1. WHAT IS SQL?

SQL stands for Structured Query Language. It's a standardized way of communicating with relational databases, which are a type of database that stores information in tables with rows and columns. SQL lets us access, manipulate, and retrieve data from these databases.

2. EXPLAIN TYPES SQL STATEMENTS?

There are five main types of SQL statements, categorized based on their function in dealing with databases:

Data Definition Language (DDL) Statements: These statements are used to define and modify the structure of your database. They control the schema, which is essentially the blueprint for your data. DDL statements include:

CREATE: Used to create databases, tables, indexes, and other database objects.

ALTER: Allows you to modify the structure of existing tables by adding, removing, or changing columns.

DROP: Used to delete databases, tables, and other database objects.

Data Manipulation Language (DML) Statements: DML statements are all about working with the actual data stored within your database tables. They are used to insert, update, and delete data. Common DML statements include:

INSERT: Adds new rows of data to a table.

UPDATE: Modifies existing data in a table based on certain criteria.

DELETE: Removes rows of data from a table based on specific conditions.

Data Control Language (DCL) Statements: DCL statements manage user access and permissions within the database. They control who can access the data and what they can do with it (read, write, etc.). Some examples include:

GRANT: Gives users specific permissions on database objects.

REVOKE: Takes away permissions that were previously granted.

Transaction Control Language (TCL) Statements: TCL statements are used to manage database transactions. A transaction is a unit of work that involves one or more DML statements. TCL statements ensure data integrity by controlling how changes are committed or rolled back. Examples include:

COMMIT: Makes all the changes from the current transaction permanent.

ROLLBACK: Undoes all the changes from the current transaction.

Data Query Language (DQL) Statements: DQL statements, also known as Select statements, are used to retrieve data from a database. They allow you to specify which data you want to see and how you want it filtered or ordered. SELECT is the primary DQL statement.

3. HOW TO CREATE A DATABASE AND DATABASE NAME CRITERIAS?

4. WHAT ARE THE DATABASE OBJECTS?

Tables:

A table is the basic unit of storage in a database. It consists of rows and columns, where each row represents a record, and each column represents a field or attribute.

Views:

A view is a logical table based on one or more existing tables. It contains no data of its own but provides a way to access and manipulate data from the base tables.

Sequences:

A sequence generates unique integers that can be used as primary key values. It’s a user-created object shared by multiple users.

Indexes:

An index improves query performance by providing fast access to rows in a table. It uses pointers to locate data quickly.

Stored Procedures:

A stored procedure is a set of SQL statements that can be executed as a single unit. It allows us to encapsulate business logic and reuse it.

Functions:

A function is similar to a stored procedure but returns a value. It can be used in queries or expressions.

5. WHAT IS A DATABASE?

A database is a structured collection of data that is organized and stored in a computer system. It is designed to efficiently manage, retrieve, and manipulate large amounts of data to support various applications and information needs.

6. WHAT IS A TABLE?

Tables: A table is a collection of related data organized into rows and columns. Each row represents a record or entry, while each column represents a specific attribute or field of the data.

7. WHAT IS RELATIONAL DATABASE?

A relational database is a type of database that organizes and stores data in tables consisting of rows and columns, following the relational model of data.

8. WHAT IS SQL NORMALIZATION FORMS?

Certainly! **Normalization** in SQL is the process of **structuring a relational database** to minimize redundancy, improve data integrity, and eliminate anomalies. It involves dividing large tables into smaller, related tables while adhering to specific rules called **normal forms**. Let's explore the common normalization forms: 1NF, 2NF, 3NF, BCNF etc.

9. TYPES OF CONSTRAINTS IN SQL?

In SQL, there are several types of constraints that help maintain data accuracy and integrity within database tables. Let's explore them:

NOT NULL: Ensures that a column cannot have a **NULL** value. In other words, it mandates that every entry in that column must contain valid data.

UNIQUE: Guarantees that all values in a column are different. No duplicate values are allowed within the specified column.

PRIMARY KEY: Combines the properties of **NOT NULL** and **UNIQUE**. It uniquely identifies each row in a table. Typically, the primary key is chosen from one or more columns.

FOREIGN KEY: Establishes a relationship between two tables. It ensures that values in a specific column (the foreign key) correspond to values in another table's primary key. This constraint maintains referential integrity.

CHECK: Validates that the values in a column satisfy a specific condition. For instance, we can use it to ensure that ages are positive integers or that email addresses follow a specific format.

DEFAULT: Sets a default value for a column if no value is explicitly specified during insertion.

CREATE INDEX: Although not strictly a constraint, it significantly improves data retrieval speed by creating an index on a column or set of columns.

Creating a table with NOT NULL:

```sql
CREATE TABLE Employees (
    EmpID INT PRIMARY KEY,
    EmpName VARCHAR(100) NOT NULL,
    Department VARCHAR(50) NOT NULL,
    Salary DECIMAL(10,2) NULL
);
```

```sql
CREATE TABLE Employees (
    EmpID INT PRIMARY KEY,
    EmpName VARCHAR(100) NOT NULL,
    Department VARCHAR(50),
    Salary DECIMAL(10,2)
);
```

```sql
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Email VARCHAR(100) UNIQUE,
    Username VARCHAR(50) UNIQUE
);
```

```sql
CREATE TABLE Departments (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(100) NOT NULL
);

CREATE TABLE Employees (
    EmpID INT PRIMARY KEY,
    EmpName VARCHAR(100) NOT NULL,
    DeptID INT,
    FOREIGN KEY (DeptID) REFERENCES Departments(DeptID)
);
```

```sql
CREATE TABLE Employees (
    EmpID INT PRIMARY KEY,
    EmpName VARCHAR(100) NOT NULL,
    Age INT CHECK (Age >= 18)
);
```

```sql
CREATE TABLE Employees (
    EmpID INT PRIMARY KEY,
    EmpName VARCHAR(100) NOT NULL,
    Department VARCHAR(50) DEFAULT 'General',
    JoiningDate DATE DEFAULT GETDATE()   -- SQL Server (for current date)
);
```

```sql
ALTER TABLE Employees
ADD CONSTRAINT PK_Employees PRIMARY KEY (EmpID);
```

10. WHAT IS PRIMARY KEY AND UNIQUE KEY?

UNIQUE: Guarantees that all values in a column are different. No duplicate values are allowed within the specified column.

PRIMARY KEY: Combines the properties of **NOT NULL** and **UNIQUE**. It uniquely identifies each row in a table. Typically, the primary key is chosen from one or more columns.

11. CAN WE HAVE 2 PRIMARY KEYS IN THE SAME TABLE?

In SQL, a table can have **only one primary key**. The primary key uniquely identifies each row in the table. However, you can achieve a similar effect by using a **composite primary key**, which consists of multiple columns. Each combination of values in these columns must be unique.

12. CAN WE HAVE 2 UNIQUE KEYS IN THE SAME TABLE?

Yes, we can.

13. WHAT CHAR, VARCHAR AND NVARCHAR?

CHAR:

Fixed-length character data type.

Stores non-Unicode characters (usually ASCII characters).

Reserves storage space for the specified number of characters, even if not fully utilized.

Example: If you define a CHAR(10) column, it will always occupy 10 bytes, regardless of the actual data length.

Suitable for storing data with a consistent length.

VARCHAR:

Variable-length character data type.

Stores non-Unicode characters.

Only uses space for the characters actually stored (no reserved space).

Example: If you define a VARCHAR(10) column and store "Hello," it will occupy only 6 bytes (5 characters + 1 byte for length information).

Ideal for flexible-length data.

NVARCHAR:

Variable-length character data type.

Stores Unicode characters (essential for extended character sets, multilingual support, and special characters).

Similar to VARCHAR but supports Unicode.

Example: If you define an NVARCHAR(10) column and store "こんにちは" (Japanese greeting), it will use 20 bytes (10 characters × 2 bytes each).

Use NVARCHAR when dealing with internationalization or Unicode data.

14. HOW TO CREATE TABLE WITH AUTO IDENTITY?

- In SQL Server, use the `IDENTITY` property to create an auto-incrementing primary key:

SQL

```
CREATE TABLE Persons (
    Personid INT IDENTITY(1,1) PRIMARY KEY,
    LastName VARCHAR(255) NOT NULL,
    FirstName VARCHAR(255),
    Age INT
);
```

15. WHAT IS THE DIFFERENCE BETWEEN DROP/DELETE/TRUNCATE A TABLE?

1. **DELETE**:
    - Removes specific rows (records) from a table.
    - Retains the table structure.
    - Uses DML (Data Manipulation Language).
    - Can be rolled back.
    - Slower than TRUNCATE.
    - Ideal for selective deletions.
2. **TRUNCATE**:
    - Quickly removes all rows from a table.
    - Retains the table structure.
    - Uses DDL (Data Definition Language).

- Cannot be rolled back.
- Faster than DELETE.
- Ideal for clearing entire tables.

3. **DROP**:
   - Permanently deletes the entire table.
   - Removes both data and structure.
   - Uses DDL.
   - Irreversible.
   - Ideal for removing entire tables.

16. HOW TO CONNECT SQL TO OTHER DATABASE SERVER TO RETRIVE DATA?

17. HOW DO YOU GET DATA TO SQL SERVER DATABASE?

We can directly use the import option from the object explore window, 1st we need to right click on the appropriate database and select the import data…. From and do the work or we can directly use the below code to directly import the data.

SQL

```
BULK INSERT MyTable
FROM 'C:\Data\MyData.csv'
WITH (FIELDTERMINATOR = ',', ROWTERMINATOR = '\n');
```

18. HOW TO DESCRIBE A DATABASE?

By using “USE” command.

19. HOW TO DESCRIBE A TABLE?

By using the “SELECT & FORM” Command.

20. HOW TO SEE THE COUNT OF RECORDS AND COUNT OF FIELDS IN A TABLE?

1. **Count of Records (Rows)**: To find the total number of records (rows) in a table, you can use the **COUNT()** function. Here are a couple of ways to achieve this:
   - Using the wildcard `*`:

     SQL

     ```
     SELECT COUNT(*) AS [Number of Records]
     FROM your_table_name;
     ```

   - Using a specific column (e.g., `id`):

     SQL

     ```
     SELECT COUNT(id) AS [Number of Records]
     FROM your_table_name;
     ```

2. **Count of Fields (Columns)**: To get the total number of fields (columns) in a table, you can query the **INFORMATION_SCHEMA.COLUMNS** view. Here’s how:

   SQL

   ```
   SELECT COUNT(*) AS [Number of Fields]
   FROM INFORMATION_SCHEMA.COLUMNS
   WHERE TABLE_NAME = 'your_table_name';
   ```

21. WHAT IS INDEX?

In SQL (Structured Query Language), an index is a database object that improves the speed of data retrieval operations on a table. Indexes work by creating a sorted structure of key values from one or more columns of a table. This sorted structure allows the database management system (DBMS) to quickly locate and access rows based on the indexed columns.

Indexes are crucial for optimizing the performance of SELECT queries, especially when dealing with large tables. When you create an index on a table, the DBMS creates a separate data structure that contains pointers to the actual rows in the table. This allows the DBMS to perform efficient lookups based on the indexed columns.

22. WHAT IS THE DIFFERENCE BETWEEN CLUSTERED AND NON-CLUSTERED INDEX?

Clustered Index:

Physically orders the rows in the table based on the indexed column(s).
Each table can have only one clustered index.
Often used for columns that are frequently queried for range-based searches or when retrieving large portions of data.
Primary key constraints automatically create clustered indexes unless specified otherwise.

Non-Clustered Index:

Stores a separate sorted structure of index keys and pointers to rows.
Does not affect the physical order of rows in the table.
Each table can have multiple non-clustered indexes.
Ideal for columns frequently used in search conditions or join operations.

23. HOW TO UNIQUE OR DISTINCT VALUES IN A COLUMN?

By using the "**SELECT DISTINCT column_name**
FROM table_name;" key word.

24. HOW TO CREATE A NEW TABLE BY SELECTING FEW FIELDS FROM ANOTHER TABLE?

25. HOW TO RETRIVE DATA USING FILTERING VALUES? GIVE AN EXAMPLE

WE can filter out the data by using where keywords.

```
SELECT *
FROM products
WHERE category = 'Electronics' AND price > 500;
```

26. HOW TO GET COUNTRY AND PRODUCT WISE TOTAL SALES?

By using group by keyword.

27. EXPLAIN GROUP BY CLAUSE? CHALLENGES

The GROUP BY clause in SQL is used to group rows that have the same values in specified columns into summary rows, such as finding the total sales for each product or the average age of employees in each department. It is often used in combination with aggregate functions like SUM(), AVG(), COUNT(), MAX(), and MIN() to perform calculations on grouped data.

Challenges with GROUP BY include understanding how it affects result sets, handling NULL values, and writing efficient queries when dealing with large datasets.

28. WHAT IS THE DIFFERENCE BETWEEN WHERE AND HAVING CLAUSE?

WHERE clause:

- The WHERE clause is used to filter rows before any groupings are made.
- It is applied to individual rows in the result set based on the conditions specified.
- Typically used with SELECT, UPDATE, and DELETE statements.

HAVING clause:

- The HAVING clause is used to filter rows after the groupings are made.
- It is applied to aggregated groups of rows based on the conditions specified.
- Typically used with the GROUP BY clause to filter grouped data.

29. WHAT IS THE SEQUENCE OF CLAUSES? GIVE AN EXAMPLE

In SQL, the sequence of clauses generally follows a specific order when constructing a query. The typical sequence is:

- SELECT
- FROM
- WHERE
- GROUP BY
- HAVING
- ORDER BY

30. HOW TO CRATE IF NESTED IF CONDITIONS IN EXCEL? GIVE AN EXAMPLE

By using “CASE-WHEN-THEN-ELSE-END” clauses.

31. HOW TO UPDATE A TABLE?

To update a table in SQL, you can use the UPDATE statement followed by the table name, the columns you want to update, and the conditions that specify which rows should be updated. Here's the basic syntax for the UPDATE statement:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

32. HOW TO DELETE A TABLE BASED ON CRIERIAS? GIVE AN EXAMPLE

To delete rows from a table based on certain criteria in SQL, you use the DELETE statement along with the WHERE clause to specify the conditions for deletion. Here's the basic syntax for deleting rows from a table:

```
DELETE FROM table_name
WHERE condition;
```

33. HOW TO TRANSPOSE COLUMNS INTO ROWS AND ROWS INTO COLUMNS IN SQL? GIVE AN EXAMPLE

In SQL, We can transpose columns into rows and vice versa using the PIVOT and UNPIVOT operators or by using conditional aggregation with the CASE statement. Below are examples for both methods:

Using PIVOT and UNPIVOT:

- PIVOT: Transposes rows into columns.
- UNPIVOT: Transposes columns into rows.

```
-- Transpose rows into columns using PIVOT
SELECT *
FROM (
   SELECT ProductName, Year, SalesAmount
   FROM Sales
) AS SourceTable
PIVOT (
   SUM(SalesAmount)
   FOR Year IN ([2020], [2021])
) AS PivotTable;
```

```
SELECT Year,
       [Q1] AS Q1_Revenue,
       [Q2] AS Q2_Revenue,
       [Q3] AS Q3_Revenue,
       [Q4] AS Q4_Revenue
FROM (
    SELECT Year, Quarter, Revenue
    FROM Sales
) AS SourceTable
PIVOT (
    SUM(Revenue)
    FOR Quarter IN ([Q1], [Q2], [Q3], [Q4])
) AS PivotTable;
```

```
-----UNPIVOT
```

```
SELECT Year, Quarter, Revenue
FROM (
    SELECT Year, Q1_Revenue, Q2_Revenue, Q3_Revenue, Q4_Revenue
    FROM PivotedSales
) AS SourceTable
UNPIVOT (
    Revenue FOR Quarter IN (Q1_Revenue, Q2_Revenue, Q3_Revenue, Q4_Revenue)
) AS UnpivotTable;
```

34. HOW TO APPEND A TABLE IN SQL? EXPLAIN THE CHALLENGES

By using “UNION ALL” & “UNION” option, it’s only possible when all the attribute or columns are same for all the table.

35. WHAT IS THE DIFFERENCE BETWEEN UNION VS UNION ALL? GIVE EN EXAMPLE

UNION ALL means it’s append all the records from the both table that means if the both table having the common records that it will show the common records twice in the result but in UNION option the result show the unique records.

36. EXPLAIN JOINING IN SQL? EXAMPLE

Full join, inner join, left join, right join, left null join, right null join, self join, cross join.

37. WHAT IS NATURAL JOIN IN SQL? EXAMPLE

A natural join in SQL is a type of join that automatically matches columns with the same name in two tables. It eliminates the need to specify the join condition explicitly, as it assumes that columns with the same name are related and should be used for the join.

```
SELECT *
FROM employees
NATURAL JOIN departments;
```

38. WHAT IS SELF JOIN IN SQL? EXAMPLE

```
SELF JOIN IS A REGULAR JOIN, THE TABLE GETS JOINED TO ITSELF
       EACH ROW OF THE TABLE GET COMBINED TO ANOTHER ROW IN THE SAME TABLE OR THE SAME ROW
```

39. WHAT IS CROSS JOIN IN SQL? EXAMPLE

It just marge the two table no matter they have common fields of attributes or not.

40. WHAT IS NULL JOIN IN SQL? EXAMPLE

In SQL, a "NULL join" is not a standard type of join like inner join, outer join, or cross join. Instead, the term "NULL join" typically refers to a situation where you explicitly include NULL values in a join condition or handle NULL values in the result set of a join.

41. WHAT ARE TYPES OF RELATION IN SQL JOIN? GIVE EXAMPLES

There are 4 types of relationship in SQL they are one-to-one, one-to-many, many-to-one, many-to-many.

42. WHAT IS A DATA MODEL IN SQL? GIVE EXAMPLE

A data model in SQL defines how data is organized, stored, and accessed in a database. It includes tables, columns, relationships between tables, and constraints to ensure data integrity.

Data models serve as blueprints for designing databases, enabling developers to understand and manipulate data effectively.

43. WHAT IS A FACT TABLE?

In data modelling, a fact table is a central table in a star schema or snowflake schema that contains quantitative data (facts) about a business process or activity. Fact tables typically store numeric, additive data, such as sales revenue, quantities sold, expenses, or other measurable metrics.

44. WHAT IS A DIMENSION TABLE?

In SQL and data modelling, a dimension table is a table that contains descriptive attributes related to the data in a fact table. Dimension tables provide context and additional details about the quantitative data stored in the fact table, allowing for meaningful analysis and reporting. They typically store categorical or textual data that describes the dimensions or perspectives of the business process being analysed.

45. WHAT IS STAR SCHEMA DATA MODEL? EXAMPLE

A star schema is a type of data model used in data warehousing and database design. It organizes data into a central fact table surrounded by dimension tables. This arrangement resembles a star pattern when visualized, hence the name "star schema."

46. WHAT IS SNOWFLAKES DATA MODE? EXAMPLE

A snowflake schema is a type of data modelling technique used in data warehousing and database design. It is an extension of the star schema, where dimension tables are further normalized into multiple smaller dimension tables, resulting in a more complex but more normalized structure.

47. CAN WE HAVE 2 FACT TABLES IN A DATA MODEL?

Yes, it is possible to have multiple fact tables in a data model. Having multiple fact tables is common in complex data modelling scenarios where there are multiple business processes or activities that generate different sets of quantitative data (facts) that need to be analysed independently.

Each fact table typically corresponds to a specific business process or area of interest and contains quantitative data related to that process. For example, in a retail data model, you might have separate fact tables for sales transactions, inventory levels, and customer interactions, each capturing different aspects of the business.

48. WHAT WILL HAPPEN IF WE GO FOR A DATA MODEL WITH INNER JOIN TO LEFT JOIN TO RIGHT JOIN? EXAMPLE

49. WHAT ARE SUBQUERIES? EXAMPLE

Subqueries, also known as nested queries or inner queries, are SQL queries that are embedded within another SQL query. They are used to retrieve data that meets certain criteria or conditions based on the results of another query. Subqueries are enclosed within parentheses and can be placed in various parts of an SQL statement, such as the SELECT, FROM, WHERE, or HAVING clauses. (write a select statement inside another select statement).

50. HOW TO FIRST,2ND,3RD HIGHIEST SALARY BY EACH DEPARTMENTS?

```sql
    SELECT
        employee_id,
        first_name,
        last_name,
        department_id,
        salary,
        ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rank
    FROM
        employees
)
SELECT
    employee_id,
    first_name,
    last_name,
    department_id,
    salary,
    CASE rank
        WHEN 1 THEN 'First Highest'
        WHEN 2 THEN 'Second Highest'
        WHEN 3 THEN 'Third Highest'
        ELSE 'Other' -- Handle additional ranks if needed
    END AS rank_description
FROM
    RankedSalaries
WHERE
    rank <= 3; -- Filter for first, second, and third highest salaries
```

51. WHAT IS THE DIFFERENCE BETWEEN RANK OVER () AND DENSE_RANK OVER ()?

- RANK() assigns the same rank to duplicate values and leaves gaps in the ranking sequence.
- DENSE_RANK() also assigns the same rank to duplicate values but does not leave gaps in the ranking sequence; it fills in the missing ranks.

Let's say we have the following data in the "Students" table:

Name	Score
Alice	90
Bob	85
Charlie	80
David	85
Eve	75

The query above will rank the students based on their scores in descending order using both `RANK()` and `DENSE_RANK()`. The output might look like this:

Name	Score	RankWithGaps	DenseRank
Alice	90	1	1
Bob	85	2	2
Charlie	80	3	3
David	85	2	2
Eve	75	5	4

Notice how `RANK()` leaves a gap in the ranks when there are ties (e.g., ranks 2 and 3 are not used), whereas `DENSE_RANK()` assigns consecutive ranks to tied rows without gaps.

52. HOW TO GET %OF PARENT TOTAL IN SQL PROGRAMMING?

sql Copy code

```sql
SELECT
    Category,
    Value,
    Value / SUM(Value) OVER (PARTITION BY Category) * 100 AS PercentOfParentTotal
FROM
    YourTable;
```

53. HOW TO GET CUMULATIVE SUM OF SALES BY EACH COUNTRY BY PRODUCT WISE?

```
SUM(MARKS) OVER(PARTITION BY country,product ORDER BY country,product) AS CUMMALIVE_SUM
```

We may or may not be use order by here as per the question.

54. HOW TO GET CUMULATIVE SUM OF SALES WITHOUT USING PARTITION BY?

```
SELECT date, sales,
  SUM(sales) OVER (ORDER BY date ROWS UNBOUNDED PRECEDING) AS cumulative_sales
FROM your_sales_table
ORDER BY date;
```

55. WHAT IS LAG FUNCTION? EXAMPLE

The LAG function in SQL is a window function that allows US to access data from previous rows within the same result set. It's particularly useful for comparing values in the current row with values in preceding rows.

```
SELECT
REST_ID,
CUST_ID,
VISIT_DATE,
LAG(VISIT_DATE,1) OVER(PARTITION BY REST_ID,CUST_ID ORDER BY VISIT_DATE) AS PREVIOUS_VIST_DATE,
NUMBER_OF_PERSONS,
SPENT_AMOUNT,
LAG(SPENT_AMOUNT,1) OVER(PARTITION BY REST_ID,CUST_ID ORDER BY VISIT_DATE) AS PREVIOUS_SPENT INTO
REST_SUMMARY
FROM REST_DATA;
```

56. USE OF WITH CLAUSE IN SQL PROGRAMMING? EXAMPLE

The **WITH** clause, also known as a Common Table Expression (CTE), is a powerful tool in SQL for creating temporary named result sets. These temporary sets can be used within the main query, making complex queries more readable and manageable.

```
WITH TEMP_TABLE(AVG_SALARY) AS (SELECT AVG(SALARY) FROM EMPLOYEE)
SELECT
EMP_ID,
NAME,
SALARY,
AVG_SALARY
FROM EMPLOYEE,TEMP_TABLE
WHERE EMPLOYEE.SALARY>=TEMP_TABLE.AVG_SALARY;
```

57. HOW TO CONCATENATE 2 FIELD VALUES USING DELIMITERS? EXAMPLE

There are two main ways to concatenate (join) two field values using delimiters in SQL:

1. Using the + operator (concatenation operator):

- This is the simplest method, but it has some limitations:
- NULL handling: If any of the fields you're concatenating contain NULL values, the entire result will be NULL.
- Data type conversion: The + operator might implicitly convert data types, which can lead to unexpected results.

```
SQL

SELECT field1 + delimiter + field2 AS combined_field
FROM your_table;
```

58. WHAT IS THE DIFFERENCE BETWEEN PATINDEX AND CHAR INDEX? EXAMPLE

```
2.CHARINDEX: FUNCTION RETIURNS THE SUBSTRING POSITION FROM A SPECIFIED CHARACTER
3.PATINDEX: CANANOT WORK UNDERSCORE(_)

So basically PATINDEX and CHARINDEX both are same but PATINDEX is not working at time of use of
underscore(_).

SELECT 'LAKSHMI_NR' AS NAME,CHARINDEX('_','LAKSHMI_NR') AS DELIMITER_SPACE_POSITION;
SELECT 'LAKSHMI_NR' AS NAME,PATINDEX('%_%','LAKSHMI_NR') AS DELIMITER_SPACE_POSITION;
```

59. WHAT IS SUBSTRING FUNCTION? EXAMPLE

SUBSTRING: RETURN A PORTION OF STRING GIVEN FROM START POINT AND END POINT.
SYNTAX: SUBSTRING (EXPRESSION, STARTING POINT, LENGTH IN INT)

```
SELECT 'INDIA IS A COUNTRY';
SELECT SUBSTRING ('INDIA AS COUNTRY',12,7);
```

60. HOW TO GET FORMAT VALUE IN % FORMAT?

```
SELECT FORMAT(34.5678,'P0',) AS percentage_SALES;
```

61. HOW TO GET NUMERIC VALUE IN INR CURRENCY FORMAT?

```
SELECT FORMAT(1234.5678,'C0','INR-IN') AS SALES;
```

62. HOW TO GET DIFFERENCE BETWEEN 2 DATES IN DAYS, MONTHS AND YEARS GAP? EXAMPLE

```
SELECT '10-15-2010' AS DV_DOB, GETDATE() AS TODAY,
DATEDIFF(YEAR,'10-15-2010',GETDATE()) YEARS_OLD;

SELECT '10-15-2010' AS DV_DOB, GETDATE() AS TODAY,
DATEDIFF(MONTH,'10-15-2010',GETDATE()) MONTH_OLD;

SELECT '10-15-2010' AS DV_DOB, GETDATE() AS TODAY,
DATEDIFF(DAY,'10-15-2010',GETDATE()) DAY_OLD;
```

63. HOW TO ADD YEARS, MONTHS AND DAYS VALUES TO DATE VALUE? EXAMPLE

```
SELECT DATEADD(MONTH,2,CONVERT(DATE,GETDATE())) AS MONTH_ADD;
SELECT DATEADD(YEAR,2,CONVERT(DATE,GETDATE())) AS YEAR_ADD;
SELECT DATEADD(DAY,2,CONVERT(DATE,GETDATE())) AS DAY_ADD;
```

64. HOW TO CONVERT CHARACTER VALUES TO NUMERIC AND NUMERIC VALUES TO CHARACTER IN SQL? EXAMPLE

```
SELECT CAST(sales_code AS INT) AS numeric_code
FROM products;
 SELECT CONVERT(VARCHAR,100) AS SALES; COMPILER CONVERSION
SELECT CAST(100 AS CHAR) AS SALES; PROGRAMMER CONVERSION
```

65. HOW TO GET DATE VALUE FROM CHARACTER DATE VALUES? EXAMPLE

```
SELECT CAST(character_date AS DATE) AS date_value
FROM your_table;
OR

DECLARE @date_string VARCHAR(10) = '04-01-2024';

SELECT CONVERT(
  DATE,
  CAST(SUBSTRING(@date_string, 7, 4) AS INT),   -- Extract year
  CAST(SUBSTRING(@date_string, 1, 2) AS INT),   -- Extract month
  CAST(SUBSTRING(@date_string, 4, 2) AS INT)    -- Extract day
) AS formatted_date;
```

66. WHAT IS USER DEFINED FUNCTIONS? EXAMPLE

User-defined functions (UDFs) in SQL are functions that we create ourself to perform specific tasks or calculations within a database. These functions can encapsulate complex logic and allow us to reuse that logic across multiple queries or procedures. There are two main types of user-defined functions in SQL.

1.TABLE VALUED FUNCTIONS

```
CREATE FUNCTION SEL_GEN(@GEN CHAR(10))
RETURNS TABLE
AS
RETURN
(SELECT
CUSTOMER_ID,
COMPANY,
GENDER,
AGE,
STATE_CODE,
SPENT_AMOUNT
FROM MED_2023
WHERE GENDER=@GEN);

SELECT * FROM SEL_GEN('FEMALE');
```

```sql
SELECT * FROM SEL_GEN('MALE');
```

2.SCALAR VALUED FUNCTIONS:

```sql
SELECT 200+300 AS VALUE;

CREATE FUNCTION ADD_100(@NUM AS INT)
RETURNS DECIMAL
AS
BEGIN
RETURN(@NUM+100)
END;

SELECT [dbo].[ADD_100](1000);

SELECT
CUSTOMER_ID,
COMPANY,
GENDER,
AGE,
STATE_CODE,
SPENT_AMOUNT,
([dbo].[ADD_100](SPENT_AMOUNT)) AS NEW_SPENT
FROM MED_2023;
```

67. WHAT IS A VIEW? EXAMPLE

A view is a logical table based on one or more existing tables. It contains no data of its own but provides a way to access and manipulate data from the base tables.

```sql
CREATE VIEW
MED_2023_SUMMARY_VIEW
AS
SELECT
STATE_CODE,
COMPANY,
GENDER,
COUNT(CUSTOMER_ID) AS SUBS,
SUM(NO_OF_TRIPS) AS VISITS,
SUM(SPENT_AMOUNT) AS TOTAL_SPENT
FROM MED_2023
GROUP BY STATE_CODE,COMPANY,GENDER;
```

68. BENEFITS OF HAVING SQL VIEWS?

- Simplification: Views simplify complex queries and provide a clear, concise way to access data.
- Abstraction: Views abstract underlying table structures, improving data security and privacy.
- Data Security: Views allow for controlled access to specific columns or rows of data, enhancing data security.
- Performance: Views can improve query performance by pre-computing results and avoiding repetitive computations.
- Code Reusability: Views promote code reusability by encapsulating common SQL logic, reducing duplication and improving maintainability.
- Flexibility: Views make it easier to adapt to changes in database schema and facilitate business logic implementation directly in the database.

69. WHAT IS STORE PROCEDURE? EXAMPLE

A stored procedure in SQL is a precompiled set of SQL statements that performs a specific task or a series of tasks. It's stored in the database and can be executed multiple times without needing to recompile the code each time. Stored procedures are often used to encapsulate business logic, improve performance, enhance security, and promote code reusability. They can accept parameters, return values, and be called from various applications or scripts to execute their functionality.

70. BENEFITS OF HAVING SQL STORE PROCEDURE?

Here are the benefits of using SQL stored procedures in brief:

- Code Reusability: Stored procedures encapsulate SQL logic, promoting code reusability across multiple queries or applications.
- Improved Performance: Stored procedures are precompiled and cached, leading to faster execution and reduced database load.

- Enhanced Security: Stored procedures can control access to data and prevent SQL injection attacks by parameterizing inputs.
- Modular Design: Stored procedures facilitate modular design by separating business logic from application code, making maintenance easier.
- Transaction Management: Stored procedures allow for complex transaction management, ensuring data integrity and consistency.
- Reduced Network Traffic: Using stored procedures reduces network traffic by executing logic on the database server rather than transferring large amounts of data to the client.

```sql
CREATE PROCEDURE GetEmployeeByDept
    @DeptName VARCHAR(50)
AS
BEGIN
    SELECT EmpID, EmpName, Salary
    FROM Employees
    WHERE Department = @DeptName;
END;
```

3. Stored procedure with multiple parameters

```sql
CREATE PROCEDURE GetEmployeesBySalary
    @MinSalary DECIMAL(10,2),
    @MaxSalary DECIMAL(10,2)
AS
BEGIN
    SELECT EmpName, Salary
    FROM Employees
    WHERE Salary BETWEEN @MinSalary AND @MaxSalary;
END;
```

5. Stored procedure with IF / business logic

```sql
CREATE PROCEDURE GiveBonus
    @EmpID INT,
    @Bonus DECIMAL(10,2)
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Employees WHERE EmpID = @EmpID)
        UPDATE Employees
        SET Salary = Salary + @Bonus
        WHERE EmpID = @EmpID;
    ELSE
        PRINT 'Employee not found';
END;
```

71. HOW TO DO SQL PROGRAMMING PERFOAMCE TUNING?

- Identify Slow Queries: Use profiling tools to pinpoint queries causing bottlenecks.
- Optimize Queries: Analyze slow queries and apply techniques like proper indexing, avoiding unnecessary full table scans, and using efficient joins.
- Normalize Data Model: Reduce data redundancy and improve relationships between tables.
- Hardware Tuning: Ensure sufficient RAM and CPU resources for your database workload.

- Consider Caching: Utilize query caching or materialized views for frequently used queries.

72. HOW TO CREATE ROW NUMBERS?

```
SELECT
CUSTOMER_ID,
COMPANY,
GENDER,
AGE,
STATE_CODE,
SPENT_AMOUNT,
ROW_NUMBER() OVER(ORDER BY CUSTOMER_ID) AS ROW_NUM
FROM MED_2023;
```

73. HOW TO RETRIVE RANDOM ROW VALUES FROM A TABLE?

In SQL, we can retrieve random row values from a table using the ORDER BY **NEWID()** clause in Microsoft SQL Server or **ORDER BY RANDOM**() in databases like PostgreSQL and SQLite. Here are examples for both cases:

```
SELECT TOP 1 column1, column2, ...
FROM your_table
ORDER BY NEWID();
```

74. HOW TO CREATE A COLUMN VALUES USING RANDOM NUMBERS IN SQL?

In Microsoft SQL Server, you can create a column with random numbers using the **RAND()** function. The **RAND()** function generates a random float value between 0 and 1. If you need integers or random numbers within a specific range, you can use additional functions and calculations. Here's how you can create a column with random numbers in SQL Server:

```
-- Create a table (if not exists)
IF OBJECT_ID('dbo.RandomTable', 'U') IS NULL
BEGIN
    CREATE TABLE RandomTable (
        ID INT PRIMARY KEY,
        RandomFloat FLOAT
    );
END;

-- Insert random float numbers into the table
INSERT INTO RandomTable (ID, RandomFloat)
VALUES
    (1, RAND()),
    (2, RAND()),
    (3, RAND());
```

75. WHAT IS TRIGGER?

A trigger in SQL is a special type of stored procedure that automatically executes in response to certain database events, such **as insertions, updates**, or **deletions** of records in a table. Triggers are used to enforce business rules, maintain data integrity, and automate tasks based on database actions. They can be defined to execute either before or after the triggering event occurs, allowing for actions such as validating data, updating related tables, logging changes, or sending notifications.

76. WHAT IS CURSOR?

A cursor in SQL is a database object that allows us to retrieve and manipulate individual rows of a result set returned by a query. It provides a way to iterate over the rows in a result set one at a time, allowing us to perform operations on each row sequentially. Cursors are typically used in stored procedures or scripts when we need to process rows one by one, perform complex data manipulations, or implement row-level operations that are not easily accomplished using set-based SQL operations. However, it's important to note that cursors can introduce performance overhead and should be used judiciously when other set-based operations are not feasible.