**DBMS Mid-2 answers**

**Module-3**

**1.** **Define Normalization? List out the problems caused by redundancy and explain them briefly. Illustrate fourth normal form with a suitable example**

**Ans:**

Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize the redundancy in relations. Normal forms are used to eliminate or reduce redundancy in database tables.

Redundancy means having multiple copies of the same data in the database. This problem arises when a database is not normalized.

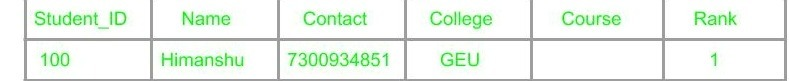
Suppose a table of student details attributes are: student Id, student name, college name, college rank, course opted.



As it can be observed that values of attribute college name, college rank, course are being repeated which can lead to problems. Problems caused due to redundancy are: Insertion anomaly, Deletion anomaly, and Updation anomaly.

1. Insertion Anomaly –

If a student detail has to be inserted whose course is not being decided yet then insertion will not be possible till the time course is decided for student



This problem happens when the insertion of a data record is not possible without adding some additional unrelated data to the record.

2. Deletion Anomaly –

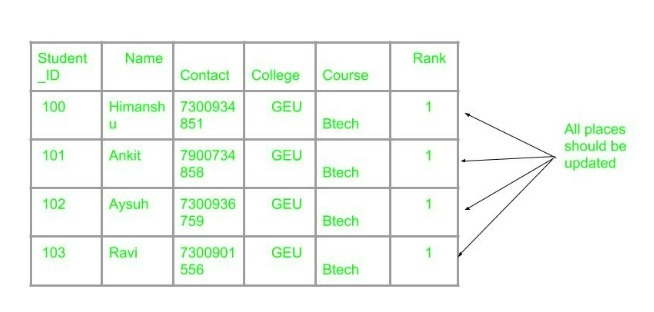
If the details of students in this table are deleted then the details of college will also get deleted which should not occur by common sense.

This anomaly happens when deletion of a data record results in losing some unrelated information that was stored as part of the record that was deleted from a table.

It is not possible to delete some information without loosing some other information in the table as well.

3. Updation Anomaly –

Suppose if the rank of the college changes then changes will have to be all over the database which will be time-consuming and computationally costly.



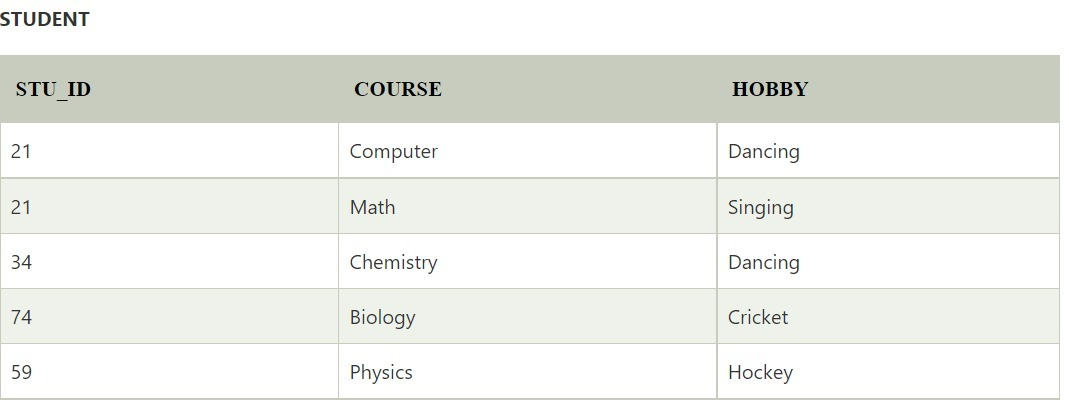
If updates do not occur at all places then the database will be in an inconsistent state.

**Fourth normal form (4NF)**

A relation will be in 4NF if it is in Boyce Codd normal form and has no multivalued dependency.

For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multivalued dependency.

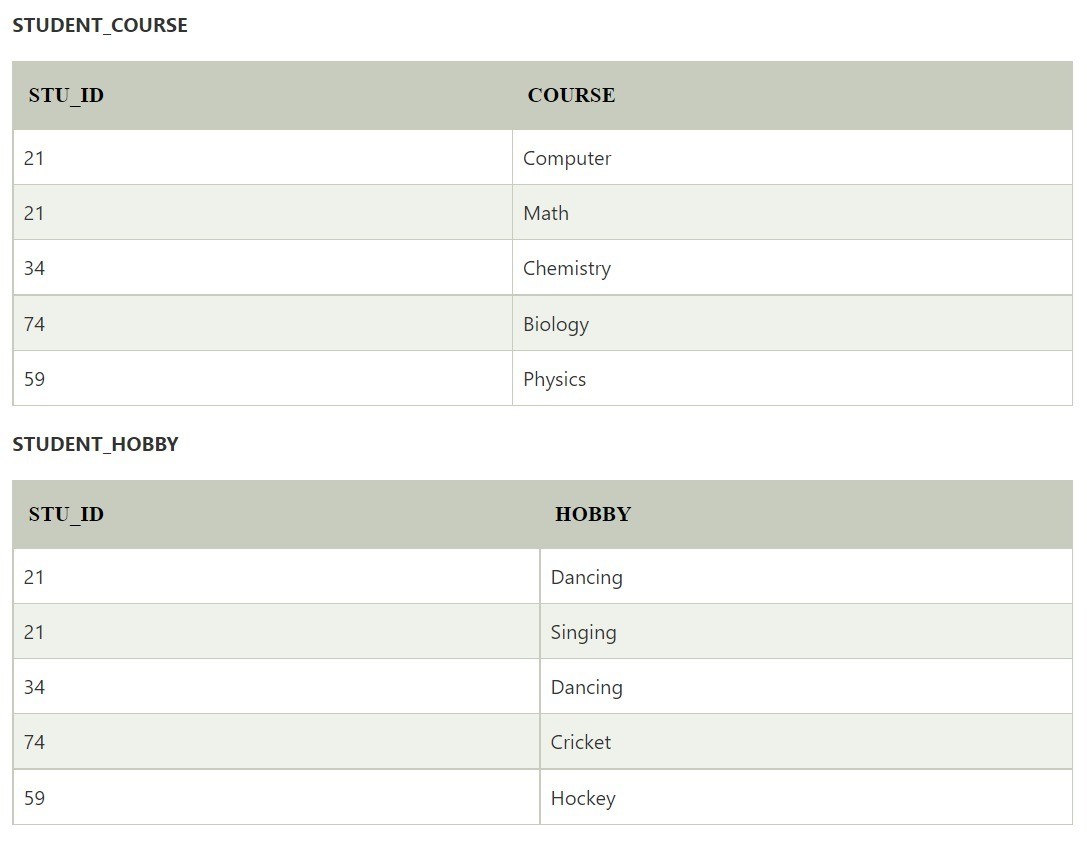
**Example:**



The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entities. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU\_ID, 21 contains two courses, Computer and Math and two hobbies, Dancing and Singing. So there is a Multi-valued dependency on STU\_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:



**2.** **Define functional dependency?   Briefly explain about first normal form by taking suitable table.**

**Ans:**

**Functional Dependency**

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

X → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

***For example:***

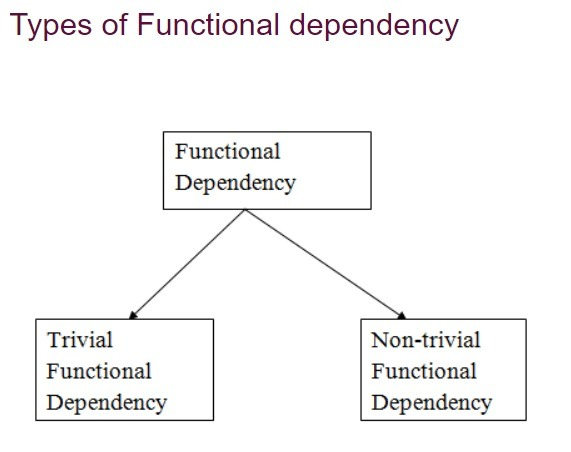
Assume we have an employee table with attributes: Emp\_Id, Emp\_Name, Emp\_Address.

Here Emp\_Id attribute can uniquely identify the Emp\_Name attribute of employee table because if we know the Emp\_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

Emp\_Id → Emp\_Name

We can say that Emp\_Name is functionally dependent on Emp\_Id.



**First Normal Form (1NF):**

If a relation contains a composite or multi-valued attribute, it violates the first normal form, or the relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is singled valued attribute.

A table is in 1 NF iff:

* There are only Single Valued Attributes.
* Attribute Domain does not change.
* There is a unique name for every Attribute/Column.
* The order in which data is stored does not matter.



A database design is considered as bad if it is not even in the First Normal Form (1NF)

**3. What is meant by multi-valued dependency? Explain with suitable example.**

**Ans:**

**Multivalued Dependency**

Multivalued dependency occurs when two attributes in a table are independent of each other but both depend on a third attribute.

A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes. We use multivalued conditions when we encounter these two different ways:

When we want to test the relations or decide if these happen to be lawful under some arrangement of practical as well as multivalued dependencies.

When we want to determine what limitations are there on the arrangement of the lawful relations. Thus, we will concern ourselves with just the relations that fulfill a given arrangement of practical as well as multivalued dependencies.

If a table has attributes P, Q and R, then Q and R are multi-valued facts of P.

It is represented by double arrow −

**->->**

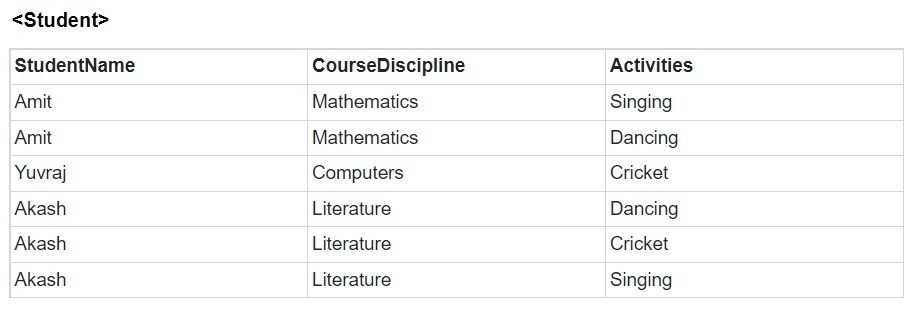
For our example:

**P->->QP->->R**

In the above case, Multivalued Dependency exists only if Q and R are independent attributes.

A table with multivalued dependency violates the 4NF.

**Example:**



In the above table, we can see Students Amit and Akash have interest in more than one activity.

This is a multivalued dependency because CourseDiscipline of a student are independent of Activities, but are dependent on the student.

Therefore, multivalued dependency −

**StudentName ->-> CourseDisciplineStudentName ->-> Activities**

The above relation violates Fourth Normal Form in Normalization.

**4.Explain in detail about 3NF and BCNF. List the differences between 3NF and BCNF**

**Ans:**

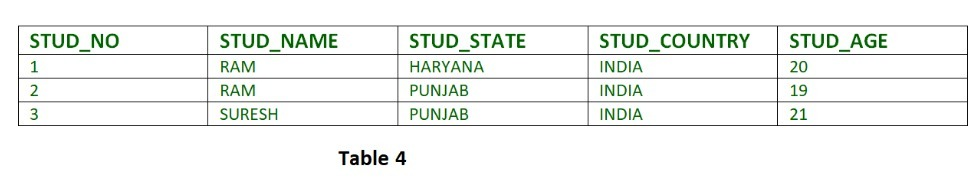
**Third Normal Form –**

A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

A relation is in 3NF if at least one of the following condition holds in every non-trivial functional dependency X –> Y

X is a super key.

Y is a prime attribute (each element of Y is part of some candidate key).



In relation STUDENT given in Table 4,

FD set: {STUD\_NO -> STUD\_NAME, STUD\_NO -> STUD\_STATE, STUD\_STATE -> STUD\_COUNTRY, STUD\_NO -> STUD\_AGE}

**Candidate Key**: {STUD\_NO}

For this relation in table 4, STUD\_NO -> STUD\_STATE and STUD\_STATE -> STUD\_COUNTRY are true. So STUD\_COUNTRY is transitively dependent on STUD\_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_COUNTRY\_STUD\_AGE) as:

STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_AGE)

STATE\_COUNTRY (STATE, COUNTRY)

**Boyce-Codd Normal Form (BCNF) –**

A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key. A relation is in BCNF iff in every non-trivial functional dependency X –> Y, X is a super key.

**Example 1** – Find the highest normal form of a relation R(A,B,C,D,E) with FD set as {BC->D, AC->BE, B->E}

***Step 1***. As we can see, (AC)+ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can’t be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.

***Step 2***. Prime attributes are those attributes that are part of candidate key {A, C} in this example and others will be non-prime {B, D, E} in this example.

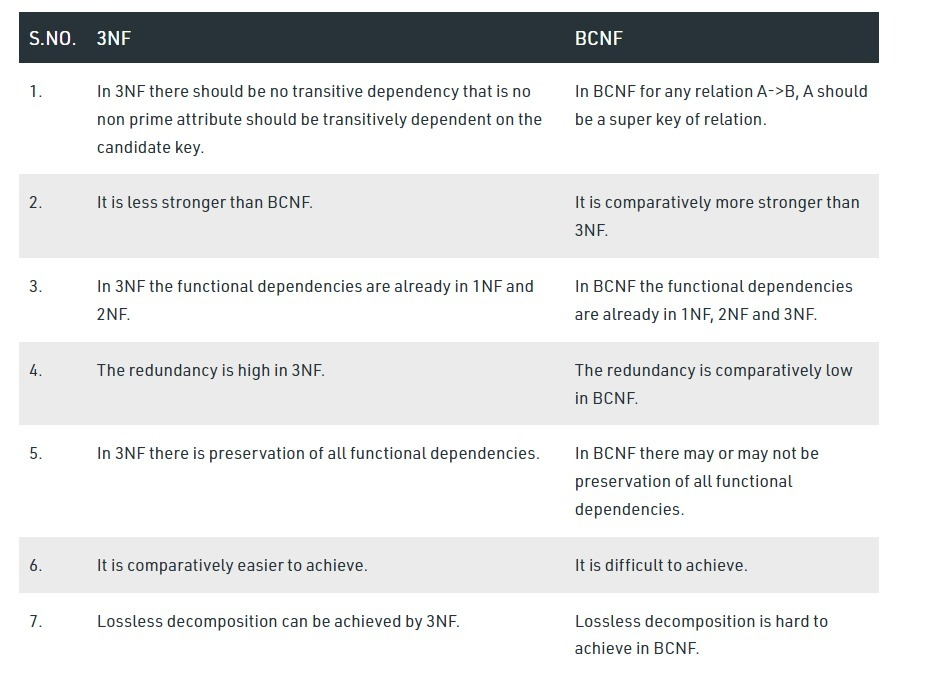
***Step 3***. The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.

The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not a proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).

The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B->E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute.

So the highest normal form of relation will be 2nd Normal form.

**Differences:**



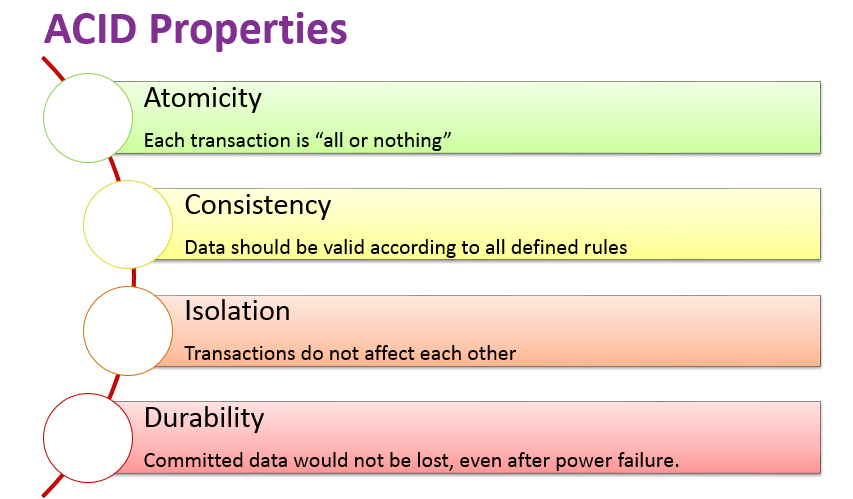
**Module-4**

**1.Discuss the ACID properties. Illustrate them through examples?**

**Ans:**

A transaction is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called ACID properties.



**Atomicity:**

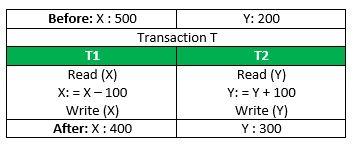
By this, we mean that either the entire transaction takes place at once or doesn’t happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.

—Abort: If a transaction aborts, changes made to the database are not visible.

—Commit: If a transaction commits, changes made are visible.

Atomicity is also known as the ‘All or nothing rule’.

Consider the following transaction T consisting of T1 and T2: Transfer of 100 from account X to account Y.



If the transaction fails after completion of T1 but before completion of T2.( say, after write(X) but before write(Y)), then the amount has been deducted from X but not added to Y. This results in an inconsistent database state. Therefore, the transaction must be executed in its entirety in order to ensure the correctness of the database state.

**Consistency:**

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,

The total amount before and after the transaction must be maintained.

Total before T occurs = 500 + 200 = 700.

Total after T occurs = 400 + 300 = 700.

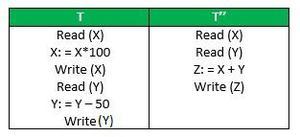
Therefore, the database is consistent. Inconsistency occurs in case T1 completes but T2 fails. As a result, T is incomplete.

**Isolation:**

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

Let X= 500, Y = 500.

Consider two transactions T and T”.



Suppose T has been executed till Read (Y) and then T’’ starts. As a result, interleaving of operations takes place due to which T’’ reads the correct value of X but the incorrect value of Y and sum computed by

T’’: (X+Y = 50, 000+500=50, 500)

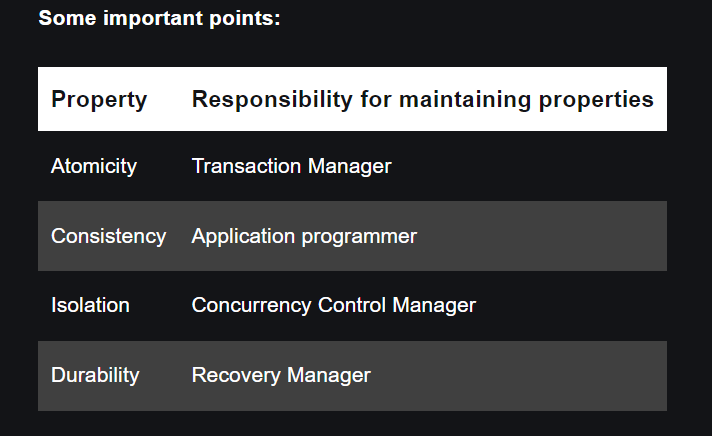
is thus not consistent with the sum at end of the transaction:

T: (X+Y = 50, 000 + 450 = 50, 450).

This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

**Durability:**

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.



The ACID properties, in totality, provide a mechanism to ensure the correctness and consistency of a database in a way such that each transaction is a group of operations that acts as a single unit, produces consistent results, acts in isolation from other operations, and updates that it makes are durably stored.

**2. Demonstrate about Transaction Management and different States of Transaction.**

**Ans:**

**Transaction Management**

Transactions in DBMS :

Transactions are a set of operations used to perform a logical set of work. A transaction usually means that the data in the database has changed. One of the major uses of DBMS is to protect the user’s data from system failures. It is done by ensuring that all the data is restored to a consistent state when the computer is restarted after a crash. The transaction is any one execution of the user program in a DBMS. Executing the same program multiple times will generate multiple transactions.

*Example –*

Transaction to be performed to withdraw cash from an ATM vestibule. hree operations can be performed in a transaction as follows.

* Read/Access data (R).
* Write/Change data (W).
* Commit.

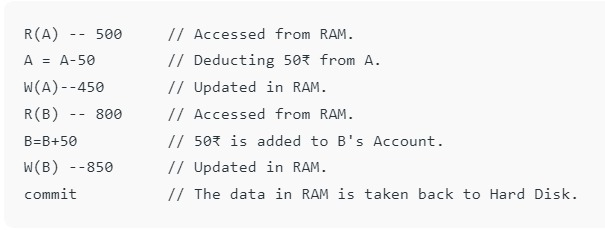
*Uses of Transaction Management* :

The DBMS is used to schedule the access of data concurrently. It means that the user can access multiple data from the database without being interfered with each other. Transactions are used to manage concurrency.

* It is also used to satisfy ACID properties.
* It is used to solve Read/Write Conflict.
* It is used to implement Recoverability, Serializability, and Cascading.
* Transaction Management is also used for Concurrency Control Protocols and Locking of data.

**Example** –

Transfer of 50₹ from Account A to Account B. Initially A= 500₹, B= 800₹. This data is brought to RAM from Hard Disk.



**Transaction States in DBMS**

 States through which a transaction goes during its lifetime. These are the states which tell about the current state of the Transaction and also tell how we will further do the processing in the transactions. These states govern the rules which decide the fate of the transaction whether it will commit or abort. These are different types of Transaction States :

**Active State** –

When the instructions of the transaction are running then the transaction is in active state. If all the ‘read and write’ operations are performed without any error then it goes to the “partially committed state”; if any instruction fails, it goes to the “failed state”.

**Partially Committed** –

After completion of all the read and write operation the changes are made in main memory or local buffer. If the changes are made permanent on the DataBase then the state will change to “committed state” and in case of failure it will go to the “failed state”.

**Failed State** –

When any instruction of the transaction fails, it goes to the “failed state” or if failure occurs in making a permanent change of data on Data Base.

**Aborted State** –

After having any type of failure the transaction goes from “failed state” to “aborted state” and since in previous states, the changes are only made to local buffer or main memory and hence these changes are deleted or rolled-back.

**Committed state**–

It is the state when the changes are made permanent on the Data Base and the transaction is complete and therefore terminated in the “terminated state”.

**Terminated State** –

If there isn’t any roll-back or the transaction comes from the “committed state”, then the system is consistent and ready for new transaction and the old transaction is terminated.

**3.Explain about conflict serializability and view serializability with suitable example.**

**Ans:**

**Conflict Serializable Schedule**

A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.

The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

*Conflicting Operations*

The two operations become conflicting if all conditions satisfy:

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

*Example:*

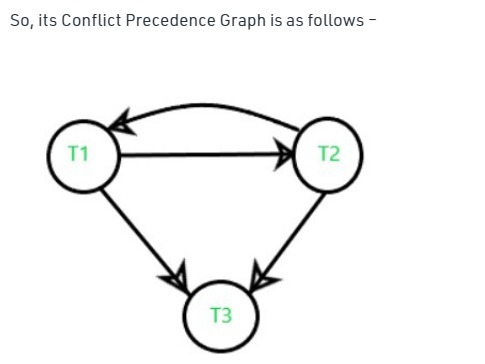
Swapping is possible only if S1 and S2 are logically equal.



**View serializability**

View serializability is a concept that is used to compute whether schedules are View-Serializable or not. A schedule is said to be View-Serializable if it is view equivalent to a Serial Schedule (where no interleaving of transactions is possible).

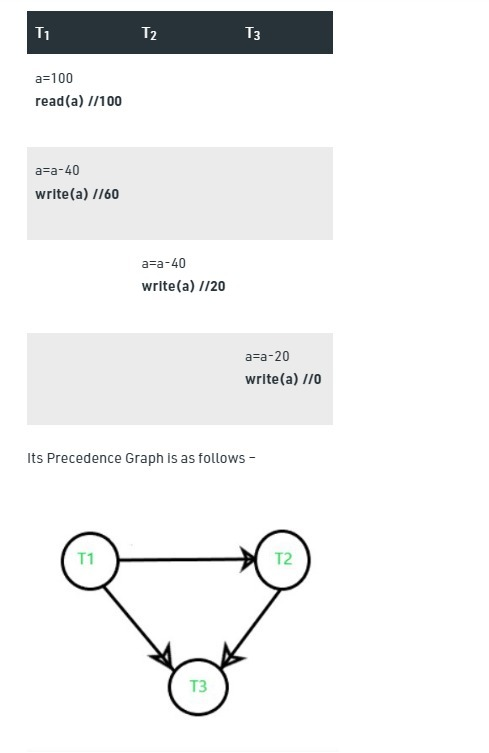




The above graph contains cycle/loop which means it is not conflict-serializable but it does not mean that it cannot be consistent and equivalent to the serial schedule it may or may not be.

LookSchedule S’1 :

In the above example if we do swapping among some transaction’s operation so our table will look like –



Now, we see that the precedence graph of the second table does not contain any cycle/loop, which means it is conflict serializable (equivalent to serial schedule, consistent) and the final result is coming the same as the first table.

**4. Define the term Lock. Explain various types of locks along with their compatibility functions.**

**Ans:**

A lock is a data variable which is associated with a data item. This lock signifies that operations that can be performed on the data item. Locks in DBMS help synchronize access to the database items by concurrent transactions. All lock requests are made to the concurrency-control manager.

**Types of Locks**

Several types of locks are used in concurrency control. To introduce locking concepts gradually, we first discuss binary locks, which are simple but restrictive and so are not used in practice. We then discuss shared/exclusive locks, which provide more general locking capabilities and are used in practical database locking schemes.

**Binary Locks**

A binary lock can have two states or values: locked and unlocked.

A distinct lock is associated with each database item A. If the value of the lock on A is 1, item A cannot be accessed by a database operation that requests the item. If the value of the lock on A is 0 then item can be accessed when requested. We refer to the current value of the lock associated with item A as LOCK (A). There are two operations, lock item and unlock item are used with binary locking A transaction requests access to an item A by first issuing a lock item (A) operation. If LOCK (A) = 1, the transaction is forced to wait. If LOCK (A) = 0 it is set to 1 (the transaction locks the item) and the transaction is allowed to access item A. When the transaction is through using the item, it issues an unlock item (A) operation, which sets LOCK (A) to 0 (unlocks the item) so that A may be accessed by other transactions. Hence binary lock enforces mutual exclusiol1 on the data item.

**Share/Exclusive (for Read/Write) Locks**

We should allow several transactions to access the same item A if they all access A’ for reading purposes only. However, if a transaction is to write an item A, it must have exclusive access to A. For this purpose, a different type of lock called a multiple-mode lock is used. In this scheme there are shared/exclusive or read/write locks are used.

*Locking operations*

There are three locking operations called read\_lock(A), write\_lock(A) and unlock(A) represented as lock-S(A), lock-X(A), unlock(A) (Here, S indicates shared lock, X indicates exclusive lock)can be performed on a data item. A lock associated with an item A, LOCK (A), now has three possible states: “read-locked”, “write-locked,” or “unlocked.” A read-locked item is also called share-locked item because other transactions are allowed to read the item, whereas a write-locked item is caused exclusive-locked, because a single transaction exclusively holds the lock on the item.

**Update lock (U)** – this lock is similar to an exclusive lock but is designed to be more flexible in a way. An update lock can be imposed on a record that already has a shared lock. In such a case, the update lock will impose another shared lock on the target row. Once the transaction that holds the update lock is ready to change the data, the update lock (U) will be transformed to an exclusive lock (X). It is important to understand that update lock is asymmetrical in regards of shared locks. While the update lock can be imposed on a record that has the shared lock, the shared lock cannot be imposed on the record that already has the update lock

**Intent locks (I)** – this lock is a means used by a transaction to inform another transaction about its intention to acquire a lock. The purpose of such lock is to ensure data modification to be executed properly by preventing another transaction to acquire a lock on the next in hierarchy object. In practice, when a transaction wants to acquire a lock on the row, it will acquire an intent lock on a table, which is a higher hierarchy object. By acquiring the intent lock, the transaction will not allow other transactions to acquire the exclusive lock on that table (otherwise, exclusive lock imposed by some other transaction would cancel the row lock).

**5.** **Explain concurrency control with Time-Stamp based protocols with suitable example?**

**Ans:**

**Timestamp based Concurrency Control**

Concurrency Control can be implemented in different ways. One way to implement it is by using Locks

Timestamp is a unique identifier created by the DBMS to identify a transaction. They are usually assigned in the order in which they are submitted to the system. Refer to the timestamp of a transaction T as TS(T).

**Timestamp Ordering Protocol –**

The main idea for this protocol is to order the transactions based on their Timestamps. A schedule in which the transactions participate is then serializable and the only equivalent serial schedule permitted has the transactions in the order of their Timestamp Values. Stating simply, the schedule is equivalent to the particular Serial Order corresponding to the order of the Transaction timestamps. An algorithm must ensure that, for each item accessed by Conflicting Operations in the schedule, the order in which the item is accessed does not violate the ordering. To ensure this, use two Timestamp Values relating to each database item X.

W­\_TS(X) is the largest timestamp of any transaction that executed write(X) successfully.

R\_TS(X) is the largest timestamp of any transaction that executed read(X) successfully.

**Basic Timestamp Ordering –**

Every transaction is issued a timestamp based on when it enters the system. Suppose, if an old transaction Ti has timestamp TS(Ti), a new transaction Tj is assigned timestamp TS(Tj) such that TS(Ti) < TS(Tj). The protocol manages concurrent execution such that the timestamps determine the serializability order. The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. Whenever some Transaction T tries to issue a R\_item(X) or a W\_item(X), the Basic TO algorithm compares the timestamp of T with R\_TS(X) & W\_TS(X) to ensure that the Timestamp order is not violated. This describes the Basic TO protocol in the following two cases.

1.Whenever a Transaction T issues a W\_item(X) operation, check the following conditions:

* If R\_TS(X) > TS(T) or if W\_TS(X) > TS(T), then abort and rollback T and reject the operation. else,
* Execute W\_item(X) operation of T and set W\_TS(X) to TS(T).

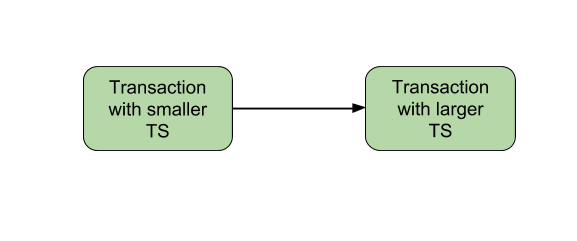
2.Whenever a Transaction T issues a R\_item(X) operation, check the following conditions:

* If W\_TS(X) > TS(T), then abort and reject T and reject the operation, else
* If W\_TS(X) <= TS(T), then execute the R\_item(X) operation of T and set R\_TS(X) to the larger of TS(T) and current R\_TS(X).

Whenever the Basic TO algorithm detects two conflicting operations that occur in an incorrect order, it rejects the latter of the two operations by aborting the Transaction that issued it. Schedules produced by Basic TO are guaranteed to be conflict serializable. Already discussed that using Timestamp can ensure that our schedule will be deadlock free.

One drawback of the Basic TO protocol is that Cascading Rollback is still possible. Suppose we have a Transaction T1 and T2 has used a value written by T1. If T1 is aborted and resubmitted to the system then, T must also be aborted and rolled back. So the problem of Cascading aborts still prevails.

Let’s gist the Advantages and Disadvantages of Basic TO protocol:



Timestamp Ordering protocol ensures serializability since the precedence graph will be of the form:

**Image** – Precedence Graph for TS ordering

* Timestamp protocol ensures freedom from deadlock as no transaction ever waits.
* But the schedule may not be cascade free, and may not even be recoverable.

**Strict Timestamp Ordering –**

A variation of Basic TO is called Strict TO ensures that the schedules are both Strict and Conflict Serializable. In this variation, a Transaction T that issues a R\_item(X) or W\_item(X) such that TS(T) > W\_TS(X) has its read or write operation delayed until the Transaction T‘ that wrote the values of X has committed or aborted.

**6.a. Explain the various concurrency control problems with an examples.**

**Ans:**

Concurrency can simply be said to be executing multiple transactions at a time. It is required to increase time efficiency. If many transactions try to access the same data, then inconsistency arises. Concurrency control required to maintain consistency data.

For example, if we take ATM machines and we do not use concurrency, then multiple persons cannot draw money at a time in different places. This is where we need concurrency.

To run transactions concurrently, we interleave their operations. Each transaction gets a share of the computing time.

This leads to the following problems −

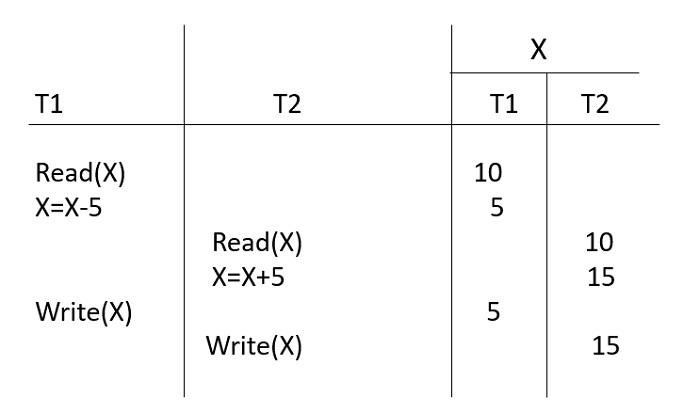
* Lost update problem (write-write conflict)
* Temporary update or dirty read problem (write-read conflict).
* Unrepeatable read or incorrect analysis problem (read-write conflict).

All arises because isolation is broken.

## **Lost update problem**

One transaction does some changes and another transaction deletes those changes. One transaction nullifies the updates of another transaction.

Two transactions T1 and T2 read, modify, write to the same data item in an interleaved fashion for which an incorrect value is stored in x. T2 reads the value of X before T1 changes it hence the updated value resulting from T1 is lost.

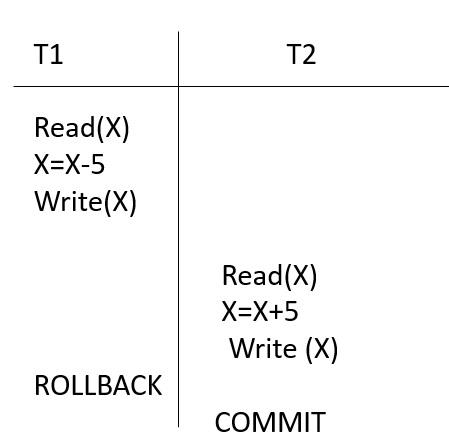


The final value of x is 15, which is incorrect.

## **Temporary update problem or dirty read problem**

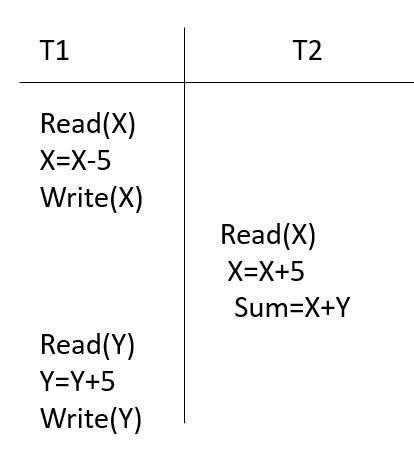
One variable has been updated in one transaction, at the same time another transaction has started and deleted the value of the variable where the variable is not getting updated or committed. That has been done on the first transaction; this gives us false values or the previous values of the variables; this is a major problem.

T2 reads the update value of X made by T1, but T1 fails and rolls back. So, T2 reads an incorrect value of X.



## **Unrepeatable read or incorrect analysis problem**

One transaction is updating multiple different variables and another transaction is in a process to update those variables. The problem occurs is inconsistency of the same variable in different instances.



T1 consists of two parts – subtract 5 from X and add 5 to Y.

In T2, the value of X is updated but the value of Y is not updated. The sum variable stores an incorrect value. Following protocols are used to control concurrency and preserve consistency.

**b.** **What is Transaction? Explain the ACID properties of a transaction**

**Ans:**

#### **Transaction**

A transaction is a single logical unit of work formed by a set of operations. the operations which are between the beginning and the end of the transaction are counted as a single logical unit. The database is inconsistent during the transaction. It goes into a consistent state only when the transaction has occurred successfully. It is very important to have a successful transaction.

**ACID PROPERTIES OF A TRANSACTION IS EXPLAINED IN 1ST ANSWER**

**7.** **Define the term Lock. Explain various types of locks along with their compatibility functions.**

**Ans:**

**“SAME AS 4TH ANSWER”**

**8.** **Discuss in detail about the various locking methods of concurrency control**

**Ans:**

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories −

* Lock based protocols
* Time stamp based protocols

**Lock-based Protocols**

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds −

**Binary Locks** − A lock on a data item can be in two states; it is either locked or unlocked.

**Shared/exclusive** − This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

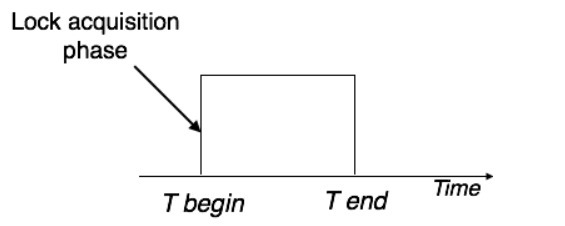
There are four types of lock protocols available −

**Simplistic Lock Protocol**

Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the ‘write’ operation.

**Pre-claiming Lock Protocol**

Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.



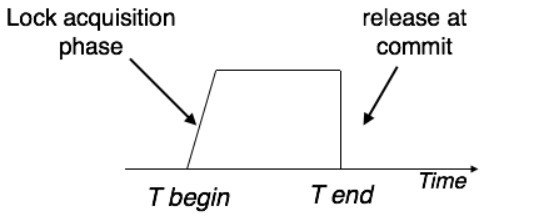
**Two-Phase Locking 2PL**

This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks



**Strict Two-Phase Locking**

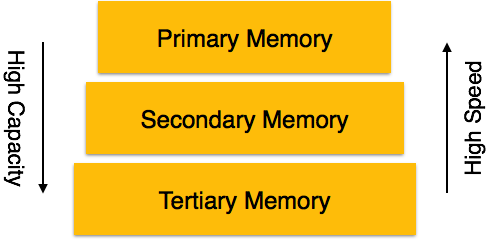
The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.



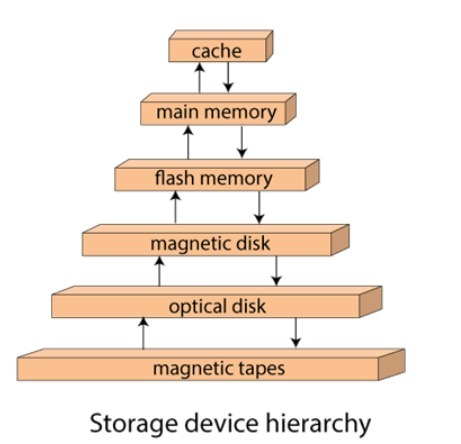
**Module-5**

**1.** **Summarize in detail Storage System with suitable example**

**Ans:**

Databases are stored in file formats, which contain records. At physical level, the actual data is stored in electromagnetic format on some device. These storage devices can be broadly categorized into three types − 

* **Primary Storage** − The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires continuous power supply in order to maintain its state. In case of a power failure, all its data is lost.
* **Secondary Storage** − Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.
* **Tertiary Storage** − Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system. Optical disks and magnetic tapes are widely used as tertiary storage.
* **Optical Storage**: An optical storage can store megabytes or gigabytes of data. A Compact Disk (CD) can store 700 megabytes of data with a playtime of around 80 minutes. On the other hand, a Digital Video Disk or a DVD can store 4.7 or 8.5 gigabytes of data on each side of the disk.
* **Tape Storage**: It is the cheapest storage medium than disks. Generally, tapes are used for archiving or backing up the data. It provides slow access to data as it accesses data sequentially from the start. Thus, tape storage is also known as sequential-access storage. Disk storage is known as direct-access storage as we can directly access the data from any location on disk.



**2. What is file organization? Explain in detail about types of file organizations**.

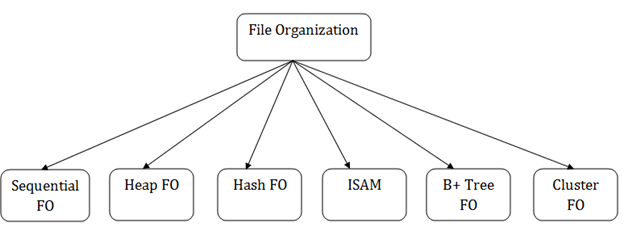
# Ans: File Organization

* File organization is a logical relationship among various records. This method defines how file records are mapped onto disk blocks.
* File organization is used to describe the way in which the records are stored in terms of blocks, and the blocks are placed on the storage medium.

## **Types of file organization:**

File organization contains various methods. These particular methods have pros and cons on the basis of access or selection. In the file organization, the programmer decides the best-suited file organization method according to his requirement.

Types of file organization are as follows:



**1.Sequential File Organization**

This method is the easiest method for file organization. In this method, files are stored sequentially.

## **Pros of sequential file organization**

* It contains a fast and efficient method for the huge amount of data.
* In this method, files can be easily stored in cheaper storage mechanism like magnetic tapes.
* It is simple in design. It requires no much effort to store the data.
* This method is used when most of the records have to be accessed like grade calculation of a student, generating the salary slip, etc.
* This method is used for report generation or statistical calculations.

## **Cons of sequential file organization**

* It will waste time as we cannot jump on a particular record that is required but we have to move sequentially which takes our time.
* Sorted file method takes more time and space for sorting the records.

**2.Heap File Organization**

* It is the simplest and most basic type of organization. It works with data blocks. In heap file organization, the records are inserted at the file's end. When the records are inserted, it doesn't require the sorting and ordering of records.
* When the data block is full, the new record is stored in some other block. This new data block need not to be the very next data block, but it can select any data block in the memory to store new records. The heap file is also known as an unordered file.
* In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.

## **Insertion of a new record**



If we want to search, update or delete the data in heap file organization, then we need to traverse the data from staring of the file till we get the requested record.

If the database is very large then searching, updating or deleting of record will be time-consuming because there is no sorting or ordering of records. In the heap file organization, we need to check all the data until we get the requested record.

## **Pros of Heap file organization**

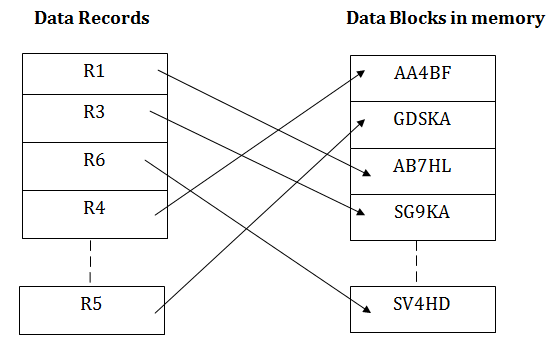
* It is a very good method of file organization for bulk insertion. If there is a large number of data which needs to load into the database at a time, then this method is best suited.
* In case of a small database, fetching and retrieving of records is faster than the sequential record.

## **Cons of Heap file organization**

* This method is inefficient for the large database because it takes time to search or modify the record.
* This method is inefficient for large databases.

**3.Hash File Organization**

Hash File Organization uses the computation of hash function on some fields of the records. The hash function's output determines the location of disk block where the records are to be placed.

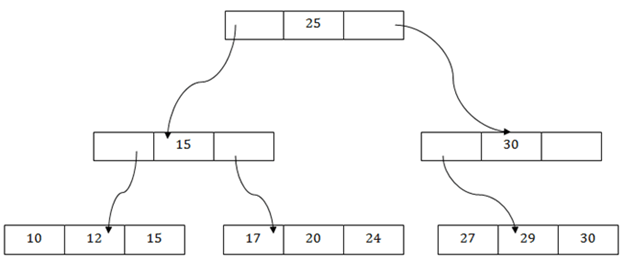


When a record has to be received using the hash key columns, then the address is generated, and the whole record is retrieved using that address. In the same way, when a new record has to be inserted, then the address is generated using the hash key and record is directly inserted. The same process is applied in the case of delete and update.

In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.

**4.B+ File Organization**

* B+ tree file organization is the advanced method of an indexed sequential access method. It uses a tree-like structure to store records in File.
* It uses the same concept of key-index where the primary key is used to sort the records. For each primary key, the value of the index is generated and mapped with the record.
* The B+ tree is similar to a binary search tree (BST), but it can have more than two children. In this method, all the records are stored only at the leaf node. Intermediate nodes act as a pointer to the leaf nodes. They do not contain any records.



## **The above B+ tree shows that:**

* There is one root node of the tree, i.e., 25.
* There is an intermediary layer with nodes. They do not store the actual record. They have only pointers to the leaf node.
* The nodes to the left of the root node contain the prior value of the root and nodes to the right contain next value of the root, i.e., 15 and 30 respectively.
* There is only one leaf node which has only values, i.e., 10, 12, 17, 20, 24, 27 and 29.
* Searching for any record is easier as all the leaf nodes are balanced.
* In this method, searching any record can be traversed through the single path and accessed easily.

## **Pros of B+ tree file organization**

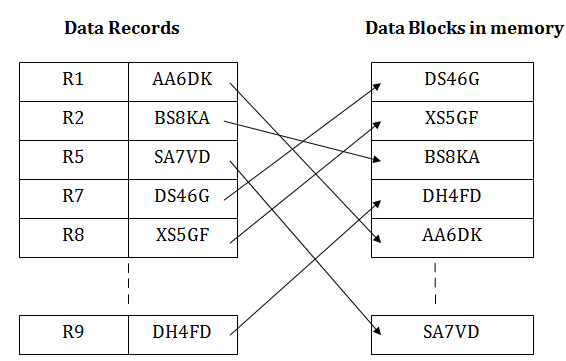
* In this method, searching becomes very easy as all the records are stored only in the leaf nodes and sorted the sequential linked list.
* Traversing through the tree structure is easier and faster.
* The size of the B+ tree has no restrictions, so the number of records can increase or decrease and the B+ tree structure can also grow or shrink.
* It is a balanced tree structure, and any insert/update/delete does not affect the performance of tree.

## **Cons of B+ tree file organization**

* This method is inefficient for the static method.

**5.Indexed Sequential Access Method(ISAM)**

ISAM method is an advanced sequential file organization. In this method, records are stored in the file using the primary key. An index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file.



If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

## **Pros of ISAM:**

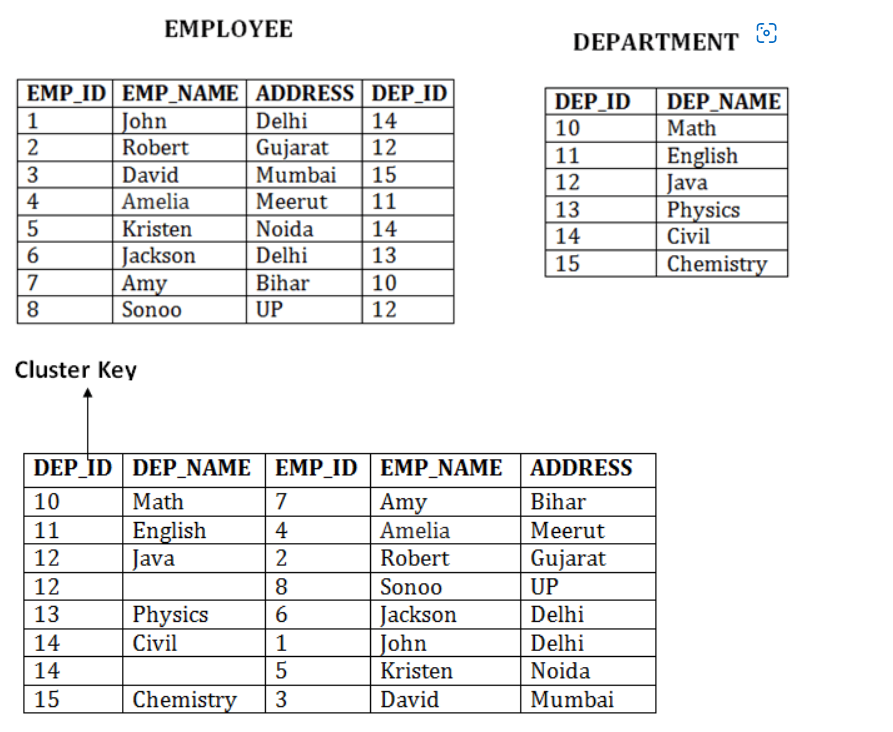
* In this method, each record has the address of its data block, searching a record in a huge database is quick and easy.
* This method supports range retrieval and partial retrieval of records. Since the index is based on the primary key values, we can retrieve the data for the given range of value. In the same way, the partial value can also be easily searched, i.e., the student name starting with 'JA' can be easily searched.

## **Cons of ISAM**

* This method requires extra space in the disk to store the index value.
* When the new records are inserted, then these files have to be reconstructed to maintain the sequence.
* When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

**6.Cluster File Organization**

* When the two or more records are stored in the same file, it is known as clusters. These files will have two or more tables in the same data block, and key attributes which are used to map these tables together are stored only once.
* This method reduces the cost of searching for various records in different files.
* The cluster file organization is used when there is a frequent need for joining the tables with the same condition. These joins will give only a few records from both tables. In the given example, we are retrieving the record for only particular departments. This method can't be used to retrieve the record for the entire department.

****

## **Pros of Cluster file organization**

* The cluster file organization is used when there is a frequent request for joining the tables with same joining condition.
* It provides the efficient result when there is a 1:M mapping between the tables.

## **Cons of Cluster file organization**

* This method has the low performance for the very large database.
* If there is any change in joining condition, then this method cannot use. If we change the condition of joining then traversing the file takes a lot of time.
* This method is not suitable for a table with a 1:1 condition.

**3. Explain sparse and dense indices in detail with examples.**

**Ans:**

Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.

The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Indexing in Database is defined based on its indexing attributes. Two main types of indexing methods are:

* Primary Indexing
* Secondary Indexing

## Primary Index in DBMS

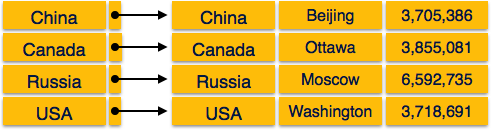
Primary Index is an ordered file which is fixed length size with two fields. The first field is the same a primary key and second, filed is pointed to that specific data block. In the primary Index, there is always one to one relationship between the entries in the index table.

The primary Indexing in DBMS is also further divided into two types.

* Dense Index
* Sparse Index

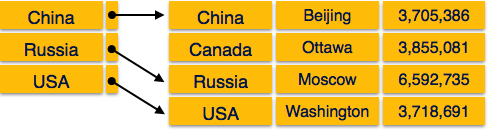
## **Dense Index**

In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.



## **Sparse Index**

In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.



**4.Write in detail about Indexing in DBMS.**

**Ans:**

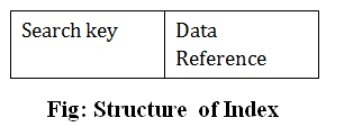
**Indexing in DBMS**

Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.

The index is a type of data structure. It is used to locate and access the data in a database table quickly.

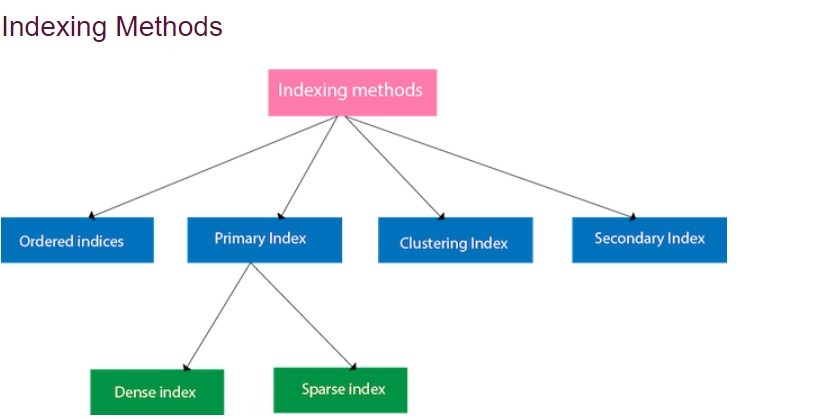
**Index structure:**

Indexes can be created using some database columns.



The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.

The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.



The indexing has various attributes:

**Access Types:** This refers to the type of access such as value based search, range access, etc.

**Access Time**: It refers to the time needed to find particular data element or set of elements.

**Insertion Time**: It refers to the time taken to find the appropriate space and insert a new data.

**Deletion Time**: Time taken to find an item and delete it as well as update the index structure.

**Space Overhead**: It refers to the additional space required by the index.

CAN REFER FROM 5TH ANSWER(if points r needed)

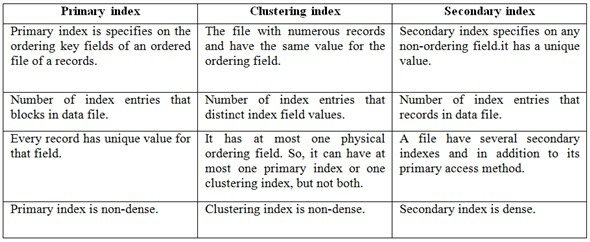
**5 &6. Find the major differences among primary, secondary and clustering indexes?**

**Ans:**

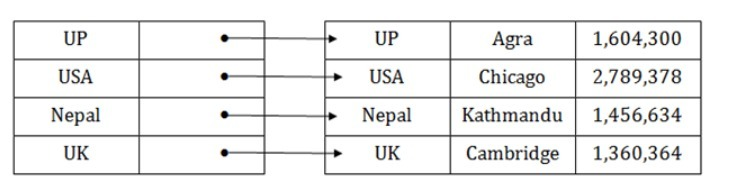
Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing is defined based on its indexing attributes. Indexing can be of the following types −

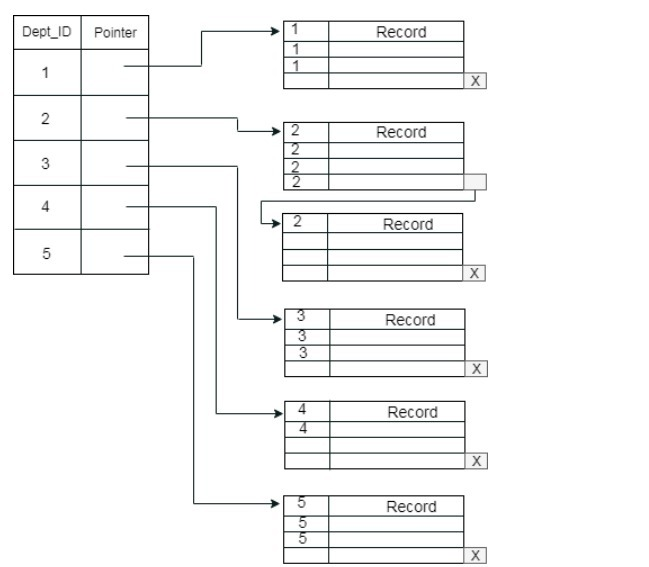
* **Primary Index** − Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
* **Secondary Index** − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
* **Clustering Index** − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.



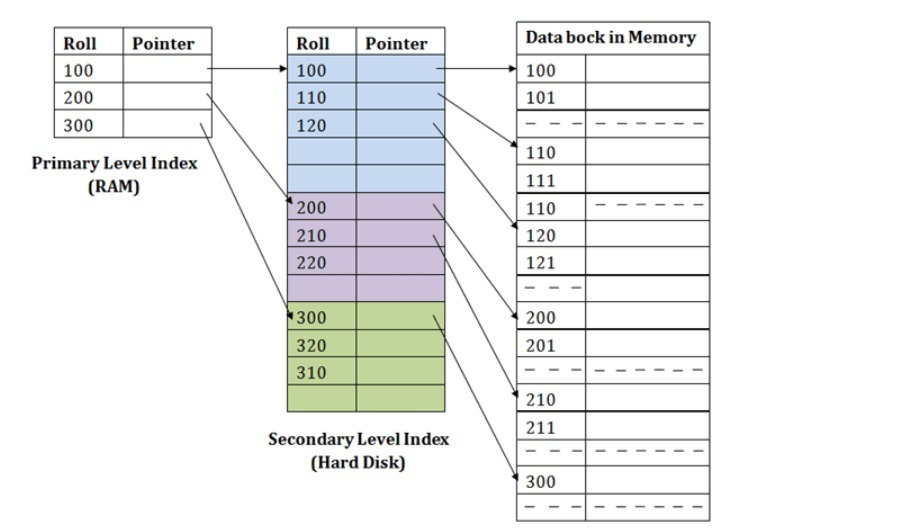
*Primary index*



*Clustering index*



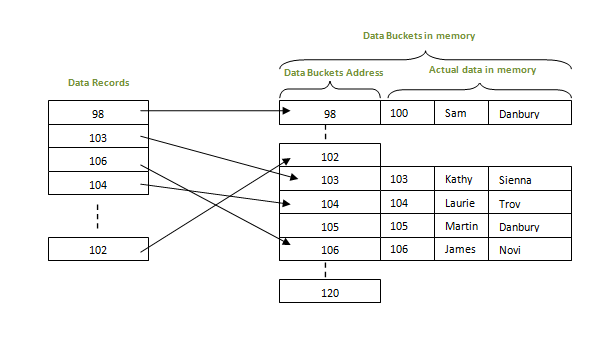
*Secondary index*



**7.** **Discussion detail about Linear Hashing with pros and cons.**

**Ans:**

Hash File organization method is the one where data is stored at the data blocks whose address is generated by using hash function. The memory location where these records are stored is called as data block or data bucket. This data bucket is capable of storing one or more records.



**Definition:** Linear Hashing is a dynamically updateable disk-based index structure which implements a hashing scheme and which grows or shrinks one bucket at a time. The index is used to support exact match queries, i.e., find the record with a given key**.**

**Advantages of Hash:**

* Hash provides better synchronization than other data structures.
* Hash tables are more efficient than search trees or other data structures.
* Hash provides constant time for searching, insertion and deletion operations on average.

**Disadvantages of Hash:**

* Hash is inefficient when there are many collisions.
* Hash collisions are practically not be avoided for large set of possible keys.
* Hash does not allow null values.

**8.** **Explain about B+ Tree and the structure of B+ Tree in detail with suitable examples.**

**Ans:**

**B+ Tree**

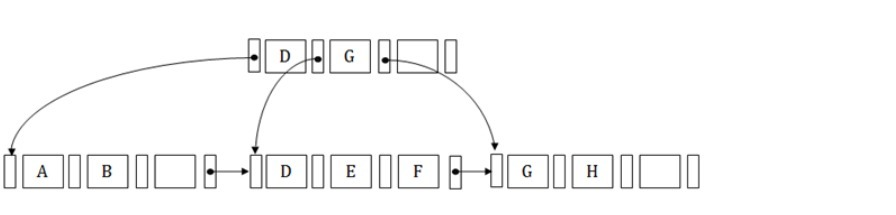
The B+ tree is a balanced binary search tree. It follows a multi-level index format.

In the B+ tree, leaf nodes denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height.

In the B+ tree, the leaf nodes are linked using a link list. Therefore, a B+ tree can support random access as well as sequential access. Structure of B+ Tree

In the B+ tree, every leaf node is at equal distance from the root node. The B+ tree is of the order n where n is fixed for every B+ tree.

It contains an internal node and leaf node.

 **INTERNAL NODE**

An internal node of the B+ tree can contain at least n/2 record pointers except the root node.

At most, an internal node of the tree contains n pointers.

**Leaf node**

The leaf node of the B+ tree can contain at least n/2 record pointers and n/2 key values.

At most, a leaf node contains n record pointer and n key values.

Every leaf node of the B+ tree contains one block pointer P to point to next leaf node.

Example of tree

