**Stored Procedure -**

1)

CREATE PROCEDURE GetCustomerInfo

@customer\_id VARCHAR(5),

@first\_name VARCHAR(50) OUTPUT,

@last\_name VARCHAR(50) OUTPUT,

@address VARCHAR(100) OUTPUT,

@city VARCHAR(50) OUTPUT

AS

BEGIN

SELECT

@first\_name = First\_Name,

@last\_name = Last\_Name,

@address = Address,

@city = City

FROM Customer

WHERE Customer\_ID = @customer\_id;

END;

-- Declare variables for input and output parameters

DECLARE @CustomerID VARCHAR(5) = 'C001';

DECLARE @FirstName VARCHAR(50);

DECLARE @LastName VARCHAR(50);

DECLARE @CustomerAddress VARCHAR(100); -- Renamed the variable to avoid conflicts

DECLARE @CustomerCity VARCHAR(50);

-- Execute the stored procedure

EXEC GetCustomerInfo

@customer\_id = @CustomerID,

@first\_name = @FirstName OUTPUT,

@last\_name = @LastName OUTPUT,

@address = @CustomerAddress OUTPUT,

@city = @CustomerCity OUTPUT;

-- Display the result

SELECT

'Customer ID: ' + @CustomerID AS 'Input Parameter',

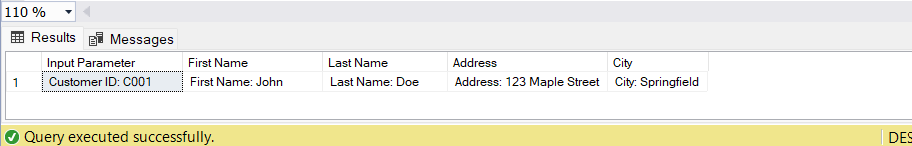
'First Name: ' + ISNULL(@FirstName, 'N/A') AS 'First Name',

'Last Name: ' + ISNULL(@LastName, 'N/A') AS 'Last Name',

'Address: ' + ISNULL(@CustomerAddress, 'N/A') AS 'Address',

'City: ' + ISNULL(@CustomerCity, 'N/A') AS 'City';

Output -



2)

CREATE PROCEDURE GetBillingDetails

@invoice\_number VARCHAR(50),

@customer\_id VARCHAR(5) OUTPUT,

@plan\_name VARCHAR(100) OUTPUT,

@payment\_status VARCHAR(20) OUTPUT

AS

BEGIN

SELECT

@customer\_id = c.Customer\_ID,

@plan\_name = p.Plan\_Name,

@payment\_status = b.Payment\_Status

FROM Billing b

INNER JOIN Customer c ON b.Customer\_ID = c.Customer\_ID

INNER JOIN Plans p ON b.Plan\_ID = p.Plan\_ID

WHERE b.Invoice\_Number = @invoice\_number;

END;

-- Declare variables for input and output parameters

DECLARE @InvoiceNumber VARCHAR(50) = 'INV0001';

DECLARE @CustomerID VARCHAR(5);

DECLARE @PlanName VARCHAR(100);

DECLARE @PaymentStatus VARCHAR(20);

-- Execute the stored procedure

EXEC GetBillingDetails

@invoice\_number = @InvoiceNumber,

@customer\_id = @CustomerID OUTPUT,

@plan\_name = @PlanName OUTPUT,

@payment\_status = @PaymentStatus OUTPUT;

-- Display the result

SELECT

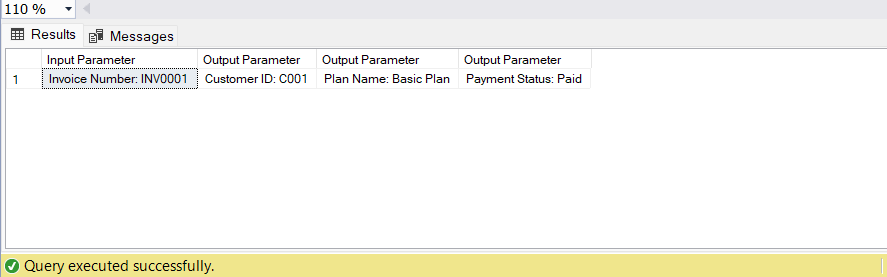
'Invoice Number: ' + @InvoiceNumber AS 'Input Parameter',

'Customer ID: ' + ISNULL(@CustomerID, 'N/A') AS 'Output Parameter',

'Plan Name: ' + ISNULL(@PlanName, 'N/A') AS 'Output Parameter',

'Payment Status: ' + ISNULL(@PaymentStatus, 'N/A') AS 'Output Parameter';

**Output →**



3)

CREATE PROCEDURE UpdateEmployeeSalary

@employee\_id VARCHAR(5),

@new\_salary DECIMAL(10, 2),

@message VARCHAR(255) OUTPUT

AS

BEGIN

UPDATE Employee

SET Salary = @new\_salary

WHERE Employee\_ID = @employee\_id;

IF @@ROWCOUNT > 0

SET @message = 'Salary updated successfully.';

ELSE

SET @message = 'Employee not found.';

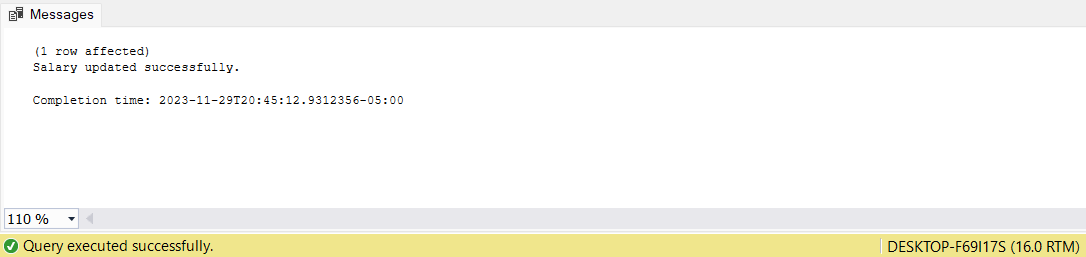
END;

DECLARE @outputMessage NVARCHAR(255);

EXEC UpdateEmployeeSalary 'E001', 80000.00, @message = @outputMessage OUTPUT;

PRINT @outputMessage;

**Output →**



4)

CREATE PROCEDURE CalculateEmployeeBonus

@EmployeeID VARCHAR(5),

@BonusPercentage DECIMAL(5, 2),

@BonusAmount DECIMAL(10, 2) OUTPUT

AS

BEGIN

DECLARE @EmployeeSalary DECIMAL(10, 2);

-- Get the employee's salary

SELECT @EmployeeSalary = Salary

FROM Employee

WHERE Employee\_ID = @EmployeeID;

-- Calculate the bonus

SET @BonusAmount = @EmployeeSalary \* (@BonusPercentage / 100);

END;

-- Declare variables for input and output parameters

DECLARE @employeeID VARCHAR(5) = 'E001';

DECLARE @bonusPercentage DECIMAL(5, 2) = 10;

DECLARE @bonusAmount DECIMAL(10, 2);

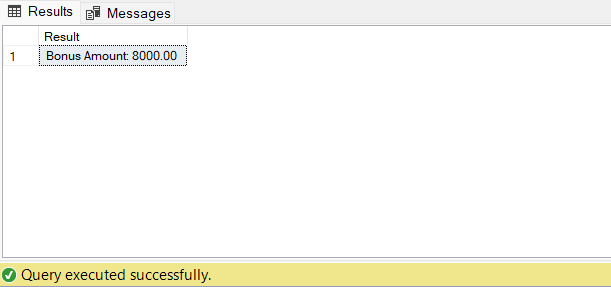
-- Execute the stored procedure

EXEC CalculateEmployeeBonus @employeeID, @bonusPercentage, @bonusAmount OUTPUT;

-- Display the calculated bonus

SELECT 'Bonus Amount: ' + CAST(@bonusAmount AS VARCHAR(20)) AS Result;

**Output →**



**VIEW (often used for reporting purposes) →**

1.

CREATE VIEW CustomerBillingView AS

SELECT

c.Customer\_ID,

c.First\_Name,

c.Last\_Name,

b.Invoice\_Number,

b.Payment\_Status

FROM Customer c

INNER JOIN Billing b ON c.Customer\_ID = b.Customer\_ID;

SELECT \* FROM CustomerBillingView;

**Output →**



2)

CREATE VIEW EmployeeDepartmentView AS

SELECT

e.Employee\_ID,

e.First\_Name,

e.Last\_Name,

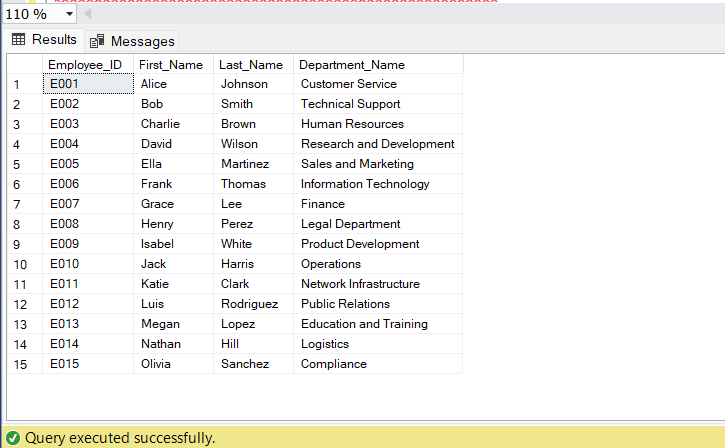
d.Department\_Name

FROM Employee e

INNER JOIN Department d ON e.Department\_ID = d.Department\_ID;

SELECT \* FROM EmployeeDepartmentView;

**Output →**



3)

CREATE VIEW ServiceSubscriptionDetailsView AS

SELECT

ss.Subscription\_ID,

ss.Start\_Date,

ss.End\_Date,

p.Plan\_Name,

s.Service\_Type

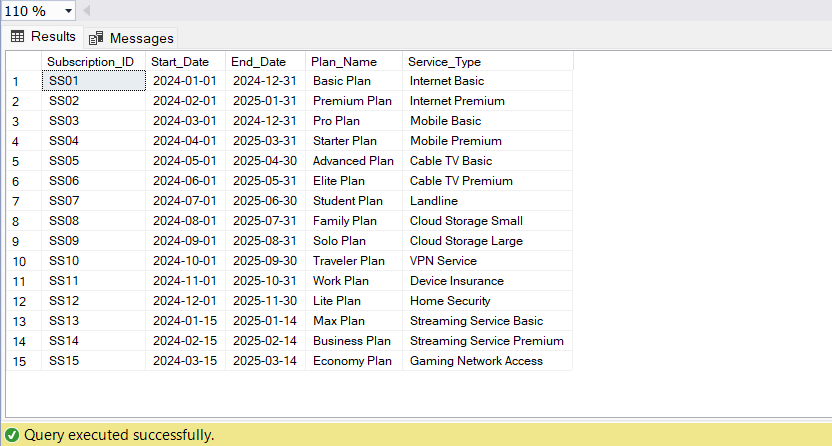
FROM ServiceSubscription ss

INNER JOIN Plans p ON ss.Plan\_ID = p.Plan\_ID

INNER JOIN Service s ON ss.Service\_ID = s.Service\_ID;

SELECT \* FROM ServiceSubscriptionDetailsView;

**Output →**



4)

-- View 4th Code

CREATE VIEW EmployeeLocationView AS

SELECT

E.Employee\_ID,

E.First\_Name,

E.Last\_Name,

E.Salary,

L.Location\_Name,

L.Population

FROM

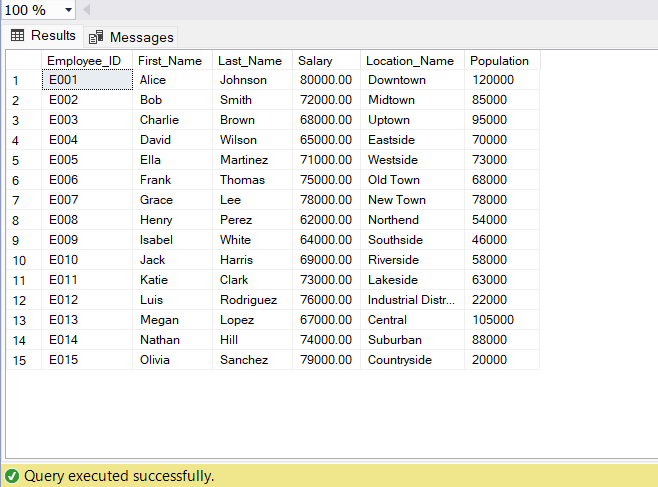
Employee E

INNER JOIN

Location L ON E.Location\_ID = L.Location\_ID;

SELECT \* FROM EmployeeLocationView;

**Output →**



* **Trigger →**

CREATE TRIGGER UpdateServiceSubscription

ON ServiceSubscription

AFTER UPDATE

AS

BEGIN

-- Trigger logic to perform actions after updating ServiceSubscription table

-- Example: Logging the update in an audit table

INSERT INTO ServiceSubscriptionAudit (Subscription\_ID, Updated\_At)

SELECT Subscription\_ID, GETDATE()

FROM INSERTED;

END;

* **Check Constraints →**

ALTER TABLE Customer

ADD CONSTRAINT CheckAgeConstraint CHECK (DATEDIFF(YEAR, Date\_of\_Birth, GETDATE()) >= 18);

ALTER TABLE ServiceSubscription

ADD CONSTRAINT CheckEndDateConstraint CHECK (End\_Date > Start\_Date);

ALTER TABLE Location

ADD CONSTRAINT CheckPopulationConstraint CHECK (Population >= 0);

* **User- Defined function (UDF) →**

CREATE FUNCTION dbo.CalculateDiscount

(

@InvoiceAmount DECIMAL(10, 2)

)

RETURNS DECIMAL(5, 2)

AS

BEGIN

DECLARE @Discount DECIMAL(5, 2);

-- Example discount calculation logic: 10% discount for amounts greater than 100

IF @InvoiceAmount > 100

SET @Discount = 0.10 \* @Invoice\_Amount;

ELSE

SET @Discount = 0;

RETURN @Discount;

END;

ALTER TABLE Billing

ADD Invoice\_Amount FLOAT;

UPDATE Billing

SET Invoice\_Amount = RAND() \* 1000;

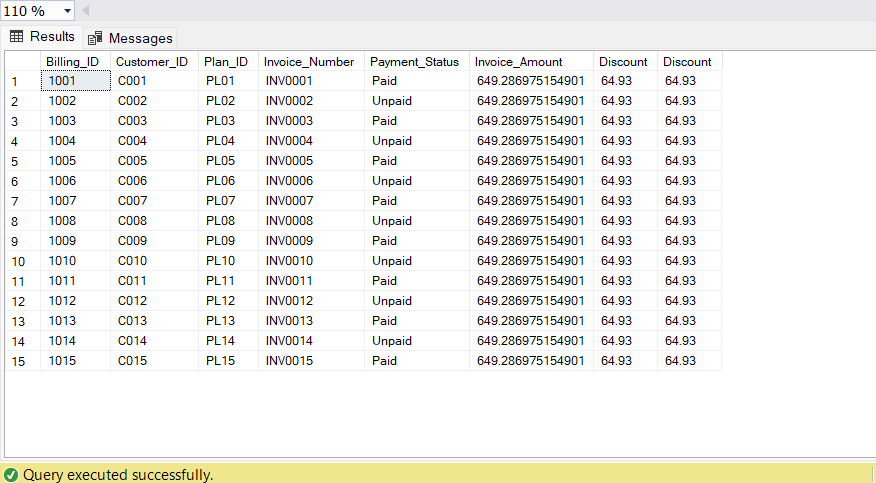
ALTER TABLE Billing

ADD Discount AS dbo.CalculateDiscount(Invoice\_Amount);

SELECT \* , Discount

FROM Billing;

**Output →**



* **Column Data Encryption →**

CREATE SYMMETRIC KEY Customer\_IDKey

WITH ALGORITHM = AES\_256

ENCRYPTION BY PASSWORD = 'DMDD123';

ALTER TABLE Customer

ADD EncryptedCustomer\_ID VARBINARY(MAX);

OPEN SYMMETRIC KEY Customer\_IDKey

DECRYPTION BY PASSWORD = 'DMDD123';

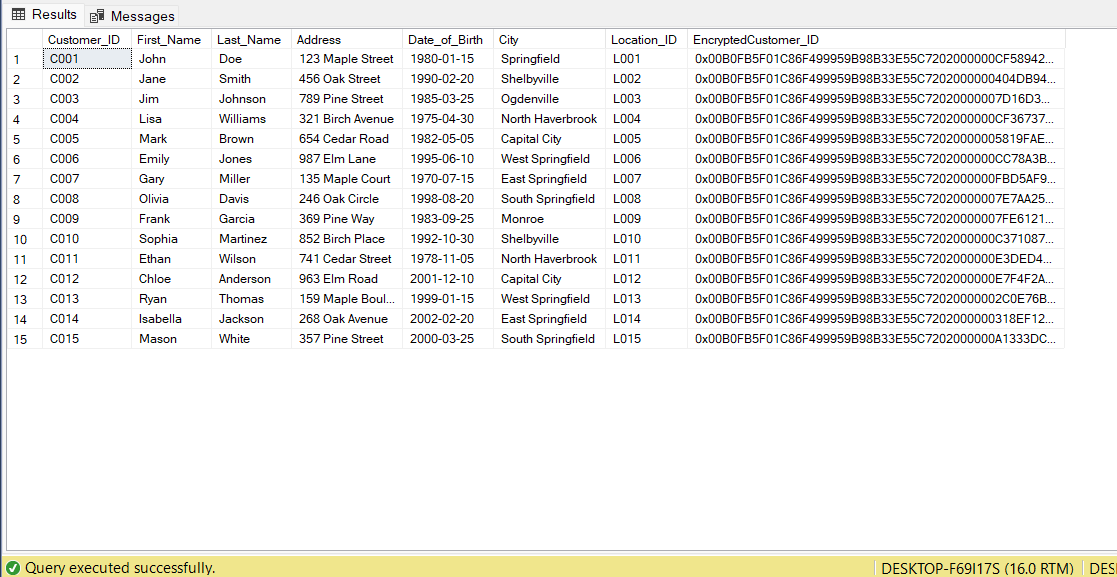
UPDATE Customer

SET EncryptedCustomer\_ID = ENCRYPTBYKEY(KEY\_GUID('Customer\_IDKey'), CONVERT(VARBINARY(MAX), Customer\_ID));

CLOSE SYMMETRIC KEY Customer\_IDKey;

SELECT \* FROM Customer;

**Output →**

****

* **Non-Clustered Indexes →**

CREATE NONCLUSTERED INDEX IX\_Customer\_LastName

ON Customer(Last\_Name);

CREATE NONCLUSTERED INDEX IX\_Employee\_DepartmentID

ON Employee(Department\_ID);

CREATE NONCLUSTERED INDEX IX\_Billing\_PlanID

ON Billing(Plan\_ID);