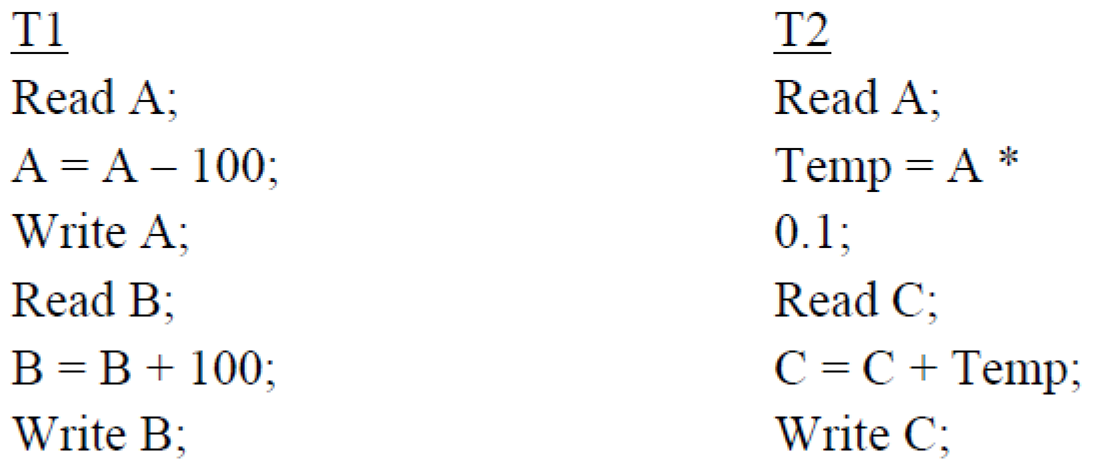
A *schedule* is a collection of many transactions which is implemented as a unit. Depending upon how these transactions are arranged in within a schedule, a schedule can be of two types:

- Serial: The transactions are executed one after another, in a non-preemptive manner.
- Concurrent: The transactions are executed in a preemptive, time shared method.

Schedule..cont

T1 is the transaction in which we have two accounts A and B, each containing Rs 1000/-. We now start a transaction to deposit Rs 100/- from account A to Account B.

T2 is a transaction which deposits to account C 10% of the amount in account A.



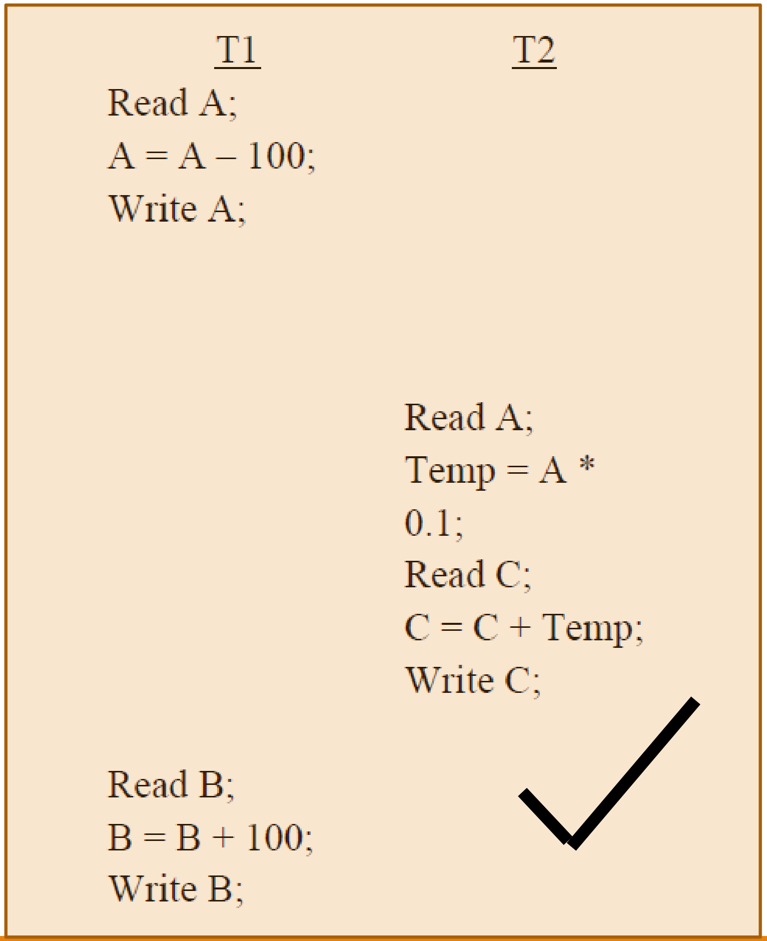
T1	T2
Read A;	Read A;
A = A – 100;	Temp = A *
Write A;	0.1;
Read B;	Read C;
B = B + 100;	C = C + Temp;
Write B;	Write C;

Schedule..cont

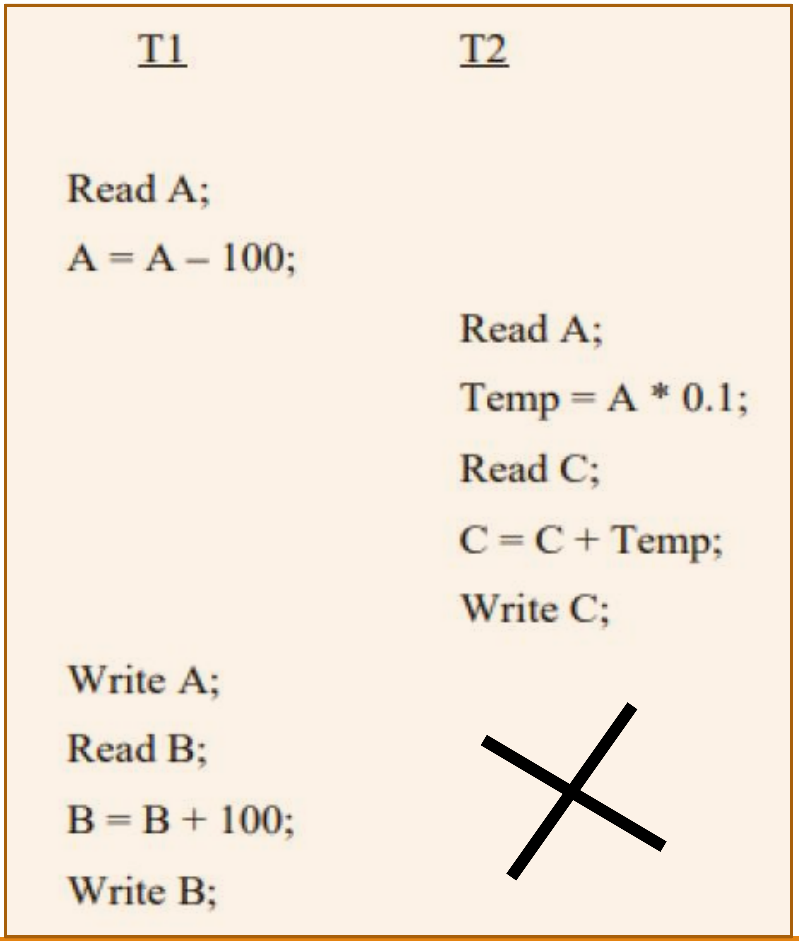
If we prepare **a serial schedule**, then either T1 will completely finish before T2 can begin, or T2 will completely finish before T1 can begin.

However, if we want to create **a concurrent schedule**, then some Context Switching need to be made, so that some portion of T1 will be executed, then some portion of T2 will be executed and so on.

Concurrent Schedule



T1	T2
Read A;	
A = A – 100;	
Write A;	
	Read A;
	Temp = A * 0.1;
	Read C;
	C = C + Temp;
	Write C;
Read B;	✓
B = B + 100;	
Write B;	



T1	T2
Read A;	
A = A – 100;	
	Read A;
	Temp = A * 0.1;
	Read C;
	C = C + Temp;
	Write C;
Write A;	
Read B;	✗
B = B + 100;	
Write B;	

Serializability

To create error free concurrent schedules we must follow some well formed rules to arrange instructions of the transactions.

When several concurrent transactions are trying to access the same data item, the instructions within these concurrent transactions must be ordered in some way so as there are no problem in accessing and releasing the shared data item.

Serializability…cont

There are two aspects of serializability:

1. **Conflict Serializability**
2. **View Serializability**

Conflict Serializability

Two instructions of two different transactions may want to access the same data item in order to perform a read/write operation.

Conflict Serializability deals with detecting whether the instructions are conflicting in any way, and specifying the order in which these two instructions will be executed in case there is any conflict.

A conflict arises if at least one (or both) of the instructions is a write operation. The following rules are important in Conflict Serializability:

1. If two instructions of the two concurrent transactions are both for read operation, then they are not in conflict, and can be allowed to take place in any order.

Conflict Serializability

2. If one of the instructions wants to perform a read operation and the other instruction wants to perform a write operation, then they are in conflict, hence their ordering is important. If the read instruction is performed first, then it reads the old value of the data item and after the reading is over, the new value of the data item is written. It the write instruction is performed first, then updates the data item with the new value and the read instruction reads the newly updated value.

3. 3. If both the transactions are for write operation, then they are in conflict but can be allowed to take place in any order, because the transaction do not read the value updated by each other. However, the value that persists in the data item after the schedule is over is the one written by the instruction that performed the last write.

View Serializability

This is another type of serializability that can be derived by creating another schedule out of an existing schedule, involving the same set of transactions. These two schedules would be called View Serializable if the following rules are followed while creating the second schedule out of the first.

Let us consider that the transactions T1 and T2 are being serialized to create two different schedules S1 and S2 which we want to be View Equivalent and both T1 and T2 wants to access the same data item.

1. If in S1, T1 reads the initial value of the data item, then in S2 also, T1 should read the initial value of that same data item.

2. If in S1, T1 writes a value in the data item which is read by T2, then in S2 also, T1 should write the value in the data item before T2 reads it.

3. If in S1, T1 performs the final write operation on that data item, then in S2 also, T1 should perform the final write operation on that data item. Except in these three cases, any alteration can be possible while creating S2 by modifying S1.

Concurrency-control Schemes

Concurrency-control schemes are also used to ensure serializability. All these schemes either delay an operation or abort the transaction that issued the operation.

Most commonly used Concurrency-control schemes are:

- locking protocols
- timestamp based protocols

Lock based protocols

A locking protocol is a set of rules that state when a transaction may lock and unlock each of the data items in the database.

- ***Two-phase locking protocol****:* this protocol allows a transaction to lock a new data item only if that transaction has not yet unlocked any data item. This protocol ensures serializability, but not deadlock freedom.
- ***Strict two-phase locking protocol:*** It permits release of exclusive locks only at the end of transactions, in order to ensure recoverability and cascadelessness of the resulting schedules.
- ***Rigorous two-phase locking protocol:*** This protocol releases all locks only at the end of the transaction.

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.

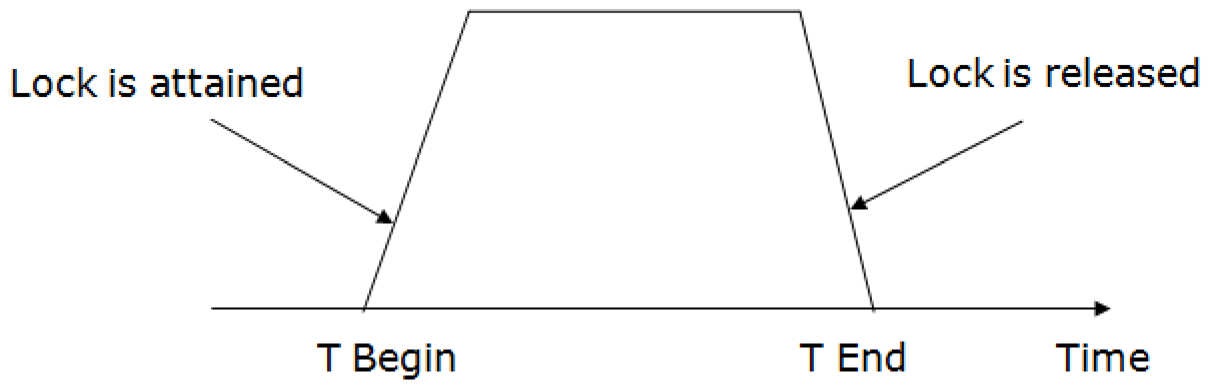
Two-phase locking (2PL)

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.



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:

Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.

Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

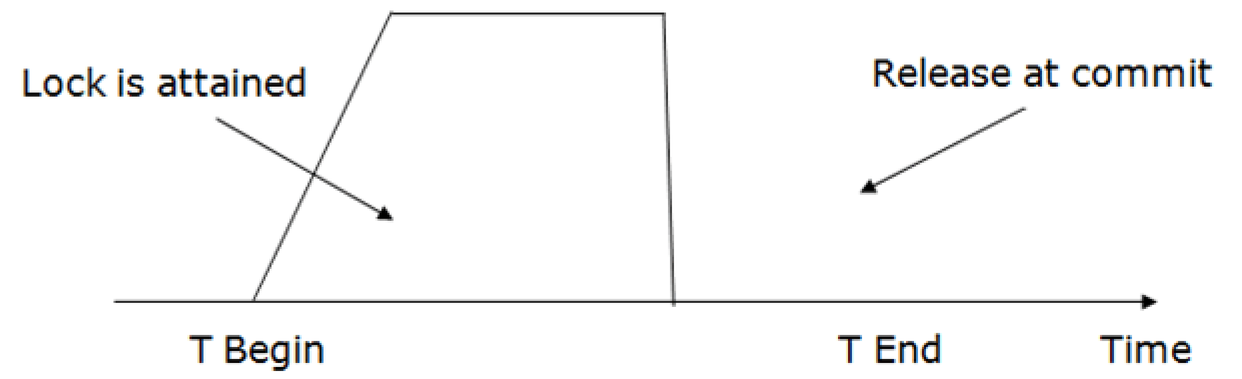
Strict Two-phase locking (Strict-2PL)

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.

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.

Working of Timestamp ordering protocol

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

If W_TS(X) >TS(Ti) then the operation is rejected.

If W_TS(X) <= TS(Ti) then the operation is executed.

Timestamps of all the data items are updated.

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

If TS(Ti) < R_TS(X) then the operation is rejected.

If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where,

TS(TI) denotes the timestamp of the transaction Ti.

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

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

Precedence graph for TS ordering

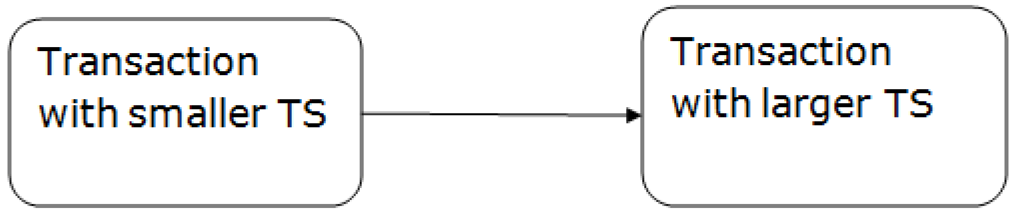


Image: Precedence Graph for TS ordering

Advantages and Disadvantages of TO protocol

TO protocol ensures serializability as per the precedence graph

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.

Intent Locks

The **intent lock** occurs when SQL Server wants to acquire the shared (S) **lock** or exclusive (X) **lock** on some of the resources lower in the **lock** hierarchy. In practice, when SQL Server acquires a **lock** on a page or row, the **intent lock** is required in the table.

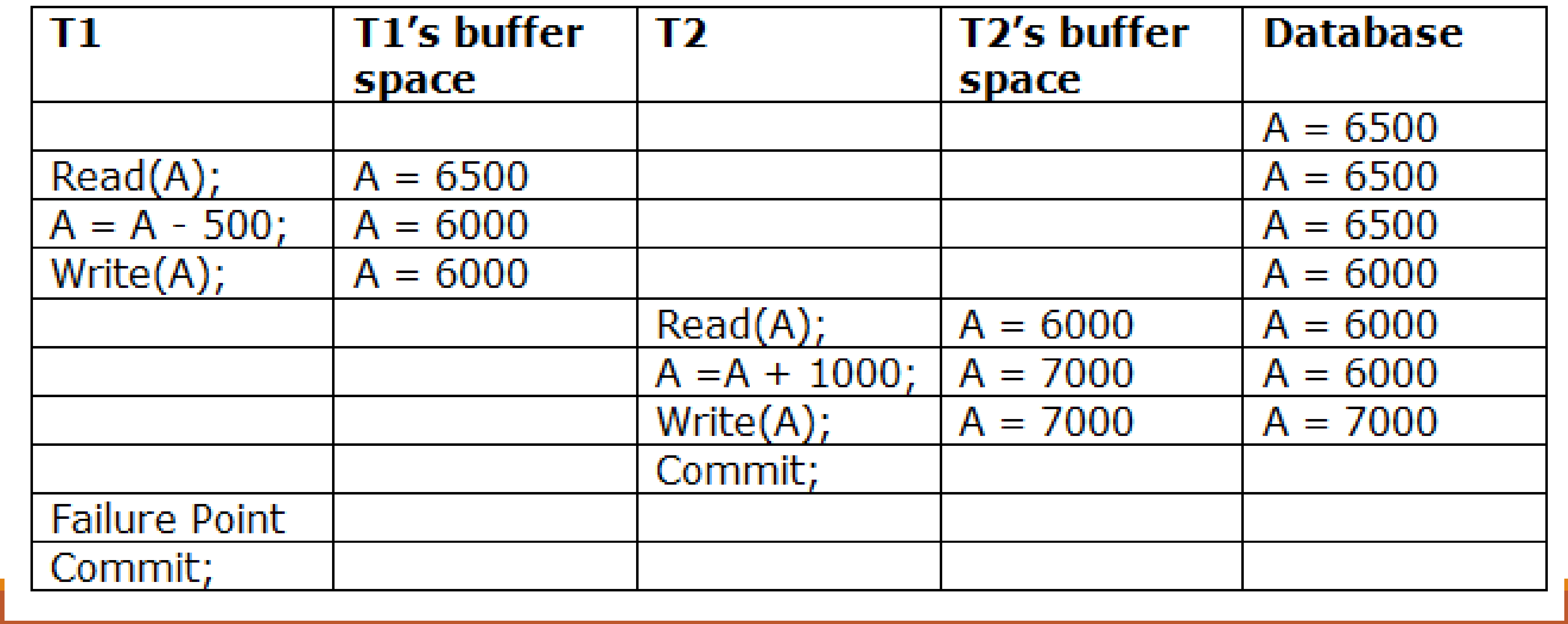
Recoverability of Schedule

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.

Transaction must be committed in order

Irrecoverable Schedule

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

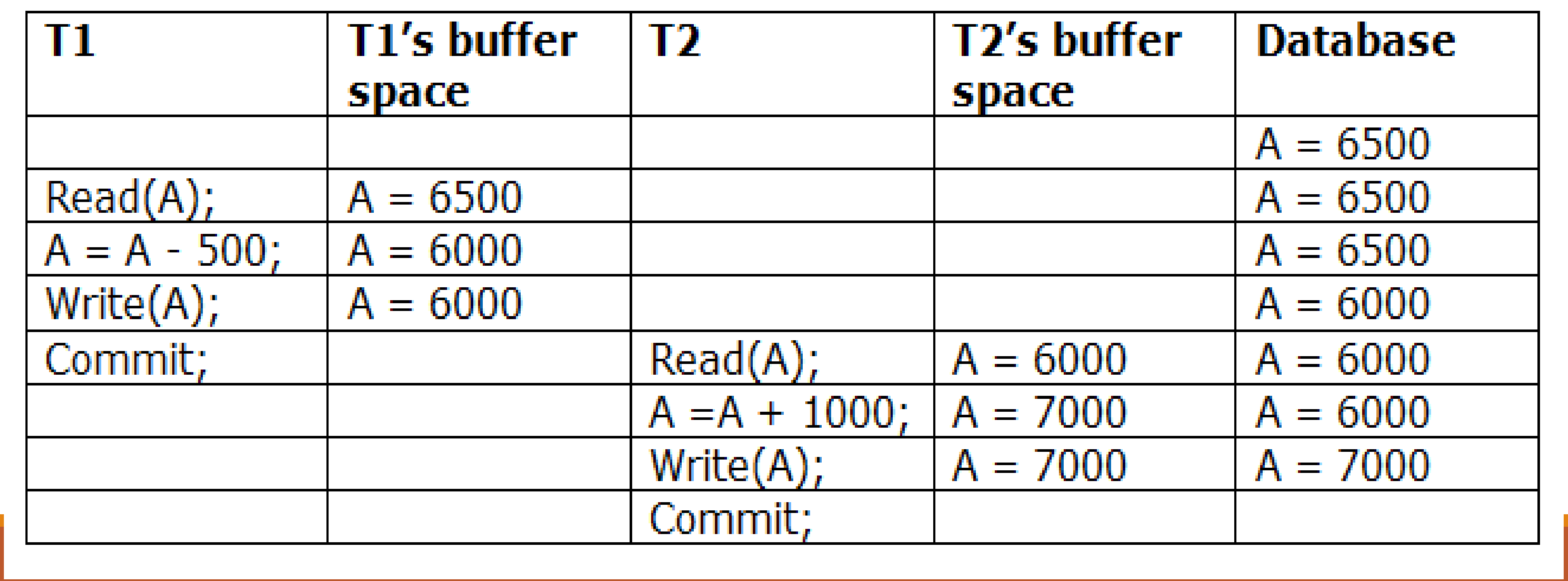


T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

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.

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.



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;		

Log-Based Recovery

The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.

If any operation is performed on the database, then it will be recorded in the log.

But the process of storing the logs should be done before the actual transaction is applied in the database.

Log-Based Recovery

Example: Transaction to modify the City of a student. The following logs are written for this transaction.

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

<Tn, Start>

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

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

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

<Tn, Commit>

Log-Based Recovery

Two approaches to modify the database:

1. Deferred database modification:

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

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

2. Immediate database modification:

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

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

Log-Based Recovery

Recovery using Log records

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

If the log contains the record <Ti, Start> and <Ti, Commit> or <Ti, Commit>, then the Transaction Ti needs to be redone.

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

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.

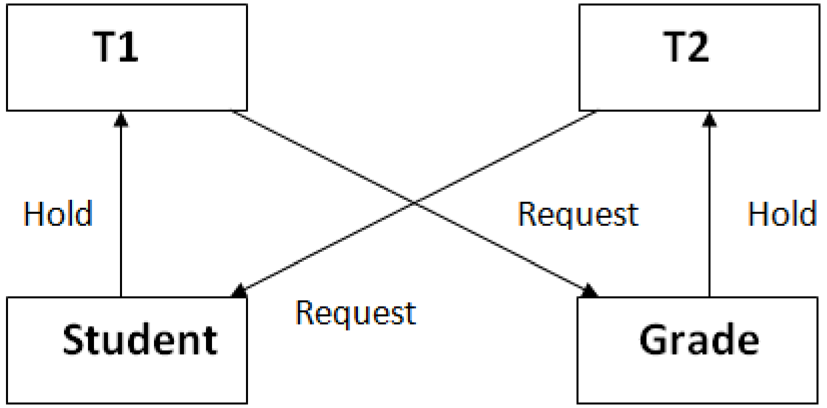


Figure: Deadlock in DBMS

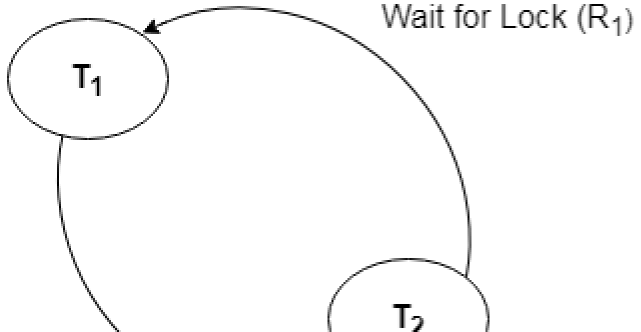
Deadlock Avoidance

When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resource.

Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "wait for graph" is used for detecting the deadlock situation but this method is suitable only for the smaller database. For the larger database, deadlock prevention method can be used.

Deadlock Detection

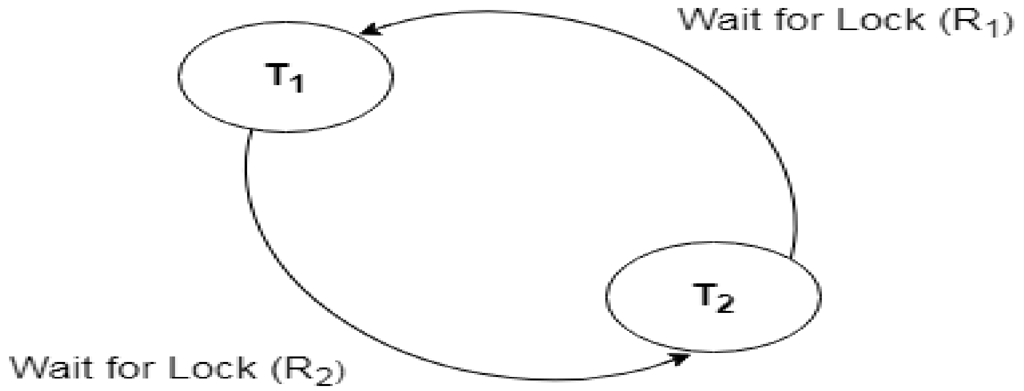
In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.



Wait for Graph

This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.

The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.



Deadlock Prevention

Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.

The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.

Wait-Die scheme

In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.

Let's assume there are two transactions Ti and Tj and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS:

Check if TS(Ti) < TS(Tj) - If Ti is the older transaction and Tj has held some resource, then Ti is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.

Check if TS(T_i) < TS(Tj) - If Ti is older transaction and has held some resource and if Tj is waiting for it, then Tj is killed and restarted later with the random delay but with the same timestamp.

Wound wait scheme

In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After the minute delay, the younger transaction is restarted but with the same timestamp.

If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.

END

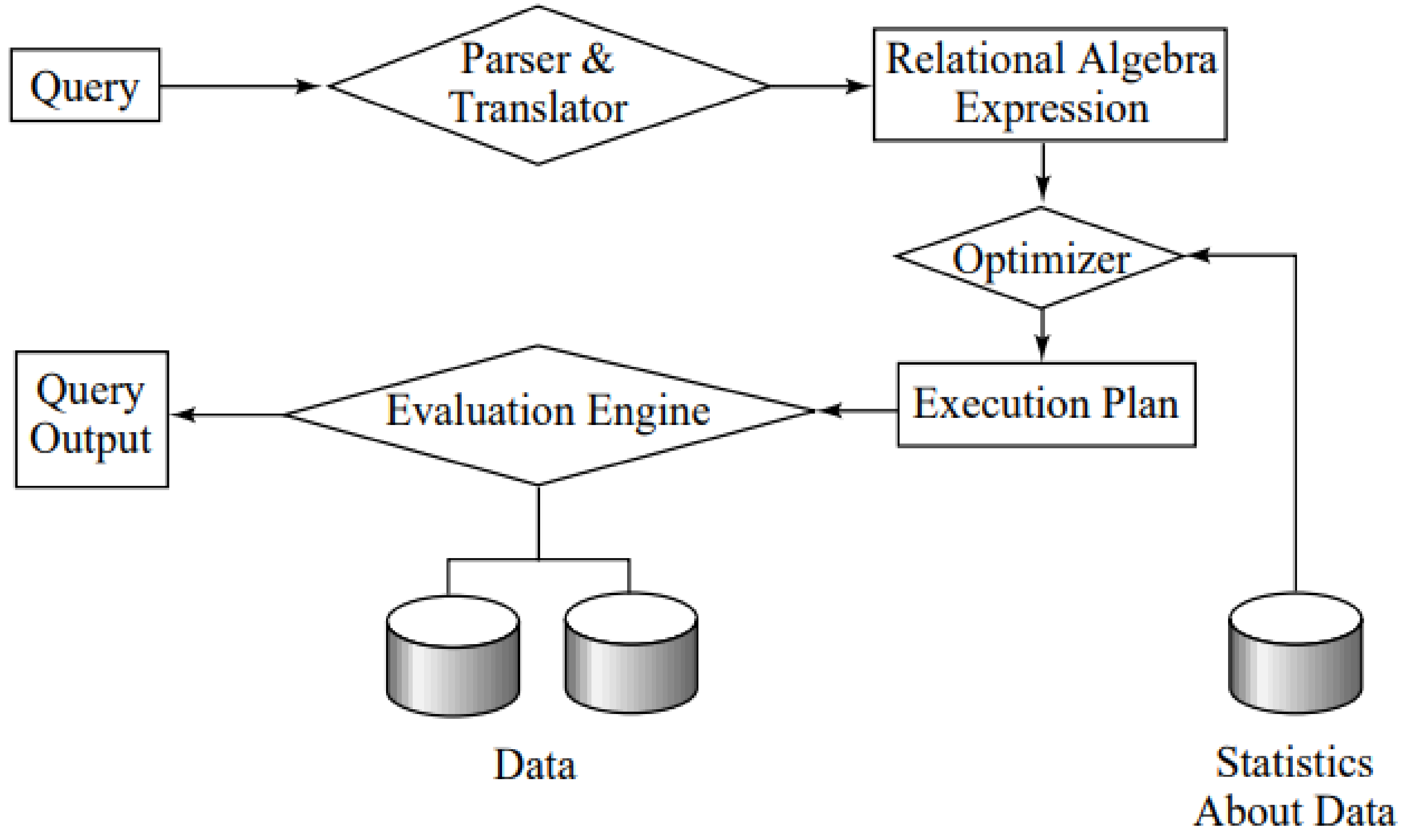
Query Processing Optimization

Query Processing Optimization

Query Processing includes translations on high level Queries into low level expressions that can be used at physical level of file system, query optimization and actual execution of query to get the actual result.

Basic Steps in Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Basic Steps in Query Processing

Parsing and translation

Translate the query into its internal form. This is then translated into relational algebra.

Parser checks syntax, verifies relations

Evaluation

The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.

Optimization

Finding the cheapest evaluation plan for a query.

END