What is a database?

A Database is a **logical, consistent, and organized collection of data** that can easily be **accessed, managed and updated. CRUD**
Database mostly **consists of the objects (tables), and tables include the records and fields.**
Fields are the **basic units of data storage**, which contain information about a particular aspect or attribute of the entity described by the database.
DBMS is used for the extraction of data from the database in the form of queries.

What is a database system?

The **collection of database and DBMS software together** is known as a database system. Through the database system, we can perform many activities such as-

The data can be stored in the **database with ease**, and there are **no issues of data redundancy and data inconsistency.**

Database Management System

A database management system is software **that is used to manage the database.**
DBMS provides an interface **to perform various operations like database creation, storing data in it, updating data, creating** a table in the database, and a lot more.
It provides **protection and security** to the database.
In the case of **multiple users**, it also maintains data **consistency.**

For example, **MySQL, Oracle**, etc are a very popular commercial database which is used in different applications.

Advantages of DBMS

Controls database redundancy: It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.

Data sharing: In DBMS, the authorized users of an organization can share the data among multiple users.

Easily Maintenance: It can be easily maintainable due to the centralized nature of the database system.

Reduce time: It reduces development time and maintenance needs.

Backup: It provides backup and recovery subsystems that create an automatic backup of data from hardware and software failures and restores the data if required.

multiple user interfaces: It provides different types of user interfaces like graphical user interfaces, application program interfaces

Redundancy control
Restriction for unauthorized access
Provides multiple user interfaces
Provides backup and recovery
Enforces integrity constraints
Ensure data consistency
Easy accessibility
Easy data extraction and data processing due to the use of queries

Disadvantages of DBMS

Cost of Hardware and Software: It requires a high-speed data processor and large memory size to run DBMS software.

Size: It occupies a large space of disks and large memory to run them efficiently.

Complexity: Database system creates additional complexity and requirements.

Higher impact of failure: Failure has highly impacted the database because, in most of the organization, all the data is stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

What is the E-R model?

E-R model is a short name for the **Entity-Relationship model**. This model is based on the **real world**. It contains **necessary objects (known as entities) and the relationship among these objects.**
it's a conceptual model that gives the **graphical representation of the logical structure of the database**.
An ER diagram is mainly composed of the following three components- **Entity Sets, Attributes**, and **Relationship Set.**
In the E-R diagram, **entities** are represented by **rectangles**, **relationships** are represented by **diamonds**, **attributes** are the characteristics of entities and represented by **ellipses**, and **data flow** is represented through a straight **line.**

What is an Entity?

The **Entity is a set of attributes in a database**. An **entity can be a real-world object** which **physically exists in this world.**

What is an Entity Set?

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

For example, a set of people, a set of students, a set of companies, etc.

● **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.
The 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.**
Discriminators can identify a group of entities from the entity set.
The discriminator is represented by underlining with a dashed line.

What is a Relationship?

The Relationship is defined **as an association among two or more entities.**

● **Unary Relationship Se**t - Unary relationship set is a relationship set where only **one** entity 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.

There are three types of relationships in DBMS-

One-To-One: Here **one record of any object can be related to one record** of another object.
Ex:- Indian citizen and aadhar no.

One-To-Many (many-to-one): Here **one record of any object can be related to many records** of other objects and vice versa.
Ex:- Indian citizen and mobile no.

Many-to-many: Here **more than one record of an object can be related to n number of records** of another object.
Ex: Teacher and student

Cardinality Constraint

Cardinality constraint defines t**he 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** that are **owned by each entity of an Entity Set.**

Types of Attributes:

- **Simple Attributes** - Simple attributes are those attributes **that cannot be divided** further. Ex. Age
- **Composite Attributes** - Composite attributes are **those attributes that are composed of many other simple** attributes. Ex. Name, Address
- **Multi-Valued Attributes** - Multi-valued attributes are those attributes **that 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 that can be derived** from other attributes (s). Ex. Age can be derived from DOB.
- **Key Attributes** - Key attributes are those attributes **that can identify an entity uniquely** in an entity set. **Ex. Roll No.**

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 the primary key must contain a null value in any relation.
- **Referential Integrity Constraint -** It specifies that all the values taken by the foreign key must either be available in relation of the primary key or be null.

Key

A **key is a set of attributes that can identify each tuple uniquely in the given relation.**

Super Key - A superkey 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**
A primary key is a field or the combination of fields that **uniquely identify each record in the table**.
Primary Keys are **unique** and **NOT NULL**

Alternate Key

Candidate keys that are left **unimplemented or unused** after implementing the primary key are called alternate keys.

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** in the table. Once assigned, **its value cannot be changed i.e. it is non-updatable**. It **may have a NULL value**.

Foreign key

The foreign key is used to **link one or more tables** together. It is also known as the **referencing key**.
A foreign key is **specified as a key that is related to the primary key of another table.** It means a **foreign key field in one table refers to the primary key field** of the other table.
It identifies each row of another table **uniquely** that maintains the **referential integrity.**
An **attribute 'X' is called 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 **the referenced relation**.
The relation in which attribute 'X' is present is called **the referencing relation.**

Functional Dependency:

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

Types of Functional Dependency:

● **Trivial Functional Dependencies** – 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** – 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.

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**. When the **sub relations are joined back, the same relation is obtained that was decomposed**.
● **Dependency Preservation** - Dependency preservation ensures 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:** Consider there is a relation R which is decomposed into sub relations R1, R2, …., Rn. This decomposition is called lossless join decomposition **when the join of the sub relations results in the same relation R that was decomposed**.

● **Lossy Join Decomposition:** Consider there is a relation R which is decomposed into sub relations R1, R2, …., Rn.
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**.

What is normalization?

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.**

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

- Reducing the **redundancies**
- Ensuring the integrity of data through **lossless decomposition**

Normalization is considered an essential process as **it is used to avoid data redundancy, insertion anomaly, update anomaly, deletion anomaly.**

Their most commonly used normal forms are:

First Normal Form(1NF)
Second Normal Form(2NF)
Third Normal Form(3NF)
Boyce & Codd Normal Form(BCNF)

What is 1NF?

A relation is in first normal form **if it does not contain any composite or multi-valued attribute**. A relation is in its first normal form **if every attribute in that relation is a single valued attribute.**

Create separate **tables for each group of related data** and identify each **row with a unique column**

What is 2NF?

To be in the second normal form, a relation must be in the first **normal form and the relation** must **not contain any partial dependency**. A relation is in 2NF if it has **No Partial Dependency,** i.e.**,** **no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.**

Partial Dependency - If the **proper subset of candidate keys determines non-prime attributes**, it is called partial dependency.

What is 3NF?

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

A relation is in 3NF if at least **one of the following conditions 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).

A → B is called a **transitive dependency** if and only if- **A is not a super key** and **B is a non-prime attribute.**

Transitive dependency - If A->B and B->C are two FDs then A->C is called transitive dependency.

What is BCNF?

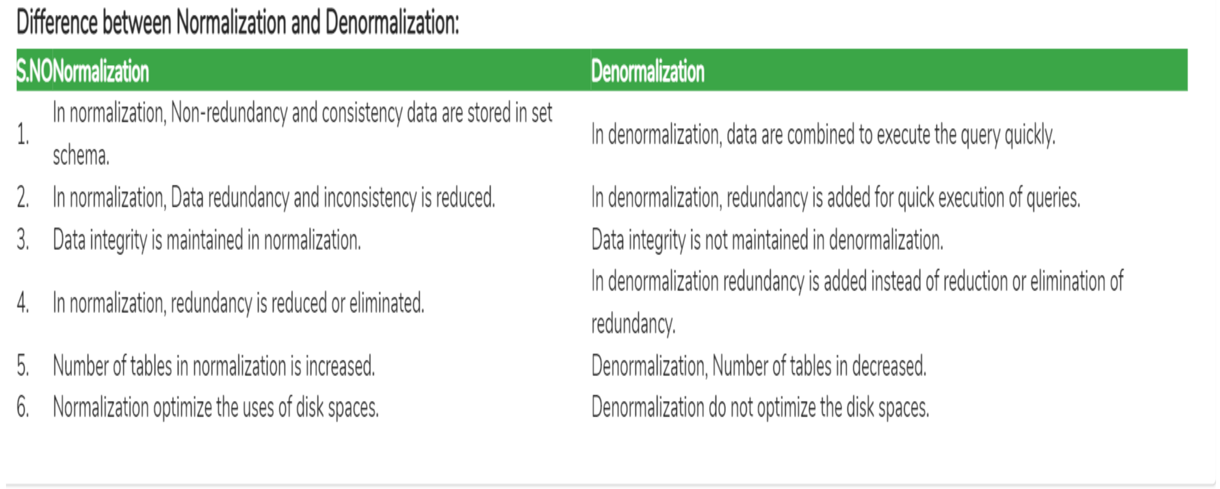
A relation R is in **BCNF if R is in Third Normal Form** and for every FD, **LHS is super key.** A relation is in **BCNF iff in every non-trivial functional dependency X --> Y,**

X is a superkey

BCMF stands for Boyce-Codd Normal Form. It is an advanced version of 3NF, so it is also referred to as 3.5NF. BCNF is stricter than 3NF.
A table complies with BCNF if it satisfies the following conditions:
It is in **3NF**.
For every functional dependency X->Y, **X should be the super key of the table**. It merely means that X cannot be a non-prime attribute if Y is a prime attribute

What is Denormalization?

Denormalization is the process of **boosting up database performance** and **adding redundant data** which helps to get rid of complex data. Denormalization is a part of the **database optimization technique**. This process is used to avoid the **use of complex and costly joins**. Denormalization doesn't refer to the thought of not normalizing; instead, denormalization **takes place after normalization**.



Difference between Normalization and Denormalization:

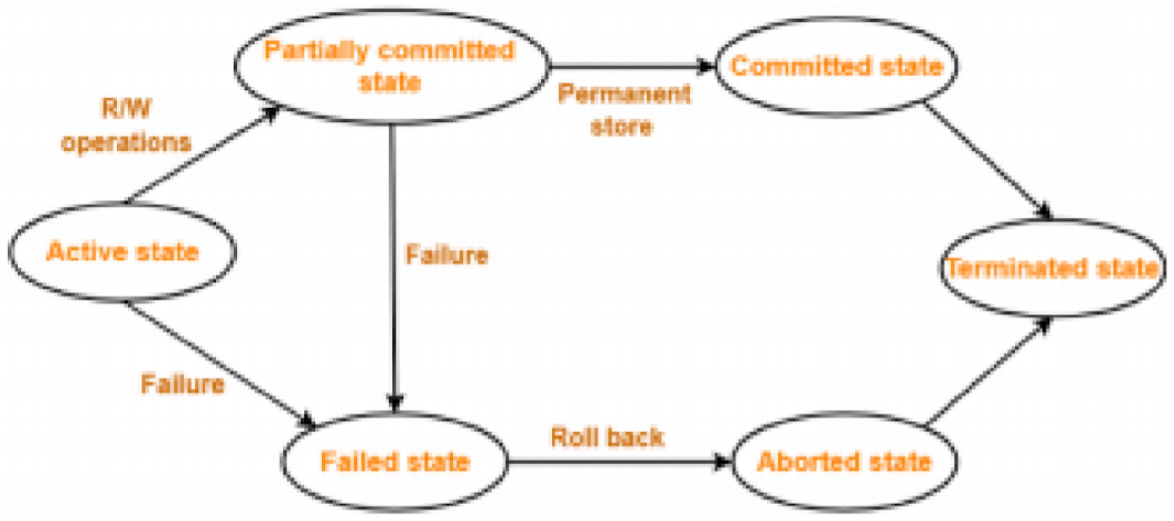
S.NO	Normalization	Denormalization
1.	In normalization, Non-redundancy and consistency data are stored in set schema.	In denormalization, data are combined to execute the query quickly.
2.	In normalization, Data redundancy and inconsistency is reduced.	In denormalization, redundancy is added for quick execution of queries.
3.	Data integrity is maintained in normalization.	Data integrity is not maintained in denormalization.
4.	In normalization, redundancy is reduced or eliminated.	In denormalization redundancy is added instead of reduction or elimination of redundancy.
5.	Number of tables in normalization is increased.	Denormalization, Number of tables in decreased.
6.	Normalization optimize the uses of disk spaces.	Denormalization do not optimize the disk spaces.

Transaction

The transaction is a **single logical unit of work formed by a set of operations. which accesses and possibly modifies the contents of a database. Transactions access data using read and write 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 the main memory.
● **Write Operation** – Write(A) will write the updated value of ‘A’ from the buffer to the database.



- **Active State** –
  - This is the first state in the life cycle of a transaction.
  - A transaction is called in an **active state as long as its instructions are executed.**
  - All the changes made by the transaction now are stored in the buffer in the main memory.

- **Partially Committed State** –
  - After the last instruction of the transaction has been executed, it enters into a partially committed state.
  - 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 the main memory.

- **Committed State** –
  - After all the **changes made by the transaction have been successfully stored in the database,** it enters into a committed state.
  - Now, the transaction is considered to be fully committed.

- **Failed State** –

- 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 –**
  - After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.
  - To undo the changes made by the transaction, it becomes necessary to roll back the transaction.
  - After the **transaction has rolled back completely, it enters into an aborted state.**

- **Terminated State –**
  - This is the last state in the life cycle of a transaction.
  - 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.**

Explain ACID properties

ACID properties are some basic rules, which have to be **satisfied by every transaction to preserve integrity.** These properties and rules are:

ATOMICITY: **all or nothing rule**.

- This property ensures that either the transaction **occurs completely or it does not occur at all.**
- In other words, it ensures that **no transaction occurs partially.**

CONSISTENCY: This property refers to the **uniformity of the data.**

- This property ensures that **integrity constraints are maintained**.
- In other words, it ensures that the **database remains consistent before and after the transaction.**

ISOLATION:

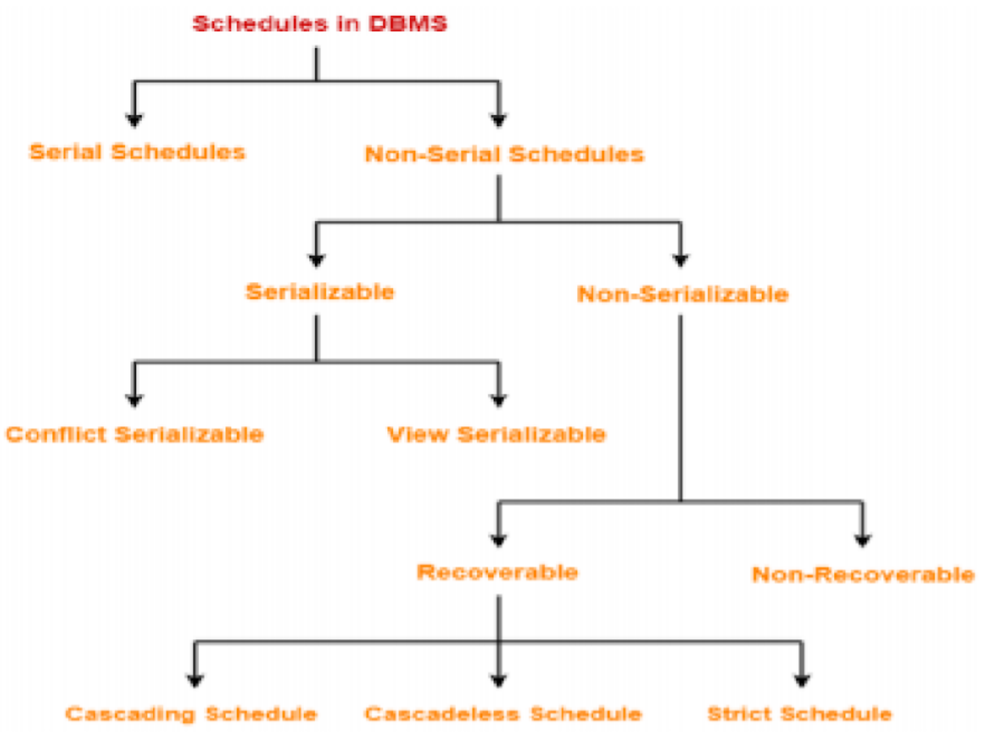
- This property ensures that **multiple transactions can occur simultaneously** without **causing any inconsistency**.
- The resultant state of the system **after executing all the transactions is the same** as the state that would be achieved if the transactions were **executed serially one after the other.**

DURABILITY:

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



Serial Schedules :

- All the transactions **execute serially** one after the other.
- When **one transaction executes**, **no other transaction** is allowed to execute.
- Serial schedules are always- **Consistent, Recoverable, Cascadeless, and Strict**

Non-Serial Schedules

- **Multiple transactions** execute concurrently.
- Operations of all the transactions are **interleaved or mixed with each other**.
- 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 –

- If a given **non-serial schedule of ‘n’ transactions is equivalent to some serial schedule of ‘n’ transactions**, then it is called a serializable schedule.
- 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 **that is not serializable** is called a non-serializable schedule.
● A non-serializable schedule is **not guaranteed to produce the same effect as produced by a 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,

- A transaction performs a **dirty read operation** from an **uncommitted transaction**
- And if it **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
- Its **commit operation is delayed until the uncommitted transaction either commits or rolls back,** 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 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 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 been written is committed or aborted**, then such a schedule is called a Strict Schedule.

What is RDBMS?

RDBMS stands for **Relational Database Management Systems**.
It is based on the **relational model.**
It is used to maintain the **data records and indices in tables**.
A relational database is the **most commonly used database**. It contains a **number of tables and each table has its own primary key**.
Due to a collection of an **organized set of tables, data can be accessed easily in RDBMS.**
RDBMS is the system that enables you to perform different operations such as **update, insert, delete, manipulate and administer** a relational database with minimal difficulties.
Most of the time RDBMS uses **SQL language because it is easily understandable and is used often**.

Relational databases have the following properties:

Values are atomic.
All of the values in a column have the same data type.
Each row is unique.
The sequence of columns is insignificant.
The sequence of rows is insignificant.
Each column has a unique name.
Integrity constraints maintain data consistency across multiple tables.

What is data abstraction in DBMS?

Data Abstraction is a **process of hiding unwanted or irrelevant details from the end-user.** It provides a different view and helps in achieving data independence which is used to enhance the security of data.

The database systems consist of complicated data structures and relations. For users to access the data easily, these complications are kept hidden, and only the relevant part of the database is made accessible to the users through data abstraction.

For example, We know that most of the users prefer those systems which have a simple GUI that means no complex processing. So, to keep the user tuned and for making access to the data easy, it is necessary to do data abstraction. In addition, data abstraction divides the system into different layers to make the work specified and well defined

there are three levels of abstraction for DBMS, which are as follows −

Physical or Internal Level
Logical or Conceptual Level
View or External Level

Physical or Internal Level

It is the lowest level of abstraction for DBMS which defines how the data is actually stored, it defines data structures to store data and access methods used by the database.
Actually, it is decided by developers or database application programmers how to store the data in the database.

So, overall, the entire database is described at the physical or internal level. It is a very complex level to understand.

For example, customer information is stored in tables, and data is stored in the form of blocks of storage such as bytes, gigabytes, etc.

What is 2-Tier architecture?

The 2-Tier architecture is the same as basic client-server. In the two-tier architecture, **applications on the client** end can directly communicate with the **database at the server-side**.

What is the 3-Tier architecture?

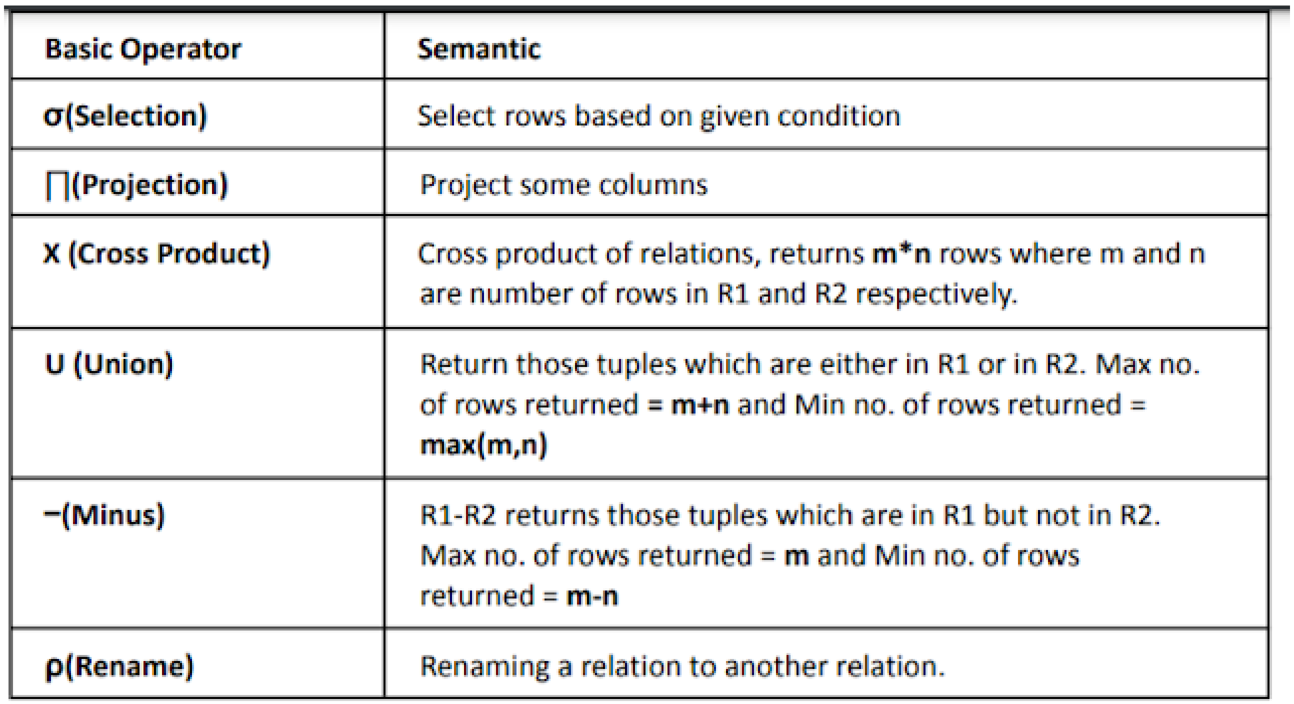
The 3-Tier architecture contains another layer between the client and server. The introduction of 3-tier architecture is for the ease of the users as it provides the GUI, which makes the system secure and much more accessible.
In this architecture, the application on the **client-end interacts with an application on the server** which further communicates with **the database system.**

What is Functional Dependency?

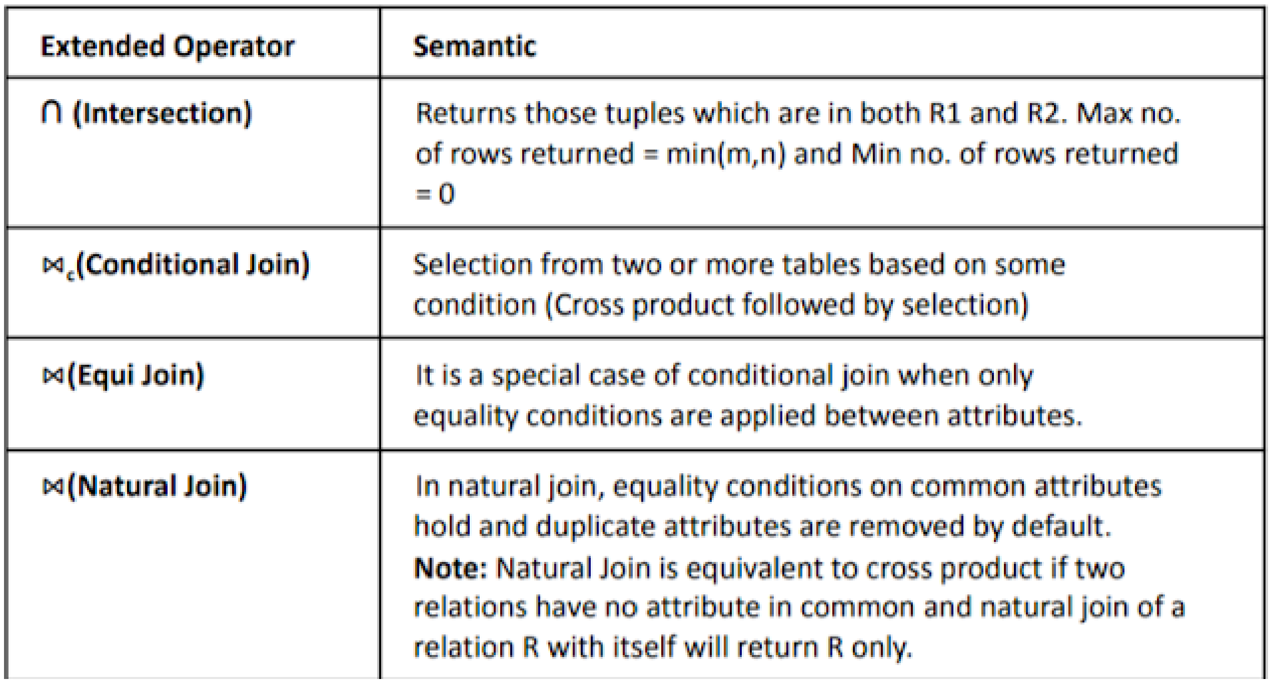
Functional Dependency is the starting point of normalization.
It exists when a relation between two attributes allows you to determine the corresponding attribute's value uniquely.
The functional dependency is also known as database dependency and is defined as the relationship which occurs when one attribute in a relation uniquely determines another attribute.
It is written as A->B which means B is functionally dependent on A.

Relational Algebra:

Relational Algebra is a **procedural query language** that takes a **relation as an input** and **generates a relation as an output**.



Basic Operator	**Semantic**
σ(Selection)	Select rows based on given condition
∏(Projection)	Project some columns
X (Cross Product)	Cross product of relations, returns **m*n** rows where m and n are number of rows in R1 and R2 respectively.
U (Union)	Return those tuples which are either in R1 or in R2. Max no. of rows returned **= m+n** and Min no. of rows returned = **max(m,n)**
−(Minus)	R1-R2 returns those tuples which are in R1 but not in R2. Max no. of rows returned = **m** and Min no. of rows returned = **m-n**
ρ(Rename)	Renaming a relation to another relation.



Extended Operator	**Semantic**
∩ (Intersection)	Returns those tuples which are in both R1 and R2. Max no. of rows returned = min(m,n) and Min no. of rows returned = 0
$\bowtie_c$**(Conditional Join)**	Selection from two or more tables based on some condition (Cross product followed by selection)
⋈(Equi Join)	It is a special case of conditional join when only equality conditions are applied between attributes.
⋈(Natural Join)	In natural join, equality conditions on common attributes hold and duplicate attributes are removed by default. **Note:** Natural Join is equivalent to cross product if two relations have no attribute in common and natural join of a relation R with itself will return R only.

B Trees

At every level, we have **Key** and **Data Pointer** and **data pointer points to either block or record**.

Properties of B-Trees:

Roots of a B-tree can have **children between 2 and P**, where P is the Order of the tree.

Order of tree – **Maximum number of children a node** can have.

An internal node can have **children** between ⌈ **P/2** ⌉ **and P**
An 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 **they don't have record pointers in non-leaf** because of which **depth will decrease**.

Indexing:

Indexing is a way to **optimize the performance of a database by minimizing the number of disk accesses** required when a query is processed. It is a data structure technique **that is used to quickly locate and access the data in a database.**

Indexes are created using a few database columns.

- The first column is the **Search key** that contains a **copy of the primary key or candidate key of the table.** These values are stored in sorted order so that the corresponding data can be accessed quickly.
  Note: The data may or may not be stored in sorted order.
- The second column is the **Data Reference** or **Pointer which contains a set of pointers holding the address of the disk block** where that particular key value can be found.

File Structures:

● **Primary Index:** A primary index is **an ordered file**, with records of fixed length with two fields. The first field is the same **as the primary key** as a data file and the second field is a pointer to the data block, where the key is available.
The average number of block accesses using **index = log2Bi + 1**, where Bi = number of index blocks.
● **Clustering Index:** A clustering index is created on a data file whose records are **physically ordered on a non-key field** (called the Clustering field).
● **Secondary Index:** Secondary index provides secondary means of accessing a file for which primary access already exists.

SQL

What is SQL

SQL is a standard language for **storing, manipulating, and retrieving data** in databases.

SQL stands for the **Structured Query Language.**
It is the standard language used to **maintain the relational database and perform many different data manipulation operations on the data.**
SQL was initially invented in 1970.
It is a database language used for **database creation, deletion, fetching and modifying rows, etc.** sometimes, it is pronounced as 'sequel.' We can also use it to handle organized data composed of entities (variables) and relations between different entities of the data.

What are the usages of SQL?

SQL is responsible for maintaining **the relational data and the data structures present in the database**. Some of the common usages are given below:

To execute queries against a database
To retrieve data from a database
To inserts records in a database
To update records in a database
To delete records from a database
To create new databases
To create new tables in a database
To create views in a database
To perform complex operations on the database

What are tables and fields in the database?

A table is a set of organized data in the form of rows and columns.
It enables users to store and display records in the structure format. It is similar to worksheets in the spreadsheet application. Here, **rows refer to the tuples, representing the simple data item**, and **columns are the attribute of the data items present in a particular row.**
Columns can be categorized as vertical, and Rows are horizontal.

Fields are the components to provide the structure for the table.
It stores the same category of data in the same data type.
A table contains a fixed number of columns but can have any number of rows known as the record. It is also called a column in the table of the database.
It represents the attribute or characteristics of the entity in the record.

How many types of database languages are there?

Data Definition Language (DDL)

Data definition language (DDL) is a computer language used to **create and modify the structure of database objects** in a database. These database objects include views, schemas, tables, indexes, etc.
SQL commands that can be used to define the database schema.
e.g., CREATE, ALTER, DROP, TRUNCATE, RENAME, etc.

● **CREATE** - to create a database and its objects like (table, index, views, store procedure, function, and triggers)
● **ALTER** - alters the structure of the existing database
● **DROP** - delete objects from the database
● **TRUNCATE** - remove all records from a table, including all spaces allocated for the records are removed
● **RENAME** - rename an object

DQL (Data Query Language) :

DQL statements are used for **performing queries on the data within schema objects.** The purpose of the DQL Command is to get some schema relation based on the query passed to it.

Example of DQL:

- SELECT **– is used to retrieve data from the database.**

Data Manipulation Language (DML)

These commands are used for the **manipulation of already updated data**
This manipulation involves **inserting** data into database tables, **retrieving** existing data, **deleting** data from existing tables, and **modifying** existing data.
DML is mostly incorporated in SQL databases
e.g., SELECT, UPDATE, INSERT, DELETE, etc.

● **SELECT** - retrieve data from a database
● **INSERT** - insert data into a table
● **UPDATE** - updates existing data within a table
● **DELETE** - Delete all records from a database table
● **MERGE** - UPSERT operation (insert or update)

Data Control Language (DCL)

DCL is used to **control user access** in a database.
This command is related to **security issues.**
Using the DCL command, **allows or restricts** the user from accessing data in the database schema.
e.g., GRANT and REVOKE

● **GRANT** - allow users **access privileges** to the database
● **REVOKE** - **withdraw users access privileges** given by using the GRANT command

Transaction Control Language (TCL)
Transaction Control Language(TCL) commands are **used to manage transactions in the database**
These are used to **manage the changes** made to the data in a table by **DML statements**.
e.g., COMMIT, ROLLBACK, and SAVEPOINT.

● **COMMIT** - commits a Transaction
● **ROLLBACK** - rollback a transaction in case of any error occurs
● **SAVEPOINT** - to roll back the transaction making points within groups

Nested Query:
In nested queries, **a query is written inside a query.** The result of the **inner query is used in the execution of the outer query.**

Independent

Correlated

SELECT:

The SELECT statement is used to **select data** from a database.

Syntax -

● SELECT column1, column2, ... FROM table_name;

● Here, column1, column2, ... are the field names of the table you want to select data from.

If you want to **select all the fields** available in the table, use the following syntax:

● SELECT * FROM table_name;

Ex –

● SELECT CustomerName, City FROM Customers;

SELECT DISTINCT:

The SELECT DISTINCT statement is used to return **only distinct (different) values**.

● SELECT DISTINCT Country FROM Customers;

WHERE:

The WHERE clause is used to **filter records.**

Ex – ● SELECT * FROM Customers WHERE Country='Mexico';

AND, OR and NOT:

The WHERE clause can be combined with AND, OR, and NOT operators.

● The AND operator displays a record if all the conditions separated by AND are TRUE.

● The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE

ORDER BY:

The ORDER BY keyword is used to **sort the result-set** in ascending or descending order.
The ORDER BY keyword sorts the **records in ascending order by default.**
To sort the records in descending order, use the **DESC keyword**.

Ex –
● SELECT * FROM Customers ORDER BY Country;
● SELECT * FROM Customers ORDER BY Country ASC, CustomerName DESC;

INSERT INTO:

The INSERT INTO statement is used to i**nsert new records in a table**.
Ex –
● INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

NULL Value:

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
We will have to use the **ISNULL** and **IS NOT NULL** operators instead.
● SELECT column_names
FROM table_name
WHERE column_name IS NULL;
● SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;

UPDATE:

The UPDATE statement is used to modify the **existing records in a table**.
● UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;

DELETE:

The DELETE statement is used to **delete existing records** in a table.
● DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

SELECT TOP:
The SELECT TOP clause is used to specify the **number of records to return.**
● SELECT TOP 3 * FROM Customers;
● SELECT * FROM Customers LIMIT 3;
● SELECT * FROM Customers FETCH FIRST 3 ROWS ONLY;

Aggregate Functions:

MIN():
The MIN() function returns the **smallest value of the selected column.**
● SELECT MIN(column_name) FROM table_name WHERE condition;

MAX():
The MAX() function returns the **largest value** of the selected column.
● SELECT MAX(column_name) FROM table_name WHERE condition;

COUNT():
The COUNT() function returns the **number of rows that matches a specified criterion**.
● SELECT COUNT(ProductID) FROM Products;

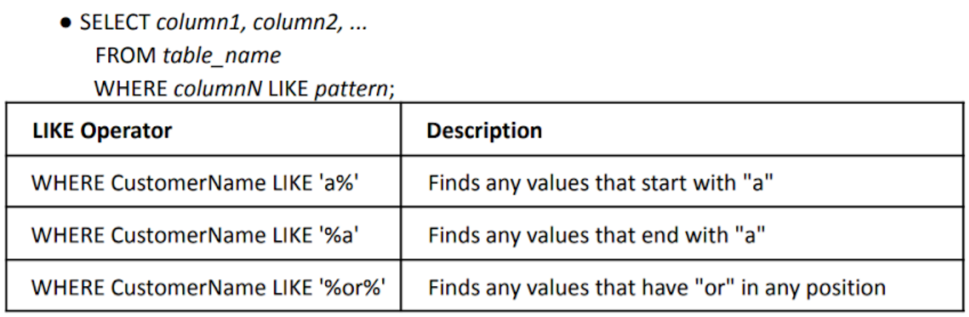
AVG():
The AVG() function returns the **average value of a numeric column**
● SELECT AVG(Price) FROM Products;

SUM():
The SUM() function returns the **total sum of a numeric column**.
● SELECT SUM(Quantity) FROM OrderDetails;

LIKE Operator:

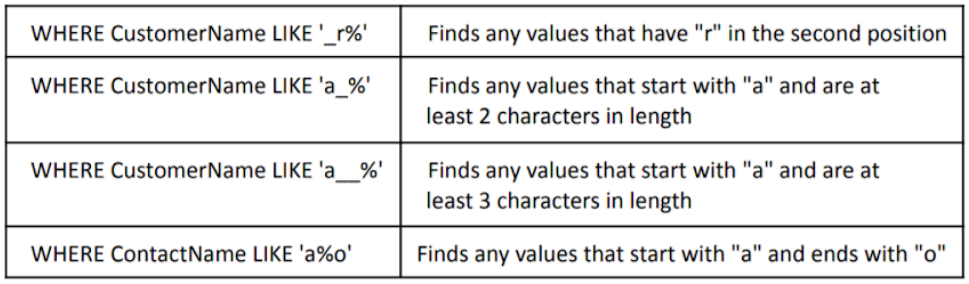
The LIKE operator is used in a WHERE clause to **search for a specified pattern** in a column. There are two wildcards often used in conjunction with the LIKE operator:

- The **percent sign (%)** represents zero, one, or multiple characters.
- The **underscore sign (_)** represents one, single character.



- SELECT *column1, column2, ...*
  FROM *table_name*
  WHERE *columnN* LIKE *pattern*;

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position



WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

IN:

The IN operator **allows you to specify multiple values** in a WHERE clause. The IN operator is a shorthand for **multiple OR conditions**.

Ex –

- SELECT * FROM Customers WHERE Country IN ('Germany', 'France', 'UK');

BETWEEN:

The BETWEEN operator selects values **within a given range.**
The values can be numbers, text, or dates.
The BETWEEN operator is inclusive: **begin and end values** are included

UNION:

The UNION operator is used to combine the **result-set of two or more SELECT statements**.

● Every SELECT statement within **UNION must have the same number of columns**

● The columns must also have **similar data types**.

● The columns in every SELECT statement must also be in the same order The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL

Ex –

● SELECT City FROM Customers
UNION
SELECT City FROM Suppliers ORDER BY City;

GROUP BY:

GROUP BY statement **groups rows that have the same values** into summary rows,
like "find the number of customers in each country".
The GROUP BY statement is often used with aggregate functions
(COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more.

● SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country ORDER BY COUNT(CustomerID) DESC;

HAVING:

The HAVING clause was added to SQL because the **WHERE keyword cannot be used with aggregate functions**. ***WHERE is given priority over HAVING**.

● SELECT COUNT(CustomerID),
Country FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;

CREATE DATABASE:

The CREATE DATABASE statement is used to **create a new SQL database**.
Syntax –
● CREATE DATABASE database name;

DROP DATABASE:

The DROP DATABASE statement is used to **drop an existing SQL database.**
Syntax –
● DROP DATABASE database name;

CREATE TABLE:

The CREATE TABLE statement is used to **create a new table in a database.**
Syntax –
● CREATE TABLE table_name (column1 datatype, column2 datatype, column3 datatype, ...);

DROP TABLE:

The DROP TABLE statement is used to **drop an existing table** in a database.
Syntax –
● DROP TABLE table_name;

TRUNCATE TABLE:

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself. Syntax –
● TRUNCATE TABLE table_name;

ALTER TABLE:

The ALTER TABLE statement is used **to add, delete, or modify columns** in an existing table.
The ALTER TABLE statement is **also used to add and drop** various constraints on an existing table.

Ex –
● ALTER TABLE Customers ADD Email varchar(255);
● ALTER TABLE Customers DROP COLUMN Email;
● ALTER TABLE Persons ALTER COLUMN DateOfBirth year;

What is Join?

The Join operation is **one of the most useful activities in relational algebr**a. It is the most commonly used way to **combine information from two or more relations.**
A JOIN clause is used to **combine rows from two or more tables, based on a related column between them.**

There are the following types of join:

Inner joins:
Inner join is of 3 categories. They are:
Theta join
Natural join
Equi join

Outer joins: Outer join has three types. They are:
Left outer join
Right outer join
Full outer join

What is INNER JOIN in SQL?

The INNER JOIN keyword **selects all rows from both the tables as long as the condition satisfies.** This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e **value of the common field will be the same**

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

What is Left Join in SQL?

The Left Join is used **to fetch all rows from the left-hand table and common records between the specified tables.**
It returns **all the rows from the left-hand side table even though there are no matches on the right-hand side table**.
If it does not find any matching record from the right side table, **then it returns null. This join can also be called a Left Outer Join.**

What is the Right JOIN in SQL?

The Right join is used to **retrieve all rows from the right-hand table and only** those rows from the other table that fulfill the join condition.
It returns all the rows from the right-hand side table even though there are no matches in the left-hand side table.
If it finds unmatched records from the left side table, it **returns a Null value. This join is also known as Right Outer Join.**

What is self-join and what is the requirement of self-joining?

A SELF JOIN is used to join a table with self. This join can be performed using table aliases, which allow us to avoid repeating the same table name in a single sentence. It will throw an error if we use the same table name more than once in a single query without using table aliases.

A SELF JOIN is required when we want to combine data with other data in the same table itself. It is often very useful to convert a hierarchical structure to a flat structure.

The following syntax illustrates the SELF JOIN:

```
SELECT column_lists
FROM table1 AS T1, table1 AS T2
WHERE join_conditions;
```

Highest Salary

```
SELECT name, MAX(salary) as salary
FROM employee
```

Second highest Salary

Using nested

```
SELECT name, MAX(salary) AS salary
FROM employee
WHERE salary < (SELECT MAX(salary)
FROM employee);
```

SELECT salary

FROM employee

ORDER BY salary desc limit 1,1

Third Highest Salary:

```
SELECT name, MAX(salary) AS salary
 FROM employee
WHERE salary < (SELECT MAX(salary)
        FROM employee
        WHERE salary < (SELECT MAX(salary)
                        FROM employee)  );
```

- SELECT salary
  FROM employee
  ORDER BY salary desc limit 2,1

Find Nth highest element

- SELECT salary
  FROM employee
  ORDER BY salary desc limit n-1,1

JOIN K CODES

i)

SELECT **IF(GRADE < 8, NULL, NAME)**, GRADE, MARKS
FROM STUDENTS JOIN GRADES
WHERE MARKS **BETWEEN MIN_MARK AND MAX_MARK**
ORDER BY GRADE DESC, NAME

ii) Adding more than two tables

```
// more than two tables

select h.hacker_id, h.name
from submissions s
inner join challenges c
on s.challenge_id = c.challenge_id
inner join difficulty d
on c.difficulty_level = d.difficulty_level
inner join hackers h
on s.hacker_id = h.hacker_id
where s.score = d.score and c.difficulty_level =
d.difficulty_level
group by h.hacker_id, h.name
having count(s.hacker_id) > 1
order by count(s.hacker_id) desc, s.hacker_id asc
```