Q1. Types of Databases:

- **Relational Databases (RDBMS):** Organize data into tables with rows and columns (e.g., MySQL, PostgreSQL).
- **NoSQL Databases:** Store data in formats other than tables, such as key-value pairs, documents, or graphs (e.g., MongoDB, Cassandra).
- **Object-Oriented Databases:** Store data as objects, similar to object-oriented programming (e.g., ObjectDB).
- **Distributed Databases:** Data is stored across multiple physical locations (e.g., Google Spanner).
- **In-Memory Databases:** Store data in the main memory rather than on disk for faster access (e.g., Redis).
- **Graph Databases:** Focus on the relationships between data, using nodes, edges, and properties (e.g., Neo4j).

Q2. Difference between DBMS and RDBMS:

- **DBMS (Database Management System):** A software that manages databases, allowing storage, retrieval, and updating of data (e.g., Microsoft Access).
- **RDBMS (Relational Database Management System):** A type of DBMS that uses a relational model to store data in tables with rows and columns, supporting ACID properties (e.g., MySQL, Oracle).

Q3. Normalization:

- **Concept:** The process of organizing data to reduce redundancy and improve data integrity.
- **Types:**
    - **1NF (First Normal Form):** Ensures that each column contains atomic (indivisible) values and each record is unique.
    - **2NF (Second Normal Form):** Removes partial dependencies; a table must be in 1NF and all non-key attributes must depend on the entire primary key.
    - **3NF (Third Normal Form):** Removes transitive dependencies; a table must be in 2NF, and no non-key attribute should depend on another non-key attribute.
    - **BCNF (Boyce-Codd Normal Form):** A stricter version of 3NF where every determinant must be a candidate key.
    - **4NF (Fourth Normal Form):** Removes multi-valued dependencies.
    - **5NF (Fifth Normal Form):** Ensures that a table is free from any join dependency.

Q4. Primary Key:

- A unique identifier for each record in a table. It is important because it ensures that each record can be uniquely identified, which is essential for maintaining data integrity and establishing relationships between tables.

```
CREATE TABLE Employees (
  EmployeeID INT PRIMARY KEY,
  Name VARCHAR(100)
);
```

Q5. Foreign Key:

- A field (or collection of fields) in one table that uniquely identifies a row of another table, establishing a link between the two tables. It ensures referential integrity by enforcing a relationship between the foreign key and the primary key of the referenced table.

```
CREATE TABLE Orders (
  OrderID INT PRIMARY KEY,
  CustomerID INT,
  FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Q6. Difference between `DELETE`, `TRUNCATE`, and `DROP`:

- **DELETE:** Removes rows from a table based on a condition. It can be rolled back (transactional).
- **TRUNCATE:** Removes all rows from a table, but the table structure remains. It cannot be rolled back in most DBMS.
- **DROP:** Completely removes a table, including its structure and data.

```
DELETE FROM Employees WHERE EmployeeID = 1; -- Deletes specific rows
TRUNCATE TABLE Employees; -- Deletes all rows but keeps structure
DROP TABLE Employees; -- Removes the table entirely
```

Q7. Types of Joins in SQL:

- **INNER JOIN:** Returns records with matching values in both tables.

- **LEFT JOIN (LEFT OUTER JOIN):** Returns all records from the left table and matched records from the right table. If no match, returns NULL from the right side.
- **RIGHT JOIN (RIGHT OUTER JOIN):** Returns all records from the right table and matched records from the left table. If no match, returns NULL from the left side.
- **FULL JOIN (FULL OUTER JOIN):** Returns records when there is a match in either left or right table.
- **CROSS JOIN:** Returns the Cartesian product of two tables (all possible combinations of rows).
- **SELF JOIN:** A table is joined with itself.

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
-- INNER JOIN

SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
LEFT JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
 -- LEFT JOIN

SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
FULL OUTER JOIN Customers ON Orders.CustomerID = Customers.CustomerID; --
FULL JOIN
```

Q8. Difference between Clustered and Non-Clustered Indexes:

- **Clustered Index:** Sorts and stores the data rows of the table based on the indexed column(s). There can be only one clustered index per table because it defines the physical order of data.
- **Non-Clustered Index:** Contains a sorted list of references to the table data without altering the physical order of the rows. A table can have multiple non-clustered indexes.

Q9. ACID Properties:

- **Atomicity:** Ensures that a transaction is fully completed or not done at all.
- **Consistency:** Guarantees that a transaction takes the database from one valid state to another.
- **Isolation:** Ensures that transactions are executed independently, without interference.
- **Durability:** Once a transaction is committed, it remains in the system, even in case of a failure.

Q10. Transaction:

- A sequence of one or more SQL operations treated as a single unit of work. Transactions ensure data integrity and consistency, and they follow ACID properties.

Q11. Stored Procedure:

- A precompiled collection of SQL statements stored in the database that can be executed as a unit. It is used to encapsulate complex business logic, improve performance, and enhance security.

Q12. Triggers:

- Special stored procedures that automatically execute in response to certain events on a table or view, such as `INSERT`, `UPDATE`, or `DELETE`. They are used to enforce business rules, maintain audit trails, and replicate data.

```
CREATE TRIGGER trgAfterInsert ON Employees
AFTER INSERT
AS
BEGIN
  INSERT INTO EmployeeLog (EmployeeID, LogDate)
  SELECT EmployeeID, GETDATE() FROM inserted;
END;
```

Q13: Difference between `HAVING` and `WHERE` Clauses:

- **WHERE:** Filters rows before any groupings are made.
- **HAVING:** Filters groups after the `GROUP BY` clause has been applied.

```
SELECT Department, COUNT(*) AS EmployeeCount
FROM Employees
WHERE Age > 30 -- Filter before grouping
GROUP BY Department
HAVING COUNT(*) > 5; -- Filter after grouping
```

Q14. Advantages of Using Views:

- **Simplify complex queries:** Encapsulate complex logic in a simple, reusable interface.
- **Security:** Restrict access to specific data by allowing users to query a view instead of the underlying tables.
- **Data abstraction:** Provide a customized presentation of the data.

```
CREATE VIEW EmployeeView AS
SELECT EmployeeID, Name, Department
FROM Employees
WHERE Age > 30;
```

Q15. Difference between `INNER JOIN` and `OUTER JOIN`:

- **INNER JOIN:** Returns only the rows that have matching values in both tables.
- **OUTER JOIN:** Returns all rows from one table and the matched rows from the other table. If no match exists, `NULL` is returned for unmatched rows.

Q16. Referential Integrity:

- Ensures that relationships between tables remain consistent. When a foreign key in one table refers to a primary key in another, referential integrity ensures that the referenced key exists or that the foreign key is set to `NULL` (if allowed).

Q17. Various Constraints in a Relational Database:

- **Primary Key:** Uniquely identifies each record in a table.
- **Foreign Key:** Enforces a relationship between two tables.
- **Unique:** Ensures that all values in a column are unique.
- **Not Null:** Ensures that a column cannot have a `NULL` value.
- **Check:** Enforces a condition that must be true for all values in a column.
- **Default:** Assigns a default value if no value is provided for a column.

Q18. Implementing a Many-to-Many Relationship:

- Achieved using a junction table (also known as a linking or bridge table) that holds foreign keys referencing the primary keys of the two related tables.

Q19. ER (Entity-Relationship) Diagram:

- A visual representation of the entities in a database and their relationships. It is used in the design phase of database development to map out the structure and relationships.

Q20. Difference between `UNION` and `UNION ALL`:

- **UNION:** Combines the results of two or more `SELECT` queries and removes duplicate rows.
- **UNION ALL:** Combines the results of two or more `SELECT` queries without removing duplicates.

```
SELECT Name FROM Customers WHERE Country = 'USA'
UNION
SELECT Name FROM Suppliers WHERE Country = 'USA'; -- UNION (removes duplicates)

SELECT Name FROM Customers WHERE Country = 'USA'
UNION ALL
SELECT Name FROM Suppliers WHERE Country = 'USA'; -- UNION ALL (includes duplicates)
```

Q21. Indexing and Its Types:

- **Indexing:** A technique used to optimize database performance by reducing the time taken to retrieve data.
- **Types:**
  - **Clustered Index:** Alters the physical order of the table data.
  - **Non-Clustered Index:** Creates a separate structure for the index that refers to the table data.
  - **Composite Index:** An index on multiple columns.
  - **Unique Index:** Ensures that the indexed column(s) contain unique values.

Q22. Differences between OLTP and OLAP Systems:

- **OLTP (Online Transaction Processing):** Focuses on managing transaction-oriented applications, typically involving a large number of short online transactions (e.g., banking systems).
- **OLAP (Online Analytical Processing):** Designed for query and analysis rather than transaction processing, involving complex queries that aggregate data (e.g., data warehousing).

Q23. GROUP BY Clause:

- Used to group rows that have the same values in specified columns into summary rows, such as counting the number of rows in each group.

```
SELECT Department, COUNT(*) AS EmployeeCount
FROM Employees
GROUP BY Department;
```

Q24. Deadlock in DBMS:

- A situation where two or more transactions are waiting for each other to release resources, leading to a standstill. It can be avoided by using timeout, deadlock detection, or ordering resources.

```
SET DEADLOCK_PRIORITY LOW;
```

Q25. Database Sharding:

- A method of partitioning data into smaller, more manageable pieces called shards, which are distributed across multiple servers. It improves scalability and performance by distributing the load and enabling parallel processing.