**Summary –Day3**

**Name:Tejaswini Gokanakonda**

## **Roll no**:**DE142**

## **Date:06-11-2024**

**Functions in SQL**

**1. Overview of Functions in SQL**

* SQL has built-in functions to work with data in tables.
* Functions help with tasks like formatting, calculations, and summaries.

**2. Types of Functions**

* **String Functions**: Work with text.
* **Date/Time Functions**: Handle dates and times.
* **Math Functions**: Do calculations.
* **Aggregate Functions**: Summarize data.
* **Ranking Functions**: Give ranks or order to rows.

**3. String Functions (for text)**

* **LEN():** Counts characters, e.g., LEN('hello') gives 5.
* **UPPER() / LOWER():** Changes text to upper/lower case.
* **REPLACE():** Replaces part of a string, e.g., REPLACE('test', 't', 'b') → "besb".
* **REVERSE():** Reverses text, e.g., REVERSE('abc') → "cba".

**4. Date and Time Functions**

* **GETDATE():** Gives today’s date and time.
* **DATEADD():** Adds to date, e.g., DATEADD(day, 10, '2023-01-01').
* **DATEDIFF():** Finds difference, e.g., days between two dates.
* **YEAR(), MONTH(), DAY():** Extracts part of a date.

**5. Math Functions**

* **ABS():** Absolute value, e.g., ABS(-10) → 10.
* **ROUND():** Rounds numbers, e.g., ROUND(4.567, 2) → 4.57.
* **CEILING() and FLOOR():** Rounds up or down.

**6. Aggregate Functions (for summary)**

* **SUM():** Adds up values.
* **AVG():** Average of values.
* **MIN() / MAX():** Finds smallest/largest value.
* **COUNT():** Counts rows.

**7. Ranking Functions**

* **ROW\_NUMBER():** Gives each row a number.
* **RANK():** Ranks rows, with gaps if tied.
* **DENSE\_RANK():** Ranks without gaps.

**Database Schema**

**1. What is a Database?**

* A structured place to store information.
* Can hold simple or complex data.

**2. What is a Database Schema?**

* A schema is the logical structure of a database.
* It shows how data is organized and how tables relate.
* Contains objects like tables, fields ,keys (Primary Key, Foreign Key), and relationships.

**3. Types of Database Schemas**

* **Physical Schema**: How data is stored on disk (like files, indexes).
* **Logical Schema**: Defines tables, columns, and relationships.
* **View Schema**: How end-users see data, focused on presentation.

**4. Key Terms**

* **Primary Key**: Unique identifier for rows in a table.
* **Foreign Key**: Links rows between tables.

**5. Database Models**

* **Flat Model**: Simple table format, no relationships (like a spreadsheet).
* **Hierarchical Model**: Tree structure, one-to-many relationships (parent-child).
* **Network Model**: Like hierarchical but allows many-to-many relationships.
* **Relational Model**: Organizes data into tables with relationships.
* **Star Schema**: Central "fact" table with linked "dimension" tables; used in data warehouses.
* **Snowflake Schema**: Extension of star schema with normalized dimension tables.

**6. Database Schema vs. Database Instance**

* **Schema**: Structure or plan of the database (static).
* **Instance**: Snapshot of the data at a point in time (changes).

**7. Creating Schemas in Different Databases**

* **MySQL**: CREATE SCHEMA is like CREATE DATABASE.
* **Oracle**: CREATE USER defines the schema.
* **SQL Server**: CREATE SCHEMA creates a new schema.

**How to Calculate Subtotals in SQL Queries**

**1. What is a Subtotal?**

* A subtotal is the sum of a specific group of data.
* Commonly used in reports, sales, and accounting to show intermediate totals before the final total.

**2. GROUP BY Extensions for Subtotals**

* SQL has special GROUP BY extensions to help calculate subtotals:
  + **ROLLUP**: Adds subtotals and a grand total for each level in the hierarchy.
  + **GROUPING SETS**: Lets you create custom groupings for multiple subtotals in one query.

**3. Key Functions**

* **GROUPING**: Helps identify subtotal or grand total rows, useful for labeling.

**4. Examples of ROLLUP for Subtotals**

* Basic subtotal by year:

SELECT SalesYear, SUM(SalesTotal) AS TotalSales

FROM SalesData

GROUP BY ROLLUP(SalesYear)

* Subtotal by year and quarter:

SELECT SalesYear, SalesQuarter, SUM(SalesTotal) AS TotalSales

FROM SalesData

GROUP BY ROLLUP(SalesYear, SalesQuarter)

**5. Using CASE with GROUPING**

* Label NULL values in subtotal rows:

SELECT

CASE WHEN GROUPING(SalesYear) = 1 THEN 'Subtotal' ELSE CAST(SalesYear AS VARCHAR) END AS Year,

SalesQuarter,

SUM(SalesTotal) AS TotalSales

FROM SalesData

GROUP BY ROLLUP(SalesYear, SalesQuarter)

**6. Using GROUPING SETS for Flexible Subtotals**

* Allows different groupings in one query without UNION ALL:

SELECT SalesYear, SalesQuarter, SUM(SalesTotal) AS TotalSales

FROM SalesData

GROUP BY GROUPING SETS ((SalesYear), (SalesQuarter), ())

**SQL ORDER OF EXECUTION**

**SQL order of execution:**

SQL order of execution refers to the sequence in which the database processes different parts of a query to retrieve accurate results.

* **Importance:**

Ensures accuracy of query results.

Helps in query optimization for faster performance.

Reduces resource usage on the database server.

Simplifies troubleshooting and modifying complex queries.

* **Order of Execution:**

**FROM:** Identifies tables and accesses necessary data.

**WHERE:** Filters data based on specified conditions.

**GROUP BY:** Groups data and applies aggregation functions.

**HAVING:** Filters grouped data based on conditions.

**SELECT:** Chooses columns for the final result.

**ORDER BY:** Sorts the result set.

**LIMIT/OFFSET:** Limits the number of rows in the output.

* **Application:**

Understanding execution order aids in building efficient queries and enhances readability, especially in complex queries with multiple clauses.

**STORED PROCEDURES**

**Definition:** A stored procedure is reusable SQL code saved in a database to run specific queries or actions.

**Benefits:**

**Efficiency:** Reduces repeated SQL code writing.

**Parameterization:** Accepts parameters, making queries flexible.

**Security**: Controls data access through defined procedures.

* **Syntax for Creation:**

Use CREATE PROCEDURE followed by the procedure name and SQL statement(s).

* **Execution:**

Use EXEC followed by the procedure name to run the stored procedure.

**Example:**

**Procedure to fetch all customer records:**

CREATE PROCEDURE SelectAllCustomers AS SELECT \* FROM Customers;

Run with EXEC SelectAllCustomers;

* **Parameterized Procedures:**

**Single Parameter**: Allows filtering, e.g., by City.

**Example:** CREATE PROCEDURE SelectAllCustomers @City nvarchar(30) AS SELECT \* FROM Customers WHERE City = @City;

Execute with EXEC SelectAllCustomers @City = 'London';

**Multiple Parameters:** Filters by multiple fields, like City and Postal Code.

**Example:** CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10) AS SELECT \* FROM Customers WHERE City = @City AND PostalCode = @PostalCode;

Execute with EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';