1) Describe the concept of Centralized vs. Client/server Model. 7M

DBMS - Centralized and Client-Server Architecture

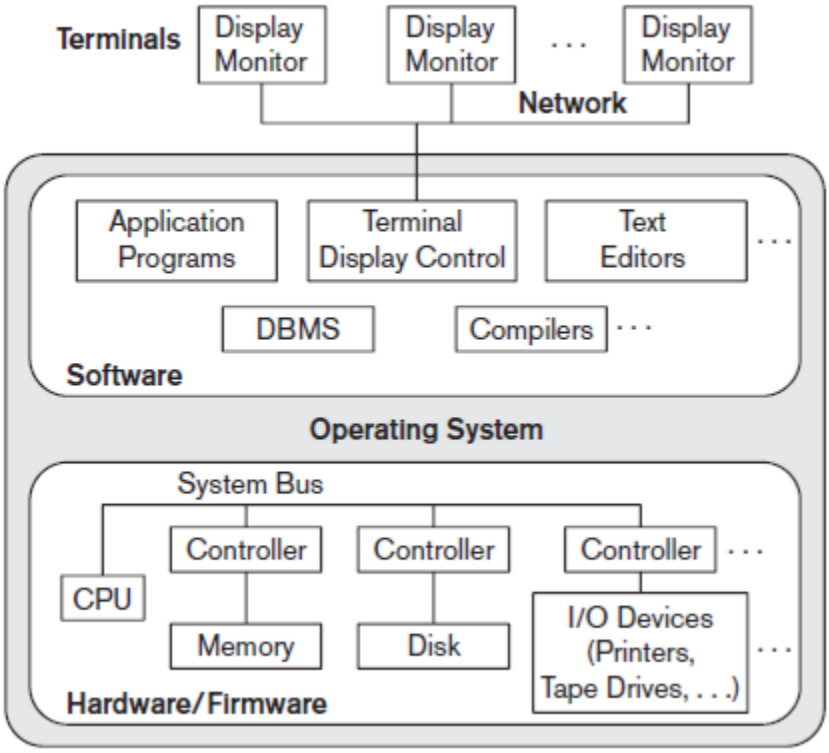
The architecture of a DBMS determines how the data is stored, processed, and accessed by users.

we will explore centralized and client-server DBMS architectures, along with their key features and examples.

Centralized DBMS Architecture

A centralized system operates through a central node. In a centralized DBMS architecture, all database operations, user interfaces, and applications are managed by a single central computer typically a **mainframe** or a powerful server.

This type of architecture was common in the early days of computing, when most processing power resided in a central location. Users would access the mainframe through terminals connected via a network. These terminals had minimal processing capabilities and served primarily as input/output devices.



Working of Centralized DBMS Architecture

All user interactions in a centralized DBMS take place through the terminals. Terminals are some basic interfaces connected to the mainframe. These terminals do not process the data themselves.

They send the input commands to the server. The command handles the execution. The server processes the commands and returns the results to the terminals for display.

Let's understand this with an **example**. Consider a university that uses a centralized DBMS to manage its student database. All student-related queries such as checking grades or registering for classes are processed on a central mainframe. Terminals located across the campus send requests to this central server, which processes the queries, retrieves the required data, and sends the results back to the terminals for display.

The following table highlights the benefits and drawbacks of using a centralized DBMS architecture –

Benefits	**Drawbacks**
Centralized Control: The mainframe has full control over data access, making security management more straightforward.	Scalability Issues: As the number of users grows, the mainframe may struggle to handle the increased load.
Simplified Maintenance: All software updates and changes are applied to a single system. It reduces administrative overhead.	Single Point of Failure: If the mainframe fails, the entire database system becomes unavailable.

Client-Server DBMS Architecture

The client-server system is evolvement of centralized system. This architecture divides the workload between clients (user-facing systems) and servers (back-end systems). The clients handle the user interface and local processing. It is the servers manage data storage, complex processing, and business logic.

Basic Structure of Client-Server Architecture

In its simplest form, the client-server model has the following objects:

- **Clients**: Machines or software applications where the users interact. These handle user inputs and present results.
- **Servers**: Systems that store the database and execute data processing tasks.

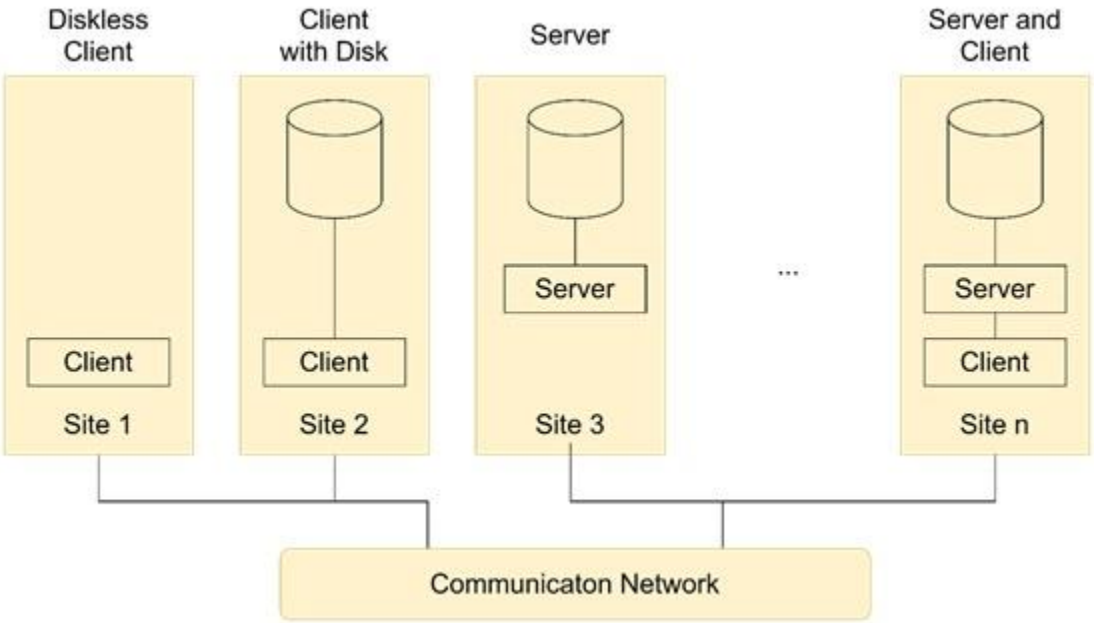
For example, a customer service platform where employees use desktop computers (clients) to access a centralized customer database (server). The client computers send queries to the server, which processes them and returns the necessary information.

Distributing the workload between client and server helps balance resource use and reduces the load on a single system. Clients and servers can be located on different machines, facilitating remote access and distributed data processing.

Two-Tier Client-Server Architecture

In a two-tier client-server architecture, the client handles the user interface and application logic. On the other hand the server focuses on data storage and retrieval. The connection between the client and the server allows clients to submit requests (queries). The queries uses the server processes and returns as results.

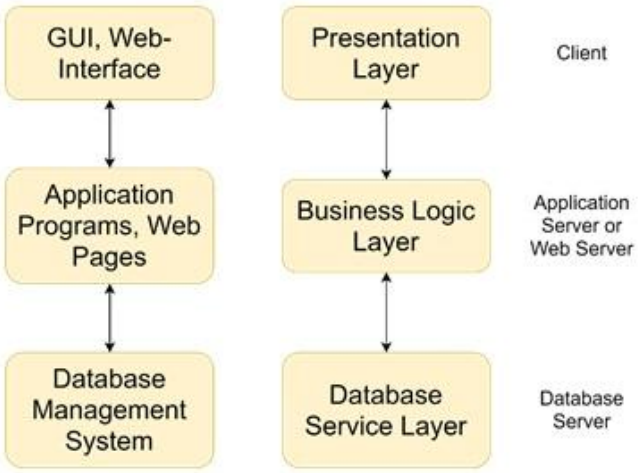
For example, a banking application where tellers use client-side software to check account balances. The client software connects to a centralized database server, retrieves data, and displays it on the teller's screen.



The following table highlights the benefits and drawbacks of client-server DBMS architecture

Benefits	**Drawbacks**
Simplicity − The structure is simple and straightforward and easy to implement.	**Scalability Limits** − As more clients connect, the server may face performance bottlenecks.
Direct Interaction − Clients interact directly with the server, resulting in faster query processing for simple tasks.	**Maintenance** − Each client may require updates if changes are made to the application.

The speciality of three-tier architecture is that, it has an additional layer between the client and the server. This is known as application server. This intermediate layer helps manage business logic, application rules, and data processing more effectively.



Following are the **components** of three-tier client-server DBMS architecture −

- **Presentation Layer (Client)** − Displays data and collects user input.
- **Application Layer (Middle Tier)** − Processes user requests and interacts with the database.
- **Data Layer (Server)** − Handles database storage and management.

Example − A web-based ordering system for a restaurant. The client (user's web browser) interacts with a web server (application layer), which processes orders and retrieves data from the database server (data layer). This structure allows for better load distribution and supports complex business logic.

Following are the **advantages of using three-tier client-server DBMS architecture** −

- **Better Load Management** − The middle tier processes requests before sending them to the database server, reducing the server's direct load.
- **Enhanced Security** − The application server acts as a gatekeeper, validating user requests and providing controlled database access.

Beyond Three Tiers: n-Tier DBMS Architectures

Three-tier systems are common, some applications extend it to n-tier architectures. Such architectures can include additional processing layers for more specialized tasks such as separate layers for authentication, data aggregation, or specific application services.

Example − Large enterprise applications, like those used in CRM or ERP, often utilize n-tier structures to balance tasks across various layers.

3) Explain the advantages of DBMS. 5M

Ans:

Database Management System (DBMS) is a collection of interrelated data and a set of software tools/programs that access, process, and manipulate data.

It allows access, retrieval, and use of that data by considering appropriate security measures and is useful for better data integration and security.

The advantages of database management systems are:

1. **Data Security:** DBMS enhances data security through access control and encryption. It enforces privacy and prevents unauthorized access. When the number of users increases, the chances of attacks can grow. But big systems usually add stronger security as they grow, which helps reduce the risk of breaches.
2. **Data integration:** DBMS unifies data from different sources into a centralized system. This provides a coherent organizational view of operations. It helps to keep track of how one segment of the company affects another segment.
3. **Data abstraction:** Since many complex algorithms are used by the developers to increase the efficiency of databases that are being hidden by the users through various data abstraction levels to allow users to easily interact with the system.
4. **Reduction in data Redundancy:** DBMS avoids data duplication by enforcing unique constraints. It removes unnecessary repetitive entries in databases. This ensures efficient use of storage and improves consistency. for e.g. - If there are two same students in different rows, then one of the duplicate data will be deleted.
5. **Data sharing:** A DBMS provides a platform for sharing data across multiple applications and users, which can increase productivity and collaboration.
6. **Data consistency and accuracy:** DBMS enforces integrity constraints to maintain valid data. It minimizes discrepancies by syncing updates across all views. This reduces errors and ensures reliable & consistent data.
7. **Data organization:** A DBMS provides a systematic approach to organizing data in a structured way, which makes it easier to retrieve and manage data efficiently.
8. **Efficient data access and retrieval:** DBMS allows for efficient data access and retrieval by providing indexing and query optimization techniques which reduces time taken to retrieve large datasets. It boosts system performance and user satisfaction.
9. **Concurrency and maintained Atomicity**: That means, if some operation is performed on one particular table of the database, then the change must be reflected for the entire

database. The DBMS allows concurrent access to multiple users by using the synchronization technique.

10. **Scalability and flexibility:** DBMS is highly scalable and can easily accommodate changes in data volumes and user requirements. It allows flexible schema modifications and expansion.

4) Discuss about different types of Keys. 5M

Ans:

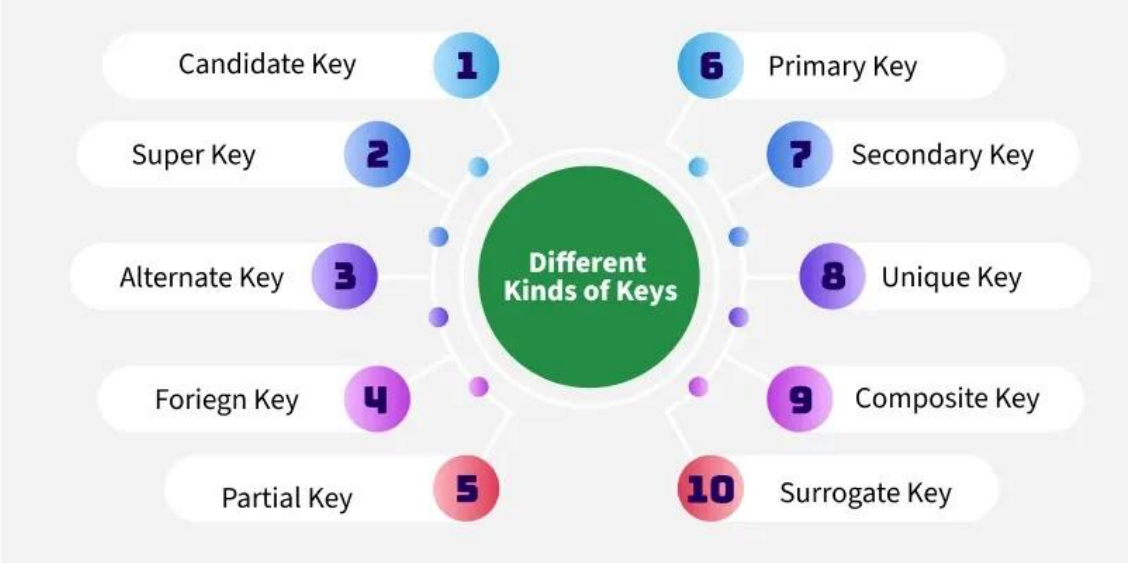
Keys in Relational Model

In the context of a relational database, keys are one of the basic requirements of a relational database model.

- Keys are fundamental components that ensure data integrity, uniqueness and efficient access. It is widely used to identify the tuples(rows) uniquely in the table.
- We also use keys to set up relations amongst various columns and tables of a relational database.

Keys are important in a Database Management System (DBMS) for several reasons:

- Uniqueness: Keys ensure that each record in a table is unique and can be identified distinctly.
- Data Integrity: Keys prevent data duplication and maintain the consistency of the data.
- Efficient Data Retrieval: By defining relationships between tables, keys enable faster querying and better data organization. Without keys, it would be extremely difficult to manage large datasets and queries would become inefficient and prone to errors.



Keys in DBMS

Types of Keys

1. Super Key

The set of one or more attributes (columns) that can uniquely identify a tuple (record) is known as Super Key. It may include extra attributes that aren't important for uniqueness but still uniquely identify the row.

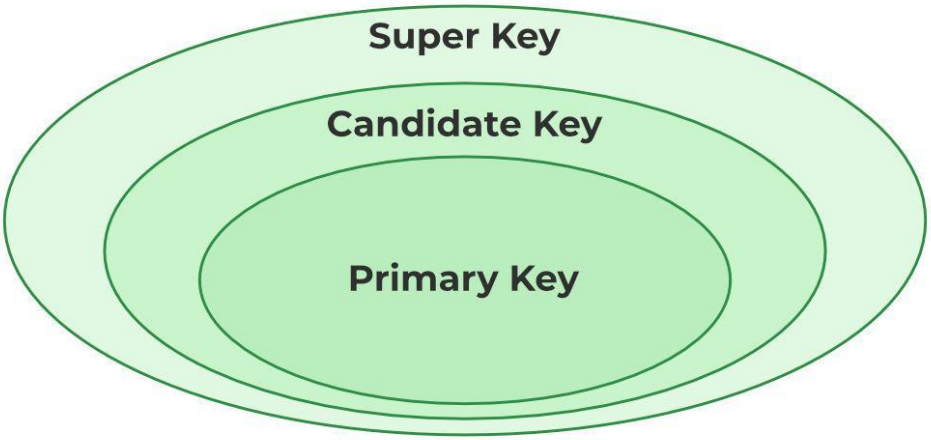
For Example, STUD_NO, (STUD_NO, STUD_NAME), etc.

- A super key is a group of single or multiple keys that uniquely identifies rows in a table. It supports NULL values in rows.
- A super key can contain extra attributes that aren’t necessary for uniqueness.
- For example, if the "STUD_NO" column can uniquely identify a student, adding "SNAME" to it will still form a valid super key, though it's unnecessary.

Example: Consider the STUDENT table

STUD_NO	SNAME	ADDRESS	PHONE
1	Shyam	Delhi	123456789
2	Rakesh	Kolkata	223365796
3	Suraj	Delhi	175468965

A super key could be a combination of STUD_NO and PHONE, as this combination uniquely identifies a student.



Relation between Primary Key, Candidate Key, and Super Key

2. Candidate Key

The minimal set of attributes that can uniquely identify a tuple is known as a candidate key. For Example, STUD_NO in STUDENT relation.

- A candidate key is a minimal super key, meaning it can uniquely identify a record but contains no extra attributes.
- It is a super key with no repeated data is called a candidate key.
- The minimal set of attributes that can uniquely identify a record.
- A candidate key must contain unique values, ensuring that no two rows have the same value in the candidate key's columns.
- Every table must have at least a single candidate key.
- A table can have multiple candidate keys but only one primary key.

Example: For the STUDENT table below, STUD_NO can be a candidate key, as it uniquely identifies each record.

STUD_NO	SNAME	ADDRESS	PHONE
1	Shyam	Delhi	123456789
2	Rakesh	Kolkata	223365796
3	Suraj	Delhi	175468965

Table: STUDENT_COURSE

STUD_NO	TEACHER_NO	COURSE_NO
1	001	C001
2	056	C005

A composite candidate key example: {STUD_NO, COURSE_NO} can be a candidate key for a STUDENT_COURSE table.

3. Primary Key

There can be more than one candidate key in relation out of which one can be chosen as the primary key.

For Example, STUD_NO, as well as STUD_PHONE, are candidate keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).

- A primary key is a unique key, meaning it can uniquely identify each record (tuple) in a table.
- It must have unique values and cannot contain any duplicate values.
- A primary key cannot be NULL, as it needs to provide a valid, unique identifier for every record.
- A primary key does not have to consist of a single column. In some cases, a composite primary key (made of multiple columns) can be used to uniquely identify records in a table.
- Databases typically store rows ordered in memory according to primary key for fast access of records using primary key.

Example:

STUDENT table -> Student(STUD_NO, SNAME, ADDRESS, PHONE) , STUD_NO is a primary key

Table: STUDENT

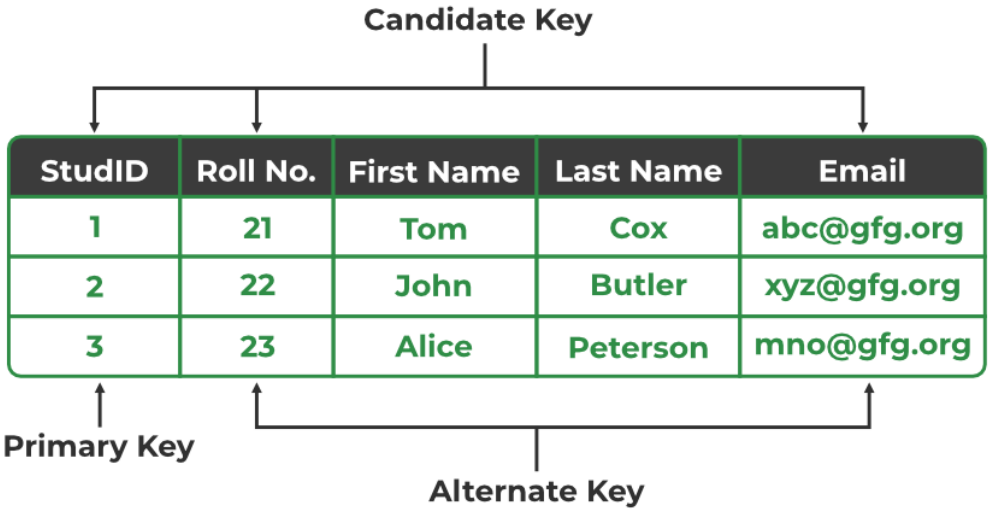
STUD_NO	SNAME	ADDRESS	PHONE
1	Shyam	Delhi	123456789
2	Rakesh	Kolkata	223365796
3	Suraj	Delhi	175468965

4. Alternate Key

An alternate key is any candidate key in a table that is not chosen as the primary key. In other words, all the keys that are not selected as the primary key are considered alternate keys.

- An alternate key is also referred to as a secondary key because it can uniquely identify records in a table, just like the primary key.
- An alternate key can consist of one or more columns (fields) that can uniquely identify a record, but it is not the primary key

Example: In the STUDENT table, both STUD_NO and PHONE are candidate keys. If STUD_NO is chosen as the primary key, then PHONE would be considered an alternate key.

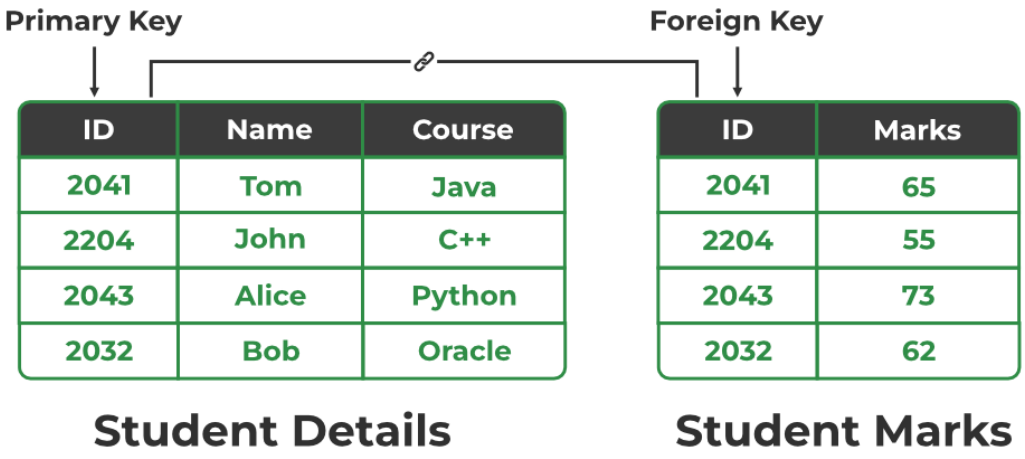


Primary Key, Candidate Key, and Alternate Key

5. Foreign Key

A foreign key is an attribute in one table that refers to the primary key in another table.

The table that contains the foreign key is called the referencing table and the table that is referenced is called the referenced table.



Relation between Primary Key and Foreign Key

- A foreign key in one table points to the primary key in another table, establishing a relationship between them.

- It helps connect two or more tables, enabling you to create relationships between them. This is important for maintaining data integrity and preventing data redundancy.
- They act as a cross-reference between the tables.

Example: Consider the STUDENT_COURSE table

STUD_NO	TEACHER_NO	COURSE_NO
1	005	C001
2	056	C005

Explanation:

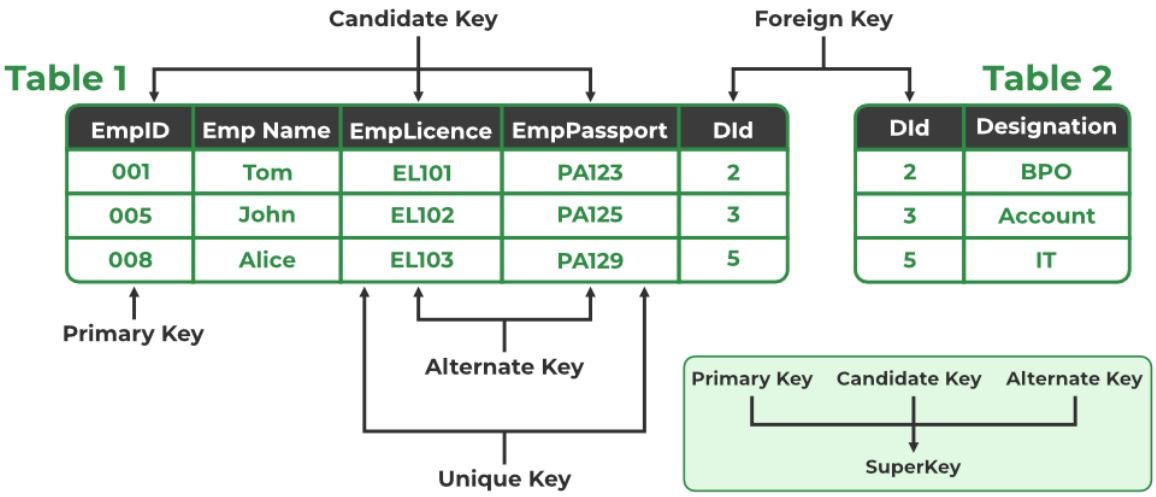
- Here, STUD_NO in the STUDENT_COURSE table is a foreign key that references the STUD_NO primary key in the STUDENT table.
- Unlike the Primary Key of any given relation, Foreign Key can be NULL as well as may contain duplicate tuples i.e. it need not follow uniqueness constraint. For Example, STUD_NO in the STUDENT_COURSE relation is not unique.
- It has been repeated for the first and third tuples. However, the STUD_NO in STUDENT relation is a primary key and it needs to be always unique and it cannot be null.

6. Composite Key

Sometimes, a table might not have a single column/attribute that uniquely identifies all the records of a table. To uniquely identify rows of a table, a combination of two or more columns/attributes can be used. It still can give duplicate values in rare cases. So, we need to find the optimal set of attributes that can uniquely identify rows in a table.

- It acts as a primary key if there is no primary key in a table
- Two or more attributes are used together to make a composite key .
- Different combinations of attributes may give different accuracy in terms of identifying the rows uniquely.

Example: In the STUDENT_COURSE table, {STUD_NO, COURSE_NO} can form a composite key to uniquely identify each record.



Different Types of Keys

5) Explain Generalization and Specialization in DBMS using ER diagrams. 5M

Ans:

Explain Generalization and Specialization in DBMS using ER diagrams

Ans:

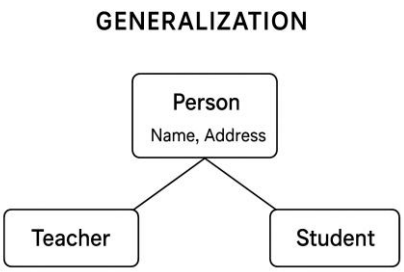
Inheritance

- **Definition: Inheritance is a concept where a child entity (subclass) acquires attributes and relationships of a parent entity (superclass).**
- **It is used in E-R modeling to represent "IS-A" relationships.**
- **Example:**
    - **Color (Superclass)**
    - **Green, Red (Subclasses) inherit properties of Color.**

1. **Superclass → Higher-level entity (common attributes).**
2. **Subclass → Lower-level entity (inherits attributes + can have extra attributes).**
3. **IS-A Relationship → Connects superclass and subclass.**

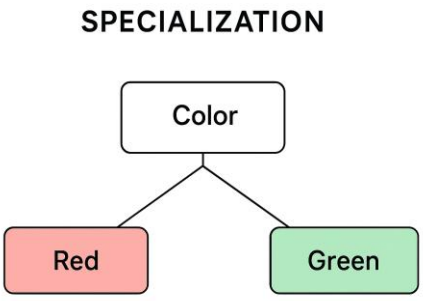
1. Generalization

- **Definition: Generalization is the process of combining two or more lower-level entities into a higher-level entity based on common features.**
- **It is a bottom-up approach.**
- **Example:**
    - **Entities: Teacher and Student**
    - **Both have common attributes like Name, Address.**
    - **These common attributes are taken to form a higher-level entity Person.**
- **Generalization: Many → One (combine entities into a super entity).**

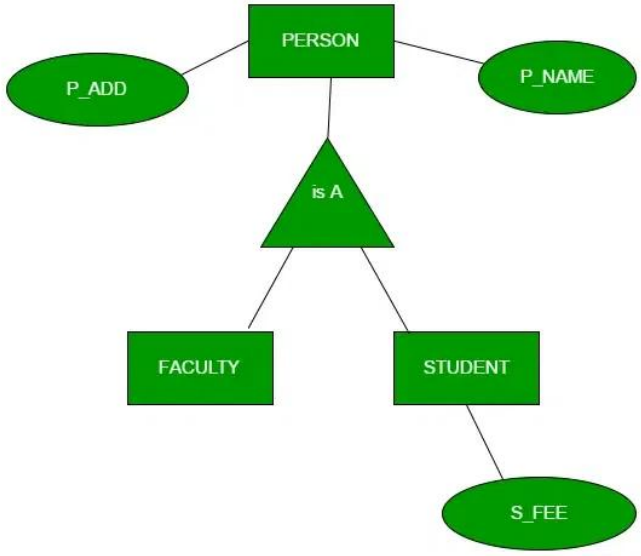


2. Specialization

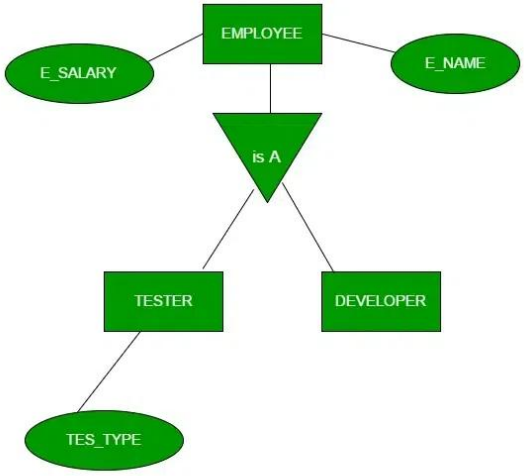
- **Definition: Specialization is the process of creating sub-entities from a higher-level entity based on some distinguishing characteristics.**
- **It is a top-down approach.**
- **Example:**
    - **Higher entity: Employee**
    - **Specialized into: Engineer, Manager based on role.**
- **Specialization: One → Many (split an entity into sub-entities).**



ER Diagram to represent Generalization



ER Diagram to represent Specialization:



Specialization

6. Explain in detail about the strong entity set and weak entity set in ER diagrams 5M

Ans:

An entity is a "thing" or "object" in the real world. An entity contains attributes, which describe that entity.

Entities in the database and must be distinguishable, i.e., easily recognized from the group.

Strong Entity:

A strong entity is not dependent on any other entity in the schema. A strong entity will always have a primary key.

Strong entities are represented by a single rectangle.

The relationship of two strong entities is represented by a single diamond.

Various strong entities, when combined together, create a strong entity set.

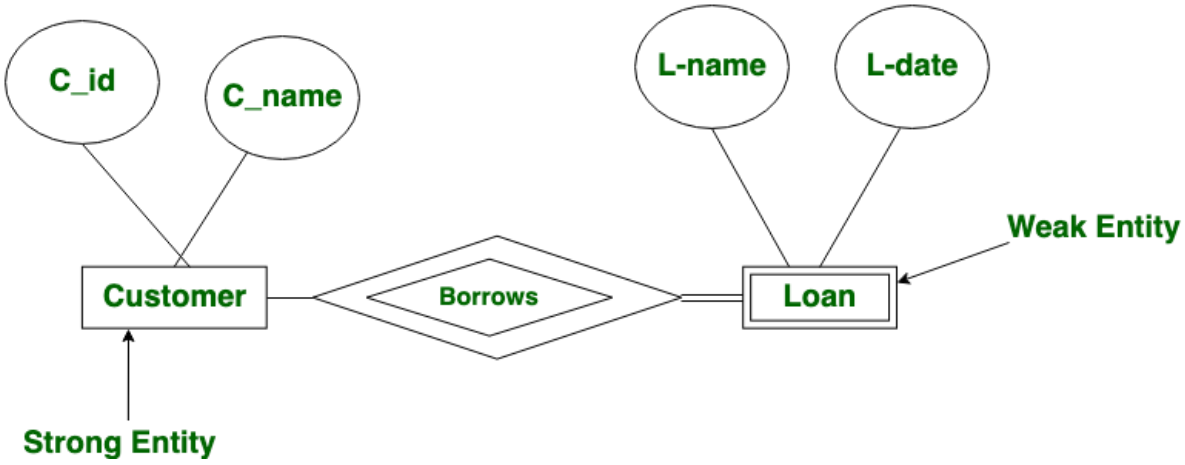
Weak Entity:

A weak entity is dependent on a strong entity to ensure its existence.

Unlike a strong entity, a weak entity does not have any primary key. It instead has a partial discriminator key.

A weak entity is represented by a double rectangle.

The relation between one strong and one weak entity is represented by a double diamond. This relationship is also known as an identifying relationship.



In ER models, strong entities can exist independently, whereas weak entities depend on strong entities.

Difference Between Strong and Weak Entity

Strong Entity	Weak Entity
Strong entity always has a primary key.	While a weak entity has a partial discriminator key.

Strong Entity	Weak Entity
Strong entity is not dependent on any other entity.	Weak entity depends on strong entity.
Strong entity is represented by a single rectangle.	Weak entity is represented by a double rectangle.
Two strong entity's relationship is represented by a single diamond.	While the relation between one strong and one weak entity is represented by a double diamond.
Strong entities have either total participation or partial participation.	A weak entity has a total participation constraint.

7. Draw and explain the detailed system architecture of DBMS. 5M

Ans:

Database Architecture

- Also included as Components of database management or database structure.
- A database system is partitioned into modules that deal with each of the responsibilities of the overall system.
- Functional components of a database system can be broadly divided into,
    - Storage Manager
    - Query Processor

1. Disk Storage

- All data or information stored in disk storage
- Various components of disk storage
    - **Data** → Data files which stores the database itself
    - **Data directory**
        - It contains meta data → data about data
        - Schema of table is an example of meta data
        - A database system consults the data directory before reading and modifying the actual data.
    - **Indices** → It provides fast access to data items that holds particular values.

- **Statistical data** → Stores information/statistics of data stored.

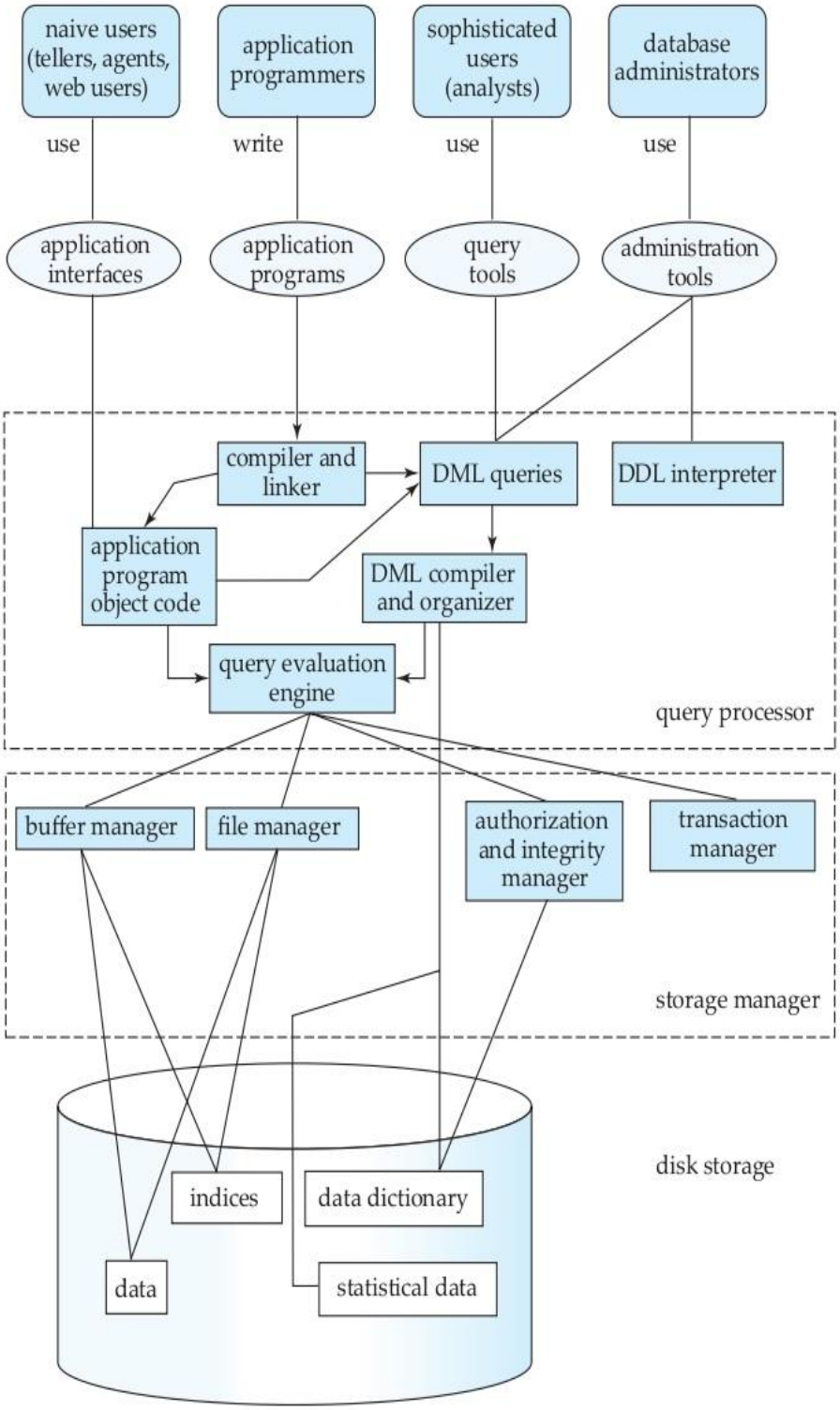


Figure 1.5 System structure.

2. Storage Manager

- Storage manager is a program module that provide interface between,
  - Low level data stored in database and
  - Application programs and
  - Queries submitted to system.
- Storage manager is responsible for the interaction with file manager.
- Storage manager translates DML statements into low-level file system commands.
- Storage manager is responsible for
  - Storing
  - Retrieving
  - Updating data in database

Components of Storage manager

a) **Authorization and integrity manager**
   - Checks **authority of users** accessing the data
   - It **tests integrity and constraints of data**

b) **Transaction manager**
   - It ensures that the database remains in a consistent state despite of failure.
   - **Concurrent execution in maintained without any conflict.**

c) **File manager**
   - It manages **allocation of space** on disk storage
   - Information stored on disk is represented using database.

d) **Buffer manager**
   - It is responsible for fetching data from disk storage into main memory.
   - It decides what data to cache the main memory

2. Query processor

- Query processor is an important part of the database system
- It helps the database system to simplify and facilitate access to data.

Various components of Query processor

DDL Interpreter
   - It interprets DDL statements into low level.
   - Records the definition in the data directory.

b) **DML Compiler**
   - It translates DML query statement in query language into low-level instruction.
   - Query evaluation engine understands only low level instruction.
   - Query can be translated into many number of evaluation plans which produces same result.
   - Query optimization is picking up the lowest cost evaluation plan among many alternatives. It is performed by DML compiler.

c) Query evaluation engine
   - Executes low level instruction generated by DML compiler.

8. Explain about Database Applications. 5M

Ans:

A Database Application is a computer program that interacts with a Database Management System (DBMS) to access, manage, and manipulate data. It provides an interface between users and the database.

These applications allow users to insert, update, delete, and retrieve information in an efficient and organized way.

DBMS applications range from banking to education, ensuring the effective management of large amounts of data.

1. Railway Reservation System

In the rail route reservation framework, the information base is needed to store the record or information of ticket appointments, status of train's appearance, and flight. Additionally, if trains get late, individuals become acquainted with it through the information base update.

2. Library Management System

There are many books in the library so; it is difficult to store the record of the relative multitude of books in a register or duplicate. Along these lines, DBMS is utilized to keep up all the data identified with the name of the book, issue date, accessibility of the book, and its writer.

3. Banking

Database the executive's framework is utilized to store the exchange data of the client in the information base.

4. Education Sector

Presently, assessments are led online by numerous schools and colleges. They deal with all assessment information through the data set administration framework (DBMS).

5. Credit card exchanges

The database Management framework is utilized for buying on charge cards and age of month to month proclamations.

6. Social Media Sites

We all utilization of online media sites to associate with companions and to impart our perspectives to the world. Every day, many people group pursue these online media accounts like Pinterest, Facebook, Twitter, and Google in addition to. By the utilization of the data set administration framework, all the data of clients are put away in the information base and, we become ready to interface with others.

7. Broadcast communications

Without DBMS any media transmission organization can't think. The Database is fundamental for these organizations to store the call subtleties and month to month postpaid bills in the information base.

8. Accounting and Finance

The information base administration framework is utilized for putting away data about deals, holding and acquisition of monetary instruments, for example, stocks and bonds in a data set.

9. E-Commerce Websites

These days, web-based shopping has become a major pattern. Nobody needs to visit the shop and burn through their time. Everybody needs to shop through web based shopping sites, (for example, Amazon, Flipkart, Snapdeal) from home. So all the items are sold and added uniquely with the assistance of the information base administration framework (DBMS). Receipt charges, installments, buy data these are finished with the assistance of DBMS.

10. Human Resource Management

Big firms or organizations have numerous specialists or representatives working under them. They store data about worker's compensation, assessment, and work with the assistance of an information base administration framework (DBMS).

11. Manufacturing

Manufacturing organizations make various kinds of items and deal them consistently. To keep the data about their items like bills, acquisition of the item, amount, inventory network the executives, information base administration framework (DBMS) is utilized.

12. Airline Reservation System

This framework is equivalent to the railroad reservation framework. This framework additionally utilizes an information base administration framework to store the records of flight takeoff, appearance, and defer status.

13. Healthcare System

DBMS is used in healthcare to manage patient data, medical records, and billing information.

14. Security

DBMS provides security features to ensure that only authorized users have access to the data.

15. Telecommunication

Database Management Systems (DBMS) are essential to the telecommunications industry because they manage enormous volumes of data on billing, customer information, and network optimization.

9) What are the different data models? Explain its types in detail. 5M

Ans:

Data Model: Underlying the structure of a database is the data model: a collection of conceptual tools for describing

- **Data**
- **data relationships**
- **data semantics and**
- **consistency constraints.**

A data model provides a way to describe the design of a database at the physical, logical, and view levels.

1) Relational Model: The relational model uses a collection of tables to represent both data and the relationships among those data.

Each table has multiple columns, and each column has a unique name. Tables are also known as relations.

The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types.

Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record type.

The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.

Eg.

Emp_id	Emp_name	Job_name	Salary	Mobile_no	Dep_id	Project_id
AfterA001	John	Engineer	100000	9111037890	2	99
AfterA002	Adam	Analyst	50000	9587569214	3	100
AfterA003	Kande	Manager	890000	7895212355	2	65

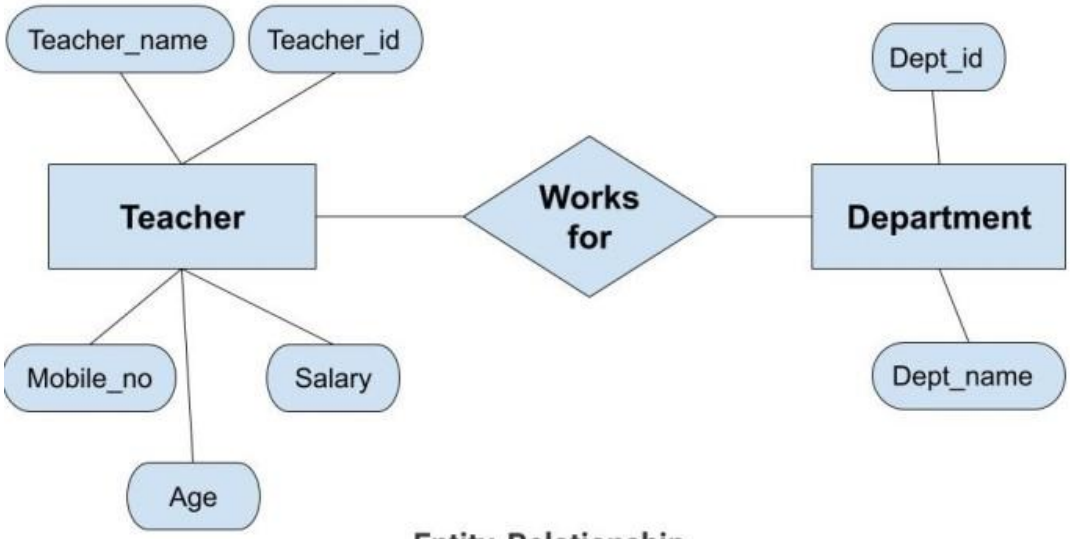
EMPLOYEE TABLE

Entity-Relationship Model:

The entity-relationship (E-R) data model uses a collection of basic objects, called *entities*, and *relationships* among these objects.

An entity is a “thing” or “object” in the real world that is distinguishable from other objects.

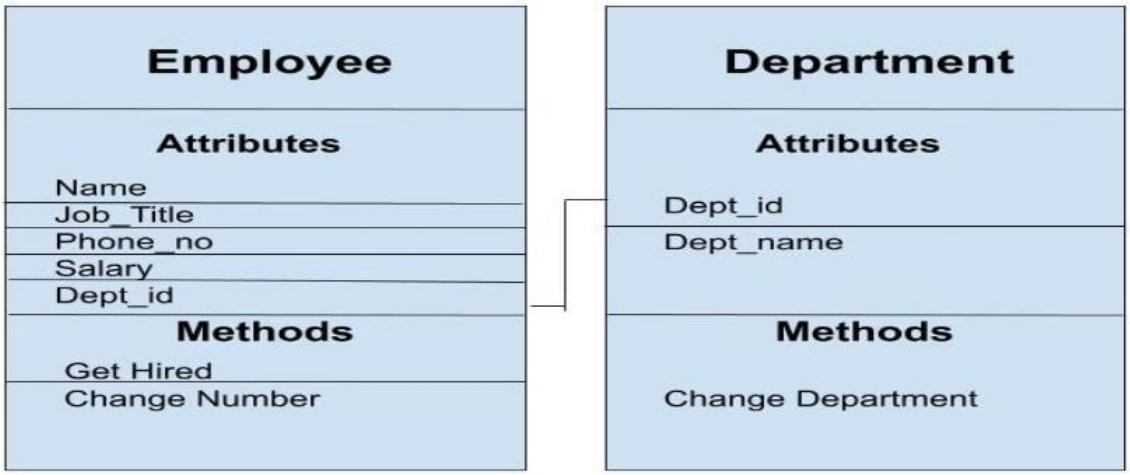
The entity-relationship model is widely used in database design.



Entity-Relationship Model

Object-Based Data Model:

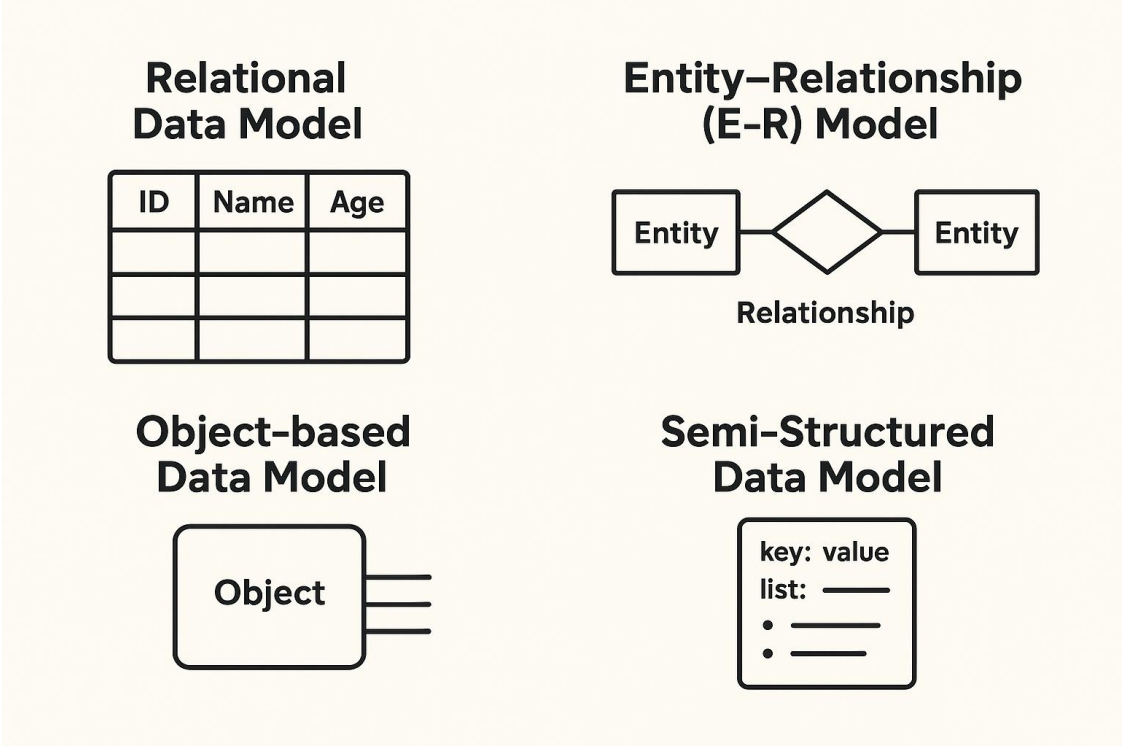
Object-oriented programming (especially in Java,C++, or C#) has become the dominant software-development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. The object-relational data model combines features of the object-oriented data model and relational data model.



Object_Oriented_Model

Semistructured Data Model:

The semi-structured data model permits the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The Extensible Markup Language (XML) is widely used to represent semi-structured data.

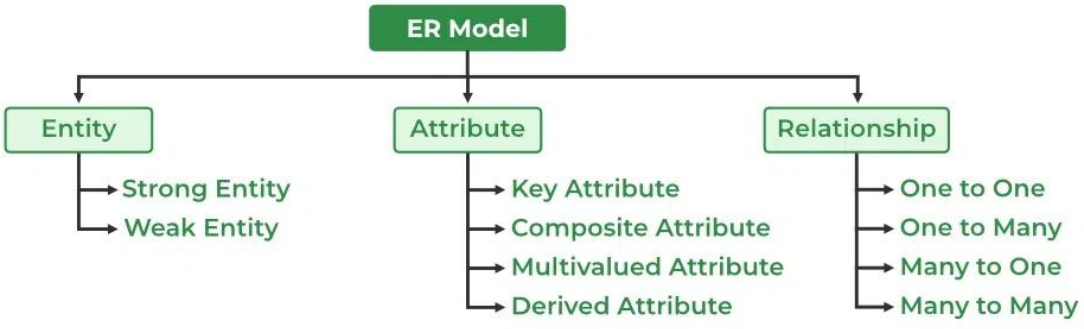


10) Discuss Components of ER-Model. Write about Relationship types in ER Model. 8M

Ans:

The Entity-Relationship Model (ER Model) is a conceptual model for designing a databases. This model represents the logical structure of a database, including entities, their attributes and relationships between them.

- **Entity:** An objects that is stored as data such as *Student*, *Course* or *Company*.
- **Attribute:** Properties that describes an entity such as *StudentID*, *CourseName*, or *EmployeeEmail*.
- **Relationship:** A connection between entities such as "a *Student* enrolls in a *Course*".



What is an Entity?

An Entity represents a real-world object, concept or thing about which data is stored in a database. It act as a building block of a database. Tables in relational database represent these entities.

Example of entities:

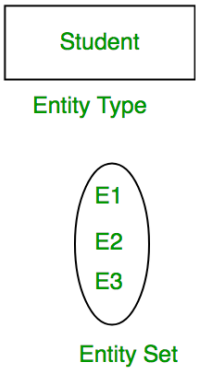
- **Real-World Objects:** Person, Car, Employee etc.
- **Concepts:** Course, Event, Reservation etc.
- **Things:** Product, Document, Device etc.

The entity type defines the structure of an entity, while individual instances of that type represent specific entities.

What is an Entity Set?

An entity refers to an individual object of an entity type, and the collection of all entities of a particular type is called an entity set. For example, E1 is an entity that belongs to the entity type "Student," and the group of all students forms the entity set.

In the ER diagram below, the entity type is represented as:



We can represent the entity sets in an ER Diagram but we can't represent individual entities because an entity is like a row in a table, and an ER diagram shows the structure and relationships of data, not specific data entries (like rows and columns). An ER diagram is a visual representation of the data model, not the actual data itself.

Types of Entity

There are two main types of entities:

1. Strong Entity

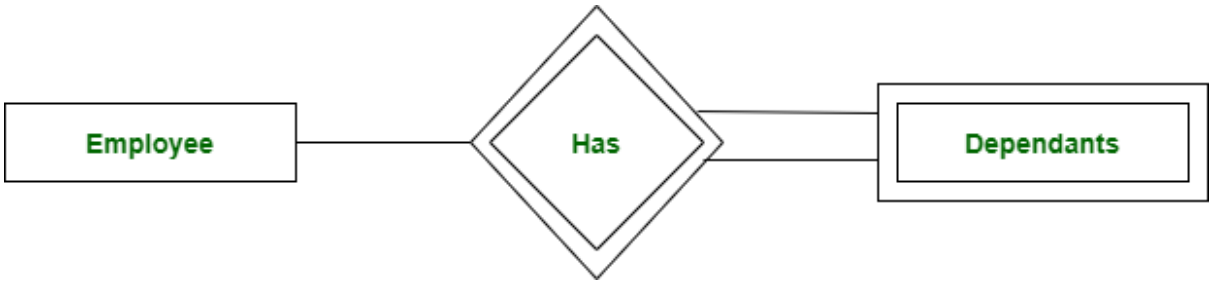
A Strong Entity is a type of entity that has a key Attribute that can uniquely identify each instance of the entity. A Strong Entity does not depend on any other Entity in the Schema for its identification. It has a primary key that ensures its uniqueness and is represented by a rectangle in an ER diagram.

2. Weak Entity

A Weak Entity cannot be uniquely identified by its own attributes alone. It depends on a strong entity to be identified. A weak entity is associated with an identifying entity (strong entity), which helps in its identification. A weak entity are represented by a double rectangle. The participation of weak entity types is always total. The relationship between the weak entity type and its identifying strong entity type is called identifying relationship and it is represented by a double diamond.

Example:

A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents can't exist without the employee. So dependent will be a Weak Entity Type and Employee will be identifying entity type for dependent, which means it is Strong Entity Type.



Strong Entity and Weak Entity

Attributes in ER Model

Attributes are the properties that define the entity type. For example, for a Student entity Roll_No, Name, DOB, Age, Address, and Mobile_No are the attributes that define entity type Student. In ER diagram, the attribute is represented by an oval.

Attribute

Types of Attributes

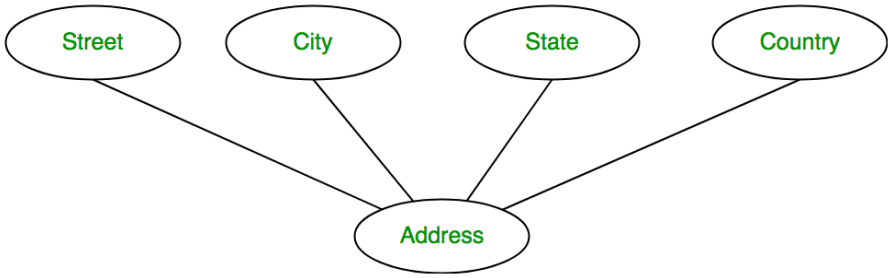
1. Key Attribute

The attribute which uniquely identifies each entity in the entity set is called the key attribute. For example, Roll_No will be unique for each student. In ER diagram, the key attribute is represented by an oval with an underline.

Key Attribute

2. Composite Attribute

An attribute composed of many other attributes is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country. In ER diagram, the composite attribute is represented by an oval comprising of ovals.



Composite Attribute

3. Multivalued Attribute

An attribute consisting of more than one value for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, a multivalued attribute is represented by a double oval.

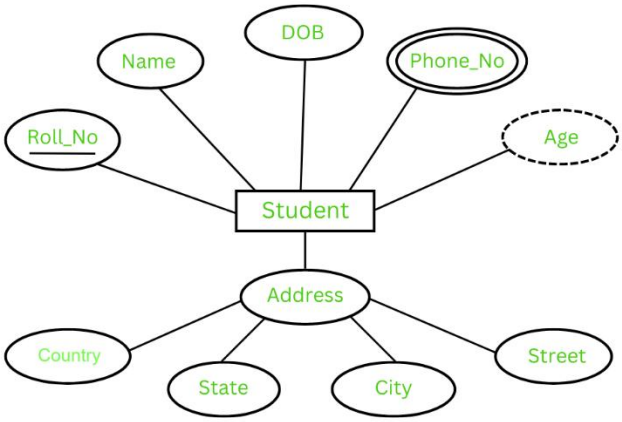
Multivalued Attribute

4. Derived Attribute

An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, the derived attribute is represented by a dashed oval.

Derived Attribute

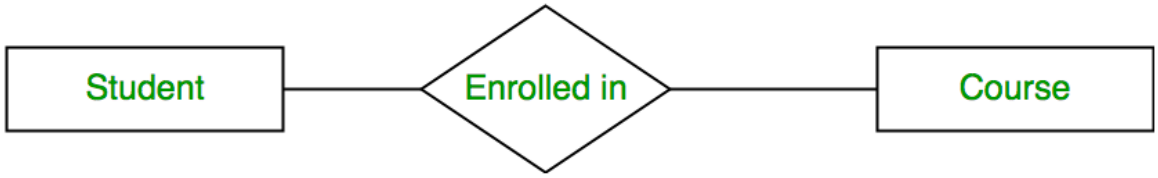
The Complete Entity Type Student with its Attributes can be represented as:



Entity and Attributes

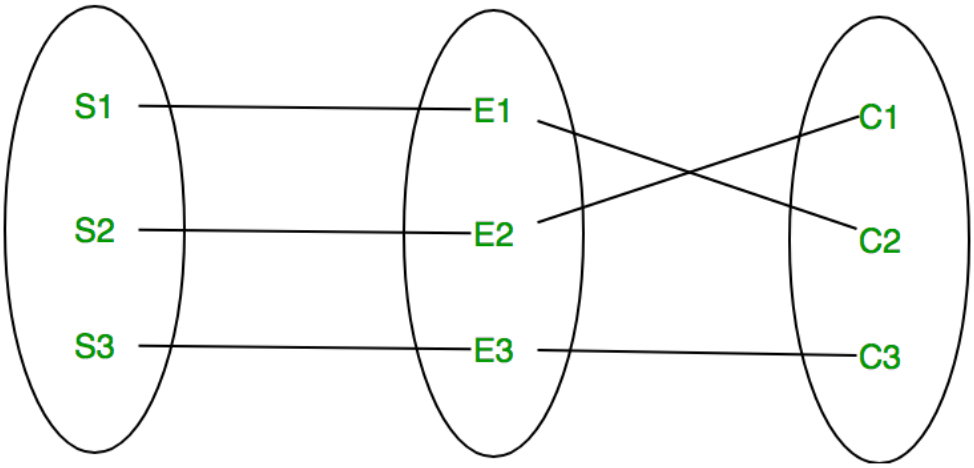
Relationship Type and Relationship Set

A Relationship Type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, the relationship type is represented by a diamond and connecting the entities with lines.



Entity-Relationship Set

A set of relationships of the same type is known as a relationship set. The following relationship set depicts S1 as enrolled in C2, S2 as enrolled in C1, and S3 as registered in C3.

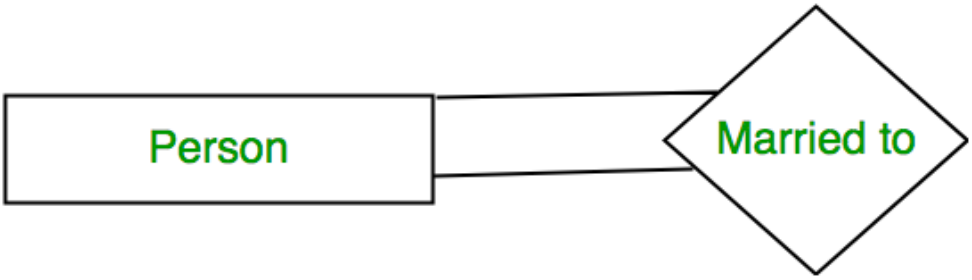


Relationship Set

Degree of a Relationship Set

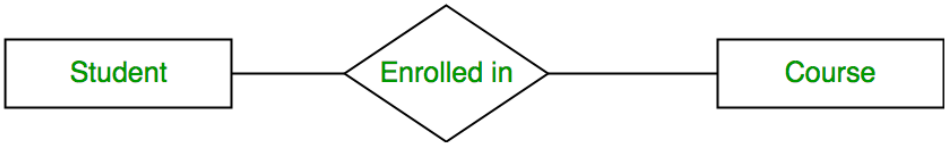
The number of different entity sets participating in a relationship set is called the degree of a relationship set.

1. Unary Relationship: When there is only ONE entity set participating in a relation, the relationship is called a unary relationship. For example, one person is married to only one person.



Unary Relationship

2. Binary Relationship: When there are TWO entities set participating in a relationship, the relationship is called a binary relationship. For example, a Student is enrolled in a Course.



Binary Relationship

3. Ternary Relationship: When there are three entity sets participating in a relationship, the relationship is called a ternary relationship.

4. N-ary Relationship: When there are n entities set participating in a relationship, the relationship is called an n-ary relationship.

Cardinality in ER Model

The maximum number of times an entity of an entity set participates in a relationship set is known as cardinality.

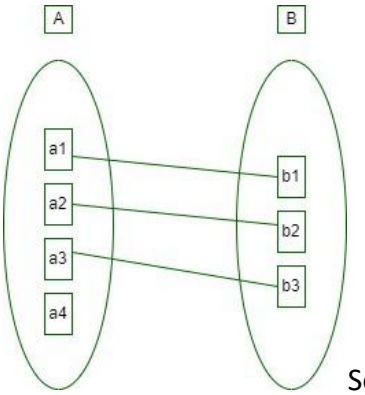
Cardinality can be of different types:

1. One-to-One

When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one. Let us assume that a male can marry one female and a female can marry one male. So the relationship will be one-to-one.

One to One Cardinality

Using Sets, it can be represented as:



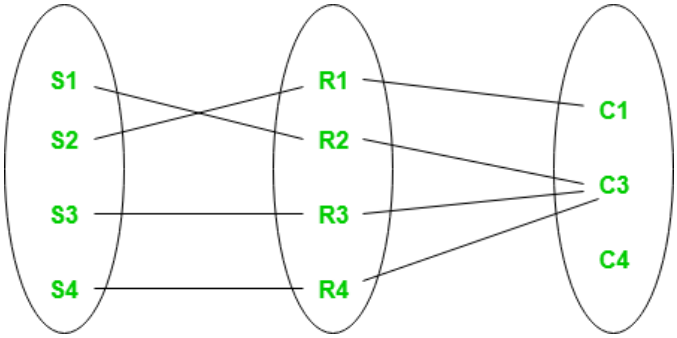
Set Representation of One-to-One

2. One-to-Many

In one-to-many mapping as well where each entity can be related to more than one entity. Let us assume that one surgeon department can accommodate many doctors. So the Cardinality will be 1 to M. It means one department has many Doctors.

one to many cardinality

Using sets, one-to-many cardinality can be represented as:



Set Representation of One-to-Many

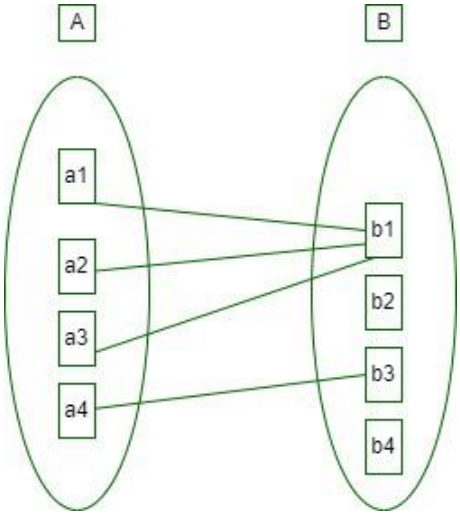
3. Many-to-One

When entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set, cardinality is many to one.

Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.

many to one cardinality

Using Sets, it can be represented as:



Set Representation of Many-to-One

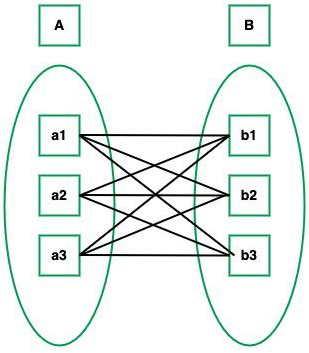
In this case, each student is taking only 1 course but 1 course has been taken by many students.

4. Many-to-Many

When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.

many to many cardinality

Using Sets, it can be represented as:



Many-to-Many Set Representation

In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3, and S4. So it is many-to-many relationships.

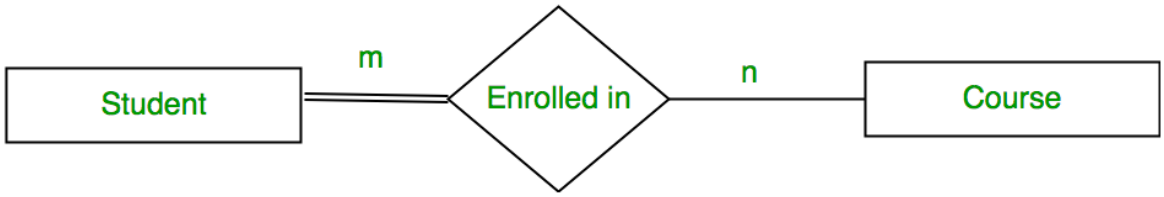
Participation Constraint

Participation Constraint is applied to the entity participating in the relationship set.

1. Total Participation: Each entity in the entity set must participate in the relationship. If each student must enroll in a course, the participation of students will be total. Total participation is shown by a double line in the ER diagram.

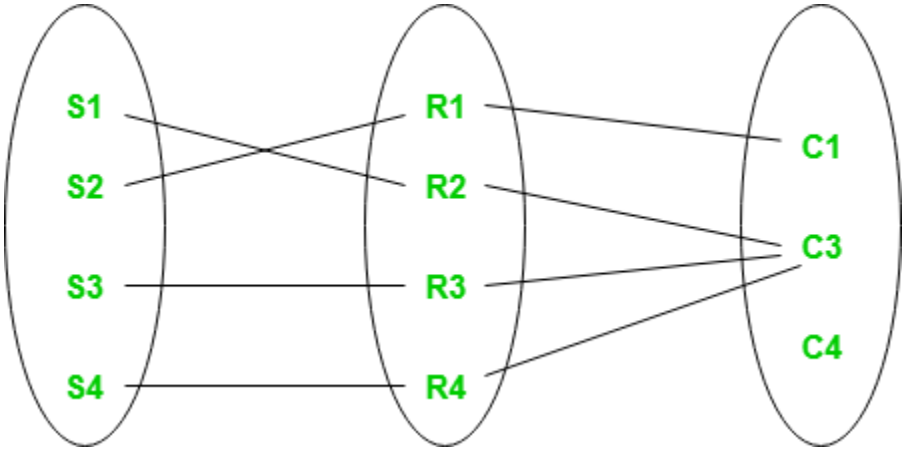
2. Partial Participation: The entity in the entity set may or may NOT participate in the relationship. If some courses are not enrolled by any of the students, the participation in the course will be partial.

The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



Total Participation and Partial Participation

Using Set, it can be represented as,



Set representation of Total Participation and Partial Participation

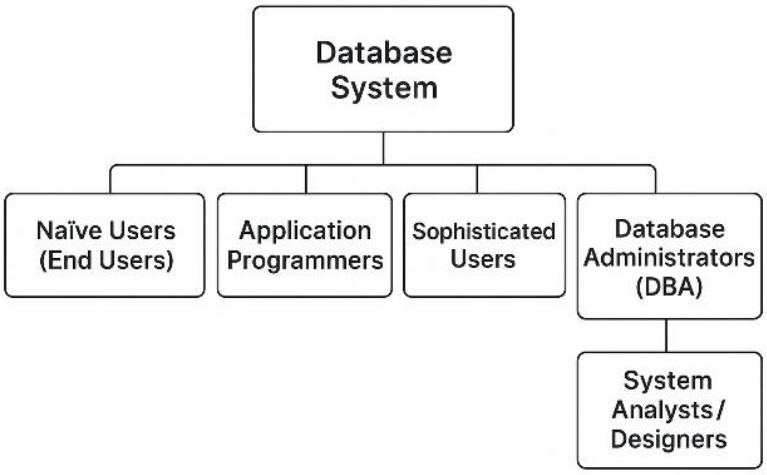
Every student in the Student Entity set participates in a relationship but there exists a course C4 that is not taking part in the relationship.

12) Discuss in detail about different types of database users. 5M

Ans:

Database Users : In DBMS, users are classified based on how they interact with the database. Every category of user has different roles, responsibilities, and knowledge levels about the system.

- **Naïve Users** → Just use the system.
- **Application Programmers** → Develop applications.
- **Sophisticated Users** → Directly query the DB.
- **DBA** → Manage and control the DB.
- **System Analysts/Designers** → Design database structure.



1. Naïve Users (End Users)

- They interact with the database through application programs.
- Do not write queries.
- Example: Bank customers using an ATM, online shopping users.

2. Application Programmers

- Write application programs that interact with the database.
- Use programming languages with embedded SQL (like Java, C#, Python).
- Example: Developer creating payroll software.

3. Sophisticated Users

- Write complex queries using SQL.
- Use analytical tools and database query interfaces.
- Example: Data scientists running data mining queries.

4. Database Administrators (DBA)

- Manage the entire DBMS.

- Responsibilities:
    - Schema definition
    - Storage management
    - Backup & recovery
    - Security & authorization
- Example: IT staff in a company managing Oracle, MySQL, etc.

5. System Analysts / Designers

- Design database structure.
- Ensure database meets business needs.
- Act as a bridge between end users and programmers.

13) Explain about data independence. 5M

Ans:

Data independence in DBMS is the **ability to modify the database schema at one level without requiring changes to the schema at the next higher level**.

Types of Data Independence

Physical Data Independence

- This is the **capacity to change the internal (physical) schema** (such as file organization, storage device, or indexing strategy) without having to change the conceptual (logical) schema.
- Examples: moving data files to a new drive, switching storage devices (HDD to SSD), or changing indexing methods—all without impacting the database structure visible to users or applications.

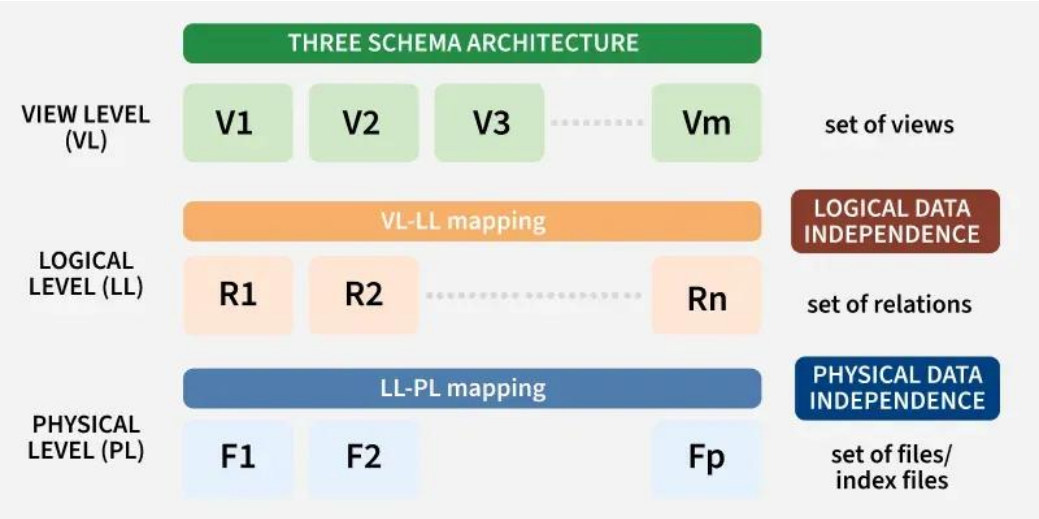
Logical Data Independence

- This refers to the **ability to modify the logical (conceptual) schema** (such as adding/removing tables, attributes, or relationships) without affecting the external views or requiring changes in the application programs.
- Examples: adding a new column to a table, merging tables, or splitting a record without the need to rewrite application code.

Importance of Data Independence

- **Reduces maintenance**: Applications remain unaffected by changes made in storage or structure, which minimizes code rewrites.
- **Increases flexibility**: Enables database administrators to optimize or reorganize internal structures as needed, supporting long-term growth.

- **Supports data abstraction**: Hides implementation details from users, providing a stable and simple interface for data access.



Type	What It Means	Example
Physical	Change storage/internal schema without affecting logical	Move files, change indexes, new storage device
Logical	Change logical/conceptual schema without affecting views	Add column/table, merge records, update relationships

Data independence is a core feature of DBMS, making systems easier to maintain and adapt as needs evolve.

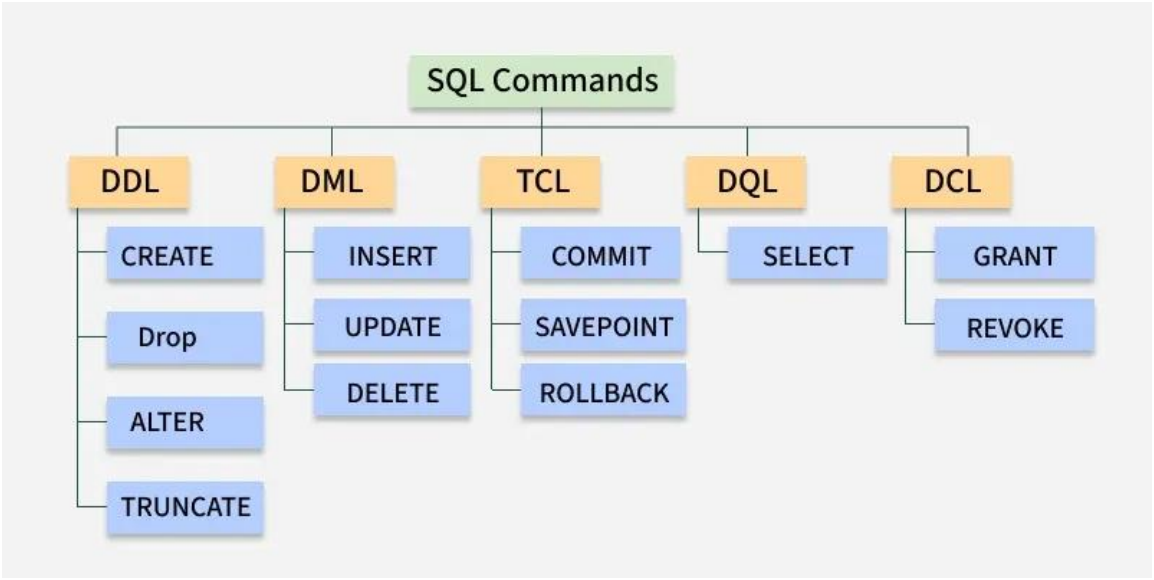
14) Classify DDL and DML Commands in SQL with Example for each. 5M

Ans:

SQL commands are used to interact with database with some operations.

It is also used to perform specific tasks, functions and queries of data. SQL can perform various tasks like creating a table, adding data to tables, dropping the table, modifying the table, set permission for users.

SQL Commands are mainly categorized into five categories:



SQL Commands

1. DDL - Data Definition Language

DDL (Data Definition Language) actually consists of SQL commands that can be used for defining, altering and deleting database structures such as tables, indexes and schemas. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database

Common DDL Commands

Command	Description	Syntax
CREATE	Create database or its objects (table, index, function, views, store procedure and triggers)	CREATE TABLE table_name (column1 data_type, column2 data_type, ...);
DROP	Delete objects from the database	DROP TABLE table_name;
ALTER	Alter the structure of the database	ALTER TABLE table_name ADD COLUMN column_name data_type;
TRUNCATE	Remove all records from a table, including all spaces allocated for the records are removed	TRUNCATE TABLE table_name;
COMMENT	Add comments to the data dictionary	COMMENT ON TABLE table_name IS 'comment_text';

Command	Description	Syntax
RENAME	Rename an object existing in the database	RENAME TABLE old_table_name TO new_table_name;

Example:

CREATE TABLE employees (
employee_id INT PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
hire_date DATE
);

In this example, a new table called employees is created with columns for employee ID, first name, last name and hire date.

2. DQL - Data Query Language

DQL is used to fetch data from the database. The main command is SELECT, which retrieves records based on the query. The output is returned as a result set (a temporary table) that can be viewed or used in applications.

DQL Command

Command	Description	Syntax
SELECT	It is used to retrieve data from the database	SELECT column1, column2, ...FROM table_name WHERE condition;
FROM	Indicates the **table(s)** from which to retrieve data.	SELECT column1 FROM table_name;
WHERE	Filters rows **before** any grouping or aggregation	SELECT column1 FROM table_name WHERE condition;
GROUP BY	Groups rows that have the same values in specified columns.	SELECT column1, AVG_FUNCTION(column2) FROM table_name GROUP BY column1;

Command	Description	Syntax
HAVING	Filters the results of GROUP BY	SELECT column1, AVG_FUNCTION(column2) FROM table_name GROUP BY column1 HAVING condition;
DISTINCT	Removes **duplicate rows** from the result set	SELECT DISTINCT column1, column2, ... FROM table_name;
ORDER BY	Sorts the result set by one or more columns	SELECT column1 FROM table_name ORDER BY column1 [ASC | DESC];
LIMIT	By default, it sorts in **ascending order** unless specified as DESC	SELECT * FROM table_name LIMIT number;

Note: *DQL has only one command,* ***SELECT****. Other terms like FROM, WHERE, GROUP BY, HAVING, ORDER BY, DISTINCT and LIMIT are* ***clauses*** *of SELECT, not separate commands.*

Example:

SELECT first_name, last_name, hire_date
FROM employees
WHERE department = 'Sales'
ORDER BY hire_date DESC;

This query retrieves employees first and last names, along with their hire dates, from the employees table, specifically for those in the 'Sales' department, sorted by hire date.

3. DML - Data Manipulation Language

DML commands are used to manipulate the data stored in database tables. With DML, you can insert new records, update existing ones, delete unwanted data or retrieve information.

Common DML Commands

Command	Description	Syntax
INSERT	Insert data into a table	INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);

Command	Description	Syntax
UPDATE	Update existing data within a table	UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;
DELETE	Delete records from a database table	DELETE FROM table_name WHERE condition;

Example:

INSERT INTO employees (first_name, last_name, department)
VALUES ('Jane', 'Smith', 'HR');

This query inserts a new record into employees table with first name 'Jane', last name 'Smith' and department 'HR'.

4. DCL - Data Control Language

DCL (Data Control Language) includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions and other controls of the database system. These commands are used to control access to data in the database by granting or revoking permissions.

Common DCL Commands

Command	Description	Syntax
GRANT	Assigns new privileges to a user account, allowing access to specific database objects, actions or functions.	GRANT privilege_type [(column_list)] ON [object_type] object_name TO user [WITH GRANT OPTION];
REVOKE	Removes previously granted privileges from a user account, taking away their access to certain database objects or actions.	REVOKE [GRANT OPTION FOR] privilege_type [(column_list)] ON [object_type] object_name FROM user [CASCADE];

Example:

GRANT SELECT, UPDATE ON employees TO user_name;

This command grants the user user_name the permissions to select and update records in the employees table.

5. TCL - Transaction Control Language

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group are successfully completed. If any of the tasks fail, transaction fails. Therefore, a transaction has only two results: success or failure.

Common TCL Commands

Command	Description	Syntax
BEGIN TRANSACTION	Starts a new transaction	BEGIN TRANSACTION [transaction_name];
COMMIT	Saves all changes made during the transaction	COMMIT;
ROLLBACK	Undoes all changes made during the transaction	ROLLBACK;
SAVEPOINT	Creates a savepoint within the current transaction	SAVEPOINT savepoint_name;

Example:

```
BEGIN TRANSACTION;
UPDATE employees SET department = 'Marketing' WHERE department = 'Sales';
SAVEPOINT before_update;
UPDATE employees SET department = 'IT' WHERE department = 'HR';
ROLLBACK TO SAVEPOINT before_update;
COMMIT;
```

In this example, a transaction is started, changes are made and a savepoint is set. If needed, the transaction can be rolled back to the savepoint before being committed.

15) Explain in detail about Relational Calculus. 5M

Ans:

RELATIONAL CALCULUS

Relational Algebra (RA) and **Relational Calculus (RC)** are two **formal query languages** in Data Base Management System:

Relational Algebra is a PROCEDURAL LANGUAGE

- we must explicitly provide a *sequence of operations* to generate a desired output

Relational Calculus is a DECLARATIVE LANGUAGE (NON-PROCEDURAL)

- we specify what to retrieve, not how to retrieve it

Both Relational Calculus and Relational Algebra are based on 1st order predicate calculus.

If a retrieval can be specified in the relational calculus, it can be specified in the relational algebra, and vise versa.

That means, expressive power of both the languages are identical.

A query language L is Relationally complete if L can express any query that can be expressed in the relational calculus

RELATIONAL ALGEBRA vs. **RELATIONAL CALCULUS**

- Procedural
- Meaning: How to get the result
- Operators: σ =; π ⍼ × ×
- Step-by-step process: sequence of operations must be specified

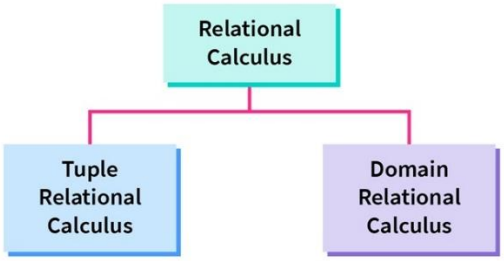
- Non-procedural (Declarative)
- Meaning: What result we want
- Uses logical predicates
- Focuses on specifiying properties of the result

$\pi_{Name}(\sigma_{Dept} = \text{'CSE'}\ (\text{Student}))$

Example: $\begin{cases} t.\text{Name} \mid t \in \text{Student} \\ \wedge\ t.\text{Dept} = \text{'CSE'} \end{cases}$

There are two types of relational calculus:

- Tuple Relational Calculus (TRC)
- Domain Relational Calculus (DRC)



1) Tuple Relational Calculus (TRC)

Tuple Relational Calculus is specified to select tuples (rows or records) in a relation. In TRC filtering variable was tuple of a relation. The result can have one or more tuples of the relation.

A query is of the form:

$$\{t \mid P(t)\}$$

Where:

- **t** = tuple variable
- **P(t)** = a predicate (condition) that must be true

Operations in Tuple Relational Calculus:

Selection (conditions)

Projection (specific attributes)

Conjunction (AND-∧), Disjunction (OR- ∨), Negation (NOT- ¬)

Quantifiers: ∃ (exists), ∀ (for all)

1. Selection (σ in RA)

- Select tuples that satisfy a condition.
- Example: Find students in CSE:

$$\{t \mid t \in Student \land t.Dept =' CSE'\}$$

2. Projection (π in RA)

- Retrieve specific attributes.
- Example: Get names of students in CSE:

$$\{t.Name \mid t \in Student \land t.Dept =' CSE'\}$$

3. Conjunction (AND, ∧)

- Combine multiple conditions.
- Example: Students in CSE **and** Percentage > 70:

$$\{t \mid t \in Student \land t.Dept =' CSE' \land t.Percentage > 70\}$$

4. Disjunction (OR, ∨)

- At least one condition must be true.
- Example: Students in CSE **or** ECE:

$$\{t \mid t \in Student \land (t.Dept =' CSE' \lor t.Dept =' ECE')\}$$

5. Negation (NOT, ¬)

- Exclude certain conditions.
- Example: Students **not** in CSE:

$$\{t \mid t \in Student \land \neg(t.Dept =' CSE')\}$$

6. Existential Quantifier (∃)

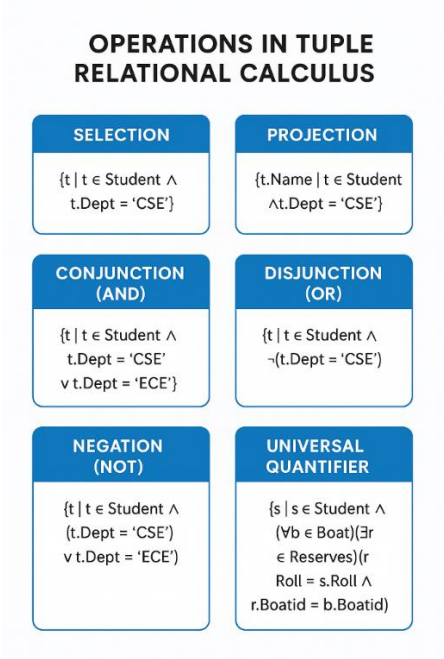
- "There exists" at least one tuple satisfying condition.
- Example: Find students who reserved at least one boat:

$$\{s \mid s \in Student \land (\exists r \in Reserves)(r.Roll = s.Roll)\}$$

7. Universal Quantifier (∀)

- "For all" tuples, condition must hold.
- Example: Students who reserved **all boats**:

$$\{s \mid s \in Student \land (\forall b \in Boat)(\exists r \in Reserves)(r.Roll = s.Roll \land r.Boatid = b.Boatid)\}$$



2) DOMAIN RELATIONAL CALCULUS:

Domain Relational Calculus (DRC) describes queries by specifying a set of conditions or formulas that the data must satisfy.

These conditions are written using domain variables and predicates, and it returns a relation that satisfies the specified conditions.

DRC can represent constraint domain/attribute values more directly

DRC is **query-by-condition on domain variables**.

The general form of a DRC expression is:

$$\{\langle x_1, x_2, ..., x_n \rangle \mid P(x_1, x_2, ..., x_n)\}$$

Where:

- $x_1, x_2, ..., x_n$ = domain variables (values from attributes).
- $P(x_1, x_2, ..., x_n)$ = a **predicate (condition)** involving comparisons, logical connectives (AND, OR, NOT), and quantifiers (∃, ∀).

Example : Simple Selection

Suppose we have a relation:
STUDENT(Roll, Name, Dept, Percentage)

Find the names of students in the "CSE" department.

DRC Query:

$$\{\langle n \rangle \mid \exists r, d, p(STUDENT(r, n, d, p) \wedge d = "CSE")\}$$

Meaning: Select all **n (Name)** such that there exist values of roll, dept, and percentage where dept = "CSE".

Example : With Condition

Find roll numbers of students with **Percentage > 70**.

DRC Query:

$$\{\langle r \rangle \mid \exists n, d, p(STUDENT(r, n, d, p) \wedge p > 70)\}$$

Operators Used in DRC

- **Comparison operators:** =, ≠, <, >, ≤, ≥
- **Logical operators:** ∧ (AND), ∨ (OR), ¬ (NOT)

- **Quantifiers:**
    - ∃ → "there exists"
    - ∀ → "for all"

16) Explain various Join operations in Relational Algebra. 5M

Ans:

Join is an operation in DBMS(Database Management System) that combines the rows of two or more tables based on related columns between them.

The main purpose of join is to retrieve the data from multiple tables in other words Join is used to perform multi-table queries. It is denoted by ⋈.

Syntax

R3 <- ⋈(R1) <join_condition> (R2)

where R1 and R2 are two relations to be joined and R3 is a relation that will hold the result of the join operation.

Example

Temp <- ⋈(student) S.roll=E.roll(Exam)

where S and E are aliases of the student and exam respectively.

JOIN Example

Consider the two tables below as follows:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Table 1 - Student

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

Table 2 - Student_Course

Both these tables are connected by one common key (column) i.e. ROLL_NO.

We can perform a JOIN operation using the given relational algebra:

$$Student \bowtie Student\_course$$

Output:

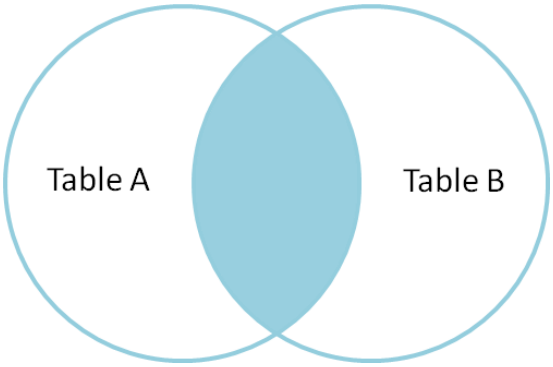
ROLL_NO	NAME	ADDRESS	PHONE	AGE	COURSE_ID
1	HARSH	DELHI	xxxxxxxxxx	18	1
2	PRATIK	BIHAR	xxxxxxxxxx	19	2
3	PRIYANKA	SILIGURI	xxxxxxxxxx	20	2
4	DEEP	RAMNAGAR	xxxxxxxxxx	18	3
5	SAPTARHI	KOLKATA	xxxxxxxxxx	19	1

Types of Join

There are many types of Joins in SQL. Depending on the use case, you can use different types of SQL JOIN clauses. Here are the frequently used SQL JOIN types:

1. Inner Join

Inner Join is a join operation in DBMS that combines two or more tables based on related columns and returns only rows that have matching values among tables. Inner join has two types.



Inner Join

- Conditional join
- Equi Join
- Natural Join

(a) Conditional Join

Conditional join or Theta join is a type of inner join in which tables are combined based on the specified condition.

In conditional join, the join condition can include <, >, <=, >=, ≠ operators in addition to the '=' operator.

Example: Suppose two tables A and B

Table A

R	S
10	5
7	20

Table B

T	U
10	12
17	6

$A \bowtie S<T\ B$

Output

R	S	T	U
10	5	10	12

Explanation: This query joins the table A, B and projects attributes R, S, T, U were the condition S < T is satisfied.

(b) Equi Join

Equi Join is a type of inner join where the join condition uses the equality operator ('=') between columns.

Example: Suppose there are two tables Table A and Table C

Table A

Column A	Column B
a	a
a	b

Table C

Column A	Column B
a	**a**
a	**c**

A ⋈ A.Column B = C.Column B (C)

Output

Column A	Column B
a	a

Explanation: The data value "a" is available in both tables Hence we write that "a" is the table in the given output.

(c) Natural Join

Natural join is a type of inner join in which we do not need any comparison operators. In natural join, columns should have the same name and domain. There should be at least one common attribute between the two tables.

Example: Suppose there are two tables Table A and Table B

Table A

Number	Square
2	4
3	9

Table B

Number	Cube
2	8
3	27

$A \bowtie B$

Output

Number	Square	Cube
2	4	8
3	9	27

Explanation - Column Number is available in both tables Hence we write the "Number column once " after combining both tables.

2. Outer Join

Outer join is a type of join that retrieves matching as well as non-matching records from related tables. There are three types of outer join

- Left outer join
- Right outer join

- Full outer join

(a) Left Outer Join

It is also called left join. This type of outer join retrieves all records from the left table and retrieves matching records from the right table.

Example: Suppose there are two tables Table A and Table B

Table A

Number	Square
2	4
3	9
4	16

Table B

Number	Cube
2	8
3	27
5	125

$A \bowtie B$

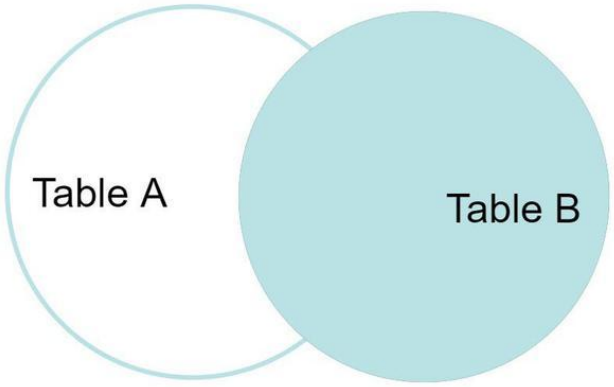
Output

Number	Square	Cube
2	4	8
3	9	27
4	16	NULL

Explanation: Since we know in the left outer join we take all the columns from the left table (Here Table A) In the table A we can see that there is no Cube value for number 4. so we mark this as NULL.

(b) Right Outer Join

It is also called a right join. This type of outer join retrieves all records from the right table and retrieves matching records from the left table. And for the record which doesn't lies in Left table will be marked as NULL in result Set.



Right Outer Join

Example: Suppose there are two tables Table A and Table B

$A \rtimes B$

Output:

Number	Square	Cube
2	4	8
3	9	27
5	NULL	125

Explanation: Since we know in the right outer join we take all the columns from the right table (Here Table B) In table A we can see that there is no square value for number 5. So we mark this as NULL.

(c) Full Outer Join

FULL JOIN creates the result set by combining the results of both LEFT JOIN and RIGHT JOIN. The result set will contain all the rows from both tables. For the rows for which there is no matching, the result set will contain *NULL* values.

Example: Table A and Table B are the same as in the left outer join

$A \rtimes B$

Output:

Number	Square	Cube
2	4	8
3	9	27
4	16	NULL
5	NULL	125

Explanation: Since we know in full outer join we take all the columns from both tables (Here Table A and Table B) In the table A and Table B we can see that there is no Cube value for number 4 and No Square value for 5 so we mark this as NULL.

17) Explain in detail about the strong entity set and weak entity set in ER diagrams 5M

Ans:

1. Strong Entity Set

- **Definition:**
  An **entity set that has a primary key** (its own unique identifier).
  It does **not depend on any other entity set** for its identification.
- **Key Points:**
  - Always has a **primary key** (e.g., Student_ID, Emp_ID).
  - Represented by a **single rectangle** in an ER diagram.
  - Can exist independently.
- **Example:**
  - Entity set: **Student(Student_ID, Name, Age)**
  - Here, Student_ID is the **primary key**, so Student is a **strong entity set**.

2. Weak Entity Set

- **Definition:**
  An **entity set that does not have sufficient attributes to form a primary key**.
  It depends on a **strong entity set** (called its **owner**) for identification.
- **Key Points:**

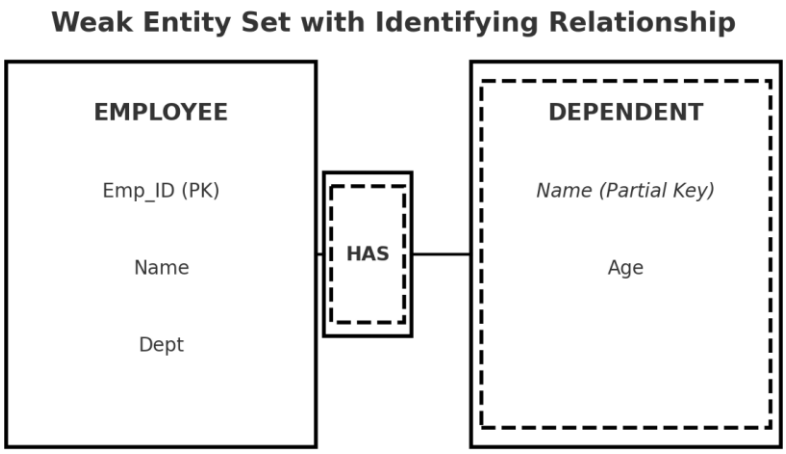
- Cannot be uniquely identified by its own attributes.
- Has a **partial key (discriminator)** that distinguishes entities **within the owner's context**.
- Existence depends on the strong entity (Existence dependency).
- Represented by a **double rectangle** in an ER diagram.
- Relationship with owner is shown by a **double diamond**.

- **Example:**
  - Entity set: **Dependent(Name, Age, Relation)**
  - Cannot uniquely identify Dependent without linking it to an Employee.
  - **Partial key:** Name (but not unique across all dependents).
  - Strong entity: Employee(Emp_ID, Name, Dept)
  - Weak entity: Dependent(Name, Age)
  - Relationship: Emp_ID + Dependent.Name uniquely identifies a dependent.
  - Rectangle → Strong Entity (Employee)
  - Double Rectangle → Weak Entity (Dependent)
  - Double Diamond → Identifying Relationship (e.g., "Has")
  - Dashed underline → Partial Key

3. Comparison Table

Feature	**Strong Entity Set**	**Weak Entity Set**
Key	Has **primary key**	No primary key, only partial key
Existence Dependency	Independent	Dependent on strong entity
ER Diagram	Single rectangle	Double rectangle
Relationship	Normal relationship	Identifying relationship (double diamond)

- Strong entity = has full primary key, independent.
- Weak entity = no primary key, relies on strong entity + partial key, dependent.



18) Explain Set Operations in Relational Algebra. 5M

Ans:

For set operations, the relations must be **union-compatible**:

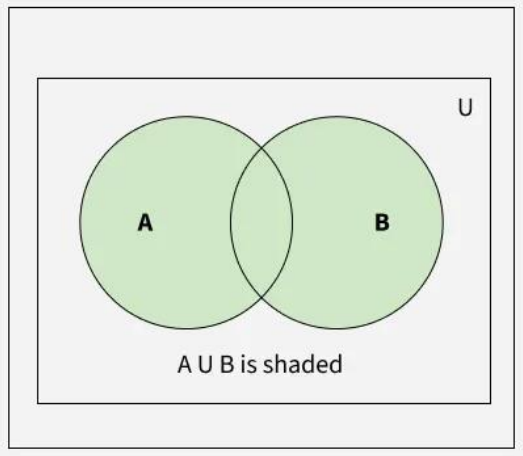
- Same number of attributes.
- Corresponding attributes have the same domain (data type).

1. Union (∪)

- **Definition:** Returns all tuples that are in **R** or **S** (or both).
- **Duplicates are removed** (since sets don't allow duplicates).
- **Notation:**

RUSR ∪ SRUS

- **Example:**
  If R = {1,2,3} and S = {3,4,5}
  → R ∪ S = {1,2,3,4,5}



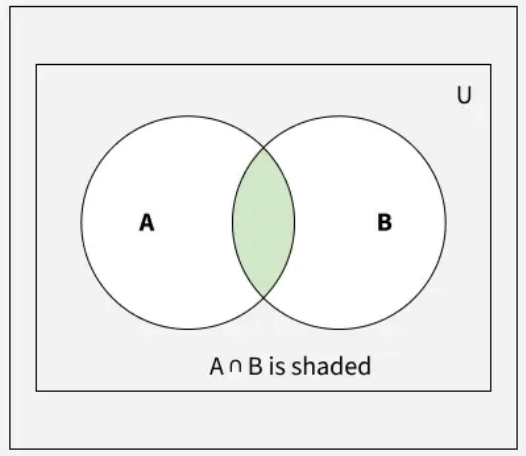
2. Intersection (∩)

- **Definition:** Returns only tuples that are present in **both R and S**.

- **Notation:**

R∩SR ∩ SR∩S

- **Example:**
  If R = {1,2,3} and S = {3,4,5}
  → R ∩ S = {3}

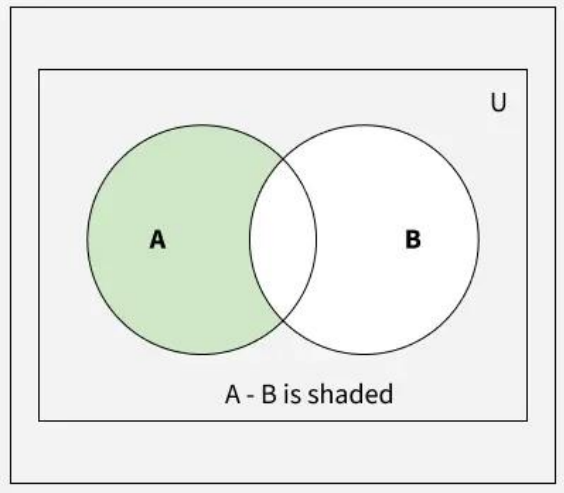


3. Difference (−)

- **Definition:** Returns tuples that are in **R but not in S**.
- **Notation:**

R−SR − SR−S

- **Example:**
  If R = {1,2,3} and S = {3,4,5}
  → R − S = {1,2}
  (Note: S − R = {4,5} is different.)

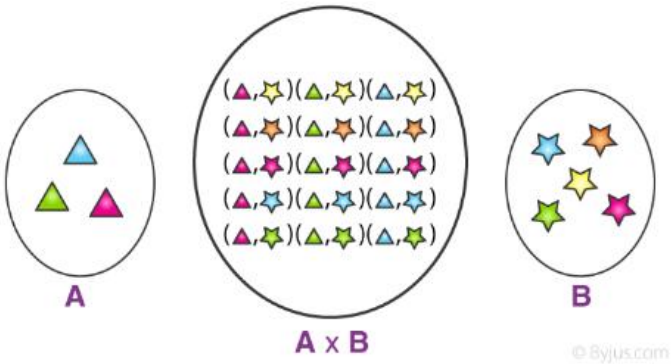


4. Cartesian Product (×)

- **Definition:** Combines each tuple of **R** with every tuple of **S**.
- **Notation:**

R×S

- **Result:** A relation with attributes from both R and S.
- **Example:**



Student

SID	SName
1	Alex
2	Bella

×

Course

CID	CName
101	DBMS
102	AI

↓

(SID, SName, CID, CName)

SID	SName	CID	CName
1	Alex	101	DBMS
1	Alex	102	AI
2	Bella	101	DBMS
2	Bella	102	AI

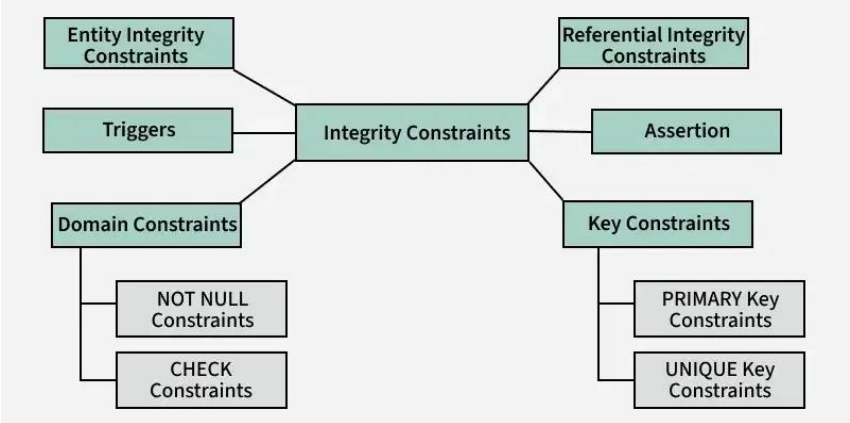
5. Rename (ρ) (not exactly a set operator, but often grouped here)

- **Definition:** Used to rename the attributes of a relation (for clarity in queries).

Operator	**Symbol**	**Meaning**
Union	∪	Tuples in R or S (all unique)
Intersection	∩	Tuples common to both R and S
Difference	−	Tuples in R but not in S
Cartesian Product	×	Combines every tuple from R with S
Rename	ρ	Renames relation or attributes

19) Explain constraints (Domain ,Key constraints, integrity constraints) and their importance.
5M

Ans:



- Maintains **accuracy**, **reliability**, and **consistency** of data.
- Prevents **invalid, duplicate, or inconsistent** entries.
- Supports **trustworthy decision-making** based on correct data.

1. Domain Constraint

- **Definition**: Specifies that the value of an attribute must come from a predefined domain (set of valid values).
- **Example**: Age must be between 0 and 120, Gender must be either 'M' or 'F'.
- **Importance**: Ensures only valid and meaningful data is stored in the database.

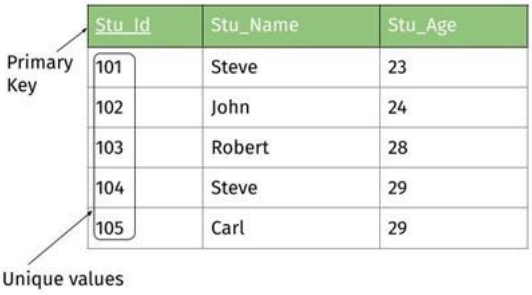
DOMAIN INTEGRITY CONSTRAINT

EmpID	Name	Age	Gender
101	Ravi	25	M
102	Priya	150	✗ Invalid
103	Ankit	30	✗ Invalid

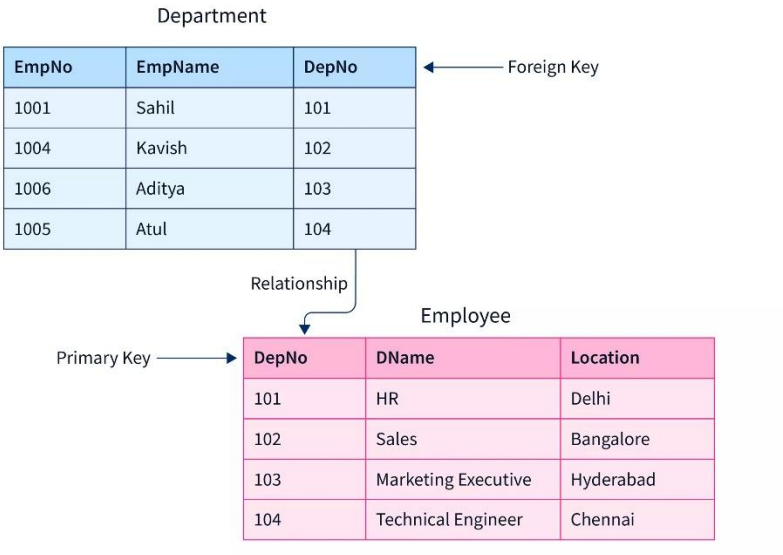
Age ✗ Invalid

2. Key Constraints

- **Definition**: Uniquely identify tuples (rows) in a relation.
- **Types**:
    - **Primary Key**: Uniquely identifies each record (cannot be NULL).



- **Unique Key**: Similar to primary key but can have NULL values.
- **Foreign Key**: Links one table to another, ensuring referential integrity.



- **Importance**: Prevents duplicate records and maintains data consistency across tables.

3. Integrity Constraints

- **Definition**: General rules that ensure correctness and consistency of data in the database.
- **Types**:
  - **Entity Integrity**: Primary key must not be NULL.
  - **Referential Integrity**: A foreign key must match a primary key in another table (or be NULL).
- **Importance**: Guarantees that relationships between tables remain consistent, preventing orphan or invalid records.

20) Illustrate Integrity Constraints (NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, DEFAULT, CHECK) over relations with examples

Ans: Integrity constraints are rules applied on database columns to maintain accuracy, consistency, and completeness of data.

1. NOT NULL Constraint

- Ensures that a column **cannot have NULL values**.
- Example: A student's name must always be present.

e.g.

```
CREATE TABLE Student (    RollNo INT,    Name VARCHAR(50) NOT NULL,    Dept VARCHAR(30),
   Percentage DECIMAL(5,2) );
```

Here, Name cannot be left empty

2. UNIQUE Constraint

- Ensures that all values in a column are **distinct** (no duplicates).
- Example: Email must be unique for each student.

e.g.,

```
CREATE TABLE Student (    RollNo INT,    Name VARCHAR(50) NOT NULL,    Email VARCHAR(50) UNIQUE,
   Dept VARCHAR(30) );
```

Here, Email must be different for every student.

3. PRIMARY KEY Constraint

- Combination of **NOT NULL + UNIQUE**.
- Each row is uniquely identified by the primary key.

e.g.,

```
CREATE TABLE Student (    RollNo INT PRIMARY KEY,    Name VARCHAR(50) NOT NULL,    Dept VARCHAR(30),
   Percentage DECIMAL(5,2));
```

Here, RollNo uniquely identifies each student and cannot be NULL.

4. FOREIGN KEY Constraint

- Links two tables using a **referential integrity rule**.
- Example: A student belongs to a department, and the department must exist in the Department table.

e.g.,

```
CREATE TABLE Department (    DeptID INT PRIMARY KEY,    DeptName VARCHAR(50) UNIQUE);
CREATE TABLE Student (    RollNo INT PRIMARY KEY,    Name VARCHAR(50) NOT NULL,    DeptID INT,
   FOREIGN KEY (DeptID) REFERENCES Department(DeptID));
```

Here, Any student’s DeptID must match an existing department in the Department table.

5. DEFAULT Constraint

- Assigns a default value when no value is provided.

e.g.,

```
CREATE TABLE Student (    RollNo INT PRIMARY KEY,    Name VARCHAR(50) NOT NULL,
   Dept VARCHAR(30) DEFAULT 'CSE',    Percentage DECIMAL(5,2));
```

If no department is given, it defaults to 'CSE'.

6. CHECK Constraint

- Ensures that values in a column satisfy a specific condition.

e.g.,

```
CREATE TABLE Student (    RollNo INT PRIMARY KEY,    Name VARCHAR(50) NOT NULL,
   Percentage DECIMAL(5,2),    CHECK (Percentage >= 0 AND Percentage <= 100));
```

Here, Percentage must always be between 0 and 100.

Final Combined Example:

```
CREATE TABLE Student (
   RollNo INT PRIMARY KEY,                  -- Primary Key
   Name VARCHAR(50) NOT NULL,               -- NOT NULL
   Email VARCHAR(50) UNIQUE,                -- UNIQUE
   DeptID INT DEFAULT 101,                  -- DEFAULT
   Percentage DECIMAL(5,2) CHECK (Percentage >= 0 AND Percentage <= 100), -- CHECK
   FOREIGN KEY (DeptID) REFERENCES Department(DeptID)  -- FOREIGN KEY);
```

21) Compare the differences between DBMS and FILE System. 8M

Ans:

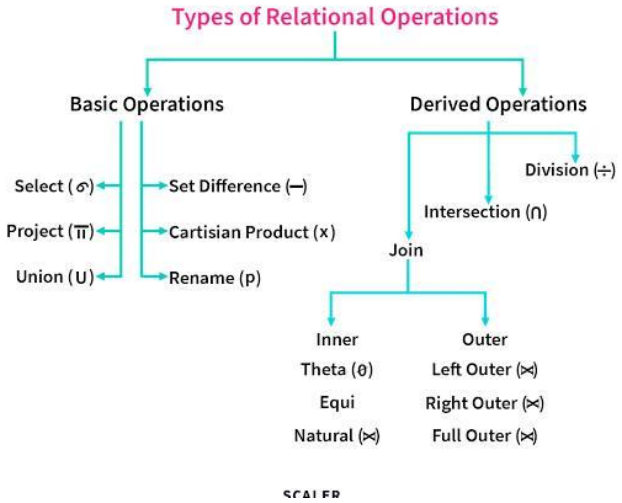
Difference Between File System and DBMS

Basis of differentiation	**File Processing System**	**DBMS**
Data redundancy & inconsistency	The problem of duplication and inconsistency in data exists in FPS.	There is no redundancy and inconsistency in data due to centralization of the database.
Ease of data access	Accessing data in file system isn't as easy as DBMS.	Accessing data is easier in DBMS as compared to a file system.
Data independence	There is no data independence in file system.	Data independence exists in DBMS.

Basis of differentiation	File Processing System	DBMS
Atomicity	Atomicity is not present in file system.	DBMS provides atomicity of transactions.
Concurrency control	It doesn't have concurrency control.	DBMS has concurrency control.
Recovery	File system doesn't provide the facility of recovery, in case of data loss.	DBMS provides the facility of data backup and recovery.
Security	Data security is less in file processing system.	DBMS offers high data security.
Cost	File system is relatively cheaper as compared to DBMS.	DBMS is costlier as compared to a file system.
Scalability	Limited scalability in file systems; can become unwieldy for large datasets.	DBMS can handle large datasets and is scalable to accommodate growing data needs.
Query Language	File systems lack a standardized query language, making complex queries challenging.	DBMS offers SQL (Structured Query Language) for efficient and complex data queries.
Multi-User Support	File systems often lack built-in support for multiple users accessing data simultaneously.	DBMS provides robust support for concurrent access by multiple users.

22) What are the standard operations available in relational algebra? Explain with suitable examples? 10M

Ans:



Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output.

It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output.

The fundamental operations of relational algebra are as follows −

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation − $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −

$\sigma_{subject = "database"}(Books)$

Output − Selects tuples from books where subject is 'database'.

$\sigma_{subject = "database" \text{ and } price = "450"}(Books)$

Output − Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}(Books)$

Output − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1, A_2, A_n}(r)$

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example −

$\prod_{subject, author}(Books)$

Selects and projects columns named as subject and author from the relation Books.

Union Operation (∪)

It performs binary union between two given relations and is defined as −

r ∪ s = { t | t ∈ r or t ∈ s}

Notation − r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)

Output − Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − **r** − **s**

Finds all the tuples that are present in **r** but not in **s**.

$\prod_{author}$ (Books) − $\prod_{author}$ (Articles)

Output − Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

$\sigma_{author = 'tutorialspoint'}$(Books X Articles)

Output − Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

Notation − ρ_x (E)

Where the result of expression **E** is saved with name of **x**.

Additional operations are –

- Set intersection
- Assignment
- Natural join

23) Classify DDL, DML commands in SQL with example for each. 7M

Ans: Refer Q .No 14 Page No: 34

24) Discuss Data Types in SQL. 5M

Ans:

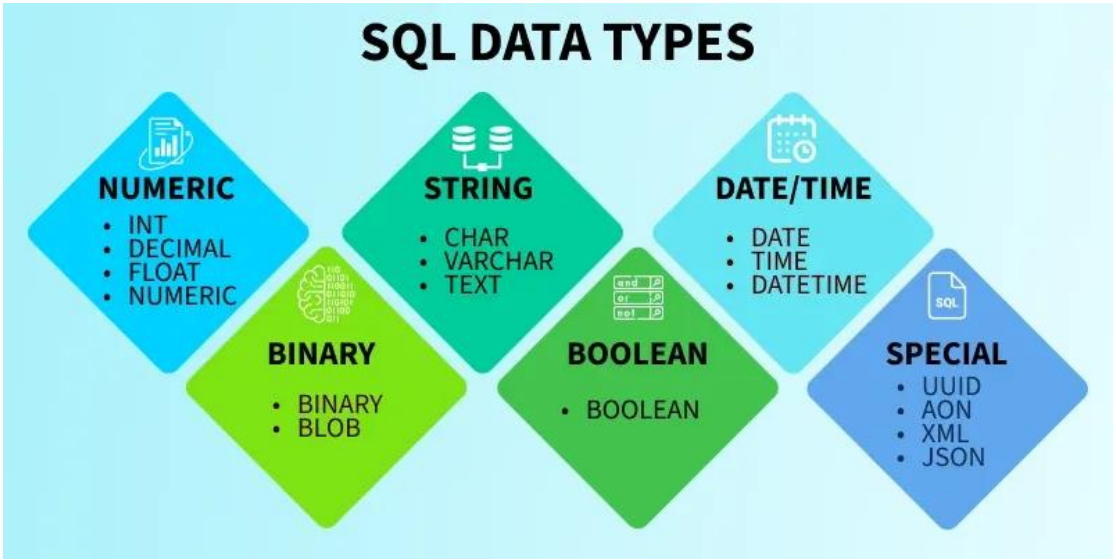
In SQL, each column must be assigned a data type that defines the kind of data it can store, such as integers, dates, text, or binary values.

Choosing the correct data type is crucial for data integrity, query performance and efficient indexing.

Benefits of using the right data type:

- Memory-efficient storage
- Accurate operations (e.g., calculations, sorting)
- Consistency in stored values
- Validation of input data

SQL data types are broadly categorized into several groups:



1. Numeric Data Types

Numeric data types are fundamental to database design and are used to store numbers, whether they are integers, decimals or floating-point numbers.

These data types allow for mathematical operations like addition, subtraction, multiplication and division, which makes them essential for managing data.

Exact Numeric Datatype

Exact numeric types are used when precise numeric values are needed, such as for financial data, quantities, and counts. Some common exact numeric types include:

Data Type	Description	Range
BIGINT	Large integer numbers	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
INT	Standard integer values	-2,147,483,648 to 2,147,483,647
SMALLINT	Small integers	-32,768 to 32,767
TINYINT	Very small integers	0 to 255
DECIMAL	Exact fixed-point numbers (e.g., for financial values)	-10^38 + 1 to 10^38 - 1
NUMERIC	Similar to DECIMAL, used for precision data	-10^38 + 1 to 10^38 - 1
MONEY	For storing monetary values	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
SMALLMONEY	Smaller monetary values	-214,748.3648 to 214,748.3647

Approximate Numeric Datatype

These types are used to store approximate values, such as scientific measurements or large ranges of data that don't need exact precision.

Data Type	Description	Range
FLOAT	Approximate numeric values	-1.79E+308 to 1.79E+308
REAL	Similar to FLOAT, but with less precision	-3.40E+38 to 3.40E+38

2. Character and String Data Types

Character data types are used to store text or character-based data. The choice between fixed-length and variable-length data types depends on the nature of your data.

Data Type	Description
Char	The maximum length of 8000 characters. (Fixed-Length non-Unicode Characters)
Varchar	The maximum length of 8000 characters. (Variable-Length non-Unicode Characters)
Varchar(max)	The maximum length of **2^31 - 1** characters(SQL Server 2005 only). (Variable Length non-Unicode data)
Text	The maximum length of 2,127,483,647 characters(Variable Length non-Unicode data)

Unicode Character String Data Types

Unicode data types are used to store characters from any language, supporting a wider variety of characters. These are given in below table.

Data Type	Description
Nchar	The maximum length of 4000 characters (Fixed-Length Unicode Characters)
Nvarchar	The maximum length of 4000 characters.(Variable-Length Unicode Characters)
Nvarchar(max)	The maximum length of 2^31 - 1 characters(SQL Server 2005 only). (Variable Length Unicode data)

3. Date and Time Data Type

SQL provides several data types for storing date and time information. They are essential for managing timestamps, events and time-based queries. These are given in the below table.

Data Type	Description	Storage Size
DATE	stores the data of date (year, month, day)	3 Bytes

Data Type	Description	Storage Size
TIME	stores the data of time (hour, minute,second)	3 Bytes
DATETIME	store both the data and time (year, month, day, hour, minute, second)	8 Bytes

4. Binary Data Types in SQL

Binary data types are used to store binary data such as images, videos or other file types. These include:

Data Type	Description	Max Length
Binary	Fixed-length binary data.	8000 bytes
VarBinary	Variable-length binary data.	8000 bytes
Image	Stores binary data as images.	2,147,483,647 bytes

5. Boolean Data Type in SQL

The BOOLEAN data types are used to store logical values, typically TRUE or FALSE. It is commonly used for flag fields or binary conditions.

6. Special Data Types

SQL also supports some specialized data types for advanced use cases:

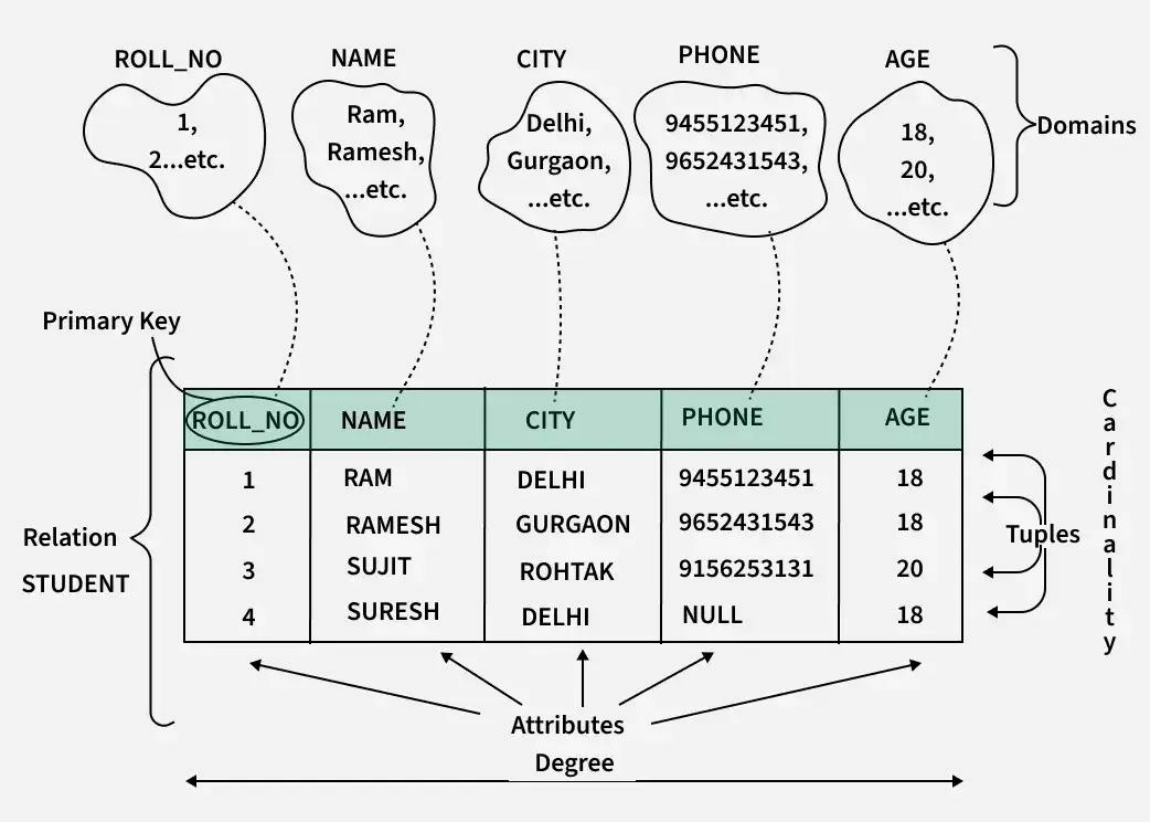
- **XML Data Type:** Used to store XML data and manipulate XML structures in the database
- **Spatial Data Type (Geometry):** stores planar spatial data, such as points, lines, and polygons, in a database table.

25) Discuss the concepts of domain, attribute, tuple, relation, NULL values and constraints 5M

Ans:

1. Domain

- **Definition:** A domain is the set of all possible values that an attribute (column) can take.
- **Example:** For an attribute Age, the domain could be all integers from 1 to 100.
- **Importance:** Domains ensure **data integrity** by restricting values to a valid set



2. Attribute

- **Definition:** An attribute is a **column in a table** representing a property of an entity.
- **Example:** In a table Student(roll_no, name, age, department), the attributes are roll_no, name, age, and department.
- **Characteristics:** Each attribute has a **name** and a **domain**.

3. Tuple

- **Definition:** A tuple is a **row in a table**, representing a single record or instance of an entity.
- **Example:** (101, 'Alice', 20, 'CSE') is a tuple in the Student table.
- **Importance:** Tuples store actual **data values** in the database.

4. Relation

- **Definition:** A relation is a **table** consisting of tuples (rows) and attributes (columns).
- **Characteristics:**
  - Each row (tuple) is unique.
  - Each column (attribute) has a specific domain.
  - The order of tuples is irrelevant, but the order of attributes matters.
- **Example:**

roll_no	name	age	department
101	Alice	20	CSE
102	Bob	21	ECE

This table is a **relation**.

5. NULL Values

- **Definition:** NULL represents **missing or unknown data** in a table.
- **Example:** If a student's age is not known: (103, 'Charlie', NULL, 'ME').
- **Importance:**
    - Distinguishes between **no value** and **zero or empty string**.
    - Important in queries and constraints (e.g., NOT NULL constraint).

6. Constraints

- **Definition:** Constraints are **rules enforced on the data** to maintain **accuracy and integrity**.
- **Types of Constraints:**
    1. **Domain Constraint:** Ensures attribute values are from a valid domain.
        - Example: age must be between 1 and 100.
    2. **Primary Key Constraint:** Ensures each row is uniquely identifiable.
        - Example: roll_no in Student table.
    3. **Unique Constraint:** Ensures all values in an attribute are unique.
    4. **Foreign Key Constraint:** Ensures referential integrity between tables.
        - Example: dept_id in Student references Department(dept_id).
    5. **Not Null Constraint:** Attribute cannot have NULL values.
    6. **Check Constraint:** Ensures values satisfy a condition.
        - Example: CHECK(age > 0).

Concept	Definition/Role	Example
Domain	Set of valid values for an attribute	Age: 1–100

Concept	Definition/Role	Example
Attribute	Column/property of a table	name, age
Tuple	Row/record in a table	(101, 'Alice', 20, 'CSE')
Relation	Table consisting of tuples and attributes	Student table
NULL Value	Unknown or missing value	(103, 'Charlie', NULL, 'ME')
Constraint	Rule for maintaining integrity	PRIMARY KEY(roll_no)

26) Illustrate Three-tier Schema Architecture for Data Independence. How are these different schema layers related to the concepts of logical and physical data independence? 10M

Ans: Refer Q.NO 13 Page No 33

27) Discuss in detail about importance of NULL values. 3M

Ans:

Importance of NULL Values

1. Represents Missing Information – NULL allows storing incomplete data without forcing invalid values.

2. Avoids Wrong Assumptions – 0 salary vs NULL salary are different meanings.

3. Ensures Data Accuracy & Integrity – NULL indicates 'unknown' instead of forcing dummy values.

5. Helps in Queries with Aggregate Functions – COUNT, AVG, SUM ignore NULL values, preventing misleading results.

Example (SQL):

SELECT AVG(Salary) FROM Employee;

This query considers only rows where Salary is NOT NULL.

Problems with NULL (Special Handling Required)

• Comparisons with NULL do not work normally: NULL = NULL → FALSE.

• Must use IS NULL or IS NOT NULL.

Example :

*SELECT * FROM Employee WHERE Salary IS NULL;*

• Logical expressions with NULL give UNKNOWN (e.g., 5 < NULL → UNKNOWN).

But it requires special handling in SQL queries using IS NULL or IS NOT NULL.

28) Create a table Employee (eno, ename, dno) insert 5 rows into the table and use alter,update,delete commands on the above table. 7M

Ans:

```
-- 1. Create Table
CREATE TABLE Employee (
    eno INT PRIMARY KEY,     -- Employee Number
    ename VARCHAR(50),       -- Employee Name
    dno INT                  -- Department Number
);
-- 2. Insert 5 rows
INSERT INTO Employee VALUES (101, 'Ravi', 10);
INSERT INTO Employee VALUES (102, 'Neha', 20);
INSERT INTO Employee VALUES (103, 'Amit', 30);
INSERT INTO Employee VALUES (104, 'Priya', 20);
INSERT INTO Employee VALUES (105, 'Kiran', 10);

-- 3. ALTER Command (Add a new column for Salary)
ALTER TABLE Employee ADD salary DECIMAL(10,2);

-- 4. UPDATE Command (Update salary of an employee)
UPDATE Employee
SET salary = 50000
WHERE eno = 101;

-- Another update: Change department of 'Priya'
UPDATE Employee
SET dno = 40
WHERE ename = 'Priya';
```

-- 5. DELETE Command (Delete one employee record)

DELETE FROM Employee

WHERE eno = 105;

Explanation:

1. **CREATE** – defines the table with attributes eno, ename, dno.
2. **INSERT** – adds 5 employee records.
3. **ALTER** – modifies structure of table (adds new column salary).
4. **UPDATE** – modifies existing data (salary of Ravi, department of Priya).
5. **DELETE** – removes a row (employee Kiran)

29) Explain about Domain key constraints, Integrity key constraints and Referential constraints with suitable examples.

1. Domain Constraints

Definition:
Domain constraint specifies that the value of an attribute must come from a valid domain (data type, range, format, or set of values). Every attribute in a relation is associated with a domain.

Example :

```
CREATE TABLE Student (
RollNo INT NOT NULL,
Name VARCHAR(50) NOT NULL,
Age INT CHECK (Age >= 17),
Dept VARCHAR(10) CHECK (Dept IN ('CSE', 'ECE', 'MECH', 'CIVIL'))
);
```

Explanation:
- Age must be greater than or equal to 17.
- Dept must be one of the listed values.
If someone tries: INSERT INTO Student VALUES (101, 'Raj', 15, 'BIO'); → Rejected.

2. Key Constraints (Integrity Constraints)

Definition:
Key constraints ensure that a tuple (row) in a relation can be uniquely identified using keys (Primary Key, Unique Key). No two tuples can have the same value for the primary key attribute.

Example :

```
CREATE TABLE Employee (
EmpID INT PRIMARY KEY,
Name VARCHAR(50),
```

Salary DECIMAL(10,2)
);

Explanation:
- EmpID is the primary key, so it must be unique and not null.
Duplicate insertion with same EmpID will cause an error.

3. Referential Integrity Constraints

Definition:
A referential constraint (foreign key) ensures that a value in one relation must exist in another relation. This maintains consistency across tables.

Example :

CREATE TABLE Department (
DeptID INT PRIMARY KEY,
DeptName VARCHAR(50));

CREATE TABLE Teacher (
TID INT PRIMARY KEY,
TName VARCHAR(50),
DeptID INT,
FOREIGN KEY (DeptID) REFERENCES Department(DeptID));

Explanation:
- DeptID in Teacher must exist in Department.
For example:
INSERT INTO Department VALUES (10, 'CSE'); → Valid
INSERT INTO Teacher VALUES (101, 'Neha', 10); → Valid
INSERT INTO Teacher VALUES (102, 'Raj', 20); → Error (DeptID 20 not found)

Summary :

Constraint Type	Purpose	Example
Domain Constraint	Values must come from a valid domain (data type, range, format).	Age INT CHECK(Age >= 17)
Key Constraint (Integrity)	Uniquely identifies tuples, avoids duplicates.	EmpID INT PRIMARY KEY
Referential Constraint	Foreign key ensures referenced value exists in another table.	FOREIGN KEY (DeptID) REFERENCES Department(DeptID)