Unit 6

Transactions Management and Concurrency and Recovery

Transaction concept

- A transaction is a logical unit of work of database processing that includes one or more database access operations. A transaction can be defined as an action or series of actions that is carried out by a single user or application program to perform operations for accessing the contents of the database.
- The operations can include retrieval, (Read), insertion (Write), deletion and modification.
- A transaction must be either completed or aborted.

A transaction can include the following basic database access operations:

Operations	**Descriptions**
Retrive	To retrive data stored ina database.
Insert	To store new data in database.
Delete	To delete existing data from database.
Update	To modify existing data in database.
Commit	To save the work done permanently.
Rollback	To undo the work done.

- **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**

```
Open_Account(X)
Old_Balance = X.balance
New_Balance = Old_Balance - 800
X.balance = New_Balance
Close_Account(X)
```

- **Y's Account**

```
Open_Account(Y)
Old_Balance = Y.balance
New_Balance = Old_Balance + 800
Y.balance = New_Balance
Close_Account(Y)
```

• 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.

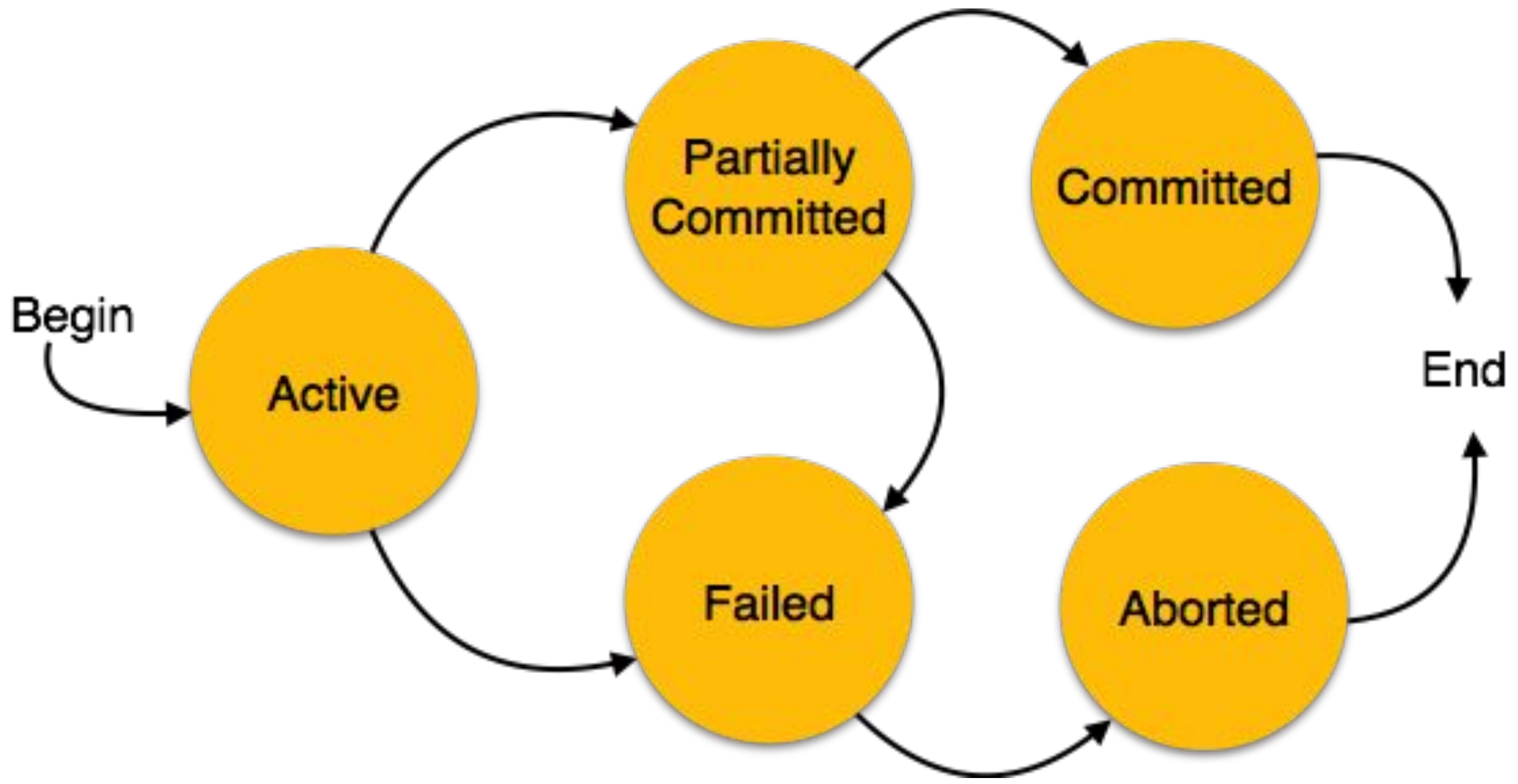
example to debit transaction from an account which consists of following operations:

```
R(X);
X = X - 500;
 W(X);
```

- 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 states



1. 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.

2. 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.

3. 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.

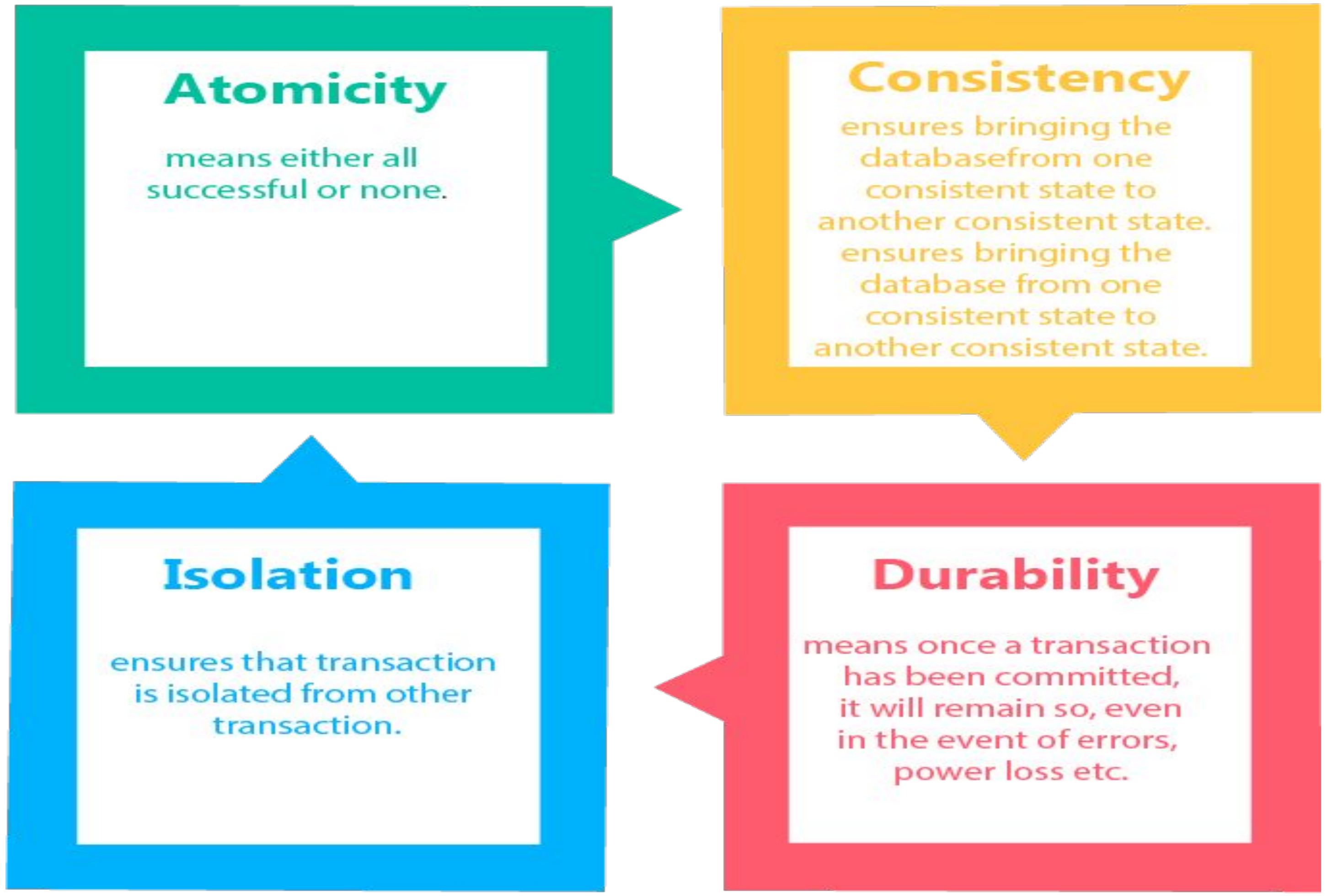
4. Failed state

- 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.

5.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

ACID properties


Atomicity
means either all successful or none.
Consistency
ensures bringing the databasefrom one consistent state to another consistent state. ensures bringing the database from one consistent state to another consistent state.
Isolation
ensures that transaction is isolated from other transaction.
Durability
means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

1.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.

2.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

3.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.

4.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.

Transaction Control Commands

- **Needs of Transaction Control Language**
- The Transaction Control Language manages the changes in the data which are made by the DML operations.
- The alteration made by the DML commands such as UPDATE, INSERT or DELETE is not permanent and these changes can be canceled before the present session gets closed.
- To control the changes and processing of data, TCL is used simultaneously with the Data Manipulation Language.
- As we perform many operations on the data, the database might become inconsistent between the transactions.
- So the Transaction Control Language (TCL) is used to maintain consistency and manage the transaction in a database.
- After the commit is performed the database state is changed from one to another consistent state.
- The Transactions are used on all the DDL and DML queries automatically.

TCL Commands

- **1. COMMIT**
- This command is used to make a transaction permanent in a database. So it can be said that commit command saves the work done as it ends the current transaction by making permanent changes during the transaction.
- syntax :

```
COMMIT;
```

- For instance, we want to update the location of an employee in the table “EMPLOYEE”.
- Sample EMPLOYEE table is given below:

EMP_ID	EMP_NAME	EMP_LOC
1356	Raju	Delhi
2678	Neeta	Bangalore
9899	Sanjay	Hyderabad

- Let us update the EMP_ LOC for Raju as below:

- **Query:**

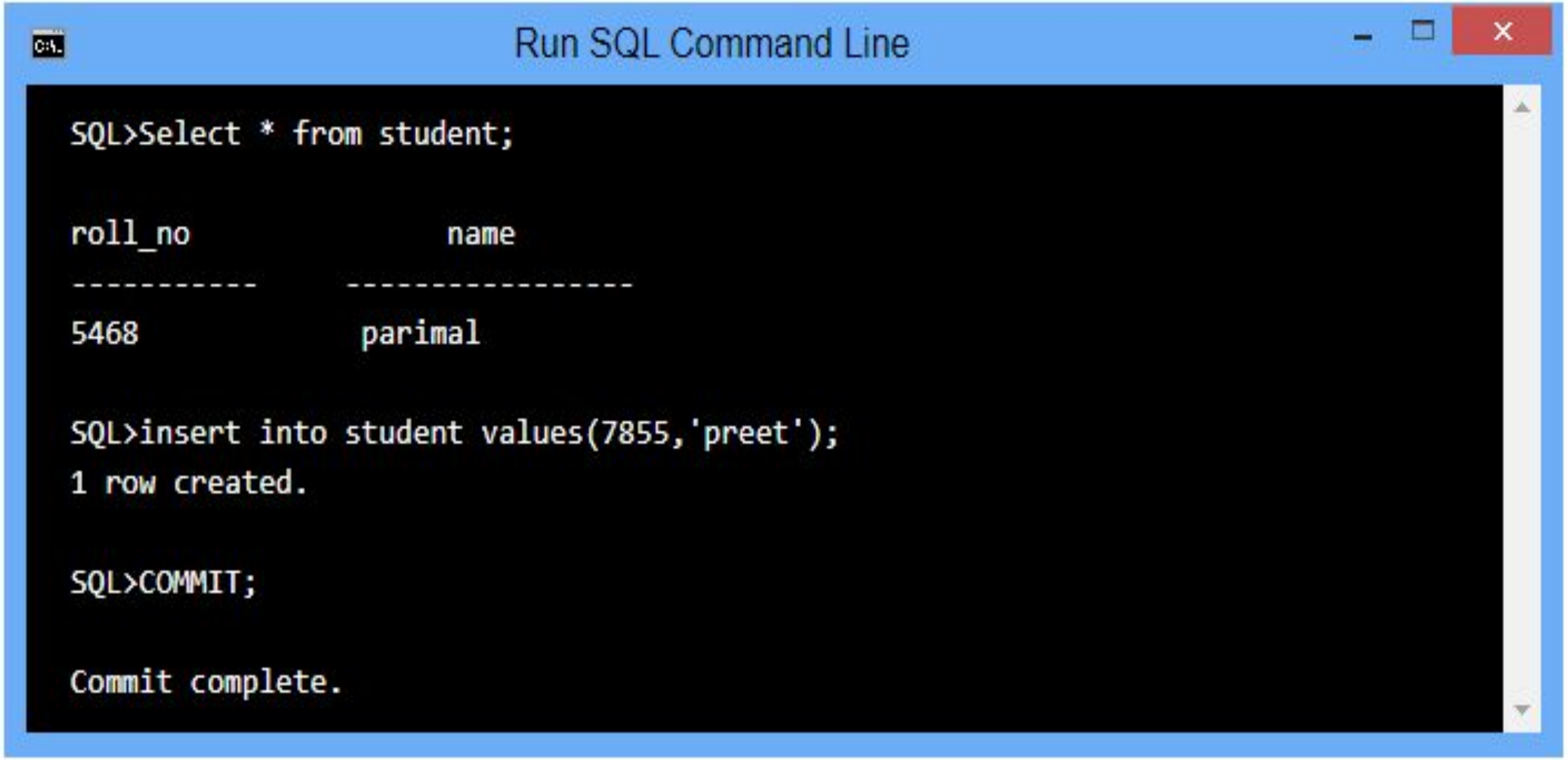
```
UPDATE EMPLOYEE SET EMP_ LOC = 'Hyderabad' WHERE
EMP_NAME= 'Raju';

COMMIT;
```

The update transaction is completed with the commit command as above and the usage of the above statements will update the location of the employee ‘Raju’ and the change will be saved in the database permanently.

- The updated table is as shown below:

EMP_ID	EMP_NAME	EMP_LOC
1356	Raju	Hyderabad
2678	Neeta	Bangalore
9899	Sanjay	Hyderabad



Run SQL Command Line

```
SQL>Select * from student;

roll_no              name
-----------     -----------------
5468             parimal

SQL>insert into student values(7855,'preet');
1 row created.

SQL>COMMIT;

Commit complete.
```

- *ROLLBACK :*
- The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.

- syntax :

ROLLBACK;

- Also, the ROLLBACK command is used along with savepoint command to leap to a save point in a transaction.
- syntax :

```
ROLLBACK TO <savepoint_name>;
```

- Let us consider that we have updated EMP_LOC for Raju to Bangalore later and realize that the update was done mistakenly as below. Then we can restore the EMP_LOC for ‘Raju’ to Hyderabad again by using the Rollback command as below.

- **Query:**

```
UPDATE EMPLOYEE SET EMP_LOC= 'Bangalore' WHERE EMP_NAME =
'Raju';
ROLLBACK;
```

- After the wrong update the table is as below:

EMP_ID	EMP_NAME	EMP_LOC
1356	Raju	Bangalore
2678	Neeta	Bangalore
9899	Sanjay	Hyderabad

- After the Rollback is performed, the location for Raju is restored to the last committed state as shown below.

EMP_ID	EMP_NAME	EMP_LOC
1356	Raju	Hyderabad
2678	Neeta	Bangalore
9899	Sanjay	Hyderabad

- **3. SAVEPOINT**
- This command is used to save the transaction temporarily. So the users can rollback to the required point of the transaction.
- syntax :

```
SAVEPOINT savepoint_name;
```

- Let us take the example of a table “ORDERS” with columns as ORDER_ID and ITEM_NAME.

ORDER_ID	**ITEM_NAME**
199	TELEVISION
290	CAMERA

- Let us insert the below values to the ORDERS table below and perform the updates using savepoint.
- **Query:**

```
INSERT INTO ORDERS VALUES ('355' , 'CELL PHONE');
COMMIT;
UPDATE ORDERS SET ITEM_NAME = 'SMART PHONE' WHERE
ORDER_ID= '355';
SAVEPOINT A;
INSERT INTO ORDERS VALUES ('566' , 'BLENDER');
SAVEPOINT B;
```

- Now the ORDERS table will be as below:

ORDER_ID	ITEM_NAME
199	TELEVISION
290	CAMERA
355	SMART PHONE
566	BLENDER

- Now we can use the SAVEPOINT command to Rollback the transaction. Let us Rollback the transaction to savepoint A.
- **Query:**

```
ROLLBACK TO A;
```

The ORDERS table will be as below:

ORDER_ID	ITEM_NAME
199	TELEVISION
290	CAMERA
355	SMART PHONE

Concurrent Executions

- 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 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

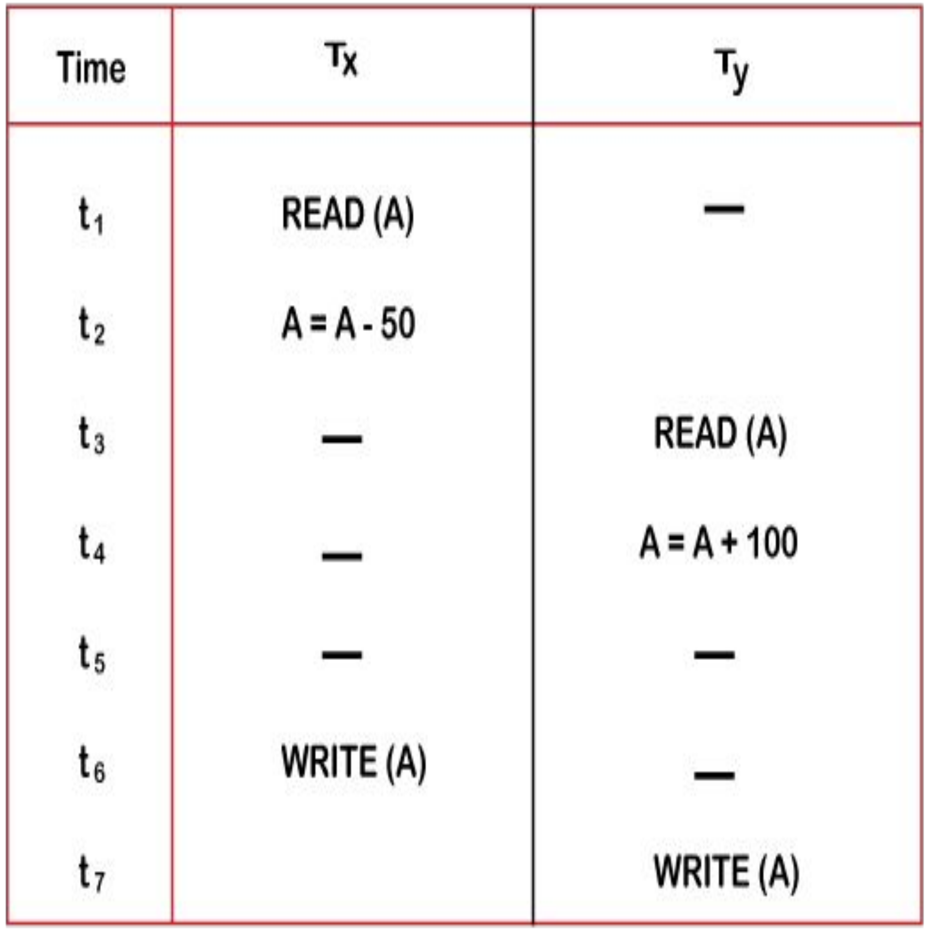
- In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the 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	Tx	Ty
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

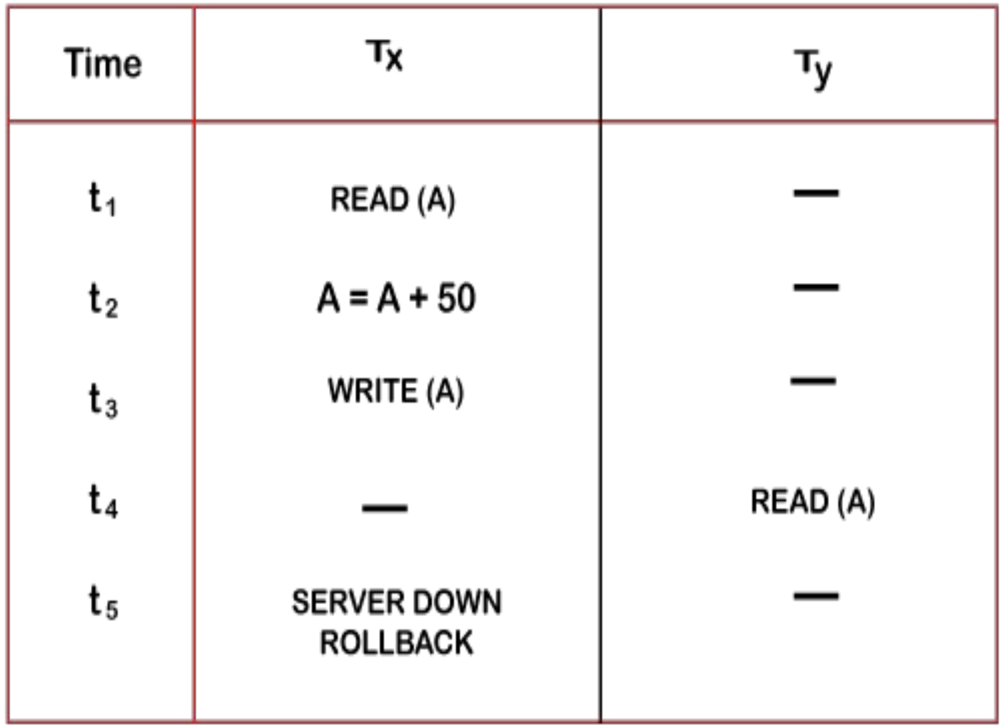
- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, 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 t7, transaction T_Y writes the values of account A, so it will write as done at time t4 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.

2. 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	Tx	Ty
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 t1, transaction TX reads the value of account A, i.e., $300.
At time t2, transaction TX adds $50 to account A that becomes $350.
At time t3, transaction TX writes the updated value in account A, i.e., $350.
Then at time t4, transaction TY reads account A that will be read as $350.
Then at time t5, transaction TX rollbacks due to server problem, and the value changes back to $300 (as initially).
But the value for account A remains $350 for transaction TY as committed, which is the dirty read and therefore known as the Dirty Read Problem.

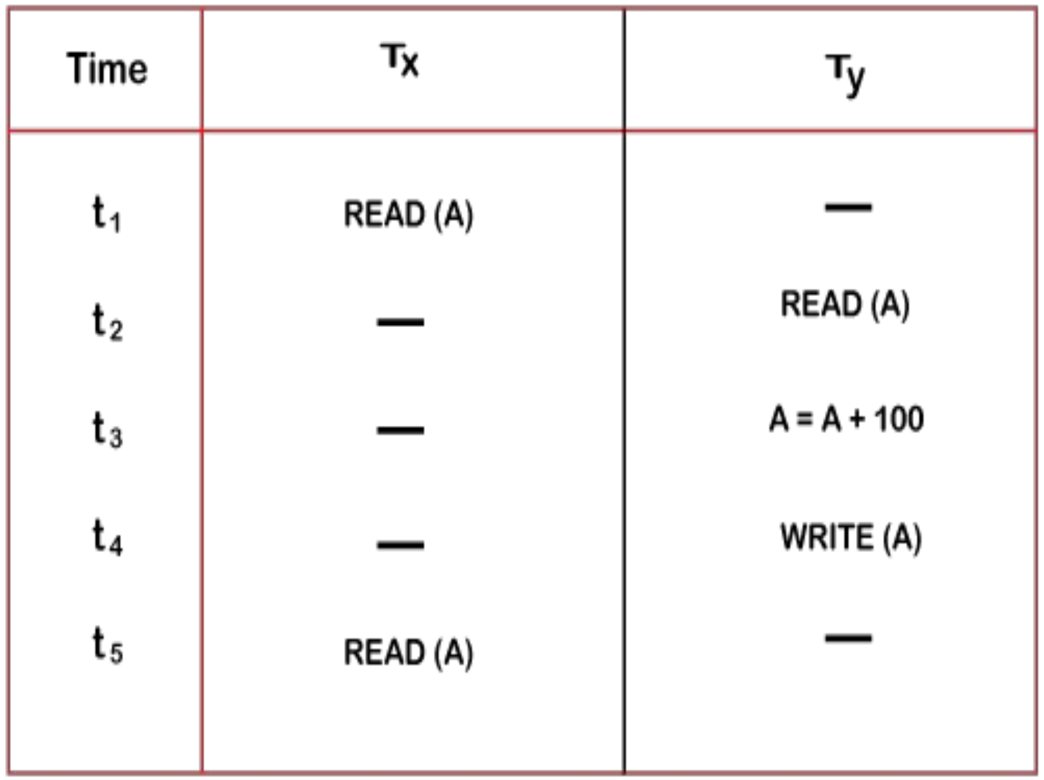
- At time t1, transaction T_X reads the value of account A, i.e., \$300.
- At time t2, transaction T_X adds \$50 to account A that becomes \$350.
- At time t3, transaction T_X writes the updated value in account A, i.e., \$350.
- Then at time t4, transaction T_Y reads account A that will be read as \$350.
- Then at time t5, 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.

3. 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, TX and TY, performing the read/write operations on account A, having an available balance = $300.



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 t1, transaction TX reads the value from account A, i.e., $300.
At time t2, transaction TY reads the value from account A, i.e., $300.
At time t3, transaction TY updates the value of account A by adding $100 to the available balance, and then it becomes $400.
At time t4, transaction TY writes the updated value, i.e., $400.
After that, at time t5, transaction TX reads the available value of account A, and that will be read as $400.
It means that within the same transaction TX, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction TY, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

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:

1. Lock Based Concurrency Control Protocol
2. Time Stamp Concurrency Control Protocol
3. Validation Based Concurrency Control Protocol

1.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.

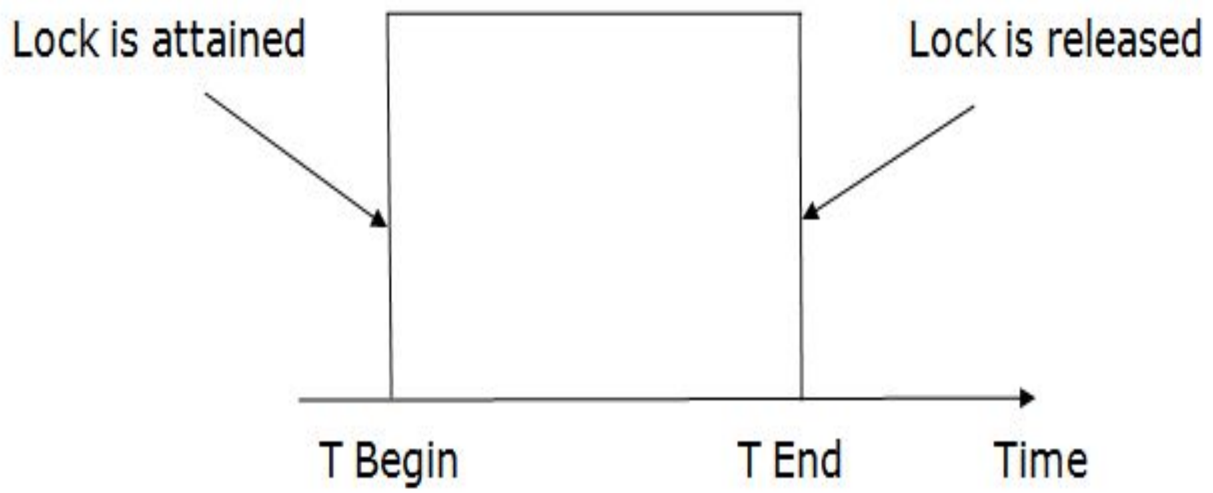
- There are four types of lock protocols available:

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.

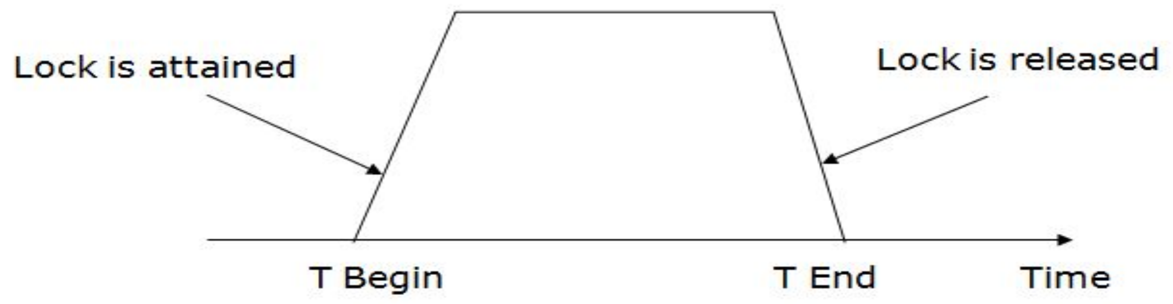
2. Pre-claiming Lock Protocol

- 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.


Lock is attained
Lock is released
T Begin
T End
Time

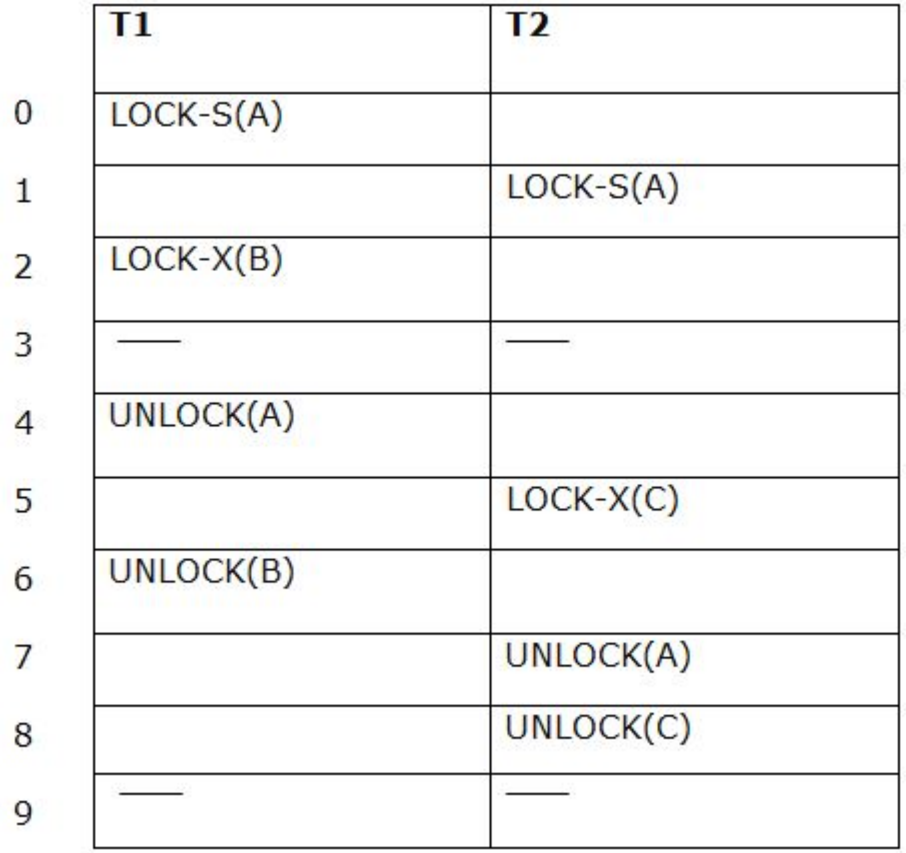
3. Two-phase locking (2PL)

- 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:
- 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.

- **Example:**



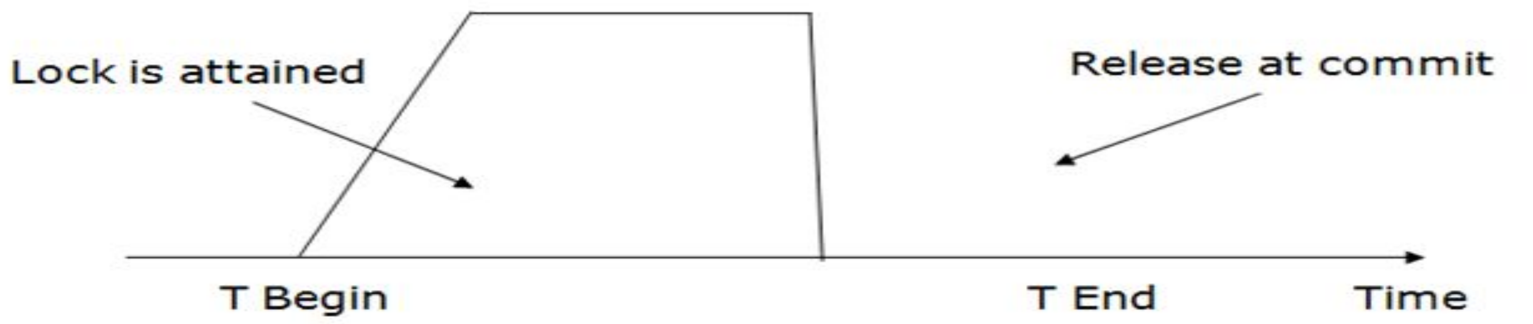
	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

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.



2.Timestamp Based Protocol

- The Timestamp-based protocol is the most commonly used concurrency control protocol and is used to order the execution of the concurrent transaction based on their Timestamp. This protocol uses the **logical counter** or **system time** to determine the timestamp of the transaction. It also maintains the timestamp of last **‘read’** and **‘write’** operations on the data.
- The lock-based protocol acquires locks at the time of execution. But in this protocol, as soon as the transaction is created, it assigns the order of the concurrent transactions. The priority of an older transaction is higher; that’s why it starts its execution first.

- **Two types of Timestamp:**
- **1. W_Timestamp(X):** This represents the largest timestamp of any transaction that executes **write(X)** successfully.
- **2. R_Timestamp(X):** This represents the largest timestamp of any transaction that executes **Read(X)** successfully.

- **Timestamp ordering protocol**
- The timestamp ordering protocol ensures that any conflicting read and write operation are executed in timestamp order.

Following are the two cases which describe the work of basic timestamp ordering protocol:

1. Suppose a transaction Ti read an item (X), check the following condition:

- If Timestamp(Ti) < W_Timestamp(X), then the read operation is rejected and Ti is rolled back.
- If Timestamp(Ti) > W_Timestamp(X), then the operation is executed.

2. Suppose a transaction Ti write an item (X), check the following condition:

- If Timestamp(Ti) < R_Timestamp(X) then the operation is rejected.
- If Timestamp(Ti) < W_Timestamp(X) then the write operation is rejected and Ti transaction is rolled back otherwise, other operations are executed.

What is Serializability?

- Serializability of schedules ensures that a non-serial schedule is equivalent to a serial schedule. It helps in maintaining the transactions to execute simultaneously without interleaving one another. In simple words, serializability is a way to check if the execution of two or more transactions are maintaining the database consistency or not.

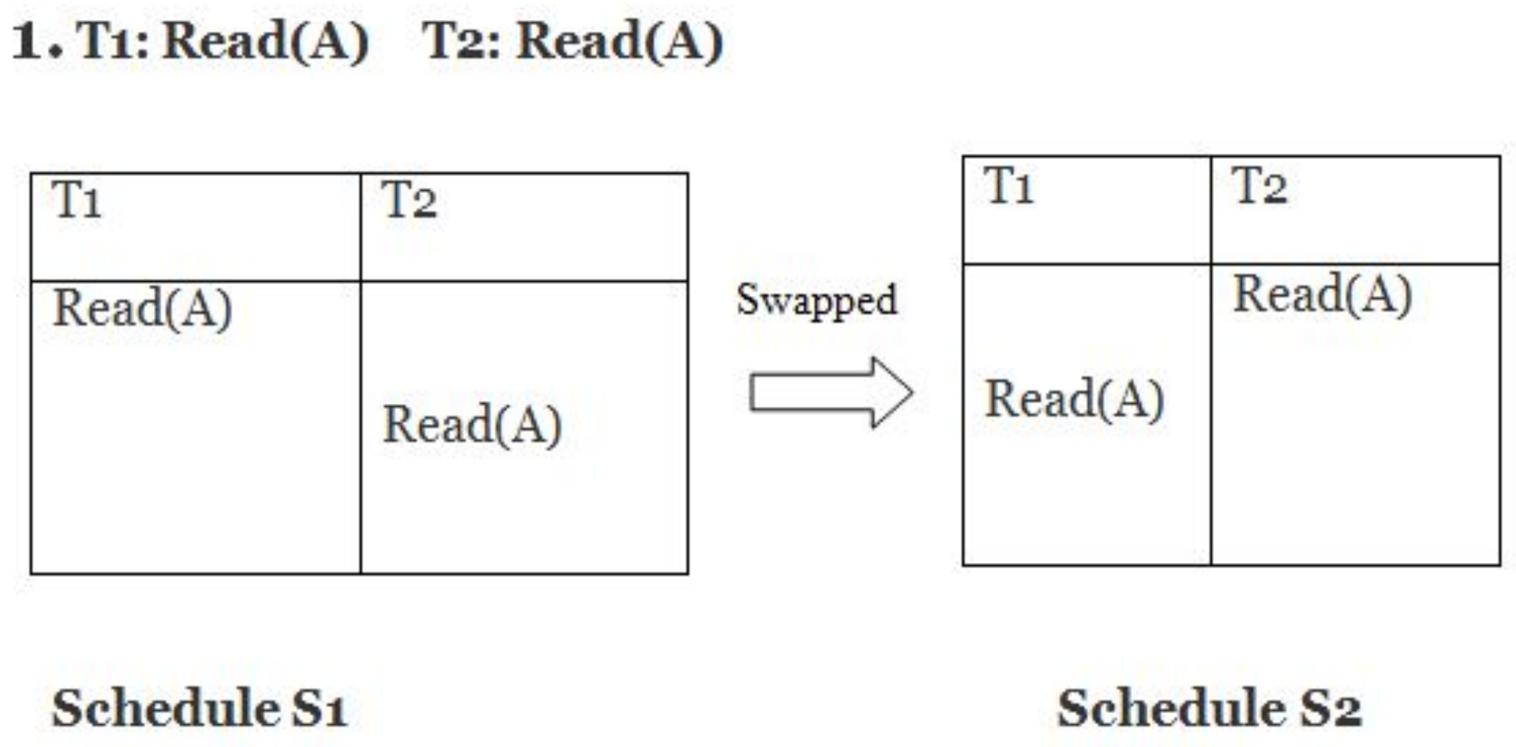
- Conflict Serializable Schedule
  - A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
  - The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

Conflicting Operations

The two operations become conflicting if all conditions satisfy:

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

- Here, S1 = S2. That means it is non-conflict.


1. T1: Read(A)   T2: Read(A)

T1	T2
Read(A)	
	Read(A)

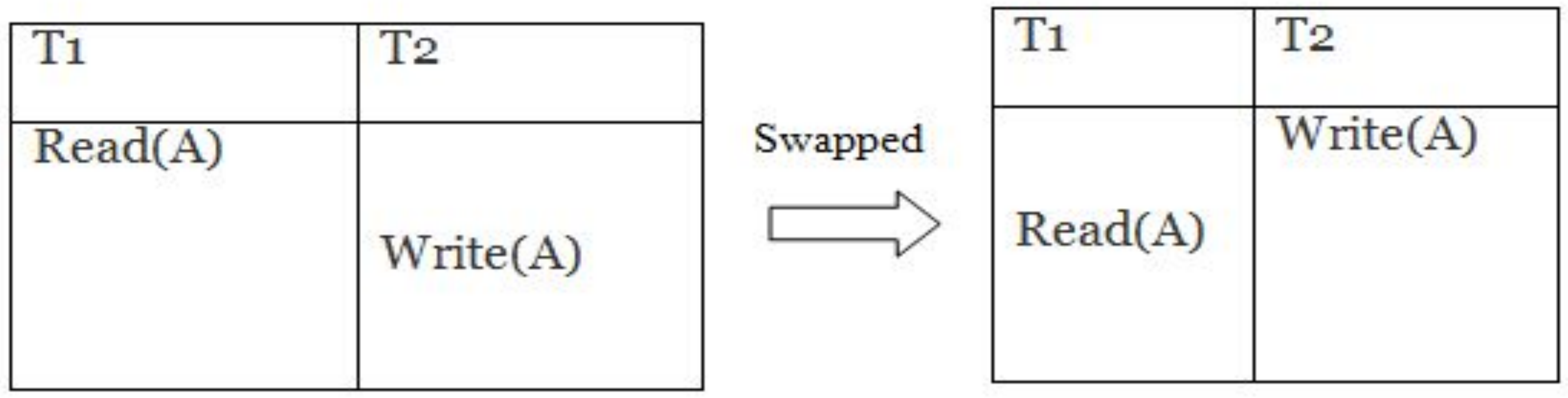
T1	T2
	Read(A)
Read(A)	

Schedule S1

Schedule S2

Here, S1 = S2. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)



T1	T2
Read(A)	
	Write(A)

T1	T2
	Write(A)
Read(A)	

Schedule S1 **Schedule S2**

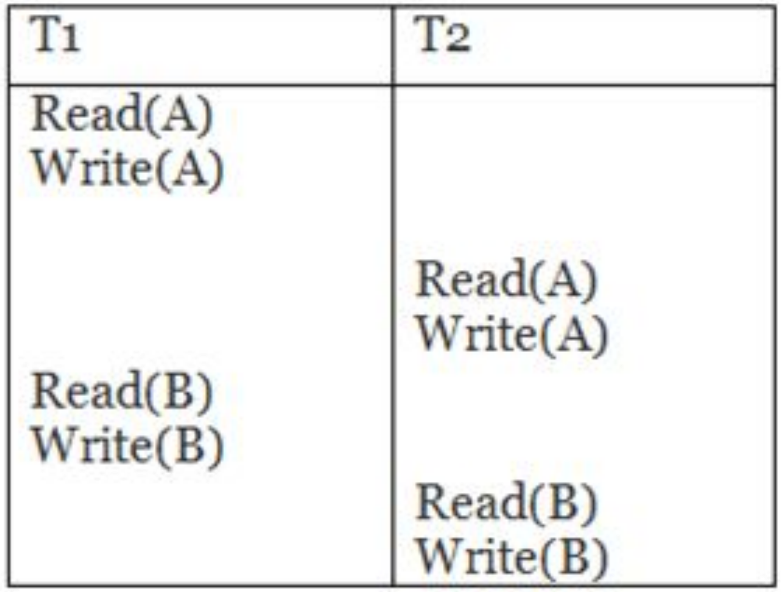
Here, S1 ≠ S2. That means it is conflict.

- Conflict Equivalent
- In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations. In the given example, S2 is conflict equivalent to S1 (S1 can be converted to S2 by swapping non-conflicting operations).
- Two schedules are said to be conflict equivalent if and only if:

1. They contain the same set of the transaction.
2. If each pair of conflict operations are ordered in the same way.

Example:

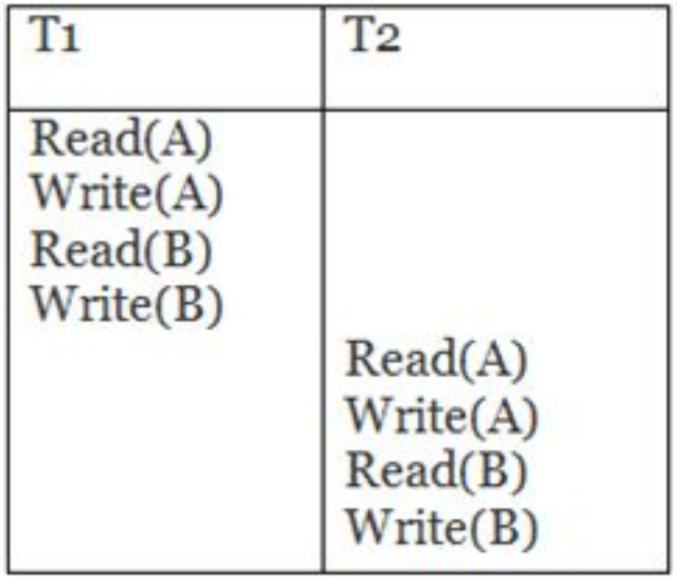
Non-serial schedule



T1	T2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

Schedule S1

Serial Schedule



T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Schedule S2

- Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2. Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operations of S1.

After swapping of non-conflict operations, the schedule S1 becomes:

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

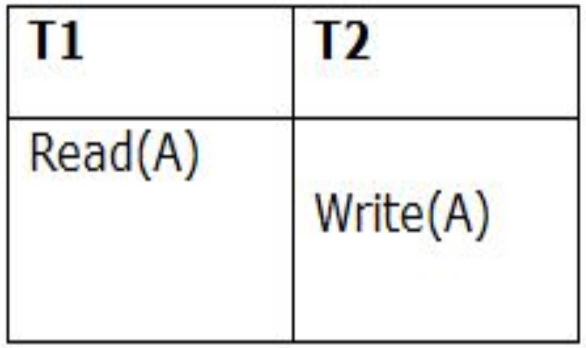
2.View Serializability

- 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:

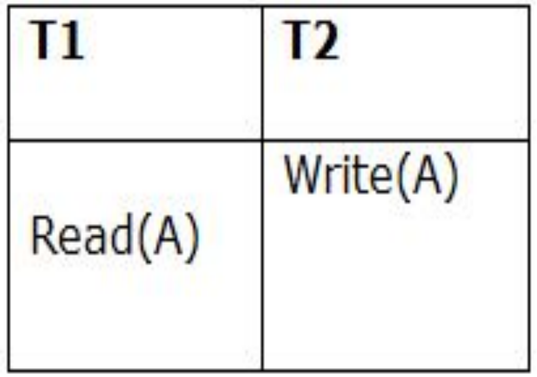
1. Initial Read

- 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



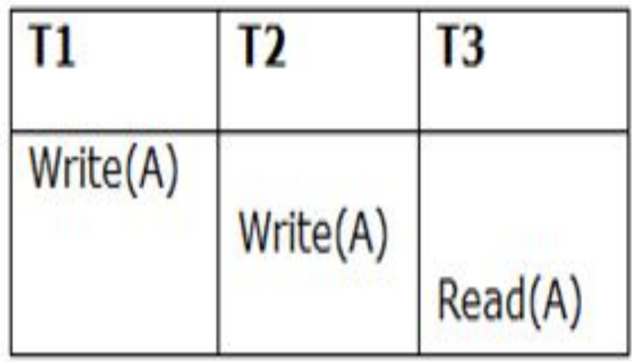
T1	T2
	Write(A)
Read(A)	

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.

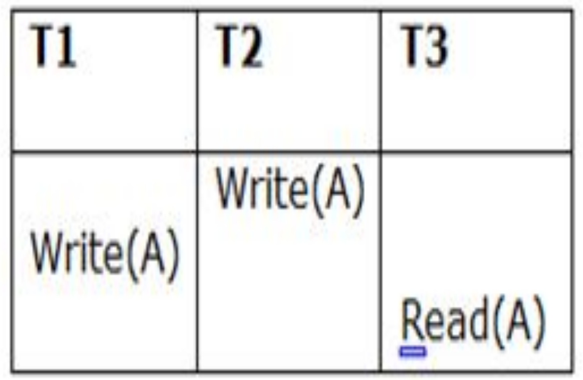
2. Updated Read

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



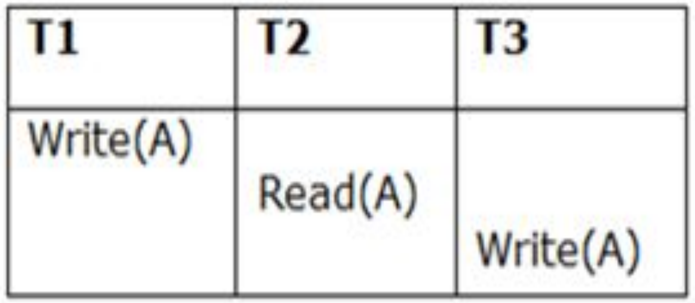
T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

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.

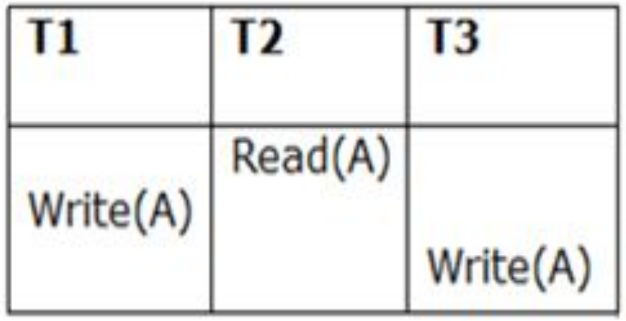
3. Final Write

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
	Read(A)	
Write(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.

1.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.

- Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.
- When the transaction is initiated, then it writes 'start' log.

<Tn, Start>

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

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

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

<Tn, Commit>

There are two approaches to modify the database:

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:

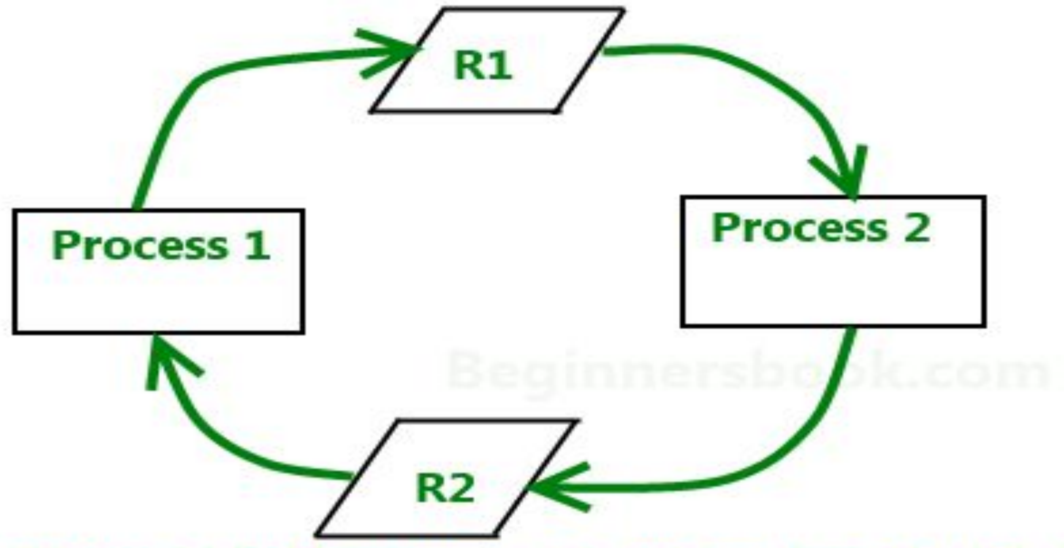
- The Immediate modification technique occurs if database modification occurs while the transaction is still active.
- In this technique, the database is modified immediately after every operation. It follows an actual database modification.

- 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.

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

Deadlock in DBMS

- A deadlock is a condition wherein two or more tasks are waiting for each other in order to be finished but none of the task is willing to give up the resources that other task needs. In this situation no task ever gets finished and is in waiting state forever



Process P1 holds resource R2 and requires R1 while Process P2 holds resource R1 and requires R2.

- 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

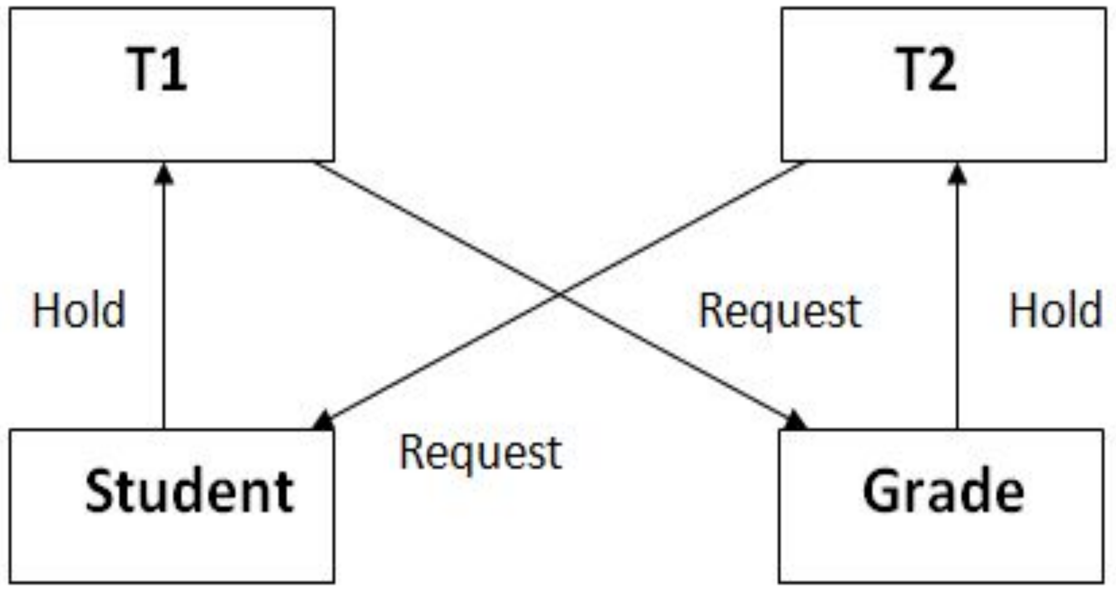
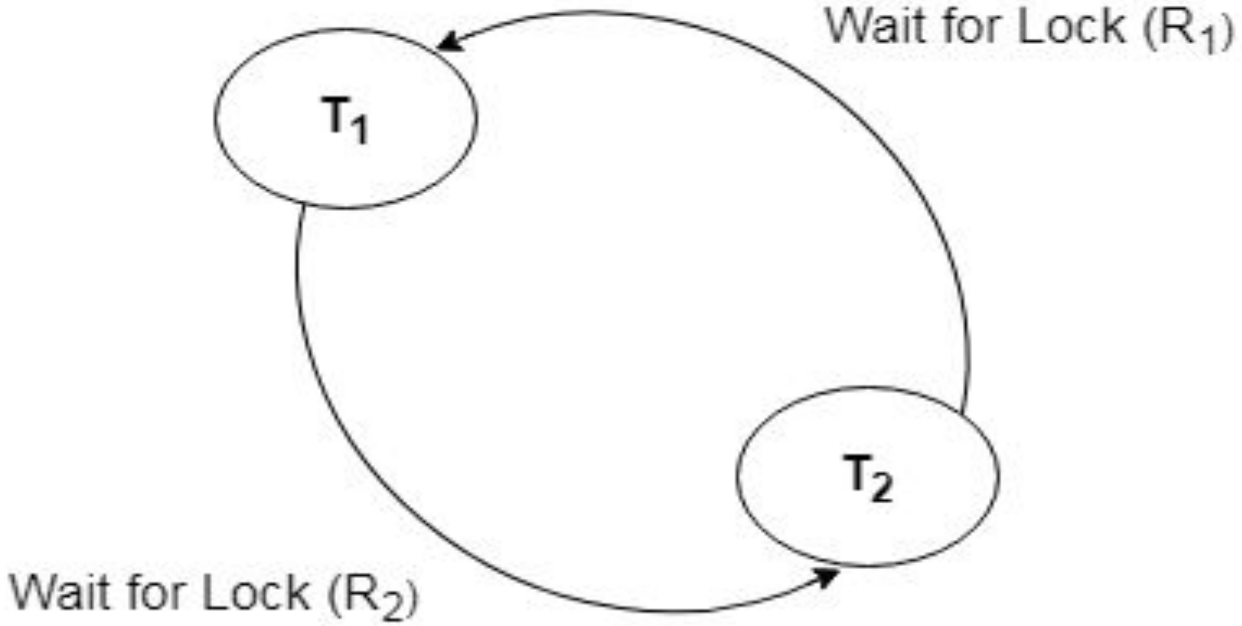


Figure: Deadlock in DBMS

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.


Wait for Lock (R_1)
T_1
T_2
Wait for Lock (R_2)

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.

1.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.

- Consider 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.

2.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.