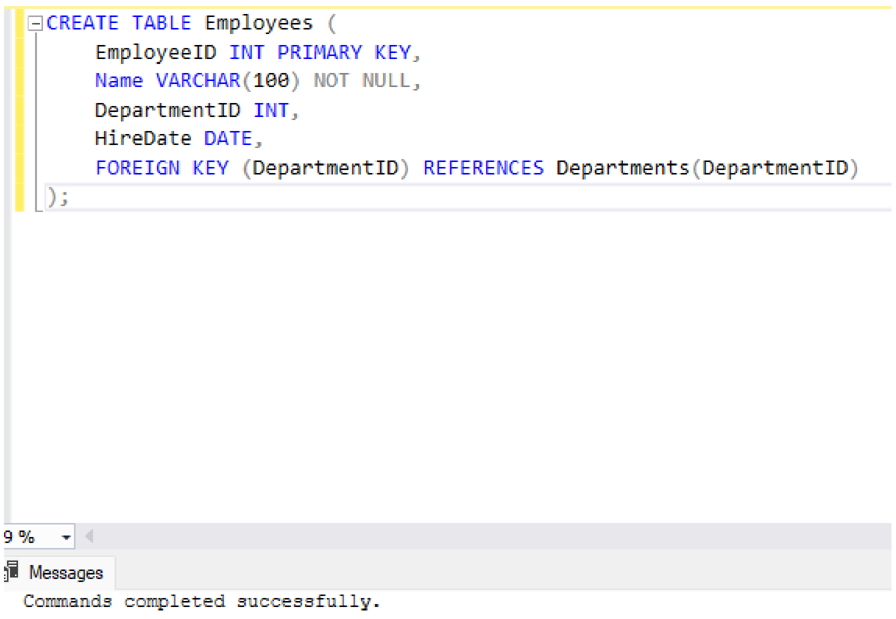
Database Task: Employee Management System

1. **Database Schema:**
Employees table:



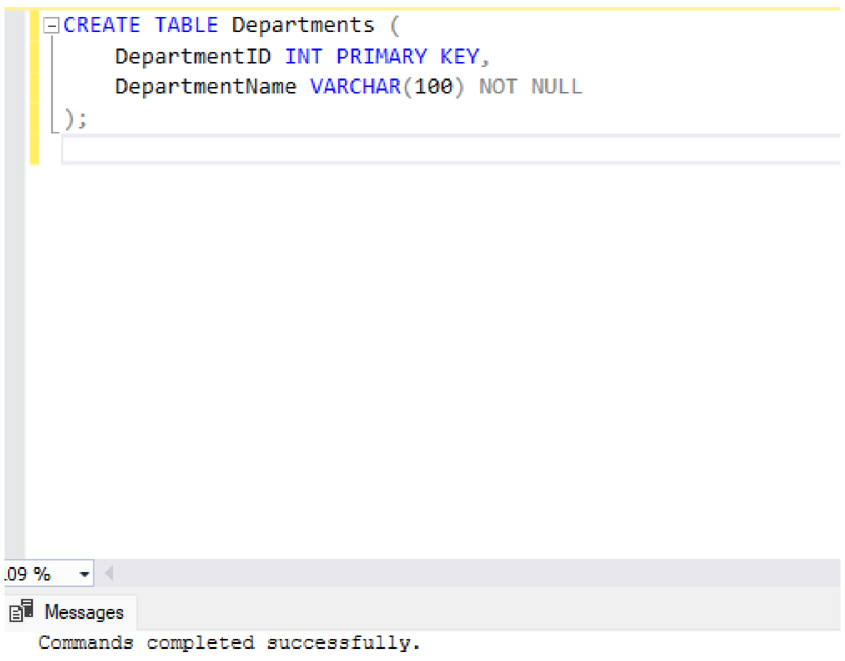
```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    DepartmentID INT,
    HireDate DATE,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

9 %

Messages

```
Commands completed successfully.
```

Departments table:



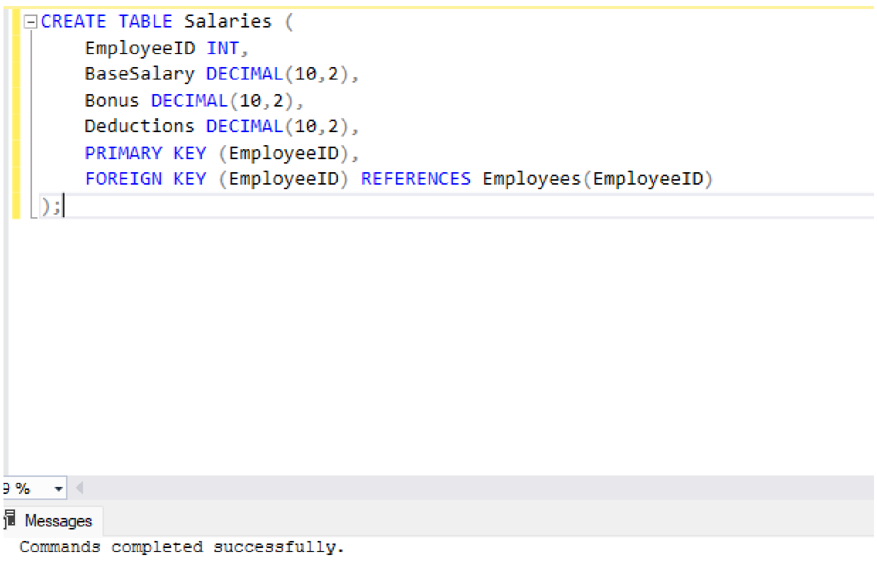
```
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(100) NOT NULL
);
```

.09 %

Messages

```
Commands completed successfully.
```

Salaries table

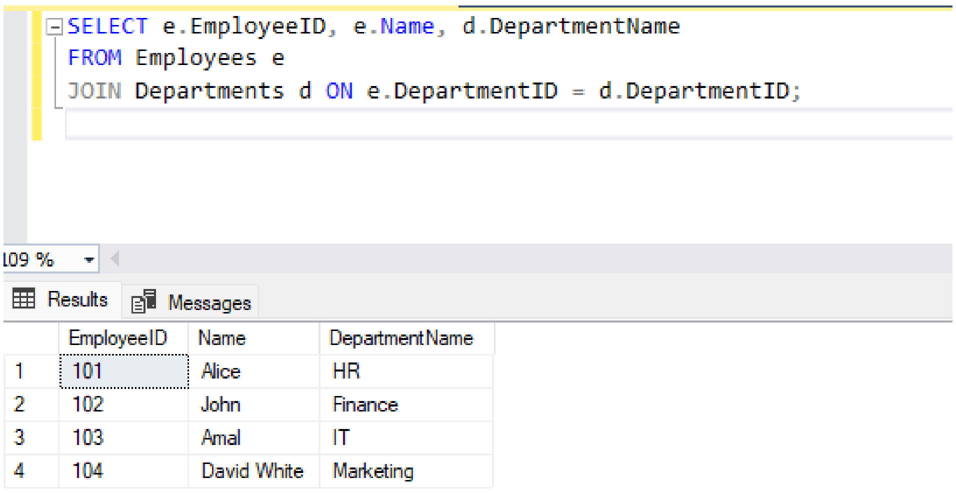


```
CREATE TABLE Salaries (
    EmployeeID INT,
    BaseSalary DECIMAL(10,2),
    Bonus DECIMAL(10,2),
    Deductions DECIMAL(10,2),
    PRIMARY KEY (EmployeeID),
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);
```

```
Commands completed successfully.
```

2. **SQL Queries:**

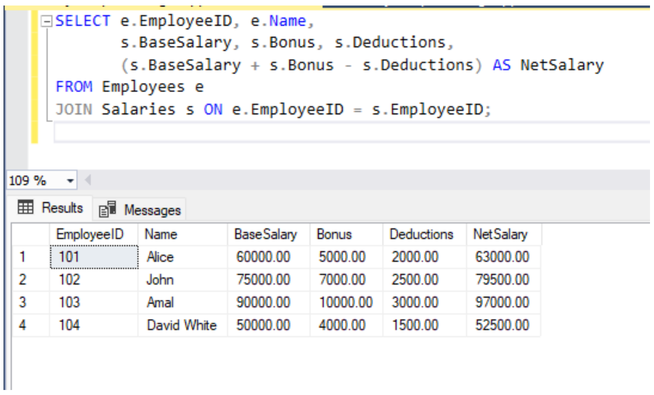
List all employees along with their department names.



```
SELECT e.EmployeeID, e.Name, d.DepartmentName
FROM Employees e
JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

	EmployeeID	Name	DepartmentName
1	101	Alice	HR
2	102	John	Finance
3	103	Amal	IT
4	104	David White	Marketing

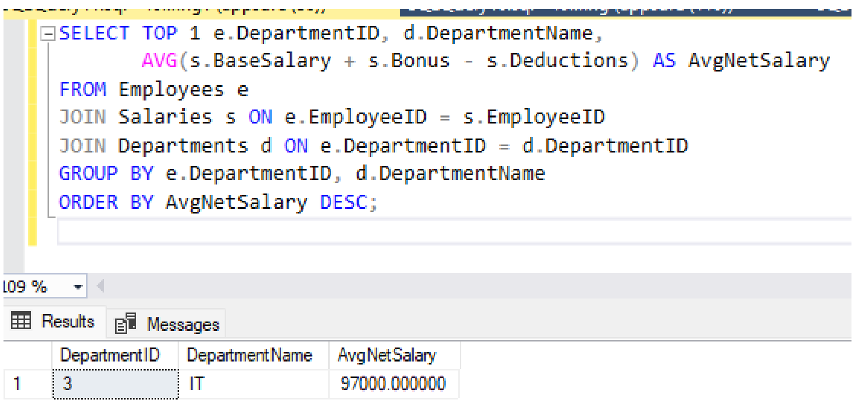
Calculate the net salary for each employee using: **Net Salary = BaseSalary + Bonus - Deductions.**



```
SELECT e.EmployeeID, e.Name,
       s.BaseSalary, s.Bonus, s.Deductions,
       (s.BaseSalary + s.Bonus - s.Deductions) AS NetSalary
FROM Employees e
JOIN Salaries s ON e.EmployeeID = s.EmployeeID;
```

	EmployeeID	Name	BaseSalary	Bonus	Deductions	NetSalary
1	101	Alice	60000.00	5000.00	2000.00	63000.00
2	102	John	75000.00	7000.00	2500.00	79500.00
3	103	Amal	90000.00	10000.00	3000.00	97000.00
4	104	David White	50000.00	4000.00	1500.00	52500.00

Identify the department with the highest average salary.



```
SELECT TOP 1 e.DepartmentID, d.DepartmentName,
       AVG(s.BaseSalary + s.Bonus - s.Deductions) AS AvgNetSalary
FROM Employees e
JOIN Salaries s ON e.EmployeeID = s.EmployeeID
JOIN Departments d ON e.DepartmentID = d.DepartmentID
GROUP BY e.DepartmentID, d.DepartmentName
ORDER BY AvgNetSalary DESC;
```

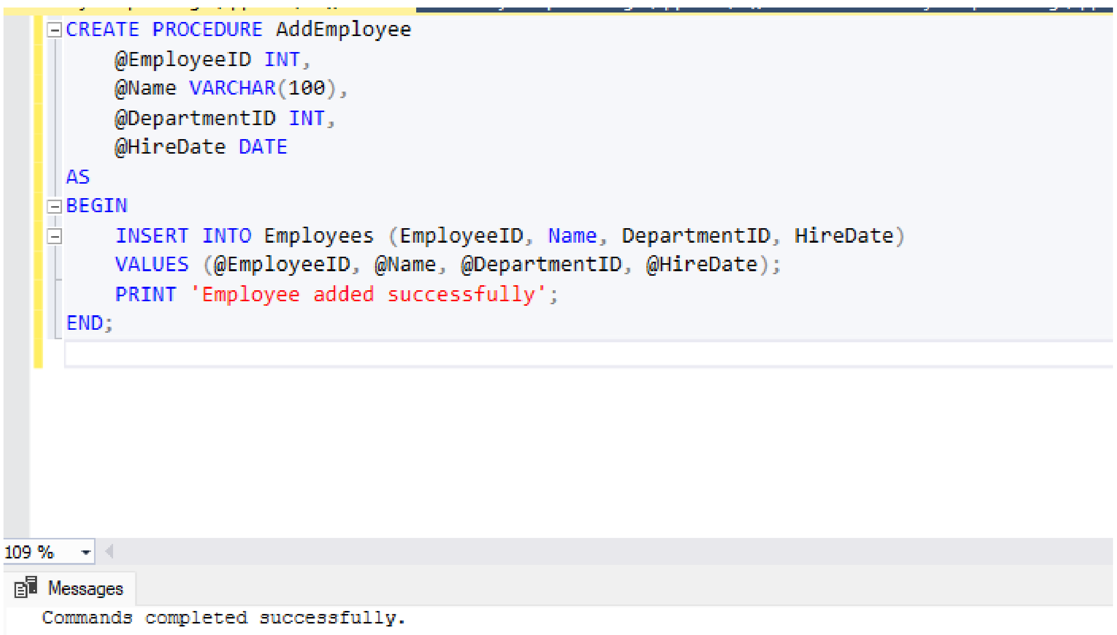
109 %

Results | Messages

	DepartmentID	DepartmentName	AvgNetSalary
1	3	IT	97000.000000

3. **Stored Procedures:**

Add Employee: A procedure to insert a new employee into the `Employees` table, ensuring valid `DepartmentID` and other constraints.

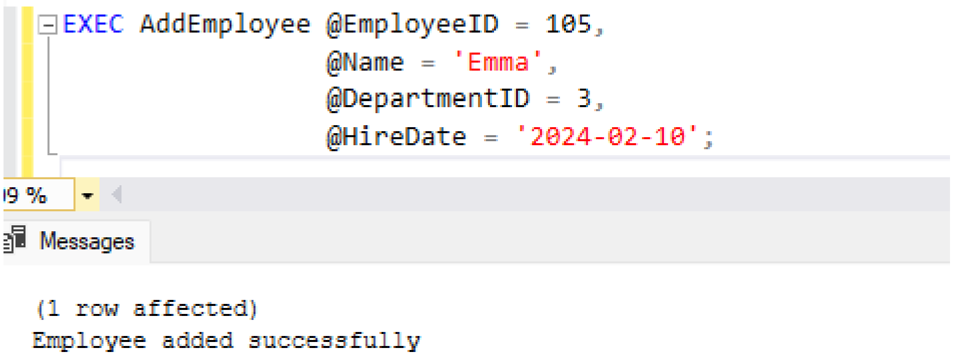


```
CREATE PROCEDURE AddEmployee
    @EmployeeID INT,
    @Name VARCHAR(100),
    @DepartmentID INT,
    @HireDate DATE
AS
BEGIN
    INSERT INTO Employees (EmployeeID, Name, DepartmentID, HireDate)
    VALUES (@EmployeeID, @Name, @DepartmentID, @HireDate);
    PRINT 'Employee added successfully';
END;
```

109 %

Messages

```
Commands completed successfully.
```

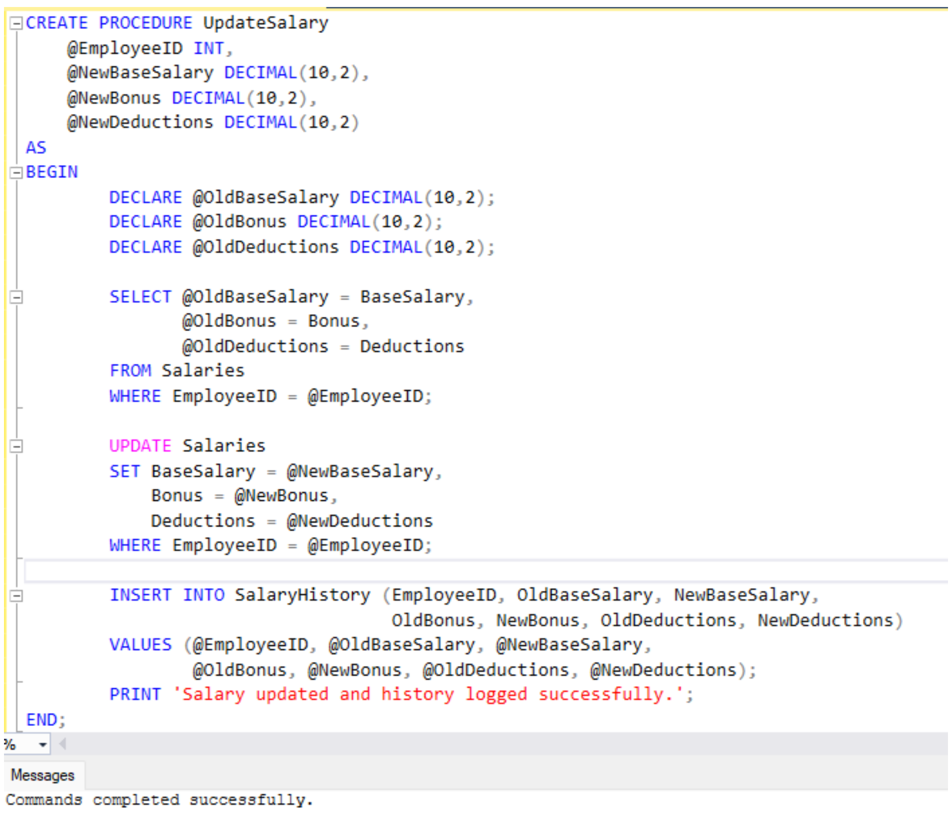


```
EXEC AddEmployee @EmployeeID = 105,
                 @Name = 'Emma',
                 @DepartmentID = 3,
                 @HireDate = '2024-02-10';
```

Messages

```
(1 row affected)
Employee added successfully
```

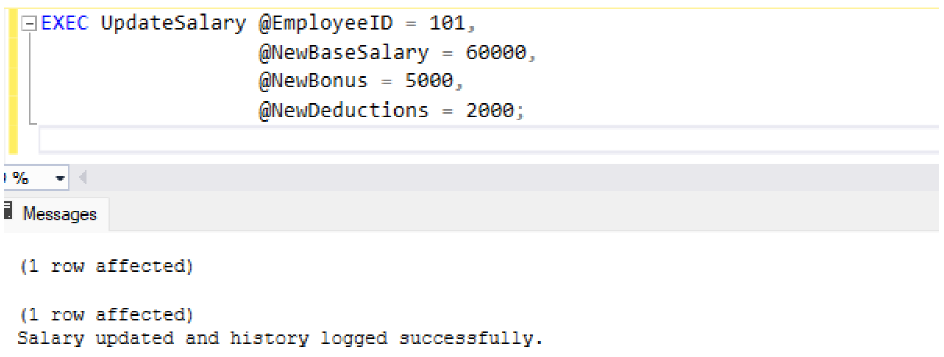
Update Salary: A procedure to update the salary details of an employee, automatically logging the changes into a `SalaryHistory` table.



```
CREATE PROCEDURE UpdateSalary
    @EmployeeID INT,
    @NewBaseSalary DECIMAL(10,2),
    @NewBonus DECIMAL(10,2),
    @NewDeductions DECIMAL(10,2)
AS
BEGIN
        DECLARE @OldBaseSalary DECIMAL(10,2);
        DECLARE @OldBonus DECIMAL(10,2);
        DECLARE @OldDeductions DECIMAL(10,2);

        SELECT @OldBaseSalary = BaseSalary,
               @OldBonus = Bonus,
               @OldDeductions = Deductions
        FROM Salaries
        WHERE EmployeeID = @EmployeeID;

        UPDATE Salaries
        SET BaseSalary = @NewBaseSalary,
            Bonus = @NewBonus,
            Deductions = @NewDeductions
        WHERE EmployeeID = @EmployeeID;

        INSERT INTO SalaryHistory (EmployeeID, OldBaseSalary, NewBaseSalary,
                                   OldBonus, NewBonus, OldDeductions, NewDeductions)
        VALUES (@EmployeeID, @OldBaseSalary, @NewBaseSalary,
                @OldBonus, @NewBonus, @OldDeductions, @NewDeductions);
        PRINT 'Salary updated and history logged successfully.';
END;
```

Messages

```
Commands completed successfully.
```

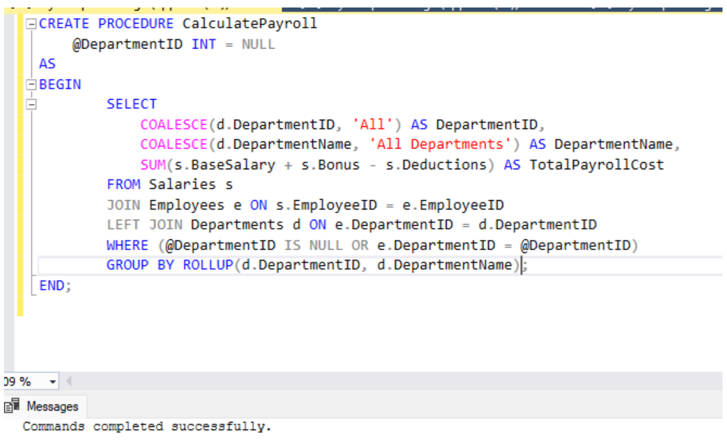


```
EXEC UpdateSalary @EmployeeID = 101,
                  @NewBaseSalary = 60000,
                  @NewBonus = 5000,
                  @NewDeductions = 2000;
```

Messages

```
(1 row affected)

(1 row affected)
Salary updated and history logged successfully.
```

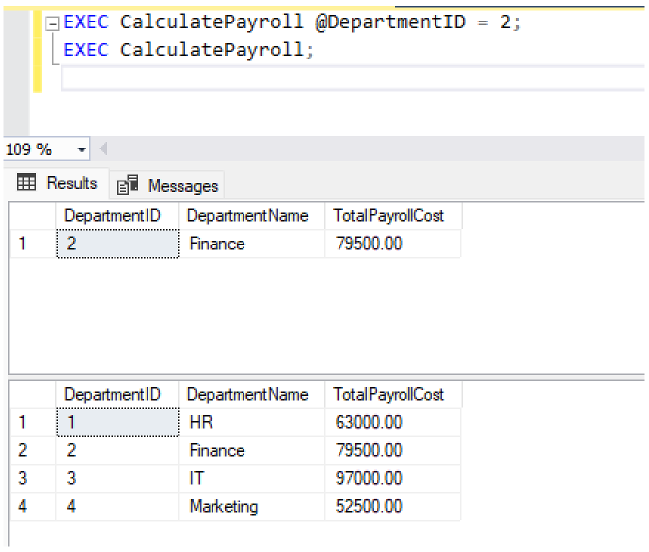
Calculate Payroll: A procedure to compute and return the total payroll cost for a department or the entire organization.



```
CREATE PROCEDURE CalculatePayroll
    @DepartmentID INT = NULL
AS
BEGIN
        SELECT
            COALESCE(d.DepartmentID, 'All') AS DepartmentID,
            COALESCE(d.DepartmentName, 'All Departments') AS DepartmentName,
            SUM(s.BaseSalary + s.Bonus - s.Deductions) AS TotalPayrollCost
        FROM Salaries s
        JOIN Employees e ON s.EmployeeID = e.EmployeeID
        LEFT JOIN Departments d ON e.DepartmentID = d.DepartmentID
        WHERE (@DepartmentID IS NULL OR e.DepartmentID = @DepartmentID)
        GROUP BY ROLLUP(d.DepartmentID, d.DepartmentName);
END;
```

Messages

```
Commands completed successfully.
```


```sql
EXEC CalculatePayroll @DepartmentID = 2;
EXEC CalculatePayroll;
```

109 %

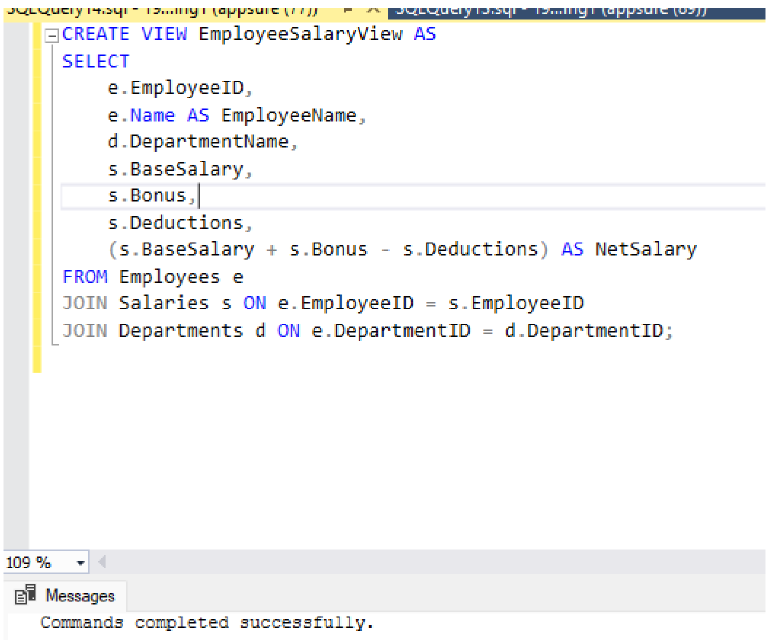
Results | Messages

	DepartmentID	DepartmentName	TotalPayrollCost
1	2	Finance	79500.00

	DepartmentID	DepartmentName	TotalPayrollCost
1	1	HR	63000.00
2	2	Finance	79500.00
3	3	IT	97000.00
4	4	Marketing	52500.00

4. Views:

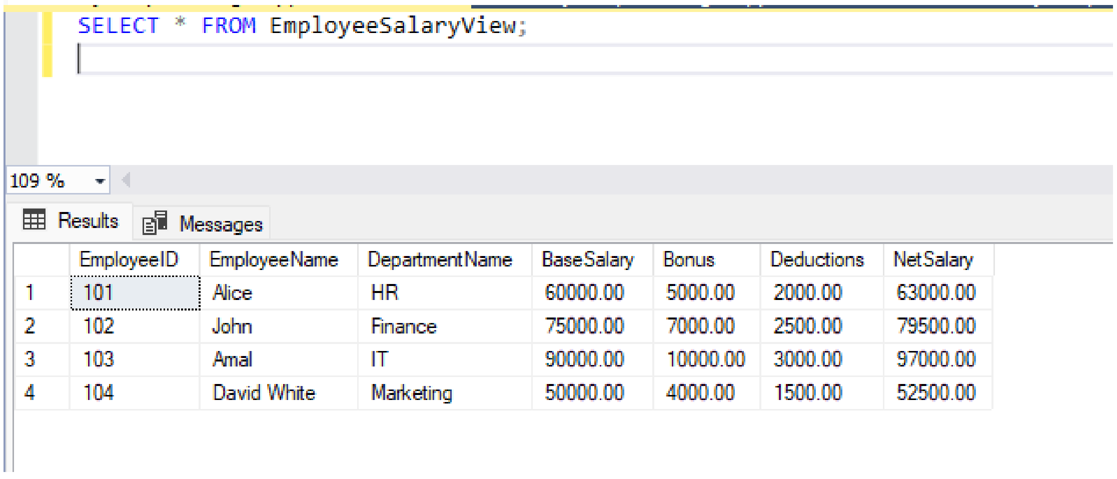
EmployeeSalaryView: A view that combines `Employees`, `Departments`, and `Salaries` to provide a detailed report of employee salaries with department names and net salaries.


```sql
CREATE VIEW EmployeeSalaryView AS
SELECT
    e.EmployeeID,
    e.Name AS EmployeeName,
    d.DepartmentName,
    s.BaseSalary,
    s.Bonus,
    s.Deductions,
    (s.BaseSalary + s.Bonus - s.Deductions) AS NetSalary
FROM Employees e
JOIN Salaries s ON e.EmployeeID = s.EmployeeID
JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

109 %

Messages

```
Commands completed successfully.
```

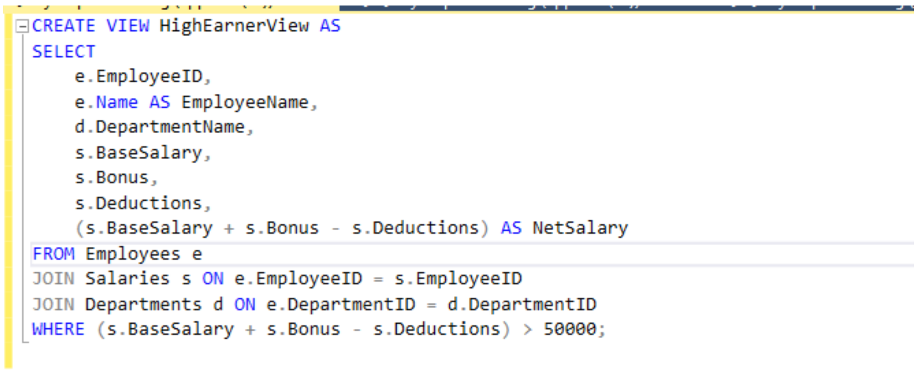

```sql
SELECT * FROM EmployeeSalaryView;
```

109 %

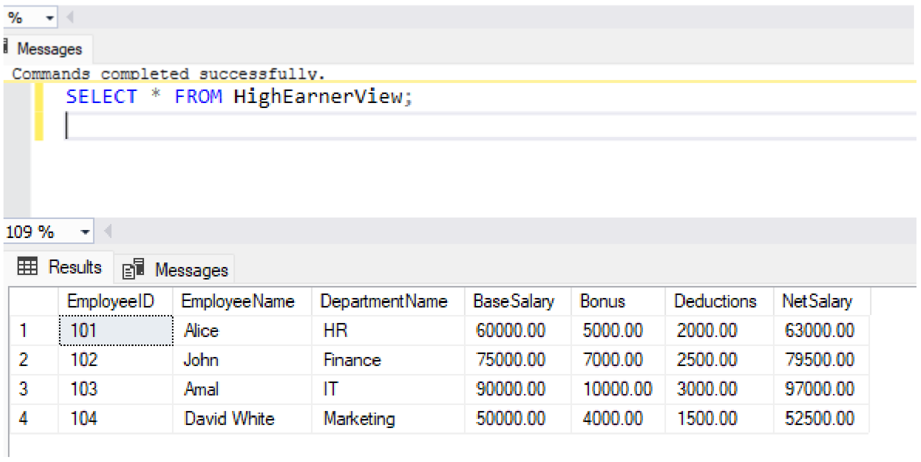
Results | Messages

	EmployeeID	EmployeeName	DepartmentName	BaseSalary	Bonus	Deductions	NetSalary
1	101	Alice	HR	60000.00	5000.00	2000.00	63000.00
2	102	John	Finance	75000.00	7000.00	2500.00	79500.00
3	103	Amal	IT	90000.00	10000.00	3000.00	97000.00
4	104	David White	Marketing	50000.00	4000.00	1500.00	52500.00

HighEarnerView: A view that lists employees earning above a certain threshold (e.g., a parameter or predefined limit).



```sql
CREATE VIEW HighEarnerView AS
SELECT
    e.EmployeeID,
    e.Name AS EmployeeName,
    d.DepartmentName,
    s.BaseSalary,
    s.Bonus,
    s.Deductions,
    (s.BaseSalary + s.Bonus - s.Deductions) AS NetSalary
FROM Employees e
JOIN Salaries s ON e.EmployeeID = s.EmployeeID
JOIN Departments d ON e.DepartmentID = d.DepartmentID
WHERE (s.BaseSalary + s.Bonus - s.Deductions) > 50000;
```



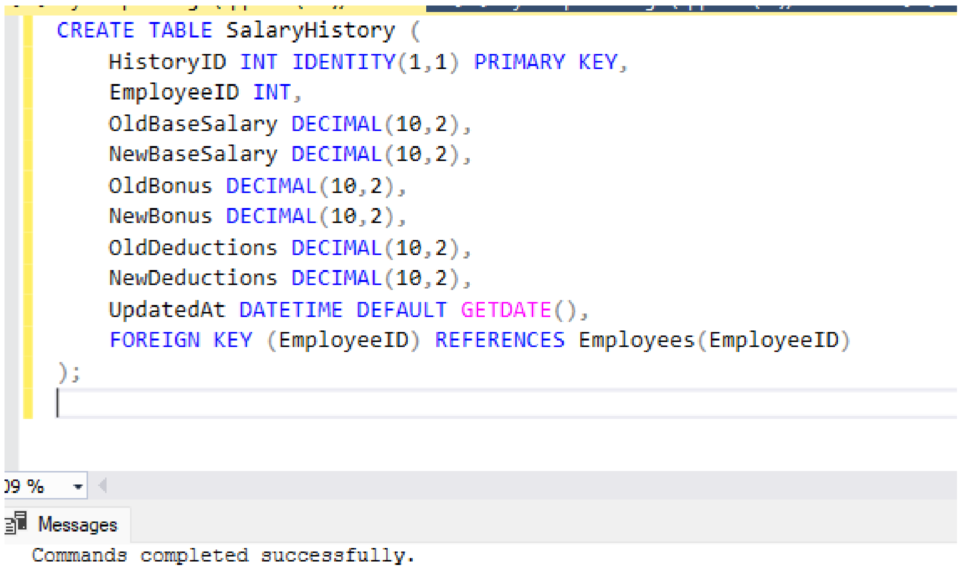
```
Messages
Commands completed successfully.
```

```sql
SELECT * FROM HighEarnerView;
```

```
109 %
Results    Messages
```

	EmployeeID	EmployeeName	DepartmentName	BaseSalary	Bonus	Deductions	NetSalary
1	101	Alice	HR	60000.00	5000.00	2000.00	63000.00
2	102	John	Finance	75000.00	7000.00	2500.00	79500.00
3	103	Amal	IT	90000.00	10000.00	3000.00	97000.00
4	104	David White	Marketing	50000.00	4000.00	1500.00	52500.00

5. **Bonus Tasks:**
   Add a `SalaryHistory` table to log salary updates with triggers.



```sql
CREATE TABLE SalaryHistory (
    HistoryID INT IDENTITY(1,1) PRIMARY KEY,
    EmployeeID INT,
    OldBaseSalary DECIMAL(10,2),
    NewBaseSalary DECIMAL(10,2),
    OldBonus DECIMAL(10,2),
    NewBonus DECIMAL(10,2),
    OldDeductions DECIMAL(10,2),
    NewDeductions DECIMAL(10,2),
    UpdatedAt DATETIME DEFAULT GETDATE(),
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);
```

```
Messages
Commands completed successfully.
```