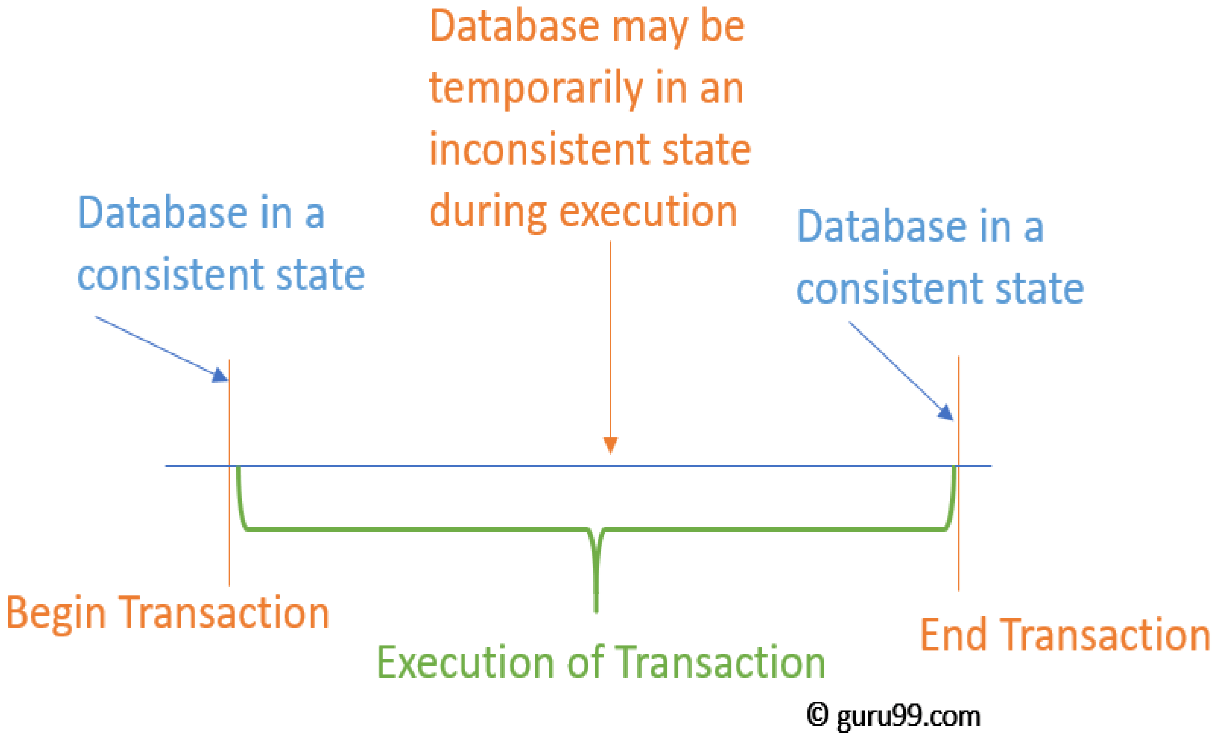
Chapter 6:Transaction management and Concurrency and Recovery

What is a Database Transaction?

A Database Transaction is a logical unit of processing in a DBMS which entails one or more database access operations. In a nutshell, database transactions represent real-world events of any enterprise.
All types of database access operations which are held between the beginning and end transaction statements are considered as a single logical transaction in DBMS. During the transaction the database is inconsistent. Only once the database is committed the state is changed from one consistent state to another.


Database may be temporarily in an inconsistent state during execution
Database in a consistent state
Database in a consistent state
Begin Transaction
Execution of Transaction
End Transaction
© guru99.com

Facts about Database Transactions

- **A transaction is a program unit whose execution may or may not change the contents of a database.**
- **The transaction concept in DBMS is executed as a single unit.**
- **If the database operations do not update the database but only retrieve data, this type of transaction is called a read-only transaction.**
- **A successful transaction can change the database from one CONSISTENT STATE to another**
- **DBMS transactions must be atomic, consistent, isolated and durable**
- **If the database were in an inconsistent state before a transaction, it would remain in the inconsistent state after the transaction.**

Why do you need concurrency in Transactions?

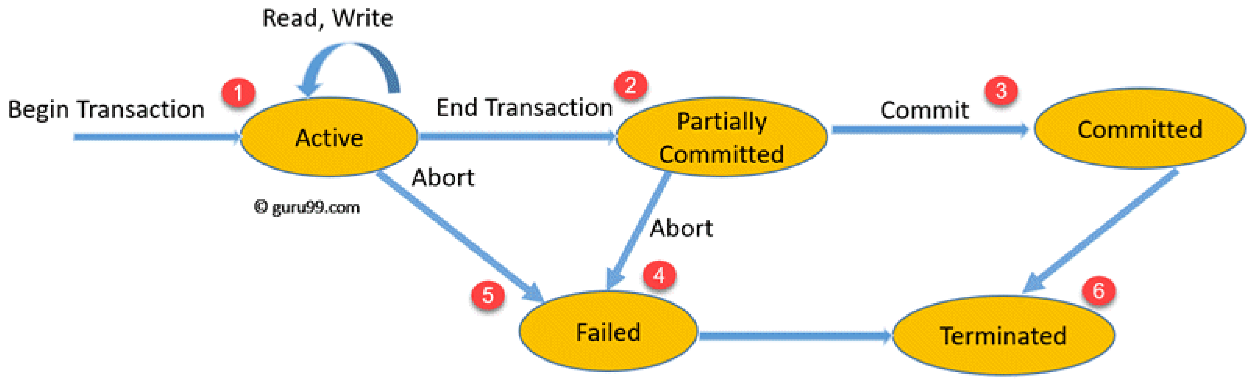
A database is a shared resource accessed. It is used by many users and processes concurrently. For example, the banking system, railway, and air reservations systems, stock market monitoring, supermarket inventory, and checkouts, etc.
Not managing concurrent access may create issues like:

- **Hardware failure and system crashes**
- **Concurrent execution of the same transaction, deadlock, or slow performance**

States of Transactions

The various states of a transaction concept in DBMS are listed below:

State	Transaction types
Active State	**A transaction enters into an active state when the execution process begins. During this state read or write operations can be performed.**
Partially Committed	**A transaction goes into the partially committed state after the end of a transaction.**
Committed State	**When the transaction is committed to the state, it has already completed its execution successfully. Moreover, all of its changes are recorded to the database permanently.**
Failed State	**A transaction is considered a failure when any one of the checks fails or if the transaction is aborted while it is in the active state.**
Terminated State	**State of the transaction reaches a terminated state when certain transactions which are leaving the system can't be restarted.**



State Transition Diagram for a Database Transaction

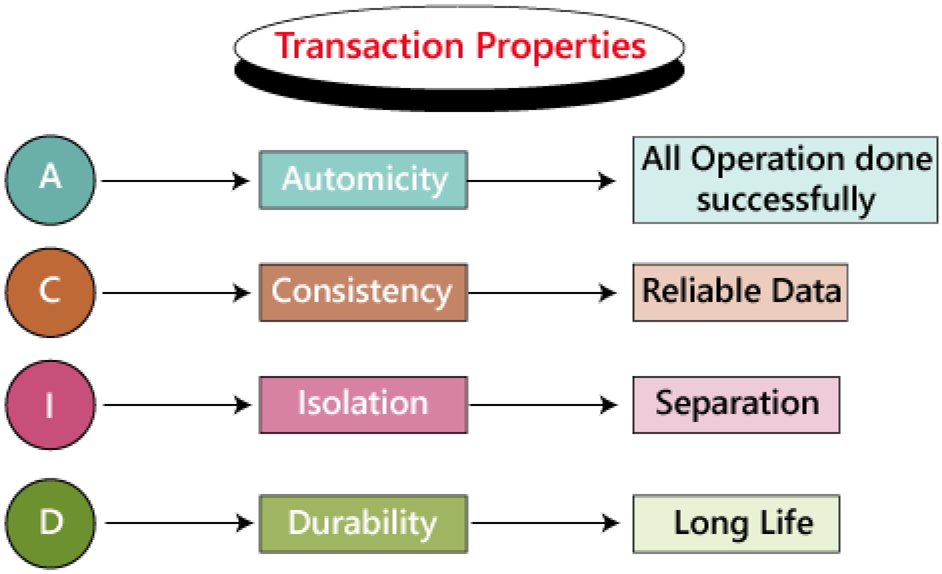
Let’s study a state transition diagram that highlights how a transaction moves between these various states.

1. **Once a transaction states execution, it becomes active. It can issue READ or WRITE operations.**
2. **Once the READ and WRITE operations complete, the transaction becomes partially committed state.**
3. **Next, some recovery protocols need to ensure that a system failure will not result in an inability to record changes in the transaction permanently. If this check is a success, the transaction commits and enters into the committed state.**
4. **If the check is a fail, the transaction goes to the Failed state.**
5. **If the transaction is aborted while it’s in the active state, it goes to the failed state. The transaction should be rolled back to undo the effect of its write operations on the database.**
6. **The terminated state refers to the transaction leaving the system.**

What are ACID Properties?

ACID Properties are used for maintaining the integrity of the database during transaction processing. ACID in DBMS stands for Atomicity, Consistency, Isolation, and Durability.

- **Atomicity: A transaction is a single unit of operation. You either execute it entirely or do not execute it at all. There cannot be partial execution.**
- **Consistency: Once the transaction is executed, it should move from one consistent state to another.**
- **Isolation: Transactions should be executed in isolation from other transactions (no Locks). During concurrent transaction execution, intermediate transaction results from simultaneously executed transactions should not be made available to each other.**
- **Durability: · After successful completion of a transaction, the changes in the database should persist. Even in the case of system failures.**



Let’s take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

A’s Account

```
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)
```

B’s Account

```
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)
```

Transaction control commands:

The commands are used to control the transactions, these commands are given below-

- **COMMIT- It is used to save the changes.**
- **ROLLBACK- ROLLBACK is used to roll back the changes.**
- **SAVEPOINT - SAVEPOINT creates the group of transactions with the help of rollback command.**

Transactional commands are used with the help of DML commands such as INSERT, UPDATE and DELETE. It cannot be applied when we are creating or dropping the tables.

COMMIT Command

It is also known as Transactional Command. The command is used by the database to save changes.

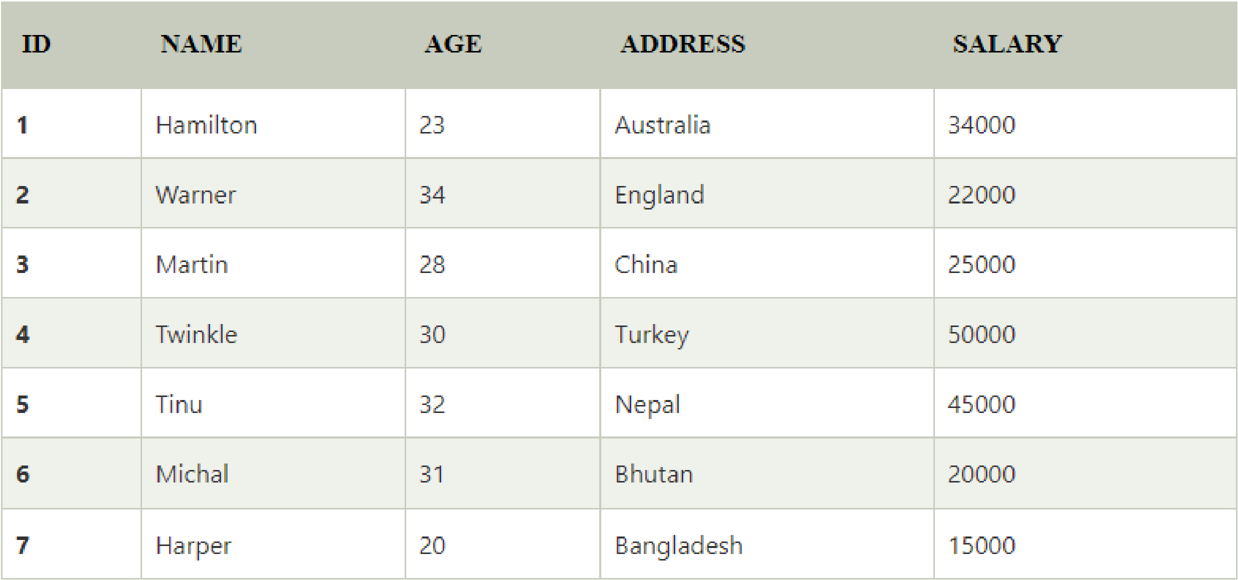
Syntax:

The syntax of the COMMIT command is given below.

1. **COMMIT;**

Example

See the EMPLOYEES table, which has the records.

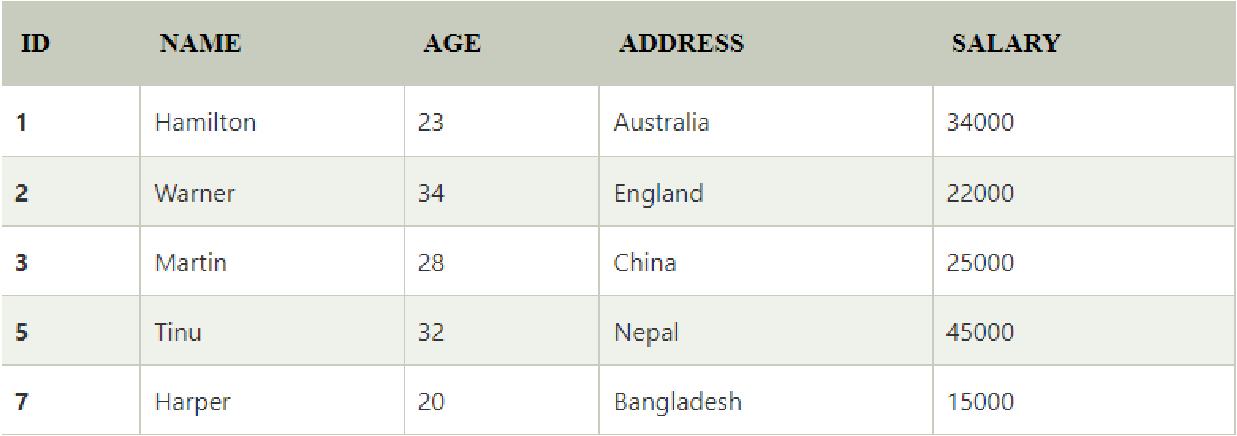


ID	NAME	AGE	ADDRESS	SALARY
1	Hamilton	23	Australia	34000
2	Warner	34	England	22000
3	Martin	28	China	25000
4	Twinkle	30	Turkey	50000
5	Tinu	32	Nepal	45000
6	Michal	31	Bhutan	20000
7	Harper	20	Bangladesh	15000

Following command example will delete records from the table having age = 30 and then COMMIT the changes in the database.

1. **Begin Tran**
2. **DELETE FROM EMPLOYEES**
3. **WHERE AGE = 30**
4. **COMMIT**

In the result part, 4 and 6 rows from the table have been deleted. SELECT statement is generating the output given below:



ID	NAME	AGE	ADDRESS	SALARY
1	Hamilton	23	Australia	34000
2	Warner	34	England	22000
3	Martin	28	China	25000
5	Tinu	32	Nepal	45000
7	Harper	20	Bangladesh	15000

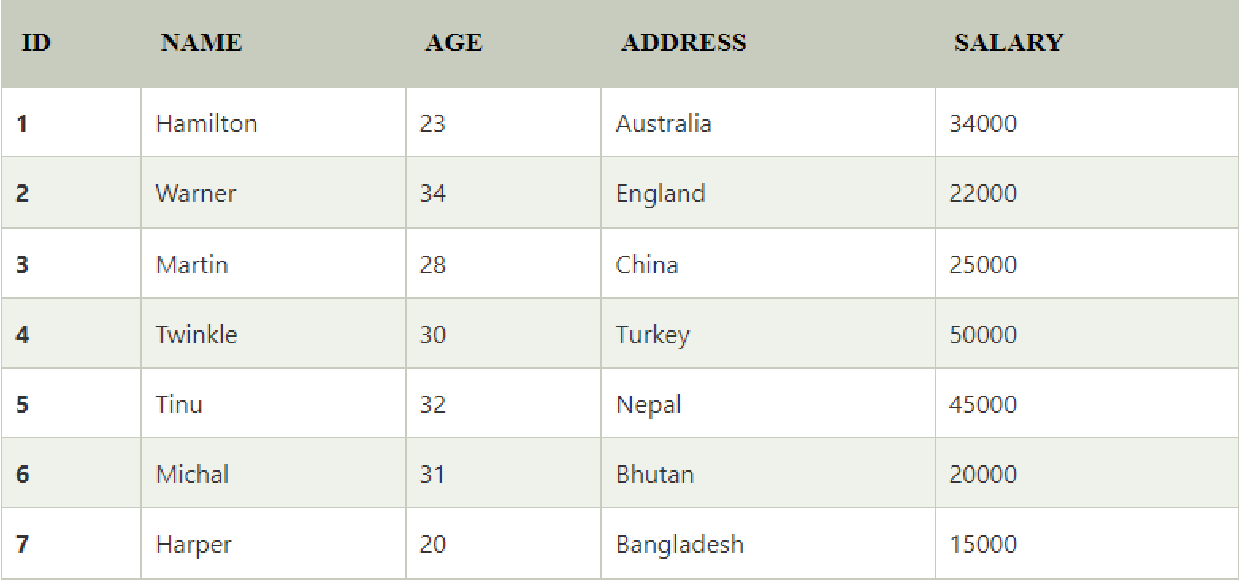
ROLLBACK Command:

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. Rollback command is used to undo the set of transactions.

Syntax:

```
ROLLBACK
```

Example:
See the EMPLOYEES table, which has the following records.

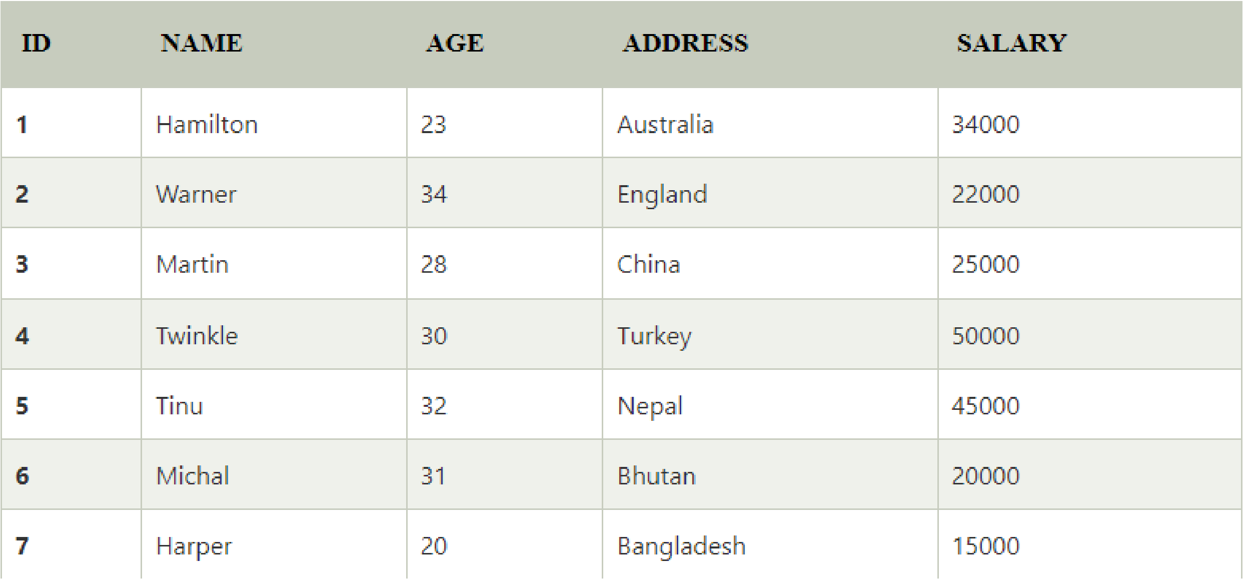


ID	NAME	AGE	ADDRESS	SALARY
1	Hamilton	23	Australia	34000
2	Warner	34	England	22000
3	Martin	28	China	25000
4	Twinkle	30	Turkey	50000
5	Tinu	32	Nepal	45000
6	Michal	31	Bhutan	20000
7	Harper	20	Bangladesh	15000

The following command example will delete records from the table having age = 20 and then ROLLBACK the changes in the database.

```
Begin Tran
DELETE FROM EMPLOYEES
WHERE AGE = 20;
ROLLBACK
```

The delete operations have no effect on the result of the table.



ID	NAME	AGE	ADDRESS	SALARY
1	Hamilton	23	Australia	34000
2	Warner	34	England	22000
3	Martin	28	China	25000
4	Twinkle	30	Turkey	50000
5	Tinu	32	Nepal	45000
6	Michal	31	Bhutan	20000
7	Harper	20	Bangladesh	15000

SAVEPOINT Command:

SAVEPOINT is the point in a transaction when we roll the transaction back to a certain point without rolling the entire operation.

Syntax:

The syntax of SAVEPOINT command:

```
1. SAVE TRANSACTION SAVEPOINT_NAME
```

The order serves the creation of SAVEPOINT between transaction statements.

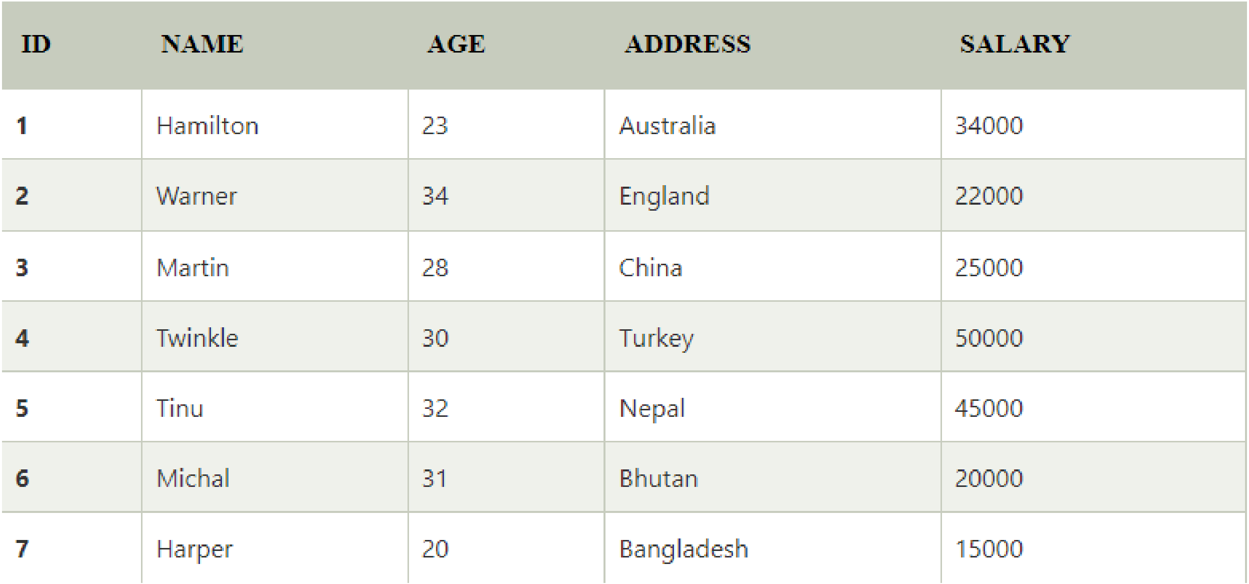
SAVEPOINT Syntax:

```
1. ROLLBACK TO SAVEPOINT_NAME
```

In the below example, we delete three different records from the EMPLOYEES table. Then create a SAVEPOINT before each delete so that we can load SAVEPOINT at the time to return the data into its original state.

Example:

Consider the EMPLOYEES table which has the following records -



ID	NAME	AGE	ADDRESS	SALARY
1	Hamilton	23	Australia	34000
2	Warner	34	England	22000
3	Martin	28	China	25000
4	Twinkle	30	Turkey	50000
5	Tinu	32	Nepal	45000
6	Michal	31	Bhutan	20000
7	Harper	20	Bangladesh	15000

Following are the series of operations -

Begin Tran

1. **SAVE Transaction SP1**

Save point created.

1. **DELETE FROM EMPLOYEES WHERE ID = 1**

1 row deleted.

1. **SAVE Transaction SP2**

Save point created.

```
DELETE FROM EMPLOYEES WHERE ID = 2
```

1 row deleted.

These commands took their place.

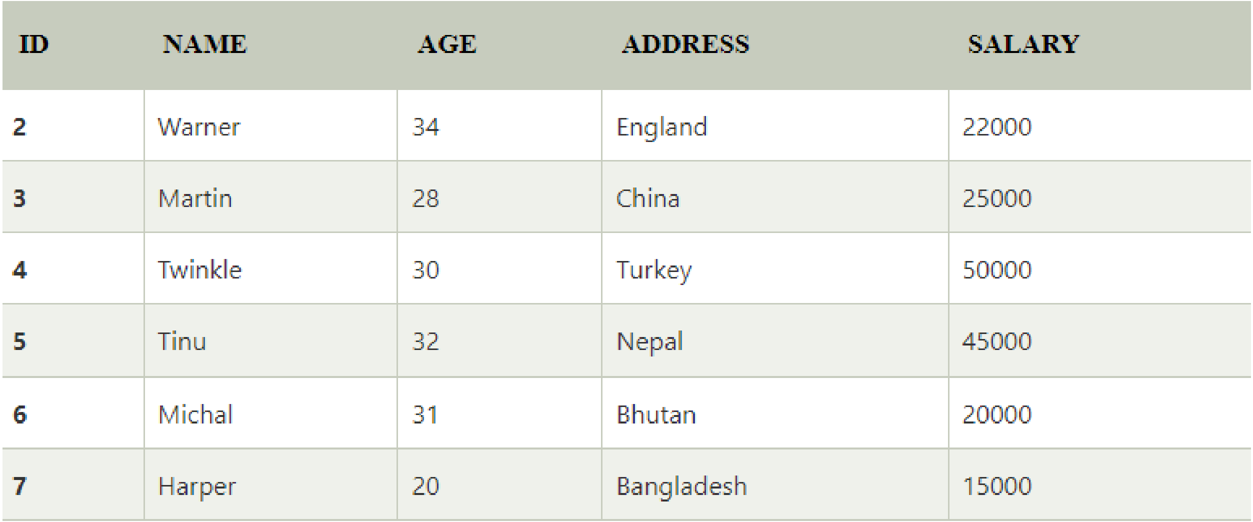
We have decided to ROLLBACK the SAVEPOINT, which is recognized as SP2. Because SP2 was created after 1 deletion, and 2 last deletions have not been done.

```
ROLLBACK Transaction SP2
```

Rollback has been completed.

```
SELECT * FROM EMPLOYEES
```

6 rows have been selected

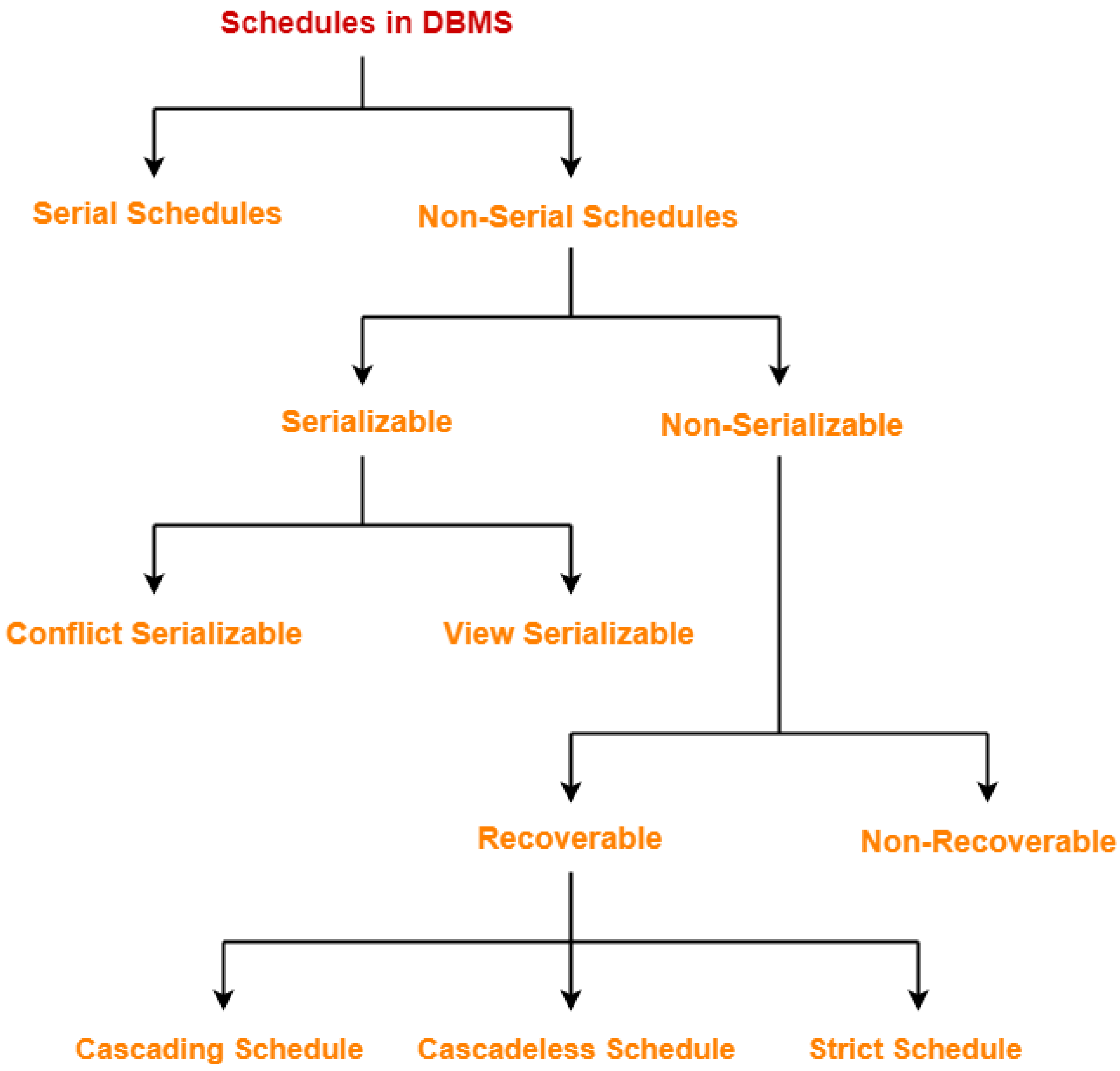


ID	NAME	AGE	ADDRESS	SALARY
2	Warner	34	England	22000
3	Martin	28	China	25000
4	Twinkle	30	Turkey	50000
5	Tinu	32	Nepal	45000
6	Michal	31	Bhutan	20000
7	Harper	20	Bangladesh	15000

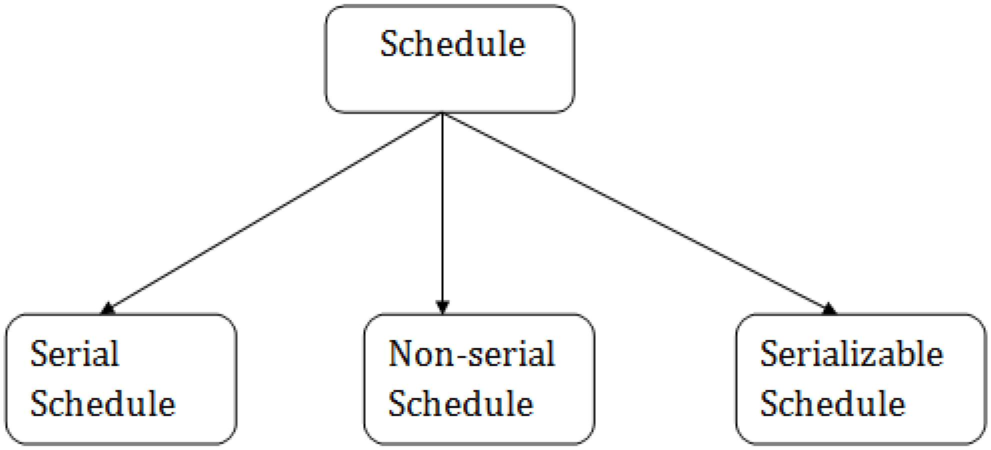
.

CONCEPT OF SCHEDULE:

Schedule



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



1. Serial Schedule

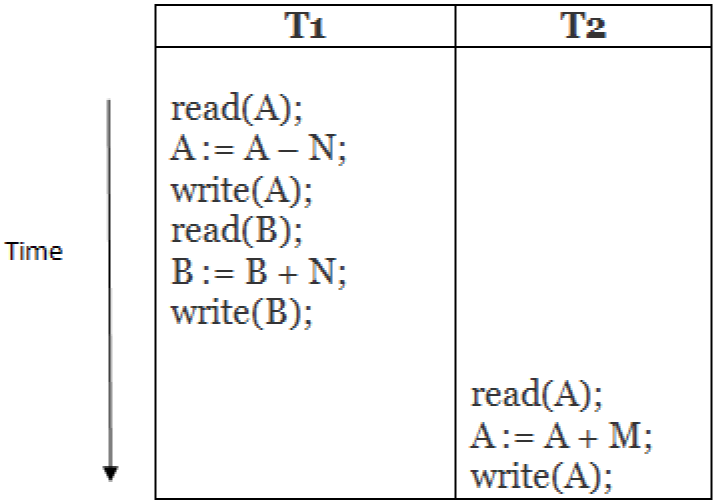
The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

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.**

(a)

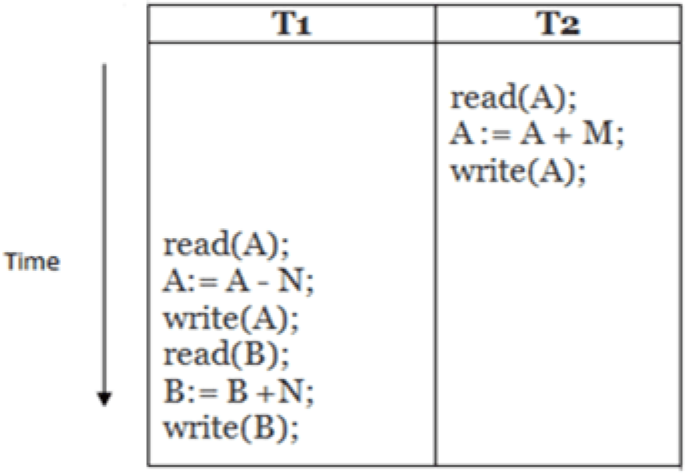


T1	T2
read(A);	
A := A – N;	
write(A);	
read(B);	
B := B + N;	
write(B);	
	read(A);
	A := A + M;
	write(A);

Schedule A

- **In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.**

(b)



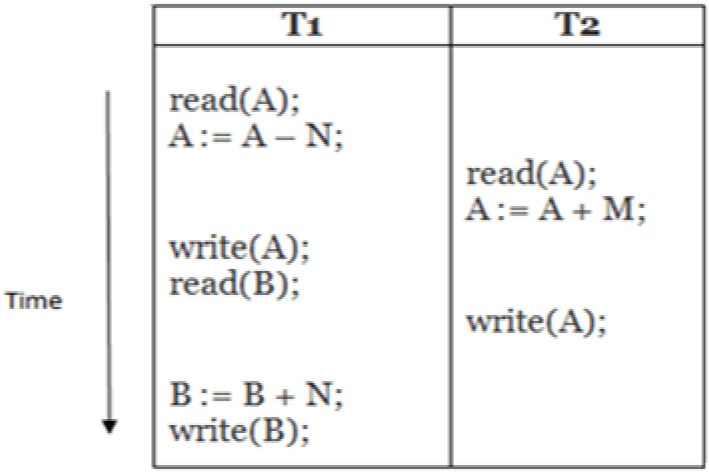
T1	T2
	read(A);
	A := A + M;
	write(A);
read(A);	
A:= A - N;	
write(A);	
read(B);	
B:= B +N;	
write(B);	

Schedule B

2. Non-serial Schedule

- **If interleaving of operations is allowed, then there will be a 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.**

(c)



T1	T2

read(A);
A := A – N;	

	read(A);
	A := A + M;

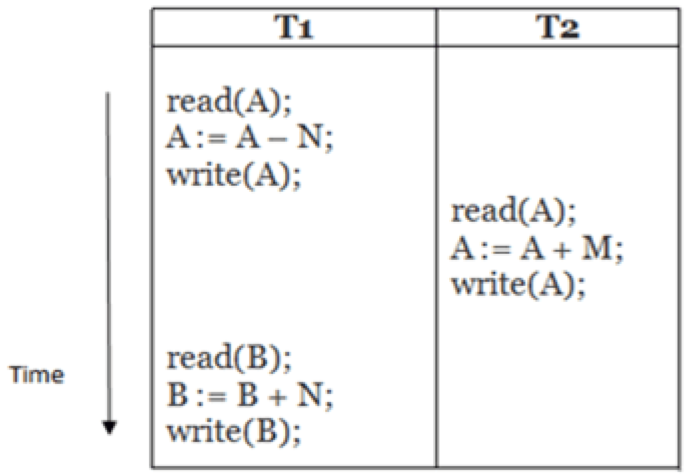
write(A);
read(B);	

	write(A);

B := B + N;
write(B);	

Schedule C

(d)



T1	T2

read(A);
A := A – N;
write(A);	

	read(A);
	A := A + M;
	write(A);

read(B);
B := B + N;
write(B);	

Schedule D

3. Serializable schedule

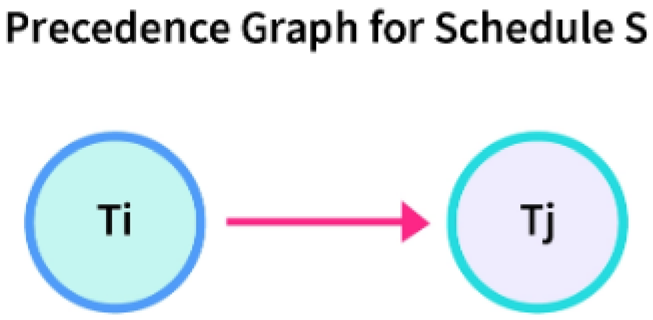
- **The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.**
- **It identifies which schedules are correct when executions of the transaction have interleaving of their operations.**
- **A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.**
- **What is a serializable schedule?**
- **A serializable schedule always leaves the database in a consistent state. A serial schedule is always a serializable schedule because in a serial schedule, a transaction only starts when the other transaction finishes execution. However a non-serial schedule needs to be checked for Serializability.**

- **A non-serial schedule of n number of transactions is said to be a serializable schedule, if it is equivalent to the serial schedule of those n transactions. A serial schedule doesn't allow concurrency, only one transaction executes at a time and the other starts when the already running transaction finishes.**

Testing of Serializability

To test the serializability of a schedule, we can use Serialization Graph or Precedence Graph. A serialization Graph is nothing but a Directed Graph of the entire transactions of a schedule.

It can be defined as a Graph G(V, E) consisting of a set of directed-edges E = {E1, E2, E3, ..., En} and a set of vertices V = {V1, V2, V3, ...,Vn}. The set of edges contains one of the two operations - READ, WRITE performed by a certain transaction.



Ti -> Tj, means Transaction-Ti is either performing read or write before the transaction-Tj.

NOTE:

If there is a cycle present in the serialized graph then the schedule is non-serializable because the cycle resembles that one transaction is dependent on the other transaction and vice versa. It also means that there are one or more conflicting pairs of operations in the transactions. On the other hand, no-cycle means that the non-serial schedule is serializable.

What is a conflicting pair in transactions?

Two operations inside a schedule are called conflicting if they meet these three conditions:

1. They belong to two different transactions.
2. They are working on the same data piece.
3. One of them is performing the WRITE operation.

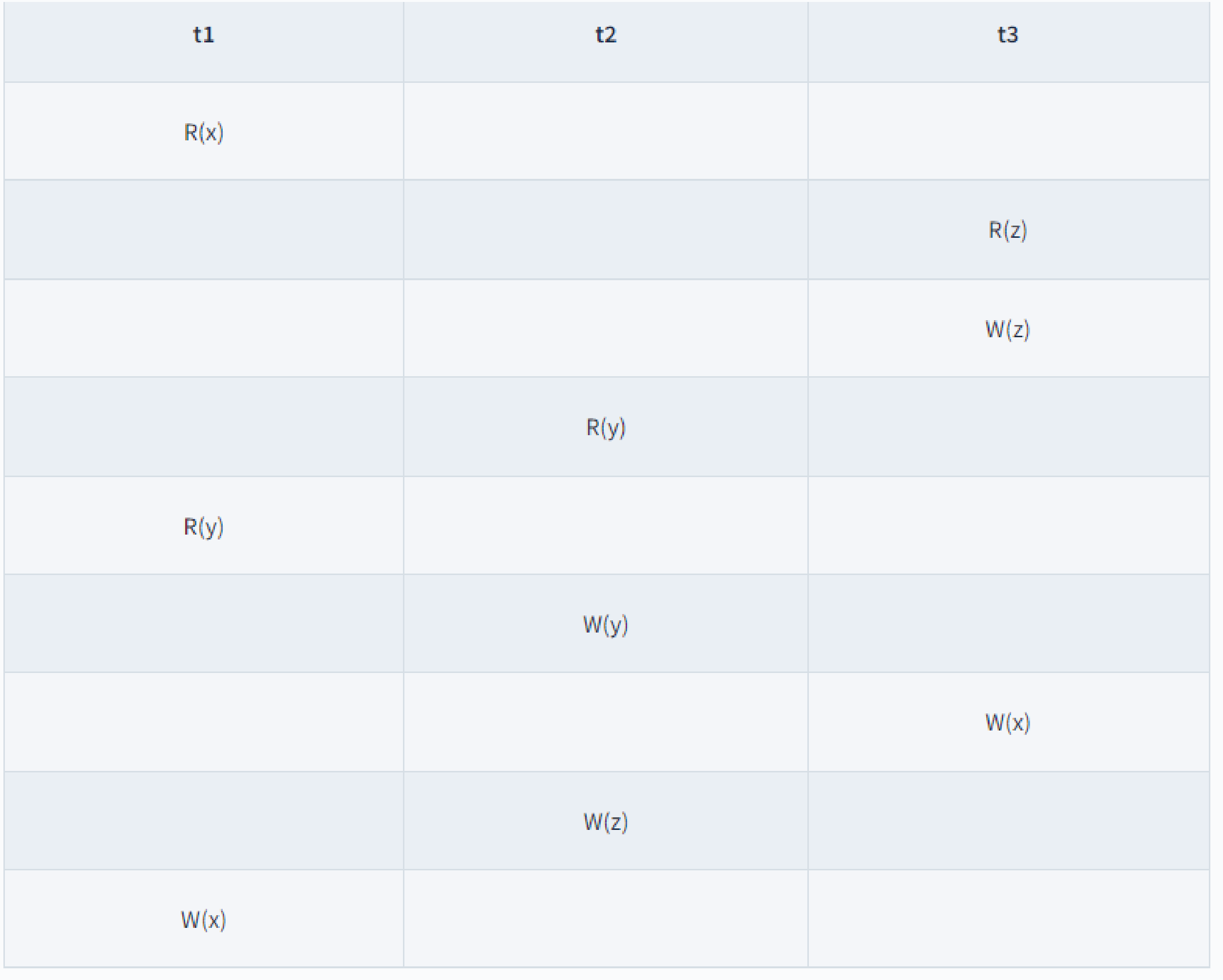
To conclude, let’s take two operations on data: "a". The conflicting pairs are:

1. READ(a) - WRITE(a)
2. WRITE(a) - WRITE(a)
3. WRITE(a) - READ(a)

Note:

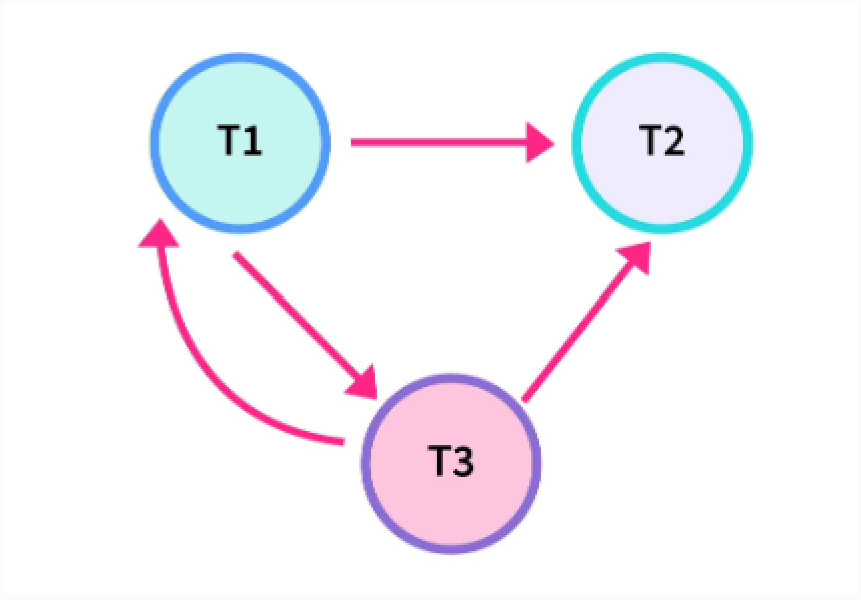
There can never be a read-read conflict as there is no change in the data.

Let's take an example of schedule "S" having three transactions t1, t2, and t3 working simultaneously, to get a better understanding.



t1	t2	t3
R(x)		
		R(z)
		W(z)
	R(y)	
R(y)		
	W(y)	
		W(x)
	W(z)	
W(x)		

PRECEDENCE GRAPH



Non-serializable schedule. R(x) of T1 conflicts with W(x) of T3, so there is a directed edge from T1 to T3. R(y) of T1 conflicts with W(y) of T2, so there is a directed edge from T1 to T2. W(y\x) of T3 conflicts with W(x) of T1, so there is a directed edge from T3 to T. Similarly, we will make edges for every conflicting pair. Now, as the cycle is formed, the transactions cannot be serializable.

Types of Serializability

Serializability of any non-serial schedule can be verified using two types mainly: Conflict Serializability and View Serializability.

Conflict Serializability

Conflict Serializability is one of the types of Serializability, which can be used to check whether a non-serial schedule is conflict serializable or not.

What is Conflict Serializability?

A schedule is called conflict serializable if we can convert it into a serial schedule after swapping its non-conflicting operations.

Conflicting operations

Two operations are said to be in conflict, if they satisfy all the following three conditions:

1. Both the operations should belong to different transactions.

2. Both the operations are working on the same data item.

3. At least one of the operations is a write operation.

Lets see some examples to understand this:

Example 1: Operation W(X) of transaction T1 and operation R(X) of transaction T2 are conflicting operations, because they satisfy all the three conditions mentioned above. They belong to different transactions, they are working on the same data item X, one of the operations in the write operation.

Example 2: Similarly Operations W(X) of T1 and W(X) of T2 are conflicting operations.

Example 3: Operations W(X) of T1 and W(Y) of T2 are non-conflicting operations because both the write operations are not working on the same data item so these operations don’t satisfy the second condition.

Example 4: Similarly R(X) of T1 and R(X) of T2 are non-conflicting operations because none of them is write operation.

Example 5: Similarly W(X) of T1 and R(X) of T1 are non-conflicting operations because both the operations belong to the same transaction T1.

Conflict Equivalent Schedules

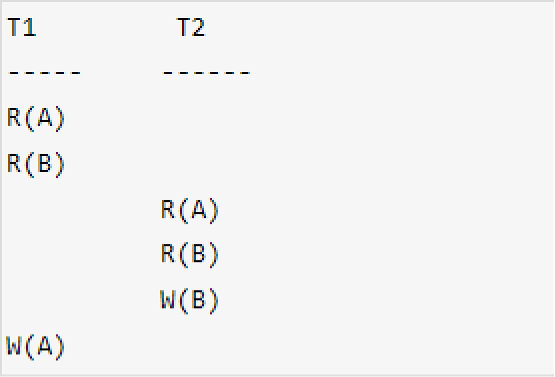
Two schedules are said to be conflict Equivalent if one schedule can be converted into another schedule after swapping non-conflicting operations.

Conflict Serializable check

Lets check whether a schedule is conflict serializable or not. If a schedule is conflict Equivalent to its serial schedule then it is called Conflict Serializable schedule. Let's take a few examples of schedules.

Example of Conflict Serializability

Let's consider this schedule:



```
T1          T2
-----     ------
R(A)
R(B)
          R(A)
          R(B)
          W(B)
W(A)
```

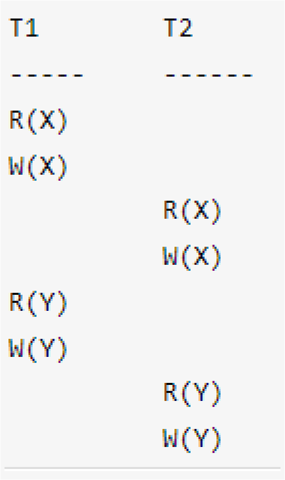
To convert this schedule into a serial schedule we must have to swap the R(A) operation of transaction T2 with the W(A) operation of transaction T1. However we cannot swap these two operations because they are conflicting operations, thus we can say that this given schedule is not Conflict Serializable.

What is View Serializability?

View Serializability is a process to find out if a given schedule is viewed serializable or not.

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

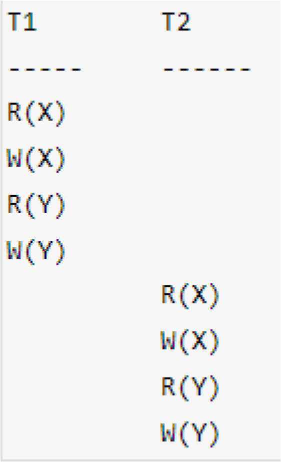
Given Schedule:



```
T1          T2
-----       ------
R(X)
W(X)
            R(X)
            W(X)
R(Y)
W(Y)
            R(Y)
            W(Y)
```

Serial Schedule of the above given schedule:

As we know that in the Serial schedule a transaction only starts when the current running transaction is finished. So the serial schedule of the above given schedule would look like this:



```
T1          T2
-----       ------
R(X)
W(X)
R(Y)
W(Y)
            R(X)
            W(X)
            R(Y)
            W(Y)
```

If we can prove that the given schedule is View Equivalent to its serial schedule then the given schedule is called view Serializable.

Why we need View Serializability?

We know that a serial schedule never leaves the database in inconsistent state because there are no concurrent transactions execution. However a non-serial schedule can leave the database in inconsistent state because there are multiple transactions running concurrently. By checking that a given non-serial schedule is view serializable, we make sure that it is a consistent schedule.

You may be wondering instead of checking that a non-serial schedule is serializable or not, can't we have serial schedule all the time? The answer is no, because concurrent execution of transactions fully utilize the system resources and are considerably faster compared to serial schedules.

View Equivalent

Lets learn how to check whether the two schedules are view equivalent.

Two schedules T1 and T2 are said to be view equivalent, if they satisfy all the following conditions:

1. Initial Read: Initial read of each data item in transactions must match in both schedules. For example, if transaction T1 reads a data item X before transaction T2 in schedule S1 then in schedule S2, T1 should read X before T2.

Read vs Initial Read: You may be confused by the term initial read. Here initial read means the first read operation on a data item, for example, a data item X can be read multiple times in a schedule but the first read operation on X is

called the initial read. This will be more clear once we will get to the example in the next section of this same article.

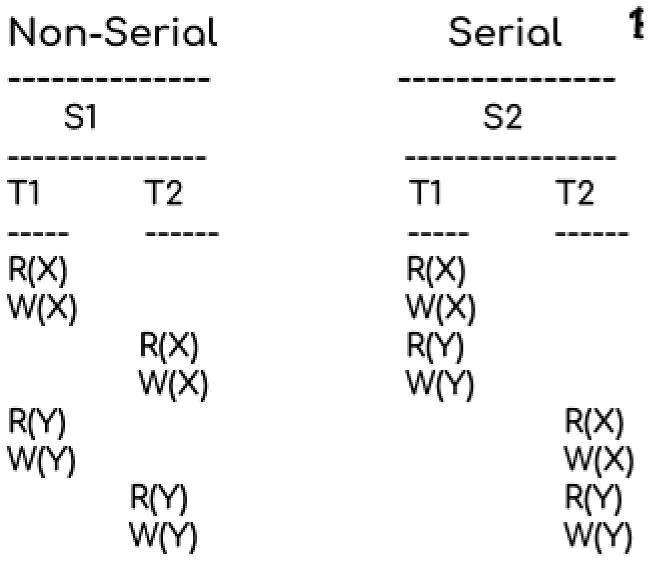
2. Final Write: Final write operations on each data item must match in both the schedules. For example, a data item X is last written by Transaction T1 in schedule S1 then in S2, the last write operation on X should be performed by the transaction T1.

3. Update Read: If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on same data item. For example, In schedule S1, T1 performs a read operation on X after the write operation on X by T2 then in S2, T1 should read the X after T2 performs write on X.

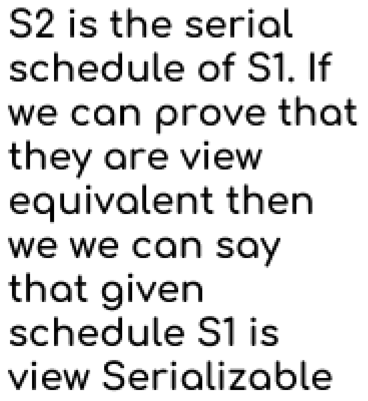
View Serializable

If a schedule is view equivalent to its serial schedule then the given schedule is said to be View Serializable. Lets take an example.

View Serializable Example



```
Non-Serial                 Serial
---------------         ----------------
    S1                        S2
----------------        -----------------
T1         T2           T1          T2
-----      ------       -----       ------
R(X)                    R(X)
W(X)                    W(X)
           R(X)         R(Y)
           W(X)         W(Y)
R(Y)                                R(X)
W(Y)                                W(X)
          R(Y)                      R(Y)
          W(Y)                      W(Y)
```



S2 is the serial schedule of S1. If we can prove that they are view equivalent then we we can say that given schedule S1 is view Serializable

Lets check the three conditions of view serializability:

Initial Read

In schedule S1, transaction T1 first reads the data item X. In S2 also transaction T1 first reads the data item X.

Lets check for Y. In schedule S1, transaction T1 first reads the data item Y. In S2 also the first read operation on Y is performed by T1.

We checked for both data items X & Y and the initial read condition is satisfied in S1 & S2.

Final Write

In schedule S1, the final write operation on X is done by transaction T2. In S2 also transaction T2 performs the final write on X.

Lets check for Y. In schedule S1, the final write operation on Y is done by transaction T2. In schedule S2, final write on Y is done by T2.

We checked for both data items X & Y and the final write condition is satisfied in S1 & S2.

Update Read

In S1, transaction T2 reads the value of X, written by T1. In S2, the same transaction T2 reads the X after it is written by T1.

In S1, transaction T2 reads the value of Y, written by T1. In S2, the same transaction T2 reads the value of Y after it is updated by T1.

The update read condition is also satisfied for both the schedules.

Result: Since all the three conditions that checks whether the two schedules are view equivalent are satisfied in this example, which means S1 and S2 are view equivalent. Also, as we know that the schedule S2 is the serial schedule of S1, thus we can say that the schedule S1 is view serializable schedule.

Concurrency Control in DBMS

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

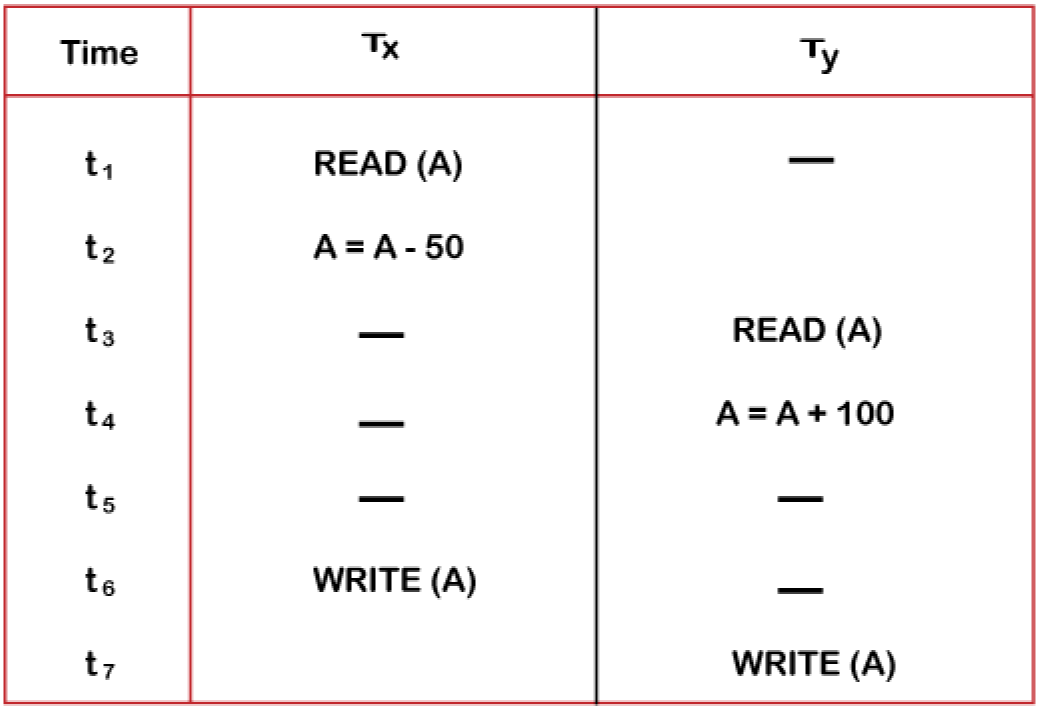
In a database transaction, the two main operations are READ and WRITE operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following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

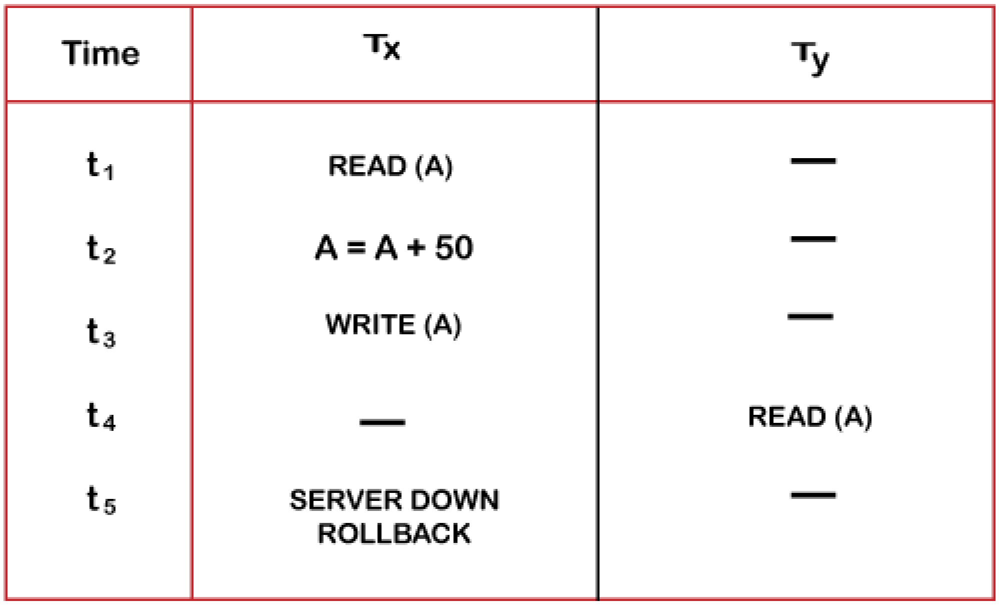
Hence data becomes incorrect, and database sets to inconsistent.

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 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.**

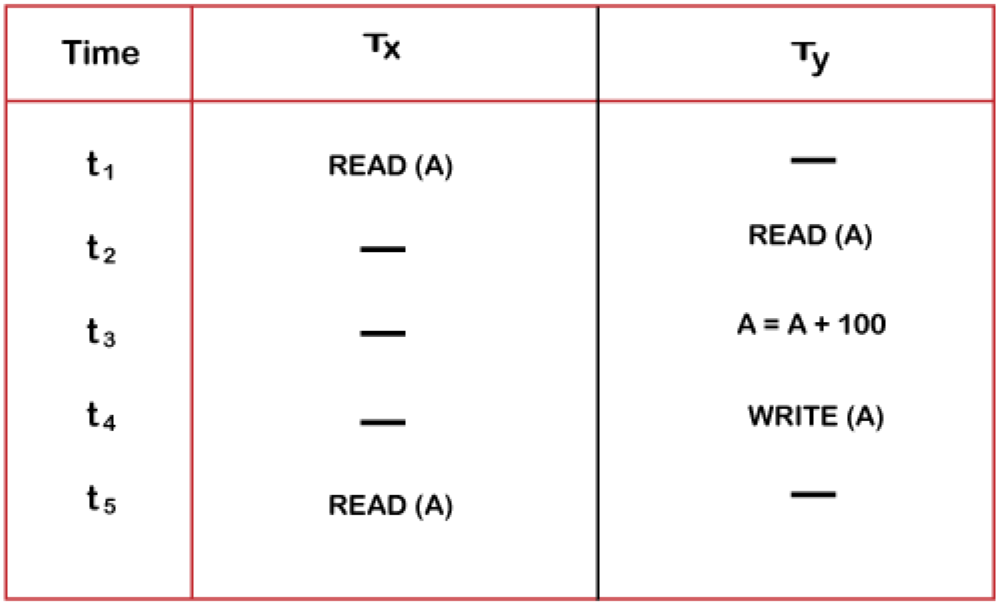
- **Then at time t4, transaction T_Y reads account A that will be read as $350.**
- **Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).**
- **But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.**

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 t1, transaction T_X reads the value from account A, i.e., $300.**

- **At time t2, transaction T_Y reads the value from account A, i.e., $300.**
- **At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.**
- **At time t4, transaction T_Y writes the updated value, i.e., $400.**
- **After that, at time t5, 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**

- **Timestamp 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.**

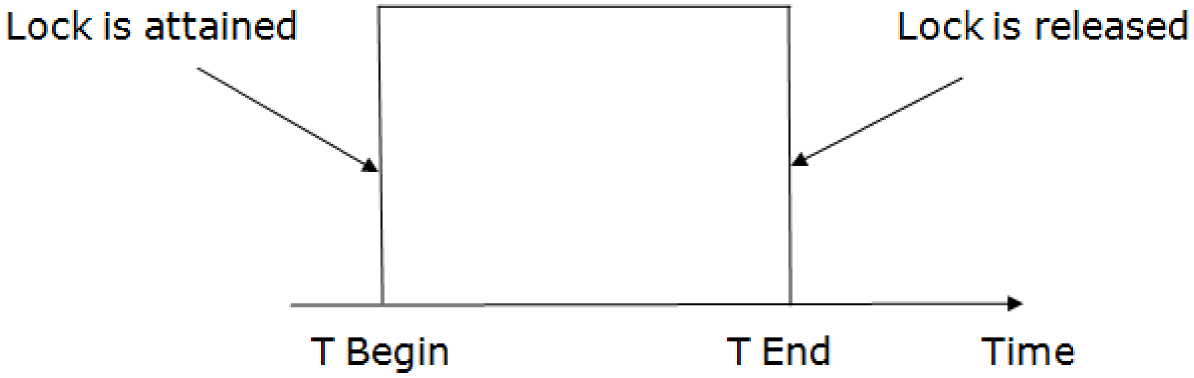
There are four types of lock protocols available:

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

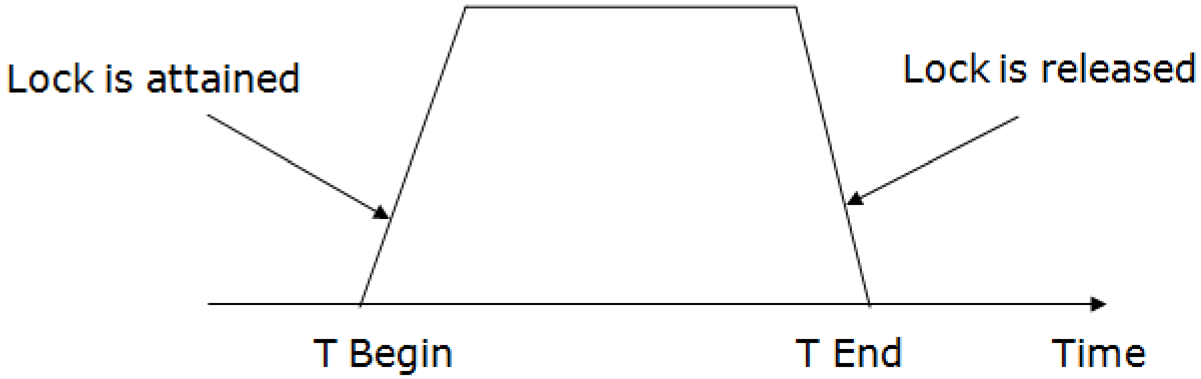
- **Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.**
- **Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.**
- **If all the locks are granted then this protocol allows the transaction 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:

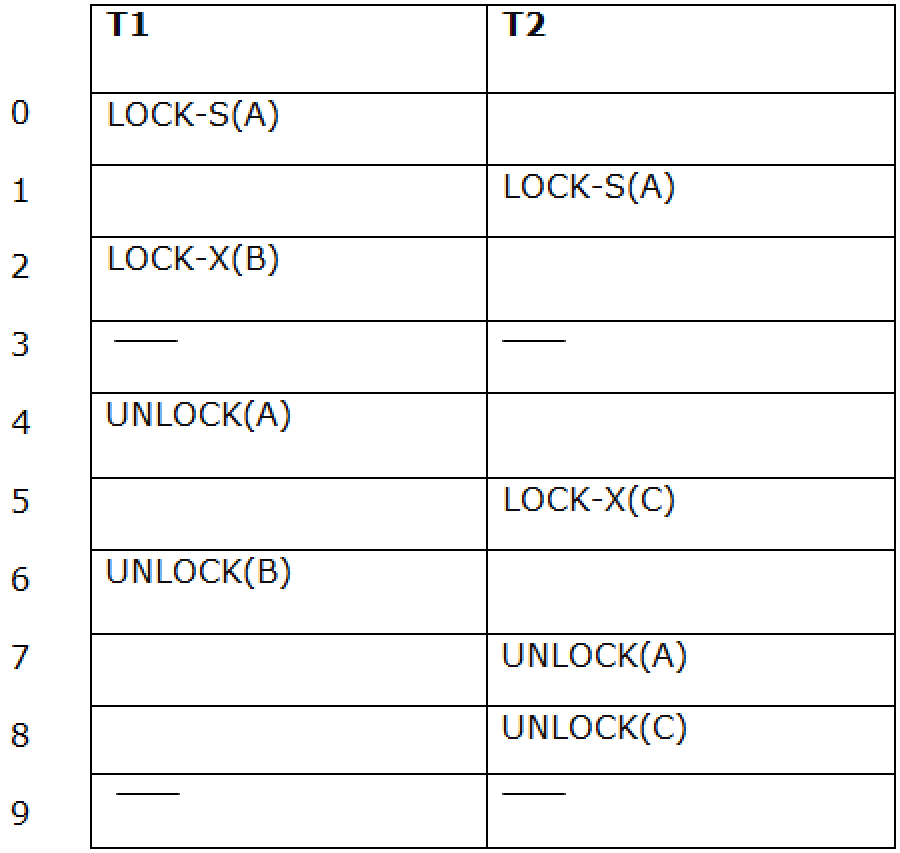
Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing locks 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 the growing phase.**
2. **Downgrading of lock (from X(a) to S(a)) must be done in a 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:

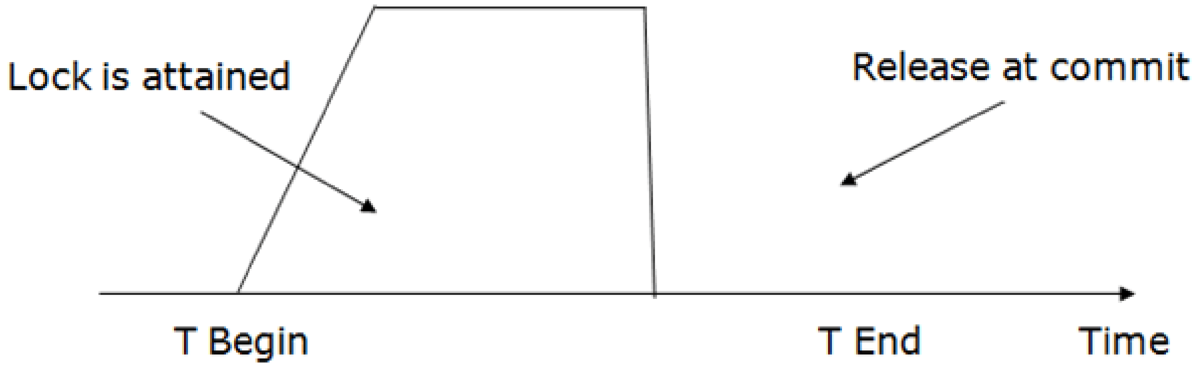
- **Growing phase: from step 1-3**
- **Shrinking phase: from step 5-7**
- **Lock point: at 3**

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 a shrinking phase of lock release.**



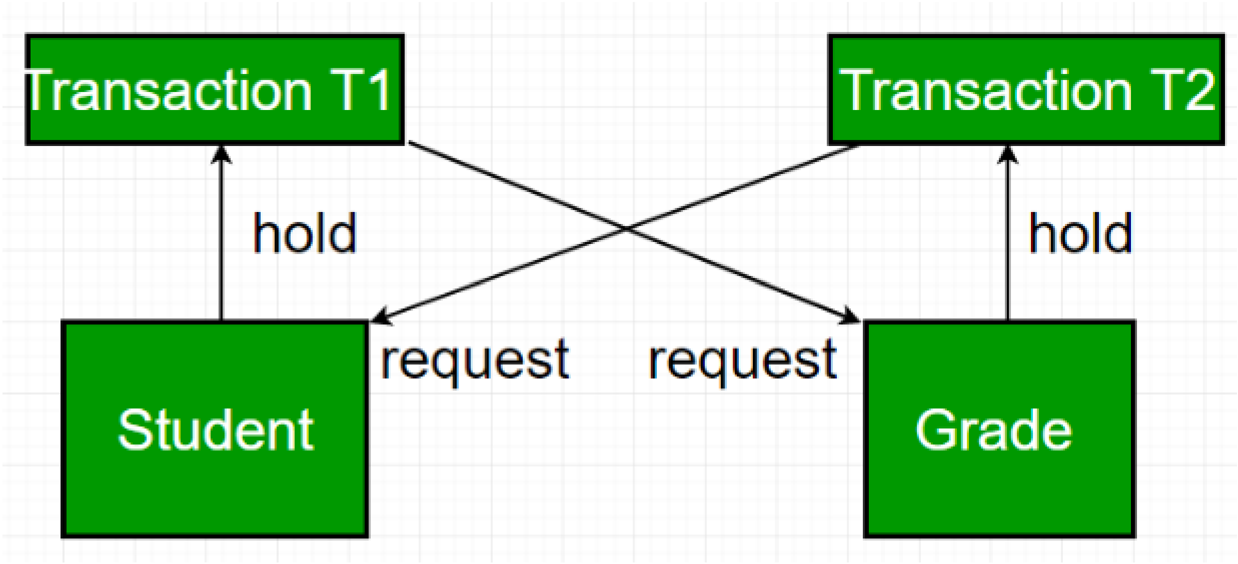
Deadlock in DBMS

In a database, a deadlock is an unwanted situation in which 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 it brings the whole system to a Halt.

Example – let us understand the concept of Deadlock with an example :

Suppose, Transaction T1 holds a lock on some rows in the Students table and needs to update some rows in the Grades table. Simultaneously, Transaction T2 holds locks on those very rows (Which T1 needs to update) in the Grades table but needs to update the rows in the Student table held by Transaction T1.

Now, the main problem arises. Transaction T1 will wait for transaction T2 to give up the lock, and similarly, transaction T2 will wait for transaction T1 to give up the lock. As a consequence, All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.



Deadlock Avoidance –

When a database is stuck in a deadlock, It is always better to avoid the deadlock rather than restarting or aborting the database. The deadlock avoidance method is suitable for smaller databases whereas the deadlock prevention method is suitable for larger databases.

One method of avoiding deadlock is using application-consistent logic. In the above-given example, Transactions that access Students and Grades should always access the tables in the same order. In this way, in the scenario described above, Transaction T1 simply waits for transaction T2 to release the lock on Grades before it begins. When transaction T2 releases the lock, Transaction T1 can proceed freely.

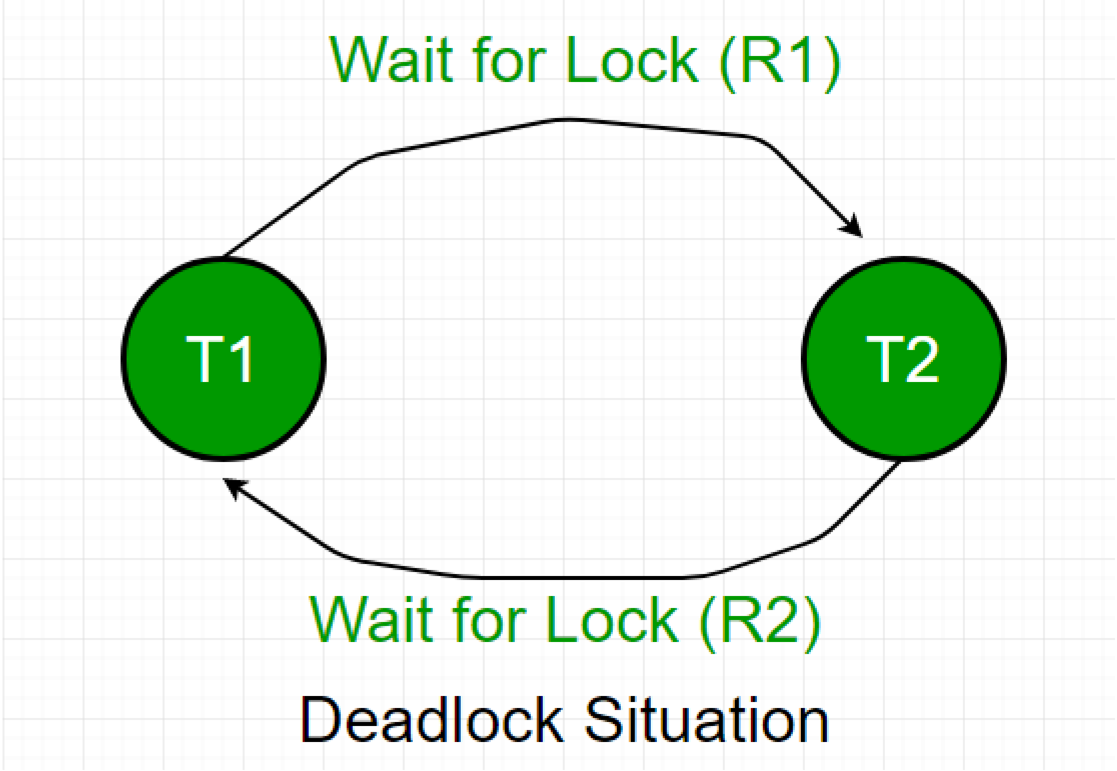
Another method for avoiding deadlock is to apply both row-level locking mechanism and READ COMMITTED isolation level. However, It does not guarantee to remove deadlocks completely.

Deadlock Detection –

When a transaction waits indefinitely to obtain a lock, The database management system should detect whether the transaction is involved in a deadlock or not.

Wait-for-graph is one of the methods for detecting the deadlock situation. This method is suitable for smaller databases. In this method, a graph is drawn based on the transaction and their lock on the resource. If the graph created has a closed-loop or a cycle, then there is a deadlock.

For the above-mentioned scenario, the Wait-For graph is drawn below



Deadlock prevention –

For a large database, the deadlock prevention method is suitable. A deadlock can be prevented if the resources are allocated in such a way that deadlock never occurs. The DBMS analyzes the operations whether they can create a deadlock situation or not, If they do, that transaction is never allowed to be executed.

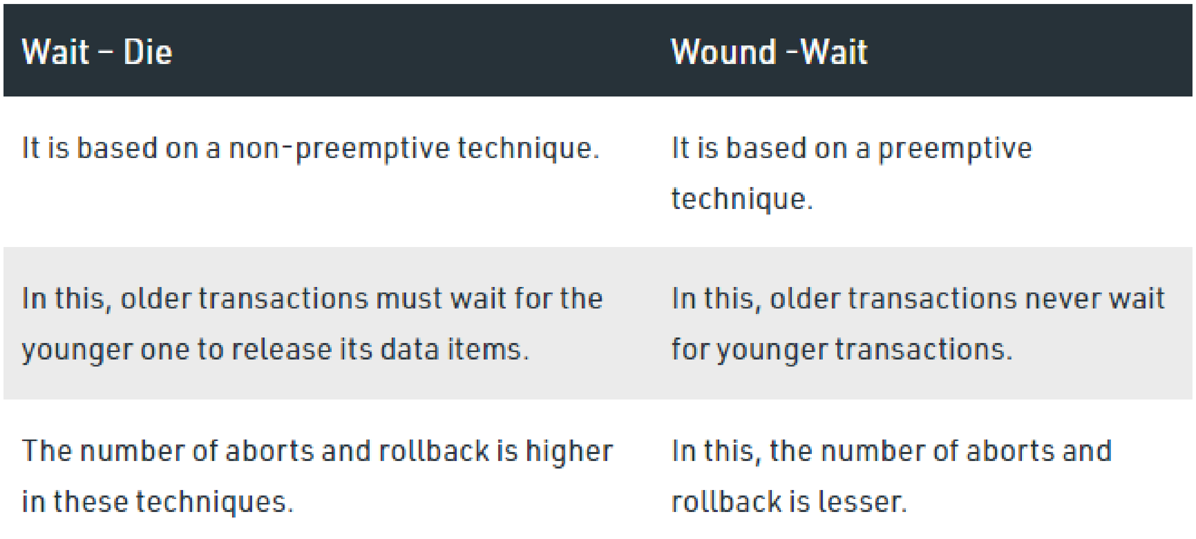
Deadlock prevention mechanism proposes two schemes :

- **Wait-Die Scheme –**
  In this scheme, If a transaction requests a resource that is locked by another transaction, then the DBMS simply checks the timestamp of both transactions and allows the older transaction to wait until the resource is available for execution.

Suppose, there are two transactions T1 and T2, and Let the timestamp of any transaction T be TS (T). Now, If there is a lock on T2 by some other transaction and T1 is requesting for resources held by T2, then DBMS performs the following actions:
Checks if TS (T1) < TS (T2) – if T1 is the older transaction and T2 has held some resource, then it allows T1 to wait until resource is available for execution. That means if a younger transaction has locked some resource and an older transaction is waiting for it, then an older transaction is allowed to wait for it till it is available.
If T1 is an older transaction and has held some resource with it and if T2 is waiting for it, then T2 is killed and restarted later with random delay but with the same timestamp. i.e. if the older transaction has held some resource and the younger transaction waits for the resource, then the younger transaction is killed and restarted with a very minute delay with the same timestamp.
This scheme allows the older transaction to wait but kills the younger one.

- **Wound Wait Scheme –**
  In this scheme, if an older transaction requests for a resource held by a younger transaction, then an older transaction forces a younger transaction to kill the transaction and release the resource. The younger transaction is restarted with a minute delay but with the same timestamp. If the younger transaction is requesting a resource that is held by an older one, then the younger transaction is asked to wait till the older one releases it.

Following table list the differences between Wait – Die and Wound -Wait scheme prevention schemes :



Wait – Die	Wound -Wait
It is based on a non-preemptive technique.	It is based on a preemptive technique.
In this, older transactions must wait for the younger one to release its data items.	In this, older transactions never wait for younger transactions.
The number of aborts and rollback is higher in these techniques.	In this, the number of aborts and rollback is lesser.

Database Recovery Techniques in DBMS

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.**
    1. **<Tn, Start>**
- **When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.**
    1. **<Tn, City, 'Noida', 'Bangalore' >**
- **When the transaction is finished, then it writes another log to indicate the end of the transaction.**
    1. **<Tn, Commit>**

There are two approaches to modify the database:

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.**