1. What is SQL?

SQL (Structured Query Language) is a standard programming language used to communicate with **relational databases**. It allows users to create, read, update, and delete data, and provides commands to define database schema and manage database security.

2. What is a database?

A database is an **organized collection of data** stored electronically, typically structured in tables with rows and columns. It is managed by a **database management system** (DBMS), which allows for efficient **storage**, **retrieval**, and **manipulation** of data.

3. What are the main types of SQL commands?

SQL commands are broadly classified into:

- **DDL (Data Definition Language):** CREATE, ALTER, DROP, TRUNCATE.
- **DML (Data Manipulation Language):** SELECT, INSERT, UPDATE, DELETE.
- **DCL (Data Control Language):** GRANT, REVOKE.
- **TCL (Transaction Control Language):** COMMIT, ROLLBACK, SAVEPOINT.

4. What is the difference between CHAR and VARCHAR2 data types?

- **CHAR:** Fixed-length storage. If the defined length is not fully used, it is padded with spaces.
- **VARCHAR2:** Variable-length storage. Only the actual data is stored, saving space when the full length is not needed.

5. What is a primary key?

A primary key is a unique identifier for each record in a table. It ensures that no two rows have the same value in the primary key column(s), and it does not allow NULL values.

6. What is a foreign key?

A foreign key is a column (or set of columns) in one table that refers to the primary key in another table. It establishes and enforces a relationship between the two tables, ensuring data integrity.

7. What is the purpose of the DEFAULT constraint?

The DEFAULT constraint assigns a default value to a column when no value is provided during an **INSERT operation**. This helps maintain consistent data and simplifies data entry.

8. What is normalization in databases?

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

9. What is denormalization, and when is it used?

Denormalization is the process of combining **normalized tables** into larger tables for performance reasons. It is used when **complex queries** and joins slow down data retrieval, and the performance benefits outweigh the **drawbacks of redundancy**.

10. What is a query in SQL?

A query is a SQL statement used to retrieve, update, or manipulate data in a **database**. The most common type of query is a SELECT statement, which fetches data from one or more tables based on specified conditions.

1. What are the different operators available in SQL?

- **Arithmetic Operators:** +, -, *, /, %
- **Comparison Operators:** =, !=, <>, >, <, >=, <=
- **Logical Operators:** AND, OR, NOT
- **Set Operators:** UNION, INTERSECT, EXCEPT
- **Special Operators:** BETWEEN, IN, LIKE, IS NULL

12. What is a view in SQL?

A view is a **virtual table** created by a **SELECT query**. It does not store data itself, but presents data from one or more tables in a structured way. Views simplify complex queries, improve readability, and enhance security by restricting access to specific rows or columns.

13. What is the purpose of the UNIQUE constraint?

The UNIQUE constraint ensures that all values in a column (or combination of columns) are **distinct**. This prevents duplicate values and helps maintain data integrity.

14. What are the different types of joins in SQL?

- **INNER JOIN:** Returns rows that have matching values in both tables.
- **LEFT JOIN (LEFT OUTER JOIN):** Returns all rows from the left table, and matching rows from the right table.
- **RIGHT JOIN (RIGHT OUTER JOIN):** Returns all rows from the right table, and matching rows from the left table.
- **FULL JOIN (FULL OUTER JOIN):** Returns all rows when there is a match in either table.
- **CROSS JOIN:** Produces the Cartesian product of two tables.

15. What is the difference between INNER JOIN and OUTER JOIN?

- **INNER JOIN:** Returns only rows where there is a match in both tables.
- **OUTER JOIN:** Returns all rows from one table (LEFT, RIGHT, or FULL), and the matching rows from the other table. If there is no match, NULL values are returned for the non-matching side.

16. What is the purpose of the GROUP BY clause?

The GROUP BY clause is used to arrange **identical data** into **groups**. It is typically used with aggregate functions (such as COUNT, SUM, AVG) to perform calculations on each group rather than on the entire dataset.

17. What are aggregate functions in SQL?

Aggregate functions perform calculations on a set of values and return a single value. Common aggregate functions include:

- COUNT(): Returns the number of rows.
- SUM(): Returns the total sum of values.
- AVG(): Returns the average of values.
- MIN(): Returns the smallest value.

- MAX(): Returns the largest value.

18. What is a subquery?

A subquery is a query nested within another query. It is often used in the **WHERE clause** to filter data based on the results of another query, making it easier to handle complex conditions.

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

- **WHERE:** Filters rows before any grouping takes place.
- **HAVING:** Filters grouped data after the GROUP BY clause has been applied.
  In short, WHERE applies to individual rows, while HAVING applies to groups.

20. What are indexes, and why are they used?

Indexes are **database objects** that improve query performance by allowing **faster retrieval of rows**. They function like a book's index, making it quicker to find specific data without scanning the entire table. However, indexes require **additional storage** and can slightly slow down **data modification** operations.

21. What is the difference between DELETE and TRUNCATE commands?

- **DELETE:** Removes rows one at a time and records each deletion in the transaction log, allowing rollback. It can have a WHERE clause.
- **TRUNCATE:** Removes all rows at once without logging individual row deletions. It cannot have a WHERE clause and is faster than DELETE for large data sets.

22. What is the purpose of the SQL ORDER BY clause?

The ORDER BY clause sorts the result set of a query in either **ascending** (default) or **descending order**, based on one or more columns. This helps present the data in a more meaningful or readable sequence.

23. What are the differences between SQL and NoSQL databases?

- **SQL Databases:**
  - Use structured tables with rows and columns.

    - Rely on a fixed schema.
    - Offer **ACID** properties.
- **NoSQL Databases:**
    - Use flexible, schema-less structures (e.g., key-value pairs, document stores).
    - Are designed for horizontal scaling.
    - Often focus on performance and scalability over strict consistency.

24. What is a table in SQL?

A table is a **structured collection** of related data organized into rows and columns. Columns define the type of data stored, while rows contain individual records.

25. What are the types of constraints in SQL?

Common constraints include:

- **NOT NULL:** Ensures a column cannot have NULL values.
- **UNIQUE:** Ensures all values in a column are distinct.
- **PRIMARY KEY:** Uniquely identifies each row in a table.
- **FOREIGN KEY:** Ensures referential integrity by linking to a primary key in another table.
- **CHECK:** Ensures that all values in a column satisfy a specific condition.
- **DEFAULT:** Sets a default value for a column when no value is specified.

26. What is a cursor in SQL?

A cursor is a database object used to **retrieve**, **manipulate**, and traverse through rows in a result set one row at a time. Cursors are helpful when performing operations that must be processed sequentially rather than in a set-based manner.

27. What is a trigger in SQL?

A trigger is a set of SQL statements that automatically execute in response to certain events on a table, such as **INSERT**, **UPDATE**, or **DELETE**. Triggers help

maintain **data consistency**, enforce business rules, and implement complex integrity constraints.

28. What is the purpose of the SQL SELECT statement?

The SELECT statement retrieves data from one or more tables. It is the most commonly used command in SQL, allowing users to filter, sort, and display data based on specific criteria.

29. What are NULL values in SQL?

NULL represents a missing or unknown value. It is different from zero or an empty string. NULL values indicate that the data is not available or applicable.

30. What is a stored procedure?

A stored procedure is a precompiled set of SQL statements stored in the **database**. It can take input parameters, perform logic and queries, and return output values or result sets. Stored procedures improve **performance** and **maintainability** by centralizing business logic.

SQL Intermediate Interview Questions

31. What is the difference between DDL and DML commands?

1. DDL (Data Definition Language):

These commands are used to **define** and **modify the structure of database** objects such as **tables**, **indexes**, and **views**. For example, the **CREATE command** creates a new table, the **ALTER command** modifies an existing table, and the **DROP command** removes a table entirely. DDL commands primarily focus on the schema or structure of the database.

Example:

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

2. DML (Data Manipulation Language):

These commands deal with the **actual data stored** within database objects. For instance, the **INSERT command** adds rows of data to a table,

the **UPDATE** command modifies existing data, and the **DELETE** command removes rows from a table. In short, DML commands allow you to query and manipulate the data itself rather than the structure.

Example:

INSERT INTO Employees (ID, Name) VALUES (1, 'Alice');

32. What is the purpose of the ALTER command in SQL?

The ALTER command is used to **modify the structure** of an existing database object. This command is essential for adapting our **database schema** as requirements evolve.

- Add or drop a column in a table.
- Change a column's data type.
- Add or remove constraints.
- Rename columns or tables.
- Adjust indexing or storage settings.

33. What is a composite primary key?

A c**omposite primary key** is a primary key made up of two or more columns. Together, these columns must form a unique combination for each row in the table. It's used when a single column isn't sufficient to uniquely identify a record.

Example:

Consider an Orders table where OrderID and ProductID together uniquely identify each record because multiple orders might include the same product, but not within the same order.

```
CREATE TABLE OrderDetails (
   OrderID INT,
   ProductID INT,
   Quantity INT,
   PRIMARY KEY (OrderID, ProductID)
);
```

34. How is data integrity maintained in SQL databases?

Data integrity refers to the **accuracy**, **consistency**, and **reliability** of the data stored in the database. SQL databases maintain data integrity through several mechanisms:

- **Constraints:** Ensuring that certain conditions are always met. For example, NOT NULL ensures a column cannot have missing values, FOREIGN KEY ensures a valid relationship between tables, and UNIQUE ensures no duplicate values.
- **Transactions:** Ensuring that a series of operations either all succeed or all fail, preserving data consistency.
- **Triggers:** Automatically enforcing rules or validations before or after changes to data.
- **Normalization:** Organizing data into multiple related tables to minimize redundancy and prevent anomalies.
  These measures collectively ensure that the data remains reliable and meaningful over time.

35. What are the advantages of using stored procedures?

- **Improved Performance:** Stored procedures are precompiled and cached in the database, making their execution faster than sending multiple individual queries.
- **Reduced Network Traffic:** By executing complex logic on the server, fewer round trips between the application and database are needed.
- **Enhanced Security:** Stored procedures can restrict direct access to underlying tables, allowing users to execute only authorized operations.
- **Reusability and Maintenance:** Once a procedure is written, it can be reused across multiple applications. If business logic changes, you only need to update the stored procedure, not every application that uses it.

36. What is a UNION operation, and how is it used?

The UNION operator combines the result sets of two or more SELECT queries into a single result set, removing duplicate rows. This is useful when we need a **consolidated view** of data from multiple tables or queries that have similar structure.

Example:

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

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

- **UNION:** Removes duplicate rows from the result set, ensuring only unique rows are returned.
- **UNION ALL:** Includes all rows from each query, including duplicates.
- Performance-wise, UNION ALL is faster because it doesn't require an additional step to remove duplicates.

Example:

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

38. What are the differences between SQL's COUNT() and SUM() functions?

1. COUNT(): Counts the number of rows or non-NULL values in a column.

Example:

```
SELECT COUNT(*) FROM Orders;
```

2. SUM(): Adds up all numeric values in a column.

Example:

```
SELECT SUM(TotalAmount) FROM Orders;
```

39. What are window functions, and how are they used?

Window functions perform calculations across a set of rows that are related to the current row. Unlike aggregate functions, they don't collapse the result set.

Example: Calculating a running total

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

40. What is the difference between an index and a key in SQL?

1. Index

- An index is a database object created to **speed up data retrieval**. It stores a sorted reference to table data, which helps the database engine find rows more quickly than scanning the entire table.
- **Example:** A non-unique index on a column like LastName allows quick lookups of rows where the last name matches a specific value.

2. **Key**

- A key is a logical concept that enforces rules for uniqueness or relationships in the data.
- For instance, a **PRIMARY KEY** uniquely identifies each row in a table and ensures that no duplicate or NULL values exist in the key column(s).
- A **FOREIGN KEY** maintains referential integrity by linking rows in one table to rows in another.

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

1. Clustered Index:

- Organizes the physical data in the table itself in the order of the indexed column(s).
- A table can have only one clustered index.
- Improves range queries and queries that sort data.
- Example: If EmployeeID is the clustered index, the rows in the table are stored physically sorted by EmployeeID.

2. Non-Clustered Index:

- Maintains a separate structure that contains a reference (or pointer) to the physical data in the table.
- A table can have multiple non-clustered indexes.
- Useful for specific query conditions that aren't related to the primary ordering of the data.
- Example: A non-clustered index on LastName allows fast lookups by last name even if the table is sorted by another column.

42. How do constraints improve database integrity?

Constraints enforce rules that the data must follow, preventing invalid or inconsistent data from being entered:

- **NOT NULL:** Ensures that a column cannot contain NULL values.
- **UNIQUE:** Ensures that all values in a column are distinct.
- **PRIMARY KEY:** Combines NOT NULL and UNIQUE, guaranteeing that each row is uniquely identifiable.
- **FOREIGN KEY:** Ensures referential integrity by requiring values in one table to match primary key values in another.
- **CHECK:** Validates that values meet specific criteria (e.g., CHECK (Salary > 0)).
  By automatically enforcing these rules, constraints maintain data reliability and consistency.

SQL Advanced Interview Questions

43. What are the ACID properties of a transaction?

ACID is an acronym that stands for Atomicity, Consistency, Isolation, and Durability—four key properties that ensure database transactions are processed reliably.

1. **Atomicity:**

- A transaction is treated as a single unit of work, meaning all operations must succeed or fail as a whole.
- If any part of the transaction fails, the entire transaction is rolled back.

2. **Consistency:**

- A transaction must take the database from one valid state to another, maintaining all defined rules and constraints.
- This ensures data integrity is preserved throughout the transaction process.

3. **Isolation:**

- Transactions should not interfere with each other.
- Even if multiple transactions occur simultaneously, each must operate as if it were the only one in the system until it is complete.

4. **Durability:**

- Once a transaction is committed, its changes must persist, even in the event of a system failure.
- This ensures the data remains stable after the transaction is successfully completed.

44. What are the differences between isolation levels in SQL?

Isolation levels define the extent to which the operations in one transaction are isolated from those in other transactions. They are critical for **managing concurrency** and ensuring data integrity. Common isolation levels include:

1. Read Uncommitted:

- Allows reading uncommitted changes from other transactions.
- Can result in dirty reads, where a transaction reads data that might later be rolled back.

2. Read Committed:

- Ensures a transaction can only read committed data.
- Prevents dirty reads but does not protect against non-repeatable reads or phantom reads.

3. Repeatable Read:

- Ensures that if a transaction reads a row, that row cannot change until the transaction is complete.
- Prevents dirty reads and non-repeatable reads but not phantom reads.

4. Serializable:

- The highest level of isolation.
- Ensures full isolation by effectively serializing transactions, meaning no other transaction can read or modify data that another transaction is using.

- Prevents dirty reads, non-repeatable reads, and phantom reads, but may introduce performance overhead due to locking and reduced concurrency.

Query Based SQL Interview Questions

45. Write a query to find the second-highest salary of an employee in a table.

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

Explanation:

This query identifies the second-highest salary by selecting the maximum salary that is less than the overall highest salary. The subquery determines the top salary, while the outer query finds the next highest value.

46. Write a query to retrieve employees who earn more than the average salary.

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

Explanation:

This query fetches details of employees whose salary exceeds the average salary. The subquery calculates the average salary, and the main query filters rows based on that result.

83. Write a query to fetch the duplicate values from a column in a table.

```
SELECT ColumnName, COUNT(*)
FROM TableName
GROUP BY ColumnName
HAVING COUNT(*) > 1;
```

Explanation:

The query uses GROUP BY to group identical values and HAVING COUNT(*) > 1 to identify values that appear more than once in the specified column.

84. Write a query to find the employees who joined in the last 30 days.

```
SELECT *
FROM Employee
WHERE JoiningDate > DATE_SUB(CURDATE(), INTERVAL 30 DAY);
```

Explanation:

By comparing the JoiningDate to the current date minus 30 days, this query retrieves all employees who joined within the last month.

85. Write a query to fetch top 3 earning employees.

```
SELECT *
FROM Employee
ORDER BY Salary DESC
LIMIT 3;
```

Explanation:

The query sorts employees by salary in descending order and uses LIMIT 3 to return only the top three earners.

86. Write a query to delete duplicate rows in a table without using the ROWID keyword.

```
DELETE FROM Employee
WHERE EmployeeID NOT IN (
  SELECT MIN(EmployeeID)
  FROM Employee
  GROUP BY Column1, Column2
);
```

Explanation:

This query retains only one row for each set of duplicates by keeping the row with the smallest EmployeeID. It identifies duplicates using GROUP BY and removes rows not matching the minimum ID.

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

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

Explanation:

An INNER JOIN is used to find rows present in both tables by matching a common column (in this case, ID).

88. Write a query to fetch employees whose names start and end with ‘A’.

```
SELECT *
FROM Employee
WHERE Name LIKE 'A%' AND Name LIKE '%A';
```

Explanation:

The query uses LIKE with wildcard characters to filter rows where the Name column starts and ends with the letter ‘A’.

89. Write a query to display all departments along with the number of employees in each.

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

Explanation:

By grouping employees by their DepartmentID and counting rows in each group, the query produces a list of departments along with the employee count.

90. Write a query to find employees who do not have managers.

```
SELECT *
FROM Employee
WHERE ManagerID IS NULL;
```

Explanation:

This query selects employees whose ManagerID column is NULL, indicating they don’t report to a manager.

91. Write a query to fetch the 3rd and 4th highest salaries.

```
SELECT Salary
FROM Employee
ORDER BY Salary DESC
LIMIT 2 OFFSET 2;
```

Explanation:

Using ORDER BY and the combination of LIMIT and OFFSET, the query skips the top two salaries and retrieves the next two highest.

92. Write a query to transpose rows into columns.

```
SELECT
  MAX(CASE WHEN ColumnName = 'Condition1' THEN Value END) AS
Column1,
  MAX(CASE WHEN ColumnName = 'Condition2' THEN Value END) AS
Column2
FROM TableName;
```

Explanation:

This query converts specific row values into columns using conditional aggregation with CASE. Each column's value is determined based on a condition applied to rows.

93. Write a query to fetch records updated within the last hour.

```
SELECT *
FROM TableName
WHERE UpdatedAt >= NOW() - INTERVAL 1 HOUR;
```

Explanation:

By comparing the UpdatedAt timestamp to the current time minus one hour, the query retrieves rows updated in the last 60 minutes.

94. Write a query to list employees in departments that have fewer than 5 employees.

```
SELECT *
FROM Employee
WHERE DepartmentID IN (
  SELECT DepartmentID
  FROM Employee
  GROUP BY DepartmentID
  HAVING COUNT(*) < 5
);
```

Explanation:

The subquery counts employees in each department, and the main query uses those results to find employees working in departments with fewer than 5 members.

95. Write a query to check if a table contains any records.

```
SELECT CASE
    WHEN EXISTS (SELECT * FROM TableName) THEN 'Has Records'
    ELSE 'No Records'
    END AS Status;
```

Explanation:

The query uses EXISTS to determine if any rows exist in the table, returning a status of ‘Has Records’ or ‘No Records’ based on the result.

96. Write a query to find employees whose salaries are higher than their managers.

```
SELECT e.EmployeeID, e.Salary
FROM Employee e
JOIN Employee m ON e.ManagerID = m.EmployeeID
WHERE e.Salary > m.Salary;
```

Explanation:

This query joins the Employee table with itself to compare employee salaries to their respective managers’ salaries, selecting those who earn more.

Write a query to fetch alternating rows from a table.

```
SELECT *
FROM Employee
WHERE MOD(RowID, 2) = 0;
```

Explanation:

By applying a modulo operation on a unique identifier (RowID), the query returns rows with even IDs, effectively fetching every other row.

98. Write a query to find departments with the highest average salary.

```
SELECT DepartmentID
FROM Employee
GROUP BY DepartmentID
ORDER BY AVG(Salary) DESC
LIMIT 1;
```

Explanation:

Grouping by DepartmentID and ordering by the average salary in descending order, the query returns the department with the highest average.

99. Write a query to fetch the nth record from a table.

```
SELECT *
FROM Employee
LIMIT 1 OFFSET (n-1);
```

Explanation:

Using LIMIT 1 OFFSET (n-1), this query retrieves a single row corresponding to the specified position (nth) in the ordered result set.

100. Write a query to find employees hired in the same month of any year.

```
SELECT *
FROM Employee
WHERE MONTH(JoiningDate) = MONTH(CURDATE());
```

Explanation:

By comparing the month of JoiningDate to the current month, the query selects all employees who were hired in that month regardless of the year.