

# CHƯƠNG 5

## THỦ TỤC LƯU TRỮ HÀM - TRIGGER

Phạm Thị Thu Thúy

Khoa CNTT, Trường Đại học Nha Trang

[thuthuy@ntu.edu.vn](mailto:thuthuy@ntu.edu.vn)

# ***NỘI DUNG:***

- I. Thủ tục lưu trữ (Stored Procedure).**
- II. Hàm (Function).**
- III. Trigger.**



# THỦ TỤC LƯU TRỮ - LỢI ÍCH

---

- Tăng tốc độ thực hiện:
  - Thực thi tại server, biên dịch một lần
- Tốc độ truy nhập dữ liệu nhanh hơn:
  - SQL không phải lựa chọn cách tốt nhất để xử lý các lệnh SQL và truy xuất CSDL mỗi khi chúng được biên dịch
- Modular programming:
  - Một thủ tục có thể phân thành các thủ tục nhỏ hơn, các thủ tục này có thể được dùng chung giữa các thủ tục khác->giảm thời gian thiết kế và thực thi các thủ tục đồng thời cũng dễ quản lý và gỡ rối.
- Sự nhất quán.
- Cải thiện sự bảo mật:
  - Nâng cao an toàn bảo mật. Có thể chỉ ra quyền thực thi cho các thủ tục vì vậy nó thực hiện đúng tác vụ người dùng.



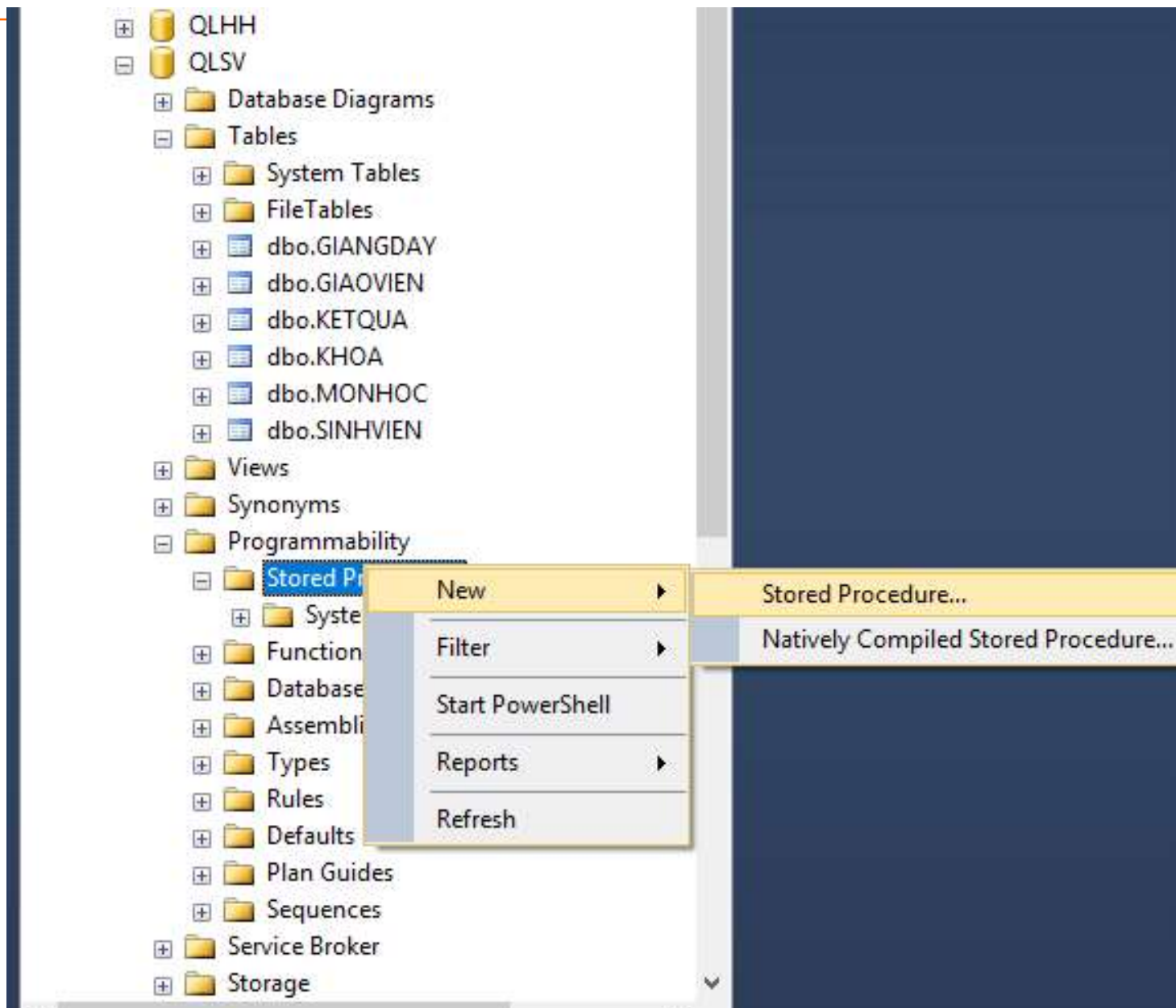
# TẠO THỦ TỤC LƯU TRỮ BẰNG SSMS

## Tạo Stored Procedure

- Mở DB cần tạo stored procedure bằng cách click vào dấu + bên trái tên của DB để hiển thị các mục con trong DB. Ví dụ:
- Diagrams, Tables, Views, Stored procedure,...
- Click phải chuột vào mục Stored procedures
- Chọn chức năng New Stored procedure... trong menu sổ xuống, xuất hiện dialog soạn thảo nội dung của stored procedure



# TẠO THỦ TỤC LƯU TRỮ BẰNG SSMS



# TẠO THỦ TỤC LƯU TRỮ BẰNG SSMS

## ❖ Tạo Stored Procedure

- Đặt tên Stored trong [PROCEDURE NAME]
- Viết nội dung stored
- Sau khi viết xong nội dung xử lý trong stored
  - Nhấn Check Syntax: để kiểm tra cú pháp của các câu lệnh trong stored
  - Nhấn OK: Lưu stored
  - Nhấn Cancel: hủy bỏ thao tác tạo

## ❖ Sửa Stored Procedure

- Click vào mục Stored procedures để hiển thị danh sách Stored procedure tương ứng (bên phải)
- Chọn Stored procedure trong danh sách Stored procedures
- Click phải chuột vào mục Stored procedure cần sửa, vd: MyStoredPro
- Chọn mục Properties trong menu context
- Xuất hiện màn hình tương tự như màn hình Stored procedure
- Thực hiện các thao tác tương tự như phần tạo

## ❖ Xóa Stored Procedure

- Click vào mục Stored procedures để hiển thị danh sách Stored procedure tương ứng (bên phải)
- Chọn Stored procedure trong danh sách Stored procedure
- Click phải chuột vào mục Stored procedure cần xóa, vd: MyStoredPro
- Chọn mục Delete trong menu context

# TẠO THỦ TỤC LƯU TRỮ BẰNG CÂU LỆNH T-SQL

---

- Tạo thủ tục lưu trữ trong CSDL hiện thời bằng Cú pháp:  
**CREATE PROC[EDURE] <tên thủ tục>**  
**[(<DSách tham số> )**  
**[WITH ECOMPILE| ENCRYPTION| RECOMPILE,**  
**ENCRYPTION]**  
**AS**  
**[DECLARE <biến cục bộ>] <Các câu lệnh của thủ tục>**
- Các thủ tục lưu trữ có quyền truy cập tới tất cả các đối tượng khi thủ tục được gọi.
- **2100 tham số** có thể được sử dụng trong một thủ tục lưu trữ. Tham số bắt đầu bởi @, cần chỉ ra kiểu dữ liệu của tham số
- Có thể tạo lập nhiều biến cục bộ trong thủ tục
- **Dung lượng tối đa của thủ tục lưu trữ là 128 MB.**



# THỦ TỤC LƯU TRỮ

❖ Ví dụ:

**Create procedure** Them\_NhanVien

( @MaNV **varchar**(5),  
@HoTenNV **nvarchar**(40),  
@NgaySinh **datetime**,  
@Ngaylamviec **datetime**,  
@Diachi **nvarchar**(225),  
@SoDT **varchar**(12),  
@LuongCB **float**,  
@AnhNV **image** )

**as**

**begin**

**insert into** NhanVien(MaNV,HoTenNV, Ngaysinh, Ngaylamviec,  
Diachi,SoDT, LuongCB, AnhNV)

**values** (@MaNV,@HoTenNV,@NgaySinh,  
@Ngaylamviec,@Diachi,@SoDT,@LuongCB, @AnhNV)

**end**





### 3. Lời gọi thủ tục:

Có 2 cách để gọi 1 thủ tục :

Cách 1: Tên\_thủ\_tục\_lưu\_trữ[danh\_sách\_tham\_số]

**Them\_NhanVien**

( @MaNV **varchar**(5),  
@HoTenNV **nvarchar**(40),  
@NgaySinh **datetime**,  
@Ngaylamviec **datetime**,  
@Diachi **nvarchar**(225),  
@SoDT **varchar**(12),  
@LuongCB **float**,  
@AnhNV **image** )

Cách 2: EXEC Tên\_thủ\_tục\_lưu\_trữ[danh\_sách\_tham\_số]

**EXEC Them\_NhanVien**



The screenshot displays the SQL Server Enterprise Manager interface on the left and a SQL Query window on the right. The Enterprise Manager shows the database structure for 'TNB-SQL.CSDL0...DL09', including tables like 'dbo.Khoa' and 'dbo.SinhVien'. The 'dbo.Khoa' table structure is expanded, showing columns: 'MaKhoa' (PK, varchar(10), not null), 'TenKhoa' (nvarchar(50), null), and 'NamTL' (int, null). The SQL Query window shows the following code:

```
--Tao Stored procedure để thêm dữ liệu vào table
--Khoa
CREATE PROCEDURE DBO.SP_THÊM_KHOA
@MaKhoa varchar(10),
@TenKhoa nvarchar(50),
@NamTL int
AS
INSERT INTO KHOA(MAKHOA, TENKHOA, NAMTL)
VALUES (@MaKhoa, @TenKhoa, @NamTL)
```

Three red arrows point from the table structure in the Enterprise Manager to the corresponding parameters in the stored procedure: from 'MaKhoa' to '@MaKhoa', from 'TenKhoa' to '@TenKhoa', and from 'NamTL' to '@NamTL'.

**Tạo STORE PROCEDURE để thêm mới dữ liệu vào Table KHOA**

Object Explorer

Connect

DBSample01

MaKhoa (PK, varchar(10), not null)  
TenKhoa (nvarchar(50), null)  
NamTL (int, null)

Keys

Thực thi thủ tục  
**SP\_THEM\_KHOA** để  
thêm mới dữ liệu vào  
Table KHOA

Thực thi thủ tục  
**SP\_THEM\_KHOA** để  
thêm mới dữ liệu vào  
Table KHOA

TNB-SQL.CSDL0...DL09-STORE.sql TNB-SQL.CSDL0...SQLQuery2.sql\*

```
--Tao Stored procedure de them du lieu vao table
--Khoa
CREATE PROCEDURE DBO.SP_THEM_KHOA
@MaKhoa varchar(10),
@TenKhoa nvarchar(50),
@NamTL int
AS
INSERT INTO KHOA(MAKHOA, TENKHOA, NAMTL)
VALUES (@MaKhoa, @TenKhoa, @NamTL)
/*-----
Thuc thi (Execute) stored de them du lieu
-----*/
EXEC SP_THEM_KHOA 'DIA', N'DIA LÝ', 1980

--THEM MOI KHOA MOI
EXEC SP_THEM_KHOA 'GDCT', N'GIÁO DỤC CHÍNH TRỊ', 1981
```



# THỦ TỤC LƯU TRỮ

## 4. SỬ DỤNG BIẾN TRONG THỦ TỤC LƯU TRỮ:

Trong trường hợp các câu lệnh trong thân thủ tục lưu trữ quá phức tạp, chúng ta có thể sử dụng biến để lưu trữ các kết quả tính toán tạm thời và giảm bớt sự phức tạp.

Ví dụ:

```
Create procedure Xoa_NhanVien  
( @MaNV varchar(5) )  
as  
begin  
    delete from NhanVien  
    where MaNV =@MaNV  
end
```



# KHAI BÁO BIẾN

- Biến là một đối tượng có thể chứa dữ liệu
  - Dữ liệu có thể đưa vào các câu lệnh SQL dùng cục bộ
  - Tên các biến cục bộ phải bắt đầu bằng @
  - Từ khóa **SET** hay **SELECT** được dùng để gán giá trị cho biến cục bộ.



# Khai báo biến (tt)

---

- **DECLARE @Tên\_biến Kiểu\_dữ\_liệu [, ...]**
  - Kiểu dữ liệu **text**, **ntext** hoặc **image** không được chấp nhận khi khai báo biến
  - **Ví dụ:** Để khai báo các biến lưu trữ giá trị tổng số lượng đặt hàng, họ tên nhà cung cấp, ngày xuất hàng. Sử dụng lệnh DECLARE như sau:

```
DECLARE @Tongsldat INT,  
        @Hotenncc CHAR(50)  
DECLARE @Ngayxh DATETIME
```



# GÁN GIÁ TRỊ CHO BIẾN

---

- Từ khóa **SET** hay **SELECT** được dùng để gán giá trị cho biến.

- **Cú pháp:** SET @<tên biến cục bộ> = <giá trị>

Hoặc là:

SELECT @<Tên biến cục bộ> = <giá trị>

- **Chú ý:** Phạm vi hoạt động của biến chỉ nằm trong một thủ tục hoặc một lô có chứa lệnh khai báo biến đó



# GÁN GIÁ TRỊ CHO BIẾN

---

- Ví dụ:

- Để tính ra số lượng đặt hàng cao nhất của mặt hàng “Đầu DVD Hitachi 1 đĩa” có mã vật tư là “DD01”. Sử dụng lệnh SELECT như sau:

```
DECLARE @MaxSldat INT  
SELECT @MaxSldat=MAX (SLDAT)  
FROM CTDONDH  
WHERE MAVTU='DD01'
```





# XEM GIÁ TRỊ HIỆN HÀNH CỦA BIẾN

---

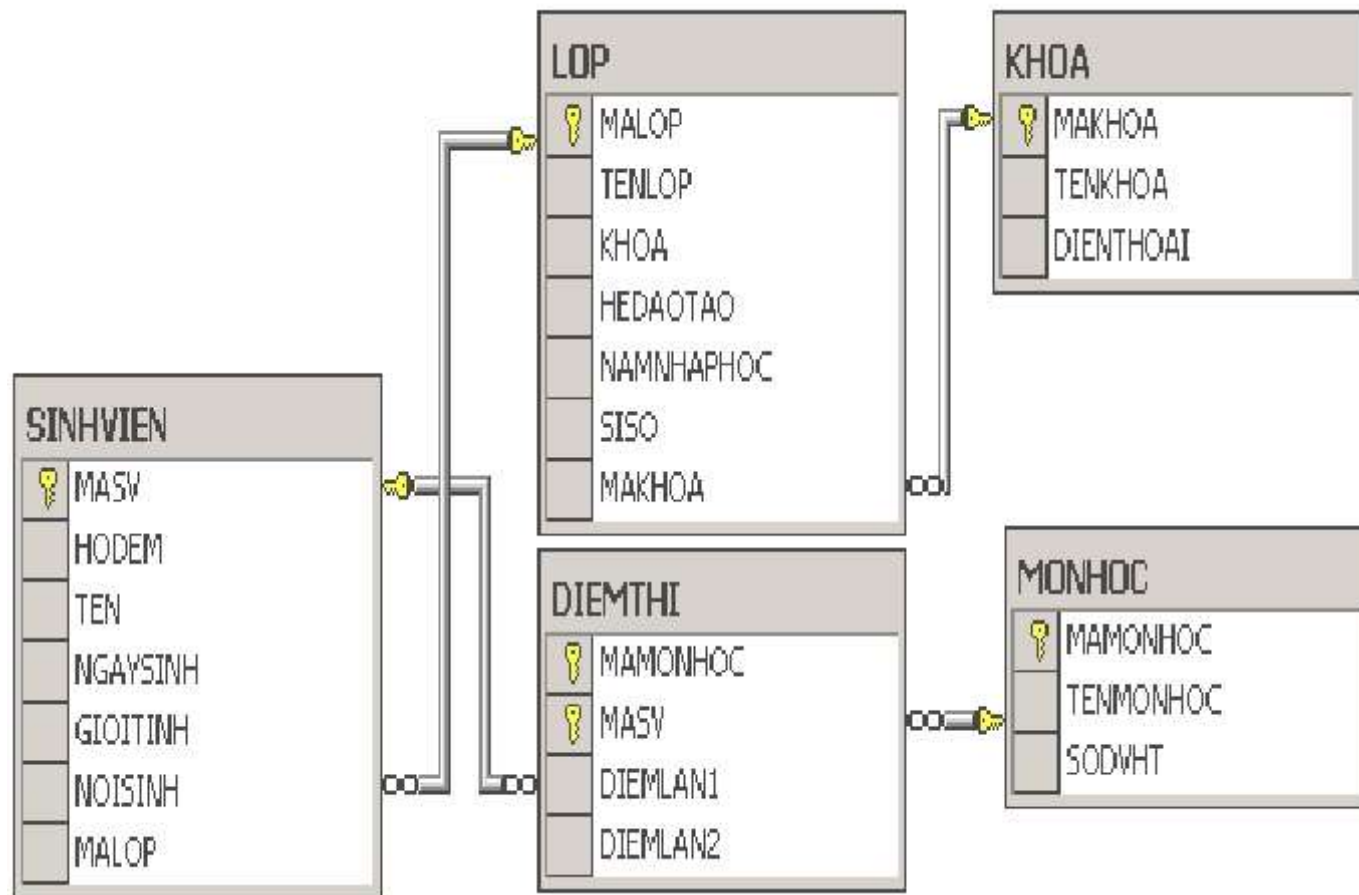
- Để tính đồng thời giá trị số lượng đặt hàng thấp nhất và cao nhất, hiển thị kết quả ra màn hình. Ta sử dụng lệnh **SELECT** và **PRINT**:

```
DECLARE @MinSldat INT, @MaxSldat INT
SELECT @MinSldat=MIN(SLDAT),
@MaXSldat=MAX(SLDAT)
FROM CTDONDH
PRINT "Số lượng thấp nhất là : "
PRINT @MinSldat
PRINT "Số lượng cao nhất là : " +
CONVERT(VARCHAR(10), @MaxSldat)
```



# TẠO THỦ TỤC LƯU TRỮ BẰNG T-SQL (TT)

- Ví dụ: Cho CSDL quản lý điểm thi như sau:



# TẠO THỦ TỤC LƯU TRỮ BẢNG T-SQL (TT)

---

- Giả sử ta cần thực hiện một chuỗi các thao tác trên cơ sở dữ liệu
  1. **Bổ sung thêm môn học** *cơ sở dữ liệu* có mã *CST005* và số đơn vị học trình là 5 vào bảng MONHOC
  2. **Lên danh sách nhập điểm thi môn** *cơ sở dữ liệu* cho các sinh viên học lớp có mã *CDT002* (bổ sung thêm vào bảng DIEMTHI các bản ghi với cột MAMONHOC nhận giá trị *CST005*, cột MASV nhận giá trị lần lượt là mã các sinh viên học lớp có mã *CDT002* và các cột điểm là NULL).



# TẠO THỦ TỤC LƯU TRỮ BẢNG T-SQL (TT)

- Theo cách thông thường ta sẽ viết 2 lệnh như sau:

INSERT INTO monhoc

VALUES('CST005', N'Cơ sở dữ liệu',5)

- INSERT INTO diemthi(mamh,masy)

SELECT 'CST005', masv

FROM sinhvien

WHERE malop = 'CDT002'

Đây là mã  
môn học, đã  
cho trước  
nên ghi cụ  
thể ra luôn

- Lưu ý: Có cú pháp câu lệnh chèn dữ liệu vào một bảng có tên **banga** với dữ liệu lấy từ bảng có tên bangb như sau:

INSERT INTO banga (cot1, cot2)

SELECT cot1, cot2 FROM bangb



## TẠO THỦ TỤC LƯU TRỮ BẰNG T-SQL (TT)

---

- Thay vì phải viết 2 câu lệnh như trên, ta có thể **định nghĩa** một **thủ tục** với các tham số sau **@mamh**, **@tenmh**, **@sodvht**, **@malop** để nhập dữ liệu cho một môn học bất kỳ và một lớp bất kỳ do người dùng nhập vào khi sử dụng thủ tục.



# TẠO THỦ TỤC LƯU TRỮ BẰNG T-SQL (TT)

---

```
CREATE PROC sp_LenDanhSachDiem(  
    @mamh  NVARCHAR(10),  
    @tenmh  NVARCHAR(50),  
    @sodvht SMALLINT,  
    @malop  CHAR(4)  
  
AS  
BEGIN  
    INSERT INTO monhoc  
        VALUES(@mamh,@tenmh,@sodvht)  
    INSERT INTO diemthi(mamh,masv)  
        SELECT @mamh,masv  
        FROM sinhvien  
        WHERE malop=@malop  
  
END
```

- Khi thủ tục trên đã được tạo ra, thực hiện được hai yêu cầu trên qua lời gọi thủ tục:
  - **sp\_LenDanhSachDiem 'CST005','Cơ sở dữ liệu',5,'L002'**



# THỰC THI CÁC THỦ TỤC NGƯỜI DÙNG

- Lời gọi thủ tục có dạng:

**<tên\_thủ\_tục> [<danhsách\_các\_đôi\_số>]**

- Số lượng các đối số và thứ tự của chúng phải phù hợp với số lượng và thứ tự của các tham số hình thức.
- Nếu lời gọi thủ tục được thực hiện bên trong một thủ tục khác, bên trong một trigger hay kết hợp với các câu lệnh SQL khác, ta sử dụng cú pháp như sau:

**[EXECUTE] <tên\_thủ\_tục> [<danhsách\_các\_đôi\_số>]**

- Ví dụ:

EXECUTE sp\_LenDanhSachDiem 'CST005','Cơ sở dữ liệu',5,'L002'

- Thứ tự của các đối số được truyền cho thủ tục có thể không cần phải tuân theo thứ tự của các tham số như khi định nghĩa thủ tục nếu tất cả các đối số được viết dưới dạng:

**@<tên\_tham\_số> = <giá\_trị>**



## SỬ DỤNG BIẾN TRONG THỦ TỤC:

---

```
CREATE PROC sp_Vidu(@malop1 CHAR(4),
                    @malop2 CHAR(4))
AS
DECLARE @tenlop1 NVARCHAR(30)
DECLARE @namnhaphoc1 INT
DECLARE @tenlop2 NVARCHAR(30)
DECLARE @namnhaphoc2 INT

SELECT @tenlop1=tenlop,@namnhaphoc1=namnhaphoc
FROM lop WHERE malop=@malop1
SELECT @tenlop2=tenlop, @namnhaphoc2=namnhaphoc
FROM lop WHERE malop=@malop2
PRINT @tenlop1+' nhap hoc nam '+str(@namnhaphoc1)
print @tenlop2+' nhap hoc nam '+str(@namnhaphoc2)
IF @namnhaphoc1=@namnhaphoc2
    PRINT 'Hai lớp nhập học cùng năm'
ELSE
    PRINT 'Hai lớp nhập học khác năm'
```





# GIÁ TRỊ TRẢ VỀ CỦA THAM SỐ TRONG THỦ TỤC

- Trường hợp cần giữ lại giá trị của đối số sau khi kết thúc thủ tục, khai báo tham số của thủ tục theo cú pháp:

`@tên_tham_số` kiểu\_dữ\_liệu OUTPUT

- Hoặc:

`@tên_tham_số` kiểu\_dữ\_liệu OUT

- Trong lời gọi thủ tục, sau đối số được truyền cho thủ tục, cũng phải chỉ định thêm từ khoá OUTPUT (hoặc OUT)

```
CREATE PROCEDURE sp_Conghaiso(
```

```
    @a    INT,
```

```
    @b    INT,
```

```
    @c    INT OUTPUT)
```

```
AS
```

```
    SELECT @c=@a+@b
```



## GIÁ TRỊ TRẢ VỀ CỦA THAM SỐ TRONG THỦ TỤC (TT)

---

Thực hiện lời gọi thủ tục trong một tập các câu lệnh như sau:

```
DECLARE @tong INT  
SELECT @tong=0  
EXECUTE sp_Conghaiso 100,200,@tong OUTPUT  
SELECT @tong
```

=> câu lệnh “SELECT @tong” sẽ cho kết quả là:  
300



# THAM SỐ GIÁ TRỊ MẶC ĐỊNH

---

- Tham số với giá trị mặc định được khai báo theo cú pháp như sau:

@<tên\_tham\_số> <kdl> = <giá\_trị\_mặc\_định>



# THAM SỐ GIÁ TRỊ MẶC ĐỊNH

```
CREATE PROC sp_TestDefault(  
AS  
    BEGIN  
        @tenlop NVARCHAR(30)=NULL,  
        @noisinh NVARCHAR(100)='Khánh Hòa')  
    IF @tenlop IS NULL  
        SELECT hodem,ten  
        FROM sinhvien INNER JOIN lop  
        ON sinhvien.malop=lop.malop  
        WHERE noisinh=@noisinh  
    ELSE  
        SELECT hodem,ten  
        FROM sinhvien INNER JOIN lop  
        ON sinhvien.malop=lop.malop  
        WHERE noisinh=@noisinh AND  
            tenlop=@tenlop  
END
```



## THAM SỐ GIÁ TRỊ MẶC ĐỊNH (TT)

---

- Cho biết họ tên của các sinh viên sinh tại **Khánh Hòa**:
  - `sp_testdefault`
- Cho biết họ tên của các sinh viên lớp *59CTH* sinh tại *Khánh Hòa*:
  - `sp_testdefault @tenlop='59CTH'`
- Cho biết họ tên của các sinh viên sinh tại *Nghệ An*:
  - `sp_testDefault @noisinh=N'Nghệ An'`
- Cho biết họ tên của các sinh viên lớp 58TH sinh tại Đà Nẵng:
  - `sp_testdefault @tenlop='58TH',@noisinh='Đà Nẵng'`



# THỦ TỤC LƯU TRỮ

## 5. Xây dựng thủ tục đa năng:

Thủ tục lưu trữ đa năng là thủ tục chấp nhận nhiều tham số khác nhau có liên quan đến thủ tục. Dựa trên các tham số được truyền vào, thủ tục lưu trữ đa năng xác định bản ghi nào sẽ được trả về.

Tham số được sử dụng trong mệnh đề WHERE để ràng buộc kết quả trả về. Mỗi tham số trong mệnh đề WHERE có định dạng như sau:

(tên\_trường= @tên\_tham\_số OR @tên\_tham\_số IS NULL)

- ❖ Ví dụ: Trong ví dụ này, chúng ta muốn tìm kiếm các nhân viên theo 1 trong các tiêu chí sau: theo tên hoặc theo thành phố.



# THỦ TỤC LƯU TRỮ

```
CREATE PROC SP_NHANVIEN_SELECT  
( @HONVNVARCHAR(30)=NULL,  
  @DIACHINVARCHAR(100)=NULL )  
AS  
BEGIN  
SELECT * FROM NhanVien  
WHERE (HoTenNV = @HoTenNV or @HoTenNV IS NULL)  
AND (DiaChi LIKE '%' + @DiaChi + '%' or @DiaChi IS NULL)  
END
```



# THỦ TỤC LƯU TRỮ

Với cách viết này chúng ta có thể sử dụng thủ tục theo trong nhiều trường hợp như sau:

**EXEC SP\_NHANVIEN\_SELECT** - truy xuất tất cả nhân viên

**EXEC SP\_NHANVIEN\_SELECT @HoTenNV= N'Vũ Thị Lan'** — truy xuất nhân viên có tên **Vũ Thị Lan**

**EXEC SP\_NHANVIEN\_SELECT @DiaChi= N'NGUYỄN TRÃI'**—truy xuất nhân viên ở đường Nguyễn Trãi


**EXEC SP\_NHANVIEN\_SELECT @HoTenNV= N'Hoàng Anh Tuấn', @DiaChi= N'NGUYỄN TRÃI'** – truy xuất nhân viên có tên Hoàng Anh Tuấn ở đường Nguyễn Trãi





**Thủ tục đa năng cũng có thể được xây dựng như sau:**

```
CREATE PROC SP_NHANVIEN_SELECT_2
( @TENNV NVARCHAR(30) = NULL,
  @DIACHI NVARCHAR(100) = NULL )
AS
BEGIN
DECLARE @CMD NVARCHAR(MAX) DECLARE @WHERE NVARCHAR(MAX)
SET @CMD = 'SELECT * FROM NHANVIEN '
SET @WHERE = ''
IF @HoTenNV IS NOT NULL
SET @WHERE = @WHERE + 'AND DiaChi LIKE N''' +
CONVERT(NVARCHAR(1),'%') + CONVERT(NVARCHAR(100),@DiaChi) +
CONVERT(NVARCHAR(1),'%') + '''
IF LEN(@WHERE) > 0
SET @CMD += 'WHERE ' + RIGHT(@WHERE,LEN(@WHERE)-3)
EXEC SP_EXECUTESQL @CMD,N'@HoTenNV NVARCHAR(40), @DIACHI
NVARCHAR(255)',@HoTenNV = @HoTenNV, @DiaChi = @DiaChi
END
```



# THỦ TỤC LƯU TRỮ

## 6. Sửa và xóa thủ tục:

### 6.1. Sửa thủ tục:

Cú pháp:

```
ALTER PROCEDURE tên_thủ_tục [(danh_sách_tham_số)]  
[WITH RECOMPILE | ENCRYPTION | ] AS  
<Các_câu_lệnh_của_thủ_tục>
```

Ví dụ:

```
Alter proc Xoa_NhanVien  
(@MaNV smallint)  
As  
Begin  
Delete from PhanCong where MaNV=@MaNV  
End
```



## 6.2 . Xóa Thủ tục:

Để xoá một thủ tục đã có, ta sử dụng câu lệnh DROP PROCEDURE với cú pháp như sau:

**DROP PROCEDURE** tên\_thủ\_tục

Ví dụ:

**DROP PROCEDURE** Xoa\_NhanVien

## 6.3 Sử dụng biến mặc định trong procedure

**Create procedure** Them\_NhanVien

(@MaNV **varchar**(5),

@HoTenNV **nvarchar**(40),

@NgaySinh **datetime**,

@Ngaylamviec **datetime**,

@Diachi **nvarchar**(225),

@SoDT **varchar**(12),

@LuongCB **float** = 2000000,

@AnhNV **image**)



**as**

**begin**

**insertinto** NhanVien(MaNV,HoTenNV, Ngaysinh,  
Ngaylamviec, Diachi,SoDT, LuongCB, AnhNV)

**values** (@MaNV,@HoTenNV,@NgaySinh,  
@Ngaylamviec,@Diachi,@SoDT,@LuongCB,  
@AnhNV)

**end**

Tiền lương của nhân viên khi mới vào làm việc mặc định là  
2.000.000đ/tháng



# THỦ TỤC LƯU TRỮ - TỔNG KẾT

---

- Một thủ tục lưu trữ là một nhóm các câu lệnh SQL được biên dịch lại.
- Người phát triển CSDL hoặc người quản trị hệ thống viết thủ tục để chạy các nhiệm vụ quản trị thông thường, hoặc để ứng dụng các luật giao dịch phức tạp. Thủ tục lưu trữ chứa các thao tác hoặc các câu lệnh truy vấn dữ liệu.
- Các thủ tục lưu trữ tăng tốc độ thực thi của truy vấn, hỗ trợ truy cập dữ liệu nhanh, hỗ trợ việc lập trình theo mô đun, duy trì tính nhất quán, và tăng tính bảo mật.





# HÀM (FUNCTION)

# HÀM

- \_ Hàm là đối tượng CSDL tương tự như thủ tục.
- \_ Điểm khác biệt giữa hàm và thủ tục: hàm trả về một giá trị thông qua tên còn thủ tục thì không.
- \_ Các hàm do Hệ quản trị CSDL cung cấp sẵn như: `getdate()`, `datediff()`, `abs()`, `sqrt()`, `sum()`, `max()`, `count()`, `substring()`,....
- \_ Hàm do người dùng định nghĩa: người sử dụng có thể định nghĩa các hàm nhằm để phục vụ mục đích riêng của mình
- \_ Hàm có 2 loại:
  - Hàm với giá trị trả về là “dữ liệu kiểu bảng” (Table-valued Functions).
  - Hàm với giá trị trả về là một giá trị (Scalar-valued Functions)
- Hàm với giá trị trả về là một giá trị: được sử dụng để trả về một giá trị có thể có tham số truyền vào.



# HÀM

- Cú pháp:

**CREATE FUNCTION** <tên\_hàm>([danh\_sách\_tham\_số])

**RETURNS** (kiểu\_trả\_về\_của\_hàm)

**AS**

**BEGIN**

các\_câu\_lệnh\_của\_hàm

**RETURN** (giá trị trả về)

**END**

- Tên hàm: Tên của hàm cần tạo. Tên phải tuân theo qui tắc định danh và không trùng với tên của các hàm hệ thống có sẵn.
- Danh sách các tham số: Các tham số của hàm được khai báo ngay sau tên hàm và được bao bởi cặp dấu ().
- Các câu lệnh của hàm: Tập hợp các câu lệnh sử dụng trong nội dung hàm để thực hiện các yêu cầu của hàm.



# HÀM

❖ Ví dụ:

1. Viết hàm trả về lương cơ bản cao nhất của nhân viên.

+ Tạo hàm:

```
Create function LuongCB_CN ()  
returns float  
as  
begin  
    declare @LuongCN float  
    select @LuongCN = (select Top 1 luongCB  
                        from NhanVien  
                        order by luongCB desc )  
  
    return (@LuongCN)  
end
```

+ Thực hiện hàm: **exec LuongCB\_CN**



# HÀM

+ Sử dụng hàm:

```
select MaNV as 'Mã NV', HoTenNV as 'Họ và tên NV',  
       luongCB as 'Lương CB cao nhất'
```

```
from NhanVien
```

```
where luongCB = dbo.LuongCB_CN()
```

=>Kết quả

Results		Messages	
	Mã NV	Họ và tên NV	Lương CB cao nhất
1	NV01	Hoàng Anh Tuấn	450000000
2	QL01	Nguyễn Kỳ Hân	450000000

# HÀM

```
CREATE FUNCTION thu(@ngay DATETIME)
    RETURNS NVARCHAR(10)
AS
    BEGIN
        DECLARE @st NVARCHAR(10)

        SELECT @st=CASE DATEPART(DW,@ngay)
            WHEN 1 THEN 'Chủ nhật'
            WHEN 2 THEN 'Thứ hai'
            WHEN 3 THEN 'Thứ ba'
            WHEN 4 THEN 'Thứ tư'
            WHEN 5 THEN 'Thứ năm'
            WHEN 6 THEN 'Thứ sáu'
            ELSE 'Thứ bảy' END
        RETURN (@st) /* Trị trả về của hàm */
    END
```



# HÀM

2. Viết hàm trả về số năm làm việc của nhân viên

+ Tạo hàm:

```
Create function ThamNien_LV
```

```
( @MaNV varchar(5) )
```

```
returns tinyint
```

```
as
```

```
begin
```

```
    declare @Thamnien tinyint
```

```
    select @Thamnien=datediff(yy,Ngaylamviec,getdate())
```

```
    from NhanVien
```

```
    where MaNV = @MaNV
```

```
    return ( @ThamNien )
```

```
end
```

+ Thực hiện hàm: **exec Thamnien\_LV**



# HÀM

+ Sử dụng hàm:

```
select HoTenNV, dbo.Thamnien_LV('NV02') as 'Số năm  
làm việc'
```

```
from NhanVien
```

```
where MaNV ='NV02'
```

=> Kết quả

Results			Messages	
	HoTenNV	Số năm làm việc		
1	Vũ Thị Lan	4		

# HÀM VỚI GIÁ TRỊ TRẢ VỀ LÀ DỮ LIỆU KIỂU BẢNG

- Kiểu trả về của hàm được chỉ định bởi mệnh đề RETURNS TABLE.
- Trong phần thân của hàm chỉ có duy nhất một câu lệnh RETURN xác định giá trị trả về của hàm thông qua duy nhất một câu lệnh SELECT (không sử dụng bất kỳ câu lệnh nào khác trong phần thân của hàm).

Cú pháp:

```
CREATE FUNCTION tên_hàm ([danh_sách_tham_số])  
RETURNS TABLE  
AS  
RETURN (câu_lệnh_Select)
```



# HÀM VỚI GIÁ TRỊ TRẢ VỀ LÀ DỮ LIỆU KIỂU BẢNG (TT)

Ví dụ:

a. Viết hàm xem đầy đủ thông tin của một nhân viên

+ Tạo hàm:

```
Create function XemTT_NV  
( @MaNV varchar(5) )  
returns table  
as  
return (      select *  
              from NhanVien  
              where MaNV = @MaNV )
```

+ Thực hiện hàm: **exec XemTT\_NV**



# HÀM VỚI GIÁ TRỊ TRẢ VỀ LÀ DỮ LIỆU KIỂU BẢNG (TT)

+Sử dụng hàm:

```
select *  
from dbo.XemTT_NV ('NV02')
```

Results		Messages							
	MaNV	HoTenNV	Ngaysinh	Ngaylamviec	Diachi	SoDT	LuongCB	Phucap	AnhNV
1	NV02	Vũ Thị Lan	1990-07-05 00:00:00.000	2010-11-02 00:00:00.000	NULL		5000000	NULL	NULL





## HÀM VỚI GIÁ TRỊ TRẢ VỀ LÀ DỮ LIỆU KIỂU BẢNG (TT)

b) Viết hàm trả về một bảng trong đó cho biết tổng số lượng hàng bán được của mỗi mặt hàng. Sử dụng hàm này để thống kê xem tổng số lượng hàng (hiện có và đã bán) của mỗi mặt hàng là bao nhiêu?

+ Tạo hàm:

```
Create function MH_SLB ()
```

```
returns table
```

```
as
```

```
return
```

```
(select MH.Mahang, TenHang, sum(CT_DH.soluong) as SL_Ban
```

```
from MatHang MH join ChiTietDatHang CT_DH
```

```
on MH.MaHang = CT_DH.MaHang
```

```
groupby MH.MaHang, TenHang )
```

+ Thực hiện hàm: **exec MH\_SLB**



# HÀM VỚI GIÁ TRỊ TRẢ VỀ LÀ DỮ LIỆU KIỂU BẢNG (TT)

+ Sử dụng hàm:

```
select MH.Mahang as 'Mã hàng' , MH.TenHang as 'Tên hàng',  
       MH.Soluong as 'Tổng SL', MH_SLB.SL_Ban as 'SL bán',  
       ( MH.soluong - MH_SLB.SL_Ban)as 'SL hiện có'  
from MatHang MH join dbo.MH_SLB() MH_SLB  
on MH.Mahang = MH_SLB.Mahang
```

=> Kết quả

	Mã hàng	Tên hàng	Tổng SL	SL bán	SL hiện có
1	HC_VO001	Vở ghi 96 trang HC	800	45	755
2	TL_BB001	Bút bi	1000	35	965
3	TL_BM001	Bút máy	500	5	495
4	TL1_VO	Vở ghi 96 trang	600	8	592

# HÀM VỚI GIÁ TRỊ TRẢ VỀ LÀ DỮ LIỆU KIỂU BẢNG (TT)

- 2. Multi-statement table-value: cũng trả về kết quả là bảng dữ liệu nhưng phần thân có thể có nhiều câu lệnh SQL. Khi cần sử dụng nhiều câu lệnh trong phần thân hàm, sử dụng cú pháp sau:

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      //trả về kết quả cho hàm  
END
```



# HÀM VỚI GIÁ TRỊ TRẢ VỀ LÀ DỮ LIỆU KIỂU BẢNG (TT)

---

Lưu ý:

- Cấu trúc bảng trả về bởi hàm được xác định dựa vào định nghĩa của bảng trong mệnh đề RETURNS.
- Biến *@<biến\_bảng>* trong mệnh đề RETURNS có phạm vi sử dụng trong hàm và được sử dụng như một tên bảng.
- Câu lệnh RETURN trong thân hàm không chỉ định giá trị trả về. Giá trị trả về của hàm chính là các dòng dữ liệu trong bảng có tên là *@<biếnbảng>* được định nghĩa trong mệnh đề RETURNS



# HÀM VỚI GIÁ TRỊ TRẢ VỀ LÀ DỮ LIỆU KIỂU BẢNG (TT)

- ❖ Ví dụ: Viết hàm **hiển thị nhân viên và số lượng đơn đặt hàng của từng nhân viên trong tháng 7/2014**

+ Tạo hàm:

```
Create function NhanVien_SoDDH ()
```

```
returns @NV_SoDDH
```

```
table ( MaNV varchar(5), HoTenNV nvarchar(40), Soluong_DDH int)
```

```
as
```

```
begin
```

```
    insert into @NV_SoDDH
```

```
    select NV.MaNV, HoTenNV, count(SoHD)
```

```
    from NhanVien NV join DonDatHang DDH
```

```
        on NV.MaNV=DDH.MaNV
```

```
    Where month(Ngaydathang) = 7 and year(Ngaydathang) = 2014
```

```
    groupby NV.MaNV, HoTenNV
```

```
return
```

```
end
```



# HÀM VỚI GIÁ TRỊ TRẢ VỀ LÀ DỮ LIỆU KIỂU BẢNG (TT)

+ Sử dụng hàm:

`select *`

`from dbo.NhanVien_SoDDH()`

=> Kết quả



	MaNV	HoTenNV	Soluong_DDH
1	NV01	Hoàng Anh Tuấn	1
2	NV02	Vũ Thị Lan	4
3	NV03	Trần Ngọc Diệp	1

# HÀM VỚI GIÁ TRỊ TRẢ VỀ LÀ DỮ LIỆU KIỂU BẢNG (TT) – VÍ DỤ

---

```
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
```



## HÀM VỚI GIÁ TRỊ TRẢ VỀ LÀ DỮ LIỆU KIỂU BẢNG (TT)

---

```
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
```





# HÀM VỚI GIÁ TRỊ TRẢ VỀ LÀ DỮ LIỆU KIỂU BẢNG (TT)

---

- Câu lệnh:

`SELECT * FROM dbo.func_TongSV(25)`

- Sẽ cho kết quả thống kê tổng số sinh viên khoá 25 của mỗi khoa:

- Còn câu lệnh:

`SELECT * FROM dbo.func_TongSV(0)`

- Cho ta biết tổng số sinh viên hiện có (tất cả các khoá) của mỗi khoa



# THAY ĐỔI HÀM

Cú pháp:

ALTER FUNCTION tương tự như CREATE FUNCTION

❖ Ví dụ: Tính tổng lương trung bình của nhân viên từ hàm tính tổng lương của nhân viên:

+Tạo hàm:

Create function TongLuong\_NV()

Returns float

As Begin

declare @TongLuong float

select @TongLuong =(select sum(luongCB)  
from NhanVien )

return @TongLuong

end



# THAY ĐỔI HÀM (TT)

+ Sử dụng hàm:

`select` dbo.TongLuong\_NV() `as` ' Tong lương của NV'

=> Kết quả

	Tong lương của NV
1	63502000



# THAY ĐỔI HÀM (TT)

Ta thay đổi hàm: TongLuong\_NV()

+ Tạo hàm:

Alter function TongLuong\_NV()

Returns float

as

begin

declare @LuongTB float

select @LuongTB=(select avg(luongCB)  
from NhanVien )

return @LuongTB

end



# THAY ĐỔI HÀM (TT)

+ Sử dụng hàm:

`Select` dbo.TongLuong\_NV() `as` 'Luong trung binh cua NV'

=> Kết quả

	Luong trung binh cua NV
1	12700400



# XÓA HÀM

Cú pháp:

**DROP FUNCTION** tên\_hàm

Vd: **drop function** NhanVien\_SoDDH





TRIGGER

# TRIGGER – TRIGGER LÀ GÌ?

---

- 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





# CÁC XỬ LÝ BÊN TRONG TRIGGER

---

- Kiểm tra các ràng buộc dữ liệu phức tạp
  - Các ràng buộc mô tả phức tạp, không thể dùng constraint
  - Gọi hành động Rollback Tran để hủy thao tác cập nhật khi vi phạm ràng buộc
  - Bảo đảm dữ liệu luôn được toàn vẹn
  - Bảo đảm việc kiểm thử ứng dụng không làm hư dữ liệu có sẵn
- Tính toán, tự động cập nhật giá trị
  - Bổ sung các hành động cập nhật dữ liệu để đảm bảo tính toàn vẹn dữ liệu
  - Đơn giản hoá việc xây dựng ứng dụng
- Chỉ định các bẫy lỗi dễ hiểu
  - Tăng tính thân thiện của ứng dụng
  - Dễ dàng nhận ra các lỗi khi lập trình



# 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
  - GRANT/REVOKE



# CÁC LOẠI TRIGGER

---

- Có 2 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.



# CÁC BẢNG TRUNG GIAN INSERTED VÀ DELETED

---

## □ Inserted

- Chứa dữ liệu được thêm mới trong hành động INSERT/UPDATE
- Có ở cả hai loại trigger
- Cấu trúc bảng giống với bảng thực sự được cập nhật dữ liệu

## □ Deleted

- Chứa dữ liệu bị xoá trong hành động DELETE/UPDATE
- Có ở cả hai loại trigger
- Cấu trúc bảng giống với bảng thực sự được cập nhật dữ liệu

## □ Hành động update trong SQL Server

- Xoá dòng dữ liệu cũ
- Thêm vào dòng dữ liệu mới với thông tin đã cập nhật



# TẠO MỚI TRIGGER

---

Trigger có thể được tạo bằng câu lệnh hoặc bằng SQL Server Management Studio. Bấm chuột phải **Database Triggers, new Database trigger**

- Trong cả hai trường hợp, câu lệnh CREATE TRIGGER được sử dụng để tạo ra trigger.

```
CREATE TRIGGER Tên_Trigger ON Tên_bảng  
{ [ INSTEAD OF ] | [ FOR | AFTER ] }  
{ [ INSERT [, UPDATE [,DELETE ] ] ] }  
AS
```

```
[DECLARE Biến_cục_bộ]  
    Các_lệnh
```



# MÔ TẢ

---

- Tên bảng
  - Tên bảng mà trigger tạo mới sẽ liên kết
- INSTEAD OF: chỉ định đây là trigger loại instead of trigger
  - Mỗi bảng chỉ có quyền tạo một instead of trigger cho một hành động cập nhật
- FOR hoặc AFTER
  - Nếu tạo trigger thông thường
- INSERT, UPDATE, DELETE
  - Hành động cập nhật dữ liệu tác động vào bảng để kích hoạt trigger.



# XÓA TRIGGER

---

- Cú pháp

```
DROP TRIGGER Tên_trigger
```



# SỬA NỘI DUNG TRIGGER

---

## □ Sửa nội dung

```
ALTER TRIGGER Tên_Trigger ON Tên_bảng  
FOR INSERT [, UPDATE [,DELETE ]]  
AS  
[DECLARE Biến_cục_bộ]  
    Các_lệnh
```





# TRIGGER LỒNG NHAU

---

- Trigger có thể lồng nhau
  - Hành động cập nhật → Trigger → Cập nhật bảng khác → Trigger trên bảng tương ứng
  - Instead Of trigger không phát sinh lại trên chính bảng mà nó liên kết
    - Cập nhật → Instead of Trigger → Gọi câu lệnh cập nhật xuống bảng → Instead of trigger
- Số cấp lồng tối đa
  - 32 cấp
  - Sử dụng biến @NestedLevel
- Cấu hình cho phép trigger lồng nhau
  - EXEC sp\_configure '**nested triggers**', [0 | 1]





# TRIGGER KIỂM TRA RÀNG BUỘC DỮ LIỆU

# KHI THÊM MỚI MẪU TIN

---

- Thường dùng để kiểm tra
  - Khóa ngoại, Miền giá trị, 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



## KHI HỦY BỎ MẪU TIN

---

- Tương tự, kiểm tra các ràng buộc như trigger INSERT
- Nên kiểm tra ràng buộc khoá ngoại
  - Thông thường ràng buộc này dẫn đến việc phải cập nhật một số dữ liệu trên bảng khác
  - Chú ý: SQL Server 2000 có thuộc tính CASCADE DELETE



# KHI SỬA ĐỔI MẪU TIN

---

- Tương tự, kiểm tra các ràng buộc như trigger INSERT
  - Ràng buộc khoá ngoại có thể sử dụng CASCADE UPDATE để thực hiện tự động
- Xác định cột đang được cập nhật
  - If Update(Tên\_cột)
  - Xử lý



# TRIGGER CẬP NHẬT GIÁ TRỊ TỰ ĐỘNG

---

- Sau khi kiểm tra ràng buộc trigger có thể
  - Rollback nếu dữ liệu không hợp lệ
  - Thực hiện tiếp các hành động cập nhật trên bảng khác để đảm bảo toàn vẹn dữ liệu: Cập nhật giá trị tự động
  - Vd: Insert □ CTGiaoHang □ Cập nhật bảng TONKHO
- Các hành động cập nhật thường thực hiện
  - Hủy bỏ dữ liệu do quan hệ khoá ngoại
  - Tính lại các cột 'tính toán' trong các bảng liên quan
- Vị trí thực hiện
  - Trong cùng trigger kiểm tra ràng buộc đã định nghĩa
  - Sau khi kiểm tra dữ liệu đã hợp lệ (thoả mãn các ràng buộc)



# INSTEAD OF TRIGGER

---

- Bảng ảo thông thường có thể được cập nhật nhưng có nhiều giới hạn
  - Group By, Order By, Distinct
  - Ràng buộc khoá ngoại
  - Thiếu các cột NOT NULL trong bảng
- Trigger Instead of
  - Xảy ra trước khi SQL Server kiểm tra ràng buộc
  - Thay đổi hành động cập nhật vào bảng ảo bằng hành động thích hợp trên bảng gốc



# CÁC VÍ DỤ VỀ SỬ DỤNG TRIGGER

## Insert :

```
IF EXISTS (SELECT NAME FROM SYSOBJECTS
WHERE NAME = 'TR_BANHANG_INSERT' AND TYPE='TR' ) DROP
    TRIGGER TR_BANHANG_INSERT
GO
CREATE TRIGGER TR_BANHANG_INSERT ON BANHANG
FOR INSERT
AS
BEGIN
UPDATE MATHANG
SET SOLUONG = SOLUONG - INSERTED.SOLUONGBAN FROM
    INSERTED
WHERE INSERTED.MAHANG = MATHANG.MAHANG
END
```





## TA CÓ CƠ SỞ DỮ LIỆU:

Results			
Messages			
	MaHang	TenHang	SoLuong
1	HH001	A	100
2	HH002	B	100
3	HH003	C	100

	MaBanHang	MaHang	SoLuongBan
1	BH001	HH001	10
2	BH002	HH002	10
3	BH003	HH003	10

=> Kết quả

Results			
Messages			
	MaHang	TenHang	SoLuong
1	HH001	A	100
2	HH002	B	90
3	HH003	C	100

	MaBanHang	MaHang	SoLuongBan
1	BH001	HH001	10
2	BH002	HH002	10
3	BH003	HH003	10
4	BH004	HH002	10

# CÁC VÍ DỤ VỀ SỬ DỤNG TRIGGER

## Delete:

```
IF EXISTS (SELECT NAME FROM SYSOBJECTS
WHERE NAME = 'TR_BANHANG_DELETE' AND TYPE ='TR')
DROP TRIGGER TR_BANHANG_DELETE
GO
CREATE TRIGGER TR_BANHANG_DELETE ON BANHANG
FOR DELETE
AS
BEGIN
UPDATE MATHANG
SET SOLUONG = SOLUONG + DELETED.SOLUONGBAN FROM
DELETED
WHERE INSERTED.MAHANG = MATHANG.MAHANG
END
```



=> Kết quả

Results Messages			
	MaBanHang	MaHang	SoLuongBan
1	BH001	HH001	10
2	BH002	HH002	10
3	BH003	HH003	10


  

	MaHang	TenHang	SoLuong
1	HH001	A	100
2	HH002	B	100
3	HH003	C	100

# CÁC VÍ DỤ VỀ SỬ DỤNG TRIGGER

## Update:

```
IF EXISTS (SELECT NAME FROM SYSOBJECTS
WHERE NAME = 'TR_BANHANG_INSERT' AND TYPE = 'TR')
    DROPTRIGGER TR_BANHANG_UPDATE
GO
CREATE TRIGGER TR_BANHANG_UPDATE ON BANHANG
FOR UPDATE
AS
BEGIN
IF UPDATE (SOLUONGBAN) BEGIN
IF EXISTS (SELECT MH.MAHANG
FROM ( SELECT INSERTED.MAHANG,
SUM(INSERTED.SOLUONGBAN - DELETED.SOLUONGBAN) AS TONGSL
FROM INSERTED JOIN DELETED ON INSERTED.MABANHANG =
DELETED.MABANHANG
GROUP BY INSERTED.MAHANG) AS TMP JOIN MATHANG MH
```



```
ON MH.MAHANG = TMP.MAHANG
WHERE TMP.TONGSL > MH.SOLUONG)
BEGIN
ROLLBACK TRAN END
ELSE
BEGIN
UPDATE MATHANG
SET SOLUONG = SOLUONG - TMP1.TONGSL
FROM ( SELECT INSERTED.MAHANG,
SUM(INSERTED.SOLUONGBAN - DELETED.SOLUONGBAN) AS
TONGSL
FROM INSERTED JOIN DELETED ON INSERTED.MABANHANG =
DELETED.MABANHANG GROUP BY INSERTED.MAHANG) AS
TMP1
WHERE TMP1.MAHANG = MATHANG.MAHANG
END
END
END
```



○ => Kết quả

Results		Messages	
	MaBanHang	MaHang	SoLuongBan
1	BH001	HH001	10
2	BH002	HH002	15
3	BH003	HH003	10

	MaHang	TenHang	SoLuong
1	HH001	A	100
2	HH002	B	95
3	HH003	C	100

## KẾT LUẬN

- Việc sử dụng Trigger là không bắt buộc.
- Trigger thường được dùng để bảo đảm tính toàn vẹn của CSDL.

