**Database Management System**

**Database:**Database is a collection of inter-related data which helps in efficient retrieval, insertion and deletion of data from database and organizes the data in the form of tables, views, schemas, reports etc.

**Database Management System:**The software which is used to manage database is called Database Management System (DBMS). For Example, MySQL, Oracle etc.

DBMS allows users the following tasks:

**Data Definition:** It helps in creation, modification and removal of definitions that define the organization of data in database.

**Data Updation:** It helps in insertion, modification and deletion of the actual data in the database.

**Data Retrieval:** It helps in retrieval of data from the database which can be used by applications for various purposes.

**User Administration:** It helps in registering and monitoring users, enforcing data security, monitoring performance, maintaining data integrity, dealing with concurrency control and recovering information corrupted by unexpected failure.

Database management systems were developed to handle the following difficulties of typical File-processing systems supported by conventional operating systems.

1. Data redundancy and inconsistency

2. Difficulty in accessing data

3. Data isolation – multiple files and formats

4. Integrity problems

5. Atomicity of updates

6. Concurrent access by multiple users

7. Security problems

# DBMS Architecture 2-Level, 3-Level

**Two-tier architecture:**   
The two-tier architecture is similar to a basic **client-server** model. The application at the client end directly communicates with the database at the server-side. APIs like ODBC, JDBC are used for this interaction. The server side is responsible for providing query processing and transaction management functionalities. On the client-side, the user interfaces and application programs are run.

An advantage of this type is that maintenance and understanding are easier, compatible with existing systems. However, this model gives poor performance when there are a large number of users.



**Three Tier architecture:**   
In this type, there is another layer between the client and the server. The client does not directly communicate with the server. Instead, it interacts with an application server which further communicates with the database system and then the query processing and transaction management takes place.

**Advantages:** 

* **Enhanced scalability** due to distributed deployment of application servers. Now, individual connections need not be made between client and server.
* **Data Integrity** is maintained. Since there is a middle layer between client and server, data corruption can be avoided/removed.
* **Security** is improved. This type of model prevents direct interaction of the client with the server thereby reducing access to unauthorized data.

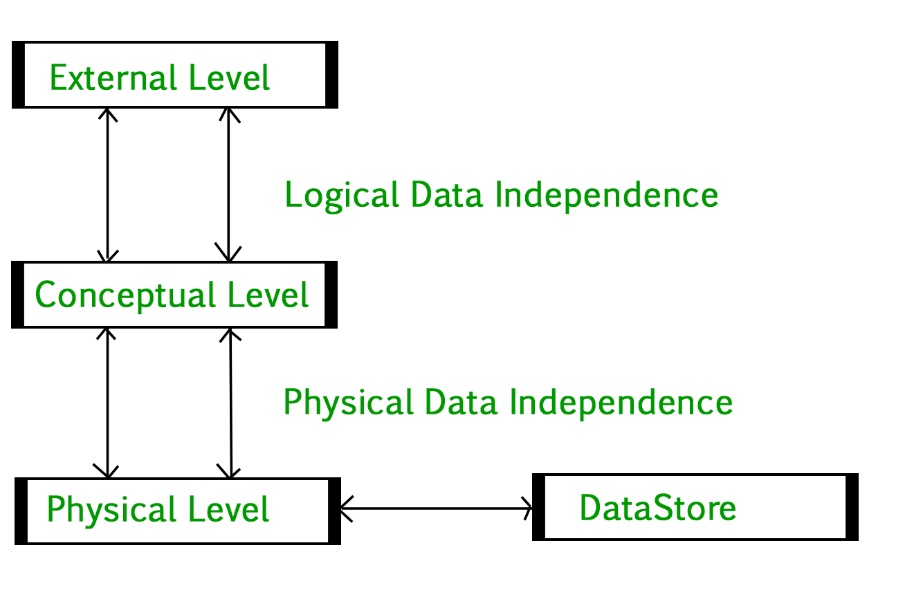
**Disadvantages**:   
Increased complexity of implementation and communication.

# 3-Tier Architecture in DBMS

**Physical Level:** At the physical level, the information about the location of database objects in the data store is kept. In simple terms, physical level of a database describes how the data is being stored in secondary storage devices like disks and tapes.

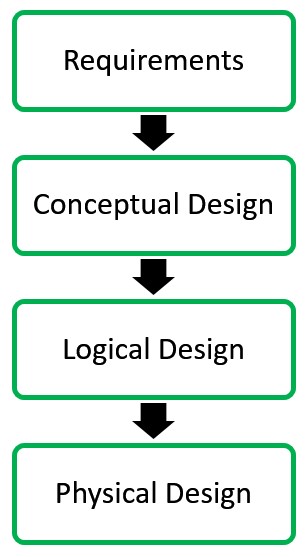
**Conceptual Level:**At conceptual level, data is represented in the form of various database tables. For Example, STUDENT database may contain STUDENT and COURSE tables which will be visible to users but users are unaware of their storage. Also referred as logical schema, it describes what kind of data is to be stored in the database.

**External Level:** An external level specifies a view of the data in terms of conceptual level tables. For Example, FACULTY of a university is interested in looking course details of students, STUDENTS are interested in looking at all details related to academics, accounts, courses and hostel details as well. So, different views can be generated for different users. The main focus of external level is data abstraction.



**Phases of database design**

DBMS software which consists of following steps shown below:



**Conceptual Design:**The requirements of database are captured using high level conceptual data model. For Example, the ER model is used for the conceptual design of the database.

**Logical Design:**Logical Design represents data in the form of relational model. ER diagram produced in the conceptual design phase is used to convert the data into the Relational Model.

**Physical Design:** In physical design, data in relational model is implemented using commercial DBMS like Oracle, DB2.

# What is Data Independence

Data independence is the ability to modify the scheme without affecting the programs and the application to be rewritten. Data is separated from the programs, so that the changes made to the data will not affect the program execution and the application.

There are two levels of data independence based on three levels of abstraction. These are as follows −

* Physical Data Independence
* Logical Data Independence

## **Physical Data Independence**

Physical Data Independence means changing the physical level without affecting the logical level or conceptual level. Using this property, we can change the storage device of the database without affecting the logical schema.

The changes in the physical level may include changes using the following −

* A new storage device like magnetic tape, hard disk, etc.
* A new data structure for storage.
* A different data access method or using an alternative files organization technique.
* Changing the location of the database.

### **Logical Data Independence**

Logical Data Independence is defined as the ability to make changes in the structure of the middle level of the Database Management System (DBMS) without affecting the highest-level schema or application programs. Hence, modification in the logical level should not result in any changes in the view levels or application programs.

**Example –**   
Changes in the middle level (logical level) are: adding new attributes to a relation, deleting existing attributes of the relation, etc. Ideally, we would not want to change any application or programs that do not require to use of the modified attribute.

# Keys

# Keys play an important role in the relational database. It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

**For example,** ID is used as a key in the Student table because it is unique for each student. In the PERSON table, passport number, license number, SSN are keys since they are unique for each person.

## **Types of keys:**

● **Super Key** - A super key is a set of attributes that can identify each tuple uniquely in the given relation. A super key may consist of any number of attributes.

● **Candidate Key** - A set of minimal attribute(s) that can identify each tuple uniquely in the given relation is called a candidate key.

● **Primary Key** - A primary key is a candidate key that the database designer selects while designing the database. Primary Keys are unique and **NOT NULL**.

● **Alternate Key** - Candidate keys that are left unimplemented or unused after implementing the primary key are called as alternate keys.

● **Foreign Key** - An attribute ‘X’ is called as a foreign key to some other attribute ‘Y’ when its values are dependent on the values of attribute ‘Y’. The relation in which attribute ‘Y’ is present is called as the referenced relation. The relation in which attribute ‘X’ is present is called as the referencing relation.

● **Composite Key** - A primary key composed of multiple attributes and not just a single attribute is called a composite key.

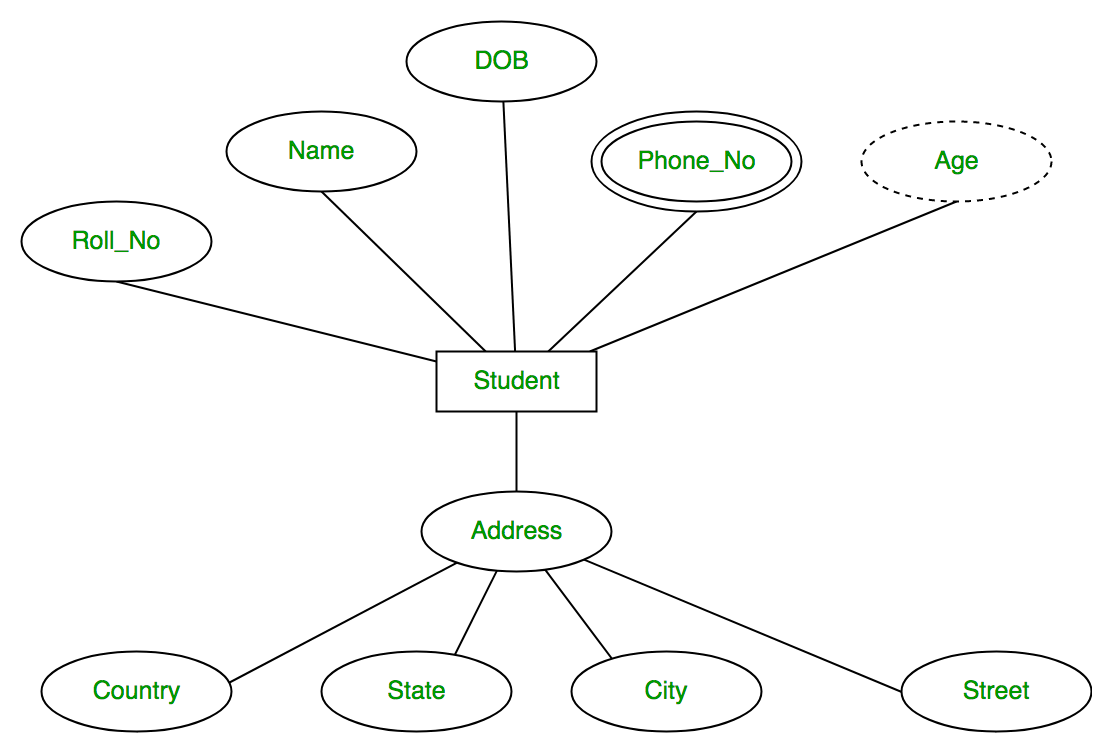
● **Unique Key** - It is unique for all the records of the table. Once assigned, its value cannot be changed i.e. it is non-updatable. It may have a NULL value.

**ER diagram:**

● ER diagram or Entity Relationship diagram is a conceptual model that gives the graphical representation of the logical structure of the database.

● It shows all the constraints and relationships that exist among the different components.

● An ER diagram is mainly composed of following three components- Entity Sets, Attributes and Relationship Set.



● Roll\_no is a primary key that can identify each entity uniquely.

● Thus, by using a student's roll number, a student can be identified uniquely

**Entity Set:**

An entity set is a set of the same type of entities.

● **Strong Entity Set:**

- A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities.

- In other words, a primary key exists for a strong entity set.

- Primary key of a strong entity set is represented by underlining it.

● **Weak Entity Set:**

- A weak entity set is an entity set that does not contain sufficient attributes to uniquely identify its entities.

- In other words, a primary key does not exist for a weak entity set.

- However, it contains a partial key called a discriminator.

- Discriminator can identify a group of entities from the entity set.

- Discriminator is represented by underlining with a dashed line.

**Relationship:**

A relationship is defined as an association among several entities.

● **Unary Relationship Set** - Unary relationship set is a relationship set where only one entity set participates in a relationship set.

● **Binary Relationship Set** - Binary relationship set is a relationship set where two entity sets participate in a relationship set.

● **Ternary Relationship Set** - Ternary relationship set is a relationship set where three entity sets participate in a relationship set.

● **N-ary Relationship Set** - N-ary relationship set is a relationship set where ‘n’ entity sets participate in a relationship set.

**Cardinality Constraint:**

Cardinality constraint defines the maximum number of relationship instances in which an entity can participate.

● **One-to-One Cardinality** - An entity in set A can be associated with at most one entity in set B. An entity in set B can be associated with at most one entity in set A.

● **One-to-Many Cardinality** - An entity in set A can be associated with any number (zero or more) of entities in set B. An entity in set B can be associated with at most one entity in set A.

● **Many-to-One Cardinality** - An entity in set A can be associated with at most one entity in set B. An entity in set B can be associated with any number of entities in set A.

● **Many-to-Many Cardinality** - An entity in set A can be associated with any number (zero or more) of entities in set B. An entity in set B can be associated with any number (zero or more) of entities in set A.

**Attributes:**

Attributes are the descriptive properties which are owned by each entity of an Entity Set.

Types of Attributes:

● **Simple Attributes** - Simple attributes are those attributes which cannot be divided further. Ex. Age

● **Composite Attributes** - Composite attributes are those attributes which are composed of many other simple attributes. Ex. Name, Address

● **Multi Valued Attributes** - Multi valued attributes are those attributes which can take more than one value for a given entity from an entity set. Ex. Mobile No, Email ID

● **Derived Attributes** - Derived attributes are those attributes which can be derived from other attribute(s). Ex. Age can be derived from DOB.

● **Key Attributes** - Key attributes are those attributes which can identify an entity uniquely in an entity set. Ex. Roll No.

**Functional Dependency:**

In any relation, a functional dependency α → β holds if- Two tuples having same value of attribute α also have same value for attribute β. Types of Functional Dependency:

● **Trivial Functional Dependencies** –

o A functional dependency X → Y is said to be trivial if and only if Y ⊆ X. o Thus, if RHS of a functional dependency is a subset of LHS, then it is called a trivial functional dependency.

● **Non-Trivial Functional Dependencies –**

o A functional dependency X → Y is said to be non-trivial if and only if Y ⊄ X. o Thus, if there exists at least one attribute in the RHS of a functional dependency that is not a part of LHS, then it is called a non-trivial functional dependency.

**Constraints:**

Relational constraints are the restrictions imposed on the database contents and operations. They ensure the correctness of data in the database.

● **Domain Constraint** - Domain constraint defines the domain or set of values for an attribute. It specifies that the value taken by the attribute must be the atomic value from its domain.

● **Tuple Uniqueness Constraint** - Tuple Uniqueness constraint specifies that all the tuples must be necessarily unique in any relation.

● **Key Constraint** - All the values of the primary key must be unique. The value of the primary key must not be null.

**● Entity Integrity Constraint** - Entity integrity constraint specifies that no attribute of primary key must contain a null value in any relation.

● **Referential Integrity Constraint** - It specifies that all the values taken by the foreign key.

**Decomposition of a Relation:**

The process of breaking up or dividing a single relation into two or more sub relations is called the decomposition of a relation.

**Properties of Decomposition:**

● **Lossless Decomposition –**

Lossless decomposition ensures No information is lost from the original relation during decomposition.

o When the sub relations are joined back, the same relation is obtained that was decomposed.

● **Dependency Preservation –**

**Dependency preservation ensures**

o None of the functional dependencies that hold on the original relation are lost.

o The sub relations still hold or satisfy the functional dependencies of the original relation.

**Types of Decomposition:**

● **Lossless Join Decomposition:**

o Consider there is a relation R which is decomposed into sub relations R1, R2, …., Rn.

o This decomposition is called lossless join decomposition when the join of the sub relations results in the same relation R that was decomposed.

o For lossless join decomposition, we always have- R1 ⋈ R2 ⋈ R3 ……. ⋈ Rn = R where ⋈ is a natural join operator

**● Lossy Join Decomposition:**

o Consider there is a relation R which is decomposed into sub relations R1, R2, …., Rn.

o This decomposition is called lossy join decomposition when the join of the sub relations does not result in the same relation R that was decomposed.

o For lossy join decomposition, we always have- R1 ⋈ R2 ⋈ R3 ……. ⋈ Rn ⊃ R where ⋈ is a natural join operator

**Normalization:**

In DBMS, database normalization is a process of making the database consistent by-

● Reducing the redundancies

● Ensuring the integrity of data through lossless decomposition Normal Forms:

● **First Normal Form (1NF)** - A given relation is called in First Normal Form (1NF) if each cell of the table contains only an atomic value i.e. if the attribute of every tuple is either single valued or a null value.

● **Second Normal Form (2NF)** - A given relation is called in Second Normal Form (2NF) if and only if o Relation already exists in 1NF.

o No partial dependency exists in the relation. A → B is called a partial dependency if and only if- A is a subset of some candidate key and B is a non-prime attribute.

● **Third Normal Form (3NF)** - A given relation is called in Third Normal Form (3NF) if and only if o Relation already exists in 2NF.

o No transitive dependency exists for non-prime attributes. A → B is called a transitive dependency if and only if- A is not a super key and B is a non-prime attribute.

**● Boyce-Codd Normal Form** - A given relation is called in BCNF if and only if o Relation already exists in 3NF.

o For each non-trivial functional dependency ‘A → B’, A is a super key of the relation

**Transaction:**

Transaction is a single logical unit of work formed by a set of operations. Operations in Transaction:

● **Read Operation** - Read(A) instruction will read the value of ‘A’ from the database and will store it in the buffer in main memory.

● **Write Operation** – Write(A) will write the updated value of ‘A’ from the buffer to the database.

**Transaction States:**

**● Active State –**

o This is the first state in the life cycle of a transaction.

o A transaction is called in an active state as long as its instructions are getting executed. o All the changes made by the transaction now are stored in the buffer in main memory.

**● Partially Committed State –**

o After the last instruction of the transaction has been executed, it enters into a partially committed state.

o After entering this state, the transaction is considered to be partially committed.

o It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.

**● Committed State –**

o After all the changes made by the transaction have been successfully stored into the database, it enters into a committed state.

o Now, the transaction is considered to be fully committed.

**● Failed State –**

o When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it becomes impossible to continue the execution, it enters into a failed state.

**● Aborted State –**

o After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.

o To undo the changes made by the transaction, it becomes necessary to roll back the transaction. o After the transaction has rolled back completely, it enters into an aborted state.

● **Terminated State –**

o This is the last state in the life cycle of a transaction.

o After entering the committed state or aborted state, the transaction finally enters into a terminated state where its life cycle finally comes to an end.

**ACID Properties:**

To ensure the consistency of the database, certain properties are followed by all the transactions occurring in the system. These properties are called as ACID Properties of a transaction.

**● Atomicity –**

o This property ensures that either the transaction occurs completely or it does not occur at all.

o In other words, it ensures that no transaction occurs partially.

**● Consistency –**

o This property ensures that integrity constraints are maintained.

o In other words, it ensures that the database remains consistent before and after the transaction.

**● Isolation –**

o This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.

o The resultant state of the system after executing all the transactions is the same as the state that would be achieved if the transactions were executed serially one after the other.

**● Durability –**

o This property ensures that all the changes made by a transaction after its successful execution are written successfully to the disk.

o It also ensures that these changes exist permanently and are never lost even if there occurs a failure of any kind.

**Schedules:**

The order in which the operations of multiple transactions appear for execution is called as a schedule

**● Serial Schedules –**

o All the transactions execute serially one after the other.

o When one transaction executes, no other transaction is allowed to execute.

o Serial schedules are always- Consistent, Recoverable, Cascadeless and Strict.

**● Non-Serial Schedules –**

o Multiple transactions execute concurrently.

o Operations of all the transactions are inter leaved or mixed with each other.

o Non-serial schedules are not always- Consistent, Recoverable, Cascadeless and Strict.

**Serializability**

● Some non-serial schedules may lead to inconsistency of the database.

● Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

● **Serializable Schedules –**

o If a given non-serial schedule of ‘n’ transactions is equivalent to some serial schedule of ‘n’ transactions, then it is called as a serializable schedule. o Serializable schedules are always- Consistent, Recoverable, Cascadeless and Strict.

**Types of Serializability –**

**● Conflict Serializability -** If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called a conflict serializable schedule.

● **View Serializability -** If a given schedule is found to be viewed as equivalent to some serial schedule, then it is called a view serializable schedule.

**Non-Serializable Schedules –**

● A non-serial schedule which is not serializable is called a non-serializable schedule.

● A non-serializable schedule is not guaranteed to produce the same effect as produced by some serial schedule on any consistent database.

**●** Non-serializable schedules**-** may or may not be consistent, may or may not be recoverable.

● **Irrecoverable Schedules –**

If in a schedule, o A transaction performs a dirty read operation from an uncommitted transaction o And commits before the transaction from which it has read the value then such a schedule is known as an Irrecoverable Schedule.

● **Recoverable Schedules –**

If in a schedule, A transaction performs a dirty read operation from an uncommitted transaction And its commit operation is delayed till the uncommitted transaction either commits or roll backs then such a schedule is known as a Recoverable Schedule.

**Types of Recoverable Schedules –**

● **Cascading Schedule -** If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Schedule or Cascading Rollback or Cascading Abort.

● **Cascadeless Schedule -** If in a schedule, a transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a Cascadeless Schedule.

● **Strict Schedule -** If in a schedule, a transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a Strict Schedule.

**B Trees**

At every level, we have Key and Data Pointer and data pointer points to either block or record. Properties of B-Trees:

Root of B-tree can have children between 2 and P, where P is Order of tree.

**Order of tree –**

Maximum number of children a node can have.

Internal node can have children between ⌈ P/2 ⌉ and P

Internal node can have keys between ⌈ P/2 ⌉ – 1 and P-1

**B+ Trees**

In B+ trees, the structure of leaf and non-leaf are different, so their order is. Order of non-leaf will be higher as compared to leaf nodes.

Searching time will be less in B+ trees, since it doesn’t have record pointers in non-leaf because of which depth will decrease.

# SQL JOIN

In SQL, JOIN clause is used to combine the records from two or more tables in a database.

**Natural Join**

A natural join is a type of join operation that creates an implicit join by combining tables based on columns with the same name and data type. It is similar to the INNER or LEFT JOIN, but we cannot use the ON or USING clause with natural join as we used in them.

**syntax**

**SELECT** [column\_names | \*]

**FROM** table\_name1

**NATURAL JOIN** table\_name2;

### **INNER JOIN**

In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

**Syntax:**

SELECT table1.column1, table1.column2, table2.column1,....

FROM table1

INNER JOIN table2

ON table1.matching\_column = table2.matching\_column;

### **LEFT JOIN**

The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it will return NULL.

**Syntax:**

SELECT table1.column1, table1.column2, table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching\_column = table2.matching\_column;

### **RIGHT JOIN**

In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the matched values from the left table. If there is no matching in both tables, it will return NULL.

**Syntax:**

SELECT table1.column1, table1.column2, table2.column1,....

FROM table1

RIGHT JOIN table2

ON table1.matching\_column = table2.matching\_column;

### **FULL JOIN**

In SQL, FULL JOIN is the result of a combination of both left and right outer join. Join tables have all the records from both tables. It puts NULL on the place of matches not found.

### **Syntax:**

SELECT table1.column1, table1.column2, table2.column1,

FROM table1

FULL JOIN table2

ON table1.matching\_column = table2.matching\_column;

* (INNER) JOIN: Returns records that have matching values in both tables
* LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
* RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
* FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table