**What is a Relational Database / RDBMS?**

* How data is stored in a relational database?
* What is a schema wrt to a relational database?

**SQL commands**

**A diagram of a software system

Description automatically generated**

* DDL
* A screenshot of a computer

  Description automatically generated
* , DML
* A screenshot of a computer

  Description automatically generated
* , DCL, TCL, DQL
* What are commands under each of these category and what each of these commands actually do?

**Data Types**

* String data type like VARCHAR, TEXT etc
* Integer data type like INT, NUMBER etc.
* DATE
* FLOAT / DECIMAL
* BOOLEAN
* Also check out IDENTITY column (Auto Increment column)

**Constraints**

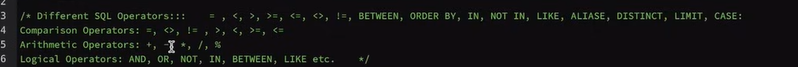
* Primary key
* Foreign key
* Check constraint, Not null constraint, Unique constraint, Default etc.
* A white background with blue text

  Description automatically generated

**Normalization in SQL**

* Different normal forms like 1NF, 2NF, 3NF, BCNF

**Operators**

****

* Arithmetic operator
* Logical operator
* Comparison operator
* UNION, UNION ALL operator
* A screen shot of a computer program

  Description automatically generated

**CASE statement**

* Simple case statement as well nested case statement.
* A screen shot of a computer

  Description automatically generated

**Important SQL clause**

* DISTINCT clause
* Order by clause
* Limit / Top clause
* A screen shot of a computer

  Description automatically generated

**INNER join**

**Union-columns should be same and data type also same,and it will remove duplicate**

**Union all-fetch all data**

* How to fetch data from multiple tables.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Concepts**

**Group By and Having clause**

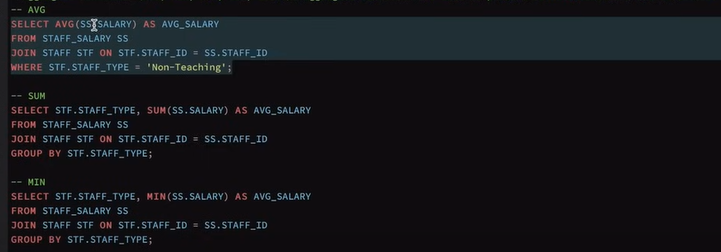
**GroupBy-it groups the data from specific column and it computes the data**

**A screen shot of a computer

Description automatically generated**

**Aggregate functions**

**Group-Group the teaching and nonteaching and claulating**

****

**Order of Execution**

**Sub-Queries**

**A screen shot of a computer

Description automatically generated**

**CTE table / WITH clause**

**All type of Joins**

* LEFT Join, RIGHT Join, FULL OUTER Join
* CROSS Join, SELF Join
* A screenshot of a computer screen

  Description automatically generated
* A screenshot of a computer

  Description automatically generated
* A screen shot of a computer

  Description automatically generated
* A screen shot of a computer program

  Description automatically generated

**In-built functions**

* String functions like Substring, Position, Coalesce etc.
* Date functions like Extract, To\_Date etc.

**Window functions**

* Most important are RANK, DENSE\_RANK, ROW\_NUMBER, LEAD, LAG
* Also good to learn FIRST\_VALUE, LAST\_VALUE, NTH\_VALUE, NTILE

**Advance SQL**

**Concepts**

**Recursive SQL Queries**

**PIVOT table / CROSSTAB function**

**Materialized Views**

1. **Precomputed Data: Materialized views store the result of a query physically, allowing for faster query responses.**
2. **Refresh Mechanism: They can be refreshed to keep the data up-to-date. The refresh can be done periodically or on-demand.**
3. **Use Cases: Ideal for scenarios where queries are complex and involve large datasets, such as data warehousing, reporting, and OLAP (Online Analytical Processing).**

**Creating and Using Materi**

**Creating a Materialized View**

**Suppose we have a table orders and we want to create a materialized view to store the total sales per customer.**

**sql**

**Copy code**

**CREATE TABLE orders (**

**order\_id serial PRIMARY KEY,**

**customer\_id int,**

**order\_date date,**

**amount numeric**

**);**

**INSERT INTO orders (customer\_id, order\_date, amount) VALUES**

**(1, '2023-01-01', 100),**

**(2, '2023-01-02', 200),**

**(1, '2023-01-03', 150);**

**CREATE MATERIALIZED VIEW total\_sales\_per\_customer AS**

**SELECT customer\_id, SUM(amount) AS total\_sales**

**FROM orders**

**GROUP BY customer\_id;**

**REFRESH MATERIALIZED VIEW view\_name;**

**Stored Procedure**

**CREATE PROCEDURE GetUserInfo @inputUserId INT, -- Input parameter @outputUserName NVARCHAR(100) OUTPUT, -- Output parameter @outputUserEmail NVARCHAR(100) OUTPUT -- Output parameter AS BEGIN -- Initialize output parameters (optional) SET @outputUserName = NULL; SET @outputUserEmail = NULL; -- Retrieve user information based on inputUserId SELECT @outputUserName = UserName, @outputUserEmail = UserEmail FROM Users WHERE UserID = @inputUserId; END;**

**DECLARE @UserName NVARCHAR(100), @UserEmail NVARCHAR(100);**

**-- Execute the stored procedure EXEC GetUserInfo @inputUserId = 1,**

**-- Example input UserID @outputUserName = @UserName OUTPUT, --**

**Variable to receive the output UserName @outputUserEmail = @UserEmail OUTPUT;**

**-- Variable to receive the output UserEmail**

**-- Select the output values SELECT @UserName AS UserName, @UserEmail AS UserEmail;**

**User Defined Functions**

 **Scalar Functions**: Return a single value.

A scalar function returns a single value of a specified data type.

**Example: Creating a Scalar Function**

Here’s how you can create a scalar function that calculates the total price after applying a discount:

sql

Copy code

CREATE OR ALTER FUNCTION dbo.CalculateDiscountedPrice(

@OriginalPrice DECIMAL(10, 2),

@DiscountPercentage DECIMAL(5, 2)

)

RETURNS DECIMAL(10, 2)

AS

BEGIN

RETURN @OriginalPrice \* (1 - @DiscountPercentage / 100.0);

END;

**Usage:**

sql

Copy code

SELECT dbo.CalculateDiscountedPrice(100.00, 10.00) AS DiscountedPrice;

 **Table-Valued Functions**: Return a table, which can be queried like a regular table.

Table-Valued Functions

Table-Valued Functions return a table that you can query. They are useful when you need to return a result set from a function.

Inline Table-Valued Functions

An inline table-valued function has a single SELECT statement that defines the table structure and content.

Example: Creating an Inline Table-Valued Function

Here’s how to create an inline table-valued function to retrieve users by their department:

sql

Copy code

CREATE OR ALTER FUNCTION dbo.GetUsersByDepartment(

@DepartmentName NVARCHAR(50)

)

RETURNS TABLE

AS

RETURN

(

SELECT UserID, UserName, UserEmail

FROM Users

WHERE Department = @DepartmentName

);

Usage:

sql

Copy code

SELECT \* FROM dbo.GetUsersByDepartment('Sales');

Multistatement Table-Valued Functions

Multistatement table-valued functions allow for more complex logic and multiple statements to populate the table. They are defined with a RETURN statement that specifies the table variable.

Example: Creating a Multistatement Table-Valued Function

Here’s how you can create a multistatement table-valued function that returns users based on their role with additional processing:

sql

Copy code

CREATE OR ALTER FUNCTION dbo.GetUsersByRole(

@RoleName NVARCHAR(50)

)

RETURNS @UserTable TABLE

(

UserID INT,

UserName NVARCHAR(100),

UserEmail NVARCHAR(100)

)

AS

BEGIN

-- Insert into the table variable

INSERT INTO @UserTable

SELECT UserID, UserName, UserEmail

FROM Users

WHERE Role = @RoleName;

-- Additional processing can be done here if needed

RETURN;

END;

Usage:

sql

Copy code

SELECT \* FROM dbo.GetUsersByRole('Admin');

 **Inline Table-Valued Functions**: Return a table with a single SELECT statement.

select TWEET\_COUNT as tweetbucket,COUNT(user\_id) as usernum from(SELECT USER\_ID,COUNT(1) AS TWEET\_COUNT

FROM tweets

WHERE EXTRACT(YEAR FROM tweet\_date) = 2022

GROUP BY USER\_ID) as total\_tweets

GROUP BY TWEET\_COUNT

Write a query to list the candidates who possess all of the required skills for the job. Sort the output by candidate ID in ascending order.

SELECT candidate\_id

FROM candidates

WHERE skill IN ('Python', 'Tableau', 'PostgreSQL')

GROUP BY candidate\_id

HAVING COUNT(skill)=3

SELECT pages.page\_id

FROM pages

LEFT OUTER JOIN page\_likes AS likes

ON pages.page\_id = likes.page\_id

WHERE likes.page\_id IS NULL;

SELECT page\_id

FROM pages

WHERE page\_id NOT IN (

SELECT page\_id

FROM page\_likes

WHERE page\_id IS NOT NULL

);

SELECT page\_id

FROM pages

WHERE NOT EXISTS (

SELECT page\_id

FROM page\_likes AS likes

WHERE likes.page\_id = pages.page\_id

;)

|  |  |  |
| --- | --- | --- |
| ALL | TRUE if all of the subquery values meet the condition | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_all) |
| AND | TRUE if all the conditions separated by AND is TRUE | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_and) |
| ANY | TRUE if any of the subquery values meet the condition | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_any) |
| BETWEEN | TRUE if the operand is within the range of comparisons | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_between) |
| EXISTS | TRUE if the subquery returns one or more records | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_exists) |
| IN | TRUE if the operand is equal to one of a list of expressions | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in) |
| LIKE | TRUE if the operand matches a pattern | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_like) |
| NOT | Displays a record if the condition(s) is NOT TRUE | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_not) |
| OR | TRUE if any of the conditions separated by OR is TRUE | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_or) |
| SOME | TRUE if any of the subquery values meet the condition |  |

SELECT ProductName

FROM Products

WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

SELECT \* FROM Products

WHERE Price > SOME (SELECT Price FROM Products WHERE Price > 20);

SELECT \* FROM Products

WHERE Price BETWEEN 50 AND 60;