A **database** is an organized collection of data that is stored and accessed electronically. It allows for efficient data retrieval, management, and manipulation using a database management system (DBMS).

DBMS (Database Management System) is software that is used to create, manage, and manipulate databases. It provides tools for storing data, retrieving it, updating records, and ensuring data security and integrity. Examples include MySQL, Oracle, and SQL Server.

Types of Databases (2-mark answer):

1. **Relational Database** – Stores data in tables with rows and columns (e.g., MySQL, Oracle).
2. **NoSQL Database** – Handles unstructured data like documents or key-value pairs (e.g., MongoDB).

SQL (Structured Query Language) is a standard programming language used to manage and manipulate relational databases. It is used for tasks like creating tables, inserting data, updating records, and retrieving information using queries.

Types of SQL Commands (2-mark answer):

1. **DDL (Data Definition Language)** – Defines structure (e.g., CREATE, ALTER, DROP)
2. **DML (Data Manipulation Language)** – Manages data (e.g., INSERT, UPDATE, DELETE)
3. **DQL (Data Query Language)** – Queries data (e.g., SELECT)
4. **DCL (Data Control Language)** – Controls access (e.g., GRANT, REVOKE)
5. **TCL (Transaction Control Language)** – Manages transactions (e.g., COMMIT, ROLLBACK, SAVEPOINT)

DDL is used to define and modify database structures (e.g., `CREATE`, `DROP`) and changes are auto-committed.

DML is used to manipulate data within tables (e.g., `INSERT`, `UPDATE`) and changes need to be committed manually.

Keys are attributes or sets of attributes that help identify records uniquely in a table and establish relationships between tables.

1. **Primary Key**: A column or set of columns that uniquely identifies each row in a table and cannot have null values.
2. **Foreign Key**: A column that creates a relationship between two tables by referring to the primary key of another table.
3. **Candidate Key**: Any column or combination of columns that can uniquely identify rows; one of them becomes the primary key.
4. **Alternate Key**: A candidate key that was not selected as the primary key.
5. **Composite Key**: A key made from two or more columns that together uniquely identify a row.

6. **Unique Key**: Ensures all values in a column are different; unlike primary key, it allows one null value.
7. A **Super Key** is a **set of one or more attributes** (columns) that can **uniquely identify** each record (row) in a table.

Constraints in SQL are rules applied to table columns to enforce data integrity, consistency, and accuracy in the database.

NOT NULL: Ensures a column cannot have a null value.

UNIQUE: Ensures all values in a column are unique (no duplicates).

PRIMARY KEY: Combines NOT NULL and UNIQUE; uniquely identifies each row.

FOREIGN KEY: Maintains referential integrity by linking to the primary key of another table.

CHECK: Ensures values in a column meet a specific condition.

DEFAULT: Assigns a default value to a column if no value is provided.

Schema in DBMS is the structure or blueprint of a database. It defines how data is organized, including tables, fields, data types, relationships, views, indexes, and constraints. A schema does not store data itself but describes the logical design of the entire database.

Cardinality of Relations refers to the number of **tuples (rows)** present in a relation (table) in a database.

Types of Cardinality:

1. One-to-One (1:1)
2. One-to-Many (1:N)
3. Many-to-One (N:1)
4. Many-to-Many (M:N)

JOINS in SQL are used to combine rows from two or more tables based on a related column between them.

1. **INNER JOIN** – Returns only matching rows from both tables.
2. **LEFT JOIN** – Returns all rows from the left table and matching rows from the right.
3. **RIGHT JOIN** – Returns all rows from the right table and matching rows from the left.
4. **FULL JOIN** – Returns all rows when there is a match in one of the tables.
5. **CROSS JOIN** – Returns the Cartesian product of both tables (all combinations).
6. **SELF JOIN** – Joins a table to itself to compare rows within the same table.
7. **Natural Join** in SQL is a type of join that automatically joins two tables based on all columns with the same names and compatible data types in both tables. It **removes duplicate columns** in the result and matches rows with equal values in those common columns.

Set Operations in SQL are used to combine the results of two or more SELECT queries.

Types of Set Operations (with short explanation):

1. **UNION** – Combines results of two queries and removes duplicates.
2. **UNION ALL** – Combines results and includes duplicates.
3. **INTERSECT** – Returns only the common records from both queries.
4. **MINUS** (or **EXCEPT** in some DBMS) – Returns records from the first query that are not in the second.

Subqueries in SQL are queries nested inside another query, usually within SELECT, INSERT, UPDATE, or DELETE.

Types of Subqueries (with short explanation):

1. **Single-row Subquery** – Returns only one row.
2. **Multiple-row Subquery** – Returns multiple rows.
3. **Correlated Subquery** – Depends on the outer query for its value.
4. **Nested Subquery** – Subquery inside another subquery.
5. **Scalar Subquery** – Returns a single value.

The **SELECT** statement is used to **retrieve data** from one or more tables in a database.

The **WHERE** clause is used to **filter records** that meet a specific condition.

The **ORDER BY** clause is used to **sort the result** of a query by one or more columns

Aggregate functions perform calculations on multiple rows and return a single value.

Common Aggregate Functions:

1. **COUNT()** – Returns the number of rows.
2. **SUM()** – Returns the total sum of a numeric column.
3. **AVG()** – Returns the average value of a numeric column.
4. **MIN()** – Returns the smallest value in a column.
5. **MAX()** – Returns the largest value in a column.

The **GROUP BY** clause is used to **group rows** that have the same values in specified columns and is often used with **aggregate functions** like COUNT(), SUM(), AVG(), etc.

SQL operators are used to perform operations in queries, such as comparisons and arithmetic.

Logical Operators
AND, OR, NOT
Used to combine multiple conditions.

Special Operators

- **BETWEEN**: Checks if a value is within a range.

- **IN:** Checks if a value is in a list.
- **LIKE**: Matches a pattern.
- **IS NULL**: Checks for null values.

The **DISTINCT** keyword is used to **remove duplicate values** from the result set of a SELECT query.

The **LIKE** operator is used in a WHERE clause to **search for a specified pattern** in a column.

A **View** is a **virtual table** based on the result of a SQL SELECT query. It does not store data physically but displays data from one or more tables.

The **GRANT** command is used to **give specific privileges** to users on database objects like tables, views, or procedures.

The **REVOKE** command is used to **take back privileges** that were previously granted to a user.

The **COMMIT** command is used to **save all changes** made by the current transaction **permanently** in the database.

The **ROLLBACK** command is used to **undo changes** made in the current transaction before a COMMIT.

The **SAVEPOINT** command is used to **set a point within a transaction** to which you can later roll back without affecting the entire transaction.

ALTER is a Data Definition Language (DDL) command used to **modify the structure** of a database table. It can add, delete, or change columns, data types, or constraints in an existing table. For example, you can use ALTER TABLE Students ADD age INT; to add a new column.

UPDATE is a Data Manipulation Language (DML) command used to **modify existing data** in a table. It changes the values of one or more columns for selected rows based on a condition. For example, UPDATE Students SET age = 20 WHERE id = 1; changes the age of the student with ID 1.

DELETE is a DML (Data Manipulation Language) command used to **remove specific rows** from a table based on a condition. It **can be rolled back** if used within a transaction and does not remove the table structure.

TRUNCATE is a DDL (Data Definition Language) command used to **remove all rows** from a table quickly without logging individual row deletions. It **cannot be rolled back** in most databases and retains the table structure.

DROP is also a DDL command that **completely deletes the table** from the database, including its structure and data. It **cannot be rolled back**.

In short:

- DELETE → removes specific rows
- TRUNCATE → removes all rows
- DROP → removes the entire table

A **Database Administrator (DBA)** is a person responsible for managing, maintaining, and securing a database system. The DBA ensures that the database is always available, performs efficiently, and is protected from unauthorized access or data loss.

The **ER Model** is a high-level data model used to **visually represent** the structure of a database. It helps in designing a database by identifying **entities**, their **attributes**, and **relationships** among them.

An **ER Diagram** is a graphical representation of the **Entity-Relationship (ER) model**, used to design and model a database structure.

An **Entity** is a **real-world object** or **concept** that can be distinctly identified and stored in a database.

Attributes are the **columns** or **fields** in a database table that define the **properties** of an entity.

A **Strong Entity** is an entity that has a **primary key** and can exist **independently** in a database. It does not rely on any other entity for its identification.

A **Weak Entity** cannot be uniquely identified by its own attributes alone and depends on a **related strong entity** for identification. It usually has a **partial key** and is connected to a strong entity through a **identifying relationship**.

Anomalies are problems that can occur in a database when it is not properly **normalized**. They lead to **data inconsistency, redundancy**, and **loss of data integrity**.

1. **Insertion Anomaly** – Occurs when you can't insert data into a table without including unrelated data.
   Example: You can't add a new student without assigning them a course.
2. **Update Anomaly** – Happens when the same data is stored in multiple places and all instances are not updated.
   Example: Changing a student's address in one row but not in another causes inconsistency.
3. **Deletion Anomaly** – Occurs when deleting data unintentionally removes useful information.
   Example: Deleting the last course a student is enrolled in also removes the student record.

These issues are usually resolved by **normalizing** the database into smaller, related tables.

Normalization is the process of organizing data in a database to **reduce redundancy** and **improve data integrity**. It involves dividing large, complex tables into smaller, related tables and defining relationships between them.

Normalization is done in stages called **normal forms (NF)** such as:

- **1NF (First Normal Form)** – Removes repeating groups; ensures atomicity.
- **2NF (Second Normal Form)** – Removes partial dependencies.
- **3NF (Third Normal Form)** – Removes transitive dependencies.

BCNF is an advanced version of the **Third Normal Form (3NF)**. A table is in BCNF if:

For every functional dependency A → B, A should be a super key.

Functional Dependency (FD) is a relationship between two attributes, typically between a **key** and a **non-key** attribute, in a **relation (table)**.

It is denoted as: **A → B**

Which means:
If two rows have the same value for attribute A, they must also have the same value for attribute B.
So, **A functionally determines B**.

A **Data Dictionary** is a **central repository** that stores **metadata**—that is, information **about the data** in a database.

Metadata is **data about data**. It describes the structure, organization, and properties of the actual data stored in a database.

A **Trigger** is a special kind of stored procedure that **automatically executes** in response to certain events on a table, such as INSERT, UPDATE, or DELETE.

Closure of a set of attributes is the **complete set of attributes** that can be functionally determined from it using given functional dependencies.

A **Minimal Cover** is a simplified set of functional dependencies that is **equivalent** to the original set but has:

- **Single attribute on the right side** of each dependency.
- No **redundant attributes** on the left side.
- No **redundant dependencies**.

A **Partial Functional Dependency** occurs when a **non-prime attribute** is functionally dependent on **part of a composite primary key** (not the whole key).

An **Index** is a database structure that **improves the speed** of data retrieval operations on a table at the cost of additional storage and slower writes.

Types of Indexes in DBMS

1. **Single-Level Index**
   - A simple index with one level of indexing.
2. **Multi-Level Index**
   - Indexes on indexes for faster searching in large tables.
3. **Clustered Index**

- Sorts and stores data rows in the table based on the indexed column. Only one clustered index per table.

A **B-Tree** is a self-balancing tree data structure used to **store sorted data** and allow **efficient insertion, deletion, and search operations**.

B+ Tree is a balanced tree where all actual data is in leaf nodes, making range searches and sequential access faster and efficient.

ACID properties make sure database transactions are **safe, consistent, and reliable**.

Atomicity: Ensures that a transaction is **all or nothing** — either all operations succeed or none do.

Consistency: Guarantees the database moves from one **valid state to another**, maintaining all rules and constraints.

Isolation: Ensures concurrent transactions do not interfere; intermediate states of a transaction are invisible to others.

Durability: Once a transaction commits, its changes are **permanently saved**, even if the system crashes.