Chapter 10. Transaction Processing and Concurrency Control

Until now we have learnt to create a database. This chapter focuses on ways to maintain a database.

What is Transaction?

- The transaction is a set of logically related operation. It contains a group of tasks.
- In other words, A transaction is a sequence of one or more SQL statements that Database treats as a Unit - either all of the statements are performed, or none of them are. We need transactions to model business processes that require that several operations be performed as a unit.

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
Following are the main operations of transaction:
Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.
Write(X): Write operation is used to write the value back to the database from the buffer.

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

1. R(X)

2. X = X - 500
3. W(X);

Let's assume the value of X before starting of the transaction is 4000.
- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

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

For example: If in the above transaction, the debt 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.

Additional Operations:
Savepoint: It marks a savepoint in a transaction - a point to which we can later roll back. Savepoints are optional, and a transaction can have multiple savepoints.
End: It indicates the ending of the transaction.

ACID Properties of Transaction:
The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

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 form account A to account B.

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

Example:
The Total amount must be maintained before or after the transaction.
Total before T occurs = 600+300 = 900
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 complete.

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 faculty transaction or by the system failure.

When a transaction is complete, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a systems failure. The recovery subsystem of the DBMS has the responsibility of Durability property.

States of Transaction:

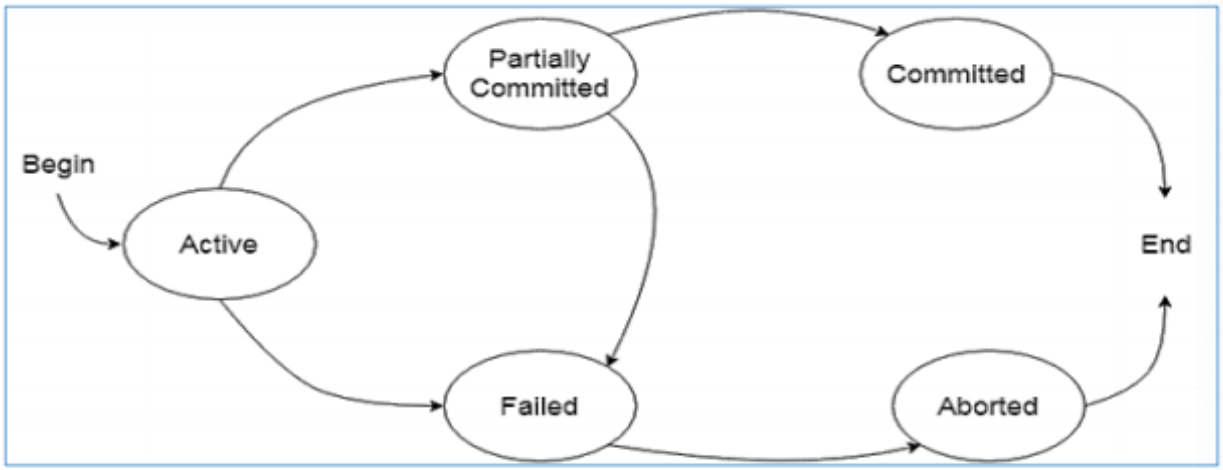


Fig: States of Transaction

In a database, the transaction can be in one of the following 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 date 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 marks 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 transaction are rolled back to its consistent state. After aborting the transaction, the database recovery module will select one of the two operations:

a. Re-start the transaction
b. Kill the transaction

TRANSACTION SCHEDULE:

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

1. Serial Schedule:

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

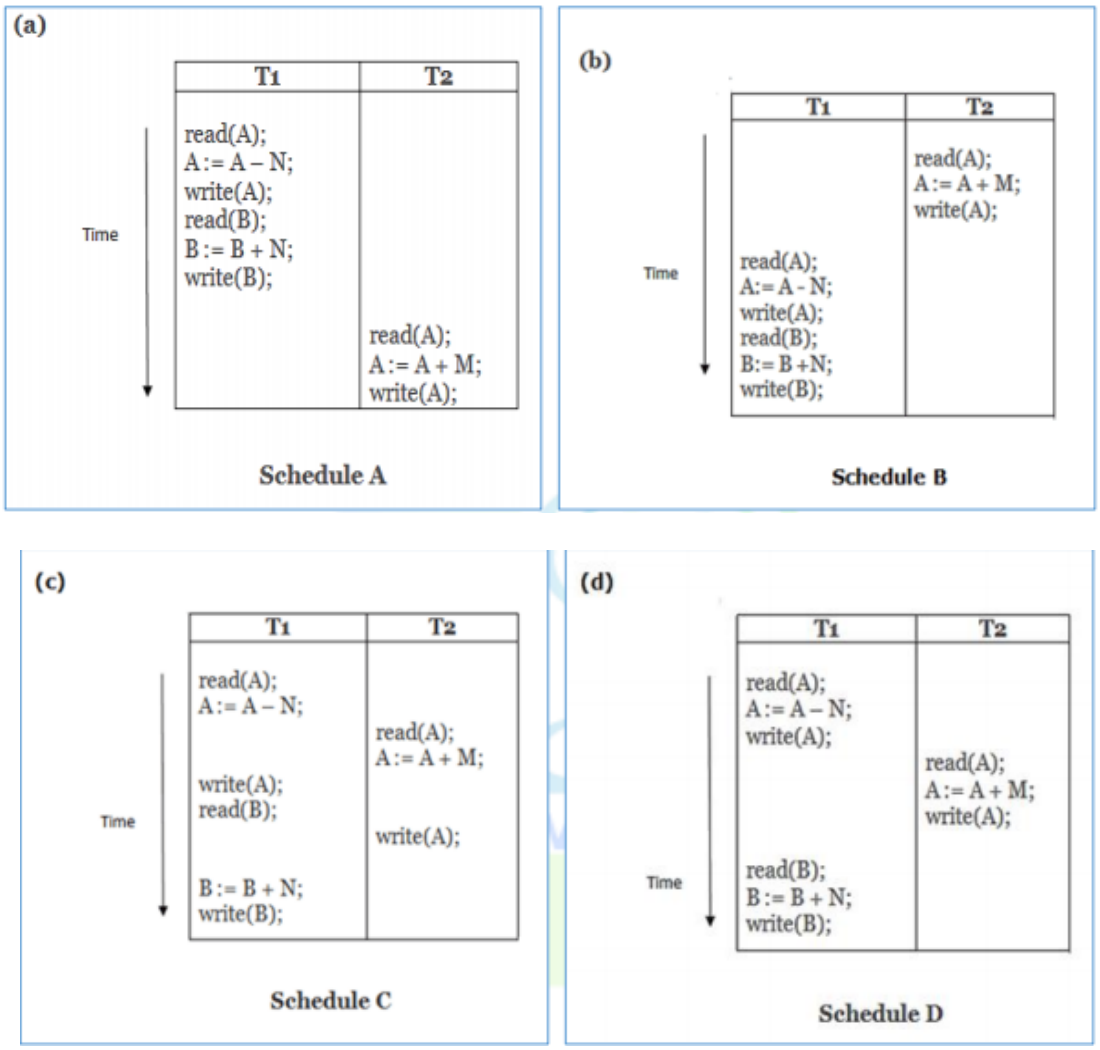
Example:

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:

a. Execute all operations of T1 which was followed by all the operations of T2.
b. Execute all the operations of T2 which was followed by all the operations of T1.

2. Non- Serial Schedule:

If interleaving of operations is allowed, then there will be non-serial schedule. It contains many possible orders in which the system can execute the individual operations of the transactions.



Here,

Schedule A and Schedule B are serial schedule

Schedule C and Schedule D are Non-serial schedule

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 execution of the transaction have interleaving go their operations. A non-serial schedule will be serializable if its result is equal to the result of tis transactions executed serially.

A serializable schedule always leaves the database in consistent state. A Serial Schedule is always a serializable schedule because in serial schedule, a transaction only starts when the other transaction finished 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 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 finished

Types of Serializability:
There are two types of Serializability

1. Conflict Serializability
2. View Serializability

CONFLICT SERIALIZABILITY

- 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.

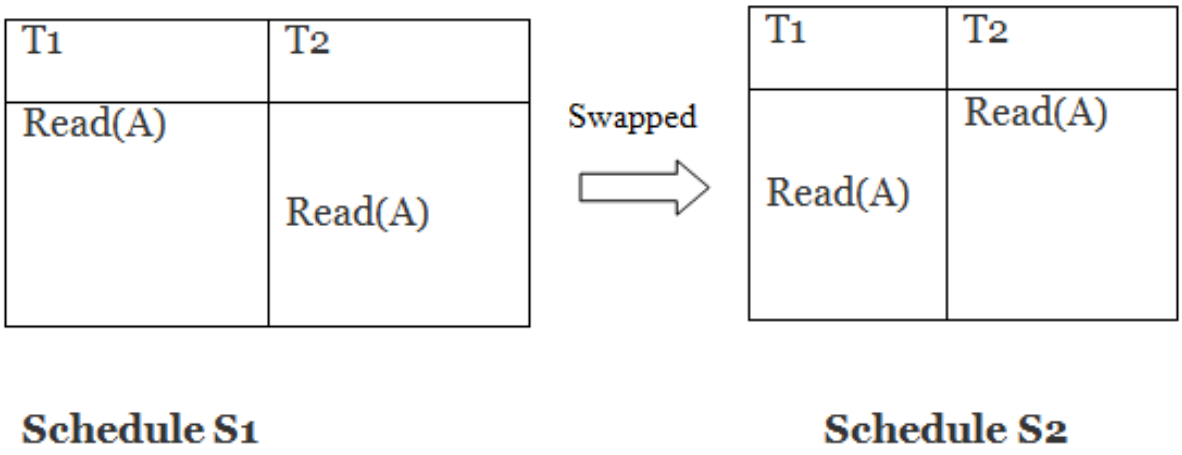
Conflicting Operations
The two operations become conflicting if all conditions satisfy:

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

Example:

Swapping is possible only if S1 and S2 are logically equal.

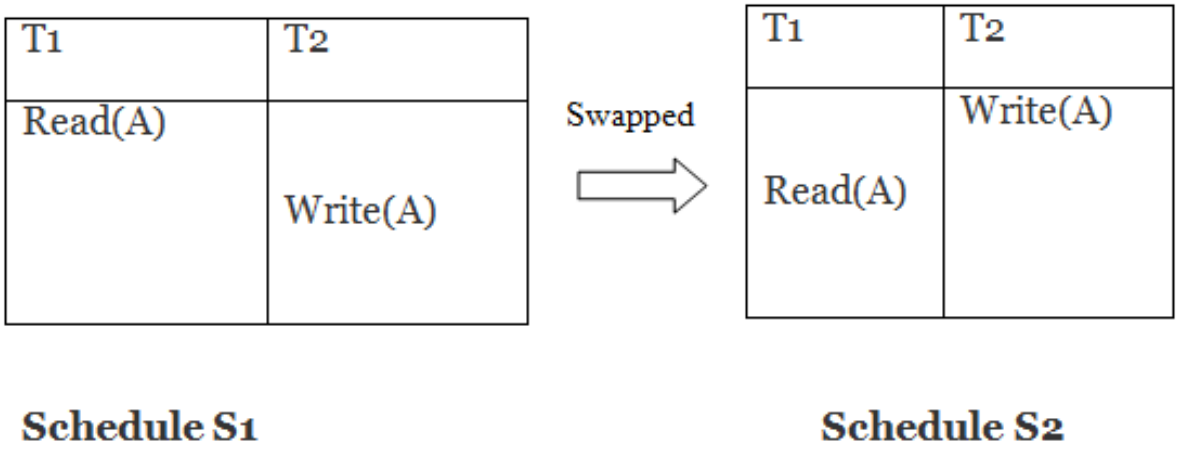
1. T1: Read(A) T2: Read(A)



Schedule S1 **Schedule S2**

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

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

Since, S1 is conflict serializable.

View Serializability

- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

1. **Initial Read:** An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	Write(A)

Schedule S1

T1	T2
Read(A)	Write(A)

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Update 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
Write(A)	Read(A)	Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

Example:

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

Schedule S

Taking first schedule S1:

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

Schedule S1

Step 1: final updating on data items

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

Step 2: Initial Read

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

Step 3: Final Write

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

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

Hence, view equivalent serial schedule is:

T1 -> T2 -> T3

CONCURRENCY CONTROL

It is the process of managing simultaneous execution of transactions in a shared database, to ensure the Serializability of transactions.

Purpose of Concurrency Control

i) To enforce isolation

ii) To preserve database consistency

iii) To resolve read-write and write-write conflicts

Concurrency Control Techniques:

1. Lock-Based Protocol:A lock grantees exclusive use of a data item to a current transaction.

a. To Access Data Item (Lock Acquired)

b. After completion of transaction (Release the lock)

Condition: It requires that all data items must be accessed in a mutually exclusive manner.

Types of Locks

a. **Shared Lock** : It is denoted by Lock-S. It is acquired when you want to read the data item value.

b. **Exclusive Lock:** It is denoted by Lock-X. It is used for both read and write

Compatibility between lock modes

	s	x
s	T	F
x	F	F

Eg of Lock Protocol

T1	T2
Example of Exclusive Lock **Lock -X(B)** **R(B)** **B-50** **W(B)** **Unlock(B)**	
	Example of Shared Lock Lock-S(B) R(B) Unlock(B)

Note: Any no of transactions can hold shared lock on an item but exclusive lock can be hold only by one transaction at a time.

Two-Phase Locking Protocol (2PL):

Two Phase Locking Protocol also known as 2PL protocol is a method of concurrency control in DBMS that ensures serializability by applying a lock to the transaction data which blocks other transactions to access the same data simultaneously. Two Phase Locking protocol helps to eliminate the concurrency problem in DBMS.

This locking protocol divides the execution phase of a transaction into three different parts.

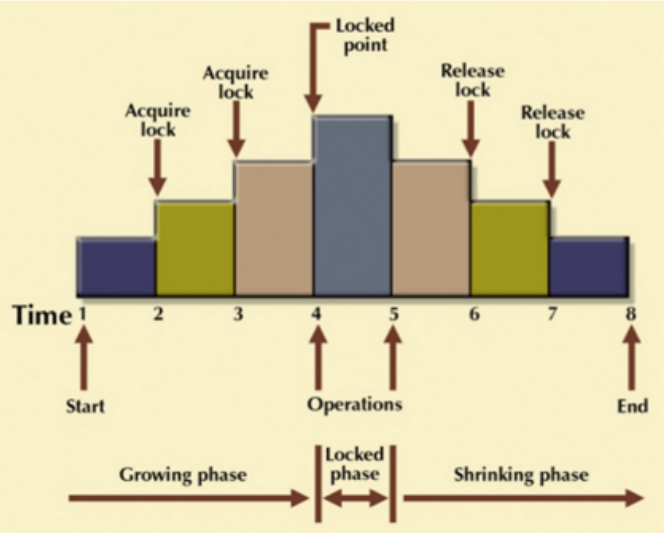
- In the first phase, when the transaction begins to execute, it requires permission for the locks it needs.
- The second part is where the transaction obtains all the locks. When a transaction releases its first lock, the third phase starts.
- In this third phase, the transaction cannot demand any new locks. Instead, it only releases the acquired locks.

Requires both locks and unlocks being done in two phases.

Phases

1. Growing(Expanding) Phase : New Locks on items can be acquired but cannot release any lock.

2. Shrinking Phase: release existing locks, but no new locks can be acquired.



Timestamp-based Protocols

Timestamp based Protocol in DBMS is an algorithm which uses the System Time or Logical Counter as a timestamp to serialize the execution of concurrent transactions. The Timestamp-based protocol ensures that every conflicting read and write operations are executed in a timestamp order.

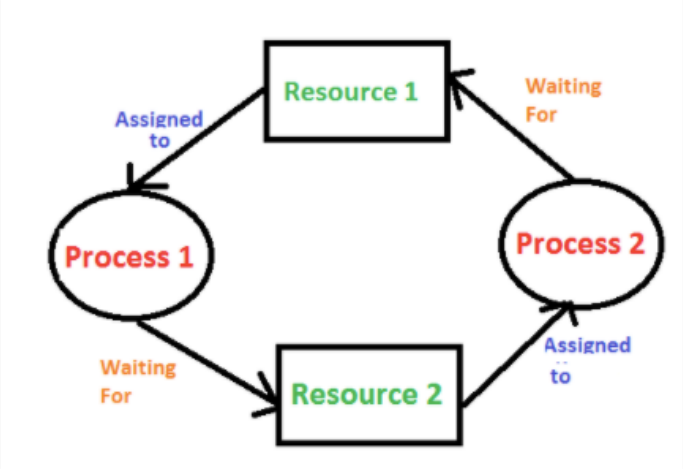
The older transaction is always given priority in this method. It uses system time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol.

Lock-based protocols help you to manage the order between the conflicting transactions when they will execute. Timestamp-based protocols manage conflicts as soon as an operation is created.

DeadLocks

In a database, a deadlock is a situation in which two or more transactions are waiting for one another to give up locks.

Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other. A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Deadlock can arise if the **following four conditions hold simultaneously (Necessary Conditions)**
Mutual Exclusion: One or more than one resource are non-shareable (Only one process can use at a time)
Hold and Wait: A process is holding at least one resource and waiting for resources.
No Preemption: A resource cannot be taken from a process unless the process releases the resource.
Circular Wait: A set of processes are waiting for each other in circular form.

Methods for handling deadlock
There are three ways to handle deadlock
1) Deadlock prevention or avoidance: The idea is to not let the system into a deadlock state.
One can zoom into each category individually, Prevention is done by negating one of above mentioned necessary conditions for deadlock.
Avoidance is kind of futuristic in nature. By using strategy of “Avoidance”, we have to make an assumption. We need to ensure that all information about resources which process will need are known to us prior to execution of the process. We use Banker’s algorithm (Which is in-turn a gift from Dijkstra) in order to avoid deadlock.

2) Deadlock detection and recovery: Let deadlock occur, then do preemption to handle it once occurred.

3) Ignore the problem altogether: If deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.