Unit-4 DBMS

Concurrency control

When multiple transactions are running simultaneously then there are chances of a conflict to occur which can leave database to an inconsistent state. To handle these conflicts we need concurrency control in DBMS, which allows transactions to run simultaneously but handles them in such a way so that the integrity of data remains intact.

Concurrency Control in Database Management System is a procedure of managing simultaneous operations without conflicting with each other.

Concurrent access is quite easy if all users are just reading data. But database have a mix of READ and WRITE operations and hence the concurrency is a challenge.

DBMS Concurrency Control is used to address such conflicts, which mostly occur with a multi-user system. Therefore, Concurrency Control is the most important element for proper functioning of a Database Management System where two or more database transactions are executed simultaneously, which require access to the same data.

Concurrency Control Techniques

Following are the Concurrency Control techniques in DBMS:

- Lock Techniques for concurrency control
- Time stamping protocols for concurrency control
- Validation-Based Protocols

1. Lock Techniques for concurrency control

Lock Based Protocols in DBMS is a mechanism in which a transaction cannot Read or Write the data until it acquires an appropriate lock.

Locks are of two kinds –

Shared/Exclusive Locking Protocol: This type of locking mechanism separates the locks in DBMS based on their uses. If a lock is acquired on a data item to perform a read-only operation, it is called a shared lock and if a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

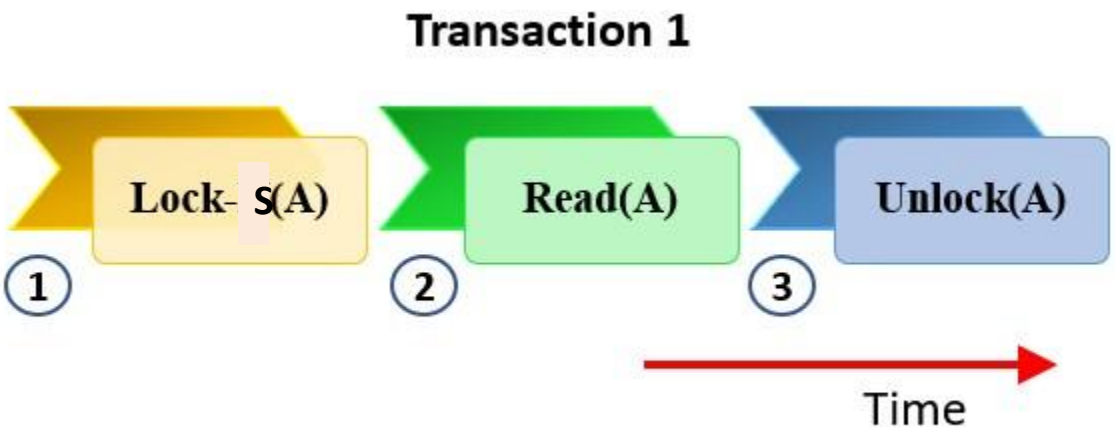
1. Shared Lock (S):

A shared lock is also called a Read-only lock. With the shared lock, the data item can be shared between transactions. This is because we will never have permission to update data on the data item.

For example, consider a case where two transactions are reading the account balance of a person. The database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

Consider a transaction (T1) which requires only reading the data item value A. The following steps take place when lock protocol is applied to this traction

1. T1 will acquire an shared lock on the data item A
2. Read the current value of data item A
3. Once the transaction is completed, the data item will be unlocked.



2. Exclusive Lock (X):

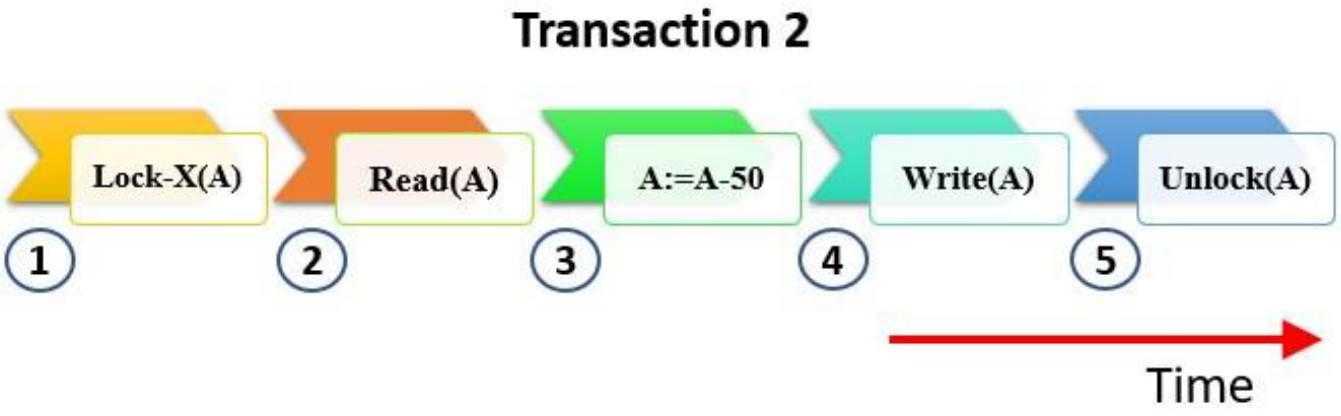
With the Exclusive Lock, a data item can be read as well as written. This is exclusive and can't be held concurrently on the same data item. X-lock is requested using lock-x instruction. Transactions may unlock the data item after finishing the 'write' operation.

For example, when a transaction needs to update the account balance of a person. We can allow this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write, exclusive lock prevent this operation.

Consider a transaction (T2) which requires updating the data item value A. The following steps take place when lock protocol is applied to this traction

1. T2 will acquire an exclusive lock on the data item A
2. Read the current value of data item A
3. Modify the data item as required. In the example illustrated, a value of 50 is subtracted from the data item A

4. Write the updated value of the data item
5. Once the transaction is completed, the data item will be unlocked.



Lock Compatibility Matrix –

Grant ↓ / Request →	S	X
S	✔	✗
X	✗	✗

The figure illustrates that when two transactions are involved, and both attempt to only read a given data item, it is permitted and no conflict arises, but when one transaction attempts to write the data item and another one tries to read or write at the same time, conflict occurs resulting in a denied interaction.

Problems associated with simple locking:

Deadlock

Deadlock refers to a specific situation where two or more processes are waiting for each other to release a resource or more than two processes are waiting for the resource in a circular chain.

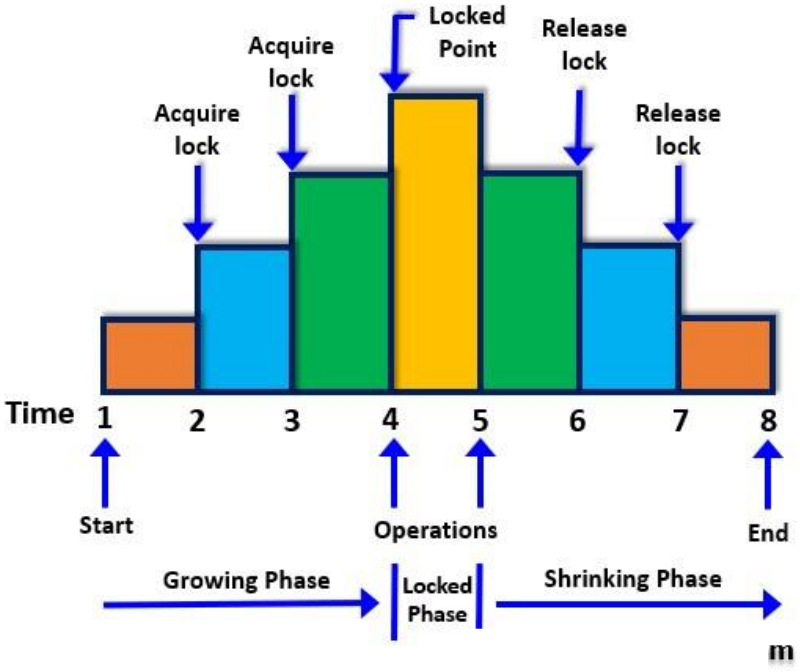
Starvation

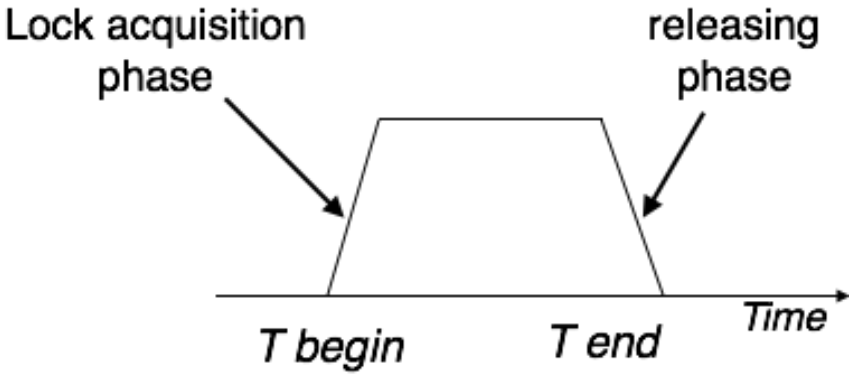
Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.

Two Phase Locking Protocol

Two Phase Locking Protocol also known as 2PL protocol is a method of concurrency control. This locking protocol divides the execution phase of a 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 part, the transaction cannot demand any new locks. It only releases the acquired locks.



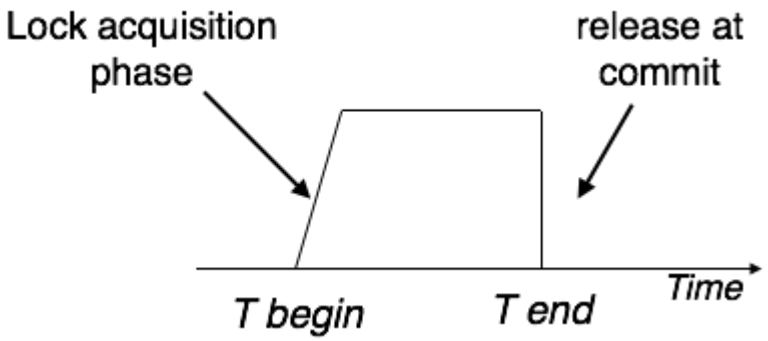


- **Growing Phase**: In this phase transaction may obtain locks but may not release any locks.
- **Shrinking Phase**: In this phase, a transaction may release locks but not obtain any new lock

Strict Two-Phase Locking Method

Strict-Two phase locking system is almost similar to 2PL. The only difference is that Strict-2PL never releases a lock after using it. It holds all the locks until the commit point and releases all the locks at one go when the process is over.

- 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.



2. Time stamping protocols for concurrency control

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp. A timestamp is a tag (unique identifier) assign to every transaction.

The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.

- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.

- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Example:

Suppose there are three transactions T1, T2, and T3.

T1 has entered the system at time 0010

T2 has entered the system at 0020

T3 has entered the system at 0030

Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.

Advantages:

Image: Precedence Graph for TS ordering

- Schedules are serializable just like 2PL protocols
- No waiting for the transaction, which eliminates the possibility of deadlocks.

We need following timestamp for this technique

- The timestamp of transaction Ti is denoted as TS(Ti).-It is the time when transaction entered in the system.
- Read time-stamp of data-item X is denoted by R-TS(X)- It is the largest timestamp of any transaction that executed **read(X)** successfully.
- Write time-stamp of data-item X is denoted by W-TS(X)- It is the largest timestamp of any transaction that executed **write(X)** successfully.

Example

Time	11.00	11.10	11.20
Timestamp	1100	1110	1120
Transaction	T1	T2	T3
	R(A)		
	W(A)		
		R(A)	
			R(A)
			W(A)
		W(A)	

TS(T1) = 1100

TS(T2) = 1110

TS(T3) = 1120

R-TS(A) = 1120

W-TS(A) =1110

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a **Read (X)** operation:
   - If W_TS(X) >TS(Ti) then the operation is rejected.
   - If W_TS(X) <= TS(Ti) then the operation is executed.
   - Timestamps of all the data items are updated.
2. Check the following condition whenever a transaction Ti issues a **Write(X)** operation:
   - If TS(Ti) < R_TS(X) or if TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Example-Allowed cases

Timestamp	100	200
Transaction	T1	T2
	R(A)	
		W(A)

Timestamp	100	200
Transaction	T1	T2
	W(A)	
		R(A)

Timestamp	100	200
Transaction	T1	T2
	W(A)	
		W(A)

Example- Not Allowed cases

Timestamp	100	200
Transaction	T1	T2
		R(A)
	W(A)	

Timestamp	100	200
Transaction	T1	T2
		W(A)
	R(A)	

Timestamp	100	200
Transaction	T1	T2
		W(A)
	W(A)	

Validation Based Protocol

Validation Based Protocol is also called **Optimistic Concurrency Control Technique**. This protocol is used in DBMS (Database Management System) for avoiding concurrency in transactions. It is called optimistic because of the assumption it makes, i.e. very less interference occurs. It is a type of concurrency control techniques that works on the validation rules and time-stamps.

In this technique, no checking is done while the transaction is been executed. Until the transaction end is reached updates in the transaction are not applied directly to the database. All updates are applied to local copies of data items kept for the transaction. At the end of transaction execution, while execution of the transaction, a **validation phase** checks whether any of transaction updates violate serializability. If there is no violation of serializability the transaction is committed and the database is updated; or else, the transaction is updated and then restarted.

The Validation based Protocol is performed in the following three phases:

1. Read Phase
2. Validation Phase
3. Write Phase

Read Phase

In the Read Phase, the data values from the database can be read by a transaction but the write operation or updates are only applied to the local data copies, not the actual database.

Validation Phase

In Validation Phase, the data is checked to ensure that there is no violation of serializability while applying the transaction updates to the database.

Write Phase

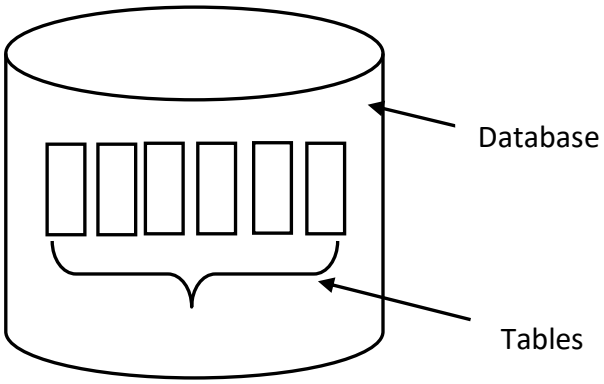
In the Write Phase, the updates are applied to the database if the validation is successful, else; the updates are not applied, and the transaction is rolled back.

There are three timestamps that control the serializability of the validation based protocol in the database management system, such as.

- **Start(Timestamp)**: The start timestamp is the initial timestamp when the data item being read and executed in the read phase of the validation protocol.
- **Validation(Timestamp):** The validation timestamp is the timestamp when T1 completed the read phase and started with the validation phase.
- **Finish(Timestamp):** The finish timestamp is the timestamp when T1 completes the writing process in the writing phase.

Granularity

Concurrency Control Techniques used different methods for serializability. A certain drawback of this technique is if a transaction Ti needs to access the entire database and a locking protocol is used, then Ti must lock each item in the database. Suppose another transaction just needs to access a few data items from a database, so locking the entire database seems to be unnecessary also it may cost us loss of Concurrency, which was our primary goal. To solve this problem between Efficiency and Concurrency we use Granularity.



***Granularity* – It is the size of data item allowed to lock.**

In general term we can say granularity is directly concern with hierarchy of level.

Multiple Granularity

- It can be defined as hierarchically breaking up the database into blocks which can be locked.
- The Multiple Granularity protocol enhances concurrency and reduces lock overhead.
- It maintains the track of what to lock and how to lock.
- It makes easy to decide either to lock a data item or to unlock a data item. This type of hierarchy can be graphically represented as a tree.

For example: Consider a tree which has four levels of nodes.

- The first level or higher level shows the entire database.
- The second level represents a node of files or tables.
- The file/table consists of children nodes which are known as Records.
- Finally, each record contains child nodes known as Fields.
- Hence, the levels of the tree starting from the top level are as follows:
    1. Database
    2. Files/Tables
    3. Records
    4. Fields

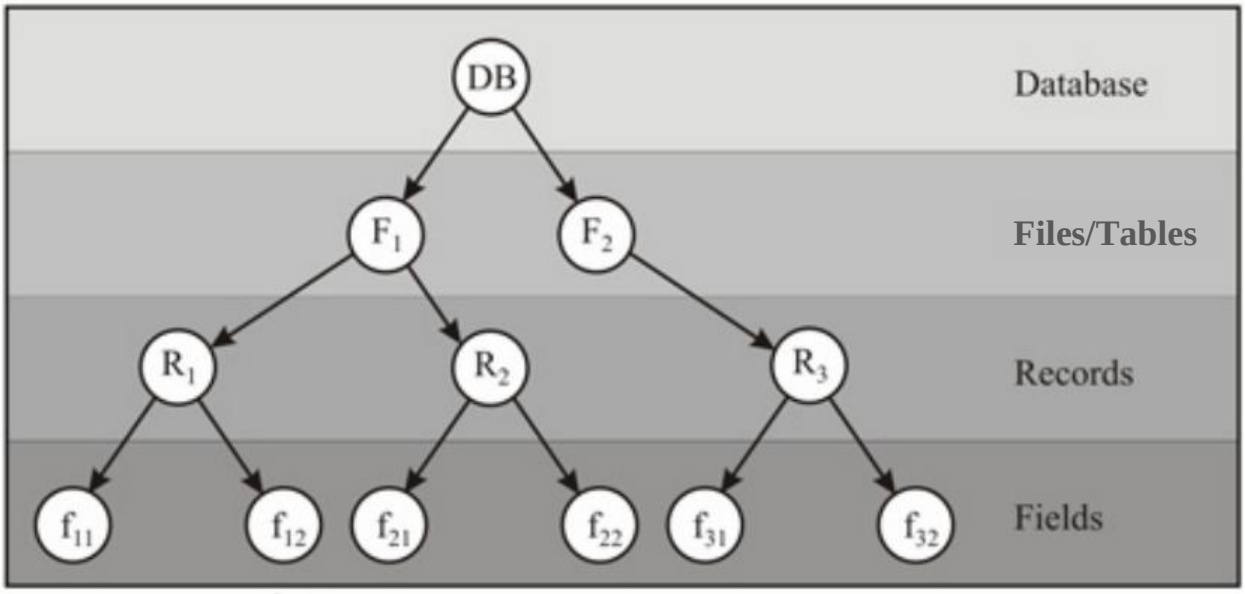


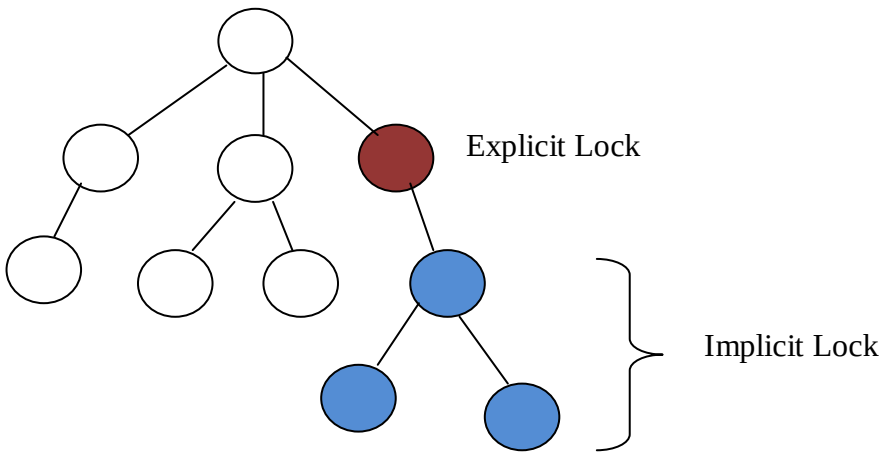
Figure – Multi Granularity tree Hierarchy

Consider the above diagram for the example, each node in the tree can be locked individually. As in the 2PL, it shall use shared and exclusive lock modes. When a transaction locks a node, in either shared or exclusive mode, the transaction also implicitly locks all the descendants of that node in the same lock mode. For example, if transaction T_i gets an explicit lock on file F_2 in exclusive mode, then it has an implicit lock in exclusive mode on all the records belonging to that file. It does not need to lock the individual records of F_2 explicitly. This is the main difference between 2PL and Hierarchical locking for multiple granularity.

Lock Concept

- If we lock the database then everything is lock.
- So when the parent node is lock then children node is automatic lock.
- This is origin point to understand the concept of **Explicit Lock** and **Implicit Lock** Mode.

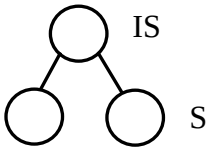
Explicit Lock If any transaction apply this lock at higher level and due to its effect if lower level node is automatic lock then it is called **implicit lock.**



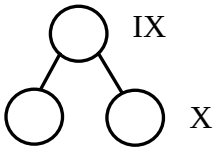
There are three additional lock modes with multiple granularity:

Intention Mode Lock

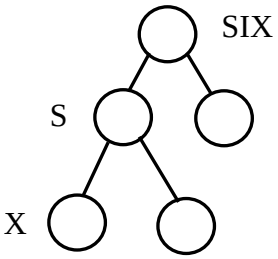
1. **Intention-shared (IS)** - It contains explicit locking at a lower level of the tree but only with shared locks.



2. **Intention-exclusive (IX)** - It contains explicit locking at a lower level with exclusive or shared locks.



3. **Shared-intention-exclusive (SIX)** - the sub-tree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive mode locks.



Recovery with concurrent transaction

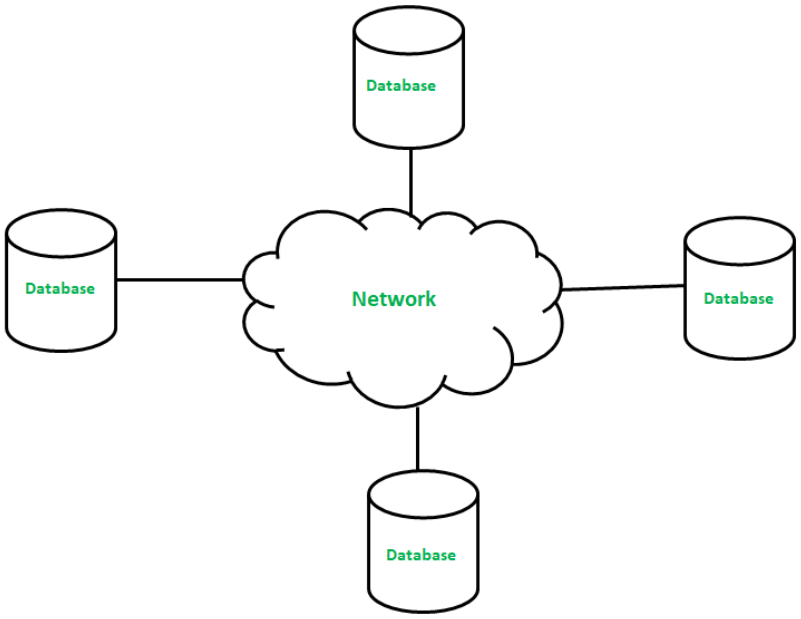
Whenever more than one transaction is being executed, then the interleaved of logs occur. During recovery, it would become difficult for the recovery system to backtrack all logs and then start recovering. To ease this situation, 'checkpoint' concept is used by most DBMS.

Distributed Database Management System (DDBMS)

A distributed database is basically a database that is not limited to one system; it is spread over different sites, i.e, on multiple computers. In a distributed database, there are a number of databases that may be geographically distributed all over the world.

A **distributed database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

This may be required when a particular database needs to be accessed by various users globally. It needs to be managed such that for the users it looks like one single database.



A distributed database management system (DDBMS) is a centralized software system that manages a distributed database in a manner as if it were all stored in a single location.

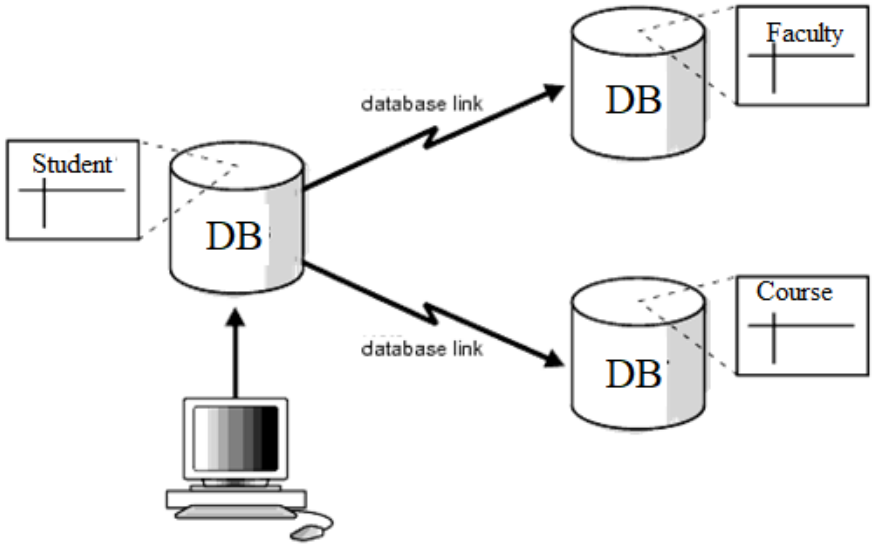
Transaction Processing in Distributed system

A **distributed transaction** is a set of operations on data that is performed across two or more data databases. It is typically coordinated across separate nodes connected by a network.

There are two possible outcomes:

1) All operations successfully complete, or

2) None of the operations are performed at all due to a failure somewhere in the system.



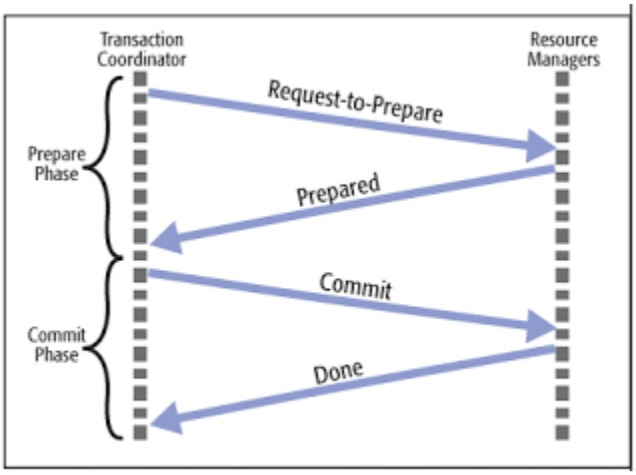
In the second case, if some work was completed prior to the failure, that work will be undone. This type of operation is in fulfillment with the "ACID" properties of databases that ensure data integrity. ACID is most commonly associated with transactions on a single database server, but distributed transactions uses "two-phase commit" (2PC) that guarantee across multiple databases.

Working of Distributed Transactions

Distributed transactions have the same processing completion requirements as regular database transactions, but they are more challenging. The multiple resources add more points of failure, such as the separate software systems that run the resources (e.g., the database software), the extra hardware servers, and network failures. This makes distributed transactions at risk to failures.

For a distributed transaction to occur, transaction managers coordinate the resources (either multiple databases or multiple nodes of a single database). The transaction manager decides whether to commit a successful transaction or rollback an unsuccessful transaction, the latter of which leaves the database unchanged.

First, an application requests the distributed transaction to the transaction manager. The transaction manager then branches to each resource, which will have its own "resource manager" to help it participate in distributed transactions.



Distributed transactions are often done in two phases.

Phase 1: Prepare Phase

- After each resource manager has locally completed its transaction, it sends a "DONE" message to the transaction manager. When the transaction manager has received "DONE" message from all resource manager, it sends a "Prepare" message to the all resource managers.

- The resource managers vote on whether they still want to commit or not. If a resource manager wants to commit, it sends a "Ready" message.
- A resource manager that does not want to commit sends a "Not Ready" message. This may happen when the resource manager has conflicting concurrent transactions or there is a timeout.

Phase 2: Commit/Abort Phase

- After the transaction manager has received "Ready" message from all the resource managers.
- The transaction manager sends a "Global Commit" message to the resource manager.
- The resource managers apply the transaction and send a "Commit ACK" message to the transaction manager.
- When the transaction manager receives "Commit ACK" message from all the resource managers, it considers the transaction as committed.

Distributed Data Storage

There are 2 ways in which data can be stored on different sites. These are:

1. Replication
2. Fragmentation

1. Data Replication

Data Replication is the process of storing data in more than one site or node. It is useful in **improving the availability of data**. It is simply copying data from a database from one server to another server so that all the users can share the same data without any inconsistency. The result is a **distributed database** in which users can access data relevant to their tasks without interfering with the work of others.

Advantages of replication

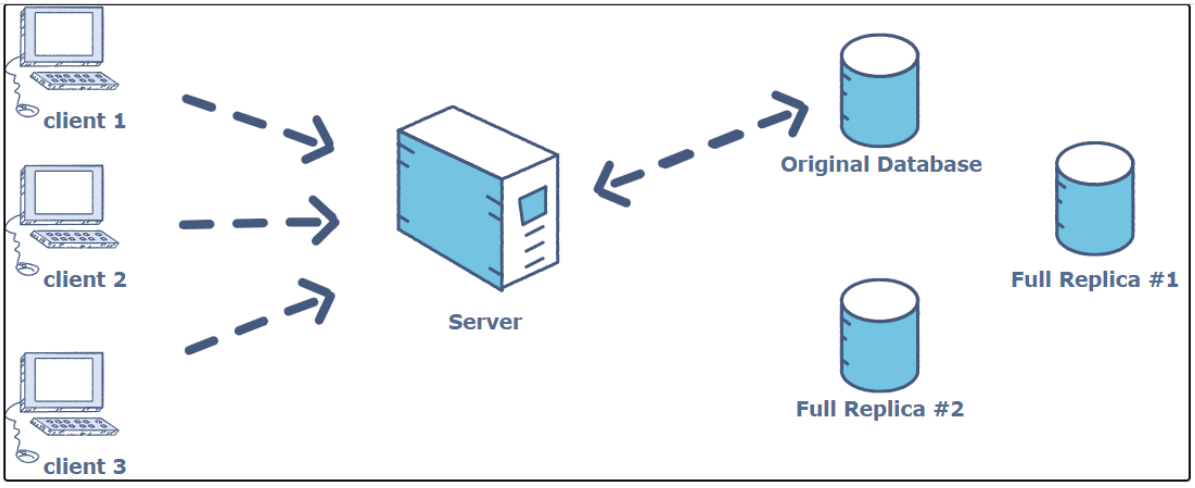
Replication in a database offers database users various benefits that are termed below

- Availability
- Has Reliability
- Gives high Performance
- Facilitates load reduction
- Provide disconnected computing
- Supports many users

There are two kinds of replication techniques:

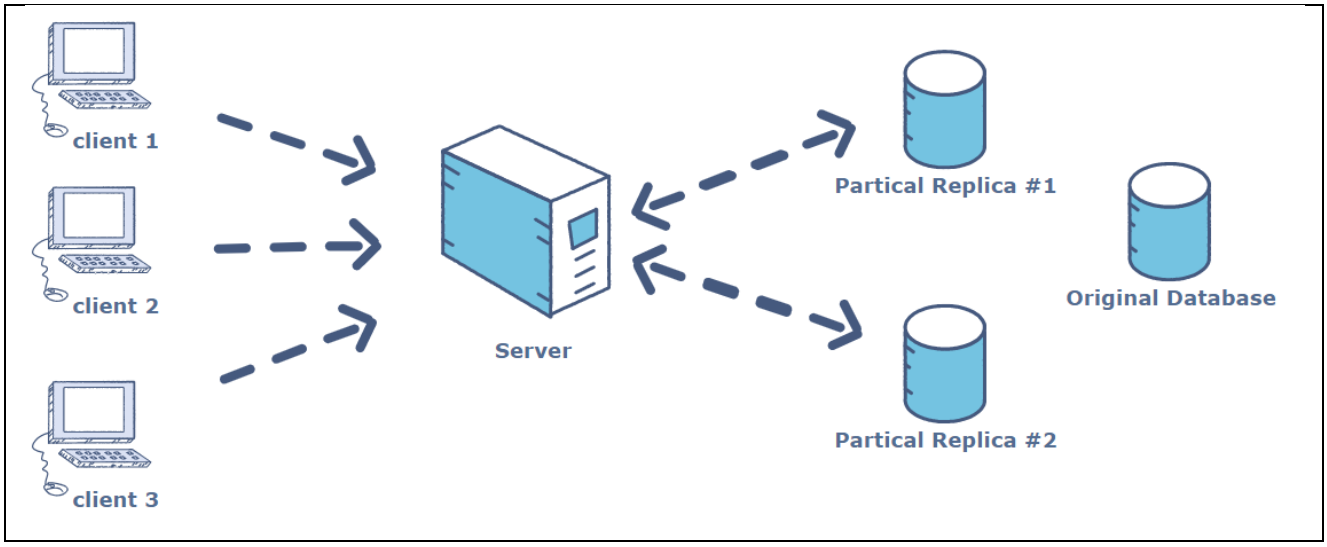
a) Full Replication

Full replication involves replicating the entire database at all sites of the distributed system. This improves data availability, but all the sites need to be constantly updated so that only *updated* data is available to every user.



b) Partial Replication

In this replication technique, only a frequently portion of the database is replicated while the other portion is left out. This does reduce the amount of data to be replicated and updated, but it also means that there are fewer sites (than full replication) that hold some particular data (data availability is compromised).



2. Data Fragmentation

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called **fragments**.

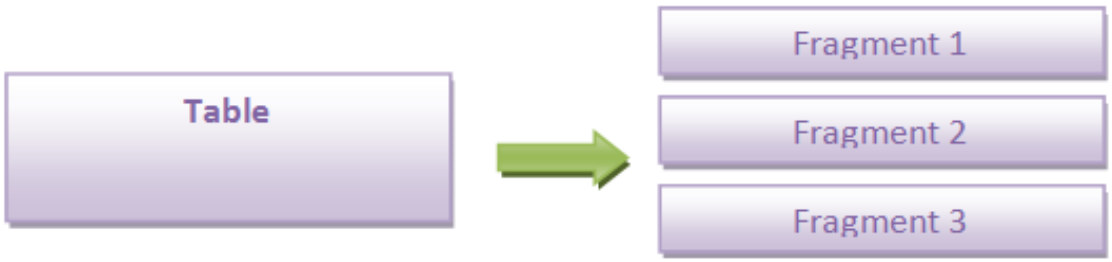
In this approach, the tables are fragmented (i.e., they're divided into smaller parts) and each of the fragments is stored in different sites where they're required. It must be made sure that the fragments are such that they can be used to reconstruct the original relation (i.e, there isn't any loss of data). Fragmentation is useful as it doesn't create copies of data, consistency is not a problem.

There are 3 types of data fragmentations in DDBMS.

a) Horizontal Data Fragmentation
b) Vertical Data Fragmentation
c) Hybrid Data Fragmentation

a) Horizontal Data Fragmentation: *(Splitting by rows)* – The relation (table) is fragmented into groups of tuples so that each tuple is assigned to at least one fragment.

As the name suggests, here the data / records are fragmented horizontally. i.e.; horizontal subset of table data is created and are stored in different database in DDB.



Students

Roll No	Name	DOB	Course
156	Ram	15.10.2000	BCA
157	Pankaj	12.12.2000	BCA
158	Suresh	01.10.2001	BCA
159	Anjali	05.10.2000	BCA
160	Shyam	02.11.1999	MCA
161	Mohd Ali	05.12.1998	MCA
162	Shane Alam	10.10.1998	MCA
163	Meeta	19.12.1999	MCA
170	Mohan	05.05.1994	PGDCA
171	Abhijeet	10.12.1995	PGDCA
172	Darshika	10.15.1996	PGDCA
173	Harpreet	02.02.1996	PGDCA

Roll No	Name	DOB	Course
156	Ram	15.10.2000	BCA
157	Pankaj	12.12.2000	BCA
158	Suresh	01.10.2001	BCA
159	Anjali	05.10.2000	BCA

Roll No	Name	DOB	Course
160	Shyam	02.11.1999	MCA
161	Mohd Ali	05.12.1998	MCA
162	Shane Alam	10.10.1998	MCA
163	Meeta	19.12.1999	MCA

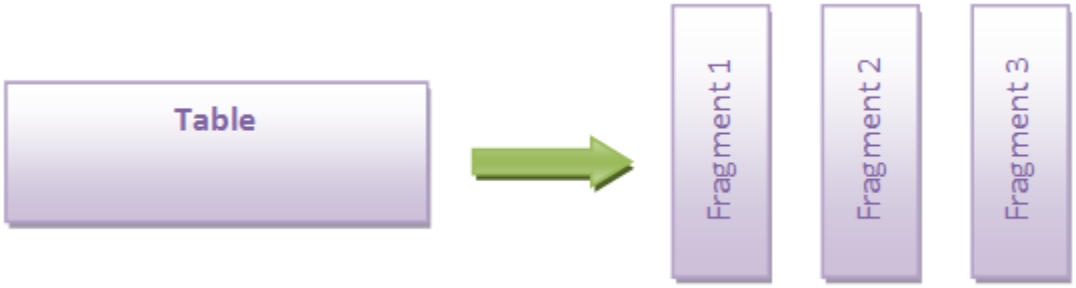
Roll No	Name	DOB	Course
170	Mohan	05.05.1994	PGDCA
171	Abhijeet	10.12.1995	PGDCA
172	Darshika	10.15.1996	PGDCA
173	Harpreet	02.02.1996	PGDCA

For example, consider the students studying in different courses in any university like BCA, MCA, PGDCA etc. number of students from of these courses are not a small number. They are huge in number. When any details of any one student are required, whole table needs to be accessed to get the information. Again the student table may present in any location in the university. But the concept of DDB is to place the data in the nearest DB so that it will be accessed quickly. Hence what we do is divide the entire student table data horizontally based on the course. i.e.;

```
SELECT * FROM Students WHERE Course = ‘BCA’;
SELECT * FROM Students WHERE Course = ‘MCA’;
SELECT * FROM Students WHERE Course = ‘PGDCA’;
```

Now these queries will give the subset of records from Students table depending on the course of the students. Any insert, update and delete on the student records will be done on the DBs at their location and it will be synched with the main table at regular intervals.

b) Vertical Data Fragmentation: *(Splitting by columns)* – The schema of the relation is divided into smaller schemas. Each fragment must contain a common candidate key so as to ensure lossless join.



Students

Roll No	Name	DOB	Course
156	Ram	15.10.2000	BCA
157	Pankaj	12.12.2000	BCA
158	Suresh	01.10.2001	BCA
159	Anjali	05.10.2000	BCA
160	Shyam	02.11.1999	MCA
161	Mohd Ali	05.12.1998	MCA
162	Shane Alam	10.10.1998	MCA
163	Meeta	19.12.1999	MCA
170	Mohan	05.05.1994	PGDCA
171	Abhijeet	10.12.1995	PGDCA
172	Darshika	10.15.1996	PGDCA
173	Harpreet	02.02.1996	PGDCA

Roll No	Name
156	Ram
157	Pankaj
158	Suresh
159	Anjali
160	Shyam
161	Mohd Ali
162	Shane Alam
163	Meeta
170	Mohan
171	Abhijeet
172	Darshika
173	Harpreet

Roll No	Course
156	BCA
157	BCA
158	BCA
159	BCA
160	MCA
161	MCA
162	MCA
163	MCA
170	PGDCA
171	PGDCA
172	PGDCA
173	PGDCA

For example consider the Students table with Roll No, Name, DOB, Course. The vertical fragmentation of this table may be dividing the table into different tables with one or more columns from Students.

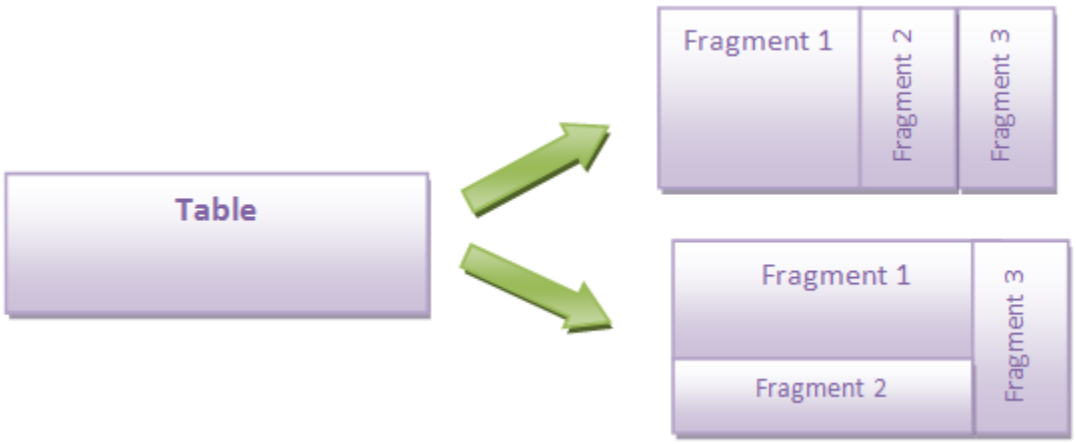
```
SELECT Roll No, Name FROM Students;
SELECT Roll No, Course FROM Students;
```

This type of fragment will have fragmented details about whole students. This will be useful when the user needs to query only few details about the students. For example consider a query to find the course of the students. This can be done by querying the second fragment of the table.

c) Hybrid (Mix) Data Fragmentation:

This is the combination of horizontal as well as vertical fragmentation. Hybrid fragmentation can be done in two alternative ways −

- At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.
- At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.



Student

Roll No	Name	DOB	Course
156	Ram	15.10.2000	BCA
157	Pankaj	12.12.2000	BCA
158	Suresh	01.10.2001	BCA
159	Anjali	05.10.2000	BCA
160	Shyam	02.11.1999	MCA
161	Mohd Ali	05.12.1998	MCA
162	Shane Alam	10.10.1998	MCA
163	Meeta	19.12.1999	MCA
170	Mohan	05.05.1994	PGDCA
171	Abhijeet	10.12.1995	PGDCA
172	Darshika	10.15.1996	PGDCA
173	Harpreet	02.02.1996	PGDCA

Roll No	Name	Course
156	Ram	BCA
157	Pankaj	BCA
158	Suresh	BCA
159	Anjali	BCA

Roll No	Name	Course
160	Shyam	MCA
161	Mohd Ali	MCA
162	Shane Alam	MCA
163	Meeta	MCA

Consider the students table with below fragmentations.

```
SELECT Roll No, Name FROM Students WHERE Course = ‘BCA;
SELECT Roll No, Name FROM Students WHERE Course = ‘MCA;
```

This is a hybrid or mixed fragmentation of Students table.

Allocation in distributed system

Allocation: Each copy of a fragment must be assigned to a particular site in the distributed system. This process is called data distribution or allocation.

The allocation technique is used for the allocation of fragments or replicas of fragments for storage at various sites.

All the information concerning data fragmentation, allocation, and replication is stored in a global directory. This directory can be accessed by the DDBS applications as and when required.

The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site.

For example, if high availability is required, transactions can be submitted at any site, and most transactions are retrieval only, a fully replicated database is a good choice.

However, if certain transactions that access particular parts of the database are mostly submitted at a particular site, the corresponding set of fragments can be allocated at that site only. Data that is accessed at multiple sites can be replicated at those sites. If many updates are performed, it may be useful to limit replication.