# Scenario Series – Tricky case sensitive /insensitive Join output question

Here are the answers for the SQL and PySpark join operations based on the provided tables and the specified considerations:

Table Structure:

| Table 1 | Table 2 |
|---|---|
| Col1 | Col2 |
| A | a |
| a | b |
| a | C |
| B | NULL |
| NULL | NULL |
| c | |

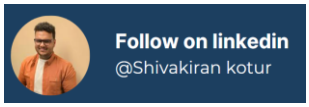
| Joins in SQL and pyspark | No of output? |
|---|---|
| 1 Inner Join | |
| 2 Left Join | |
| 3 Right Join | |
| 4 Full Outer Join | |
| 5 left anti Join | |
| 6 right anti join | |

Here's a complete summary of the results for all join types assuming **case insensitivity** in MS SQL. This includes the correct row counts and reasoning for each join type.

---

- **Inner Join (Case-Insensitive)**
- **Matching Rows**:
  - "A" matches with "a".
  - "a" matches with "a" (two instances).
  - "B" matches with "b".
  - "c" matches with "C".
- **Result**:

| Col1 | Col2 |
|---|---|
| A | a |
| a | a |
| a | a |




Follow me on LinkedIn – Shivakiran kotur

| Col1 | Col2 |
|---|---|
| B | b |
| c | C |

- **Number of Rows**: **5 rows**

---

### Left Join (Case-Insensitive)

- **All Rows from Table 1** are kept. Matching rows are combined, and non-matching rows have Col2 as NULL.
- **Result**:

| Col1 | Col2 |
|---|---|
| A | a |
| a | a |
| a | a |
| B | b |
| NULL | NULL |
| c | C |

- **Number of Rows**: **6 rows**

---

### Right Join (Case-Insensitive)

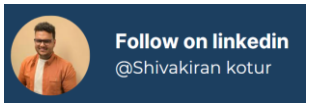
- **All Rows from Table 2** are kept. Matching rows are combined, and non-matching rows have Col1 as NULL.
- **Result**:

| Col1 | Col2 |
|---|---|
| A | a |
| a | a |
| a | a |
| B | b |
| c | C |
| NULL | NULL |
| NULL | NULL |

- **Number of Rows**: **7 rows**

---





## Full Outer Join (Case-Insensitive)

- **All Rows from Both Tables** are kept. Matching rows are combined, and for non-matching rows, NULL is filled in.
- **Result**:

| Col1 | Col2 |
|---|---|
| A | a |
| a | a |
| a | a |
| B | b |
| c | C |
| NULL | NULL |
| NULL | NULL |
| NULL | NULL |

- **Number of Rows**: **8 rows**

---

## Left Anti Join (Case-Insensitive)

- **Rows from Table 1** that do not have a match in Table 2.
- **Result**:

| Col1 |
|---|
| NULL |

- **Number of Rows**: **1 row**

---

## Right Anti Join (Case-Insensitive)

- **Rows from Table 2** that do not have a match in Table 1.
- **Result**:

| Col2 |
|---|
| NULL |
| NULL |

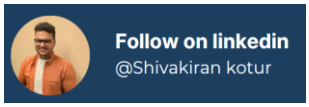
- **Number of Rows**: **2 rows**

---

### Final Summary (Case-Insensitive):

1. **Inner Join**: **5 rows**
2. **Left Join**: **6 rows**
3. **Right Join**: **7 rows**





4. **Full Outer Join: 8 rows**
5. **Left Anti Join: 1 row**
6. **Right Anti Join: 2 rows**

When considering **case sensitivity** in PySpark (the default behavior), the output for each join type changes since "A" and "a" are treated as different values, and "c" and "C" are also different. Below is a detailed explanation for each join type, assuming **case sensitivity**:

---

### Inner Join (Case-Sensitive)

- **How It Joins**:
  - Only rows where the values of Col1 and Col2 match exactly (case-sensitive) are included in the result.
- **Matching Rows**:
  - "a" matches with "a" (two instances).
  - No other matches due to case sensitivity.
- **Result**:

| Col1 | Col2 |
|---|---|
| a | a |
| a | a |

- **Number of Rows**: **2 rows**

---

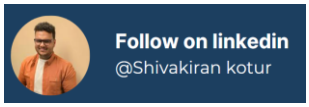
### Left Join (Case-Sensitive)

- **How It Joins**:
  - All rows from Table 1 are kept. Matching rows from Table 2 are combined. If no match is found, Col2 will have NULL values.
- **Result**:

| Col1 | Col2 |
|---|---|
| A | NULL |
| a | a |
| a | a |
| B | NULL |
| NULL | NULL |
| c | NULL |

- **Explanation**:
  - "A" from Table 1 has no match in Table 2 (case-sensitive).





- "a" matches with "a" (two instances).
- "B" from Table 1 has no match.
- "c" from Table 1 has no match because "C" is different.
- The NULL value from Table 1 matches with NULL from Table 2.

- **Number of Rows**: **6 rows**

---

## Right Join (Case-Sensitive)

- **How It Joins**:
  - All rows from Table 2 are kept. Matching rows from Table 1 are combined. If no match is found, Col1 will have NULL values.
- **Result**:

| Col1 | Col2 |
|---|---|
| a | a |
| a | a |
| NULL | b |
| NULL | C |
| NULL | NULL |
| NULL | NULL |

- **Explanation**:
  - "a" from Table 1 matches with "a" from Table 2 (two instances).
  - "b" from Table 2 has no match in Table 1, so Col1 is NULL.
  - "C" from Table 2 has no match because "c" is different.
  - The two NULL values from Table 2 are retained.
- **Number of Rows**: **6 rows**

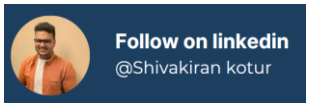
---

## Full Outer Join (Case-Sensitive)

- **How It Joins**:
  - All rows from both tables are kept. Matching rows are combined, and non-matching rows will have NULL values on the side that has no corresponding match.
- **Result**:

| Col1 | Col2 |
|---|---|
| A | NULL |
| a | a |
| a | a |





| Col1 | Col2 |
|---|---|
| B | NULL |
| NULL | b |
| c | NULL |
| NULL | C |
| NULL | NULL |
| NULL | NULL |

- **Explanation**:
  - "A" from Table 1 has no match in Table 2.
  - "a" matches with "a" (two instances).
  - "B" from Table 1 has no match.
  - "b" from Table 2 has no match.
  - "c" from Table 1 has no match because "C" is different.
  - "C" from Table 2 has no match.
  - NULL values from both tables are retained.
- **Number of Rows**: **9 rows**

---

### Left Anti Join (Case-Sensitive)

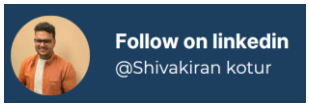
- **How It Joins**:
  - Only rows from Table 1 that do **not** have a matching value in Table 2 are returned.
- **Result**:

| Col1 |
|---|
| A |
| B |
| NULL |
| c |

- **Explanation**:
  - "A" has no match.
  - "B" has no match.
  - NULL from Table 1 has no match with NULL from Table 2 (PySpark does not consider NULL values as equal by default).
  - "c" has no match because "C" is different.
- **Number of Rows**: **4 rows**

---





### Right Anti Join (Case-Sensitive)

- **How It Joins**:
  - Only rows from Table 2 that do **not** have a matching value in Table 1 are returned.
- **Result**:

| Col2 |
| --- |
| b |
| C |
| NULL |
| NULL |

  **Explanation**:
  - "b" has no match.
  - "C" has no match because "c" is different.
  - The two NULL values from Table 2 have no match with the NULL value in Table 1 (PySpark does not consider NULL values as equal by default).
- **Number of Rows**: **4 rows**

---

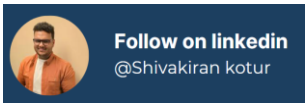
**Final Summary (Case-Sensitive in PySpark):**

1. **Inner Join**: **2 rows**
2. **Left Join**: **6 rows**
3. **Right Join**: **6 rows**
4. **Full Outer Join**: **9 rows**
5. **Left Anti Join**: **4 rows**
6. **Right Anti Join**: **4 rows**

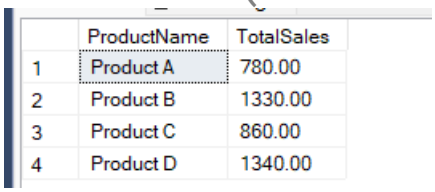
## Scenario Series 13 -- Real-World Applications of Aggregate Functions

```sql
CREATE TABLE Sales (
    SaleID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    SaleAmount DECIMAL(10, 2),
    SaleDate DATE
);

INSERT INTO Sales (SaleID, ProductName, SaleAmount,
SaleDate)
VALUES
(1, 'Product A', 100.00, '2024-01-01'),
(2, 'Product B', 200.00, '2024-01-02'),
(3, 'Product A', 150.00, '2024-01-03'),
(4, 'Product C', 250.00, '2024-01-04'),
(5, 'Product B', 300.00, '2024-01-05'),
(6, 'Product D', 400.00, '2024-01-06'),
(7, 'Product A', 120.00, '2024-01-07'),
(8, 'Product C', 220.00, '2024-01-08'),
(9, 'Product D', 320.00, '2024-01-09'),
(10, 'Product B', 210.00, '2024-01-10'),
(11, 'Product C', 130.00, '2024-01-11'),
(12, 'Product A', 140.00, '2024-01-12'),
(13, 'Product B', 180.00, '2024-01-13'),
(14, 'Product D', 340.00, '2024-01-14'),
(15, 'Product A', 160.00, '2024-01-15'),
(16, 'Product B', 190.00, '2024-01-16'),
(17, 'Product C', 260.00, '2024-01-17'),
(18, 'Product D', 280.00, '2024-01-18'),
(19, 'Product A', 110.00, '2024-01-19'),
(20, 'Product B', 250.00, '2024-01-20');
```



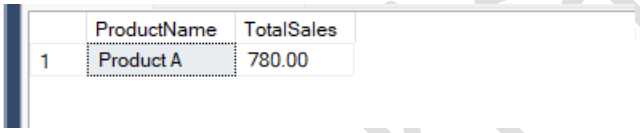


```sql
-- 1. Total Sales per Product - Products with total sales greater than $500
SELECT ProductName, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ProductName
HAVING SUM(SaleAmount) > 500;
```



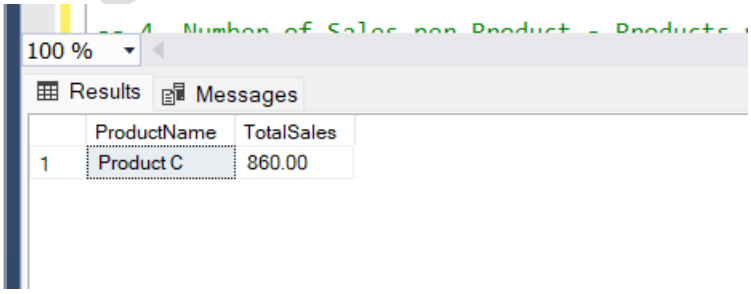
| | ProductName | TotalSales |
|---|---|---|
| 1 | Product A | 780.00 |
| 2 | Product B | 1330.00 |
| 3 | Product C | 860.00 |
| 4 | Product D | 1340.00 |

```sql
-- 2. Total Sales per Product - Products with total sales between $500 and $800
SELECT ProductName, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ProductName
HAVING SUM(SaleAmount) BETWEEN 500 AND 800;
```



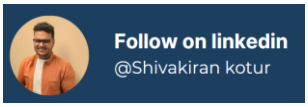
| | ProductName | TotalSales |
|---|---|---|
| 1 | Product A | 780.00 |

```sql
-- 3. Total Sales per Product - Products with total sales exactly $860
SELECT ProductName, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ProductName
HAVING SUM(SaleAmount) = 860;
```



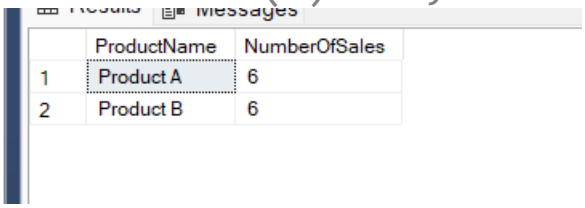

| | ProductName | TotalSales |
|---|---|---|
| 1 | Product C | 860.00 |





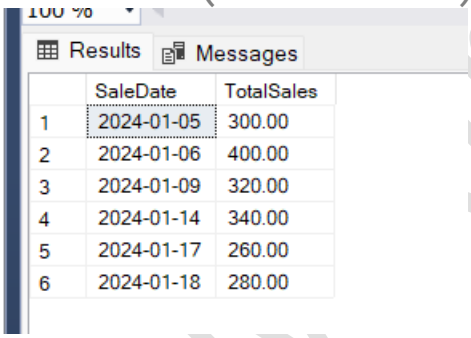

Follow me on LinkedIn – Shivakiran kotur

```sql
-- 4. Number of Sales per Product - Products with more than 5 sales records
SELECT ProductName, COUNT(*) AS NumberOfSales
FROM Sales
GROUP BY ProductName
HAVING COUNT(*) > 5;
```



| | ProductName | NumberOfSales |
|---|---|---|
| 1 | Product A | 6 |
| 2 | Product B | 6 |

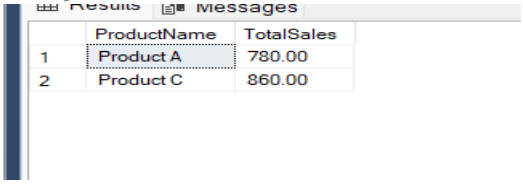
```sql
-- 5. Total Sales per Date - Dates with total sales above $250
SELECT SaleDate, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY SaleDate
HAVING SUM(SaleAmount) > 1000;
```



Results Messages

| | SaleDate | TotalSales |
|---|---|---|
| 1 | 2024-01-05 | 300.00 |
| 2 | 2024-01-06 | 400.00 |
| 3 | 2024-01-09 | 320.00 |
| 4 | 2024-01-14 | 340.00 |
| 5 | 2024-01-17 | 260.00 |
| 6 | 2024-01-18 | 280.00 |

```sql
-- 6. Total Sales per Product - Products with total sales less than $1000 and more than $700
SELECT ProductName, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ProductName
HAVING SUM(SaleAmount) < 1000 AND SUM(SaleAmount) > 700;
```



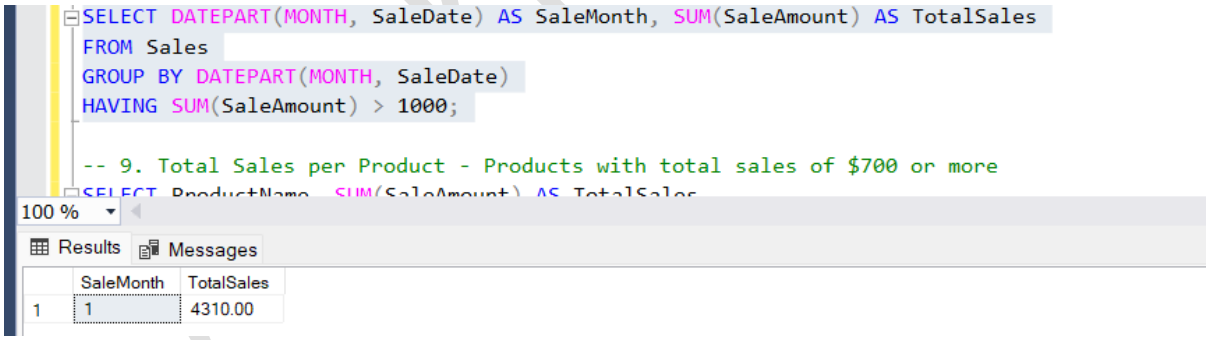
Results Messages

| | ProductName | TotalSales |
|---|---|---|
| 1 | Product A | 780.00 |
| 2 | Product C | 860.00 |

```sql
-- 7. Total Sales per Product - Products with exactly 4 sales records
SELECT ProductName, COUNT(*) AS NumberOfSales, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ProductName
HAVING COUNT(*) = 4;
```

| | ProductName | NumberOfSales | TotalSales |
|---|---|---|---|
| 1 | Product C | 4 | 860.00 |
| 2 | Product D | 4 | 1340.00 |

```sql
-- 8. Total Sales per Month - Months with total sales greater than $1000
SELECT DATEPART(MONTH, SaleDate) AS SaleMonth, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY DATEPART(MONTH, SaleDate)
HAVING SUM(SaleAmount) > 1000;
```



```sql
-- 9. Total Sales per Day of Week - Days of the week with total sales above $500
SELECT DATENAME(WEEKDAY, SaleDate) AS DayOfWeek,
SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY DATENAME(WEEKDAY, SaleDate)
HAVING SUM(SaleAmount) > 500;
```

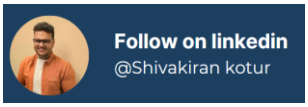
| | DayOfWeek | TotalSales |
|---|---|---|
| 1 | Friday | 550.00 |
| 2 | Saturday | 830.00 |
| 3 | Thursday | 660.00 |
| 4 | Tuesday | 710.00 |
| 5 | Wednesday | 620.00 |

## Scenario Series 11 -- Master SQL Joins and Queries Like Never Before!

```sql
-- Creating Salesman Table
CREATE TABLE Salesman (
    SalesmanID INT PRIMARY KEY,
    SalesmanName VARCHAR(100),
    City VARCHAR(100),
    Commission DECIMAL(5, 2)
);

-- Inserting Records into Salesman Table
INSERT INTO Salesman (SalesmanID, SalesmanName, City,
Commission)
VALUES
(1, 'John Doe', 'New York', 15.00),
(2, 'Jane Smith', 'Los Angeles', NULL),
(3, 'Michael Johnson', 'Chicago', 12.50),
(4, 'Emily Davis', 'New York', 11.00),
(5, 'Chris Brown', 'Chicago', 14.00),
(6, 'Patricia Taylor', 'Los Angeles', 13.00),
(7, 'Daniel Wilson', 'San Francisco', 16.00),
(8, 'Laura Martinez', 'San Francisco', NULL),
(9, 'Kevin Anderson', 'New York', 12.00),
(10, 'Elizabeth Thomas', 'Chicago', 14.50);

-- Creating Customer Table
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100),
    City VARCHAR(100),
    Grade INT
);

-- Inserting Records into Customer Table
INSERT INTO Customer (CustomerID, CustomerName, City,
Grade)
VALUES
(1, 'Alice Green', 'New York', 320),
```

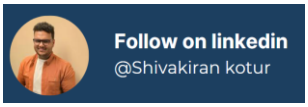
```
(2, 'Bob White', 'Los Angeles', 280),
(3, 'Charlie Black', 'Chicago', 290),
(4, 'David Blue', 'New York', 310),
(5, 'Eve Red', 'San Francisco', NULL),
(6, 'Frank Purple', 'Chicago', 330),
(7, 'Grace Yellow', 'Los Angeles', 310),
(8, 'Hannah Grey', 'San Francisco', 300),
(9, 'Ivy Orange', 'New York', 305),
(10, 'Jack Brown', 'Chicago', 290);

-- Creating Order Table
CREATE TABLE [Order] (
    OrderID INT PRIMARY KEY,
    CustomerID INT NULL,
    SalesmanID INT NULL,
    OrderAmount DECIMAL(10, 2),
    OrderDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES
Customer(CustomerID),
    FOREIGN KEY (SalesmanID) REFERENCES
Salesman(SalesmanID)
);

-- Inserting Records into Order Table
INSERT INTO [Order] (OrderID, CustomerID, SalesmanID,
OrderAmount, OrderDate)
VALUES
(1, 1, 1, 600.00, '2024-01-01'),
(2, 2, 2, 700.00, '2024-01-02'),
(3, 3, 3, 800.00, '2024-01-03'),
(4, 4, 4, 1000.00, '2024-01-04'),
(5, 5, 5, 1200.00, '2024-01-05'),
(6, 6, NULL, 900.00, '2024-01-06'),
(7, NULL, 7, 1100.00, '2024-01-07'),
(8, 8, 8, 500.00, '2024-01-08'),
(9, 9, 9, 1300.00, '2024-01-09'),
(10, 10, 10, 1400.00, '2024-01-10'),
(11, 1, 2, 1500.00, '2024-01-11'),
(12, 2, 3, 1600.00, '2024-01-12'),
(13, 3, 4, 1700.00, '2024-01-13'),
```





```
(14, 4, 5, 1800.00, '2024-01-14'),
(15, 5, 6, 1900.00, '2024-01-15'),
(16, 6, 7, 2000.00, '2024-01-16'),
(17, 7, 8, 2100.00, '2024-01-17'),
(18, 8, 9, 2200.00, '2024-01-18'),
(19, 9, 10, 2300.00, '2024-01-19'),
(20, 10, 1, 2400.00, '2024-01-20');


-- Same City Salesmen and Customers: List salesman and customer names with their cities where both belong to the same city. 🏙️
SELECT s.SalesmanName, s.City AS SalesmanCity, c.CustomerName, c.City AS CustomerCity
FROM Salesman s
JOIN Customer c ON s.City = c.City;

-- Orders Between $500 and $2000: Retrieve order numbers, amounts, customer names, and cities for orders between $500 and $2000. 💵
SELECT o.OrderID, o.OrderAmount, c.CustomerName, c.City
FROM [Order] o
JOIN Customer c ON o.CustomerID = c.CustomerID
WHERE o.OrderAmount BETWEEN 500 AND 2000;

--Salesmen and Their Customers: Identify which salesmen work with which customers. 🛠️
SELECT s.SalesmanName, c.CustomerName
FROM Salesman s
JOIN [Order] o ON s.SalesmanID = o.SalesmanID
JOIN Customer c ON o.CustomerID = c.CustomerID;

High Commission Salesmen: Find customers whose salesmen earn over 12% commission. 📈
SELECT DISTINCT c.CustomerName
FROM Salesman s
JOIN [Order] o ON s.SalesmanID = o.SalesmanID
JOIN Customer c ON o.CustomerID = c.CustomerID
WHERE s.Commission > 12;
```





Different City Salesmen: List customers whose salesmen work in a different city and earn more than 12% commission. 🌍

```sql
SELECT DISTINCT c.CustomerName
FROM Salesman s
JOIN [Order] o ON s.SalesmanID = o.SalesmanID
JOIN Customer c ON o.CustomerID = c.CustomerID
WHERE s.City <> c.City AND s.Commission > 12;
```

Order Details: Get details of each order, including number, date, amount, customer name, and salesman commission. 📋

```sql
SELECT o.OrderID, o.OrderDate, o.OrderAmount,
c.CustomerName, s.Commission
FROM [Order] o
JOIN Customer c ON o.CustomerID = c.CustomerID
JOIN Salesman s ON o.SalesmanID = s.SalesmanID;
```

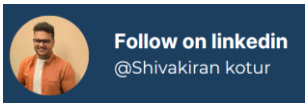
Join All Three Tables: Join Salesman, Customer, and Order tables, ensuring unique columns and relational rows. 🔗

```sql
SELECT o.OrderID, o.OrderDate, o.OrderAmount,
s.SalesmanName, c.CustomerName, s.City AS SalesmanCity,
c.City AS CustomerCity
FROM [Order] o
JOIN Salesman s ON o.SalesmanID = s.SalesmanID
JOIN Customer c ON o.CustomerID = c.CustomerID;
```

Customers With or Without Salesmen: List customers who work with a salesman or independently. 📊

```sql
SELECT c.CustomerName
FROM Customer c
LEFT JOIN [Order] o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerName
HAVING COUNT(o.OrderID) > 0 OR COUNT(o.OrderID) = 0;
```





Customers With Low Grade: Retrieve customers with grades under 300 who work with a salesman or independently.

```sql
SELECT c.CustomerName, c.Grade
FROM Customer c
LEFT JOIN [Order] o ON c.CustomerID = o.CustomerID
WHERE c.Grade < 300;
```

Order Report: Report customer name, city, order number, date, and amount for customers with multiple orders.

```sql
SELECT c.CustomerName, c.City, o.OrderID, o.OrderDate,
o.OrderAmount
FROM Customer c
JOIN [Order] o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerName, c.City, o.OrderID, o.OrderDate,
o.OrderAmount
HAVING COUNT(o.OrderID) > 1;
```

Salesman Report: Show customer name, city, order number, date, amount, salesman name, and commission, highlighting those without orders.

```sql
SELECT c.CustomerName, c.City, o.OrderID, o.OrderDate,
o.OrderAmount, s.SalesmanName, s.Commission
FROM Customer c
LEFT JOIN [Order] o ON c.CustomerID = o.CustomerID
LEFT JOIN Salesman s ON o.SalesmanID = s.SalesmanID
GROUP BY c.CustomerName, c.City, o.OrderID, o.OrderDate,
o.OrderAmount, s.SalesmanName, s.Commission;
```

Salesmen by Customer Orders: List salesmen who work for customers with one or more than three orders.

```sql
SELECT s.SalesmanName
FROM Salesman s
JOIN [Order] o ON s.SalesmanID = o.SalesmanID
GROUP BY s.SalesmanName
HAVING COUNT(o.OrderID) = 1 OR COUNT(o.OrderID) > 3;
```

High-Value Orders: Find salesmen working for customers with orders of $2000+ and a grade, or customers with no orders. 📈

```sql
SELECT DISTINCT s.SalesmanName
FROM Salesman s
JOIN [Order] o ON s.SalesmanID = o.SalesmanID
JOIN Customer c ON o.CustomerID = c.CustomerID
WHERE o.OrderAmount > 2000 AND c.Grade IS NOT NULL
UNION
SELECT DISTINCT s.SalesmanName
FROM Salesman s
LEFT JOIN [Order] o ON s.SalesmanID = o.SalesmanID
WHERE o.OrderID IS NULL;
```

Customers' Orders Report: Report customer names, cities, order numbers, dates, and amounts for those with orders. 📝

```sql
SELECT c.CustomerName, c.City, o.OrderID, o.OrderDate,
o.OrderAmount
FROM Customer c
JOIN [Order] o ON c.CustomerID = o.CustomerID;
```

Grade-Based Orders Report: Report customers with a grade who placed zero orders. 📑

```sql
SELECT c.CustomerName, c.City, c.Grade
FROM Customer c
LEFT JOIN [Order] o ON c.CustomerID = o.CustomerID
WHERE o.OrderID IS NULL AND c.Grade IS NOT NULL;
```

Cartesian Product: Find the count of Cartesian product between all salesmen and customers. 🔄

```sql
SELECT COUNT(*) AS CartesianProductCount
FROM Salesman s
CROSS JOIN Customer c;
```

Cartesian Product by City: Create a Cartesian product between all salesmen and customers belonging to a specific city. 🏙️

```sql
SELECT COUNT(*) AS CartesianProductCount
FROM Salesman s
CROSS JOIN Customer c
WHERE s.City = c.City;
```

Salesmen from City with Grade Customers: Create a Cartesian product between salesmen from a city and customers with a grade. 📊

```sql
SELECT COUNT(*) AS CartesianProductCount
FROM Salesman s
CROSS JOIN Customer c
WHERE s.City = c.City AND c.Grade IS NOT NULL;
```

Different City Salesmen with Grade Customers: Create a Cartesian product between salesmen from a city different from their customers' city, and customers with a grade. 🌐

```sql
SELECT COUNT(*) AS CartesianProductCount
FROM Salesman s
CROSS JOIN Customer c
WHERE s.City <> c.City AND c.Grade IS NOT NULL;
```

Advanced Cartesian Product: Create a Cartesian product between all salesmen and customers where salesmen belong to a different city from their customers, and customers should not have a grade.

```sql
SELECT COUNT(*) AS CartesianProductCount
FROM Salesman s
CROSS JOIN Customer c
WHERE s.City <> c.City AND c.Grade IS NULL;
```

# Scenario Series 12 -- Real-World Applications of String and Date Functions

```sql
-- Create the Employee table
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Position VARCHAR(50),
    Department VARCHAR(50),
    HireDate DATE,
    Salary DECIMAL(10, 2),
    Email VARCHAR(100)
);

-- Insert data into the Employee table
INSERT INTO Employee (EmployeeID, FirstName, LastName, Position, Department, HireDate,
Salary, Email)
VALUES
(1, 'John', 'Doe', 'Manager', 'Sales', '2018-01-15', 75000, 'john.doe@example.com'),
(2, 'Jane', 'Smith', 'Developer', 'IT', '2019-03-22', 85000,
'jane.smith@example.com'),
(3, 'Alice', 'Brown', 'Analyst', 'Finance', '2020-07-19', 65000,
'alice.brown@example.com'),
(4, 'Bob', 'Davis', 'Consultant', 'HR', '2021-05-10', 70000, 'bob.davis@example.com'),
(5, 'Eve', 'White', 'Assistant', 'Admin', '2017-12-05', 45000,
'eve.white@example.com');
```

1. 👥 **Full Name Display**: How can you display the full name of each employee by combining their first and last names?

```sql
-- CONCAT() - Display full name of each employee
SELECT EmployeeID, CONCAT(FirstName, ' ', LastName) AS FullName
FROM Employee;
```

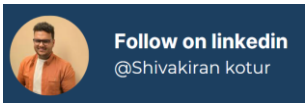
100 %

Results | Messages

| | EmployeeID | FullName |
|---|---|---|
| 1 | 1 | John Doe |
| 2 | 2 | Jane Smith |
| 3 | 3 | Alice Brown |
| 4 | 4 | Bob Davis |
| 5 | 5 | Eve White |







## 2. ✂️ **Substring Extraction**:

⚡ How would you extract the first three characters of an employee's first name?

```
-- SUBSTRING() - Extract first three characters of each employee's first name
SELECT EmployeeID, SUBSTRING(FirstName, 1, 3) AS FirstNamePart
FROM Employee;
```

100 %

Results | Messages

| | EmployeeID | FirstNamePart |
|---|---|---|
| 1 | 1 | Joh |
| 2 | 2 | Jan |
| 3 | 3 | Ali |
| 4 | 4 | Bob |
| 5 | 5 | Eve |

⚡ How could you extract the last three characters of a job title?

```
-- SUBSTRING() - Extract last three characters of the position title
SELECT EmployeeID, SUBSTRING(Position, LEN(Position) - 2, 3) AS PositionEnd
FROM Employee;
```

100 %

Results | Messages

| | EmployeeID | PositionEnd |
|---|---|---|
| 1 | 1 | ger |
| 2 | 2 | per |
| 3 | 3 | yst |
| 4 | 4 | ant |
| 5 | 5 | ant |

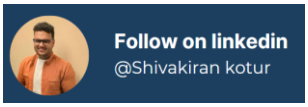
⚡ How can you pull a department name starting from the second character?

```
-- SUBSTRING() - Extract department name starting from the second character
SELECT EmployeeID, SUBSTRING(Department, 2, LEN(Department) - 1) AS DepartmentPart
FROM Employee;
```

100 %

Results | Messages

| | EmployeeID | DepartmentPart |
|---|---|---|
| 1 | 1 | ales |
| 2 | 2 | T |
| 3 | 3 | inance |
| 4 | 4 | R |
| 5 | 5 | dmin |




Follow me on LinkedIn – Shivakiran kotur

3. **Name Length Check**: How would you find the length of each employee's first name to ensure it meets specific criteria?

```sql
-- LENGTH() - Retrieve the length of each employee's first name
SELECT EmployeeID, FirstName, LEN(FirstName) AS FirstNameLength
FROM Employee;
```

100 %

Results | Messages

| | EmployeeID | FirstName | FirstNameLength |
|---|---|---|---|
| 1 | 1 | John | 4 |
| 2 | 2 | Jane | 4 |
| 3 | 3 | Alice | 5 |
| 4 | 4 | Bob | 3 |
| 5 | 5 | Eve | 3 |

4. **Case Conversion**:

⚡ How can you convert first names to uppercase for consistency?

⚡ How would you convert last names to lowercase, perhaps for generating email addresses?

```sql
-- UPPER()/LOWER() - Convert first name to uppercase and last name to lowercase
SELECT EmployeeID, FirstName, UPPER(FirstName) AS FirstNameUpper, LastName, LOWER(LastName) AS LastNameLower
FROM Employee;
```

100 %

Results | Messages

| | EmployeeID | FirstName | FirstNameUpper | LastName | LastNameLower |
|---|---|---|---|---|---|
| 1 | 1 | John | JOHN | Doe | doe |
| 2 | 2 | Jane | JANE | Smith | smith |
| 3 | 3 | Alice | ALICE | Brown | brown |
| 4 | 4 | Bob | BOB | Davis | davis |
| 5 | 5 | Eve | EVE | White | white |

5. **Text Replacement**:

⚡ How would you update all occurrences of the word "Manager" to "Lead" in job titles?

⚡ How could you replace "IT" with "Technology" in department names?

⚡ How would you replace spaces with hyphens in full names for URL-friendly formatting?

```sql
-- REPLACE() - Replace 'Manager' with 'Lead' in the position title , REPLACE() - Replace 'IT' with 'Technology' in the department name,
--REPLACE() - Replace ' ' with '-' in the full name
SELECT EmployeeID,  REPLACE(CONCAT(FirstName, ' ', LastName), ' ', '-') AS FullNameWithDash,
REPLACE(Position, 'Manager', 'Lead') AS NewPosition , REPLACE(Department, 'IT', 'Technology') AS NewDepartment
FROM Employee;
```

100 %

Results | Messages

| | EmployeeID | FullNameWithDash | NewPosition | NewDepartment |
|---|---|---|---|---|
| 1 | 1 | John-Doe | Lead | Sales |
| 2 | 2 | Jane-Smith | Developer | Technology |
| 3 | 3 | Alice-Brown | Analyst | Finance |
| 4 | 4 | Bob-Davis | Consultant | HR |
| 5 | 5 | Eve-White | Assistant | Admin |

6. **Email Masking**: What's the best way to mask an email address so that only the first three characters and the domain are visible, keeping the rest private?

```sql
SELECT EmployeeID,
       Email,
       STUFF(Email, 4, CHARINDEX('@', Email) - 4, REPLICATE('*', CHARINDEX('@', Email) - 4)) AS MaskedEmail
FROM Employee;
```

%

Results | Messages

| EmployeeID | Email | MaskedEmail |
|---|---|---|
| 1 | john.doe@example.com | joh*****@example.com |
| 2 | jane.smith@example.com | jan*******@example.com |
| 3 | alice.brown@example.com | ali********@example.com |
| 4 | bob.davis@example.com | bob******@example.com |
| 5 | eve.white@example.com | eve******@example.com |

**STUFF(Email, 4, CHARINDEX('@', Email) - 4, REPLICATE('*', CHARINDEX('@', Email) - 4)):**

- Email: The original email address.
- 4: The starting position where masking begins (the 4th character).
- CHARINDEX('@', Email) - 4: The length of the string to be replaced, which is the number of characters between the 3rd position and the @ symbol.
- REPLICATE('*', CHARINDEX('@', Email) - 4): The replacement string, which is a series of asterisks (*) to mask the characters.

## **Date Management**:

⚡ How would you extract just the date from a hire date field?

```
-- ⚡ Extract Just the Date: Get only the date part from a HireDate field
SELECT EmployeeID, HireDate, CAST(HireDate AS DATE) AS HireDateOnly
FROM Employee;
```

100 %

Results | Messages

| | EmployeeID | HireDate | HireDateOnly |
|---|---|---|---|
| 1 | 1 | 2018-01-15 | 2018-01-15 |
| 2 | 2 | 2019-03-22 | 2019-03-22 |
| 3 | 3 | 2020-07-19 | 2020-07-19 |
| 4 | 4 | 2021-05-10 | 2021-05-10 |
| 5 | 5 | 2017-12-05 | 2017-12-05 |

⚡ How can you break down the hire date into year, month, and day components?

```
-- ⚡ Break Down Hire Date: Break down the HireDate into year, month, and day components
SELECT EmployeeID, HireDate,
       YEAR(HireDate) AS HireYear,
       MONTH(HireDate) AS HireMonth,
       DAY(HireDate) AS HireDay
FROM Employee;
```

100 %

Results | Messages

| | EmployeeID | HireDate | HireYear | HireMonth | HireDay |
|---|---|---|---|---|---|
| 1 | 1 | 2018-01-15 | 2018 | 1 | 15 |
| 2 | 2 | 2019-03-22 | 2019 | 3 | 22 |
| 3 | 3 | 2020-07-19 | 2020 | 7 | 19 |
| 4 | 4 | 2021-05-10 | 2021 | 5 | 10 |
| 5 | 5 | 2017-12-05 | 2017 | 12 | 5 |

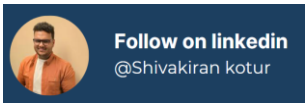
⚡ What's the best way to calculate the number of days an employee has been with the company?

```
-- ⚡ Calculate Days with Company: Calculate the number of days an employee has been with the company
SELECT EmployeeID, HireDate, DATEDIFF(DAY, HireDate, GETDATE()) AS DaysWithCompany
FROM Employee;
```

100 %

Results | Messages

| | EmployeeID | HireDate | DaysWithCompany |
|---|---|---|---|
| 1 | 1 | 2018-01-15 | 2416 |
| 2 | 2 | 2019-03-22 | 1985 |
| 3 | 3 | 2020-07-19 | 1500 |
| 4 | 4 | 2021-05-10 | 1205 |
| 5 | 5 | 2017-12-05 | 2457 |




Follow me on LinkedIn – Shivakiran kotur

## How would you add a year to the hire date for tracking anniversaries?

```sql
-- Add a Year to Hire Date: Add a year to the HireDate for tracking anniversaries
SELECT EmployeeID, HireDate, DATEADD(YEAR, 1, HireDate) AS NextAnniversary
FROM Employee;
```

100 %

Results | Messages

| | EmployeeID | HireDate | NextAnniversary |
|---|---|---|---|
| 1 | 1 | 2018-01-15 | 2019-01-15 |
| 2 | 2 | 2019-03-22 | 2020-03-22 |
| 3 | 3 | 2020-07-19 | 2021-07-19 |
| 4 | 4 | 2021-05-10 | 2022-05-10 |
| 5 | 5 | 2017-12-05 | 2018-12-05 |

## Find Employees Hired in a Specific Month (e.g., January)

```sql
-- Find Employees Hired in a Specific Month: Find employees hired in January
SELECT EmployeeID, FirstName, LastName, HireDate
FROM Employee
WHERE MONTH(HireDate) = 1; -- For January
```

100 %

Results | Messages

| | EmployeeID | FirstName | LastName | HireDate |
|---|---|---|---|---|
| 1 | 1 | John | Doe | 2018-01-15 |

## Determine the Weekday of the Hire Date

```sql
-- Determine the Weekday of Hire Date: Find out the weekday name of the HireDate
SELECT EmployeeID, HireDate, DATENAME(WEEKDAY, HireDate) AS HireDayOfWeek
FROM Employee;
```

100 %

Results | Messages

| | EmployeeID | HireDate | HireDayOfWeek |
|---|---|---|---|
| 1 | 1 | 2018-01-15 | Monday |
| 2 | 2 | 2019-03-22 | Friday |
| 3 | 3 | 2020-07-19 | Sunday |
| 4 | 4 | 2021-05-10 | Monday |
| 5 | 5 | 2017-12-05 | Tuesday |

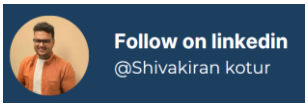
## Calculate the Number of Weeks Since the Hire Date

```sql
-- Calculate Number of Weeks Since Hire Date: Calculate the number of weeks an employee has been with the company
SELECT EmployeeID, HireDate, DATEDIFF(WEEK, HireDate, GETDATE()) AS WeeksWithCompany
FROM Employee;
```

100 %

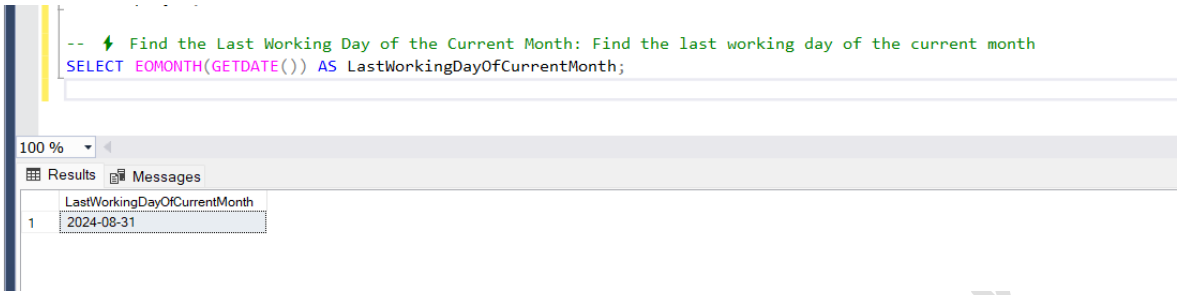
Results | Messages

| | EmployeeID | HireDate | WeeksWithCompany |
|---|---|---|---|
| 1 | 1 | 2018-01-15 | 345 |
| 2 | 2 | 2019-03-22 | 284 |
| 3 | 3 | 2020-07-19 | 214 |
| 4 | 4 | 2021-05-10 | 172 |
| 5 | 5 | 2017-12-05 | 351 |




Follow me on LinkedIn – Shivakiran kotur

## Find the Last Working Day of the Current Month



```
-- Find the Last Working Day of the Current Month: Find the last working day of the current month
SELECT EOMONTH(GETDATE()) AS LastWorkingDayOfCurrentMonth;
```

100 %

Results | Messages

| | LastWorkingDayOfCurrentMonth |
|---|---|
| 1 | 2024-08-31 |

## Scenario series 6 →Interview questions→ Work with different join in SQL

- You have two tables, table1 and table2, representing data collected from two different sources. You want to combine this data to analyse how the entries from these sources relate to each other.
- The goal is to understand various types of joins and how they impact the data returned from the tables.

**table1**

| col1 |
|---|
| 1 |
| 2 |
| 3 |
| 1 |
| 1 |
| Null |
| 2 |
| Null |

**table2**

| col2 |
|---|
| 1 |
| 2 |
| 3 |
| Null |
| 4 |

## Inner Join

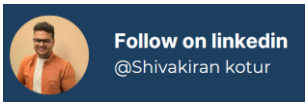
**Returns only the rows with matching values in both tables.**

```
SELECT t1.col1, t2.col2
FROM table1 t1
INNER JOIN table2 t2 ON t1.col1 = t2.col2;
```

**Result:**

| col1 | col2 |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 1 | 1 |
| 1 | 1 |
| 2 | 2 |





## Left Join (Left Outer Join)

**Returns all rows from the left table (table1), and the matched rows from the right table (table2). If there is no match, NULL values are returned for columns from the right table.**

```
SELECT t1.col1, t2.col2
FROM table1 t1
LEFT JOIN table2 t2 ON t1.col1 = t2.col2;
```

**Result:**

| col1 | col2 |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 1 | 1 |
| 1 | 1 |
| Null | Null |
| 2 | 2 |
| Null | Null |

## Right Join (Right Outer Join)

**Returns all rows from the right table (table2), and the matched rows from the left table (table1). If there is no match, NULL values are returned for columns from the left table.**

```
SELECT t1.col1, t2.col2
FROM table1 t1
RIGHT JOIN table2 t2 ON t1.col1 = t2.col2;
```

**Result:**

| col1 | col2 |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 1 | 1 |
| 1 | 1 |
| Null | Null |
| Null | 4 |

## Full Outer Join

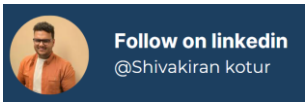
**Returns all rows when there is a match in either left (table1) or right (table2) table. If there is no match, NULL values are returned for the columns that don't have a matching row.**

```
SELECT t1.col1, t2.col2
FROM table1 t1
FULL OUTER JOIN table2 t2 ON t1.col1 = t2.col2;
```

**Result:**

| col1 | col2 |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 1 | 1 |
| 1 | 1 |
| Null | Null |
| Null | 4 |
| Null | Null |





## Cross Join

**Returns the Cartesian product of both tables, i.e., it returns all possible combinations of rows from table1 and table2.**

```
SELECT t1.col1, t2.col2
FROM table1 t1
CROSS JOIN table2 t2;
```

**Result:**

| col1 | col2 |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 1 | Null |
| 1 | 4 |
| 2 | 1 |
| 2 | 2 |
| 2 | 3 |
| 2 | Null |
| 2 | 4 |
| 3 | 1 |
| 3 | 2 |
| 3 | 3 |
| 3 | Null |
| 3 | 4 |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 1 | Null |
| 1 | 4 |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |

| col1 | col2 |
|---|---|
| 1 | Null |
| 1 | 4 |
| Null | 1 |
| Null | 2 |
| Null | 3 |
| Null | Null |
| Null | 4 |
| 2 | 1 |
| 2 | 2 |
| 2 | 3 |
| 2 | Null |
| 2 | 4 |
| Null | 1 |
| Null | 2 |
| Null | 3 |
| Null | Null |
| Null | 4 |

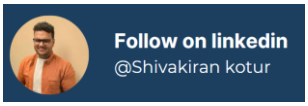
## Self Join

**A Self Join is a regular join, but the table is joined with itself. This can be useful when comparing rows within the same table.**

Let's compare the values in table1 with each other to find rows where the value in col1 is the same:

```
SELECT t1.col1 AS col1_a, t2.col1 AS col1_b
FROM table1 t1
INNER JOIN table1 t2 ON t1.col1 = t2.col1;
```





**Result:**

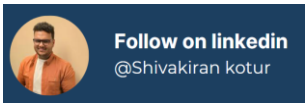
| col1_a | col1_b |
|---|---|
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 2 | 2 |
| 2 | 2 |
| 3 | 3 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |

This setup provides a comprehensive understanding of how different SQL joins work and their results using table1 and table2.

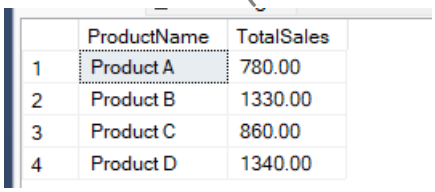
## Scenario Series 13 -- Real-World Applications of Aggregate Functions

```sql
CREATE TABLE Sales (
    SaleID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    SaleAmount DECIMAL(10, 2),
    SaleDate DATE
);

INSERT INTO Sales (SaleID, ProductName, SaleAmount,
SaleDate)
VALUES
(1, 'Product A', 100.00, '2024-01-01'),
(2, 'Product B', 200.00, '2024-01-02'),
(3, 'Product A', 150.00, '2024-01-03'),
(4, 'Product C', 250.00, '2024-01-04'),
(5, 'Product B', 300.00, '2024-01-05'),
(6, 'Product D', 400.00, '2024-01-06'),
(7, 'Product A', 120.00, '2024-01-07'),
(8, 'Product C', 220.00, '2024-01-08'),
(9, 'Product D', 320.00, '2024-01-09'),
(10, 'Product B', 210.00, '2024-01-10'),
(11, 'Product C', 130.00, '2024-01-11'),
(12, 'Product A', 140.00, '2024-01-12'),
(13, 'Product B', 180.00, '2024-01-13'),
(14, 'Product D', 340.00, '2024-01-14'),
(15, 'Product A', 160.00, '2024-01-15'),
(16, 'Product B', 190.00, '2024-01-16'),
(17, 'Product C', 260.00, '2024-01-17'),
(18, 'Product D', 280.00, '2024-01-18'),
(19, 'Product A', 110.00, '2024-01-19'),
(20, 'Product B', 250.00, '2024-01-20');
```



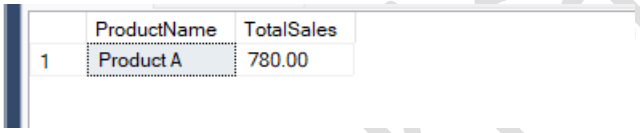


```sql
-- 1. Total Sales per Product - Products with total sales greater than $500
SELECT ProductName, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ProductName
HAVING SUM(SaleAmount) > 500;
```



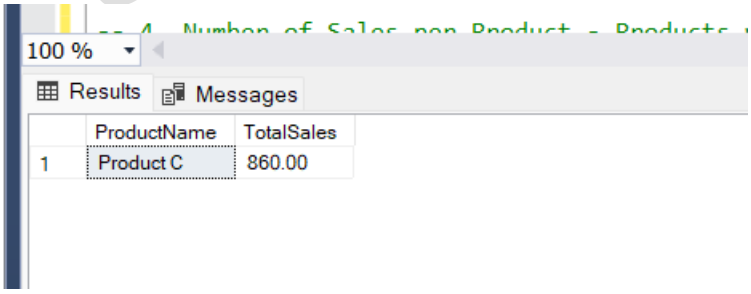
| | ProductName | TotalSales |
|---|---|---|
| 1 | Product A | 780.00 |
| 2 | Product B | 1330.00 |
| 3 | Product C | 860.00 |
| 4 | Product D | 1340.00 |

```sql
-- 2. Total Sales per Product - Products with total sales between $500 and $800
SELECT ProductName, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ProductName
HAVING SUM(SaleAmount) BETWEEN 500 AND 800;
```



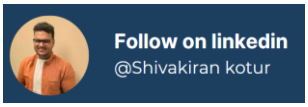
| | ProductName | TotalSales |
|---|---|---|
| 1 | Product A | 780.00 |

```sql
-- 3. Total Sales per Product - Products with total sales exactly $860
SELECT ProductName, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ProductName
HAVING SUM(SaleAmount) = 860;
```



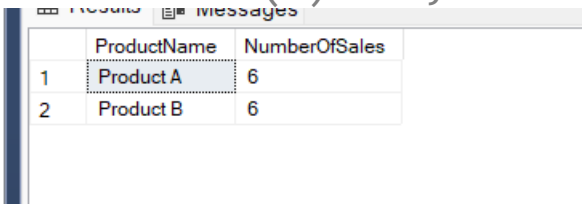

| | ProductName | TotalSales |
|---|---|---|
| 1 | Product C | 860.00 |





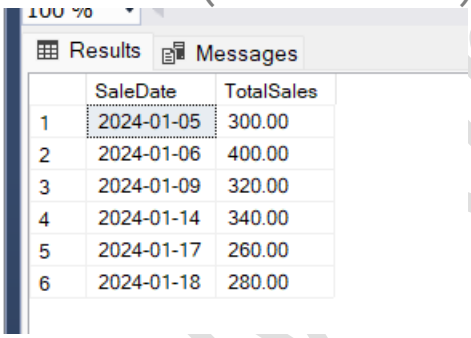

Follow me on LinkedIn – Shivakiran kotur

```sql
-- 4. Number of Sales per Product - Products with more than 5 sales records
SELECT ProductName, COUNT(*) AS NumberOfSales
FROM Sales
GROUP BY ProductName
HAVING COUNT(*) > 5;
```



| | ProductName | NumberOfSales |
|---|---|---|
| 1 | Product A | 6 |
| 2 | Product B | 6 |

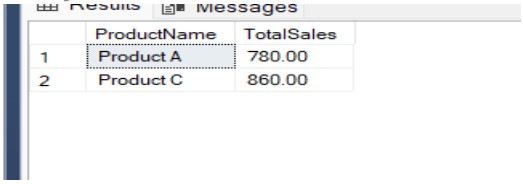
```sql
-- 5. Total Sales per Date - Dates with total sales above $250
SELECT SaleDate, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY SaleDate
HAVING SUM(SaleAmount) > 1000;
```



Results Messages

| | SaleDate | TotalSales |
|---|---|---|
| 1 | 2024-01-05 | 300.00 |
| 2 | 2024-01-06 | 400.00 |
| 3 | 2024-01-09 | 320.00 |
| 4 | 2024-01-14 | 340.00 |
| 5 | 2024-01-17 | 260.00 |
| 6 | 2024-01-18 | 280.00 |

```sql
-- 6. Total Sales per Product - Products with total sales less than $1000 and more than $700
SELECT ProductName, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ProductName
HAVING SUM(SaleAmount) < 1000 AND SUM(SaleAmount) > 700;
```



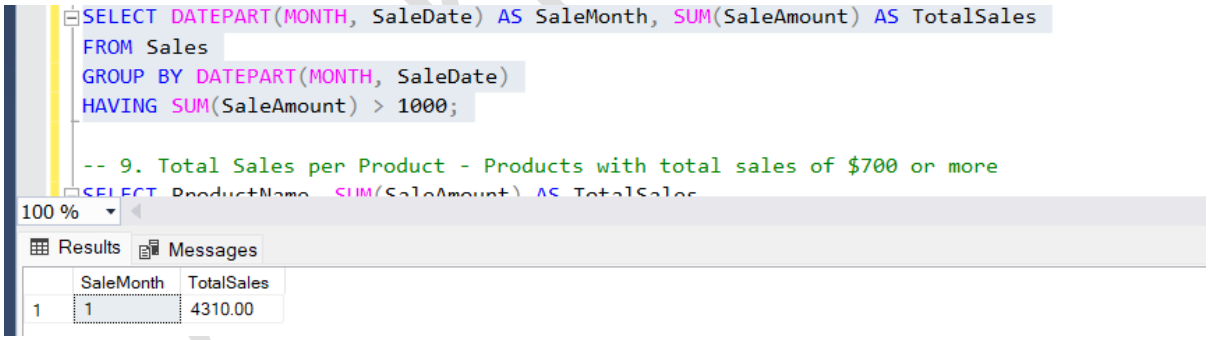
Results Messages

| | ProductName | TotalSales |
|---|---|---|
| 1 | Product A | 780.00 |
| 2 | Product C | 860.00 |

```sql
-- 7. Total Sales per Product - Products with exactly 4 sales records
SELECT ProductName, COUNT(*) AS NumberOfSales, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ProductName
HAVING COUNT(*) = 4;
```

| | ProductName | NumberOfSales | TotalSales |
|---|---|---|---|
| 1 | Product C | 4 | 860.00 |
| 2 | Product D | 4 | 1340.00 |

```sql
-- 8. Total Sales per Month - Months with total sales greater than $1000
SELECT DATEPART(MONTH, SaleDate) AS SaleMonth, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY DATEPART(MONTH, SaleDate)
HAVING SUM(SaleAmount) > 1000;
```



```sql
-- 9. Total Sales per Day of Week - Days of the week with total sales above $500
SELECT DATENAME(WEEKDAY, SaleDate) AS DayOfWeek,
SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY DATENAME(WEEKDAY, SaleDate)
HAVING SUM(SaleAmount) > 500;
```

| | DayOfWeek | TotalSales |
|---|---|---|
| 1 | Friday | 550.00 |
| 2 | Saturday | 830.00 |
| 3 | Thursday | 660.00 |
| 4 | Tuesday | 710.00 |
| 5 | Wednesday | 620.00 |

# Scenario series 8 →Interview questions→ Where vs Having Clause

### 🔥 SQL Showdown: `WHERE` vs. `HAVING` – Which One to Use and When? 🔥

Ever found yourself puzzled over when to use `WHERE` and when to reach for `HAVING`? Let's dive into the **epic battle** of these two SQL clauses with a simple example! ⚔️

## 🛠️ The Setup:

You've got a `Sales` table recording your product sales:

```
Create table Sales (Product nvarchar(50), SaleAmount int )

Insert into Sales values ('iPhone', 500)
Insert into Sales values ('Laptop', 800)
Insert into Sales values ('iPhone', 1000)
Insert into Sales values ('Speakers', 400)
Insert into Sales values ('Laptop', 600)

SELECT * FROM SALES
```

| | Product | SaleAmount |
|---|---|---|
| 1 | iPhone | 500 |
| 2 | Laptop | 800 |
| 3 | iPhone | 1000 |
| 4 | Speakers | 400 |
| 5 | Laptop | 600 |

### 🥊 Round 1: The Basics

Let's start by grouping sales by product:

```
SELECT PRODUCT, SUM(SALEAMOUNT) FROM Sales GROUP BY PRODUCT
```

| | PRODUCT | (No column name) |
|---|---|---|
| 1 | iPhone | 1500 |
| 2 | Laptop | 1400 |
| 3 | Speakers | 400 |

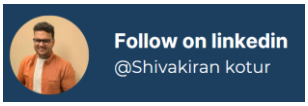
This gives us the total sales for each product. But what if you only want to see products with total sales greater than 1000?

### 🥊 Round 2: Enter HAVING

Here's where HAVING comes into play:

```
SELECT PRODUCT,SUM (SALEAMOUNT) FROM SALES GROUP BY PRODUCT HAVING
SUM(SALEAMOUNT) > 1000
```

Follow me on LinkedIn – Shivakiran kotur



| | PRODUCT | (No column name) |
|---|---|---|
| 1 | iPhone | 1500 |
| 2 | Laptop | 1400 |

✨ **Why `HAVING`?** It lets you filter after the data has been grouped and aggregated. Perfect for conditions involving `SUM()`, `COUNT()`, and other aggregate functions!

### 🥊 Round 3: The WHERE Dilemma

Now, what if you try to use WHERE instead?

```
SELECT PRODUCT,SUM(SALEAMOUNT) FROM SALES
GROUP BY PRODUCT WHERE SUM(SALEAMOUNT) > 1000
```

Messages

```
Msg 156, Level 15, State 1, Line 35
Incorrect syntax near the keyword 'WHERE'.
```

🚫 **Oops!** `WHERE` can't be used with aggregate functions because it filters **before** the aggregation. Lesson learned! 🎓

### 🥊 Round 4: `WHERE` vs. `HAVING` in Action

Let's see these two in a real-world scenario:

**Using `WHERE`:** You want to calculate total sales for just 'iPhone' and 'Speakers':

iPhone and Speaker products and then performs the sum.

```
SELECT PRODUCT, SUM(SALEAMOUNT) AS SALES FROM SALES
WHERE PRODUCT IN ('iPhone', 'Speakers')
GROUP BY PRODUCT
```

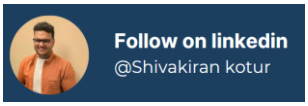
| | PRODUCT | SALES |
|---|---|---|
| 1 | iPhone | 1500 |
| 2 | Speakers | 400 |

- **`WHERE` filters individual rows before grouping.**

**Using `HAVING`:** Achieve the same but with more control:

**Calculate Total sales of iPhone and Speakers using HAVING clause:** This example retrieves all rows from Sales table, performs the sum and then removes all products except iPhone and Speakers.





```sql
SELECT PRODUCT, SUM(SALEAMOUNT) AS SALES FROM SALES
GROUP BY PRODUCT
HAVING PRODUCT IN ('iPhone', 'Speakers')
```

| | PRODUCT | SALES |
|---|---|---|
| 1 | iPhone | 1500 |
| 2 | Speakers | 400 |

## 🥇 Final Verdict:

**`WHERE` is your go-to for filtering rows before aggregation.**

**`HAVING` is the hero when you need to filter groups after aggregation.**

**Speed Tip:** `WHERE` is usually faster since it narrows down the data set first.

**Use `WHERE` in `SELECT`, `INSERT`, and `UPDATE` statements; save `HAVING` for filtering grouped data in `SELECT` statements.**

🎯 **Pro Tip:** Use both together for ultimate query control!

Level up your SQL game by mastering these clauses and use them wisely to optimize your queries! 🚀

# Scenario series 7 →Interview questions→ Handling Nulls in SQL code

SELECT * FROM EMPDETAILS

| | ID | NAME | MANAGERID |
|---|---|---|---|
| 1 | 1 | MIKE | 3 |
| 2 | 2 | ROB | 1 |
| 3 | 3 | TODD | NULL |
| 4 | 4 | BEN | 1 |
| 5 | 5 | SAM | 1 |

```
SELECT E.NAME AS EMPNAME,M.NAME AS MANAGER
FROM EMPDETAILS E
LEFT JOIN EMPDETAILS M
ON E.MANAGERID=M.ID
```

| | EMPNAME | MANAGER |
|---|---|---|
| 1 | MIKE | TODD |
| 2 | ROB | MIKE |
| 3 | TODD | NULL |
| 4 | BEN | MIKE |
| 5 | SAM | MIKE |

## 1. Replacing NULL value using ISNULL () function:

**Using ISNULL() Function**

W h a t it does: Replaces NULL with a specified replacement value.

**W h e n to use:** Simple replacements when dealing with one column.

**Result:** All NULL manager names are now displayed as 'NO MANAGER'. Neat and tidy!

→SELECT ISNULL (NULL,'NO MANAGER')

```
SELECT E.NAME AS EMPNAME, ISNULL (M.NAME,'NO MANAGER') AS MANAGER
FROM EMPDETAILS E
LEFT JOIN EMPDETAILS M
ON E.MANAGERID=M.ID
```

| | (No column name) |
|---|---|
| 1 | NO MANAGER |

| | EMPNAME | MANAGER |
|---|---|---|
| 1 | MIKE | TODD |
| 2 | ROB | MIKE |
| 3 | TODD | NO MANAGER |
| 4 | BEN | MIKE |
| 5 | SAM | MIKE |

## 2. Replacing NULL value using COALESCE () function:

What it does: Returns the first non-NULL value from a list of arguments.
When to use: When you want to check multiple columns or expressions.

```
SELECT E.NAME AS EMPNAME, COALESCE (M.NAME,'NO MANAGER') AS MANAGER
FROM EMPDETAILS E
LEFT JOIN EMPDETAILS M
ON E.MANAGERID=M.ID
```

| | EMPNAME | MANAGER |
|---|---|---|
| 1 | MIKE | TODD |
| 2 | ROB | MIKE |
| 3 | TODD | NO MANAGER |
| 4 | BEN | MIKE |
| 5 | SAM | MIKE |

You can list multiple potential columns:

```
COALESCE(M.NAME, E.DEFAULT_MANAGER, 'NO MANAGER')
```

Result: Flexible and checks multiple sources before settling on 'NO MANAGER'

## 2. Replacing NULL value using CASE Statement:

What it does: Evaluates conditions and returns specified values.
When to use: When you need complex conditional logic.

```
SELECT E.NAME AS EMPNAME, CASE WHEN M.NAME IS NULL THEN 'NO MANAGER'
ELSE M.NAME END AS MANAGER
FROM EMPDETAILS E
LEFT JOIN EMPDETAILS M
ON E.MANAGERID=M.ID
```

# SQL Mastery Series – 110 Question using 4 table – Set 1

## Emp Table:

| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 1980-12-17 | 800.00 | NULL | 20 |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 | 1600.00 | 300.00 | 30 |
| 3 | 7521 | WARD | SALESMAN | 7698 | 1981-02-22 | 1250.00 | 500.00 | 30 |
| 4 | 7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975.00 | NULL | 20 |
| 5 | 7698 | BLAKE | MANAGER | 7839 | 1981-05-01 | 2850.00 | NULL | 30 |
| 6 | 7782 | CLARK | MANAGER | 7839 | 1981-06-09 | 2450.00 | NULL | 10 |
| 7 | 7839 | KING | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL | 10 |
| 8 | 7902 | FORD | ANALYST | 7566 | 1981-12-03 | 3000.00 | NULL | 20 |
| 9 | 7934 | MILLER | CLERK | 7782 | 1982-01-23 | 1300.00 | NULL | 10 |

## DEPT

| | DEPTNO | DNAME | LOC |
|---|---|---|---|
| 1 | 10 | ACCOUNTING | NEW YORK |
| 2 | 20 | RESEARCH | DALLAS |
| 3 | 30 | SALES | CHICAGO |
| 4 | 40 | OPERATIONS | BOSTON |

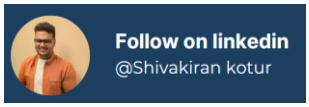
## Salgrade

| | GRADE | LOSAL | HISAL |
|---|---|---|---|
| 1 | 1 | 700.00 | 1200.00 |
| 2 | 2 | 1201.00 | 1400.00 |
| 3 | 3 | 1401.00 | 2000.00 |
| 4 | 4 | 2001.00 | 3000.00 |
| 5 | 5 | 3001.00 | 9999.00 |

## JobHistory

| | EMPNO | JOB | STARTDATE | ENDDATE | DEPTNO |
|---|---|---|---|---|---|
| 1 | 7369 | INTERN | 2019-01-01 | 2020-12-31 | 20 |
| 2 | 7369 | ASSISTANT CLERK | 2021-01-01 | 2022-12-31 | 20 |
| 3 | 7499 | JUNIOR SALESMAN | 2018-03-01 | 2019-12-31 | 30 |
| 4 | 7499 | SALESMAN | 2020-01-01 | 2021-12-31 | 30 |
| 5 | 7521 | TRAINEE | 2017-05-15 | 2019-05-15 | 30 |
| 6 | 7521 | SALESMAN | 2019-05-16 | 2020-12-31 | 30 |
| 7 | 7566 | ASSISTANT MANAGER | 2015-04-02 | 2018-12-31 | 20 |
| 8 | 7566 | MANAGER | 2019-01-01 | 2021-12-31 | 20 |
| 9 | 7698 | ASSISTANT MANAGER | 2016-05-01 | 2019-04-30 | 30 |
| 10 | 7698 | MANAGER | 2019-05-01 | 2022-12-31 | 30 |
| 11 | 7782 | SUPERVISOR | 2016-06-09 | 2018-06-08 | 10 |
| 12 | 7782 | MANAGER | 2018-06-09 | 2021-06-08 | 10 |
| 13 | 7839 | VICE PRESIDENT | 2010-11-17 | 2015-11-16 | 10 |
| 14 | 7839 | PRESIDENT | 2015-11-17 | 2023-12-31 | 10 |
| 15 | 7902 | SENIOR ANALYST | 2017-12-03 | 2019-12-02 | 20 |
| 16 | 7902 | ANALYST | 2019-12-03 | 2022-12-02 | 20 |
| 17 | 7934 | JUNIOR CLERK | 2016-01-23 | 2018-01-22 | 10 |
| 18 | 7934 | CLERK | 2018-01-23 | 2020-01-22 | 10 |






Follow me on LinkedIn – Shivakiran kotur

```
-- 1) Display all employees with their department names.
SELECT E.ENAME, D.DNAME
FROM EMP E
JOIN DEPT D ON E.DEPTNO = D.DEPTNO;
```

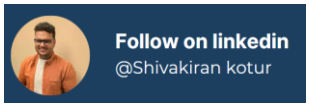
| | ENAME | DNAME |
|---|---|---|
| 1 | SMITH | RESEARCH |
| 2 | ALLEN | SALES |
| 3 | WARD | SALES |
| 4 | JONES | RESEARCH |
| 5 | BLAKE | SALES |
| 6 | CLARK | ACCOUNTING |
| 7 | KING | ACCOUNTING |
| 8 | FORD | RESEARCH |
| 9 | MILLER | ACCOUNTING |

```
-- 2) Display employees along with their manager names.
SELECT E.ENAME AS EMPLOYEE_NAME, M.ENAME AS MANAGER_NAME
FROM EMP E
LEFT JOIN EMP M ON E.MGR = M.EMPNO;
```

| | EMPLOYEE_NAME | MANAGER_NAME |
|---|---|---|
| 1 | SMITH | FORD |
| 2 | ALLEN | BLAKE |
| 3 | WARD | BLAKE |
| 4 | JONES | KING |
| 5 | BLAKE | KING |
| 6 | CLARK | KING |
| 7 | KING | NULL |
| 8 | FORD | JONES |
| 9 | MILLER | CLARK |

```
-- 3) Display employee names, salaries, and total salaries for each department.
SELECT E.ENAME, E.SAL, SUM(E.SAL) OVER (PARTITION BY E.DEPTNO) AS
TOTAL_DEPT_SALARY
FROM EMP E;
```

| | ENAME | SAL | TOTAL_DEPT_SALARY |
|---|---|---|---|
| 1 | CLARK | 2450.00 | 8750.00 |
| 2 | KING | 5000.00 | 8750.00 |
| 3 | MILLER | 1300.00 | 8750.00 |
| 4 | FORD | 3000.00 | 6775.00 |
| 5 | SMITH | 800.00 | 6775.00 |
| 6 | JONES | 2975.00 | 6775.00 |
| 7 | BLAKE | 2850.00 | 5700.00 |
| 8 | ALLEN | 1600.00 | 5700.00 |
| 9 | WARD | 1250.00 | 5700.00 |




Follow me on LinkedIn – Shivakiran kotur

-- 4) Display employee names and their annual salary (SAL * 12).

```
SELECT ENAME, SAL * 12 AS ANNUAL_SALARY
FROM EMP;
```

| | ENAME | ANNUAL_SALARY |
|---|---|---|
| 1 | SMITH | 9600.00 |
| 2 | ALLEN | 19200.00 |
| 3 | WARD | 15000.00 |
| 4 | JONES | 35700.00 |
| 5 | BLAKE | 34200.00 |
| 6 | CLARK | 29400.00 |
| 7 | KING | 60000.00 |
| 8 | FORD | 36000.00 |
| 9 | MILLER | 15600.00 |

-- 5) Display the total salary for each department.

```
SELECT DEPTNO, SUM(SAL) AS TOTAL_SALARY
FROM EMP
GROUP BY DEPTNO;
```

| | DEPTNO | TOTAL_SALARY |
|---|---|---|
| 1 | 10 | 8750.00 |
| 2 | 20 | 6775.00 |
| 3 | 30 | 5700.00 |

-- 6) Find the highest salary in each department.

```
SELECT DEPTNO, MAX(SAL) AS HIGHEST_SALARY
FROM EMP
GROUP BY DEPTNO;
```

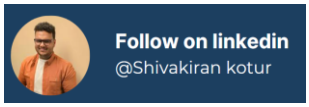
| | DEPTNO | HIGHEST_SALARY |
|---|---|---|
| 1 | 10 | 5000.00 |
| 2 | 20 | 3000.00 |
| 3 | 30 | 2850.00 |

-- 7) Display department-wise employee count.

```
SELECT DEPTNO, COUNT(*) AS EMPLOYEE_COUNT
FROM EMP
GROUP BY DEPTNO;
```

| | DEPTNO | EMPLOYEE_COUNT |
|---|---|---|
| 1 | 10 | 3 |
| 2 | 20 | 3 |
| 3 | 30 | 3 |






Follow me on LinkedIn – Shivakiran kotur

```
-- 8) Display the names of employees who earn more than the average salary of their department.
SELECT ENAME FROM EMP E
WHERE SAL > (
  SELECT AVG(SAL)
  FROM EMP
  WHERE DEPTNO = E.DEPTNO
);
```

| | ENAME |
|---|---|
| 1 | KING |
| 2 | FORD |
| 3 | JONES |
| 4 | BLAKE |

```
-- 9) Display the names of employees who have the highest salary in their department.
SELECT ENAME
FROM EMP E
WHERE SAL = (
  SELECT MAX(SAL)
  FROM EMP
  WHERE DEPTNO = E.DEPTNO
);
```

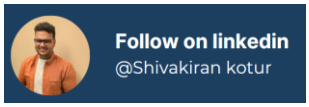
| | ENAME |
|---|---|
| 1 | BLAKE |
| 2 | FORD |
| 3 | KING |

```
-- 10) Display the department name and total salary for each department.
SELECT D.DNAME, SUM(E.SAL) AS TOTAL_SALARY
FROM EMP E
JOIN DEPT D ON E.DEPTNO = D.DEPTNO
GROUP BY D.DNAME;
```

Results Messages

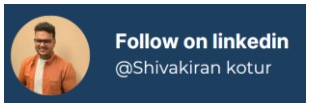
| | DNAME | TOTAL_SALARY |
|---|---|---|
| 1 | ACCOUNTING | 8750.00 |
| 2 | RESEARCH | 6775.00 |
| 3 | SALES | 5700.00 |





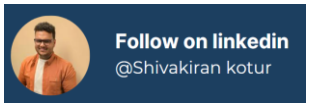
```sql
--Table 1 Employee Table
CREATE TABLE EMP (
    EMPNO INT PRIMARY KEY,        -- Employee Number
    ENAME VARCHAR(50),            -- Employee Name
    JOB VARCHAR(50),              -- Job Title
    MGR INT,                      -- Manager's Employee Number
    HIREDATE DATE,                -- Hire Date
    SAL DECIMAL(10, 2),           -- Salary
    COMM DECIMAL(10, 2),          -- Commission
    DEPTNO INT,                   -- Department Number
    CONSTRAINT FK_MGR FOREIGN KEY (MGR) REFERENCES EMP(EMPNO)
);

--department Table
CREATE TABLE DEPT (
    DEPTNO INT PRIMARY KEY,       -- Department Number
    DNAME VARCHAR(50),            -- Department Name
    LOC VARCHAR(50)               -- Location
);


-- Salary Grade
CREATE TABLE SALGRADE (
    GRADE INT,                    -- Salary Grade
    LOSAL DECIMAL(10, 2),         -- Lowest Salary for Grade
    HISAL DECIMAL(10, 2)          -- Highest Salary for Grade
);

-- Job History

CREATE TABLE JOBHISTORY (
    EMPNO INT,                    -- Employee Number
    JOB VARCHAR(50),              -- Job Title
    STARTDATE DATE,               -- Start Date of Job
    ENDDATE DATE,                 -- End Date of Job
    DEPTNO INT,                   -- Department Number
    CONSTRAINT FK_EMPNO FOREIGN KEY (EMPNO) REFERENCES EMP(EMPNO),
    CONSTRAINT FK_DEPTNO FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO)
);
```





```
INSERT INTO DEPT (DEPTNO, DNAME, LOC) VALUES
(10, 'ACCOUNTING', 'NEW YORK'),
(20, 'RESEARCH', 'DALLAS'),
(30, 'SALES', 'CHICAGO'),
(40, 'OPERATIONS', 'BOSTON');

INSERT INTO SALGRADE (GRADE, LOSAL, HISAL) VALUES
(1, 700, 1200),
(2, 1201, 1400),
(3, 1401, 2000),
(4, 2001, 3000),
(5, 3001, 9999);

INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO) VALUES
(7369, 'SMITH', 'CLERK', 7902, '1980-12-17', 800, NULL, 20),
(7499, 'ALLEN', 'SALESMAN', 7698, '1981-02-20', 1600, 300, 30),
(7521, 'WARD', 'SALESMAN', 7698, '1981-02-22', 1250, 500, 30),
(7566, 'JONES', 'MANAGER', 7839, '1981-04-02', 2975, NULL, 20),
(7698, 'BLAKE', 'MANAGER', 7839, '1981-05-01', 2850, NULL, 30),
(7782, 'CLARK', 'MANAGER', 7839, '1981-06-09', 2450, NULL, 10),
(7839, 'KING', 'PRESIDENT', NULL, '1981-11-17', 5000, NULL, 10),
(7902, 'FORD', 'ANALYST', 7566, '1981-12-03', 3000, NULL, 20),
(7934, 'MILLER', 'CLERK', 7782, '1982-01-23', 1300, NULL, 10);

CREATE TABLE JOB_HISTORY (
   EMPNO INT,
   START_DATE DATE,
   END_DATE DATE,
   JOB VARCHAR(50),
   DEPTNO INT,
   PRIMARY KEY (EMPNO, START_DATE)
);

INSERT INTO JOB_HISTORY (EMPNO, STARTDATE, ENDDATE, JOB, DEPTNO) VALUES
(7369, '2019-01-01', '2020-12-31', 'INTERN', 20),
(7369, '2021-01-01', '2022-12-31', 'ASSISTANT CLERK', 20),
```





```
(7499, '2018-03-01', '2019-12-31', 'JUNIOR SALESMAN', 30),
(7499, '2020-01-01', '2021-12-31', 'SALESMAN', 30),
(7521, '2017-05-15', '2019-05-15', 'TRAINEE', 30),
(7521, '2019-05-16', '2020-12-31', 'SALESMAN', 30),
(7566, '2015-04-02', '2018-12-31', 'ASSISTANT MANAGER', 20),
(7566, '2019-01-01', '2021-12-31', 'MANAGER', 20),
(7698, '2016-05-01', '2019-04-30', 'ASSISTANT MANAGER', 30),
(7698, '2019-05-01', '2022-12-31', 'MANAGER', 30),
(7782, '2016-06-09', '2018-06-08', 'SUPERVISOR', 10),
(7782, '2018-06-09', '2021-06-08', 'MANAGER', 10),
(7839, '2010-11-17', '2015-11-16', 'VICE PRESIDENT', 10),
(7839, '2015-11-17', '2023-12-31', 'PRESIDENT', 10),
(7902, '2017-12-03', '2019-12-02', 'SENIOR ANALYST', 20),
(7902, '2019-12-03', '2022-12-02', 'ANALYST', 20),
(7934, '2016-01-23', '2018-01-22', 'JUNIOR CLERK', 10),
(7934, '2018-01-23', '2020-01-22', 'CLERK', 10);
```

# SQL Mastery Series – 110 Question using 4 table – Set 2 Emp

## Table:

| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 1980-12-17 | 800.00 | NULL | 20 |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 | 1600.00 | 300.00 | 30 |
| 3 | 7521 | WARD | SALESMAN | 7698 | 1981-02-22 | 1250.00 | 500.00 | 30 |
| 4 | 7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975.00 | NULL | 20 |
| 5 | 7698 | BLAKE | MANAGER | 7839 | 1981-05-01 | 2850.00 | NULL | 30 |
| 6 | 7782 | CLARK | MANAGER | 7839 | 1981-06-09 | 2450.00 | NULL | 10 |
| 7 | 7839 | KING | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL | 10 |
| 8 | 7902 | FORD | ANALYST | 7566 | 1981-12-03 | 3000.00 | NULL | 20 |
| 9 | 7934 | MILLER | CLERK | 7782 | 1982-01-23 | 1300.00 | NULL | 10 |

## DEPT

| | DEPTNO | DNAME | LOC |
|---|---|---|---|
| 1 | 10 | ACCOUNTING | NEW YORK |
| 2 | 20 | RESEARCH | DALLAS |
| 3 | 30 | SALES | CHICAGO |
| 4 | 40 | OPERATIONS | BOSTON |

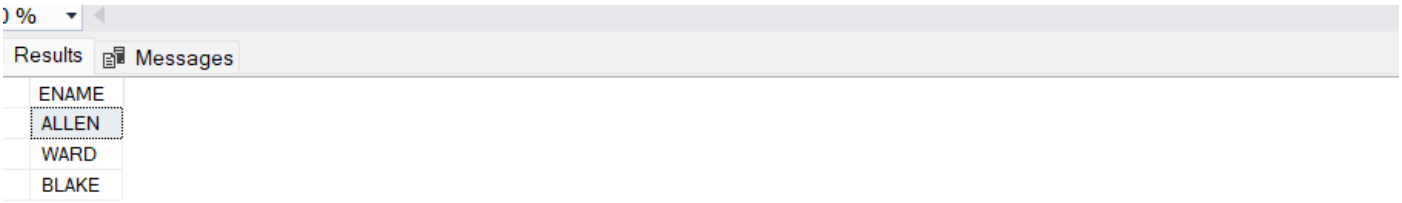
## Salgrade

| | GRADE | LOSAL | HISAL |
|---|---|---|---|
| 1 | 1 | 700.00 | 1200.00 |
| 2 | 2 | 1201.00 | 1400.00 |
| 3 | 3 | 1401.00 | 2000.00 |
| 4 | 4 | 2001.00 | 3000.00 |
| 5 | 5 | 3001.00 | 9999.00 |

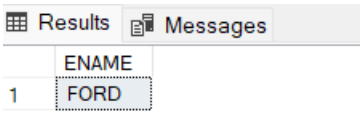
## JobHistory

| | EMPNO | JOB | STARTDATE | ENDDATE | DEPTNO |
|---|---|---|---|---|---|
| 1 | 7369 | INTERN | 2019-01-01 | 2020-12-31 | 20 |
| 2 | 7369 | ASSISTANT CLERK | 2021-01-01 | 2022-12-31 | 20 |
| 3 | 7499 | JUNIOR SALESMAN | 2018-03-01 | 2019-12-31 | 30 |
| 4 | 7499 | SALESMAN | 2020-01-01 | 2021-12-31 | 30 |
| 5 | 7521 | TRAINEE | 2017-05-15 | 2019-05-15 | 30 |
| 6 | 7521 | SALESMAN | 2019-05-16 | 2020-12-31 | 30 |
| 7 | 7566 | ASSISTANT MANAGER | 2015-04-02 | 2018-12-31 | 20 |
| 8 | 7566 | MANAGER | 2019-01-01 | 2021-12-31 | 20 |
| 9 | 7698 | ASSISTANT MANAGER | 2016-05-01 | 2019-04-30 | 30 |
| 10 | 7698 | MANAGER | 2019-05-01 | 2022-12-31 | 30 |
| 11 | 7782 | SUPERVISOR | 2016-06-09 | 2018-06-08 | 10 |
| 12 | 7782 | MANAGER | 2018-06-09 | 2021-06-08 | 10 |
| 13 | 7839 | VICE PRESIDENT | 2010-11-17 | 2015-11-16 | 10 |
| 14 | 7839 | PRESIDENT | 2015-11-17 | 2023-12-31 | 10 |
| 15 | 7902 | SENIOR ANALYST | 2017-12-03 | 2019-12-02 | 20 |
| 16 | 7902 | ANALYST | 2019-12-03 | 2022-12-02 | 20 |
| 17 | 7934 | JUNIOR CLERK | 2016-01-23 | 2018-01-22 | 10 |
| 18 | 7934 | CLERK | 2018-01-23 | 2020-01-22 | 10 |

```sql
-- 11) Display the names of employees who are working in the SALES
department.
SELECT ENAME
FROM EMP E
JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE D.DNAME = 'SALES';
```

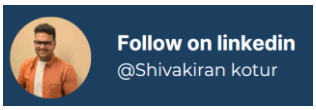



```sql
-- 12) Find the employee with the second highest salary in the
company.
SELECT TOP 1 ENAME
FROM (
  SELECT ENAME, RANK() OVER (ORDER BY SAL DESC) AS RANK
  FROM EMP
) AS SAL_RANKED
WHERE RANK = 2;
```




**Optimized way**

```sql
SELECT ENAME
FROM (
  SELECT ENAME, SAL, DENSE_RANK() OVER (ORDER BY SAL DESC) AS RANK
  FROM EMP
) AS SAL_RANKED
WHERE RANK = 2;

--Using Joins

SELECT E1.ENAME
FROM EMP E1
JOIN EMP E2 ON E1.SAL < E2.SAL
GROUP BY E1.ENAME, E1.SAL
HAVING COUNT(DISTINCT E2.SAL) = 1;
```



```sql
-- 13) Display the average salary for each job title round to 2
decimal.
SELECT JOB, ROUND(CAST(AVG(SAL) AS DECIMAL(10, 2)), 2) AS
AVERAGE_SALARY
FROM EMP
GROUP BY JOB;
```

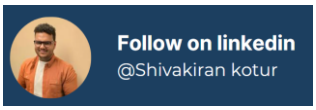
| | JOB | AVERAGE_SALARY |
|---|---|---|
| 1 | ANALYST | 3000.00 |
| 2 | CLERK | 1050.00 |
| 3 | MANAGER | 2758.33 |
| 4 | PRESIDENT | 5000.00 |
| 5 | SALESMAN | 1425.00 |

```sql
-- 14) Display the names of employees who joined after the employee
‘SMITH’.
SELECT ENAME
FROM EMP
WHERE HIREDATE > (
  SELECT HIREDATE
  FROM EMP
  WHERE ENAME = 'SMITH'
);
```

| | ENAME |
|---|---|
| 1 | ALLEN |
| 2 | WARD |
| 3 | JONES |
| 4 | BLAKE |
| 5 | CLARK |
| 6 | KING |
| 7 | FORD |
| 8 | MILLER |

**Optimized way:**

```sql
WITH SMITH_HIREDATE AS (
  SELECT HIREDATE
  FROM EMP
  WHERE ENAME = 'SMITH'
)
SELECT ENAME
FROM EMP, SMITH_HIREDATE
WHERE EMP.HIREDATE > SMITH_HIREDATE.HIREDATE;
```



```sql
-- 15) Display employee details along with their commission, but show '0' if no commission is given.
SELECT ENAME, SAL, ISNULL(COMM, 0) AS COMMISSION
FROM EMP;
```

| | ENAME | SAL | COMMISSION |
|---|---|---|---|
| 1 | SMITH | 800.00 | 0.00 |
| 2 | ALLEN | 1600.00 | 300.00 |
| 3 | WARD | 1250.00 | 500.00 |
| 4 | JONES | 2975.00 | 0.00 |
| 5 | BLAKE | 2850.00 | 0.00 |
| 6 | CLARK | 2450.00 | 0.00 |
| 7 | KING | 5000.00 | 0.00 |
| 8 | FORD | 3000.00 | 0.00 |
| 9 | MILLER | 1300.00 | 0.00 |

**Notes on ISNULL:**

1. **Definition:**
   - ISNULL is a function used in SQL to replace NULL values with a specified value.
2. **Syntax:**

```
ISNULL(expression, replacement_value)
```

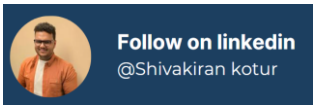
   - **expression:** The value or column to check for NULL.
   - **replacement_value:** The value to return if the expression is NULL.

```sql
-- 16) Display employees who do not have a manager.
SELECT ENAME
FROM EMP
WHERE MGR IS NULL;
```

| | ENAME |
|---|---|
| 1 | KING |

**Check out 200+ scenario based databricks and pyspark Scenario based Question in Topmate:** https://topmate.io/shivakiran_kotur/1376452

**Check out 200+ Python Question and Answer for Data Engineer asked in Interview in Topmate:** https://topmate.io/shivakiran_kotur/1337666



# SQL Mastery Series – 110 Question using 4 table – Set 3

## Emp Table:

| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 1980-12-17 | 800.00 | NULL | 20 |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 | 1600.00 | 300.00 | 30 |
| 3 | 7521 | WARD | SALESMAN | 7698 | 1981-02-22 | 1250.00 | 500.00 | 30 |
| 4 | 7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975.00 | NULL | 20 |
| 5 | 7698 | BLAKE | MANAGER | 7839 | 1981-05-01 | 2850.00 | NULL | 30 |
| 6 | 7782 | CLARK | MANAGER | 7839 | 1981-06-09 | 2450.00 | NULL | 10 |
| 7 | 7839 | KING | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL | 10 |
| 8 | 7902 | FORD | ANALYST | 7566 | 1981-12-03 | 3000.00 | NULL | 20 |
| 9 | 7934 | MILLER | CLERK | 7782 | 1982-01-23 | 1300.00 | NULL | 10 |

## DEPT

| | DEPTNO | DNAME | LOC |
|---|---|---|---|
| 1 | 10 | ACCOUNTING | NEW YORK |
| 2 | 20 | RESEARCH | DALLAS |
| 3 | 30 | SALES | CHICAGO |
| 4 | 40 | OPERATIONS | BOSTON |

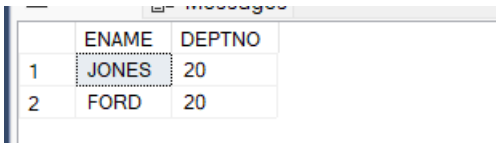
## Salgrade

| | GRADE | LOSAL | HISAL |
|---|---|---|---|
| 1 | 1 | 700.00 | 1200.00 |
| 2 | 2 | 1201.00 | 1400.00 |
| 3 | 3 | 1401.00 | 2000.00 |
| 4 | 4 | 2001.00 | 3000.00 |
| 5 | 5 | 3001.00 | 9999.00 |

## JobHistory

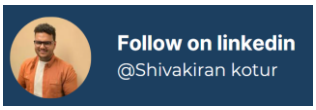
| | EMPNO | JOB | STARTDATE | ENDDATE | DEPTNO |
|---|---|---|---|---|---|
| 1 | 7369 | INTERN | 2019-01-01 | 2020-12-31 | 20 |
| 2 | 7369 | ASSISTANT CLERK | 2021-01-01 | 2022-12-31 | 20 |
| 3 | 7499 | JUNIOR SALESMAN | 2018-03-01 | 2019-12-31 | 30 |
| 4 | 7499 | SALESMAN | 2020-01-01 | 2021-12-31 | 30 |
| 5 | 7521 | TRAINEE | 2017-05-15 | 2019-05-15 | 30 |
| 6 | 7521 | SALESMAN | 2019-05-16 | 2020-12-31 | 30 |
| 7 | 7566 | ASSISTANT MANAGER | 2015-04-02 | 2018-12-31 | 20 |
| 8 | 7566 | MANAGER | 2019-01-01 | 2021-12-31 | 20 |
| 9 | 7698 | ASSISTANT MANAGER | 2016-05-01 | 2019-04-30 | 30 |
| 10 | 7698 | MANAGER | 2019-05-01 | 2022-12-31 | 30 |
| 11 | 7782 | SUPERVISOR | 2016-06-09 | 2018-06-08 | 10 |
| 12 | 7782 | MANAGER | 2018-06-09 | 2021-06-08 | 10 |
| 13 | 7839 | VICE PRESIDENT | 2010-11-17 | 2015-11-16 | 10 |
| 14 | 7839 | PRESIDENT | 2015-11-17 | 2023-12-31 | 10 |
| 15 | 7902 | SENIOR ANALYST | 2017-12-03 | 2019-12-02 | 20 |
| 16 | 7902 | ANALYST | 2019-12-03 | 2022-12-02 | 20 |
| 17 | 7934 | JUNIOR CLERK | 2016-01-23 | 2018-01-22 | 10 |
| 18 | 7934 | CLERK | 2018-01-23 | 2020-01-22 | 10 |

```sql
-- 17) Display the names of employees who work in the same
department as ‘SMITH’ and dont include smith.
SELECT ENAME
FROM EMP
WHERE DEPTNO = (
  SELECT DEPTNO
  FROM EMP
  WHERE ENAME = 'SMITH'
)
AND ENAME != 'SMITH';

--Optimized way using joins
SELECT E1.ENAME
FROM EMP E1
JOIN EMP E2 ON E1.DEPTNO = E2.DEPTNO
WHERE E2.ENAME = 'SMITH'
AND E1.ENAME != 'SMITH';

--USing CTE
WITH SmithDept AS (
  SELECT DEPTNO
  FROM EMP
  WHERE ENAME = 'SMITH'
)
SELECT ENAME
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO FROM SmithDept)
AND ENAME != 'SMITH';
```



| | ENAME | DEPTNO |
|---|---|---|
| 1 | JONES | 20 |
| 2 | FORD | 20 |

```sql
-- 18) Display the names of employees who do the same job as
‘ALLEN’.
SELECT ENAME
FROM EMP
WHERE JOB = (
  SELECT JOB
  FROM EMP
  WHERE ENAME = 'ALLEN'
);
```



```
--Give optimized query in comment
```

| | ENAME |
|---|---|
| 1 | ALLEN |
| 2 | WARD |

```
-- 19) Find employees whose job title contains the letter ‘M’.
SELECT ENAME , Job
FROM EMP
WHERE JOB LIKE '%M%';
```

| | ENAME | Job |
|---|---|---|
| 1 | ALLEN | SALESMAN |
| 2 | WARD | SALESMAN |
| 3 | JONES | MANAGER |
| 4 | BLAKE | MANAGER |
| 5 | CLARK | MANAGER |

```
-- 20) Display the details of employees whose salary is between
1000 and 2000.
SELECT *
FROM EMP
WHERE SAL BETWEEN 1000 AND 2000;
```

| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 | 1600.00 | 300.00 | 30 |
| 2 | 7521 | WARD | SALESMAN | 7698 | 1981-02-22 | 1250.00 | 500.00 | 30 |
| 3 | 7934 | MILLER | CLERK | 7782 | 1982-01-23 | 1300.00 | NULL | 10 |

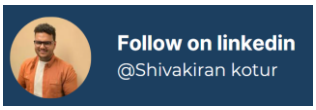
```
-- 21) Find employees who joined in the year 1981.
SELECT ENAME, HIREDATE
FROM EMP
WHERE YEAR(HIREDATE) = 1981;
```

| | ENAME | HIREDATE |
|---|---|---|
| 1 | ALLEN | 1981-02-20 |
| 2 | WARD | 1981-02-22 |
| 3 | JONES | 1981-04-02 |
| 4 | BLAKE | 1981-05-01 |
| 5 | CLARK | 1981-06-09 |
| 6 | KING | 1981-11-17 |
| 7 | FORD | 1981-12-03 |

```sql
-- 22) Find employees whose salary is higher than their manager's
salary.
SELECT E.ENAME
FROM EMP E
JOIN EMP M ON E.MGR = M.EMPNO
WHERE E.SAL > M.SAL;

--Using CTE
WITH ManagerSalaries AS (
  SELECT EMPNO, SAL
  FROM EMP
)
SELECT E.ENAME
FROM EMP E
JOIN ManagerSalaries M ON E.MGR = M.EMPNO
WHERE E.SAL > M.SAL;
```

| | ENAME |
|---|---|
| 1 | FORD |





# SQL Mastery Series – 110 Question using 4 table – Set 5

## Emp Table:

| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 1980-12-17 | 800.00 | NULL | 20 |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 | 1600.00 | 300.00 | 30 |
| 3 | 7521 | WARD | SALESMAN | 7698 | 1981-02-22 | 1250.00 | 500.00 | 30 |
| 4 | 7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975.00 | NULL | 20 |
| 5 | 7698 | BLAKE | MANAGER | 7839 | 1981-05-01 | 2850.00 | NULL | 30 |
| 6 | 7782 | CLARK | MANAGER | 7839 | 1981-06-09 | 2450.00 | NULL | 10 |
| 7 | 7839 | KING | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL | 10 |
| 8 | 7902 | FORD | ANALYST | 7566 | 1981-12-03 | 3000.00 | NULL | 20 |
| 9 | 7934 | MILLER | CLERK | 7782 | 1982-01-23 | 1300.00 | NULL | 10 |

## DEPT

| | DEPTNO | DNAME | LOC |
|---|---|---|---|
| 1 | 10 | ACCOUNTING | NEW YORK |
| 2 | 20 | RESEARCH | DALLAS |
| 3 | 30 | SALES | CHICAGO |
| 4 | 40 | OPERATIONS | BOSTON |

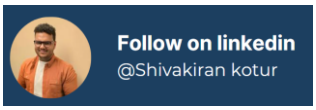
## Salgrade

| | GRADE | LOSAL | HISAL |
|---|---|---|---|
| 1 | 1 | 700.00 | 1200.00 |
| 2 | 2 | 1201.00 | 1400.00 |
| 3 | 3 | 1401.00 | 2000.00 |
| 4 | 4 | 2001.00 | 3000.00 |
| 5 | 5 | 3001.00 | 9999.00 |

## JobHistory

| | EMPNO | JOB | STARTDATE | ENDDATE | DEPTNO |
|---|---|---|---|---|---|
| 1 | 7369 | INTERN | 2019-01-01 | 2020-12-31 | 20 |
| 2 | 7369 | ASSISTANT CLERK | 2021-01-01 | 2022-12-31 | 20 |
| 3 | 7499 | JUNIOR SALESMAN | 2018-03-01 | 2019-12-31 | 30 |
| 4 | 7499 | SALESMAN | 2020-01-01 | 2021-12-31 | 30 |
| 5 | 7521 | TRAINEE | 2017-05-15 | 2019-05-15 | 30 |
| 6 | 7521 | SALESMAN | 2019-05-16 | 2020-12-31 | 30 |
| 7 | 7566 | ASSISTANT MANAGER | 2015-04-02 | 2018-12-31 | 20 |
| 8 | 7566 | MANAGER | 2019-01-01 | 2021-12-31 | 20 |
| 9 | 7698 | ASSISTANT MANAGER | 2016-05-01 | 2019-04-30 | 30 |
| 10 | 7698 | MANAGER | 2019-05-01 | 2022-12-31 | 30 |
| 11 | 7782 | SUPERVISOR | 2016-06-09 | 2018-06-08 | 10 |
| 12 | 7782 | MANAGER | 2018-06-09 | 2021-06-08 | 10 |
| 13 | 7839 | VICE PRESIDENT | 2010-11-17 | 2015-11-16 | 10 |
| 14 | 7839 | PRESIDENT | 2015-11-17 | 2023-12-31 | 10 |
| 15 | 7902 | SENIOR ANALYST | 2017-12-03 | 2019-12-02 | 20 |
| 16 | 7902 | ANALYST | 2019-12-03 | 2022-12-02 | 20 |
| 17 | 7934 | JUNIOR CLERK | 2016-01-23 | 2018-01-22 | 10 |
| 18 | 7934 | CLERK | 2018-01-23 | 2020-01-22 | 10 |





```sql
-- 31) Display those employees whose salary is less than their manager's salary but more than the salary of any other manager.
SELECT E.ENAME
FROM EMP E
JOIN EMP M ON E.MGR = M.EMPNO
WHERE E.SAL < M.SAL
AND E.SAL > ANY (SELECT SAL FROM EMP WHERE EMPNO IN (SELECT DISTINCT MGR FROM EMP WHERE MGR IS NOT NULL));
```

| | ENAME |
|---|---|
| 1 | JONES |
| 2 | BLAKE |

```sql
-- 32) Display all employee names with the total salary of the company for each employee.
SELECT ENAME, (SELECT SUM(SAL) FROM EMP) AS TOTAL_COMPANY_SALARY
FROM EMP;
```

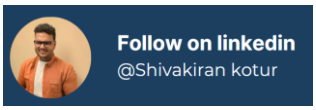
| ENAME | TOTAL_COMPANY_SALARY |
|---|---|
| SMITH | 21225.00 |
| ALLEN | 21225.00 |
| WARD | 21225.00 |
| JONES | 21225.00 |
| BLAKE | 21225.00 |
| CLARK | 21225.00 |
| KING | 21225.00 |
| FORD | 21225.00 |
| MILLER | 21225.00 |

```sql
-- 33) Find the least 5 earners in the company.
SELECT TOP 5 ENAME, SAL
FROM EMP
ORDER BY SAL ASC;
```

| ENAME | SAL |
|---|---|
| SMITH | 800.00 |
| WARD | 1250.00 |
| MILLER | 1300.00 |
| ALLEN | 1600.00 |
| CLARK | 2450.00 |

```sql
-- 34) Find the number of employees whose salary is greater than their manager's salary.
SELECT COUNT(*) as emps
FROM EMP E
JOIN EMP M ON E.MGR = M.EMPNO
WHERE E.SAL > M.SAL;
```

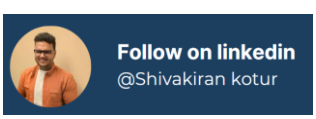
| | emps |
|---|---|
| 1 | 1 |



```sql
-- 35) Display those managers who are not working under the
president but are working under other managers.
SELECT M.ENAME
FROM EMP M
WHERE M.EMPNO IN (SELECT DISTINCT MGR FROM EMP WHERE MGR IS NOT
NULL)
AND M.MGR IS NOT NULL
AND M.MGR <> (SELECT EMPNO FROM EMP WHERE JOB = 'PRESIDENT');


--Optimized way using Joins
SELECT DISTINCT M.ENAME
FROM EMP M
JOIN EMP S ON M.EMPNO = S.MGR
WHERE M.MGR IS NOT NULL
AND M.MGR <> (SELECT EMPNO FROM EMP WHERE JOB = 'PRESIDENT');
```

| ENAME |
| --- |
| FORD |

```sql
-- 36) Delete those departments where no employee is working.
DELETE FROM DEPT
WHERE DEPTNO NOT IN (SELECT DISTINCT DEPTNO FROM EMP);


-- 37) Delete records from the employee table where the department
number is not available in the department table.
DELETE FROM EMP
WHERE DEPTNO NOT IN (SELECT DEPTNO FROM DEPT);


-- 38) Display those employee names whose salary is outside the
ranges defined in the salary grade table.
SELECT ENAME
FROM EMP E
WHERE NOT EXISTS (
  SELECT 1
  FROM SALGRADE S
  WHERE E.SAL BETWEEN S.LOSAL AND S.HISAL
);
```



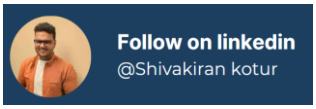
```sql
--Using Joins Optimized way
SELECT E.ENAME
FROM EMP E
LEFT JOIN SALGRADE S
ON E.SAL BETWEEN S.LOSAL AND S.HISAL
WHERE S.GRADE IS NULL;
```

```sql
-- 39) Display employee name, salary, commission, and net pay where the net pay is greater than any other employee's salary in the company.
SELECT ENAME, SAL, COMM, (SAL + ISNULL(COMM, 0)) AS NET_PAY
FROM EMP
WHERE (SAL + ISNULL(COMM, 0)) > ANY (SELECT SAL FROM EMP);
```

| ENAME | SAL | COMM | NET_PAY |
|---|---|---|---|
| ALLEN | 1600.00 | 300.00 | 1900.00 |
| WARD | 1250.00 | 500.00 | 1750.00 |
| JONES | 2975.00 | NULL | 2975.00 |
| BLAKE | 2850.00 | NULL | 2850.00 |
| CLARK | 2450.00 | NULL | 2450.00 |
| KING | 5000.00 | NULL | 5000.00 |
| FORD | 3000.00 | NULL | 3000.00 |
| MILLER | 1300.00 | NULL | 1300.00 |

```sql
-- 40) Display the names of those employees who are going to retire on 31-Dec-99, if the maximum job period is 30 years.
SELECT ENAME
FROM EMP
WHERE DATEADD(YEAR, 30, HIREDATE) = '1999-12-31';
```





# SQL Mastery Series – 110 Question using 4 table – Set 3

## Emp Table:

| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 1980-12-17 | 800.00 | NULL | 20 |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 | 1600.00 | 300.00 | 30 |
| 3 | 7521 | WARD | SALESMAN | 7698 | 1981-02-22 | 1250.00 | 500.00 | 30 |
| 4 | 7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975.00 | NULL | 20 |
| 5 | 7698 | BLAKE | MANAGER | 7839 | 1981-05-01 | 2850.00 | NULL | 30 |
| 6 | 7782 | CLARK | MANAGER | 7839 | 1981-06-09 | 2450.00 | NULL | 10 |
| 7 | 7839 | KING | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL | 10 |
| 8 | 7902 | FORD | ANALYST | 7566 | 1981-12-03 | 3000.00 | NULL | 20 |
| 9 | 7934 | MILLER | CLERK | 7782 | 1982-01-23 | 1300.00 | NULL | 10 |

## DEPT

| | DEPTNO | DNAME | LOC |
|---|---|---|---|
| 1 | 10 | ACCOUNTING | NEW YORK |
| 2 | 20 | RESEARCH | DALLAS |
| 3 | 30 | SALES | CHICAGO |
| 4 | 40 | OPERATIONS | BOSTON |

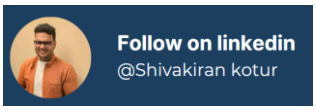
## Salgrade

| | GRADE | LOSAL | HISAL |
|---|---|---|---|
| 1 | 1 | 700.00 | 1200.00 |
| 2 | 2 | 1201.00 | 1400.00 |
| 3 | 3 | 1401.00 | 2000.00 |
| 4 | 4 | 2001.00 | 3000.00 |
| 5 | 5 | 3001.00 | 9999.00 |

## JobHistory

| | EMPNO | JOB | STARTDATE | ENDDATE | DEPTNO |
|---|---|---|---|---|---|
| 1 | 7369 | INTERN | 2019-01-01 | 2020-12-31 | 20 |
| 2 | 7369 | ASSISTANT CLERK | 2021-01-01 | 2022-12-31 | 20 |
| 3 | 7499 | JUNIOR SALESMAN | 2018-03-01 | 2019-12-31 | 30 |
| 4 | 7499 | SALESMAN | 2020-01-01 | 2021-12-31 | 30 |
| 5 | 7521 | TRAINEE | 2017-05-15 | 2019-05-15 | 30 |
| 6 | 7521 | SALESMAN | 2019-05-16 | 2020-12-31 | 30 |
| 7 | 7566 | ASSISTANT MANAGER | 2015-04-02 | 2018-12-31 | 20 |
| 8 | 7566 | MANAGER | 2019-01-01 | 2021-12-31 | 20 |
| 9 | 7698 | ASSISTANT MANAGER | 2016-05-01 | 2019-04-30 | 30 |
| 10 | 7698 | MANAGER | 2019-05-01 | 2022-12-31 | 30 |
| 11 | 7782 | SUPERVISOR | 2016-06-09 | 2018-06-08 | 10 |
| 12 | 7782 | MANAGER | 2018-06-09 | 2021-06-08 | 10 |
| 13 | 7839 | VICE PRESIDENT | 2010-11-17 | 2015-11-16 | 10 |
| 14 | 7839 | PRESIDENT | 2015-11-17 | 2023-12-31 | 10 |
| 15 | 7902 | SENIOR ANALYST | 2017-12-03 | 2019-12-02 | 20 |
| 16 | 7902 | ANALYST | 2019-12-03 | 2022-12-02 | 20 |
| 17 | 7934 | JUNIOR CLERK | 2016-01-23 | 2018-01-22 | 10 |
| 18 | 7934 | CLERK | 2018-01-23 | 2020-01-22 | 10 |





```sql
-- 23) Display employee details where the third character of the name is ‘A’.
SELECT *
FROM EMP
WHERE ENAME LIKE '__A%';
```

| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7698 | BLAKE | MANAGER | 7839 | 1981-05-01 | 2850.00 | NULL | 30 |
| 2 | 7782 | CLARK | MANAGER | 7839 | 1981-06-09 | 2450.00 | NULL | 10 |

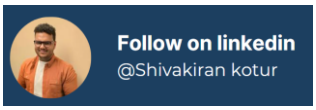
```sql
-- 24) Display employee details for employees who do not earn a commission.
SELECT *
FROM EMP
WHERE COMM IS NULL;
```

| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 1980-12-17 | 800.00 | NULL | 20 |
| 2 | 7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975.00 | NULL | 20 |
| 3 | 7698 | BLAKE | MANAGER | 7839 | 1981-05-01 | 2850.00 | NULL | 30 |
| 4 | 7782 | CLARK | MANAGER | 7839 | 1981-06-09 | 2450.00 | NULL | 10 |
| 5 | 7839 | KING | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL | 10 |
| 6 | 7902 | FORD | ANALYST | 7566 | 1981-12-03 | 3000.00 | NULL | 20 |
| 7 | 7934 | MILLER | CLERK | 7782 | 1982-01-23 | 1300.00 | NULL | 10 |

```sql
-- 25) Display employee names where the department name is ‘SALES’.
SELECT E.ENAME
FROM EMP E
JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE D.DNAME = 'SALES';
```

| | ENAME |
|---|---|
| 1 | ALLEN |
| 2 | WARD |
| 3 | BLAKE |

```sql
-- 26) Select the count of employees in each department where the count is greater than 2.
SELECT DEPTNO, COUNT(*) AS EMPLOYEE_COUNT
FROM EMP
GROUP BY DEPTNO
HAVING COUNT(*) > 2;
```

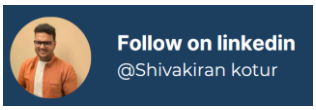


| | DEPTNO | EMPLOYEE_COUNT |
|---|---|---|
| 1 | 10 | 3 |
| 2 | 20 | 3 |
| 3 | 30 | 3 |

```sql
-- 27) Display department names where at least 3 employees are
working.
SELECT D.DNAME
FROM DEPT D
JOIN EMP E ON D.DEPTNO = E.DEPTNO
GROUP BY D.DNAME
HAVING COUNT(*) >= 3;
```

| | DNAME |
|---|---|
| 1 | ACCOUNTING |
| 2 | RESEARCH |
| 3 | SALES |

```sql
-- 28) Display names of managers whose salary is more than the
average salary of employees.
SELECT M.ENAME
FROM EMP M
WHERE M.EMPNO IN (SELECT DISTINCT MGR FROM EMP WHERE MGR IS NOT
NULL)
AND M.SAL > (SELECT AVG(SAL) FROM EMP);

-- Optimized query using joins
SELECT M.ENAME
FROM EMP M
JOIN EMP E ON M.EMPNO = E.MGR
GROUP BY M.ENAME, M.SAL
HAVING M.SAL > (SELECT AVG(SAL) FROM EMP);

--Using CTE
WITH AvgSalary AS (
  SELECT AVG(SAL) AS avg_sal
  FROM EMP
)
SELECT M.ENAME
FROM EMP M
WHERE M.EMPNO IN (SELECT DISTINCT MGR FROM EMP WHERE MGR IS NOT
NULL)
AND M.SAL > (SELECT avg_sal FROM AvgSalary);
```



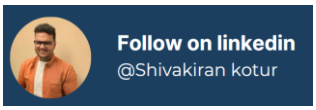
| | ENAME |
|---|---|
| 1 | JONES |
| 2 | BLAKE |
| 3 | CLARK |
| 4 | KING |
| 5 | FORD |

```sql
-- 29) Display names of managers whose salary is more than the min salary of employees. (Duplicate question)
SELECT M.ENAME
FROM EMP M
WHERE M.EMPNO IN (SELECT DISTINCT MGR FROM EMP WHERE MGR IS NOT NULL)
AND M.SAL > (SELECT min(SAL) FROM EMP);

--Give optimized query in comment section
```

| | ENAME |
|---|---|
| 1 | JONES |
| 2 | BLAKE |
| 3 | CLARK |
| 4 | KING |
| 5 | FORD |

```sql
-- 30) Display employee name, salary, commission, and net pay for those employees whose net pay is greater than or equal to any other employee's salary.
SELECT ENAME, SAL, COMM, (SAL + ISNULL(COMM, 0)) AS NET_PAY
FROM EMP
WHERE (SAL + ISNULL(COMM, 0)) >= ALL (SELECT SAL FROM EMP);

--using joins
SELECT E.ENAME, E.SAL, E.COMM, (E.SAL + ISNULL(E.COMM, 0)) AS NET_PAY
FROM EMP E
JOIN (
    SELECT MAX(SAL) AS MAX_SAL
    FROM EMP
) AS MaxSalary
ON (E.SAL + ISNULL(E.COMM, 0)) >= MaxSalary.MAX_SAL;
```

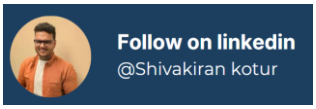




```sql
--Using CTE

WITH MaxSalary AS (
  SELECT MAX(SAL) AS MAX_SAL
  FROM EMP
)
SELECT E.ENAME, E.SAL, E.COMM, (E.SAL + ISNULL(E.COMM, 0)) AS
NET_PAY
FROM EMP E
WHERE (E.SAL + ISNULL(E.COMM, 0)) >= (SELECT MAX_SAL FROM
MaxSalary);
```

| | ENAME | SAL | COMM | NET_PAY |
|---|---|---|---|---|
| 1 | KING | 5000.00 | NULL | 5000.00 |





# SQL Mastery Series – 110 Question using 4 table – Set 6

## Emp Table:

| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 1980-12-17 | 800.00 | NULL | 20 |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 | 1600.00 | 300.00 | 30 |
| 3 | 7521 | WARD | SALESMAN | 7698 | 1981-02-22 | 1250.00 | 500.00 | 30 |
| 4 | 7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975.00 | NULL | 20 |
| 5 | 7698 | BLAKE | MANAGER | 7839 | 1981-05-01 | 2850.00 | NULL | 30 |
| 6 | 7782 | CLARK | MANAGER | 7839 | 1981-06-09 | 2450.00 | NULL | 10 |
| 7 | 7839 | KING | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL | 10 |
| 8 | 7902 | FORD | ANALYST | 7566 | 1981-12-03 | 3000.00 | NULL | 20 |
| 9 | 7934 | MILLER | CLERK | 7782 | 1982-01-23 | 1300.00 | NULL | 10 |

## DEPT

| | DEPTNO | DNAME | LOC |
|---|---|---|---|
| 1 | 10 | ACCOUNTING | NEW YORK |
| 2 | 20 | RESEARCH | DALLAS |
| 3 | 30 | SALES | CHICAGO |
| 4 | 40 | OPERATIONS | BOSTON |

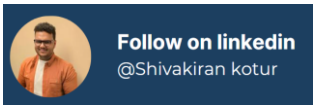
## Salgrade

| | GRADE | LOSAL | HISAL |
|---|---|---|---|
| 1 | 1 | 700.00 | 1200.00 |
| 2 | 2 | 1201.00 | 1400.00 |
| 3 | 3 | 1401.00 | 2000.00 |
| 4 | 4 | 2001.00 | 3000.00 |
| 5 | 5 | 3001.00 | 9999.00 |

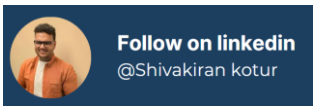
## JobHistory

| | EMPNO | JOB | STARTDATE | ENDDATE | DEPTNO |
|---|---|---|---|---|---|
| 1 | 7369 | INTERN | 2019-01-01 | 2020-12-31 | 20 |
| 2 | 7369 | ASSISTANT CLERK | 2021-01-01 | 2022-12-31 | 20 |
| 3 | 7499 | JUNIOR SALESMAN | 2018-03-01 | 2019-12-31 | 30 |
| 4 | 7499 | SALESMAN | 2020-01-01 | 2021-12-31 | 30 |
| 5 | 7521 | TRAINEE | 2017-05-15 | 2019-05-15 | 30 |
| 6 | 7521 | SALESMAN | 2019-05-16 | 2020-12-31 | 30 |
| 7 | 7566 | ASSISTANT MANAGER | 2015-04-02 | 2018-12-31 | 20 |
| 8 | 7566 | MANAGER | 2019-01-01 | 2021-12-31 | 20 |
| 9 | 7698 | ASSISTANT MANAGER | 2016-05-01 | 2019-04-30 | 30 |
| 10 | 7698 | MANAGER | 2019-05-01 | 2022-12-31 | 30 |
| 11 | 7782 | SUPERVISOR | 2016-06-09 | 2018-06-08 | 10 |
| 12 | 7782 | MANAGER | 2018-06-09 | 2021-06-08 | 10 |
| 13 | 7839 | VICE PRESIDENT | 2010-11-17 | 2015-11-16 | 10 |
| 14 | 7839 | PRESIDENT | 2015-11-17 | 2023-12-31 | 10 |
| 15 | 7902 | SENIOR ANALYST | 2017-12-03 | 2019-12-02 | 20 |
| 16 | 7902 | ANALYST | 2019-12-03 | 2022-12-02 | 20 |
| 17 | 7934 | JUNIOR CLERK | 2016-01-23 | 2018-01-22 | 10 |
| 18 | 7934 | CLERK | 2018-01-23 | 2020-01-22 | 10 |



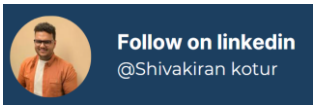


```sql
-- 41) Display those employees whose salary is an odd value.
SELECT ENAME
FROM EMP
WHERE SAL % 2 = 1;

-- 42) Display those employees whose salary contains at least 3
digits.
SELECT ENAME
FROM EMP
WHERE LEN(SAL) >= 3;

-- 43) Display those employees who joined the company in the month
of December.
SELECT ENAME
FROM EMP
WHERE MONTH(HIREDATE) = 12;

-- 44) Display those employees who joined on 1-Jan-81.
SELECT ENAME
FROM EMP
WHERE HIREDATE = '1981-01-01';

-- 45) Display the names of employees who are working in a
department located in CHICAGO.
SELECT ENAME
FROM EMP E
JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE D.LOC = 'CHICAGO';

-- 46) Display the average salary of employees, but exclude the
average salary of clerks from the result.
SELECT AVG(SAL) AS AVERAGE_SALARY
FROM EMP
WHERE JOB <> 'CLERK';

-- 47) Display the names of employees who are working in a
department with the highest average salary.
SELECT ENAME
FROM EMP
WHERE DEPTNO = (
  SELECT TOP 1 DEPTNO
  FROM EMP
  GROUP BY DEPTNO
```



```
  ORDER BY AVG(SAL) DESC
);


-- 48) Display the details of employees who are getting the same
salary as the minimum salary of any department.
SELECT *
FROM EMP
WHERE SAL IN (
  SELECT MIN(SAL)
  FROM EMP
  GROUP BY DEPTNO
);

-- 49) Display the names of employees who joined in the year 1981
and are not getting any commission.
SELECT ENAME
FROM EMP
WHERE YEAR(HIREDATE) = 1981
AND COMM IS NULL;

-- 50) Display employees who are working in the same department as
‘JONES’ or ‘SCOTT’.
SELECT ENAME
FROM EMP
WHERE DEPTNO IN (
  SELECT DEPTNO
  FROM EMP
  WHERE ENAME IN ('JONES', 'SCOTT')
);
```





# SQL Mastery Series – 110 Question using 4 table – Set 7

## Emp Table:

| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 1980-12-17 | 800.00 | NULL | 20 |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 | 1600.00 | 300.00 | 30 |
| 3 | 7521 | WARD | SALESMAN | 7698 | 1981-02-22 | 1250.00 | 500.00 | 30 |
| 4 | 7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975.00 | NULL | 20 |
| 5 | 7698 | BLAKE | MANAGER | 7839 | 1981-05-01 | 2850.00 | NULL | 30 |
| 6 | 7782 | CLARK | MANAGER | 7839 | 1981-06-09 | 2450.00 | NULL | 10 |
| 7 | 7839 | KING | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL | 10 |
| 8 | 7902 | FORD | ANALYST | 7566 | 1981-12-03 | 3000.00 | NULL | 20 |
| 9 | 7934 | MILLER | CLERK | 7782 | 1982-01-23 | 1300.00 | NULL | 10 |

## DEPT

| | DEPTNO | DNAME | LOC |
|---|---|---|---|
| 1 | 10 | ACCOUNTING | NEW YORK |
| 2 | 20 | RESEARCH | DALLAS |
| 3 | 30 | SALES | CHICAGO |
| 4 | 40 | OPERATIONS | BOSTON |

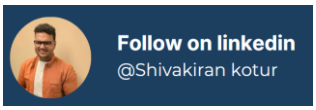
## Salgrade

| | GRADE | LOSAL | HISAL |
|---|---|---|---|
| 1 | 1 | 700.00 | 1200.00 |
| 2 | 2 | 1201.00 | 1400.00 |
| 3 | 3 | 1401.00 | 2000.00 |
| 4 | 4 | 2001.00 | 3000.00 |
| 5 | 5 | 3001.00 | 9999.00 |

## JobHistory

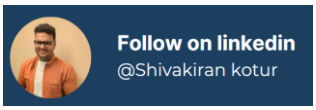
| | EMPNO | JOB | STARTDATE | ENDDATE | DEPTNO |
|---|---|---|---|---|---|
| 1 | 7369 | INTERN | 2019-01-01 | 2020-12-31 | 20 |
| 2 | 7369 | ASSISTANT CLERK | 2021-01-01 | 2022-12-31 | 20 |
| 3 | 7499 | JUNIOR SALESMAN | 2018-03-01 | 2019-12-31 | 30 |
| 4 | 7499 | SALESMAN | 2020-01-01 | 2021-12-31 | 30 |
| 5 | 7521 | TRAINEE | 2017-05-15 | 2019-05-15 | 30 |
| 6 | 7521 | SALESMAN | 2019-05-16 | 2020-12-31 | 30 |
| 7 | 7566 | ASSISTANT MANAGER | 2015-04-02 | 2018-12-31 | 20 |
| 8 | 7566 | MANAGER | 2019-01-01 | 2021-12-31 | 20 |
| 9 | 7698 | ASSISTANT MANAGER | 2016-05-01 | 2019-04-30 | 30 |
| 10 | 7698 | MANAGER | 2019-05-01 | 2022-12-31 | 30 |
| 11 | 7782 | SUPERVISOR | 2016-06-09 | 2018-06-08 | 10 |
| 12 | 7782 | MANAGER | 2018-06-09 | 2021-06-08 | 10 |
| 13 | 7839 | VICE PRESIDENT | 2010-11-17 | 2015-11-16 | 10 |
| 14 | 7839 | PRESIDENT | 2015-11-17 | 2023-12-31 | 10 |
| 15 | 7902 | SENIOR ANALYST | 2017-12-03 | 2019-12-02 | 20 |
| 16 | 7902 | ANALYST | 2019-12-03 | 2022-12-02 | 20 |
| 17 | 7934 | JUNIOR CLERK | 2016-01-23 | 2018-01-22 | 10 |
| 18 | 7934 | CLERK | 2018-01-23 | 2020-01-22 | 10 |





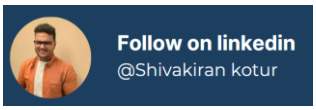
```sql
-- 51) Display employee names whose total earnings (salary + commission) is greater than the average earnings of the company.
SELECT ENAME
FROM EMP
WHERE (SAL + ISNULL(COMM, 0)) > (
  SELECT AVG(SAL + ISNULL(COMM, 0))
  FROM EMP
);

-- 52) Display the names of employees who earn a commission.
SELECT ENAME
FROM EMP
WHERE COMM IS NOT NULL;

-- 53) Display the names of employees whose salary is the same as the lowest salary in their department.
SELECT ENAME
FROM EMP E
WHERE SAL = (
  SELECT MIN(SAL)
  FROM EMP
  WHERE DEPTNO = E.DEPTNO
);

-- 54) Display those employees who joined on the 10th of any month.
SELECT ENAME
FROM EMP
WHERE DAY(HIREDATE) = 10;

-- 55) Display the names of employees who have not joined in the year 1981.
SELECT ENAME
FROM EMP
WHERE YEAR(HIREDATE) <> 1981;

-- 56) Display the names of employees who are not clerks and whose salary is not more than 3000.
SELECT ENAME
FROM EMP
WHERE JOB <> 'CLERK'
AND SAL <= 3000;
```





```sql
-- 57) Display the names of employees who have not joined in the
month of December.
SELECT ENAME
FROM EMP
WHERE MONTH(HIREDATE) <> 12;

-- 58) Display the names of employees who do not earn any
commission.
SELECT ENAME
FROM EMP
WHERE COMM IS NULL;

-- 59) Display the names of employees who earn more than the
minimum salary of their department.
SELECT ENAME
FROM EMP E
WHERE SAL > (
  SELECT MIN(SAL)
  FROM EMP
  WHERE DEPTNO = E.DEPTNO
);

-- 60) Display the names of employees who work in a department
located in CHICAGO or DALLAS.
SELECT ENAME
FROM EMP E
JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE D.LOC IN ('CHICAGO', 'DALLAS');

-- 61) Display the names of employees whose manager is working in
the same department as ‘KING’.
SELECT E.ENAME
FROM EMP E
JOIN EMP M ON E.MGR = M.EMPNO
WHERE M.DEPTNO = (SELECT DEPTNO FROM EMP WHERE ENAME = 'KING');

-- 62) Display the names of employees whose commission is less than
200.
SELECT ENAME
FROM EMP
WHERE COMM < 200;
```





```sql
-- 63) Display the names of employees whose salary is less than or equal to their manager’s salary.
SELECT E.ENAME
FROM EMP E
JOIN EMP M ON E.MGR = M.EMPNO
WHERE E.SAL <= M.SAL;

-- 64) Display the names of employees who joined in the first half of the year.
SELECT ENAME
FROM EMP
WHERE MONTH(HIREDATE) <= 6;

-- 65) Display the names of employees who have a higher salary than any clerk.
SELECT ENAME
FROM EMP
WHERE SAL > ANY (SELECT SAL FROM EMP WHERE JOB = 'CLERK');
```