1. **What is Data?**

**Data** is a collection of a distinct small unit of information. It can be used in a variety of forms like text, numbers, media, bytes, etc.

1. **What is Database?**

A **database** is an organized collection of data, so that it can be easily accessed and managed. We can organize data into tables, rows, columns, and index it to make it easier to find relevant information. The **main purpose** of the database is to operate a large amount of information by storing, retrieving, and managing data.

1. **What is DBMS?**

**Database Management System (DBMS)** is a software for storing and retrieving users' data while considering appropriate security measures. **DBMS** allows users to create their own databases as per their requirement. It provides an interface between the data and the software application.

1. **Why do we need DBMS?**

A **D**ata **B**ase **M**anagement **S**ystem is a system software for easy, efficient and reliable data processing and management. It can be used for:

* Creation of a database.
* Retrieval of information from the database.
* Updating the database.
* Managing a database.

It provides us with the many functionalities and is more advantageous than the traditional file system in many ways like-

* 1. **Processing Queries and Object Management**.
  2. **Controlling redundancy and inconsistency.**
  3. **Efficient memory management and indexing.**
  4. **Concurrency control and transaction management.**

1. **Traditional File System v/s DBMS-**

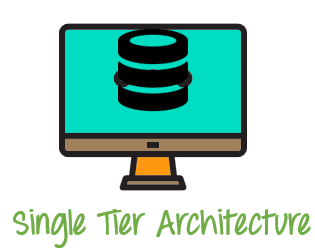
|  |  |
| --- | --- |
| **DBMS** | **File System** |
| DBMS is a collection of data. In DBMS, the user is not required to write the procedures. | File system is a collection of data. In this system, the user has to write the procedures for managing the database. |
| DBMS gives an abstract view of data that hides the details. | File system provides the detail of the data representation and storage of data. |
| DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from the system failure. | File system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will lost. |
| DBMS provides a good protection mechanism. | It is very difficult to protect a file under the file system. |
| DBMS contains a wide variety of sophisticated techniques to store and retrieve the data. | File system can't efficiently store and retrieve the data. |
| DBMS takes care of Concurrent access of data using some form of locking. | In the File system, concurrent access has many problems like redirecting the file while other deleting some information or updating some information. |

1. **What is the function of DBA(Database Administrator)?**

Functions of a **DBA** include-

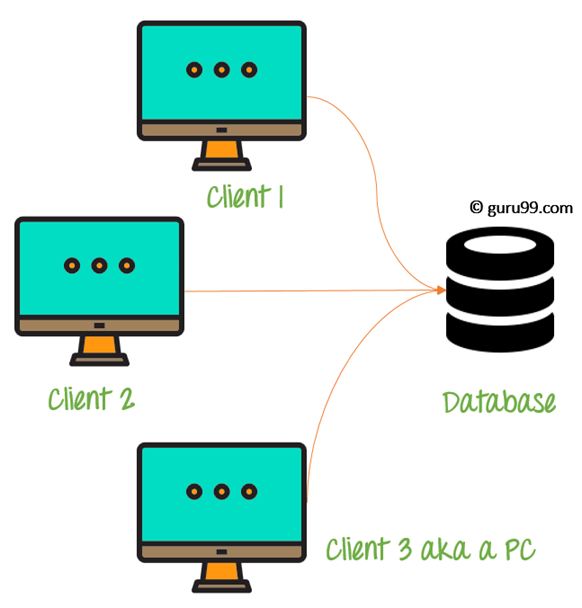
* **Schema deﬁnition**.The DBA creates the original database schema by executing a set of data deﬁnition statements in the DDL.
* **Storage structure and access-method deﬁnition.**
* **Schema and physical-organization modiﬁcation**.The DBA carries out changes to the schema and physical organization to reﬂect the changing needs of the organization, or to alter the physical organization to improve performance.
* **Granting of authorization for data access.** By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults when ever someone attempts to access the data in the system.
* **Routine maintenance**. Examples of the database administrator’s routine maintenance activities are:

1. Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as ﬂooding.
2. Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
3. Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.
4. **What are DBMS Architecture?**
   1. **1 Tier Architecture** in DBMS is the simplest architecture of Database in which the client, server, and Database all reside on the same machine. A simple one tier architecture example would be anytime you install a Database in your system and access it to practice SQL queries. But such architecture is rarely used in production.

[](https://www.guru99.com/images/1/091318_0745_DBMSArchite1.png)

1-Tier Architecture Diagram

* 1. A **2 Tier Architecture** in DBMS is a Database architecture where the presentation layer runs on a client (PC, Mobile, Tablet, etc.), and data is stored on a server called the second tier. Two tier architecture provides added security to the DBMS as it is not exposed to the end-user directly. It also provides direct and faster communication.

[](https://www.guru99.com/images/1/091318_0745_DBMSArchite2.png)

2-Tier Architecture Diagram

In the above 2 Tier client-server architecture of database management system, we can see that one server is connected with clients 1, 2, and 3.

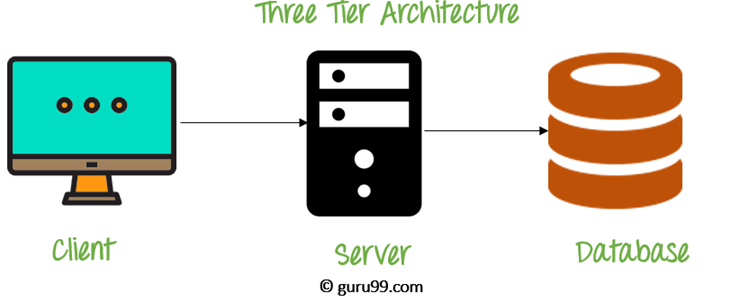
**Two Tier Architecture Example:**

A Contact Management System created using MS- Access.

* 1. A **3-Tier Architecture** in DBMS is the most popular client server architecture in DBMS in which the development and maintenance of functional processes, logic, data access, data storage, and user interface is done independently as separate modules. Three Tier architecture contains a presentation layer, an application layer, and a database server.

3-Tier database Architecture design is an extension of the 2-tier client-server architecture. A 3-tier architecture has the following layers:

1. Presentation layer (your PC, Tablet, Mobile, etc.)
2. Application layer (server)
3. Database Server

[](https://www.guru99.com/images/1/091318_0745_DBMSArchite3.png)

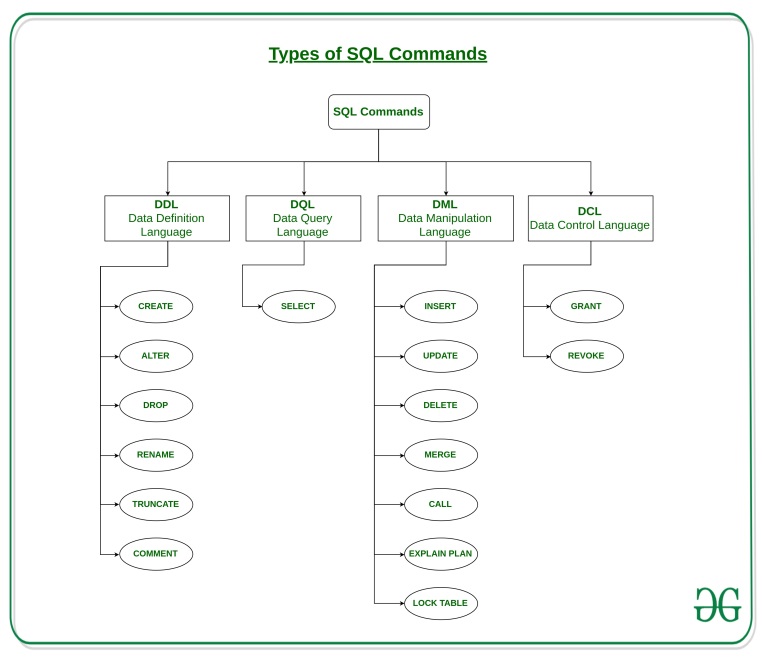
### The goal of Three Tier client-server architecture is:

* To separate the user applications and physical database
* To support DBMS characteristics
* Program-data independence
* Supporting multiple views of the data.

1. **What are Database Language?**

These SQL commands are mainly categorized into four categories as:

1. DDL – Data Definition Language
2. DQl – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language
5. TCL – Transaction Control Language(Commit, Rollback, SavePoint, Set Transaction).



1. **Instances, Schema and Sub-schema**

**Instances -** In simple words, it is the snapshot of the database taken at a particular moment. It can also be described in more significant way as the collection of the information stored in the database at that particular moment. Instance can also be called as the database state or current set of occurrence due the fact that it is information that is present at the current state. Every time we update the state say we insert, delete or modify the value of the data item in the record, it changes from one state to other. At the given time, each schema has its own set of instances.

Lets take an**example** to understand in a much better way,

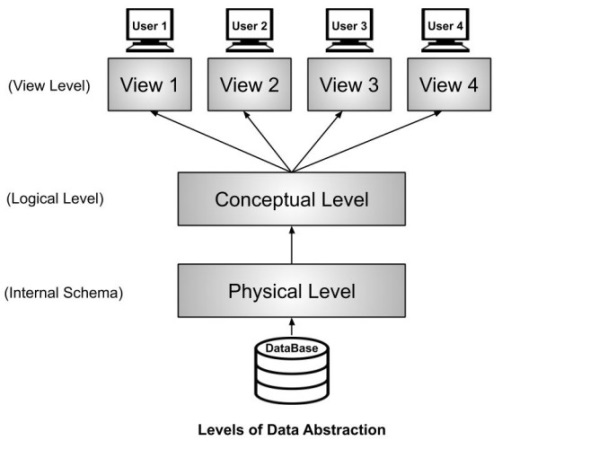
An organization with an employees database will have three different instances such as production that is used to monitor the data right at that moment, per-production that is used to test new functionality prior to release of production and the development that is used by database developers to create new functionality.

**Schema -** It is the overall description or the overall design of the database specified during the database design. Important thing to be remembered here is it should not be changed frequently. Basically, it displays the record types(entity),names of data items(attribute) but not the relation among the files.

**Sub Schema -** It can be defined as the subset or sub-level of schema that has the same properties as the schema. In simple words it is just a effective plan or the schema for the view. Well, it is interesting to note that it provides the users a window through which the user can view only that part of database which is of matter of interest to him. It Identifies subset of areas, sets, records, data names defined in database that is of interest to him. Thus a portion of database can be seen by application programs and different application programs has different view of data.

Quickly we can summarize the above things, information/data in database at particular moment is known as instance, physical arrangement of data as it appears in database can be defined as schema, and the logical view of data as it appears to the application can be called as sub schema.

1. Levels of Data Abstraction.
   1. View Level.
   2. Conceptual Level.
   3. Physical Level.



1. **Referential Integrity-**

Referential Integrity Rule in DBMS is based on Primary and Foreign Key. **Referential integrity** requires that a foreign key must have a matching primary key or it must be null. This constraint is specified between two tables (parent and child); it maintains the correspondence between rows in these tables. It means the reference from a row in one table to another table must be valid.

1. **What is RDBMS? How it is stored into memory?**

An **RDBMS** is a DBMS designed specifically for relational databases. ... An **RDBMS** may also provide a visual representation of the data. In a relational database, relationships between data items are expressed by means of tables.

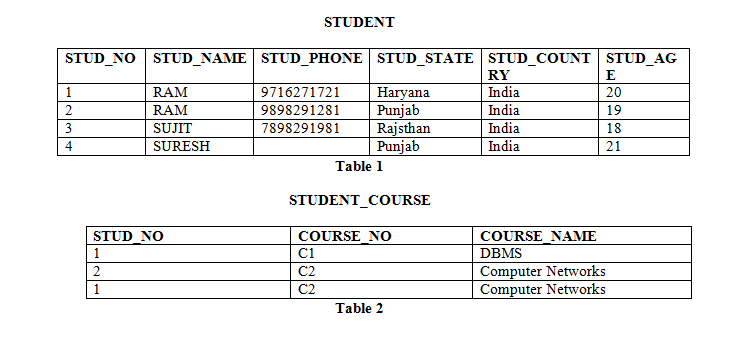
RDBMS generally uses variations of B-Tree. MySQL uses B+ tree to store data indexes. InnoDB stores both data file and index file into the memory while MyISAM stores only the index file. So, in majority of cases MyISAM can be helpful if memory is less than data size. Perhaps most important is the DBMS cache. There are often "hot spots" of data, IOW some subset of the entire database that is read/written more frequently than the rest of the database. DBMSs employ algorithms to keep the most frequently accessed data in its cache. Lookup and read about LRU (least recently used) algorithms for the most common implementation. Since all b-tree searches start from the top (the root node) of the tree, a DBMS cache will almost always have the root nodes of all the indexes commonly used, in cache. (Some indexes are rarely used and therefore might not be in cache.)

1. **Degree of Relation in DBMS-**

There are three types of relationships that can exist between two entities.

An entity-relationship (ER) diagram is created based on these three types, which are listed below:

1. **one-to-one relationship**: In relational database design, a one-to-one (1:1) relationship exists when zero or one instance of entity A can be associated with zero or one instance of entity B, and zero or one instance of entity B can be associated with zero or one instance of entity A. (abbreviated 1:1)
2. **one-to-many relationship**: (abbreviated 1:N) In relational database design, a one-to-many (1:N) relationship exists when, for one instance of entity A, there exists zero, one, or many instances of entity B; but for one instance of entity B, there exists zero or one instance of entity A.
3. **many-to-many relationship**: In relational database design, a many-to-many (M:N) relationship exists when, for one instance of entity A, there exists zero, one, or many instances of entity B; and for one instance of entity B, there exists zero, one, or many instances of entity A. (abbreviated M:N)
4. **Keys in DBMS –**



* **Candidate Key :** The minimal set of attribute which can uniquely identify a tuple is known as candidate key. For Example, STUD\_NO in STUDENT relation. The value of Candidate Key is unique and non-null for every tuple. There can be more than one candidate key in a relation.
* **Super Key:**The set of attributes which can uniquely identify a tuple is known as Super Key. For Example, STUD\_NO, (STUD\_NO, STUD\_NAME) etc.
  + Adding zero or more attributes to candidate key generates super key.
  + A candidate key is a super key but vice versa is not true.
* **Primary Key:** There can be more than one candidate key in relation out of which one can be chosen as the primary key. For Example, STUD\_NO, as well as STUD\_PHONE both, are candidate keys for relation STUDENT but STUD\_NO can be chosen as the primary key (only one out of many candidate keys).
* **Alternate Key:**The candidate key other than the primary key is called an alternate key. For Example, STUD\_NO, as well as STUD\_PHONE both, are candidate keys for relation STUDENT but STUD\_PHONE will be alternate key (only one out of many candidate keys).
* **Foreign Key:** If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute and the relation which refers to the referenced relation is called referencing relation and the corresponding attribute is called referencing attribute. The referenced attribute of the referenced relation should be the primary key for it. For Example, STUD\_NO in STUDENT\_COURSE is a foreign key to STUD\_NO in STUDENT relation.
  + It may be worth noting that unlike, Primary Key of any given relation, Foreign Key can be NULL as well as may contain duplicate tuples i.e. it need not follow uniqueness constraint.

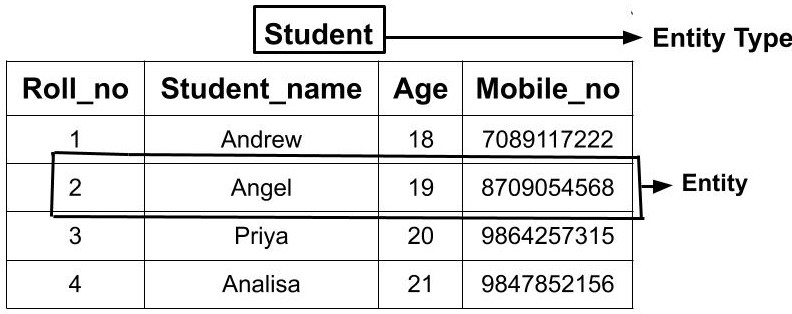
1. **Database Schema –**



A database schema can be divided broadly into two categories −

* **Physical Database Schema** − This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
* **Logical Database Schema** − This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

1. **What is Entity, Entity-Type and Entity-Set?**



#### Entity: An entity is a real-world thing which can be distinctly identified like a person, place or a concept. It is an object which is distinguishable from others. If we cannot distinguish it from others then it is an object but not an entity. An entity can be of two types:

**Tangible Entity:** Tangible Entities are those entities which exist in the real world physically. **Example:** Person, car, etc.

**Intangible Entity:** Intangible Entities are those entities which exist only logically and have no physical existence. **Example:** Bank Account, etc.

**Types of Entity Type-**

* ***Strong Entity Type:***  Strong entity are those entity types which has a key attribute. The primary key helps in identifying each entity uniquely. It is represented by a rectangle. In the above example, Roll\_no identifies each element of the table uniquely and hence, we can say that STUDENT is a strong entity type.
* ***Weak Entity Type***: Weak entity type doesn't have a key attribute. Weak entity type can't be identified on its own. It depends upon some other strong entity for its distinct identity. This can be understood with a real-life example. There can be children only if the parent exits. There can be no independent existence of children. There can be a room only if building exits. There can be no independent existence of a room. A weak entity is represented by a double outlined rectangle. The relationship between a weak entity type and strong entity type is called an identifying relationship and shown with a double outlined diamond instead of a single outlined diamond.

#### Entity Set - Entity Set is a collection of entities of the same entity type. In the above example of STUDENT entity type, a collection of entities from the Student entity type would form an entity set. We can say that entity type is a superset of the entity set as all the entities are included in the entity type.

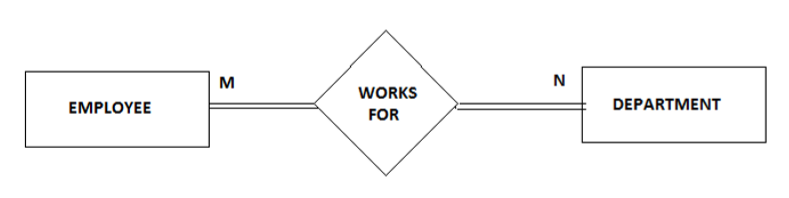
1. *Participation Constraint* – It specifies the existence of an entity when it is related to another entity in a relationship type. It is also called minimum cardinality constraint.

This constraint specifies the number of instances of an entity that can participate in a relationship type.

There are two types of Participation constraint −

## *Total Participation*

Each entity in the entity set is involved in at least one relationship in a relationship set i.e. the number of relationship in every entity is involved is greater than 0.

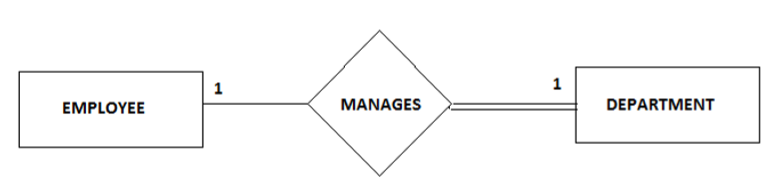


Consider two entities Employee and Department related via Works\_For relationship. Now, every Employee works in at least one department therefore an Employee entity exist if it has at least one Works\_For relationship with Department entity. Thus the participation of Employee in Works\_For is total relationship.  
  
Total Participation is represented by double line in ER diagram.

## *Partial Participation*

Each entity in entity set may or may not occur in at least one relationship in a relationship set.

For example: Consider two entities Employee and Department and they are related to each other via Manages relationship. An Employee must manage a Department, he or she could be the head of the department. But not every Employee in the company manages the department. So, participation of employee in the Manages relationship type is partial i.e. only a particular set of Employees will manage the Department but not all.



1. **What is Functional Dependency?**

**Functional Dependency (FD)** is a constraint that determines the relation of one attribute to another attribute in a Database.

### ****Types of Functional dependencies in DBMS:****

* **Trivial functional dependency** - In **Trivial Functional Dependency**, a dependent is always a subset of the determinant.  
  i.e. If **X → Y** and **Y is the subset of X**, then it is called trivial functional dependency. For eg - AB**→A,** AB**→B,** AB**→AB.**
* **Non-Trivial functional dependency** – If there is atleast one attribute in the RHS i.e., not a part of LHS. For eg - AB**→C,** AB**→AC**
* **Multivalued functional dependency** - In **Multivalued functional dependency**, entities of the dependent set are **not dependent** **on each other.**  
  i.e. If **a → {b, c}** and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency. For eg - roll\_no → {name, age}**is a multivalued functional dependency, since the dependents **name** & **age** are **not dependent** on each other(i.e. **name → age**or**age → name doesn’t exist !**)
* **Transitive functional dependency** - In transitive functional dependency, dependent is indirectly dependent on determinant.  
  i.e. If **a → b** & **b → c**, then according to axiom of transitivity, **a → c**. This is a **transitive functional dependency.**
* **Partial Functional dependency** – In a relation if a proper subset of a candidate key is finding some non-prime(not a part of key) attribute then it is Known as Partial Dependency. For eg – AB is a candidate Key in a relation then B**→C where B is a part of Candidate Key and C is a non-prime Attribute.**

1. What is Normalization?

**Normalization** is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalization in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

1. What are Anomalies in DBMS?

1NF is basically just "don't keep too much data in a single column", so I think that 2NF and 3NF are the primary fix for all 3 database anomalies, since both 2NF and 3NF involve breaking out items into their own tables:

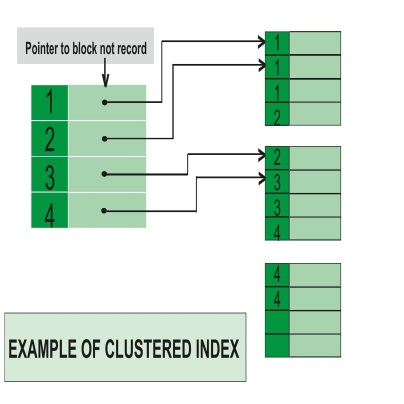
1. **Insertion anomaly**: If you have one big enrollment table that includes both "class" and "student" data (neither of which exists elsewhere), then you can't enter a new (empty) course without at least one corresponding student (because the table is a record of your enrollments). So, apply 2NF and create separate tables for classes, students, and make your original enrollment table link to both by ClassID and StudentID. Now you can enter new classes with no students, and new students with no classes.
2. **Deletion anomaly**: Same as above, if each row of your original enrollment table contains the full details of the student and the full details of the class they are enrolled in, then removing the last enrolled student for a class *removes the last bit of information about that class*. The solution is the same, apply 2NF to make separate tables, so that students can be enrolled or unenrolled without losing class information.
3. **Update anomaly**: Same as above, using the single-table method, updating information (say, the room number) for a class with multiple students enrolled might lead to a situation where some rows have been new information and other rows have the old. Applying 2NF as above is again the solution, so that class data is changed in only one place (the classes table).
4. Normalization- (Refer Knowledge Gate Videos)
   1. 1NF.
   2. 2NF.
   3. 3NF.
   4. BCNF.
5. Difference between Clustered and Non-Clustered Index.

**1. Clustered Index :**  
Clustered index is created only when both the following conditions satisfy –

1. The data or file, that you are moving into secondary memory should be in sequential or sorted order.
2. There should be non key value, means the data should be unique.

Whenever you apply clustered indexing in a table, it will perform sorting in that table only. You can create only one clustered index in a table like primary key. Clustered index is as same as dictionary where the data is arranged by alphabetical order.

In clustered index, index contains pointer to block but not direct data.

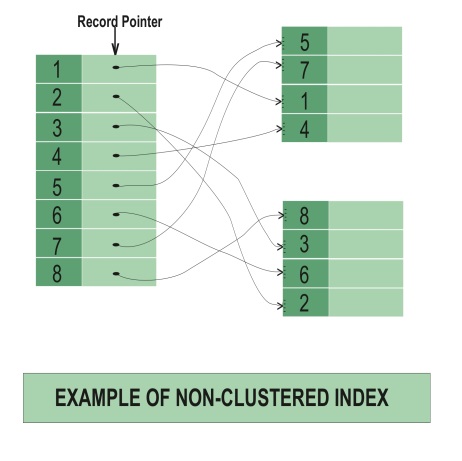


If we apply primary key to any column, then automatically it will become clustered index.

**2. Non-clustered Index :**  
Non-Clustered Index is similar to the index of a book. The index of a book consists of a chapter name and page number, if you want to read any topic or chapter then you can directly go to that page by using index of that book. No need to go through each and every page of a book.

The data is stored in one place, and index is stored in another place. Since, the data and non-clustered index is stored separately, then you can have multiple non-clustered index in a table.

In non-clustered index, index contains the pointer to data.



**Difference between Clustered and Non-clustered index :**

| CLUSTERED INDEX | NON-CLUSTERED INDEX |
| --- | --- |
| Clustered index is faster. | Non-clustered index is slower. |
| Clustered index requires less memory for operations. | Non-Clustered index requires more memory for operations. |
| In clustered index, index is the main data. | In Non-Clustered index, index is the copy of data. |
| A table can have only one clustered index. | A table can have multiple non-clustered index. |
| Clustered index has inherent ability of storing data on the disk. | Non-Clustered index does not have inherent ability of storing data on the disk. |
| Clustered index store pointers to block not data. | Non-Clustered index store both value and a pointer to actual row that holds data. |
| In Clustered index leaf nodes are actual data itself. | In Non-Clustered index leaf nodes are not the actual data itself rather they only contains included columns. |
| In Clustered index, Clustered key defines order of data within table. | In Non-Clustered index, index key defines order of data within index. |
| A Clustered index is a type of index in which table records are physically reordered to match the index. | A Non-Clustered index is a special type of index in which logical order of index does not match physical stored order of the rows on disk. |

1. What is Transaction?

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

1. What are ACID Properties?

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

* **Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
* **Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
* **Durability** − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
* **Isolation** − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

1. Advantages of Concurrency –
   1. Reduce Waiting Time.
   2. Reduce Response Time.
   3. Resource utilization.
   4. Efficiency.
2. What is serializibility?

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleaved with some other transaction.

* **Schedule** − A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.
* **Serial Schedule** − It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

In a multi-transaction environment, serial schedules are considered as a benchmark. The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion. This execution does no harm if two transactions are mutually independent and working on different segments of data; but in case these two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.

To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.

## Equivalence Schedules

An equivalence schedule can be of the following types −

### Result Equivalence

If two schedules produce the same result after execution, they are said to be result equivalent. They may yield the same result for some value and different results for another set of values. That's why this equivalence is not generally considered significant.

**View Equivalence**

Two schedules would be view equivalence if the transactions in both the schedules perform similar actions in a similar manner.

For example −

* If T reads the initial data in S1, then it also reads the initial data in S2.
* If T reads the value written by J in S1, then it also reads the value written by J in S2.
* If T performs the final write on the data value in S1, then it also performs the final write on the data value in S2.

### Conflict Equivalence

Two schedules would be conflicting if they have the following properties −

* Both belong to separate transactions.
* Both accesses the same data item.
* At least one of them is "write" operation.

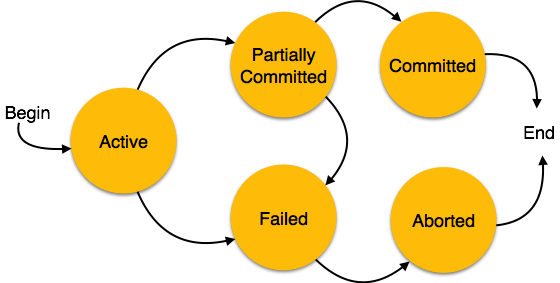
Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if −

* Both the schedules contain the same set of Transactions.
* The order of conflicting pairs of operation is maintained in both the schedules.

**Note** − View equivalent schedules are view serializable and conflict equivalent schedules are conflict serializable. All conflict serializable schedules are view serializable too.

## States of Transactions

A transaction in a database can be in one of the following states −

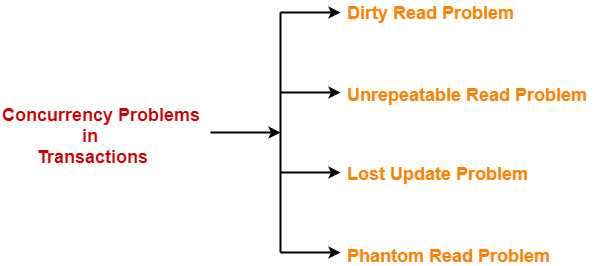


* **Active** − In this state, the transaction is being executed. This is the initial state of every transaction.
* **Partially Committed** − When a transaction executes its final operation, it is said to be in a partially committed state.
* **Failed** − A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
* **Aborted** − If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts −
  + Re-start the transaction
  + Kill the transaction
* **Committed** − If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

1. Problems on Concurrent Transaction –

When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems. Such problems are called as **concurrency problems**.

The concurrency problems are-



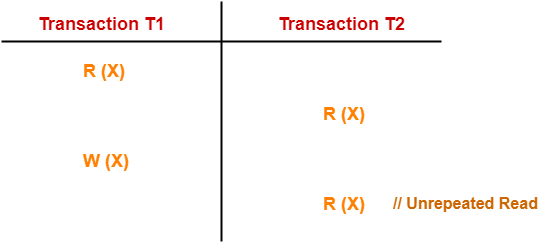
***Dirty Read Problem*** - Reading the data written by an uncommitted transaction is called as dirty read. This read is called as dirty read because-

* There is always a chance that the uncommitted transaction might roll back later.
* Thus, uncommitted transaction might make other transactions read a value that does not even exist.
* This leads to inconsistency of the database.

## *****Unrepeatable Read Problem-*****

 This problem occurs when a transaction gets to read unrepeated i.e. different values of the same variable in its different read operations even when it has not updated its value.

### ****Example-****



Here,

1. T1 reads the value of X (= 10 say).
2. T2 reads the value of X (= 10).
3. T1 updates the value of X (from 10 to 15 say) in the buffer.
4. T2 again reads the value of X (but = 15).

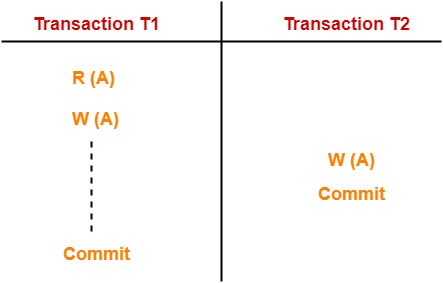
In this example,

* T2 gets to read a different value of X in its second reading.
* T2 wonders how the value of X got changed because according to it, it is running in isolation.

## *****Lost Update Problem-*****

This problem occurs when multiple transactions execute concurrently and updates from one or more transactions get lost.

### ****Example-****



Here,

1. T1 reads the value of A (= 10 say).
2. T2 updates the value to A (= 15 say) in the buffer.
3. T2 does blind write A = 25 (write without read) in the buffer.
4. T2 commits.
5. When T1 commits, it writes A = 25 in the database.

In this example,

* T1 writes the over written value of X in the database.
* Thus, update from T1 gets lost.

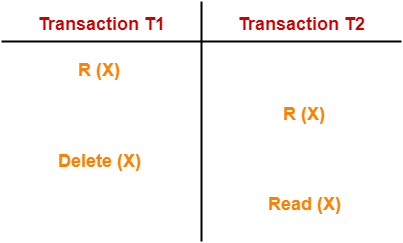
### ****NOTE-****

* This problem occurs whenever there is a write-write conflict.
* In write-write conflict, there are two writes one by each transaction on the same data item without any read in the middle.

## *****Phantom Read Problem-*****

This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later, it finds that the variable does not exist.

### ****Example-****



Here,

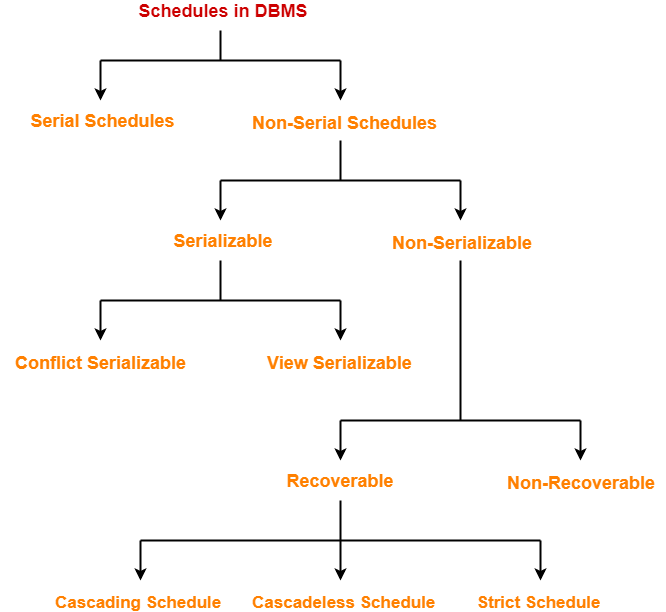
1. T1 reads X.
2. T2 reads X.
3. T1 deletes X.
4. T2 tries reading X but does not find it.

In this example,

* T2 finds that there does not exist any variable X when it tries reading X again.
* T2 wonders who deleted the variable X because according to it, it is running in isolation.

## ****Types of Schedules-****

In DBMS, schedules may be classified as-



1. Non-serial Schedule –

**Conflict Serializable:** A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

**Conflicting operations:** Two operations are said to be conflicting if all conditions satisfy:

* They belong to different transactions
* They operate on the same data item
* At Least one of them is a write operation

**View Serializable:** Two schedules S1 and S2 are said to be view-equivalent if below conditions are satisfied :

1. **Initial Read**  
   If a transaction T1 reading data item A from database in S1 then in S2 also T1 should read A from database.

T1 T2 T3

-------------------

R(A)

W(A)

R(A)

R(B)

1. **Final Write operation**  
   If a transaction T1 updated A at last in S1, then in S2 also T1 should perform final write operations.

T1 T2 T1 T2

------------ ---------------

R(A) R(A)

W(A) W(A)

W(A) W(A)

Above two schedules are not view-equivalent as Final write operation in S1 is done by T1 while in S2 done by T2.

1. **Updated Read**   
   If Ti is reading A which is updated by Tj in S1 then in S2 also Ti should read A which is updated by Tj.

T1 T2 T3 T1 T2 T3

------------------- ----------------

W(A) W(A)

W(A) R(A)

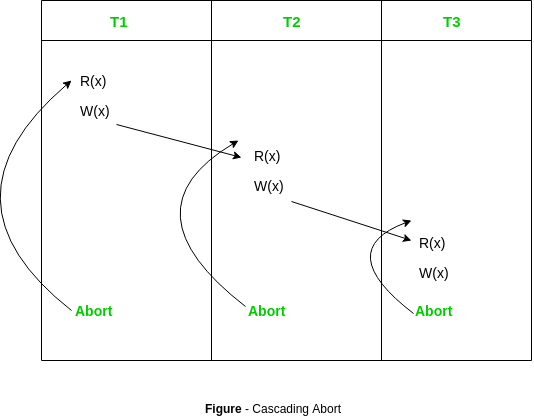
R(A) W(A)

Above two schedule are not view-equivalent as in S1 :T3 is reading A updated by T2, in S2 T3 is reading A updated by T1.

**Non-Serializable:**

[Recoverable Schedule:](https://www.geeksforgeeks.org/recoverability-in-dbms/) Schedules in which transactions commit only after all transactions whose changes they read commit are called recoverable schedules. In other words, if some transaction Tj is reading value updated or written by some other transaction Ti, then the commit of Tj must occur after the commit of Ti.

[Cascading Schedule:](https://www.geeksforgeeks.org/recoverability-in-dbms/)Also called Avoids cascading aborts/rollbacks (ACA). When there is a failure in one transaction and this leads to the rolling back or aborting other dependent transactions, then such scheduling is referred to as Cascading rollback or cascading abort. Example:



[Cascadeless Schedule:](https://www.geeksforgeeks.org/cascadeless-in-dbms/) It works on Dirty Read but does not works on Blind Write. Schedules in which transactions read values only after all transactions whose changes they are going to read commit are called cascadeless schedules. Avoids that a single transaction abort leads to a series of transaction rollbacks. A strategy to prevent cascading aborts is to disallow a transaction from reading uncommitted changes from another transaction in the same schedule.

In other words, if some transaction Tj wants to read value updated or written by some other transaction Ti, then Tj must read it after the commit of Ti.

[Strict Schedule:](https://www.geeksforgeeks.org/cascadeless-in-dbms/) A schedule is strict if for any two transactions Ti, Tj, if a write operation of Ti precedes a conflicting operation of Tj (either read or write), then the commit or abort event of Ti also precedes that conflicting operation of Tj.  
In other words, Tj can read or write updated or written value of Ti only after Ti commits/aborts. (Jab tak ek transaction commit nhi kare tab tak koi dusra transaction uss data item pe read or write nhi kar sakta).

1. Concurrency Control Technique –

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.