1. What is SQL, and why is it used?

SQL (Structured Query Language) is a standard programming language used to manage and manipulate relational databases. It is used to perform various operations on the data stored in a database, such as inserting, updating, deleting, and querying data. SQL is widely used because it provides an efficient way to handle large volumes of structured data and supports operations such as filtering, sorting, grouping, and joining.

Key Uses of SQL:

- To create and manage database structures (tables, views, indexes).
- To insert, update, and delete records.
- To query data using conditions and aggregations.
- To enforce data integrity through constraints like primary keys and foreign keys.

Example: Create a table to store employee details:

```
CREATE TABLE Employees (
    ID INT PRIMARY KEY,
    Name VARCHAR(50),
    Salary FLOAT
);
```

This creates an Employees table with columns for ID, Name, and Salary.

2. Write a query to fetch the second-highest salary from the Employee table.

To get the second-highest salary, we use a subquery to find the maximum salary less than the highest salary:

```
SELECT MAX(Salary) AS SecondHighestSalary
FROM Employees
WHERE Salary < (SELECT MAX(Salary) FROM Employees);
```

Explanation:

- The inner query SELECT MAX(Salary) FROM Employees retrieves the highest salary.
- The outer query finds the maximum salary that is less than the highest salary.

Example Result: If the salaries are 5000, 7000, and 9000, the query will return 7000 as the second-highest salary.

3. What are the different types of SQL commands?

SQL commands are categorized into five main types based on their functionality:

1. **DDL (Data Definition Language):**
    a. Defines and manages database schema.
    b. Commands: CREATE, ALTER, DROP, TRUNCATE.

Example:

```
CREATE TABLE Departments (
   DepartmentID INT PRIMARY KEY,
   DepartmentName VARCHAR(50)
);
```

2. **DML (Data Manipulation Language):**
    a. Manipulates data within tables.
    b. Commands: INSERT, UPDATE, DELETE.

Example:

```
INSERT INTO Employees (ID, Name, Salary) VALUES (1, 'John Doe', 5000);
```

3. **DQL (Data Query Language):**
    a. Retrieves data from the database.
    b. Command: SELECT.

Example:

```
SELECT * FROM Employees;
```

4. **DCL (Data Control Language):**
    a. Controls access to the database.
    b. Commands: GRANT, REVOKE.

Example:

GRANT SELECT ON Employees TO User1;

5. **TCL (Transaction Control Language):**
    a. Manages transactions.
    b. Commands: COMMIT, ROLLBACK, SAVEPOINT.

Example:

COMMIT;

4. Write a query to find duplicate records in a table.

To find duplicate records, we group by the column(s) and use the HAVING clause to filter groups with more than one record:

```
SELECT Name, COUNT(*)
FROM Employees
GROUP BY Name
HAVING COUNT(*) > 1;
```

Explanation:

- GROUP BY Name groups the rows by the Name column.
- COUNT(*) counts the rows in each group.
- HAVING COUNT(*) > 1 filters out groups that have only one record.

Example Result: If there are duplicate entries for "John", the query will list "John" and the count of duplicates.

5. What is the difference between DELETE and TRUNCATE?

Feature	DELETE	TRUNCATE
Purpose	Removes specific rows.	Removes all rows.
Rollback	Can be rolled back.	Cannot be rolled back.
Performance	Slower for large data.	Faster as it doesn't log rows.
Triggers	Activates triggers.	Does not activate triggers.

Example:

```
DELETE FROM Employees WHERE ID = 1;  -- Deletes specific record
TRUNCATE TABLE Employees;           -- Deletes all records
```

6. Write a query to get the department with the highest number of employees.

```
SELECT DepartmentID, COUNT(*) AS EmployeeCount
FROM Employees
GROUP BY DepartmentID
ORDER BY EmployeeCount DESC
LIMIT 1;
```

Explanation:

- GROUP BY DepartmentID groups the rows by departments.
- COUNT(*) counts the employees in each department.
- ORDER BY EmployeeCount DESC sorts the departments by employee count in descending order.
- LIMIT 1 retrieves only the top department.

7. What are joins in SQL? Name the types of joins.

Joins are used to combine rows from two or more tables based on a related column.

Types of Joins:

1. **Inner Join:** Returns matching rows from both tables.
2. **Left Join:** Returns all rows from the left table and matching rows from the right table.
3. **Right Join:** Returns all rows from the right table and matching rows from the left table.
4. **Full Outer Join:** Returns rows when there is a match in either table.
5. **Self Join:** Joins a table with itself.
6. **Cross Join:** Returns the Cartesian product of both tables.

Example:

```
SELECT Employees.Name, Departments.Name AS DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

This query retrieves employee names and their corresponding department names.

8. Write a query to fetch records where name starts with 'A'.

```
SELECT *
FROM Employees
WHERE Name LIKE 'A%';
```

Explanation:

- The LIKE operator is used for pattern matching.
- A% matches any string that starts with 'A'.

Example Result: If the names are "Alice", "Anna", and "Bob", this query will return "Alice" and "Anna".

9. What is a primary key, and how is it different from a unique key?

Feature	Primary Key	Unique Key
Uniqueness	Ensures row uniqueness.	Ensures column uniqueness.
NULL Values	Does not allow NULL values.	Allows a single NULL value.
Purpose	Identifies records uniquely.	Prevents duplicate values.

Example:

```
CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   Email VARCHAR(100) UNIQUE
);
```

- StudentID is the primary key.
- Email is a unique key.

10. Write a query to fetch employees who earn more than the average salary.

```
SELECT *
FROM Employees
WHERE Salary > (SELECT AVG(Salary) FROM Employees);
```

Explanation:

- The subquery SELECT AVG(Salary) FROM Employees calculates the average salary.
- The outer query fetches employees whose salaries are greater than this average.

11. What is a foreign key, and why is it important?

A **foreign key** is a column or set of columns in one table that establishes a link between the data in two tables. It acts as a reference to a primary key in another table, ensuring **referential integrity** of the data.

Importance of Foreign Key:

- Maintains data integrity by enforcing relationships between tables.
- Prevents actions that would destroy links between tables.
- Ensures consistency of data.

Example:

```
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50)
);

CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

Here, DepartmentID in the Employees table is a foreign key referencing the DepartmentID column in the Departments table.

12. Write a query to get the top 3 highest salaries in the Employee table.

```
SELECT DISTINCT Salary
FROM Employees
ORDER BY Salary DESC
LIMIT 3;
```

Explanation:

- ORDER BY Salary DESC sorts salaries in descending order.
- LIMIT 3 retrieves only the top 3 salaries.

13. What is the difference between WHERE and HAVING clauses?

Feature	WHERE	HAVING
Purpose	Filters rows before grouping.	Filters groups after grouping.
Usage	Used with SELECT, UPDATE, DELETE.	Used only with GROUP BY.
Aggregates	Cannot be used with aggregates.	Can filter based on aggregates.

Example:

```
-- Using WHERE
SELECT * FROM Employees
WHERE Salary > 5000;

-- Using HAVING
SELECT DepartmentID, AVG(Salary)
FROM Employees
GROUP BY DepartmentID
HAVING AVG(Salary) > 5000;
```

14. Write a query to fetch common records from two tables.

To find common records:

```
SELECT *
FROM TableA
INNER JOIN TableB
ON TableA.ID = TableB.ID;
```

Explanation:

- The INNER JOIN retrieves only matching rows based on the condition TableA.ID = TableB.ID.

15. What is normalization? Explain its types.

Normalization is the process of organizing data to minimize redundancy and improve data integrity. It divides tables into smaller, more manageable pieces.

Types of Normalization:

1. **1NF (First Normal Form):** Ensures each column contains atomic (indivisible) values.
2. **2NF (Second Normal Form):** Removes partial dependencies.
3. **3NF (Third Normal Form):** Removes transitive dependencies.
4. **BCNF (Boyce-Codd Normal Form):** A stricter version of 3NF.
5. **4NF and 5NF:** Deal with multi-valued and join dependencies.

Example: A non-normalized table with repeated data:

ID	Name	Department	DepartmentLocation
1	John	Sales	New York
2	Alice	Sales	New York

After normalization, split into two tables: **Employees:**

ID	Name	DepartmentID
1	John	101
2	Alice	101

Departments:

DepartmentID	DepartmentName	Location
101	Sales	New York

16. Write a query to create a table with constraints (primary key, unique, and foreign key).

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    ProductName VARCHAR(50) UNIQUE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Explanation:

- PRIMARY KEY ensures OrderID is unique and not null.
- UNIQUE ensures ProductName is unique.

- FOREIGN KEY establishes a relationship with the Customers table.

17. What are indexes in SQL, and what are their types?

Indexes are special lookup tables that speed up data retrieval.

Types of Indexes:

1. **Clustered Index:** Sorts and stores rows in the table based on key values.
2. **Non-Clustered Index:** Maintains a separate structure for data and index.
3. **Unique Index:** Ensures all index values are unique.
4. **Composite Index:** Combines multiple columns into a single index.

Example:

```
CREATE INDEX idx_name ON Employees(Name);
```

18. Write a query to count the number of employees in each department.

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

Explanation:

- GROUP BY DepartmentID groups rows by department.
- COUNT(*) counts employees in each department.

19. What is the difference between clustered and non-clustered indexes?

Feature	Clustered Index	Non-Clustered Index
Storage	Sorts and stores table rows.	Separate from table data.
Speed	Faster for range queries.	Slower for large data.
Number per table	Only one per table.	Multiple per table allowed.

20. Write a query to find employees who have not been assigned a department.

```
SELECT *
FROM Employees
WHERE DepartmentID IS NULL;
```

Explanation:

- IS NULL checks for rows where DepartmentID is not assigned.

21. What are aggregate functions in SQL? Give examples.

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

Common Aggregate Functions:

- SUM: Calculates total.
- AVG: Finds average.
- COUNT: Counts rows.
- MAX/MIN: Finds highest/lowest value.

Example:

```
SELECT COUNT(*) AS EmployeeCount, AVG(Salary) AS AverageSalary
FROM Employees;
```

22. Write a query to combine the results of two tables using UNION.

```
SELECT Name FROM Employees
UNION
SELECT Name FROM Customers;
```

Explanation:

- Combines unique rows from both tables.

23. What is the difference between UNION and UNION ALL?

Feature	UNION	UNION ALL
Duplicates	Removes duplicates.	Includes duplicates.
Performance	Slower due to duplicate removal.	Faster.

24. Write a query to fetch the nth highest salary in a table.

```
SELECT DISTINCT Salary
FROM Employees
ORDER BY Salary DESC
LIMIT 1 OFFSET (N-1);
```

Replace N with the desired rank.

25. What is a self-join, and when would you use it?

A **self-join** is a join of a table with itself. It's useful when comparing rows within the same table.

Example:

```
SELECT A.Name AS Employee1, B.Name AS Employee2
FROM Employees A, Employees B
WHERE A.ManagerID = B.EmployeeID;
```

26. Write a query to get the total salary paid to employees in each department.

```
SELECT DepartmentID, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY DepartmentID;
```

Explanation:

- SUM(Salary) calculates the total salary for each department.
- GROUP BY DepartmentID groups the rows by department.

27. What is the difference between RANK(), DENSE_RANK(), and ROW_NUMBER()?

Function	Purpose	Example Output
RANK()	Assigns a rank to each row with gaps if ties.	1, 2, 2, 4
DENSE_RANK()	Assigns a rank with no gaps for ties.	1, 2, 2, 3
ROW_NUMBER()	Assigns a unique number to each row.	1, 2, 3, 4

Example Query:

```
SELECT Name, Salary,
    RANK() OVER (ORDER BY Salary DESC) AS Rank,
    DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRank,
    ROW_NUMBER() OVER (ORDER BY Salary DESC) AS RowNumber
FROM Employees;
```

28. Write a query to update the salary of employees by 10% in the Employee table.

```
UPDATE Employees
SET Salary = Salary * 1.10;
```

Explanation: Multiplies the Salary by 1.10, effectively increasing it by 10%.

29. What are ACID properties in a database?

ACID properties ensure reliable transactions:

1. **Atomicity:** Transactions are all-or-nothing.
2. **Consistency:** Ensures the database is always in a valid state.
3. **Isolation:** Transactions do not interfere with each other.
4. **Durability:** Changes are permanent once committed.

30. Write a query to delete duplicate records from a table.

```
WITH CTE AS (
   SELECT Name, ROW_NUMBER() OVER (PARTITION BY Name ORDER BY ID) AS
RowNum
   FROM Employees
)
DELETE FROM Employees
WHERE ID IN (SELECT ID FROM CTE WHERE RowNum > 1);
```

Explanation:

- Assigns a unique row number to duplicates.
- Deletes rows with RowNum > 1.

31. Explain the order of execution of SQL.

1. **FROM/JOIN:** Selects and combines data from tables.
2. **WHERE:** Filters rows.
3. **GROUP BY:** Groups rows into summaries.
4. **HAVING:** Filters grouped rows.
5. **SELECT:** Selects columns.
6. **ORDER BY:** Sorts rows.
7. **LIMIT/OFFSET:** Limits the result set.

Example:

```
SELECT DepartmentID, COUNT(*)
FROM Employees
WHERE Salary > 5000
GROUP BY DepartmentID
HAVING COUNT(*) > 3
ORDER BY COUNT(*) DESC;
```

32. Write a query for year-on-year growth in SQL.

```
SELECT YEAR(SaleDate) AS Year,
    SUM(SaleAmount) AS TotalSales,
    SUM(SaleAmount) - LAG(SUM(SaleAmount)) OVER (ORDER BY YEAR(SaleDate)) AS
YoYGrowth
FROM Sales
GROUP BY YEAR(SaleDate);
```

33. Write a query for cumulative sum in SQL.

```
SELECT Name, Salary,
    SUM(Salary) OVER (ORDER BY Salary) AS CumulativeSalary
FROM Employees;
```

Explanation: SUM() OVER calculates a running total.

34. What is the difference between DELETE and TRUNCATE?

(See Question 5 for comparison.)

35. What is the difference between DML, DDL, and DCL?

Type	Description	Examples
DML	Data Manipulation Language.	INSERT, UPDATE.
DDL	Data Definition Language.	CREATE, DROP.
DCL	Data Control Language (permissions).	GRANT, REVOKE.

36. What are aggregate functions, and when do we use them?

Aggregate functions calculate values for multiple rows.

Examples:

- SUM: Total values.
- AVG: Average.
- COUNT: Count rows.
- MAX/MIN: Maximum/Minimum.

Example Query:

```
SELECT DepartmentID, AVG(Salary) AS AverageSalary
FROM Employees
GROUP BY DepartmentID;
```

37. Which is faster between CTE and Subquery?

- **CTE (Common Table Expression):** Better readability and reusable.
- **Subquery:** Inline and faster for small datasets.

38. What are constraints and their types?

Constraints enforce rules on table data.

Types:

- NOT NULL: Prevents null values.
- UNIQUE: Unique values in a column.
- PRIMARY KEY: Uniqueness and not null.
- FOREIGN KEY: Links to another table.
- CHECK: Validates column values.

39. What are the different types of keys?

1. **Primary Key:** Uniquely identifies rows.
2. **Unique Key:** Ensures unique values.
3. **Foreign Key:** Links tables.
4. **Composite Key:** Combines multiple columns.
5. **Candidate Key:** Potential primary keys.

40. What is the difference between varchar and nvarchar?

- **VARCHAR:** Stores ASCII characters (e.g., CHAR(50)).
- **NVARCHAR:** Stores Unicode characters (e.g., NCHAR(50)).

41. What are indexes in SQL?

(See Question 17.)

42. Write a query to combine two tables using UNION.

```
sSELECT Name FROM Employees
UNION
SELECT Name FROM Customers;
```

43. What is the difference between UNION and UNION ALL?

(See Question 23.)

44. What is a self-join, and when would you use it?

(See Question 25.)

45. Difference between Group By and Where?

Group By groups data, while **Where** filters rows before grouping.

46. What are Views in SQL?

Views are virtual tables derived from SQL queries.

47. Write a query to fetch employees who earn more than the average salary.

(See Question 10.)

48. Difference between Function and Stored Procedure?

Feature	Function	Stored Procedure
Return Type	Always returns a value.	May not return a value.
Usage	Used in queries.	Used for business logic.

49. What are triggers in SQL?

Triggers are automatic actions invoked by database events (INSERT, UPDATE, DELETE).

50. What is a retention query in SQL?

Used to measure active users retained over time.

Example:

```
SELECT UserID, COUNT(*)
FROM Logins
```

```
WHERE LoginDate BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY UserID;
```

51. Write a query for cumulative sum in SQL.

The CUMULATIVE SUM can be calculated using the SUM() window function with the OVER clause.

Query:

```
SELECT
    EmployeeID,
    Salary,
    SUM(Salary) OVER (ORDER BY EmployeeID) AS CumulativeSum
FROM Employees;
```

Explanation:

- SUM(Salary) OVER (ORDER BY EmployeeID) calculates the running total of salaries ordered by EmployeeID.

Example Output:

EmployeeID	Salary	CumulativeSum
1	5000	5000
2	7000	12000
3	8000	20000

52. Difference between Function and Stored Procedure.

Feature	Function	Stored Procedure
Return Type	Must return a value.	May or may not return a value.
Usage	Used in SELECT, WHERE, etc.	Called independently using EXEC.
Parameters	Only input parameters allowed.	Supports input, output, and in-out params.
DML Operations	Cannot perform DML operations.	Can perform DML operations.

Example of Function:

```
CREATE FUNCTION GetTotalSalary()
RETURNS FLOAT
AS
BEGIN
   RETURN (SELECT SUM(Salary) FROM Employees);
END;
```

Example of Stored Procedure:

```
CREATE PROCEDURE UpdateSalary(IN EmployeeID INT, IN Increment FLOAT)
BEGIN
   UPDATE Employees SET Salary = Salary + Increment WHERE ID = EmployeeID;
END;
```

53. Do we use variables in views?

No, variables cannot be used in SQL views. A view is a predefined SQL query stored in the database, and it does not support procedural logic like variables.

54. What are the limitations of views?

1. **Performance:** Complex views can impact performance.
2. **No Procedural Logic:** Cannot include loops, variables, or conditions.
3. **Update Restrictions:** Updating data through views with multiple joins or aggregations is restricted.

55. Write a query to create an index and explain its types.

Query to Create an Index:

```
CREATE INDEX idx_salary ON Employees(Salary);
```

Types of Indexes:

1. **Clustered Index:** Physically sorts the table data. Each table can have one clustered index.
2. **Non-Clustered Index:** Logical structure for searching. Can have multiple non-clustered indexes.

56. List the different types of relationships in SQL.

1. **One-to-One:** One record in Table A corresponds to one record in Table B.
2. **One-to-Many:** One record in Table A corresponds to multiple records in Table B.
3. **Many-to-Many:** Multiple records in Table A correspond to multiple records in Table B.

Example:

- **One-to-Many:** A department has many employees.

```
CREATE TABLE Departments (
   DepartmentID INT PRIMARY KEY,
   Name VARCHAR(50)
);

CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   Name VARCHAR(50),
   DepartmentID INT,
   FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

57. Differentiate between UNION and UNION ALL.

Feature	UNION	UNION ALL
Duplicate Removal	Removes duplicates from the result set.	Keeps duplicates in the result set.
Performance	Slower due to duplicate elimination.	Faster as no duplicate check is done.

Example:

```
SELECT Name FROM Employees_A
UNION
```

```
SELECT Name FROM Employees_B; -- Removes duplicates.

SELECT Name FROM Employees_A
UNION ALL
SELECT Name FROM Employees_B; -- Keeps duplicates.
```

58. How many types of clauses are there in SQL?

1. **SELECT Clause:** Retrieves data.
2. **WHERE Clause:** Filters rows based on conditions.
3. **GROUP BY Clause:** Groups data for aggregations.
4. **HAVING Clause:** Filters grouped data.
5. **ORDER BY Clause:** Sorts data.
6. **LIMIT Clause:** Limits the number of rows.
7. **JOIN Clause:** Combines rows from multiple tables.

59. What is the difference between UNION and UNION ALL in SQL?

This is the same as **Question 57**, refer above.

60. What are the various types of relationships in SQL?

This is the same as **Question 56**, refer above.

61. Difference between Primary Key and Secondary Key?

Feature	Primary Key	Secondary Key
Definition	Uniquely identifies a record.	Used for indexing and search.
Uniqueness	Must be unique and non-null.	Can contain duplicate values.
Purpose	Main unique identifier for the table.	Additional search or query purposes.

62. Write a query to find the second-highest salary of an employee.

This is the same as **Question 2**, refer above.

63. Write a retention query in SQL.

A **retention query** is used to calculate the percentage of users retained across two time periods.

Query:

```
SELECT
    COUNT(DISTINCT UserID) AS RetainedUsers,
    (COUNT(DISTINCT UserID) * 100.0 / (SELECT COUNT(DISTINCT UserID) FROM UserLogs
WHERE LoginDate BETWEEN '2024-01-01' AND '2024-01-31')) AS RetentionRate
FROM UserLogs
WHERE LoginDate BETWEEN '2024-02-01' AND '2024-02-28'
AND UserID IN (
    SELECT UserID FROM UserLogs WHERE LoginDate BETWEEN '2024-01-01' AND '2024-
01-31'
);
```

64. Write year-on-year growth in SQL.

Query:

```
SELECT
    EXTRACT(YEAR FROM SalesDate) AS Year,
    SUM(SalesAmount) AS TotalSales,
    (SUM(SalesAmount) - LAG(SUM(SalesAmount)) OVER (ORDER BY EXTRACT(YEAR
FROM SalesDate))) / LAG(SUM(SalesAmount)) OVER (ORDER BY EXTRACT(YEAR FROM
SalesDate)) * 100 AS YoYGrowth
FROM Sales
GROUP BY EXTRACT(YEAR FROM SalesDate);
```