SQL

1. **What is a SQL Query?**

   A SQL Query is a structured query language statement used to interact with relational databases. It allows users to retrieve, insert, update, and delete data in a database. SQL Queries are used to perform various operations on the data, such as filtering, sorting, grouping, and aggregating, to retrieve the desired information from the database.

2. **How do you retrieve specific columns from a table using SQL?**

   To retrieve specific columns from a table, you can use the SELECT statement followed by the column names you want to fetch.

   For example:

   SELECT column1, column2, column3 FROM table_name;

3. **What is the purpose of the WHERE clause in SQL Queries?**

   The WHERE clause in SQL is used to filter the rows returned by a query based on specified conditions. It allows you to retrieve only the rows that meet the specified criteria, making the query more selective and targeted.

4. **How do you sort the result of a SQL Query?**

   To sort the result of a SQL query, you can use the ORDER BY clause, followed by the column(s) you want to sort on and the sorting order (ASC for ascending and DESC for descending).

   For example:

   SELECT column1, column2 FROM table_name ORDER BY column1 ASC;

5. **Explain the difference between INNER JOIN and LEFT JOIN in SQL.**

   The differences between INNER JOIN and LEFT JOIN in SQL are explained below:

   INNER JOIN: An INNER JOIN returns only the matching rows from both tables being joined. It filters out the rows where no matching data is found in either table.

   LEFT JOIN: A LEFT JOIN returns all the rows from the left (or first) table and the matching rows from the right (or second) table. If there is no match in the right table, it returns NULL values for the columns from the right table.

6. **What is an Aggregate function in SQL? Provide examples of aggregate functions.**

   An Aggregate function in SQL performs a calculation on a set of values and returns a single value as the result. Common aggregate functions include:

   COUNT(): Returns the count of rows (number of rows) present in the result set.

   SUM(): Calculates the sum of a numeric column in a result set.

   AVG(): Calculates the average of a numeric column in a result set.

   MAX(): Returns the highest value (maximum) of a column present in the result set.

   MIN(): Returns the lowest value (minimum) of a column present in the result set

7. **What is a primary key in SQL, and why is it important?**

   A primary key in SQL is used to identify a column uniquely and without NULL values in each row of a table. It ensures data integrity by enforcing entity uniqueness and acts as a reference point for establishing relationships between tables. Primary keys are crucial for data retrieval and indexing, enabling faster search operations and maintaining data consistency.

8. **Explain the difference between the HAVING clause and the WHERE clause in SQL.**

   The WHERE clause is used to filter rows before they are grouped and aggregated, whereas the HAVING clause is used as a condition to filter the result of aggregate functions applied to grouped data. WHERE is applied before any grouping, whereas HAVING is applied after grouping.

9. **How do you create a new table in SQL? Provide an example.**

   To create a new table in SQL, you use the CREATE TABLE statement.

   For example:

```
CREATE TABLE employees (
   employee_id INT PRIMARY KEY,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   age INT,
   department VARCHAR(100)
);
```

10. **What is a foreign key in SQL, and how does it establish a relationship between tables?**

    A foreign key in SQL is used to refer to a column or a set of columns that are the primary key of another table. It establishes a relationship between two tables, known as the parent table (with the primary key) and the child table (with the foreign key). This relationship ensures data integrity by enforcing referential constraints and maintaining data consistency between related tables.

11. **How do you use the DISTINCT keyword in a SQL Query? Provide an example.**

    The DISTINCT keyword is used to retrieve unique values from a column in a SQL Query.

    For example:

    ```
    SELECT DISTINCT department FROM employees;
    ```

12. **Explain the purpose of the GROUP BY clause in SQL and when is used?**

    The GROUP BY clause is used to group rows based on specified columns in a SQL query. It is often used with aggregate functions like SUM, COUNT, AVG, etc. to perform calculations on groups of data. The GROUP BY clause is used when you want to analyse data in subsets or groups rather than the entire dataset.

13. **How can you update data in a table using SQL? Provide an example.**

    To update data in a table using SQL, you use the UPDATE statement.

    For example:

    ```
    UPDATE employees
    SET department = 'Sales'
    WHERE employee_id = 101;
    ```

14. **What is the purpose of the BETWEEN operator in SQL? Give an example of how it is used.**

    The BETWEEN operator in SQL is used to retrieve rows with values within a specified range.

    For example:

    ```
    SELECT product_name, price
    FROM products
    WHERE price BETWEEN 1000 AND 2000;
    ```

15. **Explain the LIKE operator in SQL, and how is it different from the = operator?**

The LIKE operator is used in SQL to perform pattern matching on text fields. It allows the use of wildcards (% and _) to match specific patterns.

For example:

```
SELECT product_name
FROM products
WHERE product_name LIKE 'Chair%';
```

The = operator, on the other hand, performs an exact match comparison between two values. It does not allow for pattern matching or the use of wildcards.

16. **What are SQL views, and what are their advantages?**

SQL views are virtual tables created by SQL Queries. They are not physically stored in the database but provide a way to simplify complex queries and present a customized subset of data to users. Advantages of SQL views include improved security by restricting access to underlying tables, simplified data retrieval, and reduced redundancy in queries.

17. **Explain the concept of SQL Subqueries.**

SQL Subqueries (also known as Nested Queries) are queries within another query. These subqueries are deployed to retrieve data that will be used as a condition or criteria for the main query. Subqueries are enclosed within parentheses and can be used in SELECT, INSERT, UPDATE, or DELETE statements.

18. **Write a SQL Query to retrieve the product names and their corresponding categories for products with a price greater than $50.**

Given the "Products" table:

ProductID	ProductName	Category	Price
1	Laptop	Electronics	800
2	T-shirt	Apparel	20
3	Headphones	Electronics	100
4	Jeans	Apparel	50

```
SELECT ProductName, Category
FROM Products
WHERE Price > 50;
```

19. **Explain the differences between the INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN in SQL.**

The differences between the INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN in SQL are described below:

INNER JOIN: Returns only the rows where there is a match in both tables.

LEFT JOIN: Returns all the records (rows) from the left table and the matching records (rows) from the right table. If there is no match in the right table, NULL values are returned.

RIGHT JOIN: Returns all the rows from the right table and the matching rows from the left table. If there is no match in the left table, NULL values are returned.

FULL JOIN: Returns all the rows when there is a match in either the left or right table. If there is no match in both tables, NULL values are returned for the non-matching side.

20. **What are SQL transactions, and what are the properties (ACID) associated with them?**

SQL transactions are sequences of one or more SQL operations that are executed as a single unit of work. Transactions ensure data integrity and consistency in the database. The properties associated with transactions (often referred to as ACID properties) are as follows:

Atomicity: Transactions indicate all operations or nothing. If a single unit of the transaction fails, the entire transaction is rolled back, and the database remains unchanged.

Consistency: Transactions bring the database from one valid state to another. It ensures that the data satisfies all defined integrity constraints.

Isolation: Transactions are executed in isolation from other transactions, so the intermediate states of a transaction are not visible to other transactions.

Durability: Once a transaction is saved (committed), the changes are permanent and will survive even if there is a system failure.

21. **Explain the difference between UNION and UNION ALL in SQL.**

The difference between UNION and UNION ALL in SQL is defined further:

UNION: Combines the result sets of two or more SELECT statements, removing duplicate rows from the final result set. It performs a distinct operation.

UNION ALL: Also combines the result sets of two or more SELECT statements but includes all rows, even if there are duplicates. It does not remove duplicate rows from the final result set, resulting in faster performance compared to UNION.

22. **Write a SQL Query to retrieve the count of orders placed on each date.**

Given the "Orders" table:

OrderID	CustomerID	OrderDate
1	101	12-05-2021
2	102	16-07-2019
3	103	30-11-2022

```
SELECT OrderDate, COUNT(OrderID) AS OrderCount
FROM Orders
GROUP BY OrderDate;
```

23. **What is a self-join in SQL? Provide an example of how it can be used.**

A self-join is a type of SQL join where a table is joined with itself. It is used when you need to combine rows from the same table based on a related column. One common scenario is when a table has a hierarchical relationship, such as an employee and manager relationship.

For Example:

Consider an "Employees" table with columns: EmployeeID, FirstName, LastName, and ManagerID. We can use a self-join to retrieve the manager's name for each employee.

```
SELECT e.FirstName AS EmployeeName, m.FirstName AS ManagerName
FROM Employees e
JOIN Employees m ON e.ManagerID = m.EmployeeID;
```

24. **Explain the difference between the CHAR and VARCHAR data types in SQL.**

The difference between the CHAR and VARCHAR data types in SQL is referred to as:

CHAR: The CHAR data type is used to store fixed-length strings. When you define a CHAR column, you must specify the maximum number of characters it can hold. If the actual data is shorter than the defined length, it is padded with spaces to reach the specified length.

VARCHAR: The VARCHAR data type is used to store variable-length strings. Unlike CHAR, VARCHAR columns only consume the necessary amount of storage based on the actual data length. It does not pad the data with spaces.

25. **Write a SQL Query to calculate the average total order amount.**

Given the "Orders" table:

Order ID	Customer ID	Total Amount
1	101	150
2	102	200
3	103	100

```
SELECT AVG(TotalAmount) AS AvgTotalAmount
FROM Orders;
```

26. **Write a SQL Query to retrieve the customer names (concatenated first name and last name) along with their corresponding city.**

Given the "Customers" table:

OrderID	CustomerID	TotalAmount
1	101	150
2	102	200
3	103	100

```
SELECT CONCAT(FirstName, ' ', LastName) AS CustomerName, City
FROM Customers;
```

27. **Explain the concept of a Primary Key in SQL and its importance.**

In SQL, a primary key in SQL is used to identify a column uniquely and without NULL values in each row of a table. It ensures that there are no duplicate or NULL values in the designated column(s), which helps maintain data integrity and ensures the uniqueness of each record in the table.

The importance of a Primary Key includes:

- ❖ **Uniqueness:** It enforces the uniqueness of records, ensuring that no two rows in the table have the same primary key value.
- ❖ **Data Integrity:** It helps maintain data integrity by preventing the insertion of duplicate or NULL values in the primary key column(s).
- ❖ **Indexing:** A primary key is automatically indexed, which improves the performance of search and retrieval operations on the table.

- **Relationships:** Primary keys are used to establish relationships between tables through foreign keys, creating the basis for referential integrity in the database.
- **Efficient Updates:** Updating or deleting records based on primary key values is more efficient than doing the same without a primary key.

28. Explain the differences between a view and a table in SQL.

The differences between a view and a table in SQL are described further:

Table: A table is a fundamental database object that stores data in a structured format. It is a permanent, physical storage location for data. Data in a table is directly accessible, and you can perform various operations like inserting, updating, and deleting records.

View: A view is a virtual table derived from one or more tables or views. It does not store any data itself but contains a query that defines how the data should be retrieved from the underlying tables. Views provide a way to simplify complex queries, restrict data access, and present a different perspective on the data without altering the original tables.

29. Explain the concept of SQL Common Table Expressions (CTEs) and provide an example.

Common Table Expressions (CTEs) are temporary result sets that can be used within a SELECT, INSERT, UPDATE, or DELETE statement. CTEs help simplify complex queries and make them more readable by breaking them down into smaller, manageable parts.

For Example:

Consider the “**Products**” table. We want to retrieve the products with a price greater than the average price:

```
WITH AveragePrice AS (
  SELECT AVG(Price) AS AvgPrice
  FROM Products
)
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AvgPrice FROM AveragePrice);
```

In this example, the CTE named “**AveragePrice**” calculates the average price of products. The main query then selects the product names and prices from the

"**Products**" table where the price is greater than the average price obtained from the CTE.

31. **Explain the concept of SQL Self-Joins and provide an example.**

SQL Self-Joins are used to join a table with itself. It is useful when you want to combine rows from the same table based on related columns within the table.

For Example:

Consider the "Employees" table with columns: EmployeeID, FirstName, LastName, and ManagerID. We can use a self-join to retrieve the employee names along with their manager names.

```
SELECT e.FirstName AS EmployeeName, m.FirstName AS ManagerName
FROM Employees e
JOIN Employees m ON e.ManagerID = m.EmployeeID;
```

32. **Explain the concept of SQL Index Optimization and its significance in improving database performance.**

SQL Index Optimization involves analyzing the database and creating appropriate indexes on tables to improve the speed of data retrieval operations. It plays a crucial role in enhancing database performance and optimizing query execution.

Significance of SQL Index Optimization:

- ❖ **Faster Data Retrieval:** Indexes allow the database engine to quickly find and retrieve the required data, reducing the need for full table scans and improving query performance.
- ❖ **Efficient Joins:** Indexes optimize join operations, which are essential for complex queries that involve multiple tables.
- ❖ **Minimized Disk I/O:** Indexes reduce the number of disk I/O operations, which can significantly improve the overall performance of the database.
- ❖ **Query Performance Tuning:** Properly designed indexes can make queries execute faster and help identify and fix performance bottlenecks.
- ❖ **Considerations:** While indexes improve read performance, they may have a slight negative impact on write operations. Therefore, it's essential to strike a balance between read and write operations and create indexes selectively on columns that are frequently used for filtering, sorting, or joining..

33. **Explain the concept of SQL Subquery vs. JOIN and when to use each.**

    SQL Subquery: A subquery is a query nested within another query. It is used to retrieve data that will be used as a condition or criteria for the main query. Subqueries are typically used when the result of one query depends on the result of another query and when you need to filter the data based on certain conditions before performing the main query.

    JOIN: A JOIN is used to combine rows from more than one table (two or more tables) based on a common column between them. It is used when you need to retrieve data from multiple tables and when the data in one table is related to the data in another table.

    Choosing between a subquery and a JOIN depends on the specific requirements of the query. Generally, if you need to filter data before joining, or if you need to retrieve data from multiple tables, a JOIN is more appropriate. On the other hand, if you need to filter data based on the result of another query, a subquery is the preferred choice.

34. **Explain the concept of SQL Common Table Expressions (CTEs) and provide an example.**

    SQL Common Table Expressions (CTEs) are temporary result sets that are defined within a single SQL statement like SELECT, INSERT, UPDATE and DELETE. CTEs are used to simplify complex queries by breaking them down into smaller, manageable parts. They improve the readability of queries and can be referenced multiple times within a single query.

35. **Explain the concept of SQL Window Functions.**

    SQL Window Functions are used to perform calculations among a set of related rows to the current row, without changing the result set. They are used with the OVER() clause to define the window or group of rows to which the function applies.

36. **Explain the concept of SQL Triggers.**

    SQL Triggers are special types of stored procedures that will automatically execute when a particular event occurs in the database. Triggers are used to enforce data integrity, implement business rules, or perform custom actions based on specific conditions.

37. **Explain the concept of SQL User-Defined Functions (UDFs) and provide an example.**

SQL User-Defined Functions (UDFs) are custom functions created by users to encapsulate business logic or frequently used calculations. UDFs can be called within SQL queries to perform specific tasks and return a result.

For Example:

Creating an UDF to calculate the total price of a product based on its quantity and unit price:

```
CREATE FUNCTION CalculateTotalPrice(@Quantity INT, @UnitPrice
DECIMAL(10, 2))
RETURNS DECIMAL(10, 2)
AS
BEGIN
  DECLARE @TotalPrice DECIMAL(10, 2);
  SET @TotalPrice = @Quantity * @UnitPrice;
  RETURN @TotalPrice;
END;
```

38. **Explain the difference between input parameters and output parameters in stored procedures.**

- ❖ **Input Parameters:** Input parameters are used to pass values into the stored procedure when it is called. They allow the procedure to accept external data, such as search criteria or IDs, to perform specific operations.
- ❖ **Output Parameters:** Output parameters allow the stored procedure to return values back to the caller. Unlike result sets, which return multiple rows, output parameters are used to send specific data values back to the calling program.

39. **Explain the concept of transaction management within stored procedures.**

Transaction management within stored procedures involves controlling the scope of a transaction and ensuring data consistency. Stored procedures can be executed within a transaction, and you can specify whether the transaction should be committed or rolled back based on specific conditions or outcomes within the procedure. This ensures that multiple related operations either succeed or fail together as a single unit of work.

40. **What is the purpose of the SQL EXECUTE AS clause in stored procedures?**

The EXECUTE AS clause in stored procedures specifies the security context under which the procedure will be executed. It allows you to control the permissions and privileges available for the procedure, reducing the risk of unauthorized access to sensitive data. The EXECUTE AS clause can be set to SELF (default), OWNER, CALLER, or a specific user or login.

41. **What are the advantages of using stored procedures?**

Advantages of using stored procedures include:

- **Code Reusability:** Stored procedures can be called from multiple applications, reducing code duplication.
- **Security:** Stored procedures allow controlling access to data by granting execution permissions, reducing the risk of SQL injection.
- **Performance:** Stored procedures are precompiled, resulting in improved execution speed.
- **Simplified Maintenance:** Changes to the stored procedure are done in one place, simplifying maintenance.
- **Encapsulation of Business Logic:** Business logic can be encapsulated within stored procedures, making it easier to maintain and update.

42. **Explain the concept of SQL Subqueries vs. Common Table Expressions (CTEs) and when to use each.**

The basic concept of SQL Subqueries vs. Common Table Expressions (CTEs) are defined further:

SQL Subqueries: Subqueries are queries that are nested inside another query and are usually enclosed within parentheses. They are executed first, and their result is then used in the main query. Subqueries are often used when the result of one query depends on the result of another query.

Common Table Expressions (CTEs): CTEs are temporary result sets that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement. They help simplify complex queries by breaking them down into smaller, more manageable parts. CTEs are useful when a query involves recursion or when parts of the query need to be referenced multiple times.

Choosing between a subquery and a CTE depends on the specific requirements of the query. Generally, use a subquery when you need to use the result once and a CTE when you need to reference the result multiple times within the same query.

43. **What is an SQL index, and what is its purpose?**
An SQL index is a database object that enhances data retrieval speed by providing a quick lookup method for rows in a table. It is created on one or more columns of a table and works like an index in a book, allowing the database engine to locate data more efficiently. The primary purpose of an index is to optimize query performance by reducing the number of disk I/O operations required to fetch data.

44. **Explain the difference between a clustered index and a non-clustered index.**
The difference between a clustered index and a non-clustered index is described below:
Clustered Index: A clustered index establishes the physical order of the data in the table. Each table can have only one clustered index. When a clustered index is created, the table's data rows are reorganized to match the index's logical order. This means the data is physically stored in the same order as the index key. Clustered indexes are generally used on columns that are frequently used in range queries, such as dates.
Non-clustered Index: A non-clustered index creates a separate structure from the table data. It contains a copy of the indexed columns along with pointers to the actual data rows. Each table can have multiple non-clustered indexes. Non-clustered indexes are useful for improving the performance of queries that involve JOINs or WHERE clauses.

45. **How does indexing improve database performance? Provide some examples.**
Indexing improves database performance by reducing the time it takes to find and retrieve data. It does this by creating a quick lookup mechanism based on the indexed columns. Here are some examples of how indexing can enhance performance:
    - For a large customer database, indexing the "**CustomerID**" column allows for rapid lookup when retrieving a specific customer's information.
    - Indexing the "**Date**" column in a sales table can significantly speed up queries involving date ranges, such as monthly sales reports.
    - In a product catalog, indexing the "**Category**" column can expedite searches for products within specific categories.