Introduction to DBMS

Database Management Systems (DBMS) play a crucial role in modern computing by providing an organized and efficient approach to storing, managing, and retrieving data. From small-scale applications to large enterprises, DBMS has become an indispensable component of information systems.

Definition: DBMS refers to a software system that enables users to define, create, manage, and manipulate databases.

Database : A database is a structured collection of data organized and stored electronically in a computer system. It is designed to efficiently manage and facilitate the storage, retrieval, modification, and deletion of data. Databases typically consist of one or more tables or files, each containing rows or records with fields or attributes.

Types of DBMS

Relational Database Management Systems (RDBMS):
- Definition: RDBMS organizes data into tables with rows and columns, and establishes relationships between them.
- Example: MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server.

NoSQL Databases:
- Definition: NoSQL databases are designed to handle unstructured or semi-structured data with flexible schemas.
- Example: MongoDB (Document-oriented), Redis (Key-value store), Cassandra (Column-family store), Neo4j (Graph database).

Object-Oriented Database Management Systems (OODBMS):
- Definition: OODBMS stores data as objects, encapsulating data and behavior together.
- Example: db4o, ObjectDB.

NewSQL Databases:
- Definition: NewSQL databases combine the scalability of NoSQL with the ACID properties of traditional RDBMS.
- Example: Google Spanner, CockroachDB, NuoDB.

In-memory Databases:
- Definition: In-memory databases store data primarily in system memory (RAM) for faster access.
- Example: SAP HANA, Redis (can function as both in-memory and NoSQL database).

Time-Series Databases:
- Definition: Time-series databases specialize in storing and querying time-stamped data points or events.
- Example: InfluxDB, Prometheus, TimescaleDB.

Spatial Databases:
- Definition: Spatial databases store and query geometric and geographic data, such as maps and spatial relationships.
- Example: PostGIS (extension for PostgreSQL), Oracle Spatial and Graph, MongoDB (with GeoJSON support).

File System Vs DBMS

File System:

- Basic Structure: A file system organizes and stores data on a computer's storage devices, typically using a hierarchical structure of directories and files.
- Data Organization: Data is stored in files, which can be of various types (e.g., text files, images, videos). Each file is typically self-contained and can be accessed directly through its path.
- Data Integrity: File systems often lack built-in mechanisms for ensuring data integrity and consistency. Users and applications are responsible for managing data consistency.
- Concurrency: Traditional file systems may not handle concurrent access to files well, leading to potential issues like file corruption or data loss when accessed by multiple processes simultaneously.
- Querying: Retrieving specific data from files usually involves searching through the file contents manually or using basic file system commands, which can be inefficient for complex queries or large datasets.

Database Management System (DBMS):

- Structured Storage: A DBMS organizes data into structured formats such as tables, rows, and columns within a database, enabling efficient querying and retrieval.
- Data Integrity: DBMSs offer mechanisms like transactions, ACID (Atomicity, Consistency, Isolation, Durability) properties, and referential integrity constraints to ensure data consistency and integrity.

- Concurrency Control: DBMSs employ sophisticated concurrency control mechanisms to handle simultaneous access to data, preventing issues like data corruption and ensuring data consistency.
- Querying and Manipulation: DBMSs provide powerful query languages (e.g., SQL) and APIs for querying and manipulating data, allowing for complex operations and efficient retrieval of specific information from large datasets.
- Scalability: DBMSs are designed to handle large volumes of data efficiently and can scale horizontally (across multiple servers) or vertically (by upgrading hardware) to accommodate growing data needs.
- Security and Access Control: DBMSs offer robust security features such as user authentication, authorization, and encryption to protect data from unauthorized access or malicious activities.

Client Server Architecture

In a basic client-server architecture, clients make requests to servers, and servers fulfill those requests by providing the necessary resources or services. The communication between clients and servers typically occurs over a network, such as the internet or a local area network (LAN). Here's a breakdown of the components:

1. **Client**: The client is the user interface or front-end component of the application. It interacts with the user, collects input, and sends requests to the server for processing. Clients can be desktop applications, web browsers, mobile apps, or any device capable of making network requests.

2. **Server**: The server is the back-end component of the application responsible for storing and managing data, executing business logic, and serving requests from clients. It responds to client requests by performing the necessary operations and returning results. Servers can range from simple web servers to complex application servers and database servers.

Two-Tier Architecture:

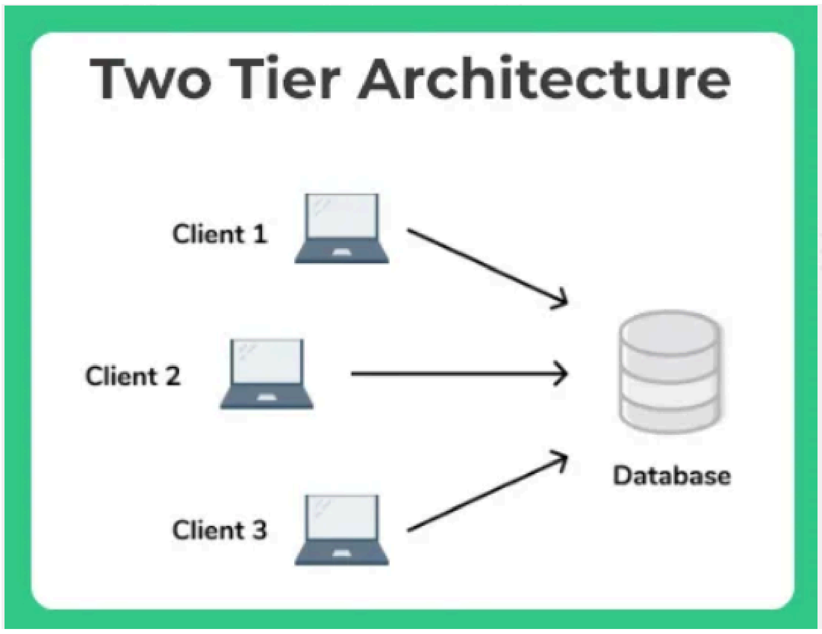
In a two-tier architecture, the client communicates directly with the server. There are two main layers:

Presentation Layer (Client): This layer consists of the user interface components responsible for displaying information to the user and collecting user input. It

handles the presentation logic and user interaction. Examples include desktop applications, web browsers, and mobile apps.
Data Storage and Processing Layer (Server): This layer handles data storage, retrieval, and processing. It contains the business logic of the application and interacts with the database management system (DBMS) to store and retrieve data. Examples include database servers, application servers, and web servers.

Characteristics of Two-Tier Architecture:

- Simplicity: Two-tier architecture is relatively straightforward to implement and manage.
- Tight Coupling: The client and server are tightly coupled, leading to potential scalability and maintenance issues as the application grows.
- Scalability Challenges: Scaling both the client and server components can be challenging due to the tightly coupled nature of the architecture.



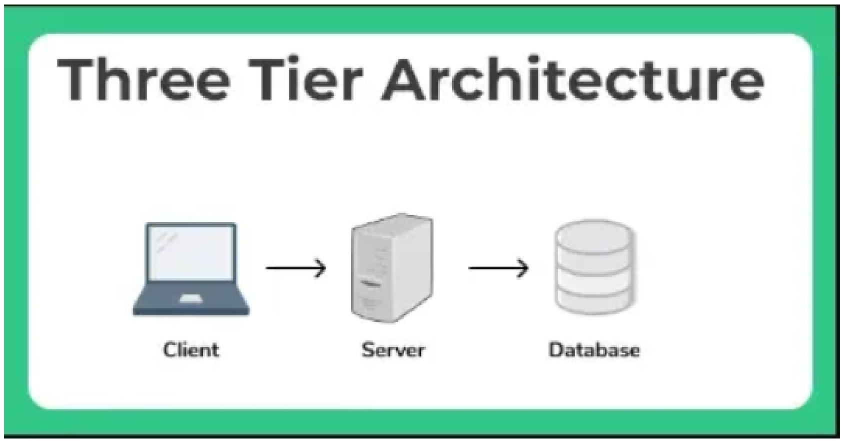
Three-Tier Architecture:

In a three-tier architecture, an additional layer is introduced between the client and server, known as the middle tier or application tier. This layer helps to improve scalability, flexibility, and maintainability by separating concerns. Here are the layers:

Presentation Layer (Client): Same as in the two-tier architecture, responsible for the user interface and user interaction.
Application Layer (Middle Tier): This layer contains the application logic and business rules. It handles client requests, processes them, and communicates with the data layer to retrieve or update data. Examples include application servers, web servers, and middleware components.
Data Storage Layer (Server): Similar to the two-tier architecture, responsible for storing and managing data. It interacts with the application layer to perform CRUD (Create, Read, Update, Delete) operations on the database.

Characteristics of Three-Tier Architecture:

- Separation of Concerns: Three-tier architecture separates presentation, application logic, and data storage, making the application more modular and easier to maintain.
- Scalability: The middle tier can be scaled independently of the client and data layers, allowing for better scalability and performance optimization.
- Flexibility: Adding or modifying components in each layer can be done without affecting the other layers, providing greater flexibility in development and maintenance.



Data Modeling

Data modeling is the process of creating a conceptual representation of data structures, relationships, properties, and constraints to facilitate understanding and communication within an organization. It involves defining the structure of data to support the specific needs and requirements of a business or application.

Entity-Relationship (ER) Modeling

The Entity-Relationship (ER) model is a conceptual data model used to represent the relationships between different entities in a database. It is widely used in database design to visualize the structure of a database and the relationships between various data entities. The ER model is based on the principle of entities, attributes, and relationships.

Entities:

- An entity is a real-world object or concept that has attributes representing its properties.
- Entities are represented by rectangles in the ER diagram.
- Each entity is uniquely identifiable and can be distinguished from other entities in the database.
- Examples of entities include "Customer," "Product," "Employee," "Order," etc.

Attributes:

- Attributes are the properties or characteristics of entities.
- They describe the features or qualities of entities and are represented as ovals in the ER diagram.
- Attributes can be simple (atomic) or composite (composed of multiple sub-attributes).
- Examples of attributes for a "Customer" entity might include "Customer ID," "Name," "Address," etc.

Relationships:

- Relationships define the associations and interactions between entities.
- They represent how entities are related to each other and are depicted by diamond shapes in the ER diagram.
- Relationships can have cardinality, which describes the number of instances of one entity that are associated with another entity.
- Relationships can be one-to-one, one-to-many, or many-to-many.
- Examples of relationships include "Customer places Order," "Employee manages Department," etc.

Keys:

- A key is an attribute or a combination of attributes that uniquely identifies each instance of an entity.
- Primary keys are unique identifiers for entities and are used to ensure data integrity and facilitate data retrieval.
- Foreign keys establish relationships between entities by referencing the primary key of another entity.

- Keys are crucial for maintaining data integrity and enforcing constraints within the database.

Types of Attributes

Simple Attributes:

- Simple attributes are atomic attributes that cannot be divided into smaller components.
- They represent basic properties of entities and are typically represented by single values.
- Examples include "Name," "Age," "Address," etc.

Composite Attributes:

- Composite attributes are made up of multiple component attributes, each representing a part of the whole attribute.
- They are used to represent complex properties that can be broken down into smaller components.
- Examples include "Address" (composed of street, city, state, and zip code), "Name" (composed of first name and last name), etc.

Single-Valued Attributes:

- Single-valued attributes have a single value for each instance of an entity.
- They represent properties that can only have one value at a time.
- Examples include "Age," "Date of Birth," "Gender," etc.

Multi-Valued Attributes:

- Multi-valued attributes can have multiple values for each instance of an entity.
- They represent properties that can have more than one value at the same time.
- Examples include "Phone Numbers" (a person may have multiple phone numbers), "Email Addresses," etc.

Derived Attributes:

- Derived attributes are attributes whose values can be derived or calculated from other attributes in the database.
- They are not stored explicitly in the database but are calculated when needed.
- Examples include "Age" (can be derived from the "Date of Birth"), "Total Price" (calculated from the quantity and unit price), etc.

Key Attributes:

- Key attributes uniquely identify each instance of an entity.

- They are used to establish the identity of entities and ensure data integrity.
- Primary keys and foreign keys are examples of key attributes.
- Examples include "Customer ID," "Product Code," etc.

Types of ER Relationships (Cardinality)

One-to-One (1:1) Relationship:

- In a one-to-one relationship, each instance of one entity is associated with exactly one instance of another entity, and vice versa.
- For example, in a database modeling employees and their office locations, each employee may have exactly one office, and each office may be assigned to only one employee.

One-to-Many (1:M) Relationship:

- In a one-to-many relationship, one instance of an entity is associated with zero or more instances of another entity, but each instance of the related entity is associated with exactly one instance of the first entity.
- For example, in a database modeling customers and their orders, one customer may place many orders, but each order is placed by exactly one customer.

Many-to-One (M:1) Relationship:

- A many-to-one relationship is the inverse of a one-to-many relationship. It occurs when many instances of one entity are associated with exactly one instance of another entity.
- For example, in a database modeling employees and their departments, many employees may belong to the same department, but each employee belongs to exactly one department.

Many-to-Many (M:N) Relationship:

- In a many-to-many relationship, each instance of one entity can be associated with zero or more instances of another entity, and vice versa.
- For example, in a database modeling students and courses, each student may enroll in multiple courses, and each course may have multiple students enrolled in it.
- To represent a many-to-many relationship in a relational database, a junction table or associative entity is often introduced to break the relationship into two one-to-many relationships.

Indexing

What is Indexing?
Just like an index in a book lists important keywords along with the page numbers where they can be found, an index in a database is a data structure that helps the database quickly locate rows in a table based on the values of certain columns.

Advantages of Indexing:

- Faster Retrieval: With indexing, the database can quickly find the rows that match a specific value or range of values in the indexed column(s), similar to flipping directly to the page you need in a book.
- Improved Performance: By reducing the amount of data that needs to be scanned, indexing can significantly speed up query execution time, making your database queries run faster.
- Efficient Sorting: Indexes can also speed up sorting operations because the database can use the index to retrieve rows in a pre-sorted order.
- Enforcement of Constraints: Indexes can be used to enforce uniqueness constraints, such as ensuring that no two rows have the same value in a certain column.
- Optimized Joins: Indexes can improve the performance of join operations by enabling the database to quickly find matching rows in the joined tables.

Types of Indexing

1. Primary Indexing
   A primary index, also known as a primary key index, is an index in a database table that uniquely identifies each row in the table. It's like a special marker that ensures each row has a unique identifier, similar to a social security number for individuals or a license plate number for cars.

2. Secondary Indexing
   secondary indexing is a technique used to improve the efficiency of queries by creating additional indexes on columns other than the primary key.

In a relational database, a primary index is typically created on the primary key column(s) of a table to facilitate fast retrieval of records based on that key. However, there are many situations where queries may need to access data based on columns that are not part of the primary key. In such cases, secondary indexing comes into play.

3. Clustered Indexing
   Clustered indexing is a type of indexing technique used in database management systems (DBMS) to physically reorder the records of a table based on the values of the indexed column(s). In a clustered index, the rows of the table are stored on disk in the order specified by the index key(s).

   Here are examples of scenarios where each type of index (primary, secondary, and clustered) might be beneficial:

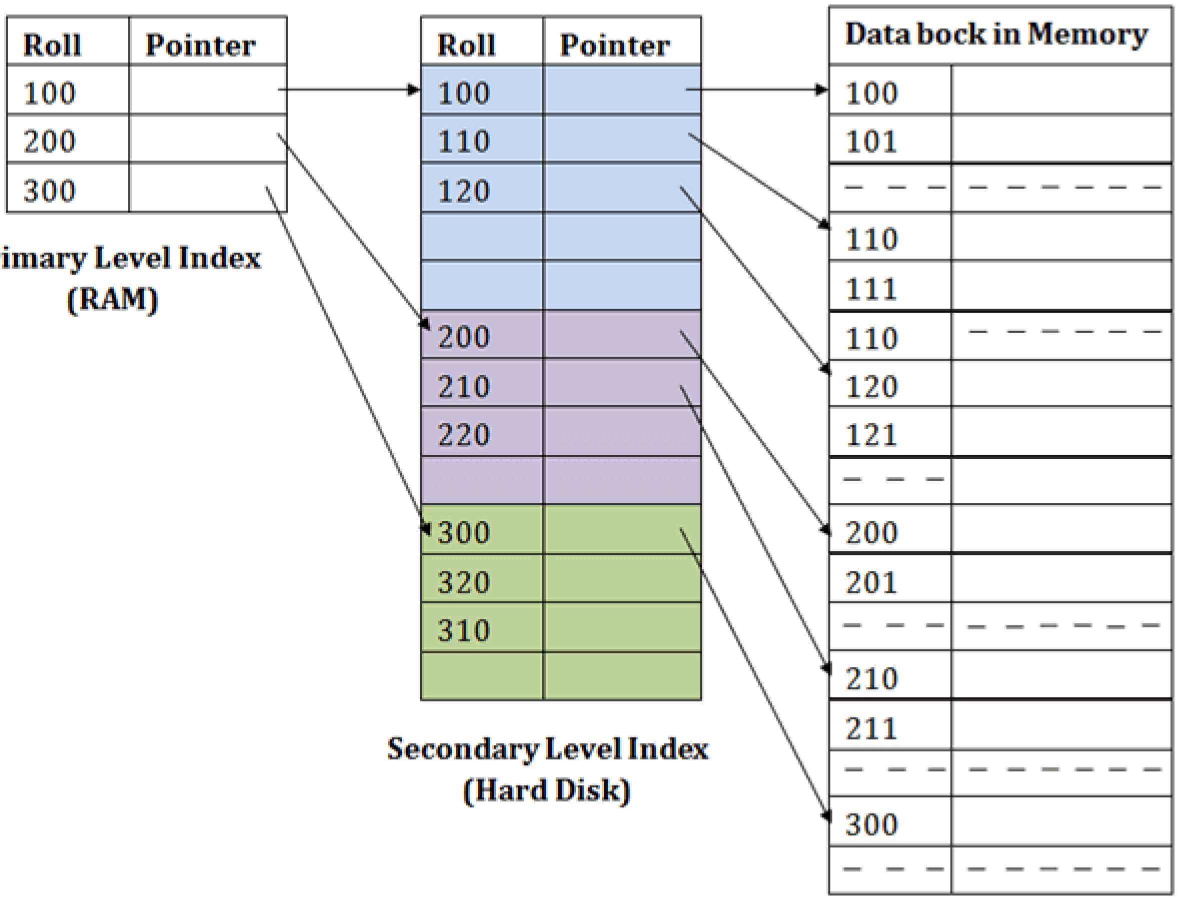
   Primary Index:

   Scenario: You have a table storing employee records, and each employee has a unique employee ID.
   Use Case: You would apply a primary index on the employee ID column.
   Benefits:
   Ensures fast retrieval of individual employee records by their unique IDs.
   Guarantees the uniqueness of employee IDs, preventing duplicates.
   Simplifies and accelerates queries that filter or join data based on the primary key.
   Secondary Index:

   Scenario: In a sales database, you frequently query orders by customer name.
   Use Case: You would apply a secondary index on the customer name column.
   Benefits:
   Improves query performance when filtering orders by customer name.
   Enables efficient retrieval of orders associated with specific customers.
   Facilitates quicker analysis and reporting on customer-specific data.
   Clustered Index:

   Scenario: You have a table containing timestamped sensor data, and queries often retrieve data within specific time ranges.
   Use Case: You would create a clustered index on the timestamp column.
   Benefits:

Organizes the sensor data physically by timestamp, making range queries faster.
Reduces the need for sorting operations when retrieving recent or historical data.
Enhances query performance for time-based analysis and trend detection.



Normalization

Normalization is a database design technique used to organize tables and minimize redundancy by decomposing them into smaller, related tables without losing information. It aims to eliminate data anomalies like insertion, deletion, and update anomalies, thereby ensuring data integrity.

Functional dependency (FD)

1. It's a relationship between the primary key attribute (usually) of the relation to that of the other attribute of the relation.
2. X -> Y, the left side of FD is known as a Determinant, the right side of the production is known as a Dependent.

A. **Types of FD**

1. Trivial FD

A → B has trivial functional dependency if B is a subset of A. A->A, B->B are also Trivial FD.

2. Non-trivial FD

A → B has a non-trivial functional dependency if B is not a subset of A. [A intersection B is NULL].

B. **Rules of FD**

1. Reflexive
If ‘A’ is a set of attributes and ‘B’ is a subset of ‘A’. Then, A→ B holds.
If A ⊇ B then A → B.

2. Augmentation
If B can be determined from A, then adding an attribute to this functional dependency won’t change anything.
If A→ B holds, then AX→ BX holds too. ‘X’ being a set of attributes.

3. Transitivity
If A determines B and B determines C, we can say that A determines C.
if A→ B and B→ C then A→ C.

C. **What happen if we have redundant data?**
Insertion, deletion and updation anomalies arises.

D. **Anomalies**
Anomalies means abnormalities, there are three types of anomalies introduced by data redundancy.

Insertion anomaly
When certain data (attribute) can not be inserted into the DB without the presence of other data.

Deletion anomaly
The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some
other important data.

Updation anomaly (or modification anomaly)

The update anomaly is when an update of a single data value requires multiple rows of data to be updated.
Due to updation to many places, may be Data inconsistency arises, if one forgets to update the data at all the
intended places.

Due to these anomalies, DB size increases and DB performance becomes very slow.
To rectify these anomalies and the effect of these of DB, we use Database optimisation technique called NORMALISATION.

Types Of Normal Forms

First Normal Form (1NF)
1. Every relation cell must have atomic value.
2. Relation must not have multi-valued attributes.
3. Eliminates repeating groups by putting each into a separate table.

Second Normal Form (2NF)
1. Relation must be in 1NF.
2. There should not be any partial dependency.
1. All non-prime attributes must be fully dependent on PK.
2. Non prime attribute can not depend on the part of the PK.

Third Normal Form (3NF)
1. Relation must be in 2NF.
2. No transitivity dependency exists.
3. Any non-prime attribute must be dependent only on the primary key, not on other non-prime attributes.

BCNF (Boyce-Codd normal form)
1. A stricter form of 3NF where every determinant is a candidate key.
2. FD: A -> B, A must be a super key.
3. We must not derive prime attribute from any prime or non-prime attribute.
4. It deals with anomalies caused by functional dependencies where a non-prime attribute determines a candidate key.

Transaction

A. Transaction
1. A unit of work done against the DB in a logical sequence.
2. Sequence is very important in transaction.
3. It is a logical unit of work that contains one or more SQL statements. The result of all these statements in a
transaction either gets completed successfully (all the changes made to the database are permanent) or if at any
point any failure happens it gets rollbacked (all the changes being done are undone.)

B. ACID Properties

1. To ensure integrity of the data, we require that the DB system maintain the following properties of the transaction.

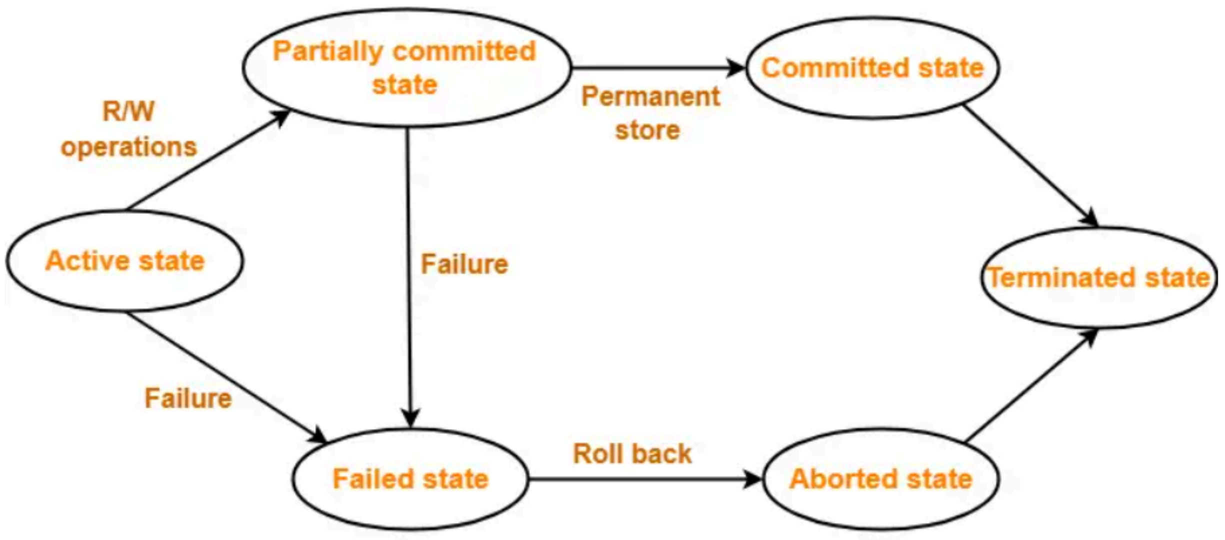
2. Atomicity
1. Either all operations of transaction are reflected properly in the DB, or none are.

3. Consistency
1. Integrity constraints must be maintained before and after transaction.
2. DB must be consistent after transaction happens.

4. Isolation
1. Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of
transactions Ti and Tj, it appears to Ti that either Tj finished execution before Ti started, or Tj started execution
after Ti finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.
2. Multiple transactions can happen in the system in isolation, without interfering each other.

5. Durability
1. After transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

C. Transaction states



Transaction States in DBMS

1. Active state
The very first state of the life cycle of the transaction, all the read and write operations are being performed. If they execute without any error the T comes to Partially committed state. Although if any error occurs then it leads to a Failed state.

2. Partially committed state
After transaction is executed the changes are saved in the buffer in the main memory. If the changes made are permanent on the DB then the state will transfer to the committed state and if there is any failure, the T will go to Failed state.

3. Committed state
When updates are made permanent on the DB. Then the T is said to be in the committed state. Rollback can't be done from the committed states. New consistent state is achieved at this stage.

4. Failed state
When T is being executed and some failure occurs. Due to this it is impossible to continue the execution of the T.

5. Aborted state
When T reaches the failed state, all the changes made in the buffer are reversed. After that the T rollback completely. T reaches abort state after rollback. DB's state prior to the T is achieved.

6. Terminated state
A transaction is said to have terminated if has either committed or aborted.

Types of Databases

1. Relational Databases
Based on Relational Model.
Relational databases are quite popular, even though it was a system designed in the 1970s. Also known as relational database management systems (RDBMS), relational databases commonly use Structured Query Language (SQL) for operations such as creating, reading, updating, and deleting data. Relational databases store information in discrete tables, which can be JOINed together by fields known as foreign keys. For example, you might have a User table which contains information about all your users, and join it to a Purchases table, which contains information about all the purchases they've made. MySQL, Microsoft SQL Server, and Oracle are types of relational databases.
3. they are ubiquitous, having acquired a steady user base since the 1970s
4. they are highly optimised for working with structured data.
5. they provide a stronger guarantee of data normalization
6. they use a well-known querying language through SQL
7. Scalability issues (Horizontal Scaling).
8. Data become huge, system become more complex.

2. Object Oriented Databases
The object-oriented data model, is based on the object-oriented-programming paradigm, which is now in widely use.
Inheritance, object-identity, and encapsulation (information hiding), with methods to provide an interface to objects, are among the key concepts of object-oriented programming that have found applications in data modeling. The object-oriented data model also supports a rich type system, including structured and collection types. While inheritance and, to some extent, complex types are also present in the E-R model, encapsulation and object-identity distinguish the object-oriented data model from the E-R model.
Sometimes the database can be very complex, having multiple relations. So, maintaining a relationship between them can be tedious at times.
1. In Object-oriented databases data is treated as an object.
2. All bits of information come in one instantly available object package instead of multiple tables.
Advantages
1. Data storage and retrieval is easy and quick.
2. Can handle complex data relations and more variety of data types that standard relational databases.
3. Relatively friendly to model the advance real world problems
4. Works with functionality of OOPs and Object Oriented languages.

Disadvantages
1. High complexity causes performance issues like read, write, update and delete operations are slowed down.
2. Not much of a community support as isn't widely adopted as relational databases.
3. Does not support views like relational databases.
4. e.g., ObjectDB, GemStone etc.

3. NoSQL Databases
NoSQL databases (aka "not only SQL") are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of data and high user loads.
2. They are schema free.
3. Data structures used are not tabular, they are more flexible, has the ability to adjust dynamically.
4. Can handle huge amount of data (big data).
5. Most of the NoSQL are open sources and has the capability of horizontal scaling.
6. It just stores data in some format other than relational.

4. Hierarchical Databases
As the name suggests, the hierarchical database model is most appropriate for use cases in which the main focus of information
gathering is based on a concrete hierarchy, such as several individual employees reporting to a single department at a
Company.
The schema for hierarchical databases is defined by its tree-like organization, in which there is typically a root "parent" directory of data stored as records that links to various other subdirectory branches, and each subdirectory branch, or child record, may link to various other subdirectory branches.
The hierarchical database structure dictates that, while a parent record can have several child records, each child record can only have one parent record. Data within records is stored in the form of fields, and each field can only contain one value. Retrieving hierarchical data from a hierarchical database architecture requires traversing the entire tree, starting at the root node. Since the disk storage system is also inherently a hierarchical structure, these models can also be used as physical models.
The key advantage of a hierarchical database is its ease of use. The one-to-many organization of data makes traversing the database simple and fast, which is ideal for use cases such as website drop-down menus or computer folders in systems like
Microsoft Windows OS. Due to the separation of the tables from physical storage structures, information can easily be added or deleted without affecting the entirety of the database. And most major programming languages offer functionality for reading tree structure databases.

The major disadvantage of hierarchical databases is their inflexible nature. The one-to-many structure is not ideal for complex structures as it cannot describe relationships in which each child node has multiple parents nodes. Also the tree-like organization of data requires top-to-bottom sequential searching, which is time consuming, and requires repetitive storage of data in multiple different entities, which can be redundant.
e.g., IBM IMS.

5. Network Databases
1. Extension of Hierarchical databases
2. The child records are given the freedom to associate with multiple parent records.
3. Organized in a Graph structure.
4. Can handle complex relations.
5. Maintenance is tedious.
6. M:N links may cause slow retrieval.
7. Not much web community support.
8. e.g., Integrated Data Store (IDS), IDMS (Integrated Database Management System), Raima Database Manager, TurboIMAGE etc.

Clustering

1**. Database Clustering** (making Replica-sets) is the process of combining more than one servers or instances connecting a single database. Sometimes one server may not be adequate to manage the amount of data or the number of requests, that is when a Data Cluster is needed. Database clustering, SQL server clustering, and SQL clustering are closely associated with SQL is the language used to manage the database information.

2. Replicate the same dataset on different servers.

3. Advantages
1. Data Redundancy: Clustering of databases helps with data redundancy, as we store the same data at multiple servers. Don’t confuse this data redundancy as repetition of the same data that might lead to some anomalies. The redundancy that clustering offers is required and is quite certain due to the synchronization. In case any of the servers had to face a failure due to any possible reason, the data is available at other servers to access.

2. Load balancing: or scalability doesn’t come by default with the database. It has to be brought by clustering regularly. It also depends on the setup. Basically, what load balancing does is allocating the workload among the different servers that are part of the cluster. This indicates that more users can be supported and if for some reasons if a huge spike in the traffic appears, there is a higher assurance that it will be able to support the new traffic. One machine is not going to get all of the hits. This can provide scaling seamlessly as required. This links directly to

high availability. Without load balancing, a particular machine could get overworked and traffic would slow down, leading to decrement of the traffic to zero.

3. High availability: When you can access a database, it implies that it is available. High availability refers the amount of time a database is considered available. The amount of availability you need greatly depends on the number of transactions you are running on your database and how often you are running any kind of analytics on your data. With database clustering, we can reach extremely high levels of availability due to load balancing and have extra machines. In case a server got shut down the database will, however, be available.

4. How does Clustering Work?
1. In cluster architecture, all requests are split with many computers so that an individual user request is executed and produced by a number of computer systems. The clustering is serviceable definitely by the ability of load balancing and high-availability. If one node collapses, the request is handled by another node. Consequently, there are few or no possibilities of absolute system failures.

Partitioning and Sharding

1. **A big problem** can be solved easily when it is chopped into several smaller sub-problems. That is what the partitioning technique does. It divides a big database containing data metrics and indexes into smaller and handy slices of data called partitions. The partitioned tables are directly used by SQL queries without any alteration. Once the database is partitioned, the data definition language can easily work on the smaller partitioned slices, instead of handling the giant database altogether. This is how partitioning cuts down the problems in managing large database tables.

2. **Partitioning** is the technique used to divide stored database objects into separate servers. Due to this, there is an increase in performance, controllability of the data. We can manage huge chunks of data optimally. When we horizontally scale our machines/servers, we know that it gives us a challenging time dealing with relational databases as it’s quite tough to maintain the relations. But if we apply partitioning to the database that is already scaled out i.e. equipped with multiple servers, we can partition our database among those servers and handle the big data easily.
3. Vertical Partitioning
1. Slicing relation vertically / column-wise.
2. Need to access different servers to get complete tuples.
4. Horizontal Partitioning
1. Slicing relation horizontally / row-wise.
2. Independent chunks of data tuples are stored in different servers.

5. When Partitioning is Applied?
1. Dataset become much huge that managing and dealing with it become a tedious task.
2. The number of requests are enough larger that the single DB server access is taking huge time and hence the system's response time become high.
6. Advantages of Partitioning
1. Parallelism
2. Availability
3. Performance
4. Manageability
5. Reduce Cost, as scaling-up or vertical scaling might be costly.
7. Distributed Database
1. A single logical database that is, spread across multiple locations (servers) and logically interconnected by network.
2. This is the product of applying DB optimisation techniques like Clustering, Partitioning and Sharding.
3. Why this is needed? READ Point 5.
8. Sharding
1. Technique to implement Horizontal Partitioning.
2. The fundamental idea of Sharding is the idea that instead of having all the data sit on one DB instance, we split it up and introduce a
Routing layer so that we can forward the request to the right instances that actually contain the data.
Pros
1. Scalability
2. Availability

Cons
1. Complexity, making partition mapping, Routing layer to be implemented in the system, Non-uniformity that creates the necessity of Re-Sharding
2. Not well suited for Analytical type of queries, as the data is spread across different DB instances.

SQL

SQL: Structured Query Language, used to access and manipulate data.
SQL used CRUD operations to communicate with DB.
1. CREATE - execute INSERT statements to insert new tuple into the relation.
2. READ - Read data already in the relations.
3. UPDATE - Modify already inserted data in the relation.
4. DELETE - Delete specific data point/tuple/row or multiple rows.

SQL is not DB, is a query language.

What is RDBMS? (Relational Database Management System)
1. Software that enable us to implement designed relational model.
2. e.g., MySQL, MS SQL, Oracle, IBM etc.
3. Table/Relation is the simplest form of data storage object in R-DB.
4. MySQL is open-source RDBMS, and it uses SQL for all CRUD operations
5. MySQL used client-server model, where client is CLI or frontend that used services provided by MySQL server.

Difference between SQL and MySQL
1. SQL is Structured Query language used to perform CRUD operations in R-DB, while MySQL is a RDBMS used to
store, manage and administrate DB (provided by itself) using SQL.

Types of SQL commands
DDL (data definition language): defining relation schema.
1. CREATE: create table, DB, view.
2. ALTER TABLE: modification in table structure. e.g, change column datatype or add/remove columns.
3. DROP: delete table, DB, view.
4. TRUNCATE: remove all the tuples from the table.
5. RENAME: rename DB name, table name, column name etc.

DRL/DQL (data retrieval language / data query language): retrieve data from the tables.
1. SELECT

DML (data modification language): use to perform modifications in the DB
1. INSERT: insert data into a relation
2. UPDATE: update relation data.
3. DELETE: delete row(s) from the relation.

DCL (Data Control language): grant or revoke authorities from user.
1. GRANT: access privileges to the DB
2. REVOKE: revoke user access privileges.

TCL (Transaction control language): to manage transactions done in the DB
1. COMMIT: apply all the changes and end transaction
2. ROLLBACK: discard changes and end transaction

Managing DB

Create database
CREATE DATABASE your_database_name;

Create table

```
CREATE TABLE your_table_name (
    column1_name datatype1,
    column2_name datatype2,
    column3_name datatype3,
    ...
);
```

Insert values

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...),
(value1, value2, value3, ...),
(value1, value2, value3, ...);
```

Drop Database

DROP: Used to remove an object from the database entirely. This could be a table, index, view, or even the database itself.
Example of dropping a table:

```
DROP TABLE table_name;
```

DELETE: Used to remove specific rows from a table based on a condition.
Example of deleting rows from a table:

```
DELETE FROM table_name WHERE condition;
```

TRUNCATE: Used to remove all rows from a table. It's faster than `DELETE` because it doesn't generate individual delete logs for each row; rather, it deallocates the whole data page at once.
Example of truncating a table:

```
TRUNCATE TABLE table_name;
```

Update table
UPDATE: Used to modify existing records in a table.
Syntax:
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;

Example:
UPDATE users

SET email = 'new_email@example.com'
WHERE user_id = 1;

ALTER TABLE: Used to modify an existing table structure.
Syntax:
ALTER TABLE table_name
action;

Examples:

- Adding a new column:
  ALTER TABLE users
  ADD COLUMN phone_number VARCHAR(15);

Modifying a column:
ALTER TABLE users
MODIFY COLUMN email VARCHAR(255) NOT NULL;

Dropping a column:
ALTER TABLE users
DROP COLUMN phone_number;

Select
The SELECT statement in SQL is used to retrieve data from a database. It allows you to specify which columns to retrieve and which rows to include based on specified conditions.

SELECT column1, column2, ...
FROM table_name
WHERE condition;

Selecting all columns from a table:
SELECT *
FROM products;

Selecting specific columns without any condition:
SELECT product_name, price
FROM products;

Selecting all columns with a condition:

```
SELECT *
FROM orders
WHERE order_date >= '2023-01-01';
```

BETWEEN

SELECT * FROM customer WHERE age between 0 AND 100;

In the above e.g., 0 and 100 are inclusive.

IN

Reduces OR conditions;

e.g., SELECT * FROM officers WHERE officer_name IN ('Lakshay', ‘Maharana Pratap', ‘Deepika’);

AND , OR and NOT

n SQL, the logical operators AND, OR, and NOT are used in conjunction with the WHERE clause to filter rows based on multiple conditions.

AND: It retrieves rows that satisfy all the specified conditions.

OR: It retrieves rows that satisfy at least one of the specified conditions.

NOT: It retrieves rows for which the specified condition is not true.

Let's illustrate these operators with examples:

Using AND:

```
SELECT *
FROM users
WHERE age > 25 AND city = 'New York';
```

Using OR:

```
SELECT *
FROM products
WHERE category = 'Electronics' OR category = 'Clothing';
```

Using NOT:

```
SELECT *
FROM orders
```

```
WHERE NOT status = 'completed';
```

You can also combine these operators to create more complex conditions. For example:

```
SELECT *
FROM products
WHERE (category = 'Electronics' OR category = 'Clothing') AND NOT price > 1000;
```

Aggregate Functions

1. COUNT()Returns the number of rows that match a specified condition.

```
SELECT COUNT(*) AS total_users FROM users;
```

2. SUM()Returns the sum of values in a column.

```
SELECT SUM(sales_amount) AS total_sales FROM sales;
```

3. AVG() Returns the average of values in a column.

```
SELECT AVG(salary) AS average_salary FROM employees;
```

4. MIN() Returns the minimum value in a column.

```
SELECT MIN(product_price) AS cheapest_product_price FROM products;
```

5. MAX() Returns the maximum value in a column.

```
SELECT MAX(product_price) AS most_expensive_product_price FROM products;
```

Groupby and Having

The GROUP BY clause in SQL is used to group rows that have the same values into summary rows, typically for use with aggregate functions like COUNT, SUM, AVG, etc. It allows you to perform aggregate functions on a set of rows rather than on a single row.

Suppose you have a table named orders which contains information about orders placed by customers:

order_id	customer_id	total_amount	order_date
1	101	50.00	2023-01-01
2	102	75.00	2023-01-02
3	101	100.00	2023-01-03
4	103	30.00	2023-01-04
5	102	45.00	2023-01-05

Now, let's say you want to find out the total amount of orders placed by each customer. You can use the GROUP BY clause to group the rows by customer_id and then calculate the total amount for each customer using the SUM aggregate function:

sql
Copy code
```
SELECT customer_id, SUM(total_amount) AS total_order_amount
FROM orders
GROUP BY customer_id;
```
The result of this query would be:

```
customer_id    total_order_amount
101      150.00
```

Having
The HAVING clause in SQL is used to filter groups of rows returned by the GROUP BY clause. While the WHERE clause is used to filter rows before any grouping takes place, the HAVING clause is applied after the grouping has occurred.

Now, let's say you want to find customers who have placed more than one order. You can use the GROUP BY clause to group the rows by customer_id and then use the HAVING clause to filter out groups with only one order:

```
SELECT customer_id, COUNT(*) AS order_count
FROM orders
GROUP BY customer_id
HAVING COUNT(*) > 1;
```

The HAVING clause in this query filters out groups where the count of orders (COUNT(*)) is greater than 1.

SQL Join
JOINING TABLES
1. All RDBMS are relational in nature, we refer to other tables to get meaningful outcomes.
2. FK are used to do reference to other table.
3. INNER JOIN
1. Returns a resultant table that has matching values from both the tables or all the tables.
2. SELECT column-list FROM table1 INNER JOIN table2 ON condition1
INNER JOIN table3 ON condition2...;

3. **Alias in MySQL (AS)**

1. Aliases in MySQL is used to give a temporary name to a table or a column in a table for the purpose of
a particular query. It works as a nickname for expressing the tables or column names. It makes the query short

and neat.
2. SELECT col_name AS alias_name FROM table_name;
3. SELECT col_name1, col_name2,... FROM table_name AS alias_name;
4. OUTER JOIN
1. LEFT JOIN
1. This returns a resulting table that all the data from left table and the matched data from the right table.
2. SELECT columns FROM table LEFT JOIN table2 ON Join_Condition;
2. RIGHT JOIN
1. This returns a resulting table that all the data from right table and the matched data from the left table.
2. SELECT columns FROM table RIGHT JOIN table2 ON join_cond;
3. FULL JOIN
1. This returns a resulting table that contains all data when there is a match on left or right table data.

Constraints
In SQL, constraints are rules enforced on columns or tables to ensure the data integrity and consistency within a database.
NOT NULL Constraint: Ensures that a column cannot have a NULL value.
UNIQUE Constraint: Ensures that all values in a column (or a set of columns) are unique.
PRIMARY KEY Constraint: Specifies a column (or a set of columns) as the primary key for a table. It uniquely identifies each row in the table.
FOREIGN KEY Constraint: Establishes a relationship between two tables. It ensures that the values in a column (or a set of columns) in one table correspond to the values in another table's column.
CHECK Constraint: Specifies a condition that must be met for all values in a column.

Example:
```
CREATE TABLE products (
   product_id INT PRIMARY KEY,
   product_name VARCHAR(100) NOT NULL,
   category_id INT,
   price DECIMAL(10, 2) NOT NULL CHECK (price >= 0),
   UNIQUE (product_name),
   FOREIGN KEY (category_id) REFERENCES categories(category_id)
);
```

Stored Procedures

A stored procedure is a set of SQL statements that are stored in the database and can be executed repeatedly by invoking the procedure's name. Stored procedures can accept parameters, perform operations, and return results. They are often used to encapsulate business logic, perform complex calculations, or execute multiple SQL statements in a single operation.

```
CREATE PROCEDURE procedure_name
    [ (parameter1 datatype, parameter2 datatype, ...) ]
AS
BEGIN
    -- SQL statements
END;
```

Here are the advantages of stored procedures explained in simple language:

Reusability: Think of a stored procedure like a recipe that you can use over and over again. Once you create it, you can call it whenever you need to perform a specific task, like cooking your favorite dish. This saves you time and effort because you don't have to write the same code repeatedly.

Performance: When you cook a meal using a recipe, you follow a set of steps without having to think too much about it. Similarly, when you use a stored procedure, the database already knows what to do because it has the instructions stored inside. This can make things faster because the database doesn't have to figure out what to do each time you call the procedure.

Security: Imagine if only certain people knew the secret recipe to your favorite dish. Stored procedures can work similarly by allowing you to control who has access to certain tasks in your database. This helps keep your data safe and secure.

Maintenance: Just like how you might tweak a recipe over time to make it even better, you can update a stored procedure to improve how it works. Because the procedure is stored centrally in the database, any changes you make will apply wherever the procedure is used. This makes it easier to maintain and update your database over time.

Encapsulation: Stored procedures let you package up complex logic into a single, easy-to-use unit. This is like putting all the steps for a complicated dish into one recipe card. It keeps your code organized and makes it easier to understand and work with.

View

A view in SQL is a virtual table based on the result set of a SELECT statement. Unlike a physical table, a view does not store data itself but rather provides a way to dynamically present data from one or more underlying tables or other views. Views can simplify complex queries, provide a layer of security by restricting access to certain columns or rows, and abstract away the complexity of underlying table structures.

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Imagine you have a large spreadsheet with lots of data, and you often need to look at only certain parts of it or perform calculations on specific columns. Instead of scrolling through the entire spreadsheet every time, you can create a smaller, more focused view that shows only the information you need.

In the context of databases:

Data Simplification: Views let you simplify complex data. Instead of dealing with all the details of a table, you can create a view that shows only the specific columns or rows you're interested in.

Data Security: Views can also help with security. For instance, if a table contains sensitive information, you can create a view that only shows the less sensitive columns, and then grant access to that view instead of the entire table.

Data Aggregation: Views can aggregate data. For example, you can create a view that calculates the total sales for each month, so you don't have to manually perform this calculation every time you need it.

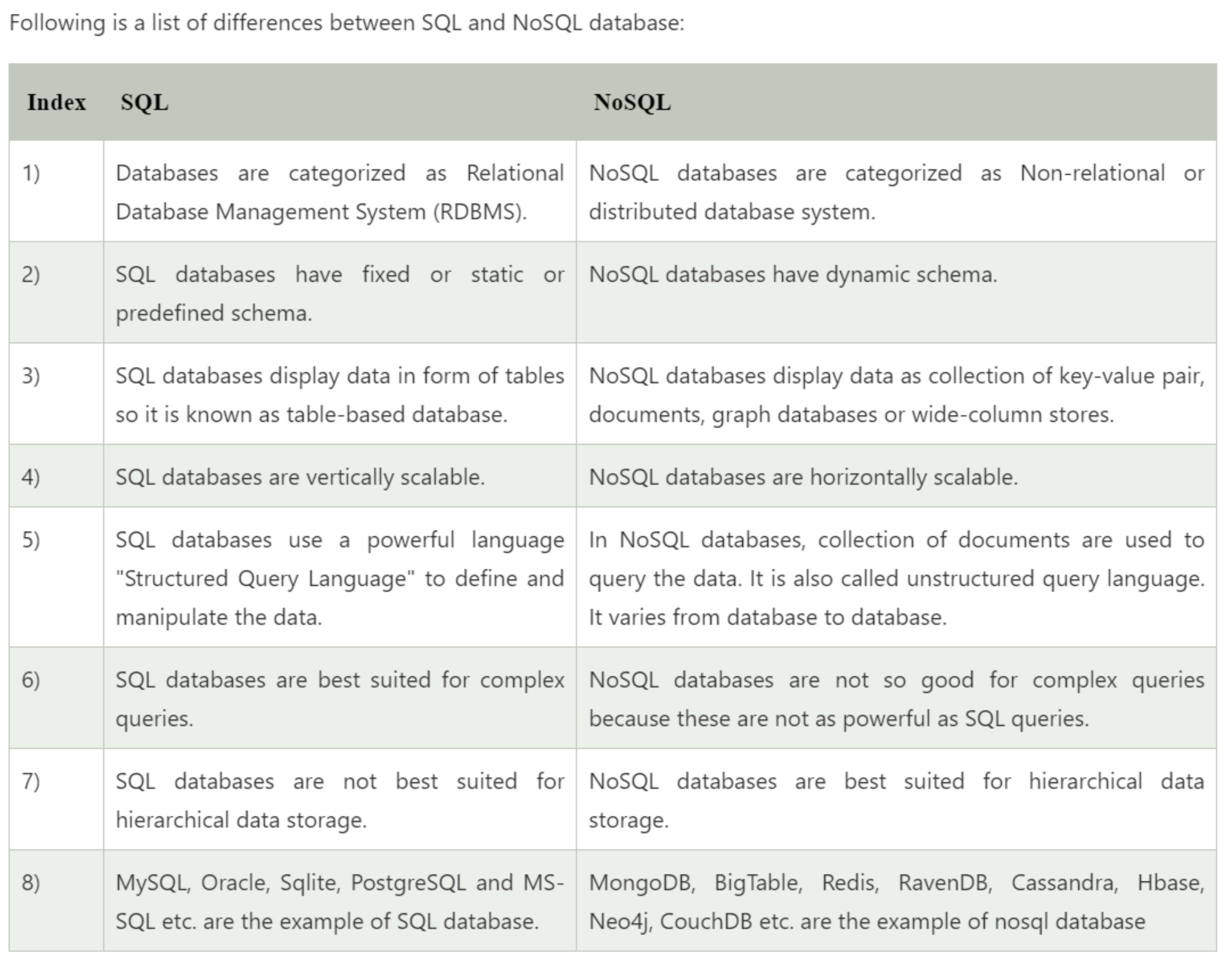
Simplified Queries: Instead of writing complex SQL queries every time you need certain information, you can create a view with the query, and then simply reference the view whenever you need that data.

Functions

In SQL, functions are reusable code blocks that accept input parameters, perform calculations, and return a single value. SQL supports various types of functions, including scalar functions, aggregate functions, and table-valued functions

```
CREATE FUNCTION getFullName (first_name VARCHAR(50), last_name VARCHAR(50))
RETURNS VARCHAR(100)
AS
BEGIN
   RETURN first_name + ' ' + last_name;
END;
```

SQL vs NoSql



Following is a list of differences between SQL and NoSQL database:

Index	SQL	NoSQL
1)	Databases are categorized as Relational Database Management System (RDBMS).	NoSQL databases are categorized as Non-relational or distributed database system.
2)	SQL databases have fixed or static or predefined schema.	NoSQL databases have dynamic schema.
3)	SQL databases display data in form of tables so it is known as table-based database.	NoSQL databases display data as collection of key-value pair, documents, graph databases or wide-column stores.
4)	SQL databases are vertically scalable.	NoSQL databases are horizontally scalable.
5)	SQL databases use a powerful language "Structured Query Language" to define and manipulate the data.	In NoSQL databases, collection of documents are used to query the data. It is also called unstructured query language. It varies from database to database.
6)	SQL databases are best suited for complex queries.	NoSQL databases are not so good for complex queries because these are not as powerful as SQL queries.
7)	SQL databases are not best suited for hierarchical data storage.	NoSQL databases are best suited for hierarchical data storage.
8)	MySQL, Oracle, Sqlite, PostgreSQL and MS-SQL etc. are the example of SQL database.	MongoDB, BigTable, Redis, RavenDB, Cassandra, Hbase, Neo4j, CouchDB etc. are the example of nosql database

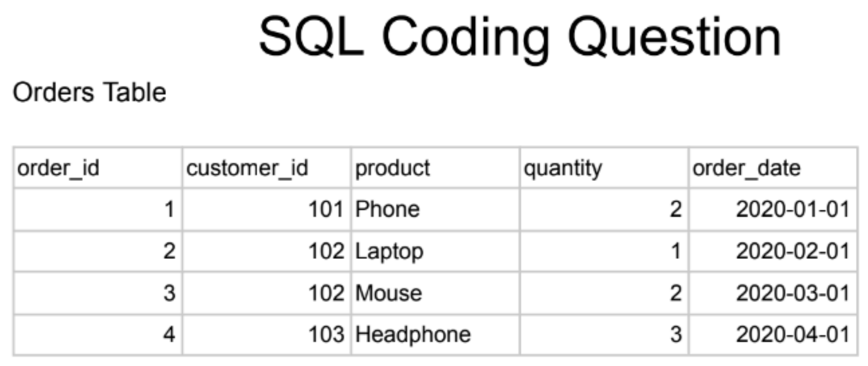
Interview Related questions

1. What is a DBMS, and why is it important in modern software development?
2. Differentiate between DBMS and RDBMS.
3. Explain the ACID properties in the context of database transactions.
4. What are the various types of database models? Explain each briefly.
5. Describe normalization and its importance in database design.
6. What is a primary key? How does it differ from a unique key?
7. Discuss the various types of joins in SQL.
8. Explain the differences between DELETE, TRUNCATE, and DROP commands in SQL.
9. What are indexes in a database? Why are they used, and what are their types?
10. What is a foreign key? How does it relate to the concept of referential integrity?
11. Describe the difference between a clustered and a non-clustered index.
12. What is a stored procedure? How does it differ from a function?
13. Explain the concept of a transaction in a database. How do you ensure transaction atomicity?
14. What is SQL injection, and how can it be prevented?
15. Describe the concept of data concurrency and how it's managed in a DBMS.
16. Explain the difference between optimistic and pessimistic concurrency control.
17. What is a view in a database? How is it different from a table?
18. Discuss the advantages and disadvantages of using NoSQL databases compared to relational databases.
19. How does a DBMS handle concurrent access by multiple users?
20. Explain the concept of data warehousing and its significance.
21. What is the difference between OLTP and OLAP databases?
22. How does replication work in a distributed database environment?
23. Describe the CAP theorem and its implications on distributed database systems.
24. Explain the concept of data mining and its relevance to database systems.
25. What are some common database design patterns and anti-patterns?
26. What is the difference between a database and a schema?
27. Explain the concept of a composite key and when it might be used.
28. Discuss the advantages and disadvantages of using stored procedures.
29. Explain the concept of data redundancy and how it can be minimized in database design.
30. What is the purpose of the COMMIT and ROLLBACK commands in SQL?
31. Describe the difference between a candidate key, a primary key, and a super key.
32. What is a deadlock in the context of database systems? How can it be detected and resolved?
33. Discuss the concept of normalization forms and their significance in database design.
34. Explain the difference between a left outer join and a right outer join.

35. What are the different types of SQL constraints? Provide examples of each.
36. Describe the concept of database indexing and how it improves query performance.
37. What is a materialized view, and how does it differ from a regular view?
38. Explain the concept of database transactions isolation levels.
39. Discuss the concept of database replication and its importance in high availability scenarios.
40. What is the purpose of the GROUP BY clause in SQL? Provide an example query.
41. Describe the difference between a heap table and a clustered table.
42. Explain the concept of database normalization and its various normal forms.
43. Discuss the role of the DBA (Database Administrator) in managing a database system.
44. What is denormalization, and when is it appropriate to use in database design?
45. Describe the process of database backup and recovery.
46. Explain the concept of a data dictionary in the context of database management.
47. Discuss the differences between a relational database and an object-oriented database.
48. What is a subquery in SQL? Provide an example scenario where a subquery is useful.
49. What is the purpose of the HAVING clause in SQL? How does it differ from the WHERE clause?
50. What is the difference between the TRUNCATE and DELETE commands in SQL?

SQL Queries Question

The tables are given and these are the most important interview questions. To practice you can create the table in your system as well.



SQL Coding Question

Orders Table

order_id	customer_id	product	quantity	order_date
1	101	Phone	2	2020-01-01
2	102	Laptop	1	2020-02-01
3	102	Mouse	2	2020-03-01
4	103	Headphone	3	2020-04-01

1. How would you select all columns from the orders table?
Answer

```
SELECT * FROM orders;
```

2. How would you select all columns from the orders table but only the rows where the quantity is greater than 1?
Answer

```
SELECT * FROM orders WHERE quantity > 1;
```

3. How would you select the product and quantity columns from the orders table where the order_date is between '2020-02-01' and '2020-03-31'?

Answer

```
SELECT product, quantity FROM orders
WHERE order_date BETWEEN '2020-02-01' AND '2020-03-31';
```

4. How would you count the number of orders in the orders table?

Answer

```
SELECT COUNT(*) FROM orders;
```

5. How would you select the unique customer_id values from the orders table?
Answer

```
SELECT DISTINCT customer_id FROM orders;
```

6. How would you calculate the total quantity of items ordered in the orders table?
Answer

```
SELECT SUM(quantity) FROM orders;
```

7. How would you find the average quantity of items ordered per order in the orders table?
Answer

```
SELECT AVG(quantity) FROM orders;
```

8. How would you select the product, quantity, and order_date columns from the orders table and order the results by the order_date in ascending order? **Answer**

```
SELECT product, quantity, order_date FROM orders
ORDER BY order_date;
```

9. How would you update the quantity to 4 for the order with an order_id of 2 in the orders table?
Answer

```
UPDATE orders SET quantity = 4 WHERE order_id = 2;
```

10. How would you delete the order with an order_id of 3 in the orders table?
Answer

```
DELETE FROM orders WHERE order_id = 3;
```

11. How would you insert a new order into the orders table with order_id 5, customer_id 104, product 'Keyboard', quantity 3, and order_date '2020-05-01'?
Answer

```
INSERT INTO orders (order_id, customer_id, product, quantity,
order_date)
VALUES (5, 104, 'Keyboard', 3, '2020-05-01');
```

12. How would you select the product and order_date columns from the orders table where the customer_id is 102?
Answer

```
SELECT product, order_date FROM orders
WHERE customer_id = 102;
```

13. How would you find the maximum quantity ordered in the orders table?
Answer

```
SELECT MAX(quantity) FROM orders;
```

14. How would you find the minimum quantity ordered in the orders table?
Answer

```
SELECT MIN(quantity) FROM orders;
```

15. How would you select the product and quantity columns from the orders table, but only for orders with a quantity greater than the average quantity of all orders in the table?
Answer

```
SELECT product, quantity FROM orders
WHERE quantity > (SELECT AVG(quantity) FROM orders);
```

16. How would you find the most frequently ordered product in the orders table?

Answer

```
SELECT product, COUNT(product) FROM orders
GROUP BY product
ORDER BY COUNT(product) DESC
LIMIT 1;
```

17. How would you join the orders table with a customers table to get all the customer information for each order in the orders table?

Answer

```
CREATE TABLE customers (
  customer_id INT PRIMARY KEY,
  name VARCHAR(50),
  address VARCHAR(100)
);

SELECT * FROM orders
JOIN customers
ON orders.customer_id = customers.customer_id;
```

18. How would you find the total quantity of each product ordered for each customer in the orders table?

Answer

```
SELECT customers.name, orders.product, SUM(orders.quantity)
FROM orders
JOIN customers
ON orders.customer_id = customers.customer_id
GROUP BY customers.name, orders.product;
```

19. Write a SQL query to find the total number of orders for each customer.

orderid	customer	date
1	A	1/1/2022
2	A	2/2/2022
3	B	3/3/2022
4	B	4/4/2022
5	C	5/5/2022

```
SELECT customer, count(*) as total_orders
FROM orders
GROUP BY customer;
```

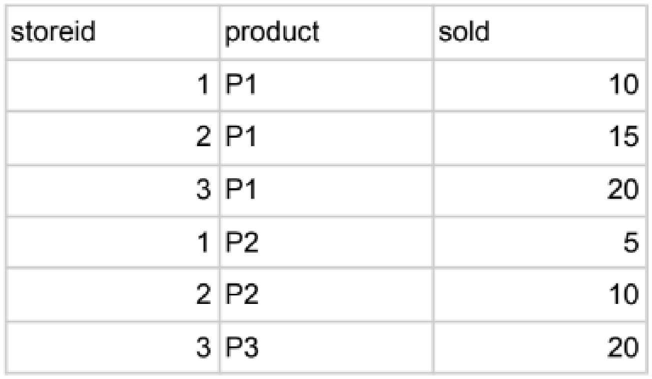
20. Write a SQL query to find the second highest salary of all employees in a company.

empid	name	salary
1	John	5000
2	Sarah	6000
3	Michael	7000
4	David	8000
5	Alice	9000

```
SELECT MAX(salary)

FROM (
SELECT salary
FROM employees
ORDER BY salary DESC
LIMIT 2
) AS second_highest_salary;
```

21. Write a SQL query to find the products that have been sold in all stores.
Dataset:

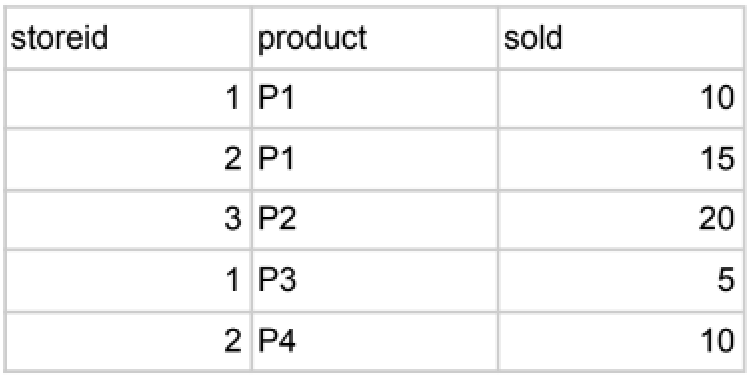


storeid	product	sold
1	P1	10
2	P1	15
3	P1	20
1	P2	5
2	P2	10
3	P3	20

```
SELECT product

FROM sales
GROUP BY product
HAVING COUNT(DISTINCT storeid) = (SELECT COUNT(DISTINCT storeid)
FROM sales);
```

22. Write a SQL query to find the products that have never been sold.
Dataset:



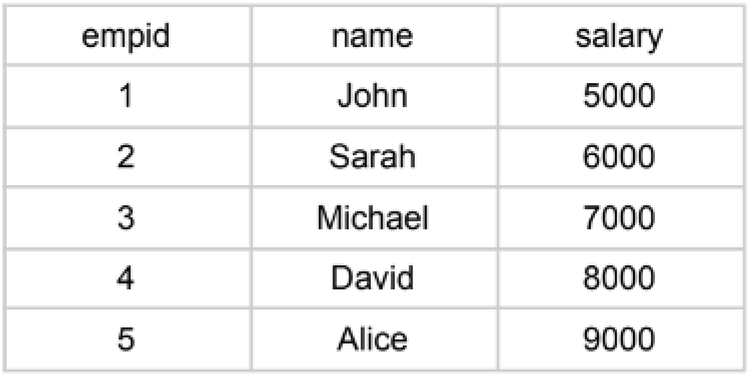
storeid	product	sold
1	P1	10
2	P1	15
3	P2	20
1	P3	5
2	P4	10

Answer-

```
SELECT product

FROM products
WHERE product NOT IN (SELECT product FROM sales);
```

23. Write a SQL query to find the names of the employees who have a salary greater
than the average salary of all employees.
Dataset:

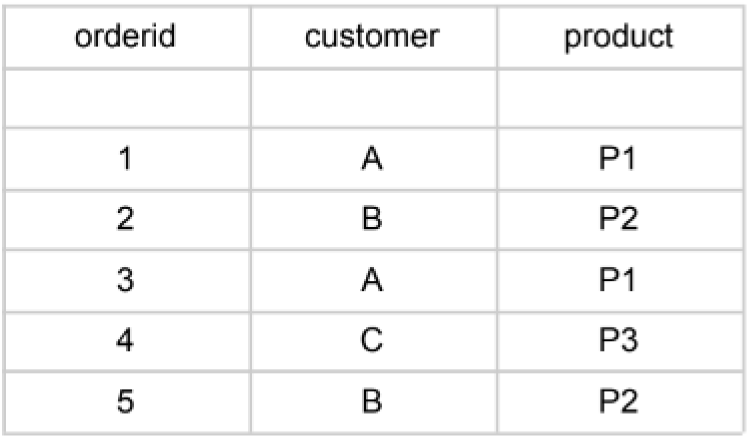


empid	name	salary
1	John	5000
2	Sarah	6000
3	Michael	7000
4	David	8000
5	Alice	9000

```
SELECT name, salary

FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

24. Write a SQL query to find the most popular product among customers who have made more than 5 orders in the past year.



orderid	customer	product
1	A	P1
2	B	P2
3	A	P1
4	C	P3
5	B	P2

```
SELECT product, COUNT(product) as product_count

FROM orders
WHERE customer IN (SELECT customer
FROM orders
GROUP BY customer
HAVING COUNT(*) > 5)
AND date >= DATE_SUB(NOW(), INTERVAL 1 YEAR)
GROUP BY product
ORDER BY product_count DESC
LIMIT 1;
```

25. Write a query to retrieve the name and salary of the highest-paid

employee in each job title.

```
SELECT job_title, name, salary

FROM employees
WHERE salary = (SELECT MAX(salary) FROM employees WHERE job_title
= employees.job_title);
```

26. Weekly_sales Table

week_date	region	platform	segment	customer_type	transactions	sales
31/8/20	ASIA	Retail	C3	New	120631	3656163
31/8/20	ASIA	Retail	F1	New	31574	996575
31/8/20	USA	Retail	null	Guest	529151	16509610
31/8/20	EUROPE	Retail	C1	New	4517	141942
31/8/20	AFRICA	Retail	C2	New	58046	1758388
31/8/20	CANADA	Shopify	F2	Existing	1336	243878
31/8/20	AFRICA	Shopify	F3	Existing	2514	519502
31/8/20	ASIA	Shopify	F1	Existing	2158	371417
31/8/20	AFRICA	Shopify	F2	New	318	49557
31/8/20	AFRICA	Retail	C3	New	111032	3888162

1. What day of the week is used for each week_date value?

Answer-
```
SELECT

DISTINCT date_part('dow', week_day)::int AS day_of_week,
to_char(week_day, 'Day') AS day_of_week_name
FROM clean_weekly_sales;
```

2. How many total transactions were there for each year in the dataset?

Answer-
```
SELECT calendar_year,

sum(transactions) AS total_transactions
FROM clean_weekly_sales
GROUP BY calendar_year
```

```
ORDER BY calendar_year;
```

3. What is the total sales for each region for each month?

Answer-

```
SELECT region,

calendar_year,
month_number,
sum(sales) AS total_sales
FROM clean_weekly_sales
GROUP BY region,
calendar_year,
month_number
ORDER BY calendar_year,
month_number,
region;
```

4. What is the total count of transactions for each platform?

Answer-

```
SELECT platform,

sum(transactions) AS total_transactions
FROM clean_weekly_sales
GROUP BY platform;
```

5. What is the percentage of sales by demographic for each year in the dataset?

Answer-

```
SELECT calendar_year,

demographics,
sum(sales) AS sales_per_demographic,
round(
100 * sum(sales) / sum(sum(sales)) OVER (PARTITION BY

calendar_year),
2
) AS percentage
FROM clean_weekly_sales
```

```
GROUP BY demographics,
calendar_year
ORDER BY calendar_year,
demographics;
```

Assignment Questions

(Answer to the question that was given in class)

SQL Query 1: Find all employees along with their department names

```
SELECT e.name AS employee_name, d.department_name
FROM Employees e
JOIN Departments d ON e.department_id = d.department_id;
```

SQL Query 2: Find the number of employees in each department

```
SELECT d.department_name, COUNT(e.employee_id) AS num_employees
FROM Departments d
LEFT JOIN Employees e ON d.department_id = e.department_id
GROUP BY d.department_name;
```

Query 3: Find all employees who have not been assigned to any department

```
SELECT e.name AS employee_name
FROM Employees e
LEFT JOIN Departments d ON e.department_id = d.department_id
WHERE d.department_id IS NULL;
```

QL Query 4: Find the average salary of employees in each department
Assuming there's a salary column in the Employees table.

```
SELECT d.department_name, AVG(e.salary) AS avg_salary
FROM Departments d
LEFT JOIN Employees e ON d.department_id = e.department_id
GROUP BY d.department_name;
```

SQL Query 5: Find the department with the highest number of employees

```
SELECT d.department_name, COUNT(e.employee_id) AS num_employees
FROM Departments d
LEFT JOIN Employees e ON d.department_id = e.department_id
GROUP BY d.department_name
ORDER BY num_employees DESC
LIMIT 1;
```

SQL Project

Here's an example of a dataset you can use to create a library management system:

Publishers Table:
- publisher_id (Primary Key)
- publisher_name
- publisher_country

Book Copies Table:
- copy_id (Primary Key)
- book_id (Foreign Key referencing Books table)
- copy_number
- condition
- shelf_location

Authors-Books Mapping Table:
- author_book_id (Primary Key)
- author_id (Foreign Key referencing Authors table)
- book_id (Foreign Key referencing Books table)

Reviews Table:
- review_id (Primary Key)
- book_id (Foreign Key referencing Books table)
- member_id (Foreign Key referencing Members table)
- rating
- review_text
- review_date

Transactions Table:
- transaction_id (Primary Key)
- member_id (Foreign Key referencing Members table)
- transaction_date
- transaction_type (e.g., borrow, return, purchase)
- amount_paid

Here's the expanded dataset:

Publishers Table:

publisher_id	publisher_name	publisher_country

1	Penguin Random House	United States
2	HarperCollins	United Kingdom
...	...	...

Book Copies Table:

copy_id	book_id	copy_number	condition	shelf_location
1	1	001	Good	A1
2	1	002	Fair	B3
...	...	...	...	...

Authors-Books Mapping Table:

author_book_id	author_id	book_id
1	1	1
2	2	2
...	...	...

Reviews Table:

review_id	book_id	member_id	rating	review_text	review_date

1	1	1	4.5	"A classic masterpiece."	2024-02-12
2	2	2	5.0	"Absolutely loved it!"	2024-02-18
...	...	...	...	...	...

Transactions Table:

transaction_id	member_id	transaction_date	transaction_type	amount_paid
1	1	2024-02-10	Borrow	0
2	2	2024-02-15	Borrow	0
...	...	...	...	...

After creating these tables perform the following queries

1. List all books borrowed by a specific member:
2. Find the most popular genres:
3. Identify books with the highest average rating:
4. List all members who have borrowed more than 5 books:
5. List all members who have borrowed less than 5 books:
6. Retrieve the top-rated books with at least 5 reviews:
7. Calculate the total revenue generated from book purchases:
8. List all books with their respective authors and publishers:
9. Find books that are currently available for borrowing:
10. Identify members who have overdue books:
11. List the top 10 most borrowed books:
12. Calculate the average number of days a book is borrowed for:
13. Find the total number of books published in each year:
14. Identify members who have borrowed books more than once:

15. List all books with their respective authors and average ratings:
16. Calculate the total number of copies available for each book:
17. Create a view of transaction table and provide privilege to another user. The user can view only member id and transaction date and privilege should be to select id who made transaction on any specific date.

Some website to practice sql

1. https://sqlzoo.net/wiki/SQL_Tutorial
2. https://sqlbolt.com/
3. https://www.sql-practice.com/
4. https://www.kaggle.com/learn/intro-to-sql

Deadline to submit the project : 8th of March