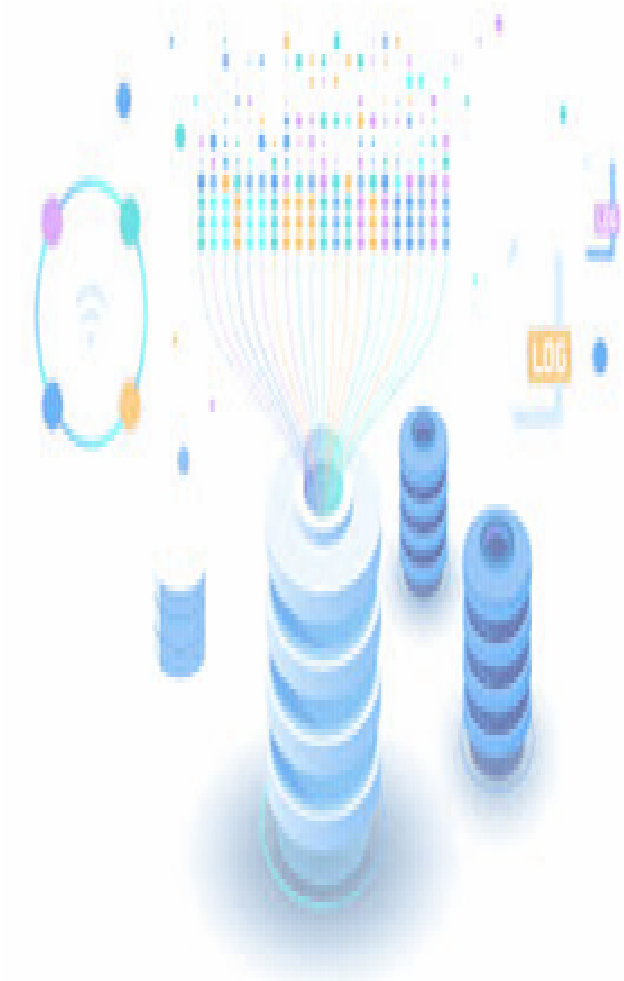


CHAPTER 8

DATABASE PROGRAMMING ON SQL SERVER

CONTENTS

SAMPLE 1 - default value

```
--1. Viết thủ tục hiển thị thông tin của nhân viên có tên được chỉ định (hoặc mặc định là Võ Việt Anh)
CREATE PROC usp_DSNV @fullname
        nvarchar(20)= N'Võ Việt Anh'
AS
BEGIN
--YOUR CODE HERE
END
--TEST
EXEC usp_List01
EXEC usp_List01 N'Trần Thiện Bảo'
```

SAMPLE 1 (default value)

--02. Viết thủ tục hiển thị danh sách nhân viên theo phòng được chỉ định

```
CREATE PROC usp_DSNVTheoPhong
        @empDep INT
AS
BEGIN
--YOUR CODE HERE
END
 --TEST
EXEC usp_DSNVTheoPhong 1
EXEC usp_DSNVTheoPhong @empDep = 1
```

SAMPLE 3 -

--03. Viết thủ tục hiển thị danh sách nhân viên từ phòng i đến phòng j (thủ tục gọi thủ tục khác)

```
CREATE PROC usp_DSNV
    @i int,
    @j int
AS
BEGIN
--YOUR CODE HERE
END
--test
EXEC usp_DSNV
```

FUNCTION (cont.)

--04. Viết hàm trả về bảng DS nhân viên theo phòng được chỉ định

```
CREATE FUNCTION fn_DSNVTheoPhong (@empDep
       nvarchar(30))
RETURNS TABLE
AS
RETURN

  ( SELECT *
    FROM tblEmployee A
    WHERE A.depNum = @empDep);
--TEST
SELECT * FROM fn_DSNVTheoPhong(1)
```

FUNCTION (cont.)

```
--05 Viết hàm trả về danh sách nhân viên có thâm niên làm
việc từ 10 năm trở lên gồm các thông tin: Mã nhân viên, họ
tên, giới tính, năm vào làm, lương, thâm niên
CREATE FUNCTION Fn_DSNV_lamviectu10nam()
RETURNS TABLE
AS
  RETURN
(SELECT A.empSSN, A.empName, A.empSex, YEAR(A.empS
tartdate)AS YearStartDate, A.empSalary, YEAR(GETDA
TE())-YEAR(A.empStartdate) as Seniority
      FROM tblEmployee A
      WHERE (YEAR(GETDATE())-
  YEAR(A.empStartdate)>=10))
--test
SELECT * FROM Fn_DSNV_lamviectu10nam()
```

FUNCTION (cont.)

```
--6- Viết hàm lấy ra danh sách những nhân viên được tăng
lương 15% so với lương cũ
--nếu làm việc từ 15 năm trờ lên
CREATE FUNCTION Fn_DSTangLuong_lamviectu15nam()
RETURNS TABLE
AS
  RETURN (SELECT A.empSSN, A.empName, A.empSalary as
OldSalary, YEAR(A.empStartdate)AS YearStartDate, NewSala
ry = ceiling(A.empSalary*0.15), YEAR(GETDATE())-
YEAR(A.empStartdate) as Seniority
      FROM tblEmployee A
      WHERE (YEAR(GETDATE())-YEAR(A.empStartdate)>=15)
      )
--test
SELECT * FROM Fn_DSTangLuong_lamviectu15nam()
```

TRIGGERS

- Triggers differ from the other constraints.
- Triggers are only awakened when certain events occur (I**NSERT, UPDATE, DELETE**).
- One awakened, the trigger tests a condition.
  - If the condition does not hold, trigger do nothing to response to occurred event
  - If the condition is satisfied, the action associated with trigger is performed by the DBMS

Some principle features of triggers

- The trigger's condition and action can be based on the database state **before or after the triggering event (insert, update, delete).**
- The condition and action can refer to both **old and/or new values of tuples** that were updated in the triggering event.
- Trigger executes either
  - Once for **each modified tuple**
  - Once for **all the tuples** that are changed in one SQL statement

THE OPTIONS FOR TRIGGER DESIGN

AFTER / BEFORE

UPDATE / INSERT/ DELETE

WHEN (<condition>)

OLD ROW / NEW ROW

BEGIN … END;

FOR EACH ROW/FOR EACH STATEMENT

CREATE TRIGGER

```
CREATE TRIGGER trigger_name ON TableName
   {AFTER [DELETE]/[INSERT]/[UPDATE]
 AS
   BEGIN
      sql_statement 1
      sql_statement 2
   END
```

Disable a TRIGGER

```
DISABLE  TRIGGER <trigger_name> ON <table_name>
```

Enable a TRIGGER

```
ENABLE  TRIGGER <trigger_name> ON <table_name>
```

Products (ProductID, ProductName, Price)

PriceHistory (HistoryID, ProductID, OldPrice, NewPrice, ChangeDate)

EXAMPLE 1

```
CREATE TABLE Products
(
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    Price DECIMAL(10, 2)
);

CREATE TABLE PriceHistory
(
    HistoryID INT IDENTITY(1,1) PRIMARY KEY,
    ProductID INT,
    OldPrice DECIMAL(10, 2),
    NewPrice DECIMAL(10, 2),
    ChangeDate DATETIME DEFAULT GETDATE()
);
```

Products (ProductID, ProductName, Price)

PriceHistory (HistoryID, ProductID, OldPrice, NewPrice, ChangeDate)

EXAMPLE 1

```
--insert values
INSERT INTO Products VALUES
      (1, 'Laptop', 1000),
      (2, 'TV', 550);
```

Create the Trigger

EXAMPLE 1

create a trigger to automatically log price change history whenever a record in the Products table is updated.

```
CREATE TRIGGER trigger_AfterPriceUpdate
ON Products
AFTER UPDATE
AS
BEGIN
    INSERT INTO PriceHistory (ProductID, OldPrice, NewPrice)
    SELECT d.ProductID, d.Price, i.Price
    FROM deleted d, inserted i
    WHERE d.ProductID = i.ProductID
            and d.Price <> i.Price; -- Only log if the price
has changed
END;
```

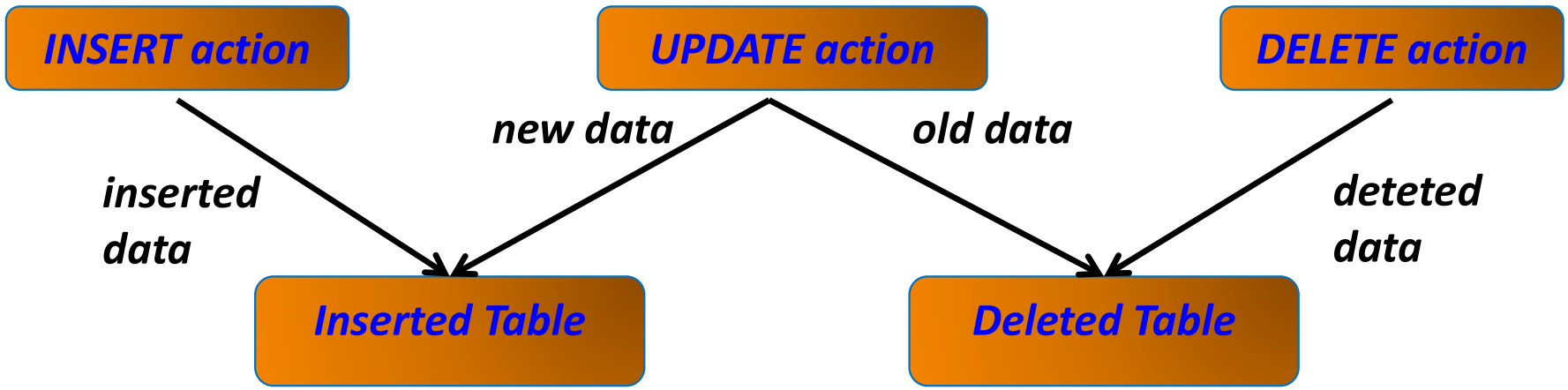
```
--check
SELECT * FROM PriceHistory;
```

IMPLEMENT TRIGGER WITH T-SQL

```
CREATE TRIGGER Tr_Employee_Insert ON tblEmployee
AFTER INSERT
AS
    RAISERROR('Insert trigger is awakened',16,1)
    ROLLBACK TRANSACTION
go
--test
INSERT INTO tblEmployee(empSSN, empName, empSalary,
depNum)
VALUES (30121050345, N'Nguyễnn Văn Tý', 10000, 1);
--not found employee whose empSSN is 30121050345
SELECT * FROM tblEmployee WHERE empSSN=30121050345
```

Deleted and Inserted tables

- When a trigger is executing, it has access to two memory-resident tables that allow access to the data that was modified: **Inserted** and **Deleted**.
- These tables are available only within the body of a trigger for read-only access.
- The structures of the inserted and deleted tables are the same as the structure of the table on which the trigger is defined



Example: using Deleted and Inserted tables

```
IF OBJECT_ID('Tr_Employee_Insert', 'TR') is not null
  drop trigger Tr_Employee_Insert;
--create a trigger
CREATE TRIGGER Tr_Employee_Insert ON tblEmployee
AFTER INSERT
AS
  DECLARE @vEmpSSN DECIMAL, @vEmpName NVARCHAR(50)
  SELECT @vEmpSSN=empSSN FROM inserted
  SELECT @vEmpName=empName FROM inserted
  PRINT 'new tuple:'
  PRINT 'empSSN=' + CAST(@vEmpSSN AS nvarchar(11)) + '
empName=' + @vEmpName;
--test
INSERT INTO tblEmployee(empSSN, empName, empSalary, depNum,
supervisorSSN)
V A L U E S   ( 3 0 1 2 1 0 5 0 3 4 5 ,   N ' N g u y ễ n   V ă n
Tý', 10000, 1, 30121050037);
```

SAMPLES

Create the trigger that refuses all under-18-year-old employee's insertion or update.

```
CREATE TRIGGER Tr_Employee_Under18 ON tblEmployee
AFTER INSERT, UPDATE
AS
  DECLARE @empBirthdate DATETIME, @age INT
  SELECT @empBirthdate=empBirthdate
  FROM inserted;
  SET @age=YEAR(GETDATE()) - YEAR(@empBirthdate)
  IF (@age < 18)
  BEGIN
    RAISERROR('Employee is under 18 years old. We
can not sign a contact with him/her.',16,1)
    ROLLBACK TRANSACTION
  END
```

Another method: using EXISTS

```
CREATE TRIGGER Tr_Employee_Under18 ON tblEmployee
AFTER INSERT, UPDATE
AS
  IF EXISTS(SELECT *
    FROM inserted
    WHERE (YEAR(GETDATE())-
YEAR(empBirthdate))<18)
  BEGIN
    RAISERROR('Employee is under 18. We cannot
sign a contact.',16,1)
    ROLLBACK TRANSACTION
  END
```

Using CURSOR in MS SQL Server

1. Declare cursor

   DECLARE cursor_name CURSOR FOR SELECT Statement

2. Open cursor

   OPEN cursor_name

3. Loop and get values of each tuple in cursor with FETCH statement

   FETCH NEXT | PRIOR | FIRST | LAST
   FROM cursor_name INTO @var1, @var2

4. Using @@FETCH_STATUS to check fetch status. The 0 value mean FETCH statement was successful.

5. CLOSE cursor_name

6. DEALLOCATE cursor_name

EXAMPLE

```
DECLARE @SSN DECIMAL, @FULLNAME NVARCHAR(50), @message NVARCHAR(200)
DECLARE employee_cursor CURSOR
FOR SELECT empSSN, empName FROM tblEmployee
OPEN employee_cursor
FETCH NEXT FROM employee_cursor INTO @SSN,@FULLNAME
IF @@FETCH_STATUS <> 0
    PRINT '         <<None>>'
WHILE @@FETCH_STATUS = 0
BEGIN
    SELECT @message = '         '+@FULLNAME
    PRINT @message
    FETCH NEXT FROM employee_cursor INTO @SSN,@FULLNAME
END
CLOSE employee_cursor
DEALLOCATE employee_cursor
```

EXAMPLE

```
IF OBJECT_ID ( 'psm_Change_Of_Project', 'P' ) IS NOT NULL
    DROP PROCEDURE psm_Change_Of_Project;
GO
CREATE PROCEDURE psm_Change_Of_Project
    @dep1 INT,
    @dep2 INT,
    @loc2 NVARCHAR(50),
    @dep3 INT,
    @loc3 NVARCHAR(50)
AS
    DECLARE @pnum INT, @locname NVARCHAR(50)
    DECLARE pro_cursor CURSOR FOR SELECT p.proNum,l.locName
                                  FROM tblProject p, tblLocation l
                                  WHERE p.locNum=l.locNum AND p.depNum = @dep1;

    OPEN pro_cursor;
    FETCH NEXT FROM pro_cursor INTO @pnum,@locname
    IF @@FETCH_STATUS <> 0
        PRINT '          <<None>>'
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF @locname = @loc2
            UPDATE tblProject SET depNum=@dep2 WHERE proNum=@pnum;
        ELSE IF @locname = @loc3
            UPDATE tblProject SET depNum=@dep3 WHERE proNum=@pnum;

        FETCH NEXT FROM pro_cursor INTO @pnum,@locname
    END
CLOSE pro_cursor
DEALLOCATE pro_cursor
GO
EXEC psm_Change_Of_Project 2,1,N'TP Hà Nội',3,N'TP Hồ Chí Minh';
GO
```

EXAMPLE

```
DECLARE @EmployeeID DECIMAL(18,0);
DECLARE @EmployeeName NVARCHAR(50), @SALARY DECIMAL(10,0);
DECLARE myCursor CURSOR FOR
                 SELECT empSSN, empName, empSalary
                 FROM tblEMPLOYEE
                 WHERE depNum=1;
OPEN myCursor;
FETCH NEXT FROM myCursor INTO @EmployeeID,@EmployeeName,@Salary;
IF @@FETCH_STATUS <> 0
    PRINT '                    <<NONE>>';
WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT cast(@EmployeeID as nvarchar(50))+'                    '+
    @EmployeeName+'              '+cast(@Salary as nvarchar(50));
    FETCH NEXT FROM myCursor INTO @EmployeeID,@EmployeeName,@Salary;
END
CLOSE myCursor;
DEALLOCATE myCursor;
```