Q5,Q13,Q14.) Types of database languages

A database is defined as a collection of related data.
There are mainly 4 different database languages present :

- Data Definition Language
- Data Manipulation Language
- Data Control Language
- Transaction Control Language

Data Definition Language

Data Definition Language is used to define and manage the structure of the database schema.

- CREATE: Used to create database objects like tables, views, or indexes.
- ALTER: Used to modify the structure of existing database objects.
- DROP: Used to delete database objects.
- TRUNCATE: Used to remove all data from a table while retaining the table structure.

Data Manipulation Language

Data Manipulation Language is used to interact with the data stored in the database. It allows users to insert, retrieve, update, and delete data from the database tables. DML commands are used to perform CRUD operations (Create, Read, Update, Delete) on the data.

- SELECT: Used to retrieve data from one or more tables.
- INSERT: Used to add new records to a table.
- UPDATE: Used to modify existing records in a table.
- DELETE: Used to remove records from a table.

Data Control Language

Data Control Language is used to manage access rights and permissions on the database objects. It ensures that only authorized users can access and manipulate the data.

- GRANT: Used to give specific privileges to users or user roles.
- REVOKE: Used to remove privileges from users or user roles.

Transaction Control Language

Transaction Control Language is used to manage transactions in the database. **Transactions are sequences of one or more database operations that are executed as a single unit of work.** TCL commands help in controlling the **ACID properties** (Atomicity, Consistency, Isolation, Durability) of transactions.

- COMMIT: Used to save changes made by a transaction to the database.
- ROLLBACK: Used to undo changes made by a transaction and restore the database to its previous state.
- SAVEPOINT: Used to define a point within a transaction to which you can later roll back.

Q6.) Acid Properties of Transactions

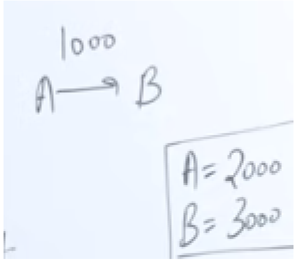
Transactions are sequences of one or more database operations that are executed as a single unit of work.

Acid stands for Atomicity, Consistency, Isolation, Durability.

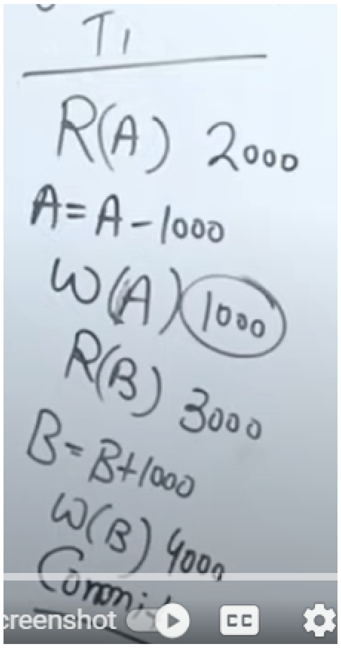
Atomicity : It means that either all the operations within a transaction are executed successfully, or none of them are. If any part of the transaction fails, the entire transaction is rolled back, and the database returns to its original state. This ensures that the database remains consistent, and partial updates are avoided.

Consistency : Consistency ensures that a transaction takes the database from one consistent state to another.

For example, let's assume that the database has information about the amount of money present in 2 bank accounts (i.e A and B). Initially, in account A there is 2000 rupees and in account B there is 3000.
Now, during the transaction, A deposits 1000 rupees to B. Once the transaction is completed, account A will be 1000 and B will have 4000 rupees.
Now, the transaction is said to make the database go from one consistent state to another consistent state since the sum of money before the transaction started was 2000 + 3000 = 5000, and after the transaction is done is again 1000 + 4000 = 5000.



Initial State


T1

R(A) 2000

A=A-1000

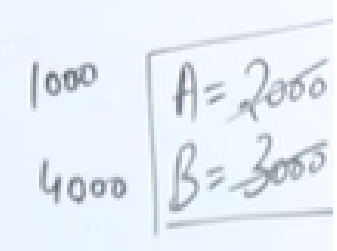
W(A) 1000

R(B) 3000

B=B+1000

W(B) 4000

Commi


1000 A=~~2000~~

4000 B=~~3000~~

Final State

Isolation : Isolation means that each transaction is executing independently of the other transactions. Each transaction operates in isolation and is unaware of other transactions running concurrently.

Durability : Durability guarantees that once a transaction is committed successfully, its changes are permanent and will survive any subsequent failures, including power outages or crashes. Committed data is stored in non-volatile storage, ensuring that it remains available even after the system restarts.

Q7.) Difference between Vertical & Horizontal Scaling

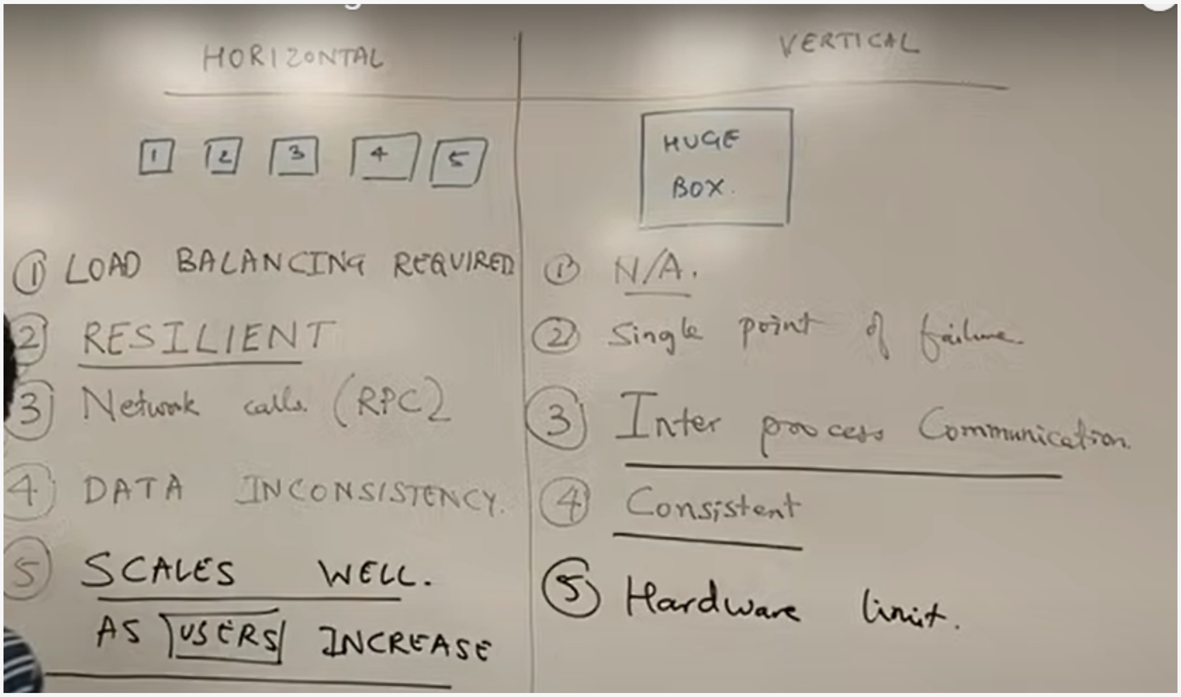
Vertical scaling and horizontal scaling are two different approaches used to increase the capacity and performance of a system, especially in the context of databases and server architectures.

Vertical Scaling :

- Also known as Scaling Up
- involves increasing the capacity of a single server by adding more power (CPU), memory (RAM), or storage to the existing hardware.
- Vertical scaling is typically done on a single machine or server and requires downtime or at least a temporary interruption in service during the upgrade process.
- Advantages :
    - Simpler to implement as it involves upgrading existing hardware.
    - Easier to manage and maintain as it involves a single machine.
- Disadvantages :
    - A single point of failure: If the upgraded server fails, the entire system may be affected.
    - Eventually, there are hardware limitations, and a point may be reached where further upgrades become impractical or prohibitively expensive.

<u>Horizontal Scaling :</u>

- Also known as Scaling Out
- involves adding more machines or servers to distribute the workload across multiple systems.
- Instead of upgrading a single machine, new servers are added to the existing infrastructure to handle increased traffic and data processing.
- often utilizes load balancing techniques to distribute incoming requests across the multiple servers.



HORIZONTAL	VERTICAL
1 2 3 4 5	HUGE BOX.
1 LOAD BALANCING REQUIRED	1 N/A.
2 RESILIENT	2 Single point of failure
3 Network calls (RPC)	3 Inter process Communication
4 DATA INCONSISTENCY.	4 Consistent
5 SCALES WELL. AS USERS INCREASE	5 Hardware limit.

Q8.) What is Sharding?

Sharding is a database architecture technique used to horizontally partition large databases into smaller, more manageable pieces called "shards." Each shard is an independent database that contains a subset of the overall data. Sharding is primarily used to improve scalability and performance for large-scale applications with high data volumes and heavy workloads.

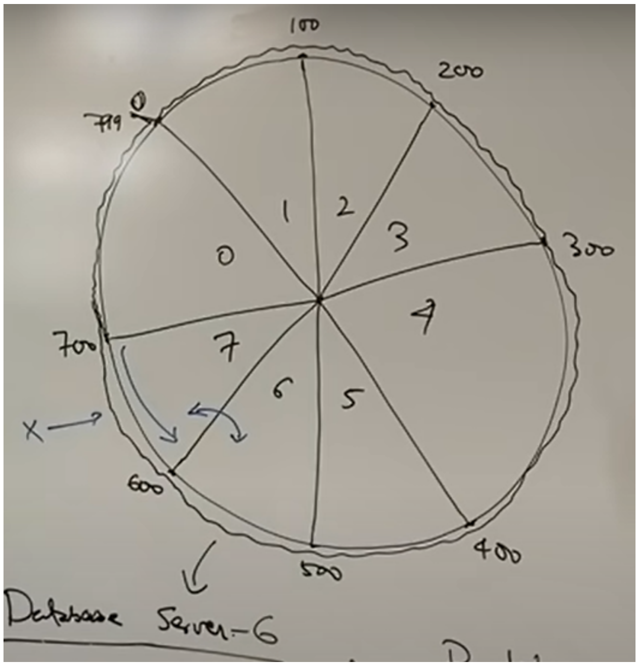
- Data is distributed across multiple database instances or servers, referred to as shards, based on a predefined sharding key or strategy.
- Each shard operates independently, handling a subset of the data and specific queries related to that data.
- Shards are typically located on separate physical machines or server clusters, allowing for parallel processing and improved performance.
- The sharding strategy can be based on various factors, such as user location, time range, or a hash of the data, to ensure an even distribution of data across shards.
- Load balancing mechanisms and intelligent routing are often employed to direct queries to the appropriate shard, ensuring that data retrieval is efficient and optimized.

Advantages :

- Improved Performance: With data partitioned into smaller, manageable chunks, queries can be processed in parallel, resulting in faster response times.
- High Availability: Sharding provides fault tolerance, as each shard operates independently. If one shard fails, the rest of the system can continue to function normally.
- Cost Efficiency: Sharding allows the use of commodity hardware for each shard, which can be more cost-effective than investing in a single, extremely powerful server.

Disadvantages :

- Joins and Transactions: Handling complex joins and transactions that involve data from multiple shards can be more challenging and may require special techniques or tools.
- Data Distribution: Selecting an appropriate sharding key and strategy is crucial to avoid data imbalance and hotspots that may affect system performance.



Q9.) Keys in DBMS?

Primary Key :

- Uniquely identifies tuples within the table/relation
- Every table should have a primary key to enforce data integrity and provide a reliable way to access individual records
- The primary key cannot have NULL values

Super Key :

- A super key is a set of one or more columns that can uniquely identify a record in a table.
- It may contain more columns than the minimum required for uniqueness, making it a broader concept than primary keys.

For example, if you have the relation R(A1,A2,..... , An) and A1 is the candidate key, then the number of super keys possible are 2^n-1.(i.e the super key has to contain at least one candidate key).

Candidate Key :

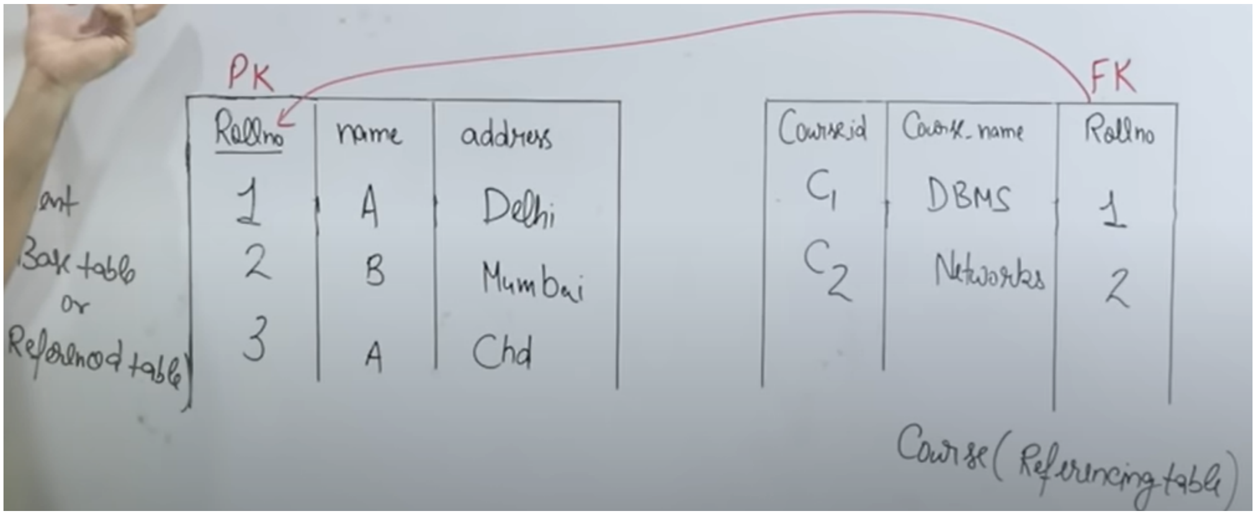
- A candidate key is a minimal super key, meaning it is the smallest set of columns that can uniquely identify a record in a table.
- From all the candidate keys, one is chosen to be the primary key.

Composite Key :

- A composite key is a key that consists of two or more columns in a table, used together to uniquely identify a record.
- This type of key is useful when no single column can guarantee uniqueness, but the combination of multiple columns does.

Foreign Key :

- A foreign key is a field in one table that refers to the primary key in another table, establishing a relationship between the two tables.
- It is used to maintain referential integrity, ensuring that data in the related tables remains consistent.
- The table containing the Foreign Key is known as the Referencing Table, and the table whose primary key is being referenced is the Referenced Table.



Rollno	name	address
1	A	Delhi
2	B	Mumbai
3	A	Chd

Courseid	Course_name	Rollno
C1	DBMS	1
C2	Networks	2

Referenced Table :

1) Insertion : No violation
2) Deletion : May cause a violation. If an entry in the referenced table is deleted and there is an entry in the referencing table which was referencing that particular entry which was deleted.
   Can use :
   - On delete cascade (i.e when a particular entry is deleted in the referenced table, all the entries which are referencing that entry will also get deleted)
   - On delete set null (i.e when a particular entry is deleted from the referenced table, all the entries which are referencing that entry will have a null for the foreign key part.)
   - On delete no action
3) Update : May cause violation.
   Can use :

- On update cascade
- On update set null
- On update no action

Referencing Table :

1) Insertion : May cause violation. If you add an entry with a foreign key value which doesn't even exist in the primary key attribute.
2) Deletion : No violation
3) Update : May cause violation. If you update the value of the foreign key value to something that doesn't exist in the primary key attribute.

Q10.) Types of Relationships?

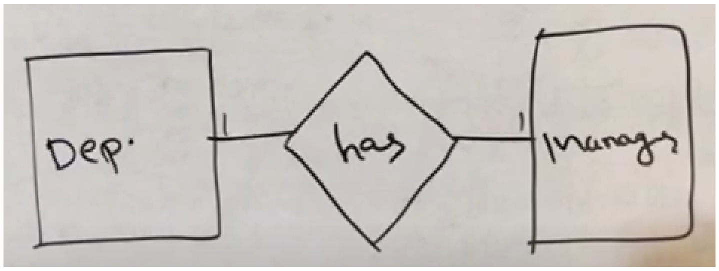
There are mainly 4 different types of relationships :

- One - to - one (1:1)
- One - to - many(1:N)
- Many - to - one(N:1)
- Many - to - many(M:N)

One - to - one :

In a one-to-one relationship, each record in one table is associated with only one record in another table, and vice versa.

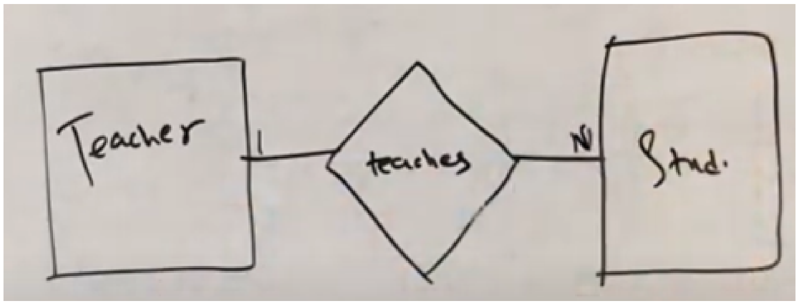
Example :



One - to - many :

- In a one-to-many relationship, a single record in one table can be associated with multiple records in another table, but each record in the second table is related to only one record in the first table.
- For example, in a typical scenario, a customer can have multiple orders, but each order is associated with only one customer.

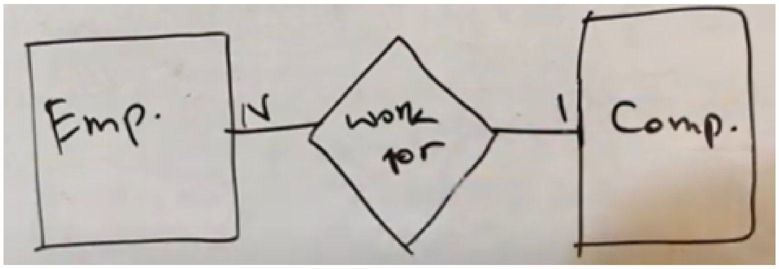
Example :



Many - to - one :

- In a many-to-one relationship, multiple records in one table can be associated with a single record in another table, but each record in the second table is related to many records in the first table.

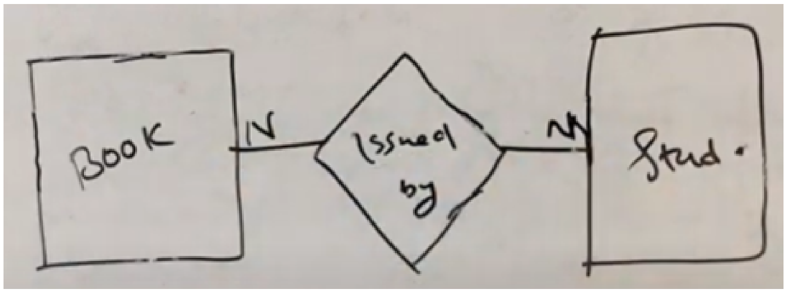
Example :



Many - to - many :

- In a many-to-many relationship, multiple records in one table can be associated with multiple records in another table, and vice versa.

Example :



Recursive Relationship :

- The same entity type will participate more than once in the relationship.
- Example : Employee(i.e boss) supervises Employee(i.e subordinate).

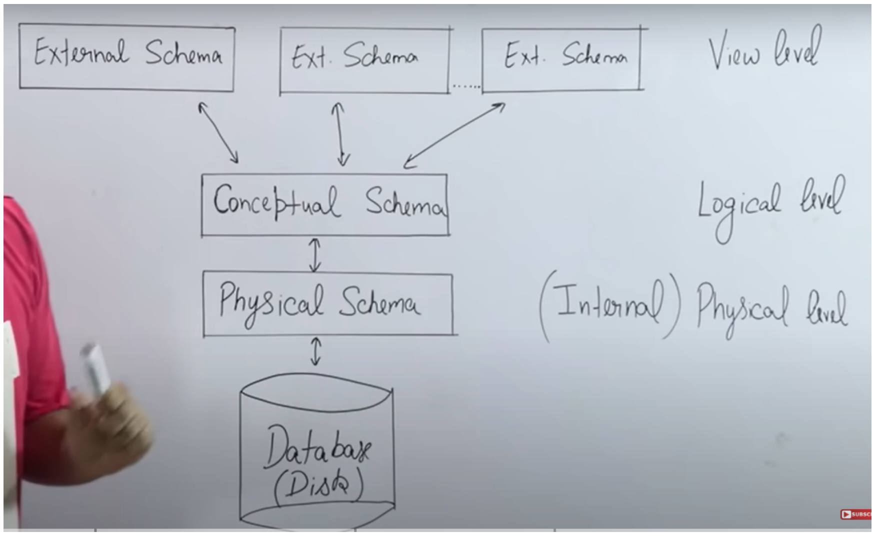
Q11.) Data Abstraction in DBMS (3 levels of it) ?

Data abstraction in DBMS refers to the process of hiding the implementation details of the database from the users and providing them with a simplified and conceptual view of the data. It allows users to interact with the database without needing to understand the complexities of how the data is stored and managed internally. The three levels of data abstraction in DBMS are:

1) **Physical/Internal Level (i.e Physical Schema) :**
    - This is the lowest level of the data abstraction, which focuses on how the data is actually being stored on the hardware & storage devices.
    - It also tells about how the DBMS is storing, accessing, and indexing the data.
    - It focuses on how to optimize data storage & retrieval so that we can obtain better performance.
    - DATABASE ADMINISTRATORS work at the physical level.
2) **Logical Level (i.e Conceptual Schema) :**
    - This is the middle level of data abstraction, which tells about the structure of the databases(i..e schema). It could involve things like what attributes need to be present, data types of attributes, constraints, etc.
    - An E-R diagram for example can be used to show the conceptual schema.
    - The logical schema provides a high-level view of the data and is independent of the specific database management system being used.
    - DATABASE DESIGNERS work at the logical level.
3) **View Level (i.e External Schema) :**
    - The view level is the highest level of data abstraction and represents the user's view of the data.
    - Different user groups will have different views of the data, allowing each group to access and manipulate only the relevant data. For example : A student will have a different view compared to a teacher who signed into an attendance management system. The student will only be able to login and see his attendance, whereas a teacher is able to edit the details of the students, and much more.

Logical Data Independence : Changes can be made to the conceptual schema without needing to change the external schema.

Physical Data Independence : Changes can be made to the physical schema without needing to change the conceptual schema.



Q15,16,20.) What is Normalization? Types of them.

Normalization is defined as the process of decomposing unsatisfactory “bad” relations by breaking up its attributes into smaller relations.

Denormalization is defined as the process of joining the relations of higher normal form, to form a base relation of lower normal form.

REFER TO NOTES

Q17.) What are functional dependencies?

Functional dependencies help denote the relationship b/w attributes of a relation. There are mainly 2 types of functional dependencies :

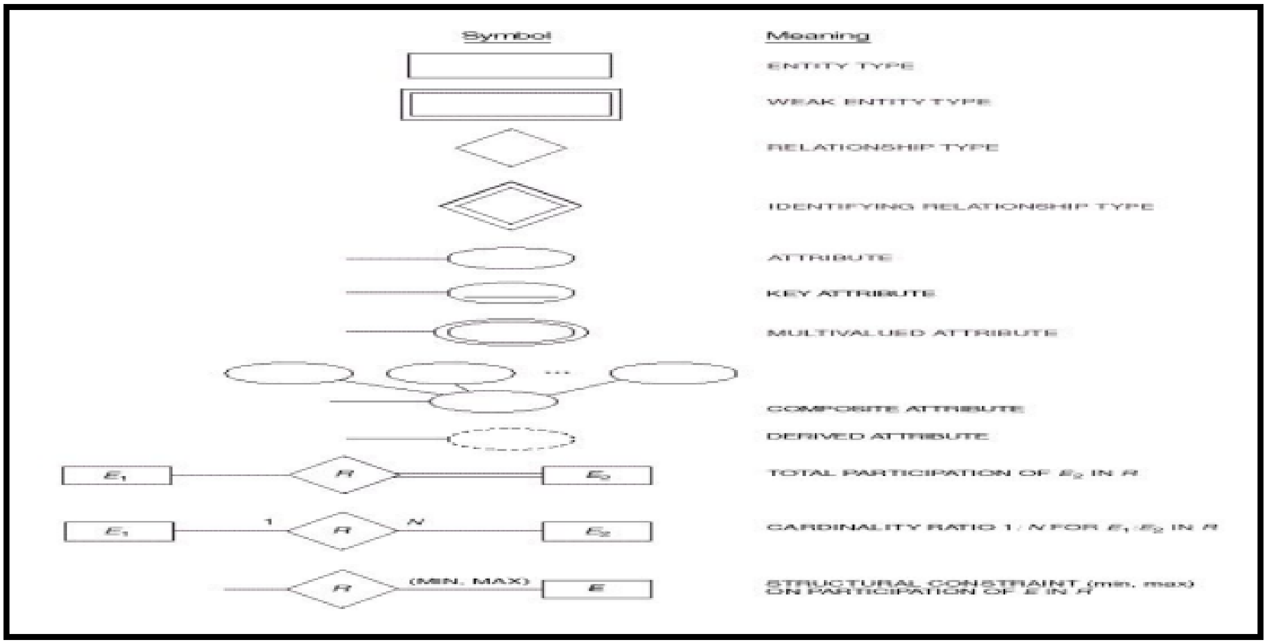
- Trivial functional dependencies : X->Y and Y is a subset of X, i.e X intersection Y != NULL
- Non-trivial functional dependencies : X->Y and X intersection Y == NULL.

There are many properties of functional dependencies, including Reflexivity, Transitivity, Augmentation, Union, Decomposition, Pseudotransitivity, etc.

REFER TO NOTES

Q18,22. E-R Model?

An Entity-Relationship (ER) model is a conceptual data modeling technique used to represent the logical structure of a database in a clear and understandable way. It helps to visualize and describe the various entities (objects) in a system and the relationships between them.



Entity : An entity represents a real-world object, concept, or thing in the system that we want to store information about.

Entity Type : Entities with the same basic attributes are grouped or typed into an entity type. Within the ER diagram, every rectangle represents an Entity Type. (i.e also known as intention)

Entity Set : For every entity type, there will exist a collection of entities stored in the database. This collection is referred to as the entity set. (i.e also known as the extension/state of the database)

Attribute : An attribute is a property or characteristic of an entity that describes the data that can be stored about that entity.

- For example, in a "Student" entity, attributes may include "Student ID," "Name," "Age," and "Address."

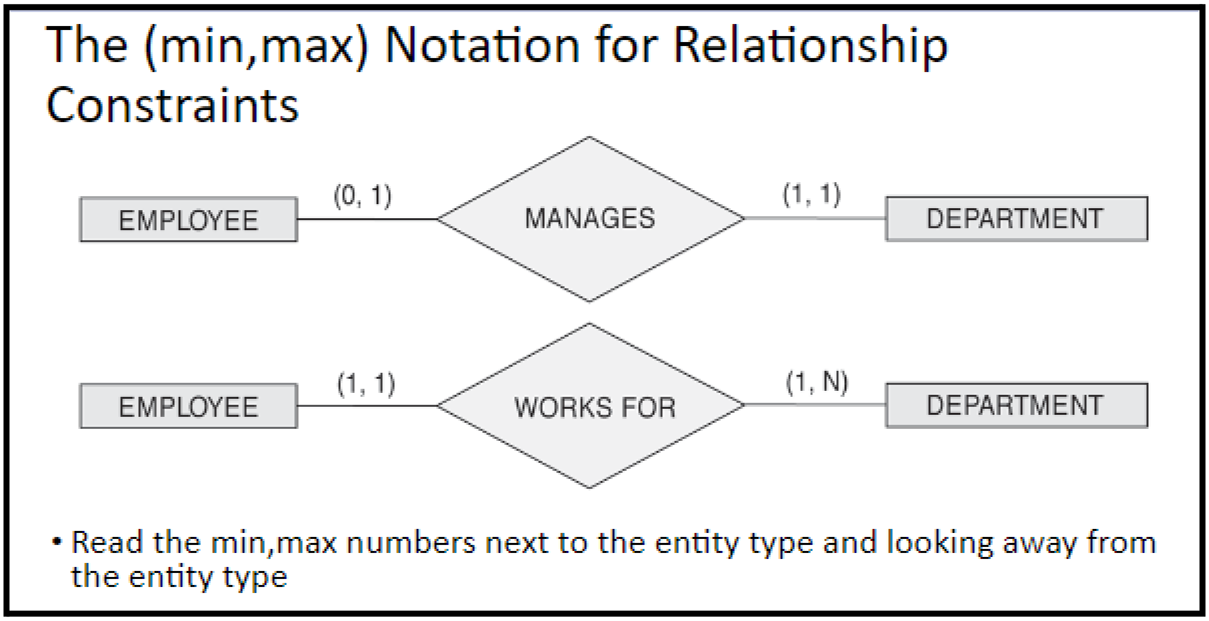
Key Attribute : An attribute of the entity which uniquely identifies every tuple within the relation.

Relationship : A relationship represents an association or connection between two or more entities.

Strong Entity : An entity which has an attribute which can act as a primary key, and does not depend on other entities for its existence.

Weak Entity : An entity that cannot be uniquely identified by its attributes alone and depends on the existence of a related entity (called a strong entity) for its identification.

By using the ER - Model, we'll get a clear idea as to which tables are required, what are the attributes/columns of each table, how the records of one table are related to the records of the other tables, and so on.



The (min,max) Notation for Relationship Constraints

- Read the min,max numbers next to the entity type and looking away from the entity type

Q28.) Difference between 2-tier & 3-tier architecture

In 2-tier architecture, there are basically 2 layers. One layer consists of the client machines, and the other layer has the database server.
In 2-tier architecture, the client machine will send a request(i.e through an interface) to fetch some data directly to the database server. The database server will process the request, and then send the required data back to the user.
In 2-tier architecture, **we only consider the authorized client machines.**
For example, if we want to book a railway ticket, we can physically go to the railway station, fill out a form containing all our details, and give it to the person behind the screen. Now the person sitting behind is authorized, so what he'll do is that he'll fill up all our details into his machine, and then will make a request to the database server to fetch the train details.

In 3-tier architecture, there are basically 3 layers. We have the client machines on top, then a business layer, after which the database server is present.
In 3-tier architecture, the clients refer to end-user machines.
In 3 -tier architecture, the clients do not directly interact with the database server. The client machine sends a request for some data to the business layer. The business layer will process this request, and will then send only a database request to the database server. The database server can now directly process the database request, and will send the output back to the business layer, which in turn helps in displaying the results on the interface of the client machine.

Differences b/w 2 -tier & 3-tier architecture:

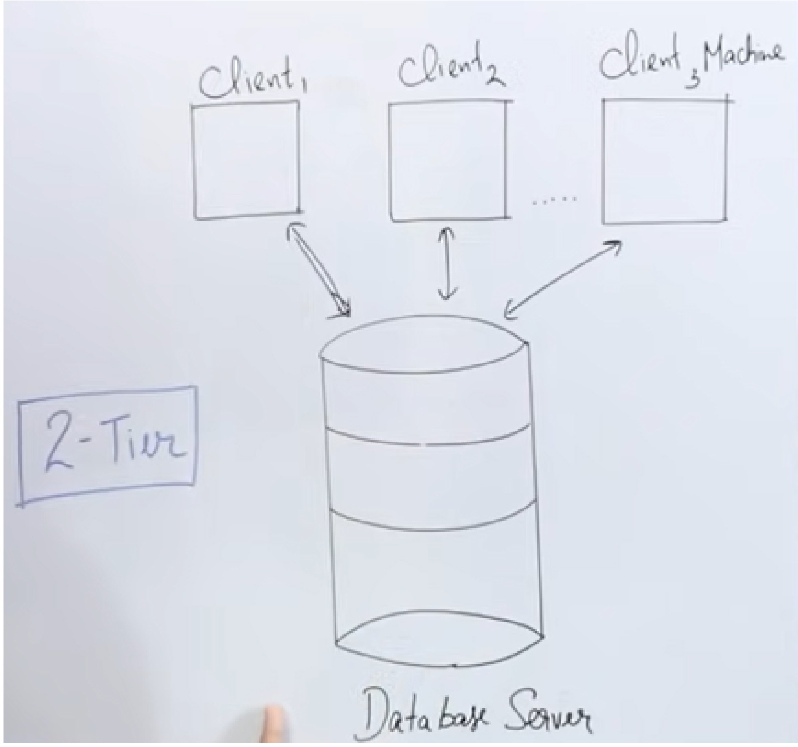
2-tier :

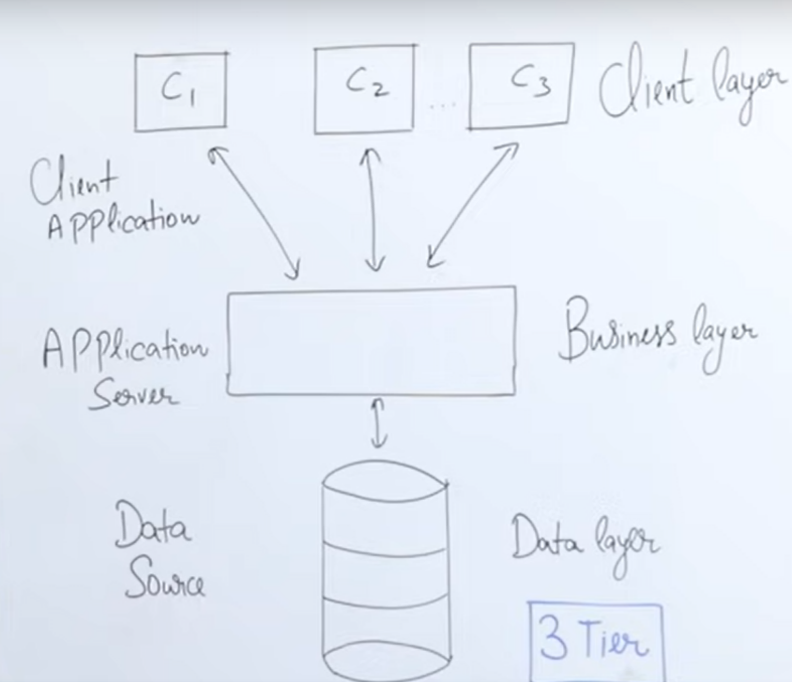
- **Maintenance :** In 2-tier, the clients are authorized. This means that there will be a limited number of clients which are making a request to the database server, making the maintenance process easier.
- **Scalability :** Since there are a limited number of clients, it is tough to scale.
- **Security :** Client machines are directly communicating with the database server, which can cause some security issues.
- **Load on database server :** The database server has to perform 2 functions : Process the request sent by the client + Retrieve the data from the database

2-tier :

- **Maintenance :** In 3-tier architecture, the clients are the end-users. This means that there isn't actually a limit on the number of clients that can make a request to the database server, making the maintenance process harder .
- **Scalability :** Since there is no limit on the number of client machines, it is easy to scale.
- **Security :** 3-tier architecture provides more security compared to 2-tier architecture since the client machines cannot directly communicate with the database server.
- **Load on the database server :** The database server only has to perform 1 function : Retrieve the data from the database (i.e since the business layer would have processed the request already.) Hence the load on the database server is lesser.

Nowadays, most of the applications work on 3-tier architecture itself.





Q29.) Difference between Truncate & Delete

Truncate :

- The truncate is a DDL operation, which is used to delete all the data present in the relation.

- When you use the TRUNCATE command, it deletes all rows from the table but keeps the table structure, including columns, indexes, and constraints intact.
- TRUNCATE is a non-logged operation, which means it does not generate individual entries in the database transaction log for each row deleted. This makes it faster than the DELETE command, especially for large tables.

Delete :

- DELETE is a Data Manipulation Language (DML) command used to remove specific rows from a table based on a condition.
- Unlike TRUNCATE, DELETE is a logged operation, which means each row deletion is logged in the transaction log. This can lead to increased overhead and slower performance, especially for large tables with many rows to delete.

Q30.) Intension vs Extension of Database?

Intension :

- Refers to the structure of the database itself. It’s the schema of the database.
- Includes things like the names of the various tables within the database, the attributes present in each table, the data types of those attributes, etc.
- The intension of a database does not actually change with time.

Extension :

- It refers to the state of the database at a particular moment of time.
- It refers to the set of tuples that are present in the various relations of the database at time ‘t’.
- The extension of a database changes with time. As tuples are added, deleted, and updated, the extension of the database changes.