**SQL-Coding Challenge**

**Submitted By-**

**Subrat Shukla, DE Batch1**

**Creating an ‘Employee’ database.**

create database Employee;

use Employee;

--creating employee table

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Position VARCHAR(50),

Department VARCHAR(50),

Salary DECIMAL(10, 2),

Location VARCHAR(50)

);

--inserting data values in employee table

INSERT INTO Employee (EmployeeID, FirstName, LastName, Position, Department, Salary, Location)

VALUES

(101, 'Amit', 'Sharma', 'Software Engineer', 'IT', 60000.00, 'Mumbai'),

(102, 'Priya', 'Singh', 'Data Analyst', 'Analytics', 55000.00, 'Delhi'),

(103, 'Rohan', 'Patel', 'Project Manager', 'Operations', 75000.00, 'Bangalore'),

(104, 'Sonal', 'Gupta', 'HR Specialist', 'HR', 50000.00, 'Chennai'),

(105, 'Vikram', 'Reddy', 'Accountant', 'Finance', 48000.00, 'Hyderabad'),

(106, 'Anita', 'Nair', 'Marketing Exec', 'Marketing', 53000.00, 'Kolkata'),

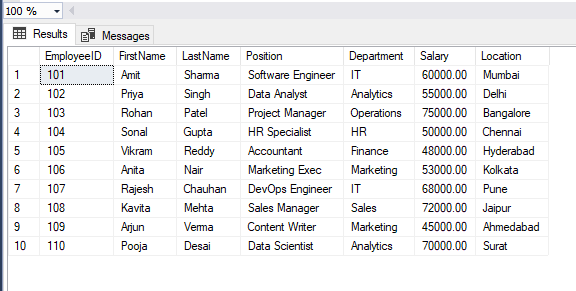
(107, 'Rajesh', 'Chauhan', 'DevOps Engineer', 'IT', 68000.00, 'Pune'),

(108, 'Kavita', 'Mehta', 'Sales Manager', 'Sales', 72000.00, 'Jaipur'),

(109, 'Arjun', 'Verma', 'Content Writer', 'Marketing', 45000.00, 'Ahmedabad'),

(110, 'Pooja', 'Desai', 'Data Scientist', 'Analytics', 70000.00, 'Surat');

select \*from Employee;



**1. Querying Data by Using Joins and Subqueries**

* **Join :** A JOIN in SQL combines rows from two or more tables based on a related column, allowing you to retrieve data from multiple tables in a single query. Different types of joins, like INNER JOIN, LEFT JOIN, and RIGHT JOIN, control how the tables are merged based on matching or non-matching rows.
* **Subquery :** A Subquery is a query nested within another query, which can be used to perform operations like comparisons, calculations, or filters. Subqueries are often used to retrieve data that will be used in the main query for further filtering or aggregation.
* **Subtotal :** A Subtotal is a calculated summary of a subset of data, typically grouped by a specific field, like a department’s total salary or the count of employees in each location. Subtotals help in breaking down large datasets to show summaries for distinct groups within the data.

We have two tables:

* Employee (already defined) and
* DepartmentBudget, which contains the department budgets for each department.

-- creating a table departmentbudget

CREATE TABLE DepartmentBudget (

Department VARCHAR(50),

Budget DECIMAL(10, 2)

);

--inserting data values into it

INSERT INTO DepartmentBudget (Department, Budget)

VALUES

('IT', 150000.00),

('Analytics', 100000.00),

('Operations', 120000.00),

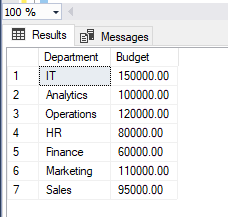
('HR', 80000.00),

('Finance', 60000.00),

('Marketing', 110000.00),

('Sales', 95000.00);

select \*from DepartmentBudget;

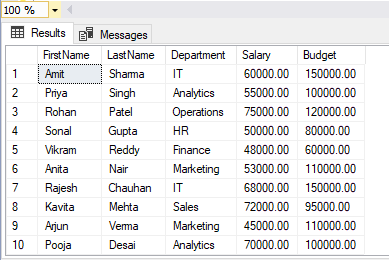


**Query 1.1: Display each employee’s name, department, salary, and the department budget (using a JOIN).**

SELECT e.FirstName, e.LastName, e.Department, e.Salary, d.Budget

FROM Employee e

INNER JOIN DepartmentBudget d ON e.Department = d.Department;

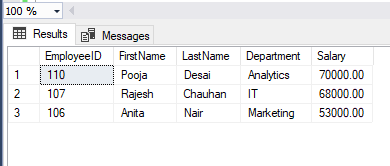


**Query 1.2: List employees whose salary is greater than the average salary of all employees in their department (using a subquery).**

SELECT EmployeeID, FirstName, LastName, Department, Salary

FROM Employee e

WHERE Salary > (SELECT AVG(Salary) FROM Employee WHERE Department = e.Department);



**Query 1.3: Find departments where the total salary of employees is less than the department budget (using a JOIN and GROUP BY).**

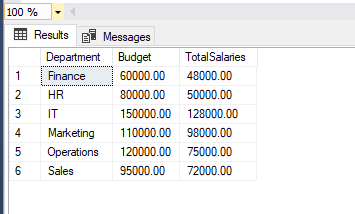
SELECT e.Department, d.Budget, SUM(e.Salary) AS TotalSalaries

FROM Employee e

INNER JOIN DepartmentBudget d ON e.Department = d.Department

GROUP BY e.Department, d.Budget

HAVING SUM(e.Salary) < d.Budget;

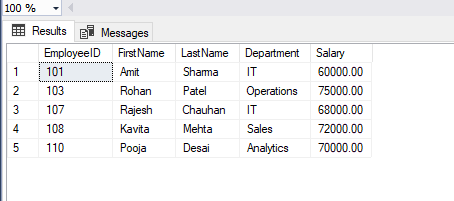


**Query 1.4: Show employees with salaries higher than the average salary of all employees (using a subquery).**

SELECT EmployeeID, FirstName, LastName, Department, Salary

FROM Employee

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

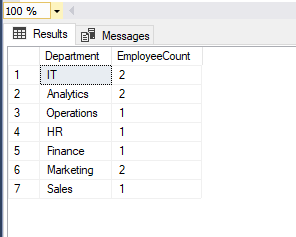


**Query 1.5: Display each department’s name and the number of employees in it (using a subquery to get the count).**

SELECT d.Department, (SELECT COUNT(\*) FROM Employee e

WHERE e.Department = d.Department) AS EmployeeCount

FROM DepartmentBudget d;



**2. Manipulating Data by Using GROUP BY and HAVING Clauses**

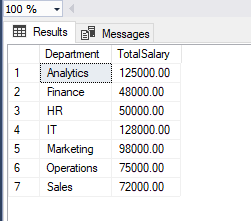
* **GroupBy :** The GROUP BY clause in SQL is used to arrange identical data into groups based on one or more columns, enabling aggregate functions like SUM, COUNT, or AVG to be applied to each group individually. This is particularly useful for generating summaries and reports that categorize data by specific attributes.
* **Having :** The HAVING clause is used in conjunction with GROUP BY to filter groups based on a specified condition, allowing you to include only those groups that meet certain criteria. Unlike the WHERE clause, which filters rows before grouping, HAVING filters groups after the aggregation has been performed.

**Query 2.1: Calculate the total salary expense by department.**

SELECT Department, SUM(Salary) AS TotalSalary

FROM Employee

GROUP BY Department;



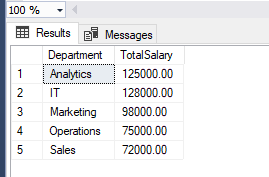
**Query 2.2: Find departments with a total salary expense greater than 70,000.**

SELECT Department, SUM(Salary) AS TotalSalary

FROM Employee

GROUP BY Department

HAVING SUM(Salary) > 70000;

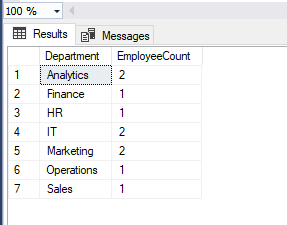


**Query 2.3: Count the number of employees in each department.**

SELECT Department, COUNT(EmployeeID) AS EmployeeCount

FROM Employee

GROUP BY Department;



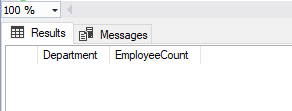
**Query 2.4: Find departments with more than 2 employees.**

SELECT Department, COUNT(EmployeeID) AS EmployeeCount

FROM Employee

GROUP BY Department

HAVING COUNT(EmployeeID) > 2;



showing null, because every department has maximum 2 employees.

**Query 2.5: Find the average salary for each department and display only those with an average salary above 50,000.**

SELECT Department, AVG(Salary) AS AverageSalary

FROM Employee

GROUP BY Department

HAVING AVG(Salary) > 50000;

