Q-1) What do you understand by database?

->> A database is an organized collection of structured data that can be easily accessed, managed, and updated.

->> It is designed to store, retrieve, and manipulate data efficiently.

->> Databases are essential for managing large amounts of information in various systems like websites, applications, and services.

->> Key concepts related to databases are includes:

- **Tables**: data in relational databases is stored in tables, which consists of rows and columns.
- **Rows**: each row in the table represents a single entry or instance.
- **Columns**: each columns represents a specific attribute or property of data.
- **Database Management Systems (DBMS)**: it is a software that allows to interact with the databases (e.g.- MySQL, oracle, etc.)
- **SQL (Structured Query Language)**: SQL is a standard language used to communicate with relational databases, for tasks like querying, updating data, etc.

->> Databases can be broadly categorized into:

- **Relational Databases (RDBMS)**: it stores data in structured table with defined relationship between them.
- **Non-Relational Databases (NoSQL)**: it handles unstructured databases or semi-structured databases and offer more flexibility.

->> Databases are used in various applications, including finance, health, social media, etc.

Q-2) What is Normalization?

->> Normalization is a database design used to organize data efficiently by minimizing redundancy and dependency.

->> it involves large tables into smaller, related tables and defining relationships between them.

->> The main goal is to ensure that each piece of data is stored only once to avoid duplication and maintain data integrity.

Q-3) What is the difference between DBMS and RDBMS?

->> The main difference between DBMS and RDBMS are as follows:

	DBMS (Data Base Management System)	RDBMS (Relational Database Management System)
Data Structure	Stores data as files or records without relations.	Stores data in tables with relation between them.

Normalization	Usually doesn't support normalization	Supports normalization to eliminate redundancy
Relationships	No explicit support for relationships between data.	Relationships between data using foreign keys.
Data Integrity	No or minimal support for data integrity constraints.	Enforces integrity through foreign keys, primary keys, and rules
Examples	File systems, XML database	MySQL, Oracle, SQL server.

Q-4) What is MF Cod Rule of RDBMS Systems?

->> Codd's rule in DBMS also known as Codd's 12 rules is a set of thirteen rules (numbered 0 to 12), that define a database to be correct Relational Database Management System (RDBMS). If a database follows Codd's 12 rules, it is called a True Relational database management system.

->> Here's a brief overview of the rules:

- **Rule 0:**

  >> This rule states that for a system to qualify as an RDBMS, it must be able to manage database entirely through the relational capabilities.

- **Rule 1: Information Rule:**

  >> All information (including metadata) is to be represented as stored data in cells of tables. The rows and columns have to be strictly unordered.

- **Rule 2: Guaranteed Access:**

  >>Each unique piece of data (atomic value) should be accessible by: Table name +key(row) + attribute(column).

- **Rule 3: Systematic treatment of NULL:**

  >>" NULL" has several meanings, it can mean missing data, not applicable or no value. It should be handled consistently. Also, Primary key must not be null, ever. Expression on "NULL" must give null.

- **Rule 4: Active online catalog:**

  >>Database dictionary(catalog) is the structure description of the complete **Database** and it must be stored online. The Catalog must be governed by same rules as rest of the database. The same query language should be used on catalog as used to query database.

- **Rule 5: Powerful and well-structured language:**

  >>One well-structured language must be there to provide all manners of access to the data stored in the database. Example: SQL, etc. If the database allows access to the data without the use of this language, then that is a violation.

- **Rule 6: View updation rule:**

>>All the view that are theoretically updatable should be updatable by the system as well.

- **Rule 7: Relational level operation:**

>>There must be Insert, Delete, Update operations at each level of relations. Set operation like Union, Intersection and minus should also be supported.

- **Rule 8: Physical data independence:**

>>The physical storage of data should not matter to the system. If say, some file supporting table is renamed or moved from one disk to another, it should not affect the application.

- **Rule 9: Logical data independence:**

>>If there is change in the logical structure (table structures) of the database the user view of data should not change. Say, if a table is split into two tables, a new view should give result as the join of the two tables. This rule is most difficult to satisfy.

- **Rule 10: Integrity Independence:**

>>The database should be able to enforce its own integrity rather than using other programs. Key and Check constraints, trigger etc, should be stored in Data Dictionary. This also make RDBMS independent of front-end.

- **Rule 11: Distribution independence:**

>>A database should work properly regardless of its distribution across a network. Even if a database is geographically distributed, with data stored in pieces, the end user should get an impression that it is stored at the same place. This lays the foundation of distributed database.

- **Rule 12: No subversion rule:**

>>If low level access is allowed to a system, it should not be able to subvert or bypass integrity rules to change the data. This can be achieved by some sort of looking or encryption.

Q-5) What do you understand by By Data Redundancy?

->> Data redundancy refers to the unnecessary duplication of data within a database or storage system. It occurs when the same piece of data is stored in multiple places, either within a single table or across multiple tables, leading to various problems.

->> Key points about Data Redundancy:

- **Waste of Storage**: storing the same data for multiple times increases storage space requirements.
- **Inconsistencies**: redundant data can lead to discrepencies when one copy of the data is updated, but others are not, resulting in inconsistent information.

- **Data Anomalies**: redundancy can lead to update anomalies, insetion anomalies, and deletion anomalies, causing errors during database operations.

Q-6) What is DDL interpreter?

->> A DDL interpreter is a component of a DBMS responsible for interpreting and executing Data Definition Language (DDL) statements. DDL statements are used to define, modify, and manage the structure of a database, such as creating or altering tables, indexes, or views.

->> Role of DDL interpreter.

- It interprets DDL commands ex- CREATE, ALTER, DROP and translates it into low-level instructions that modifies database schema.
- It updates the database to reflect the changes in structure e.g.- table creation, column addition.
- Ensures data integrity and verifies that changes adhere to database's constraints and rules.

->> Common DDL commands:

1. **CREATE**- defines new objects in database
2. **ALTER**- modifies existing database objects.
3. **DROP**- deletes database objects.

Q-7) What is DML compiler in SQL?

->> A DML Compiler in SQL is a component of a Database Management System (DBMS) that processes and converts Data Manipulation Language (DML) statements into low-level machine code or executable instructions that the database can understand and execute. DML statements are used to perform operations like retrieving, inserting, updating, or deleting data within a database.

->> Role of DML compiler:

- Interprets and passes high- level DML commands e.g.- SELECT, INSERT, UPDATE, DELETE.
- Optimizes queries by choosing the most efficient execution plan execution plan based on factors like indexing, data size, and system resources.
- Generates executable code that performs the data manipulation requested by user.

->> Common DML commands:

1. **SELECT**: retrieves data from one table to other.
2. **INSERT**: adds new data to table.
3. **UPDATE**: modifies existing records in table.
4. **DELETE**: removes record from a table.

->> The DML compiler ensures that DML commands are processed quickly and accurately, optimizing query performance and resource usage in DBMS.

Q-8) What is SQL Key Constraints writing an Example of SQL Key Constraints?

->> SQL Key Constraints are rules that ensure the integrity and validity of the data within a database by uniquely identifying records, enforcing relationships between tables, and maintaining consistency. These constraints are applied to keys, which are specific columns in a table.

->> Types of SQL Key Constraints:

1. **Primary Key (PK)**: Ensures that each record in a table is unique and cannot be NULL. A table can only have one primary key.

2. **Foreign Key (FK)**: Enforces a relationship between two tables. It ensures that the value in one table must match a value in another table.

3. **Unique Key**: Ensures that all the values in a column or a group of columns are unique, allowing NULL values (except in cases where NOT NULL is also applied).

4. **Candidate Key**: Any column or set of columns that could qualify as a unique key but isn't the primary key.

5. **Composite Key**: A primary key made up of more than one column.

->> Example of Key Constraints:

- **PRIMARY KEY**:
```
CREATE TABLE Students (StudentID INT PRIMARY KEY, Name VARCHAR (50), Age INT);
```

- **FOREIGN KEY**:
```
CREATE TABLE Enrolments (EnrollmentID INT PRIMARY KEY, StudentID INT, CourseID INT,
FOREIGN KEY (StudentID) REFERENCES Students(StudentID));
```

- **UNIQUE KEY**:
```
CREATE TABLE Teachers (TeacherID INT PRIMARY KEY, Email VARCHAR(100) UNIQUE);
```

- **COMPOSITE PRIMARY KEY**:
```
CREATE TABLE Orders (OrderID INT, ProductID INT, Quantity INT,
PRIMARY KEY (OrderID, ProductID));
```

->> Explanation:

- The Primary Key in the 'Students' table ensures that 'StudentID' uniquely identifies each student and cannot be null.

- The Foreign Key in the 'Enrollments' table links 'StudentID' to the Students table, ensuring that any student enrolled exists in the 'Students' table.

- The Unique Key in the ‘Teachers’ table ensures that the email is unique for every teacher.
- The Composite Primary Key in the ‘Orders’ table ensures that each combination of ‘OrderID’ and ‘ProductID’ is unique.

Q-9) What is save Point? How to create a save Point write a Query?

->> A SAVEPOINT in SQL is a point within a transaction that allows you to temporarily save the state of the database at a specific moment. If needed, you can roll back to that specific point without undoing the entire transaction, providing more granular control during complex transactions.

->> Keypoints about save points:

- A transaction is a sequence of operations performed as a single unit of work.
- With SAVEPOINT, you can create multiple save points within a transaction , allowing a partial rollbacks.
- After a savepoint created, you can also Rollback to the specific point without affecting the other parts of the transactions.

->> How to create a SAVEPOINT:

- You can create a savepoint using SAVEPOINT command. Later, if an error occurs you wish to undo specific operations, you can rollback to that savepoint using ROLLBACK TO SAVEPOINT command.

->> Query to Create a SAVEPOINT:

```
START TRANSACTION;

INSERT INTO Employees (EmployeeID, Name, Salary) VALUES (1, 'Raj Pandhi', 5000);

SAVEPOINT savepoint1;

INSERT INTO Employees (EmployeeID, Name, Salary) VALUES (2, 'Deadpool', 6000);

ROLLBACK TO savepoint1;

COMMIT;
```

->> Savepoint’s are particularly useful in long and complex transactions, allowing partial rollbacks without needing to undo the entire sequence of operations.

Q-10) What is a trigger and how to create a TRIGGER in SQL?

->> A trigger in SQL is a special type of stored procedure that is automatically executed (or "triggered") when specific events occur in a database. These events could be data modification operations like INSERT, UPDATE, or DELETE on a table. Triggers are useful for maintaining data integrity, enforcing business rules, and automatically updating or auditing changes.

->> Key Points about Triggers:

- **Automatic execution**: Triggers are activated automatically by specified database events (e.g., after inserting a record).
- **Event-driven**: Triggers are defined to respond to certain actions like INSERT, UPDATE, or DELETE.
- **Types of Triggers**:
    - **BEFORE Trigger**: Executed before the event (e.g., before an insert or update).
    - **AFTER Trigger**: Executed after the event (e.g., after an insert or update).
- **Row-level or statement-level**: Triggers can be defined to execute for each row affected by a query (row-level) or once per statement (statement-level).

->> Syntax to create TRIGGER:

```
CREATE TABLE AuditLog (LogID INT AUTO_INCREMENT PRIMARY KEY, Action VARCHAR(50), EmployeeID INT, ActionTime DATETIME);

CREATE TRIGGER before_employee_insert

BEFORE INSERT

ON Employees

FOR EACH ROW

BEGIN

        INSERT INTO AuditLog (Action, EmployeeID, ActionTime)

        VALUES ('Employee Inserted', NEW.EmployeeID, NOW());

END;
```

->> When to use Triggers:

- To automatically audit changes e.g. logging updates or deletions.
- To enforce complex business rules.
- To cascade updates across the tables.

- To maintain computed values e.g. updating a total balance after each and every transactions.

->> Triggers are powerfull tools, but they should be used carefully as they can increase the complexity and potential of overhead database operations.