CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT FOREIGN KEY REFERENCES Departments(DepartmentID),

Salary DECIMAL(10, 2),

JoinDate DATE

);

-- Insert Departments

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT');

-- Insert Employees

INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID, Salary, JoinDate) VALUES

(101, 'Alice', 'Johnson', 1, 50000, '2022-01-10'),

(102, 'Bob', 'Smith', 2, 60000, '2021-06-15'),

(103, 'Charlie', 'Brown', 3, 55000, '2023-03-20');

------------------------------Exercise-1-----------------------------------

CREATE VIEW vw\_EmployeeBasicInfo AS

SELECT

E.EmployeeID,

E.FirstName,

E.LastName,

D.DepartmentName

FROM

Employees E

JOIN

Departments D ON E.DepartmentID = D.DepartmentID;

----------------------------Exercise-2---------------------------------------

CREATE VIEW vw\_EmployeeFullName AS

SELECT

EmployeeID,

FirstName,

LastName,

FirstName + ' ' + LastName AS FullName

FROM

Employees

-------------------------Exercise-3----------------------------------------------

CREATE VIEW vw\_EmployeeAnnualSalary AS

SELECT

EmployeeID,

FirstName,

LastName,

Salary,

Salary \* 12 AS AnnualSalary

FROM

Employees;

----------------------------Exercise-4------------------------------------------------

CREATE VIEW vw\_EmployeeReport AS

SELECT

E.EmployeeID,

E.FirstName + ' ' + E.LastName AS FullName,

D.DepartmentName,

E.Salary \* 12 AS AnnualSalary,

(E.Salary \* 12) \* 0.10 AS Bonus

FROM

Employees E

JOIN

Departments D ON E.DepartmentID = D.DepartmentID;

------------------Exercise 1: Create a Stored Procedure to Insert Employee--------------

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

BEGIN

INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary, JoinDate)

VALUES (@FirstName, @LastName, @DepartmentID, @Salary, @JoinDate);

END;

------------- Exercise 2: Modify Stored Procedure (Add Salary in SELECT)------------------

CREATE PROCEDURE sp\_GetEmployeesByDept

@DepartmentID INT

AS

BEGIN

SELECT FirstName, LastName

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

------to modify:

ALTER PROCEDURE sp\_GetEmployeesByDept

@DepartmentID INT

AS

BEGIN

SELECT FirstName, LastName, Salary

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

---------Exercise 3: Delete a Stored Procedure-------------------

DROP PROCEDURE sp\_InsertEmployee;

--------- Exercise 4: Execute a Stored Procedure---------------

EXEC sp\_GetEmployeesByDept @DepartmentID = 2;

-----------Exercise 5: Return Data – Total Employees in Department-----

CREATE PROCEDURE sp\_CountEmployeesByDept

@DepartmentID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

-------------Exercise 6: Output Parameter – Total Salary by Department------

CREATE PROCEDURE sp\_TotalSalaryByDept

@DepartmentID INT,

@TotalSalary DECIMAL(10,2) OUTPUT

AS

BEGIN

SELECT @TotalSalary = SUM(Salary)

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

--To execute:

DECLARE @TotalSalary DECIMAL(10,2);

EXEC sp\_TotalSalaryByDept 1, @TotalSalary OUTPUT;

SELECT @TotalSalary AS TotalSalary;

----Exercise 7: Update Salary

CREATE PROCEDURE sp\_UpdateEmployeeSalary

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

END;

EXEC sp\_UpdateEmployeeSalary 1, 5500.00;

------Exercise 8: Bonus Based on Department----

CREATE PROCEDURE sp\_GiveBonus

@DepartmentID INT,

@BonusAmount DECIMAL(10,2)

AS

BEGIN

UPDATE Employees

SET Salary = Salary + @BonusAmount

WHERE DepartmentID = @DepartmentID;

END;

EXEC sp\_GiveBonus 1, 500.00;

----Exercise 9: Use Transactions------------

CREATE PROCEDURE sp\_UpdateSalaryWithTransaction

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

BEGIN TRANSACTION;

BEGIN TRY

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

THROW;

END CATCH

END;

--------Exercise 10: Dynamic SQL (Flexible Filter)------------

CREATE PROCEDURE sp\_GetEmployeesDynamic

@FilterColumn NVARCHAR(100),

@FilterValue NVARCHAR(100)

AS

BEGIN

DECLARE @SQL NVARCHAR(MAX);

SET @SQL = 'SELECT \* FROM Employees WHERE ' + QUOTENAME(@FilterColumn) + ' = @Value';

EXEC sp\_executesql @SQL, N'@Value NVARCHAR(100)', @Value = @FilterValue;

END;

---------- Exercise 11: Error Handling--------

CREATE PROCEDURE sp\_SafeUpdateSalary

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

BEGIN TRY

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

END TRY

BEGIN CATCH

PRINT 'Error: ' + ERROR\_MESSAGE();

END CATCH

END;

------- Exercise 1: Scalar Function – Annual Salary---------

CREATE FUNCTION fn\_CalculateAnnualSalary (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 12;

END;

-----------Exercise 2: Table-Valued Function – Employees by Department------

CREATE FUNCTION fn\_GetEmployeesByDepartment (@DeptID INT)

RETURNS TABLE

AS

RETURN

(

SELECT \* FROM Employees WHERE DepartmentID = @DeptID

);

----------- Exercise 3: User-Defined Function – Bonus (10%)-----------

CREATE FUNCTION fn\_CalculateBonus (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.10;

END;

---------Exercise 4: Modify Bonus Function to 15%--------------------

ALTER FUNCTION fn\_CalculateBonus (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.15;

END;

--------- Exercise 5: Delete the Bonus Function-------------

DROP FUNCTION fn\_CalculateBonus;

--------Exercise 6: Execute Annual Salary Function for All Employees------

SELECT

FirstName,

LastName,

Salary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees;

------Exercise 7: Annual Salary for a Specific Employee (ID = 1)-------

SELECT

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees

WHERE EmployeeID = 1;

------Exercise 8: Employees from Finance Department (ID = 3)------

SELECT \* FROM dbo.fn\_GetEmployeesByDepartment(3);

------Exercise 9: Nested Function – Total Compensation (Annual Salary + Bonus)----

CREATE FUNCTION fn\_CalculateBonus (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.10;

END;

CREATE FUNCTION fn\_CalculateTotalCompensation (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN dbo.fn\_CalculateAnnualSalary(@Salary) + dbo.fn\_CalculateBonus(@Salary);

END;

------ Exercise 10: Modify Nested Function to Use Updated Bonus Logic------

ALTER FUNCTION fn\_CalculateTotalCompensation (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN dbo.fn\_CalculateAnnualSalary(@Salary) + dbo.fn\_CalculateBonus(@Salary);

END;

---Exercise 1: AFTER Trigger — Log Salary Changes

-- Step 1: Create a log table

CREATE TABLE EmployeeChanges (

ChangeID INT IDENTITY(1,1) PRIMARY KEY,

EmployeeID INT,

OldSalary DECIMAL(10,2),

NewSalary DECIMAL(10,2),

ChangeDate DATETIME

);

-- Step 2: Create AFTER UPDATE trigger

CREATE TRIGGER trg\_AfterSalaryUpdate

ON Employees

AFTER UPDATE

AS

BEGIN

INSERT INTO EmployeeChanges (EmployeeID, OldSalary, NewSalary, ChangeDate)

SELECT

d.EmployeeID,

d.Salary,

i.Salary,

GETDATE()

FROM

deleted d

INNER JOIN

inserted i ON d.EmployeeID = i.EmployeeID

WHERE

d.Salary <> i.Salary;

END;

---Exercise 2: INSTEAD OF DELETE Trigger — Prevent Deletion

CREATE TRIGGER trg\_PreventDelete

ON Employees

INSTEAD OF DELETE

AS

BEGIN

RAISERROR ('Deletion from Employees table is not allowed.', 16, 1);

END;

------- Exercise 3: LOGON Trigger — Restrict Access During Maintenance

CREATE TRIGGER trg\_BlockLoginDuringMaintenance

ON ALL SERVER

FOR LOGON

AS

BEGIN

IF DATEPART(HOUR, GETDATE()) = 2

BEGIN

ROLLBACK;

PRINT 'Login is blocked due to scheduled maintenance from 2 AM to 3 AM.';

END

END;

-------Exercise 4: Modify a Trigger using SSMS

------ Exercise 5: Delete a Trigger

DROP TRIGGER trg\_PreventDelete;

-- Exercise 6: Trigger to Update Computed Column

-- Step 1: Add new column

ALTER TABLE Employees ADD AnnualSalary DECIMAL(10,2);

-- Step 2: Create trigger to update AnnualSalary

CREATE TRIGGER trg\_UpdateAnnualSalary

ON Employees

AFTER INSERT, UPDATE

AS

BEGIN

UPDATE E

SET AnnualSalary = E.Salary \* 12

FROM Employees E

JOIN inserted I ON E.EmployeeID = I.EmployeeID;

END;

--- Exercise 1: Create a Cursor to Print Employee Details

DECLARE @EmployeeID INT,

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE;

DECLARE employee\_cursor CURSOR FOR

SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary, JoinDate

FROM Employees;

OPEN employee\_cursor;

FETCH NEXT FROM employee\_cursor INTO @EmployeeID, @FirstName, @LastName, @DepartmentID, @Salary, @JoinDate;

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT 'EmployeeID: ' + CAST(@EmployeeID AS VARCHAR) +

', Name: ' + @FirstName + ' ' + @LastName +

', DeptID: ' + CAST(@DepartmentID AS VARCHAR) +

', Salary: ' + CAST(@Salary AS VARCHAR) +

', JoinDate: ' + CAST(@JoinDate AS VARCHAR);

FETCH NEXT FROM employee\_cursor INTO @EmployeeID, @FirstName, @LastName, @DepartmentID, @Salary, @JoinDate;

END

CLOSE employee\_cursor;

DEALLOCATE employee\_cursor;

------- Exercise 2: Types of Cursors

-- Static Cursor (takes snapshot, no changes seen after open)

DECLARE static\_cursor STATIC CURSOR FOR

SELECT \* FROM Employees;

OPEN static\_cursor;

-- FETCH commands go here

CLOSE static\_cursor;

DEALLOCATE static\_cursor;

-- Dynamic Cursor (reflects all changes to data)

DECLARE dynamic\_cursor DYNAMIC CURSOR FOR

SELECT \* FROM Employees;

OPEN dynamic\_cursor;

-- FETCH commands go here

CLOSE dynamic\_cursor;

DEALLOCATE dynamic\_cursor;

-- Forward-Only Cursor (default, read-only, fastest)

DECLARE forward\_cursor CURSOR FORWARD\_ONLY FOR

SELECT \* FROM Employees;

OPEN forward\_cursor;

-- FETCH NEXT only

CLOSE forward\_cursor;

DEALLOCATE forward\_cursor;

-- Keyset-Driven Cursor (keys are fixed, data may change)

DECLARE keyset\_cursor KEYSET CURSOR FOR

SELECT \* FROM Employees;

OPEN keyset\_cursor;

-- FETCH commands go here

CLOSE keyset\_cursor;

DEALLOCATE keyset\_cursor;