STORED PROCEDURE – FUNCTION – TRIGGER

Giảng viên: Đỗ Ngọc Như Loan
Biên soạn: Nguyễn Thị Uyên Nhi

T-SQL NÂNG CAO

1. Transact-SQL nâng cao

- Transact-SQL là ngôn ngữ SQL mở rộng dựa trên SQL chuẩn của ISO (International Organization for Standardization) và ANSI (American National Standards Institute) được sử dụng với P-SQL (Procedural-SQL).
- Ngôn ngữ thủ tục được thiết kế để mở rộng khả năng của **SQL** trong khi có khả năng tích hợp tốt với **SQL**. Một số tính năng như các biến địa phương và xử lý chuỗi/dữ liệu được thêm vào.

1. Transact-SQL nâng cao

- **T-SQL** tổ chức theo từng khối lệnh, một khối lệnh có thể lồng bên trong một khối lệnh khác, một khối lệnh bắt đầu bởi **BEGIN** và kết thúc bởi **END**, bên trong khối lệnh có nhiều lệnh, và các lệnh ngăn cách nhau bởi dấu chấm phẩy.

```
BEGIN
    -- Khai báo biến
   -- Các câu lệnh T-SQL
END;
```

1.1. Khai biến

a) Biến cục bộ:

- ✓ Là một đối tượng có thể chứa giá trị thuộc một kiểu dữ liệu nhất định, tên biến bắt đầu bằng một ký tự @.
- ✓ Biến cục bộ có giá trị trong một *query batch* hoặc trong một thủ tục thường trú (stored procedure) hoặc hàm (function).
- ✓ Khai báo biến cục bộ bằng lệnh declare: cung cấp tên biến và kiểu dữ liệu:

Declare *tên_biến Kiểu_dữ_liệu*

Ví dụ:

Declare @MaSinhVien *char(10)*

Declare @HoTen *nvarchar(30)*

Declare @Sum *float*, @Count *int*

1.1. Khai biến

- ✓ Để gán giá trị cho một biến cục bộ dùng lệnh *set*. Giá trị gán cho biến phải phù hợp với kiểu dữ liệu của biến.

  Set *tênbiến* = *giátrị*
  Set *tênbiến* = *tênbiến*
  Set *tênbiến* = *biểuthức*
  Set *tênbiến* = *kết_quả_truy_vấn*

- ✓ *Ví dụ:*

  Set @MaLop = 'TH2001'
  Set @SoSV = (select count (*) from SinhVien)
  Set @MaLop = 'TH'+Year(@NgayTuyenSinh)

Đưa kết quả truy vấn vào biến:

SV(MaSV: int; HoTen: nvarchar(30), Tuoi int)
Select @Var1 = HoTen, @Var1 = Tuoi
from SV where MaSV = 1

1.1. Khai biến

Tính tổng hai số

```
Begin
  Declare @v_Result Int; -- Khai báo một biến
  Declare @v_a Int = 50; -- Khai báo một biến có giá trị 50
  Declare @v_b Int = 100;  -- Khai báo một biến có giá trị 100
  -- In ra màn hình Console (Dùng cho lập trình viên).
  -- Sử dụng Cast để ép kiểu Int về kiểu chuỗi.
  -- Sử dụng toán tử + để nối 2 chuỗi.
  Print 'v_a= ' + Cast(@v_a as varchar(15));
  -- In ra màn hình Console
  Print 'v_b= ' + Cast(@v_b as varchar(15));
  -- Tính tổng
  Set @v_Result = @v_a + @v_b;
  -- In ra màn hình Console
  Print 'v_Result= ' + Cast(@v_Result as varchar(15));

End;
```

1.1. Khai biến

b) Biến toàn cục:

- ✓ Là các biến hệ thống do SQL Server cung cấp, tên biến bắt đầu bằng 2 ký tự @
- ✓ SQL tự cập nhật giá trị cho các biến này, người sử dụng không thể gán giá trị trực tiếp.
- ✓ Một số biến hệ thống thường dùng
  - ❖ **@@error:** thông báo mã lỗi, nếu @@error = 0: thao tác thực hiện thành công.
  - ❖ **@@rowcount:** cho biết số dòng bị ảnh hưởng bởi lệnh cuối (insert, update, delete).
  - ❖ **@@trancount:** cho biết số giao dịch đang hoạt động trên kết nối hiện tại.
  - ❖ **@@ fetch_status:** cho biết thao tác lấy dữ liệu từ con trỏ có thành công không.

1.2. Cấu trúc điều khiển

a) Lệnh If...else

- ✓ Chức năng: xét điều kiện để quyết định những lệnh T-SQL nào sẽ được thực hiện
- ✓ Cú pháp:

```
IF <điều kiện 1>
    Khối lệnh 1;
[ELSE IF <điều kiện 2>
    Khối lệnh 2;
]
....
[ELSE
    Khối lệnh n + 1;
]
```

Trong đó: Khối lệnh là một hoặc nhiều lệnh nằm trong cặp từ khóa **begin...end**

1.2. Cấu trúc điều khiển

- ✓ Ví du: xét 2 lược đồ quan hệ (LĐQH)

  HOCPHAN(MAHP, TENHP, SISO)

  DANGKY(MASV, MAHP)

Viết lệnh để thêm một đăng ký mới cho sinh viên có mã số 001 vào học phần HP01 (giả sử học phần này đã tồn tại trong bảng HocPhan) và phải bảo đảm SISO HOCPHAN luôn <=50.

1.2. Cấu trúc điều khiển

```
DECLARE @SISO int
SELECT @SISO = SISO FROM HOCPHAN
WHERE MAHP= ‘HP01’
IF @SISO < 50
    BEGIN
    INSERT INTO DANGKY(MASV, MAHP) VALUES(‘001’,
    ’HP01’)
    PRINT N’ĐĂNG KÝ THÀNH CÔNG’
    END
ELSE
    PRINT N’HỌC PHẦN ĐÃ ĐỦ SV’
```

1.2. Cấu trúc điều khiển

b) Vòng lặp While

- ✓ Chức năng: thực hiện lặp lại một đoạn lệnh T-SQL khi điều kiện còn đúng.
- ✓ Cú pháp:

  While *biểu_thức_điều_kiện*
  Lệnh\ Khối lệnh

- ✓ Có thể sử dụng *Break* và *Continue* trong khối lệnh của While:
  - ❖ Break: thoát khỏi vòng while hiện hành.
  - ❖ Continue : trở lại đầu vòng while, bỏ qua các lệnh sau đó.

1.2. Cấu trúc điều khiển

✓ Ví dụ:

Xét lược đồ quan hệ:

SINHVIEN(MASV: INT, HOTEN: NVARCHAR(30))

Viết lệnh xác định một mã sinh viên mới theo qui định: mã sinh viên tăng dần, nếu có chỗ trống thì mã mới xác định sẽ chèn vào chỗ trống đó.

Chẳng hạn, nếu trong bảng sinhvien đã có các mã sinh viên 1, 2, 3, 7 → mã sinh viên mới là 4.

1.2. Cấu trúc điều khiển

```
DECLARE @STT INT
SET @STT = 1
WHILE EXISTS(SELECT *
                    FROM SINHVIEN
                    WHERE MASV = @STT)
SET @STT = @STT+1
INSERT INTO SV(MASV, HOTEN) VALUES(@STT, 'NGUYEN VAN A')
```

1.2. Cấu trúc điều khiển

✓ Ví dụ BREAK - thoát khỏi vòng lặp:

```
    BEGIN
DECLARE @x integer = 0; --Khai báo biến
DECLARE @y integer = 10;
 DECLARE @step integer = 0;
 WHILE (@x < @y)
BEGIN
  SET @step = @step + 1;
  SET @x = @x + 1; -- Mỗi lần vòng lặp chạy giá trị của x tăng lên 1
  SET @y = @y - 2;  -- Mỗi lần vòng lặp chạy giá trị của y giảm đi 2
  PRINT 'Step =' + CAST(@step AS varchar(10));
  PRINT '@x =' + CAST(@x AS varchar(10)) + ' / @y = ' + CAST(@y AS varchar(10));
  -- Nếu @x > 2 thì thoát ra khỏi vòng  lặp, Mặc dù điều kiện trong WHILE vẫn đúng
  IF @x > 2
    BREAK;
  END;
-- Ghi ra log
PRINT 'x,y = ' + CAST(@x AS varchar(10)) + ', ' + CAST(@y AS varchar(10));
END;
```

1.2. Cấu trúc điều khiển

✓ Ví dụ Lệnh **CONTINUE :**

```
BEGIN
DECLARE @x integer = 0;
DECLARE @y integer = 10;
DECLARE @step integer = 0;
WHILE (@x < @y)
BEGIN
  SET @step = @step + 1;
  SET @x = @x + 1;
  SET @y = @y - 2;
    IF @x < 3 -- Nếu @x < 3 thì bỏ qua các dòng lệnh bên dưới, Tiếp tục vòng lặp mới.
    CONTINUE;
  -- Nếu @x < 3 các dòng lệnh bên dưới CONTINUE sẽ không được chạy.
  PRINT 'Step =' + CAST(@step AS varchar(10));
  PRINT '@x =' + CAST(@x AS varchar(10)) + ' / @y = ' + CAST(@y AS varchar(10));
END;
-- Ghi ra log
PRINT 'x,y = ' + CAST(@x AS varchar(10)) + ', ' + CAST(@y AS varchar(10));
 END;
```

c) Lệnh Case

- ✓ Chức năng: kiểm tra một dãy các điều kiện và trả về kết quả phù hợp với điều kiện đúng. Lệnh CASE được sử dụng như một hàm trong câu SELECT.
- ✓ Cú pháp: Có hai dạng:

- Dạng 1 (simple case):

Case *Biểu_thức_đầu_vào*
When *Giá_trị* then *kết_quả*
[...n]
[Else *kết_quả_khác]*
End

- Dạng 2 (searched case):

Case
When *biểu_thức_điều_kiện* then *kết_ quả*
[...n]
[Else *kết_quả_khác]*
End

1.2. Cấu trúc điều khiển

Ví dụ: Xét lược đồ quan hệ:

NHANVIEN(MANV, HOTEN, NGAYSINH, CAPBAC,PHAI)

Cho biết những nhân viên đến tuổi nghỉ hưu biết rằng tuổi về hưu của nam là 60, của nữ là 55.

1.2. Cấu trúc điều khiển

```
Select * from NHANVIEN
        where datediff(yy, NgaySinh, getdate()) >=
        Case Phai
                when ‘Nam’ then 60
                when ‘Nu’ then 55
        End
```

1.3. Gán dữ liệu truy vấn vào biến

```
BEGIN
DECLARE @v_Emp_ID integer = 1;
DECLARE @v_First_Name varchar(30);
DECLARE @v_Last_Name varchar(30);
DECLARE @v_Dept_ID integer;
-- Gán giá trị cho các biến lấy từ câu lệnh Select.
SELECT
 @v_First_Name = emp.First_Name,
 @v_Last_Name = emp.Last_Name,
 @v_Dept_Id = emp.Dept_Id
FROM Employee Emp
WHERE Emp.Emp_ID = @v_Emp_Id;
-- In ra các giá trị:
PRINT '@v_First_Name = ' + @v_First_Name;
PRINT '@v_Last_Name = ' + @v_Last_Name;
PRINT '@v_Dept_Id = ' + CAST(@v_Dept_ID AS varchar(15));
END;
```

1.4. Các tùy chọn lập trình thủ tục

✓ Các tùy chọn lập trình thủ tục trong Transact-SQL

Kiểu	Nhóm câu Lệnh (Batch)	Lưu trữ	Thực thi	Tham số
Mã kịch bản (Script)	Gồm nhiều nhóm câu lệnh	Trong file trên ổ đĩa	Từ công cụ client như Management Studio hoặc SQLCMD	Không
Stored Procedure	Duy nhất	Trong đối tượng của CSDL	Bởi ứng dụng hoặc trong mã kịch bản SQL	Có
Function (User-defined)	Duy nhất	Trong đối tượng của CSDL	Bởi ứng dụng hoặc trong mã kịch bản SQL	Có
Trigger	Duy nhất	Trong đối tượng của CSDL	Tự động bởi server khi một truy vấn hành động cụ thể xảy ra	Không

Bài tập

1. Tính tổng các số nguyên chẵn từ 1 đến N (N là biến, khởi tạo N = 20).

2. Xét lược đồ quan hệ:

NHAN VIEN(MANV, HOTEN, NGAYSINH, CAPBAC,PHAI)

Cho biết mã NV, họ tên và loại nhân viên (cấp bậc <=3:bình thường, cấp bậc = null: chưa xếp loại, còn lại: cấp cao).

STORED PROCEDURE

2.1. KHÁI NIỆM

- ✓ **Stored Procedure** là một tập các câu lệnh **T -SQL** thực hiện một nhiệm vụ cụ thể, được đặt tên và lưu trữ trong CSDL dưới dạng đã biên dịch.
- ✓ **Stored procedure** cung cấp một phương pháp hữu ích cho việc thực thi lặp lại cùng một nhiệm vụ
  - ➢ Giúp tái sử dụng code
  - ➢ Khi thực thi lại một nhiệm vụ, sử dụng lời gọi Stored Procedure thay vì viết và thực thi lại cùng một tập hợp các câu lệnh.
- ✓ Cách sử dụng các biến, cấu trúc điều khiển trong Stored Procedure tương tự như mã kịch bản

2.1. KHÁI NIỆM

Ý NGHĨA:

- ✓ Tính tái sử dụng, tính uyển chuyển nhờ hệ thống tham số.
- ✓ Khi biên dịch SP, các lệnh được tối ưu hóa sao cho thực thi hiệu quả nhất → được lưu bền vững. Khi gọi thực thi thủ tục không cần biên dịch và tối ưu hóa lại → tiết kiệm thời gian và tài nguyên hơn khối lệnh tương đương trong thân thủ tục.
- ✓ Trong ứng dụng triển khai theo môi trường client/server, client gửi lời gọi SP lên server thì chiếm đường truyền ít hơn rất nhiều lần so với việc gửi khối lệnh tương đương trong thân thủ tục -> Giảm khối lượng thông tin trao đổi khi ứng dụng gửi yêu cầu thực hiện công việc về cho server do đó tránh nghẽn đường truyền, giảm trì trệ.

2.1. KHÁI NIỆM

Ý NGHĨA:

- ✓ Đóng gói chỉ các thao tác cho phép trên CSDL vào các SP và quy định truy xuất dữ liệu phải thông qua SP. Ngoài ra còn có thể phân quyền trên SP -> Hỗ trợ tốt hơn cho việc đảm bảo an toàn (security) cho CSDL.
- ✓ SP giúp cho việc kết xuất báo biểu bằng Crystal Report trở nên đơn giản và hiệu quả hơn rất nhiều so với việc kết xuất dữ liệu trực tiếp từ các bảng và khung nhìn.

2.1. KHÁI NIỆM

Một số Quy tắc khi tạo SP :

- ✓ Một thủ tục không thể tạo/xóa 1 đối tượng rồi lại tham chiếu đến đối tượng đó.
- ✓ Các thủ tục có thể tham chiếu đến các bảng tạm thời.
- ✓ Các bảng tạm (Temporary table) có thể được tạo ra bên trong thủ tục và được tự động xoá khi thủ tục kết thúc.
- ✓ Có thể tham chiếu đến các đối tượng từ các CSDL khác và server từ xa.
- ✓ Cho phép các thủ tục đệ quy (recursive) – thủ tục có thể gọi chính nó.
- ✓ Nhiều nhất là 2100 Parameters trong 1 SP.
- ✓ Các thủ tục có thể được gọi lồng nhau tối đa tới 32 mức.
- ✓ Kích cỡ cực đại của 1 thủ tục là 128 MB, và còn tùy thuộc vào bộ nhớ.

2.2. TẠO STORE PROCEDURE

CREATE PROCEDURE procedure_name
*@parameter1 data_type[output] /*các tham số*/,*
@parameter2 data_type[output]
AS
BEGIN
[khai báo các biến cho xử lý]
{Các câu lệnh TRANSACT-SQL}
END
GO

Lưu ý:

- Trong đó các câu lệnh T-SQL có thể nằm trong cặp BEGIN ... END hoặc không.
- Tên tham số đặt theo qui tắc như tên biến cục bộ.
- Giá trị trả về của SP dùng một (hay một số) tham số *output*. Phần [output] là phần có thể có hoặc không để xác định loại tham số.
- Tên thủ tục chứa tối đa 128 kí tự

2.2. TẠO STORE PROCEDURE

Ví dụ 1:

```
CREATE PROCEDURE XinChao
        @hoTen nvarchar(50)
AS
BEGIN
        print N'Xin chào ' + @hoTen
END
GO
```

2.2. TẠO STORE PROCEDURE

Ví du 2: Xây dựng SP cho biết danh sách sinh viên của một lớp có mã cho trước

```
CREATE PROC DS_LOP @MALOP VARCHAR(10)
AS
        SELECT SV.MASV, SV.HOVATEN, SV.NGAYSINH
        FROM SINHVIEN SV
        WHERE SV.LOP = @MALOP
GO
```

2.3. THỰC THI SP

Cú pháp gọi thực hiện thủ tục SP

- Để gọi thực hiện một SP ta sử dụng cú pháp sau:

  EXEC|EXECUTE <Tên SP> [<Danh sách các tham số>]

✓ Khi gọi thực hiện SP, cần truyền đủ tham số với kiểu dữ liệu phù hợp và thứ tự chính xác như khai báo trong định nghĩa SP.

✓ Có thể truyền giá trị cho tham số đầu vào (input) là một hằng hoặc một biến đã gán giá trị, không truyền được một biểu thức.

✓ Để nhận được giá trị kết quả (thông qua tham số đầu ra), cần truyền vào một biến và có từ khóa output.

✓ VÍ DỤ: EXEC XinChao N'Hằng'

2.3. THỰC THI SP

- ✓ Cú pháp của câu lệnh xóa SP: DROP PROC

  DROP {PROC|PROCEDURE} procedure_name
- ✓ Cú pháp của câu lệnh sửa SP: ALTER PROC

```
ALTER PROCEDURE procedure_name
@parameter1 data_type        [output] /*các tham số*/,
@parameter2 data_type        [output]
AS
BEGIN
[khai báo các biến cho xử lý]
{Các câu lệnh transact-sql}
END
GO
```

2.4. THAM SỐ TRONG SP

- Tham số đầu vào: Đây là loại tham số mặc định, cho phép truyền các giá trị vào trong stored procedure để hỗ trợ xử lý.

VÍ DỤ:

```
CREATE PROC SUM
        @So1 int,
        @So2 int
AS
BEGIN
        DECLARE @KQ int
        SET @Kq = @So1 + @So2
        PRINT @KQ
END
GO
EXEC SUM 5,7
```

2.4. THAM SỐ TRONG SP

- Tham số đầu ra: Tham số dùng để nhận kết quả trả về từ Stored Procedure. Sử dụng từ khóa OUTPUT (hoặc viết tắt là OUT) để xác định tham số.

VÍ DỤ:

```
CREATE PROC TRU
        @So1 int,
        @So2 int,
        @KQ int OUTPUT
AS
BEGIN
        SET @KQ = @So1 - @So2
END
GO
DECLARE @test int
EXEC Tru 1, 2, @test output
PRINT @test
```

2.5. TRẢ VỀ GIÁ TRỊ TRONG SP

a) Trả về giá trị từ lệnh RETURN

Lệnh RETURN được sử dụng để trả về giá trị từ Stored Procedure mà không cần sử dụng tham số đầu ra. Giá trị trả về này có một số đặc điểm:

- ✓ Giá trị trả về chỉ có thể là số nguyên. Nếu trả về các loại giá trị khác thì lúc thực thi Stored Procedure sẽ báo lỗi (ngoại trừ 1 số kiểu dữ liệu được tự động chuyển đổi sang kiểu số nguyên như:float, double,...).
- ✓ Giá trị trả về mặc định là 0.
- ✓ Có thể nhận giá trị trả về này bằng 1 biến.
- ✓ Sau khi gọi RETURN, Stored Procedure sẽ trả về giá trị và kết thúc xử lý.

2.5. TRẢ VỀ GIÁ TRỊ TRONG SP

```
CREATE PROC Test
        @Lenh   int
AS
BEGIN
        if (@Lenh = 1) return 1
        if (@Lenh = 2) begin
                declare @float float set @float = 2.6
                return @float
        end
        if (@Lenh = 3) begin
                declare @char varchar(50)
                set @char = 'hello'
                return @char
        end
END
GO
```

2.5. TRẢ VỀ GIÁ TRỊ TRONG SP

```
DECLARE @test float
EXEC @test = Test 3
PRINT @test
```

- Nếu giá trị truyền vào là 1: stored procedure trả về giá trị “1”.
- Nếu giá trị truyền vào là 2: stored procedure trả về giá trị “2”.
- Nếu giá trị truyền vào là 3: stored procedure báo lỗi không thể chuyển chuỗi ‘hello’ thành số nguyên.
- Nếu truyền các giá trị khác: stored procedure trả về giá trị “0”.

2.5. TRẢ VỀ GIÁ TRỊ TRONG SP

b) Trả về dữ liệu từ lệnh SELECT

Mỗi lệnh SELECT đặt trong Stored Procedure sẽ trả về 1 bảng.

Ví dụ:

```
CREATE PROC TestSelect
AS
BEGIN
        SELECT *
        FROM SINHVIEN
        SELECT * FROM LOP
END
GO

EXEC TestSelect
```

2.5. TRẢ VỀ GIÁ TRỊ TRONG SP

THÊM SINH VIÊN VÀO CSDL

```
CREATE PROC ThemSinhVien
    @mssv    varchar(10),
    @hoTen  nvarchar(100),
    @namSinh  int,
    @danToc  nvarchar(20),
    @maLop  varchar(10)
AS
BEGIN
        IF(EXISTS(SELECT * FROM SinhVien s WHERE s.ma =
@mssv))
                BEGIN
                PRINT N'Mã số sinh viên ' + @mssv + N' đã tồn tại'
                RETURN -1
        END
```

2.5. TRẢ VỀ GIÁ TRỊ TRONG SP

```
        IF(NOT EXISTS(SELECT * FROM Lop L WHERE L.ma = @maLop))
                BEGIN
                PRINT N'Mã số lớp ' + @maLop + N' chưa tồn tại'
                END
        INSERT INTO SinhVien(ma, hoTen, namSinh, danToc, maLop)
        VALUES(@mssv, @hoTen, @namSinh, @danToc, @maLop)
        RETURN 0 /* procedure tự trả về 0 nếu không RETURN */
END
GO

DECLARE @KQ INT
EXEC @KQ = ThemSinhVien '0212005', N'Nguyễn Văn A', 1987, 'Kinh',
'TH2002/01'
PRINT @KQ
```

2.5. TRẢ VỀ GIÁ TRỊ TRONG SP

TRẢ VỀ DANH SÁCH SINH VIÊN TRONG LỚP

```
CREATE PROC XuatDanhSachSinhVien
        @maLop  varchar(10)
AS
BEGIN
        IF(NOT EXISTS(SELECT * FROM Lop L WHERE L.ma =
@maLop))
        BEGIN
                PRINT N'Mã số lớp ' + @maLop + N' chưa tồn tại'
                RETURN -1
        END
        SELECT * FROM SINHVIEN sv where sv.maLop = @maLop
        /*procedure luôn trả về 0 nếu không RETURN*/
END
GO
```

BÀI TẬP

Bài tập:

1. Viết stored-procedure tìm số lớn nhất trong 3 số a, b, c và in kết quả.

2. Xét 2 lược đồ quan hệ:

HOCPHAN(MAHP, TENHP, SISO)

DANGKY(MASV, MAHP)

Viết thủ tục thêm một đăng ký của sinh viên vào một học phần, cần phải bảo đảm SISO HOCPHAN luôn <=50 (nếu đã đủ SV cho HV này thì hiện thông báo và không thêm)

BÀI TẬP

Bài tập:

3. Viết stored procedure truyền vào số nguyên N in ra số lượng số chẵn từ 0 đến N và tổng các số chẵn này.

4. Xây dựng SP tính điểm trung bình và xếp loại cho sinh viên thuộc lớp cho trước. Giả sử có các quan hệ như sau:

SINHVIEN (MASV, HOTEN, DTB, XEPLOAI, LOP)

MONHOC (MAMH, TENMH)

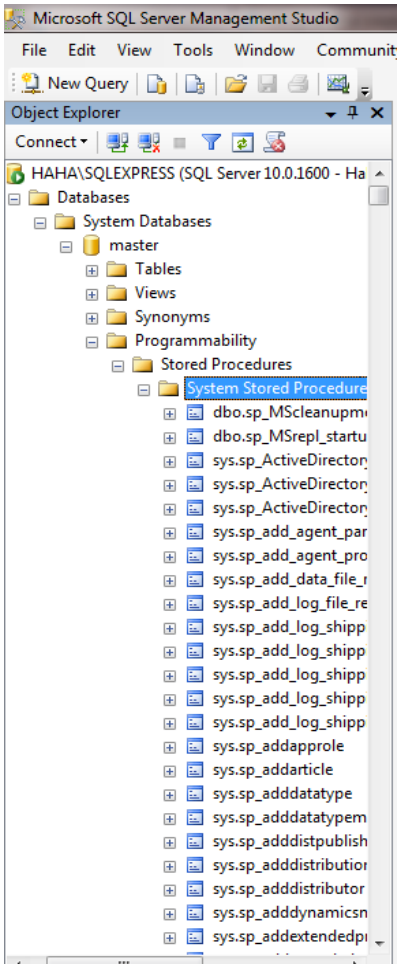
KETQUA (MAMH, MASV, LANTHI, DIEM)

trong đó:

- Điểm thi chỉ tính lần thi sau cùng.
- Xếp loại: Xuất sắc [9, 10], Giỏi [8, 8.9], Khá [7, 7.9], Trung bình [5.0, 6.9], Yếu [0,4.9].
- Kết quả xuất dạng tham số output, không ghi xuống CSDL.

2.3. System Stored Procedure

- System Stored Procedure
  - Là những Stored Procedure được cung cấp sẵn khi cài đặt SQL Server
  - Rất hữu ích khi thực hiện các hành động quản trị và xem thông tin các đối tượng trong SQL Server



2.3. System Stored Procedure

Tham khảo System Stored Procedure tại:

http://msdn.microsoft.com/en-us/library/ms187961.aspx

Trên trang này các System Stored Procedure được phân theo nhóm

Hai nhóm quan trọng chứa các System Stored Procedure hay sử dụng

Database Engine Stored Procedures: chứa nhiều System SP để xem thông tin các đối tượng trong SQL Server

Security Stored Procedures: chứa các System SP dùng cho mục đích quản trị bảo mật hệ thống

Giới thiệu một số System Stored Procedure hữu ích

Thủ tục	Mô tả
sp_Help [<tên đối tượng CSDL>]	Trả về thông tin đối tượng CSDL. (bảng, view, stored procedure…) Trả về tổng hợp tất cả đối tượng trong cơ sở dữ liệu hiện thời, nếu không có tham số.
sp_HelpText <tên>	Trả về văn bản của stored procedure, hàm người dùng định nghĩa, trigger, hay view không mã hóa.
sp_HelpDb [<tên CSDL>]	Trả về thông tin CSDL, hoặc toàn bộ cơ sở dữ liệu, nếu không chỉ định tham số.
sp_Helpfile	Xem tên database vật lý và thuộc tính của các tập tin liên quan đến database hiện tại. Dùng stored procedure này để xác định tên vật lý của database cần detach hoặc attach.

Giới thiệu một số System Stored Procedure hữu ích

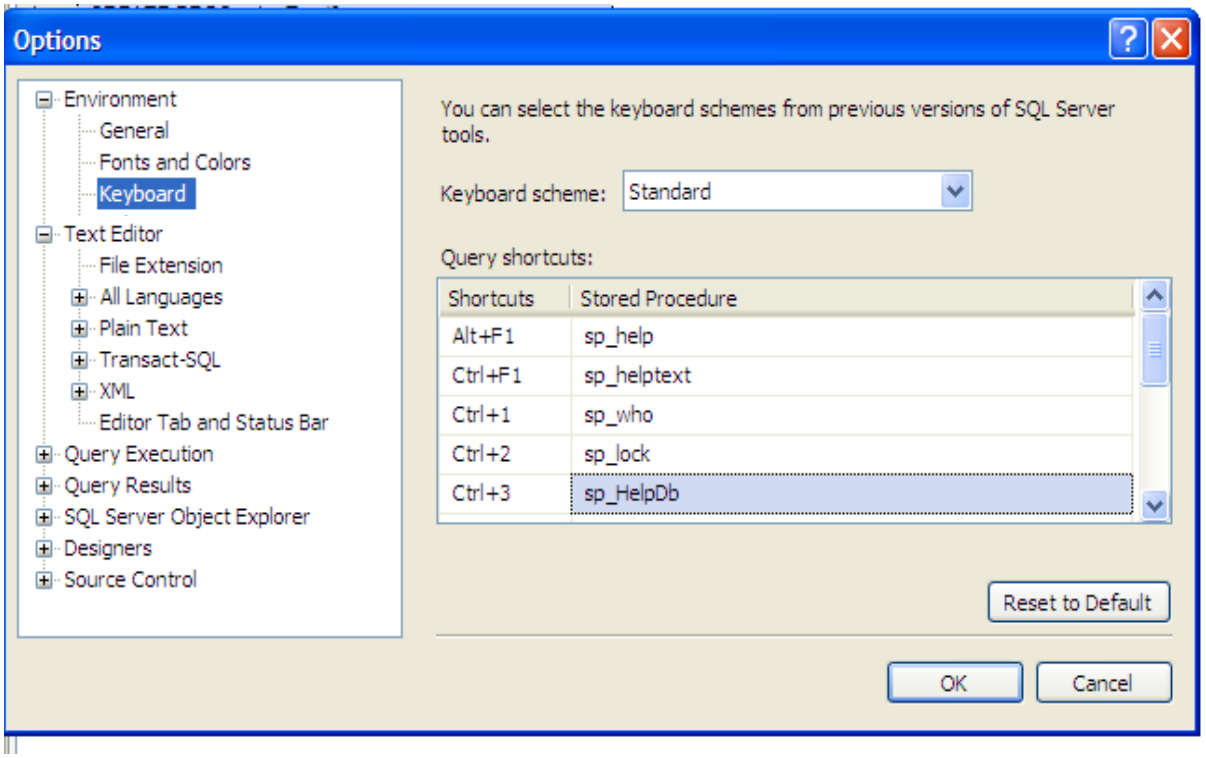
Thủ tục	Mô tả
sp_Who [<ID đăng nhập>]	Trả về thông tin người đang đăng nhập và các tiến trình đang chạy. Trả về thông tin của toàn bộ người dùng đang hoạt động, nếu không chỉ định tham số.
sp_Columns <tên>	Trả về thông tin cột được định nghĩa trong bảng hoặc view xác định.
sp_tables	Xem danh sách các đối tượng có thể truy vấn trong database hiện tại. Tất cả các đối tượng trong mệnh đề FROM.
sp_columns <tên>	Xem thông tin các cột trong 1 table hoặc view
sp_depends <tên>	Xem danh sách các Stored Procedure, View phụ thuộc vào (tham chiếu đến) bảng hoặc view trong CSDL

Giới thiệu một số System Stored Procedure hữu ích

Thủ tục	Mô tả
sp_helpsrvrole '<Tên Server Role>'	-Trả về danh sách tất cả Server Role nếu không được truyền tham số - Trả về thông tin mô tả cho Server Role được chỉ định trong câu lệnh
sp_password '<Password cũ>, '<password mới>', '<Login ID>'	Thay đổi password cho một Login ID
sp_helplogins	Cung cấp thông tin về LoginID cùng với thông tin các Database User liên kết với Login ID này
sp_addlogin	Thêm một login ID mới
Sp_adduser	Thêm một Database User cho một Login ID

Thiết lập phím tắt cho các Stored Procedure hệ thống

- Thiết lập phím tắt cho các Stored Procedure hệ thống
  - Sử dụng menu Tools > Option



FUNCTION

Khái niệm Function

- Hàm là một đối tượng trong cơ sở dữ liệu bao gồm một tập nhiều câu lệnh T-SQL được nhóm lại với nhau thành một nhóm.
- Một hàm - function - trong SQL Server được định nghĩa là một thủ tục đơn giản bao gồm một nhóm các câu lệnhSQL
- Giống như **procedure** (thủ tục), **function** (hàm) là nhóm các lệnh **T-SQL** thực hiện chức năng nào đó. Khác với thủ tục, các hàm sẽ trả về một giá trị thông qua tên hàm ngay tại lời gọi nó.
- **Hàm trong SQL Server** bao gồm 2 loại.
  - hàm có sẵn
  - hàm do người dùng định nghĩa (user-defined): Hàm vô hướng (scalar function) và hàm trả về bảng (table values Function)

Scalar Function

- **Scalar function** là hàm trả về một giá trị với kiểu dữ liệu được khai báo trong RETURNS lúc tạo hàm, ngoại trừ text, ntext, image, cursor, or timestamp.
- **Cú pháp của hàm**

CREATE FUNCTION tên_hàm ([@parameter_name parameter_data_type [= default] [,...n]])

RETURNS kiểu_dữ_liệu **AS**

BEGIN

thân hàm

RETURN value

END

Scalar Function

- *Tạo scalar function đếm số nhân viên theo phong ban (tên phòng ban là tham số).*

```
CREATE FUNCTION fn_DemSoNhanVien(@TenPhong nvarchar(30))
RETURNS int AS
BEGIN
    DECLARE @ret int;
    SELECT @ret = count(*)
    FROM NHANVIEN
    WHERE TenPhong = @TenPhong;
    IF (@ret IS NULL)
        SET @ret = 0;
    RETURN @ret;
END
```

Gọi thực thi:

```
PRINT dbo.fn_DemSoNhanVien('Phòng IT');
SELECT dbo.fn_DemSoNhanVien('Phòng IT');
```

Table-Valued Functions

- **Table-valued function** trả về một table chứa những giá trị từ câu lệnh SELECT.
- **Cú pháp hàm:**

```
CREATE FUNCTION tên_hàm ([@parameter_name parameter_data_type [ = default ] [ ,...n ] ] )
RETURNS TABLE AS RETURN
        câu_lệnh_select;
```

Table-Valued Functions

- *Tạo table-valued function hiển thị danh sách nhân viên thuộc phòng ban (với tên phòng ban là tham số)*

```
CREATE FUNCTION fn_DanhSachNhanVien (@TenPhong nvarchar(30))
RETURNS TABLE AS RETURN
        ( SELECT *
          FROM NHANVIEN
          WHERE TenPhong = @TenPhong );
```

Table-Valued Functions

- Đối với hàm nội tuyến, phần thân của hàm chỉ cho phép sự xuất hiện duy nhất của câu lệnh RETURN. Trong trường hợp cần phải sử dụng đến nhiều câu lệnh trong phần thân của hàm, ta sử dụng cú pháp :

```
CREATE FUNCTION tên_hàm([danh_sách_tham_số])
RETURNS @biến_bảng TABLE định_nghĩa_bảng AS
BEGIN
        các_câu_lệnh_trong_thân_hàm
RETURN END
```

- Dạng hàm này cũng được sử dụng trong các câu lệnh SQL với vai trò như bảng hay khung nhìn

Table-Valued Functions

Thống kê tổng số sinh viên của mỗi khoa (của 1 khóa nào đó)

```
CREATE FUNCTION Func_Tongsv(@khoa SMALLINT)
RETURNS @bangthongke TABLE ( makhoa NVARCHAR(5), tenkhoa NVARCHAR(50), tongsosv INT ) AS
BEGIN
        IF @khoa=0
        INSERT INTO @bangthongke
                SELECT khoa.makhoa,tenkhoa,COUNT(masv)
                FROM (khoa INNER JOIN lop ON khoa.makhoa=lop.makhoa) INNER JOIN sinhvien on lop.malop=sinhvien.malop
                GROUP BY khoa.makhoa,tenkhoa
        ELSE
        INSERT INTO @bangthongke
                SELECT khoa.makhoa,tenkhoa,COUNT(masv)
                FROM (khoa INNER JOIN lop ON khoa.makhoa=lop.makhoa) INNER JOIN sinhvien ON lop.malop=sinhvien.malop WHERE khoa=@khoa
                GROUP BY khoa.makhoa,tenkhoa RETURN /*Trả kết quả về cho hàm*/
END
```

Table-Valued Functions

Gọi thực thi:

Thống kê tổng số sinh viên khóa 25 của mỗi khoa

SELECT * FROM dbo.func_TongSV(25)

Thay đổi, xoá hàm

- Thay đổi hàm:

ALTER FUNCTION tên_hàm([danh_sách_tham_số])
RETURNS *scalar_data_type* | TABLE(*column_definition* | *table_constraint* [,...*n*])
[AS]
[BEGIN *function_body* END] | RETURN [(] *select_statement* [)]

- Ví dụ

```
ALTER FUNCTION fnc_DSSV(@MaLop CHAR(8))
RETURNS TABLE AS
RETURN
        SELECT MaSV, HoSV + ' '+Tenlot+' '+ TenSV As Hoten
        FROM SINHVIEN
        WHERE MaLop=@MaLop
GO
```

Thay đổi, xoá hàm

- Xoá hàm:

DROP FUNCTION { [schema_name.] function_name }

- Ví dụ

DROP FUNCTION fnc_DSSV

❖ Bài tập:

1/ Viết function tính tích 3 số a, b, c.

2/ Viết function tính tổng các số từ 1 đến n (tham số là n).

TRIGGER

Khái niệm TRIGGER

- Cấu trúc gần giống như một thủ tục nội tại nhưng
  - Không có tham số đầu vào và đầu ra
  - Phải được liên kết với một bảng/ bảng ảo trong CSDL
- Không thể gọi mà được thực hiện tự động. Sử dụng trong việc:
  - Tính toán, cập nhật giá trị tự động
  - Kiểm tra dữ liệu nhập
- Khai báo sử dụng
  - Kết hợp với các hành động INSERT/ UPDATE/ DELETE trên bảng hay bảng ảo
  - Khi tạo ra, tham gia vào transaction khởi tạo bởi câu lệnh cập nhật dữ liệu tương ứng

SỬ DỤNG TRIGGER

Trigger được sử dụng trong các cách sau:

- Có thể thay đổi đồng loạt các table có liên quan với nhau trong CSDL
- Có thể không cho phép hoặc hủy bỏ những thay đổi vi phạm ràng buộc toàn vẹn tham chiếu và các giao dịch sửa đổi dữ liệu.
- Có thể áp đặt các giới hạn phức tạp hơn những giới hạn được định nghĩa bằng ràng buộc CHECK và có thể tham chiếu đến các cột trong các bảng khác
- Có thể tìm sự khác biệt giữa các trạng thái của một table trước và sau khi sửa đổi dữ liệu và lấy ra những tác động dựa trên sự thay đổi đó

CÁC HẠN CHẾ TRÊN TRIGGER

- Không được tạo và tham chiếu bảng tạm
- Không tạo hay thay đổi, xoá cấu trúc các đối tượng sẵn có trong CSDL
  - CREATE/ALTER/DROP
- Không gán, cấp quyền cho người dùng
  - GRAND/REVOKE

CƠ CHẾ HOẠT ĐỘNG TRIGGER

- 3 biến cố kích hoạt 1 trigger
  - INSERT
  - UPDATE
  - DELETE
- Trigger lưu trữ dữ liệu của mẩu tin vừa thêm vào một table mới có tên là INSERTED.
- Trigger lưu trữ dữ liệu của mẩu tin vừa xoá vào một table có tên là DELETED.
- Trigger lưu trữ dữ liệu của mẩu tin vừa cập nhật là sự phối hợp của 2 table DELELTED và INSERTED

LỆNH CREATE TRIGGER

CREATE TRIGGER <trigger_name> ON <table name>

AFTER | FOR {DELETE, INSERT, UPDATE}

AS <Các phát biểu T-sql>

- **Tập con của { DELETE, INSERT, UPDATE} dùng chỉ định những phát biểu cập nhật nào trên Table sẽ kích hoạt Trigger.**

CÁC LOẠI TRIGGER

- Có hai loại trigger
  - Trigger thông thường: AFTER (FOR) Trigger
    - Chạy sau các hành động kiểm tra dữ liệu của các Rule, Constraint
    - Dữ liệu đã bị tạm thời thay đổi trong bảng
  - INSTEAD OF TRIGGER
    - Chạy trước các hành động kiểm tra dữ liệu
    - Dữ liệu chưa hề bị thay đổi
    - Có thể thay thế hành động cập nhật dữ liệu bằng các hành động khác

VÍ DỤ

```
CREATE TRIGGER Them_HH
ON  HANG_HOA
AFTER INSERT
AS
        Select * From Inserted
```

- Thêm dữ liệu

```
INSERT INTO HANG_HOA(MaHH, TenHH)
VALUES('TV01', 'Tivi Sony')
```

VÍ DỤ

```
CREATE TRIGGER SUA_HH
ON  HANG_HOA
AFTER UPDATE
AS BEGIN
Select * From Inserted
Select * From Deleted
END
```

- Cập nhật dữ liệu

```
UPDATE HANG_HOA
SET  Ten_HH = ‘Man Hinh Sony’
WHERE MaHH = ‘TV01’
```

VÍ DỤ

```
CREATE TRIGGER Xoa_HH
ON  HANG_HOA
AFTER DELETE
AS BEGIN
Select * From Deleted
END
```

- Xóa dữ liệu

```
DELETE HANG_HOA
WHERE MaHH = 'TV01'
```

CÁC THAO TÁC TRIGGER PHỔ BIẾN

Kiểm tra ràng buộc toàn vẹn khi:

- Thêm mới mẩu tin
- Xóa mẩu tin
- Sửa mẩu tin

a) TRIGGER – THÊM MẪU TIN MỚI

- Thường dùng để kiểm tra
  - Khóa ngoại,
  - Miền giá trị,
  - Liên bộ trong cùng một bảng
  - Liên thuộc tính trong cùng một bảng
  - Liên thuộc tính của nhiều bảng khác nhau
- 3 loại đầu tiên, chỉ dùng trigger nếu muốn cung cấp các báo lỗi cụ thể bằng tiếng Việt, nếu đã khai báo các ràng buộc này bằng constraint
- Các cấu trúc lệnh thường dùng khi kiểm tra
  - If Else
  - If Exists
  - Raiserror
  - Rollback Tran

a) TRIGGER – THÊM MẪU TIN MỚI

Raiserror : trả thông báo lỗi cho ứng dụng

Raiserror(Tbao_loi, muc_do, trang_thai[, cac_tham_so])

- Tbao_loi: thông báo lỗi do người dùng định nghĩa
- Muc_do: 0-25 thể hiện mức độ nghiêm trọng của lỗi
- Trang_thai: 1-127, xác định vị trí lỗi khi sử dụng cùng 1 tbao_loi tại nhiều điểm khác nhau
- cac_tham_so: hỗ trợ các tbao_loi khi cần tham số

a) TRIGGER – THÊM MẪU TIN MỚI

VÍ DỤ:

HOADON_DH(MaHD, NgayDH, MaKH)

PHIEU_XUAT(MaPX, NgayXuat, MaHD)

CHITIET_DH(MAHD, MaHH, SoLuong, DonGia)

Xây dựng trigger trong bảng PHIEU_XUAT để kiểm tra các ràng buộc toàn vẹn dữ liệu khi người dùng thêm mới thông tin của một phiếu xuất hàng cho một bảng hoá đơn đặt hàng trước đó. Các ràng buộc toàn vẹn dữ liệu bao gồm.

- Khoá ngoại: cần kiểm tra số đặt hàng phải tồn tại trong bảng đơn đặt hàng.
- Miền giá trị: cần kiểm tra ngày giao hàng phải ở sau ngày đặt hàng.

a) TRIGGER – THÊM MẪU TIN MỚI

```
CREATE TRIGGER tg_PhieuXuat_Insert
ON     PHIEU_XUAT
FOR    INSERT  AS
Begin
DECLARE @NgayHD datetime, @ErrMsg varchar(200)
-- Kiểm tra số hoá đơn đã có trong bảng DONDH không?
IF NOT EXISTS(Select *
                From Inserted I, HOADON_DH D
                Where I.MaHD= D.MaHD)
Begin
    Rollback Tran
    Raiserror('Số đơn đặt hàng không tồn tại', 16,1)
    Return
End
```

a) TRIGGER – THÊM MẪU TIN MỚI

```
--Tính ra ngày đặt hàng
Select  @NgayDH=NgayDH
From  HoaDon_DH D, Inserted I
Where D.MaHD = I.MaHD
-- Kiểm tra ngày giao hàng phải sau ngày đặt hàng
IF @NgayDH > (Select ngayxuat From Inserted)
Begin
    Set @ErrMsg = ‘ngày giao hàng phải ở sau ngày:’
    + Convert(char(10), ngayDH, 103 )
    Raierror(@ErrMsg,16,1)
    Rollback tran
End
End
```

b) TRIGGER – XÓA MẪU TIN MỚI

Tương tự, kiểm tra các ràng buộc như trigger INSERT, đặc biệt kiểm tra ràng buộc khóa ngoại

Ví dụ: khi xoá một số hoá đơn đặt hàng trong bảng HOADON_DH cần phải kiểm tra các RBTV dữ liệu sau:

- Kiểm tra xem đơn đặt hàng bị xoá đã được xuất hàng chưa? Nếu đã được xuất rồi thì thông báo không thể xoá đơn đặt hàng được.
- Ngược lại thì xoá dữ liệu liên quan bên bảng chi tiết đơn đặt hàng (CHITIET_DH)

b) TRIGGER – XÓA MẪU TIN MỚI

```
CREATE  TRIGGER  tg_HOADON_Delete
ON  HOADON_DH
FOR DELETE
AS
Begin
DECLARE @SoPX char(5), @ErrMsg char(200
-- Kiểm tra xem đơn hàng đã được xuất chưa
IF EXISTS(Select MaPX From PHIEU_XUAT
Where MaHD IN(Select MaHD From Deleted))
     Begin
          Select @MaPX = MaPX From PHIEU_XUAT
          Where MaHD In(Select MaHD From Deleted)
          Set @ErrMsg = ‘Đơn đặt hàng đã được nhập theo ’+ ‘số xuất
          hàng ’+ @SoPX + char(13) + ‘.Không thể huỷ được’
          RaiseError(@ErrMsg,16,1)
          Rollback tran
     End
```

b) TRIGGER – XÓA MẪU TIN MỚI

```
Else
Begin
-- Xoá tự động chi tiết các đơn đặt hàng liên quan
    Delete FROM CHITIET_DH
    Where MaHD In (Select MaHD From DELETED)
End
End
```

c) TRIGGER – SỬA ĐỔI MẪU TIN

- Tương tự, kiểm tra các ràng buộc như trigger INSERT, ràng buộc khoá ngoại có thể sử dụng UPDATE để thực hiện tự động.
- Hàm Update: kiểm tra dữ liệu của cột bên trong bảng có bị thay đổi trong các trigger sửa đổi dữ liệu
- Cú pháp : UPDATE (tên_cột)
  - Tên_cột: tên cột mà chúng ta muốn kiểm tra xem dữ liệu tại đó có bị sửa đổi trong trigger không.
  - Trả về True khi giá trị dữ liệu của cột đã bị sửa đổi, ngược lại trả về False khi giá trị dữ liệu của cột không bị sửa đổi

c) TRIGGER – SỬA ĐỔI MẪU TIN

Sửa đổi thông tin của một số đặt hàng bên trong bảng HOADON_DH cần phải kiểm tra các ràng buộc toàn vẹn dữ liệu sau:

- Không cho phép sửa đổi dữ liệu tại cột MaHD hoặc MaKH vì khi đó dữ liệu sẽ ảnh hưởng đến nhiều bảng.
- Sửa đổi giá trị cột ngày đặt hàng thì phải đảm bảo luôn luôn trước ngày giao hàng đầu tiên của số đặt hàng đó (nếu đơn đặt hàng đã có giao hàng).

c) TRIGGER – SỬA ĐỔI MẪU TIN

```
CREATE TRIGGER  tg_HOADON_DH_Update
ON HOADON_DH
FOR UPDATE
AS Begin
Declare @MinNgayXH date, @ErrMsg varchar(200)
-- Khi sửa đổi các cột MaHD hoặc MaKH
IF Update(MaHD) OR Update(MaKH)
Begin
        Rollback Tran
        Set @ErrMsg = ‘Không thể thay đổi số đặt hàng hoặc
        mã khách hàng’
        RaisError(@ErrMsg, 16, 1)
        Return
End
```

c) TRIGGER – SỬA ĐỔI MẪU TIN

```
-- Khi sửa đổi ngày đặt hàng
IF Update(NgayHD)
Begin
-- Kiểm tra đơn đặt hàng đã được xuất chưa
   IF EXISTS (Select MaPX
                    From PHIEU_XUAT PX, Deleted d
                    where px.mahd=d.mahd)
   Begin
-- Tính ra ngày nhập hàng đầu tiên
         Select @MinNgayXH = Min(NgayXuat)
         From PHIEU_XUAT PX, DELETED D
         Where PX.MaHD = D.MaHD
```

c) TRIGGER – SỬA ĐỔI MẪU TIN

```
--kiểm tra giá trị ngày đăt hàng sau khi sửa đổi phải luôn trước ngày giao hàng đầu tiên
        IF @MinNgayXH < (Select NgayHD From Inserted)
        Begin
                Rollback tran
                Set @ErrMsg = 'Ngày đặt hàng phải ở trước ngày:' + Convert(char(10), @MinNgayXH, 103)
                RaisError(@ErrMsg, 16, 1)
        End
   End
End
End
```

CHỈ ĐỊNH THỨ TỰ THỰC HIỆN CÁC TRIGGER

- Nếu cùng một thao tác trên một Table mà có nhiều Trigger (khác tên nhau). Khi đó thứ tự thực hiện các trigger được xác định dựa trên thứ tự tạo ra chúng.
- Để thay đổi thứ tự thực hiện mặc định này, sử dụng thủ tục: sp_settriggerorder[@triggername =] 'Tên trigger'
  , [@order =] 'First | Last | None'
  , [@stmttype =] 'Insert | Update | Delete'

Ví dụ: sp_SetTriggerOrder itrg_SoCTHD, 'First', 'Insert'

sp_SetTriggerOrder itrg_GiamTon, 'Last', 'Insert'

- SP này chỉ có thể chỉ định trigger nào được thực hiện đầu tiên và Trigger nào được thực hiện cuối cùng. Các Trigger còn lại sẽ thực hiện theo thứ tự tạo ra chúng.

THỰC HIỆN HAY KHÔNG THỰC HIỆN TRIGGER

ALTER TABLE <têntable>
ENABLE | DISABLE TRIGGER ALL | <têntrigger1>[, <têntrigger2>[,…]

Ví dụ: Không thực hiện tất cả Triggers của table

HOADON_DH

ALTER TABLE HOADON_DH DISABLE TRIGGER

ALL

Ví dụ: Không thực hiện tg_HOADON_Delete và tg_HOADON_DH_Update của table HOADON_DH

ALTER TABLE HOADON_DH
DISABLE TRIGGER tg_HOADON_Delete, tg_HOADON_DH_Update

SỬA, XÓA TRIGGER

1. **Sửa Trigger:**

   ALTER TRIGGER <trigger_name> ON <table name>

   AFTER | FOR {DELETE, INSERT, UPDATE}

   AS <Các phát biểu T-sql>

2. **Xóa Trigger:**

   DROP TRIGGER <tên_trigger> [,…n]

HIỂN THỊ THÔNG TIN VỀ CÁC TRIGGER

Tất cả các đối tượng trong CSDL được liệt kê trong bảng hệ thống sysobjects. Cột type trong sysobjects xác định các trigger với chữ viết tắt là TR.

```
SELECT *
FROM      sysobjects
WHERE type='TR'
```


Results

Messages

	name	id	xtype	uid	info	status
1	r1	875150163	TR	1	0	0
2	tgr_test	1707153127	TR	1	0	0

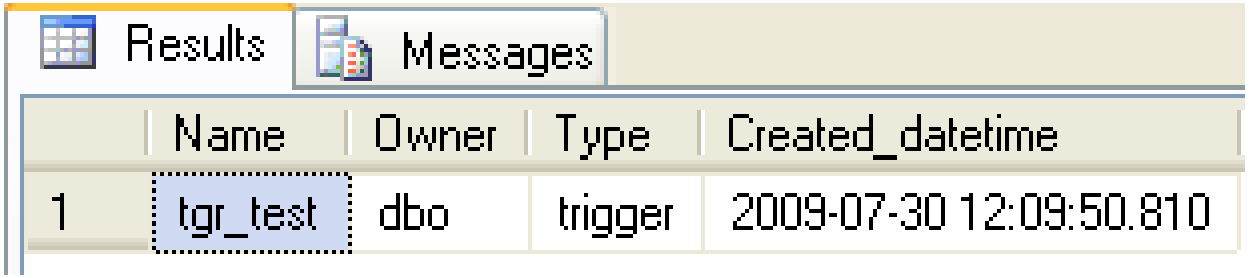
HIỂN THỊ THÔNG TIN VỀ CÁC TRIGGER

- Cú pháp hiển thị thông tin về triggger:

sp_help tên_trigger

- Hiển thị thông tin trigger tgr_test:

Sp_help tgr_test



HIỂN THỊ THÔNG TIN VỀ TRIGGER

Câu lệnh Create trigger của mỗi trigger được lưu trữ trong bảng hệ thống syscomments. Người dùng có thể hiển thị nội dung câu lệnh trigger bằng cách sử dụng thủ tục sp_helptext

Hiển thị nội dung trigger tgr_test:

Sp_helptext tgr_test