



## CHƯƠNG 3

# LẬP TRÌNH CƠ SỞ DỮ LIỆU BẰNG T-SQL

# NỘI DUNG



## 1. KHAI BÁO VÀ SỬ DỤNG BIẾN

## 2. CÁC TOÁN TỬ

## 3. CÁC CẤU TRÚC ĐIỀU KHIỂN

## 4. CÁC HÀM THÔNG DỤNG

# NỘI DUNG



**5. THỦ TỤC THƯỜNG TRÚ (STORED PROCEDURE)**

**6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA**

**7. TRIGGER**

**8. KIỂU DỮ LIỆU CURSOR**



# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



- Biến được dùng để lưu trữ các giá trị tạm thời trong quá trình tính toán các xử lý. Dữ liệu có thể được truyền đến câu lệnh SQL bằng cách sử dụng tên biến.
- Trong T – SQL, biến có thể phân thành 2 loại:
  - *Biến cục bộ*
  - *Biến hệ thống*

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Biến cục bộ

- Cũng giống như các ngôn ngữ lập trình, biến cục bộ trong T-SQL được sử dụng trong quá trình viết mã lệnh T-SQL.
- Biến cục bộ là biến do người lập trình tạo ra.
- Tên biến phải khai báo bắt đầu bằng ký hiệu @

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Khai báo biến cục bộ

Cú pháp:

```
DECLARE @<Ten_bien> <Kieu_du_lieu> [, ...]
```

**Ví dụ:** Khai báo biến dùng để lưu trữ tên môn học, điểm của sinh viên.

```
DECLARE @Tenmh Varchar(20), @Diem int
```

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Gán giá trị cho biến cục bộ

- Dùng lệnh **SET** hoặc **SELECT**.

### Cú pháp:

```
SET @<tên biến> = <giá trị>|<biểu thức>|<câu truy vấn>
```

Hay

```
SELECT @<tên biến1> = <giá trị1>|<biểu thức1>|<câu truy vấn1>,  
        @<tên biến2> = <giá trị2>|<biểu thức2>|<câu truy vấn2>,  
        ...
```



# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Gán giá trị cho biến cục bộ

- Dùng lệnh **SET** hoặc **SELECT**.

**Ví dụ 1:** Khai báo biến @ngayxh và gán giá trị ngày hiện tại cho biến vừa tạo

```
DECLARE @ngayxh DATE
```

```
SET @ngayxh = getdate ()
```

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Gán giá trị cho biến cục bộ

**Ví dụ 2:** Gán giá trị cho biến có tên là **@Diem\_max**, dữ liệu gán lấy từ bảng **KETQUA**

```
DECLARE @Diem_max int
```

```
SET @Diem_max = ( SELECT MAX (DIEM)  
                  FROM KETQUA)
```

Câu truy vấn trả về *một giá trị duy nhất* (là điểm lớn nhất từ bảng KETQUA) nên phép gán là hợp lệ.

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Gán giá trị cho biến cục bộ

**Ví dụ 3.1:** Tính tổng lương của tất cả các nhân viên phòng Nghiên cứu.

```
DECLARE @Tongluong INT
SELECT @Tongluong = SUM(luong)
FROM nhanvien nv, phongban pb
WHERE     nv.PHG = pb.MAPHG
          AND pb.TENPHG = N'Nghiên cứu'
```

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Gán giá trị cho biến cục bộ

Lưu ý:

- Lệnh **SET** chỉ sử dụng để gán giá trị cho một biến.
- Lệnh **SELECT** có thể sử dụng để gán giá trị cho nhiều biến cùng lúc.

```
declare @ngay date , @a int  
set @ngay = GETDATE()  
set @a = 5
```

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Gán giá trị cho biến cục bộ

Lưu ý:

- Lệnh **SET** chỉ sử dụng để gán giá trị cho một biến.
- Lệnh **SELECT** có thể sử dụng để gán giá trị cho nhiều biến cùng lúc.

```
declare @ngay date , @a int  
select @ngay = GETDATE() , @a = 5
```

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Khai báo biến cục bộ

Lưu ý:

- Gán giá trị khởi tạo cho biến lúc khai báo

`declare @ngay = getdate() date` ⇒ **SAI**

`declare @ngay date = getdate()` ⇒ **ĐÚNG**

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Khai báo biến cục bộ

Lưu ý:

- Khai báo biến cùng kiểu dữ liệu

`declare @a , @b int`  $\Rightarrow$  **SAI**

`declare @a int, @b int`  $\Rightarrow$  **ĐÚNG**

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Khai báo biến cục bộ

Lưu ý:

- Kiểu dữ liệu **text**, **ntext** hoặc **image** không được chấp nhận khi khai báo biến.
- Phạm vi hoạt động của biến chỉ nằm trong một thủ tục hoặc một lô có chứa lệnh khai báo biến.



# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



**Xem (in) giá trị hiện hành của biến cục bộ**

- Dùng lệnh **PRINT** hoặc **SELECT**.

**Cú pháp:**

```
PRINT @<tên biến>| <biểu thức chuỗi>
```

Hay

```
SELECT @<tên biến>|số|<biểu thức chuỗi>
```

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Xem giá trị hiện hành của biến cục bộ

**Ví dụ 3.1:** Tính tổng lương của tất cả các nhân viên phòng Nghiên cứu

```
DECLARE @Tongluong INT
SELECT @Tongluong = SUM(luong)
FROM nhanvien nv, phongban pb
WHERE nv.PHG = pb.MAPHG
      AND pb.TENPHG = N'Nghiên cứu'
PRINT 'Tong luong phong nghien cuu: '
PRINT @Tongluong
```

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Xem giá trị hiện hành của biến cục bộ

Lệnh **PRINT** được dùng để in ra giá trị của một biến duy nhất hoặc một giá trị cụ thể.

Trường hợp muốn in ra nhiều giá trị thì phải dùng nhiều lệnh **PRINT** tương ứng.

Lệnh **SELECT** có thể in ra giá trị cho nhiều biến cùng lúc.

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



**Xem giá trị hiện hành của biến cục bộ**

**Ví dụ 2: Dùng **Print** để in**

```
DECLARE @MASV CHAR(10), @TUOI INT
SET @MASV = '300907106'
SET @TUOI = 23
PRINT @MASV
PRINT @TUOI
```

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



**Xem giá trị hiện hành của biến cục bộ**

**Ví dụ 2: Dùng **Select** để in**

```
DECLARE @MASV CHAR(10), @TUOI INT
SET @MASV = '300907106'
SET @TUOI = 23
SELECT @MASV, @TUOI
```

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN

**In giá trị của biến kết hợp với chuỗi:** Nếu biến là kiểu số thì phải chuyển sang kiểu chuỗi bằng cách dùng hàm **Convert**.

**Ví dụ 3:**

```
DECLARE @MASV CHAR(10), @TUOI INT
```

```
SET @MASV = '300907106'
```

```
SET @TUOI = 23
```

```
PRINT @MASV + @TUOI
```

**SAI**

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN

In giá trị của biến kết hợp với chuỗi: Nếu biến là kiểu số thì phải chuyển sang kiểu chuỗi bằng cách dùng hàm **Convert**.

**Ví dụ 3:**

```
DECLARE @MASV CHAR(10), @TUOI INT
```

```
SET @MASV = '300907106'
```

```
SET @TUOI = 23
```

```
PRINT @MASV + CONVERT(CHAR(2), @TUOI)
```

**ĐÚNG**

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Phạm vi hoạt động của biến cục bộ

- Trong T–SQL phạm vi của biến chỉ thuộc về một thủ tục hoặc một lô (**batch**).
- Các lô được ngăn cách nhau bởi từ khóa **GO**.



# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Phạm vi hoạt động của biến cục bộ

**Ví dụ sau thể hiện 2 lô:** Tính tổng lương của tất cả các nhân viên phòng Nghiên cứu

```
DECLARE @Tongluong INT
SELECT @Tongluong = SUM(luong)
FROM nhanvien nv, phongban pb
WHERE nv.PHG = pb.MAPHG
      AND pb.TENPHG = 'Nghien cuu'
GO
PRINT 'Tong luong phong nghien cuu: '
PRINT @Tongluong --Lỗi -- Biến này không còn ý nghĩa
GO
```

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Biến hệ thống – Một vài biến thông dụng

Tên biến	Kiểu trả về	Ý nghĩa
@ @Version	Chuỗi	Phiên bản, ngày sản xuất, HĐH, CPU cài đặt máy chủ SQL.
@ @RowCount	Số nguyên	Tổng số mẫu tin được tác động bởi câu lệnh T-SQL gần nhất.
@ @ServerName	Chuỗi	Tên của Server cài đặt SQL Server.
@ @ServiceName	Chuỗi	Tên của dịch vụ chạy kèm theo SQL Server

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



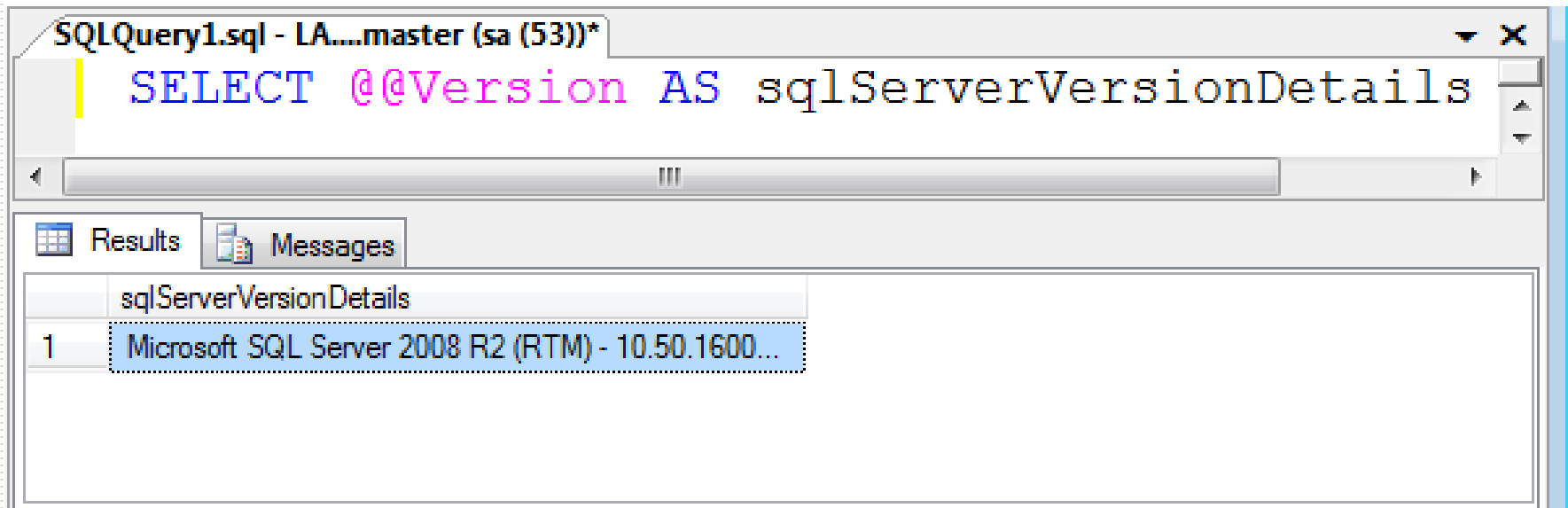
## Biến hệ thống – Một vài biến thông dụng

Tên biến	Kiểu trả về	Ý nghĩa
<b>@ @Error</b>	Số nguyên	Mã lỗi của câu lệnh gần nhất, khi câu lệnh thực hiện thành công biến này có giá trị 0.
<b>@ @Language</b>	Chuỗi	Tên ngôn ngữ mà SQL đang dùng, mặc định là US_English.
<b>@ @Fetch_Status</b>	Số nguyên	Trạng thái của việc đọc dữ liệu trong bảng theo từng dòng (Cursor). Khi đọc dữ liệu thành công biến này có giá trị 0.
<b>@ @Connections</b>	Số nguyên	Tổng số kết nối vào SQL Server.

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Biến hệ thống



**Đây là biến hệ thống được sử dụng để xem thông tin phiên bản của SQL Server**

# 1. KHAI BÁO VÀ SỬ DỤNG BIẾN



## Biến hệ thống

Ví dụ: Câu lệnh **SELECT \* FROM MONHOC** có kết quả là 3 dòng dữ liệu. Dòng lệnh sau để xem giá trị của biến @@Rowcount:

```
PRINT @@Rowcount
```

→ Kết quả: **3**

# HB1\_30.8.2024



❖ Mới giảng lý thuyết tới đây, BT 3.1-3.2 tuần sau làm



## 2. CÁC TOÁN TỬ

## 2. CÁC TOÁN TỬ



**T-SQL cung cấp cho chúng ta một số toán tử như sau:**

1. Toán tử số học: +, -, \*, /, % (lấy phần dư)
2. Toán tử nối chuỗi: +
3. Toán tử so sánh: =, >, >=, <, <=, <>, != (khác), !> (không lớn hơn), !< (không nhỏ hơn)
4. Toán tử luận lý: And, Or, Not



## 2. CÁC TOÁN TỬ



### Lưu ý: Đối với Toán tử số học

- Các toán hạng sử dụng trong biểu thức phải có một trong các kiểu dữ liệu sau: **int, bigint, smallint, tinyint, numeric, decimal, float, real, money** và **smallmoney**.
- Toán tử **chia lấy phần dư (%)** chỉ giới hạn sử dụng các kiểu dữ liệu như: **int, bigint, smallint** và **tinyint**.
- Thứ tự ưu tiên các toán tử là: **\***, **/**, **%**, **+**, **-**.

### Ví dụ:

**PRINT** 2 + 3 \* 4 % 5 → **4**



### 3. CÁC CẤU TRÚC ĐIỀU KHIỂN

### 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



#### **BEGIN ... END**

- Một tập lệnh | khối lệnh SQL được thực thi sẽ được đặt trong BEGIN...END

**Cú pháp:**

BEGIN

<lệnh | đoạn lệnh>

END

# 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



## CẤU TRÚC IF ... ELSE

Cú pháp:

```
IF <điều kiện 1>  
    <lệnh | khối lệnh 1>  
ELSE IF <điều kiện 2>  
    <lệnh | khối lệnh 2>  
...  
ELSE  
    <lệnh | khối lệnh n>
```

# 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



## CẤU TRÚC IF ... ELSE

**Lưu ý:**

- Cú pháp IF được dùng trong 1 lô hay trong 1 thủ tục, hàm hay Trigger.

### 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



#### CẤU TRÚC IF ... ELSE

**Ví dụ 3.2:** Kiểm tra xem có môn học nào có số tiết lớn hơn 30 hay không. Nếu có thì in ra danh sách những môn học đó gồm (MaMH, TenMH). Nếu không thì in ra thông báo ‘**Không có môn học nào có số tiết > 30**’.

**MONHOC** (MaMH, TenMH, SoTiet)

### 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



#### CẤU TRÚC IF ... ELSE

```
IF ( SELECT COUNT (*) FROM MONHOC
      WHERE SOTIET > 30 ) > 0
BEGIN
    PRINT 'Danh sach cac mon hoc co so tiet >30 la:'
    SELECT MAMH, TENMH
    FROM MONHOC
    WHERE SOTIET > 30
END
ELSE
    PRINT 'Khong co mon hoc nao co so tiet > 30.'
```

### 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



#### CẤU TRÚC IF ... ELSE

Ngoài ra, có thể sử dụng từ khoá **EXISTS** kết hợp với cấu trúc IF để kiểm tra sự tồn tại các dòng dữ liệu trong bảng một cách hiệu quả hơn.

#### Cú pháp:

**IF EXISTS** (Câu lệnh SELECT)

⟨Câu lệnh 1⟩ | ⟨Khối lệnh 1⟩

[**ELSE**

⟨Câu lệnh 2⟩ | ⟨Khối lệnh 2⟩]



# 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



## CẤU TRÚC IF ... ELSE

Lưu ý:

- Từ khoá **EXISTS**: Dùng để kiểm tra sự tồn tại các dòng dữ liệu trong câu lệnh SELECT sau nó.
- Kết quả IF trả về:
  - **đúng** (TRUE) khi câu lệnh SELECT trả về ít nhất một dòng dữ liệu,
  - ngược lại trả về sai (FALSE).

# 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



## CẤU TRÚC IF ... ELSE

### Ví dụ 3.2:

```
IF exists (select MAMH from MONHOC
           where SOTIET>30)
```

```
    BEGIN
```

```
        PRINT 'Danh sach cac mon hoc co so tiet >30 la:'
```

```
        SELECT MAMH, TENMH
```

```
        FROM MONHOC
```

```
        WHERE SOTIET > 30
```

```
    END
```

```
ELSE
```

```
    PRINT 'Khong co mon hoc nao co so tiet > 30.'
```

### 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



#### CẤU TRÚC IF ... ELSE

**Ví dụ 3.3:** Kiểm tra và in ra họ tên (HOTEN) những sinh viên có điểm thi lớn hơn 5, nếu không có thì in ra thông báo “Không có sinh viên nào có điểm thi lớn hơn 5”.

**SINHVIEN** (MaSV, HoTen, NgaySinh, Dchi,  
GioiTinh, MaLop)

**KETQUA** (MaSV, MaMH, LanThi, Diem)

# 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



## CẤU TRÚC IF ... ELSE

```
IF EXISTS (SELECT * FROM KETQUA WHERE DIEM > 5)
BEGIN
    PRINT 'Danh sách sinh viên có điểm thi lớn hơn 5'
    SELECT HOTEN
    FROM SINHVIEN sv, KETQUA kq
    WHERE kq.MASV = sv.MASV AND DIEM > 5
END
ELSE
    PRINT 'Không có sinh viên nào có điểm thi lớn hơn
5.'
```

# 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



## CẤU TRÚC CASE

- Trong T–SQL biểu thức **CASE** vô cùng hữu ích, nó giống như cú pháp SWITCH...CASE trong C.
- Được dùng để xử lý điều kiện cho cột được chọn trong câu lệnh truy vấn dữ liệu.
- Cú pháp của biểu thức **CASE** có 2 dạng.

### 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



#### CẤU TRÚC CASE

##### Cú pháp 1:

```
CASE <biểu thức>  
    WHEN <gtri_1> THEN <kquả_1>  
    [WHEN <gtri_2> THEN <kquả_2>  
    ...  
    ]  
    [ELSE <kquả_n>]  
END
```

### 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



#### CẤU TRÚC CASE

##### Cú pháp 2:

CASE

```
    when <bt_logic_1> then <kquả_1>  
  [when <bt_logic_2> then <kquả_2>  
  ...  
  ]  
  [ELSE <kquả_n>]
```

END

# 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



## CẤU TRÚC CASE

### Ví dụ 1:

```
SELECT MASV, DIEM, KETQUA = CASE
                                WHEN DIEM >= 5 THEN 'DAT'
                                ELSE 'CHUA DAT'
                                END
FROM SINHVIEN
```



### 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



#### CẤU TRÚC CASE – Ví dụ 2:

```
SELECT MaSV, TenSV,  
CASE  
    WHEN dtb < 5.0 THEN 'Yeu'  
    WHEN dtb >= 5.0 AND dtb < 6.5 THEN 'Trung binh'  
    WHEN dtb >= 6.5 AND dtb < 8.0 THEN 'Kha'  
    WHEN dtb >= 8.0 AND dtb < 9.0 THEN 'Gioi'  
    WHEN dtb >= 9.0 AND dtb <= 10 THEN 'Xuat sac'  
    ELSE 'Khong the xep loai'  
END AS 'Xep Loai'  
FROM SINHVIEN
```

# 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



## CẤU TRÚC CASE – Ví dụ 3:

```
UPDATE SINHVIEN
```

```
SET XepLoai =
```

```
CASE
```

```
    WHEN dtb < 5.0 THEN 'Yeu'
```

```
    WHEN dtb >= 5.0 AND dtb < 6.5 THEN 'Trung binh'
```

```
    WHEN dtb >= 6.5 AND dtb < 8.0 THEN 'Kha'
```

```
    WHEN dtb >= 8.0 AND dtb < 9.0 THEN 'Gioi'
```

```
    WHEN dtb >= 9.0 AND dtb <= 10 THEN 'Xuat sac'
```

```
    ELSE 'Khong the xep loai'
```

```
END
```

### 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



## CẤU TRÚC LẶP WHILE

Cú pháp:

**WHILE** <Biểu thức luận lý>

**BEGIN**

<Câu lệnh>

**END**

# 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



## CẤU TRÚC LẶP WHILE

**Lưu ý:**

- Cấu trúc lặp được dùng bên trong lô, trong một thủ tục, hàm, trigger.
- Break: thoát khỏi vòng lặp
- Continue: bỏ qua các lệnh sau nó, trở lại đầu vòng lặp While.

# 3. CÁC CẤU TRÚC ĐIỀU KHIỂN



## CẤU TRÚC LẶP WHILE

**Ví dụ:** Cho biết kết quả khi chạy đoạn chương trình sau:

```
DECLARE @a int, @b int
SET @a = 1
SET @b = @a + 1
WHILE (@a < 10)
BEGIN
    PRINT @a + @b
    IF (@a + @b = 6)
        BREAK
    SET @a = @a + 1
END
```

# BÀI TẬP



Cho CSDL:

SINHVIEN(MASV,HOTEN,NGAYSINH)

MONHOC(MAMH,TENMH,SOTC)

KETQUA(MASV,MAMH,DIEM)

Yêu cầu:

Sử dụng khai báo, gán giá trị và in ra giá trị biến gồm **họ tên, tuổi, điểm trung bình** và **tổng số tín chỉ đạt** của sinh viên có mã là 'SV001'

## Bài tập (tt)

DEAN(MADA, TENDA, NGAYBD, NGAYKT)

NHANVIEN(MANV, HOTEN, PHAI)

THAMGIA(MANV, MADA, SOGIO)

Sử dụng các biến và cấu trúc điều khiển viết các lệnh sau:

1/ Khai báo 2 biến có tên @tenda để lưu tên đề án có mã là 'DA001' và @sonv để lưu số nhân viên tham gia. In ra giá trị 2 biến.

2/ Nhập vào một mã đề án, kiểm tra tình trạng của đề án: Chưa thực hiện, đang thực hiện hoặc đã kết thúc.

3/ Nhập vào mã nhân viên và mã đề án, kiểm tra nhân viên có tham gia đề án đó không. Nếu có thì in ra thông báo: *Số giờ thực hiện là 20*, ngược lại in ra thông báo: *'Nhân viên Nguyễn Văn A không tham gia đề án DA001'* (các giá trị 20, Nguyễn Văn A là các giá trị ví dụ)



# CÁC HÀM THÔNG DỤNG



## 4. CÁC HÀM THÔNG DỤNG



- Hàm là tập lệnh T–SQL để thực hiện một công việc nào đó. Hàm trong SQL làm việc với dữ liệu, nhóm dữ liệu để trả về một kết quả mong đợi.
- T–SQL có các loại hàm sau:
  - Hàm tập hợp
  - Các hàm xử lý chuỗi
  - Các hàm toán học
  - Các hàm xử lý ngày giờ
  - Các hàm chuyển đổi kiểu dữ liệu

## 4. CÁC HÀM THÔNG DỤNG



### HÀM TẬP HỢP

Tên hàm	Giá trị trả về
SUM(col_name)	Hàm tính tổng, trả về tổng giá trị của col_name
AVG(col_name)	Trả tính giá trị trung bình
COUNT	Đếm số bản ghi trong bảng
MAX(col_name)	Trả về giá trị lớn nhất
MIN(col_name)	Trả về giá trị nhỏ nhất

## 4. CÁC HÀM THÔNG DỤNG



### HÀM XỬ LÝ CHUỖI

Tên hàm
UPPER (chuỗi dữ liệu)
LOWER (chuỗi dữ liệu)
LEFT (chuỗi nguồn, số ký tự)
RIGHT (chuỗi nguồn, số ký tự)
SUBSTRING (chuỗi nguồn, vị trí, số ký tự)

## 4. CÁC HÀM THÔNG DỤNG



### HÀM XỬ LÝ CHUỖI

Tên hàm	Giá trị trả về
LTRIM(chuỗi dữ liệu)	cắt bỏ khoảng trắng ở đầu chuỗi
RTRIM(chuỗi dữ liệu)	cắt bỏ khoảng trắng ở cuối chuỗi
SPACE(N)	Trả về là một chuỗi chứa N ký tự trắng
REPLICATE(chuỗi lặp,N)	Trả về là một chuỗi chứa các ký tự được lặp lại N lần
LEN(chuỗi dữ liệu)	Trả về chiều dài của chuỗi

## 4. CÁC HÀM THÔNG DỤNG



### HÀM XỬ LÝ CHUỖI

Tên hàm	Giá trị trả về
REVERSE(chuỗi dữ liệu)	Đảo ngược một chuỗi cho trước
STUFF(chuỗi nguồn, vị trí, chiều dài, chuỗi con)	<p>Trả về là một chuỗi mới sau khi đã huỷ bỏ một số ký tự hiện có và thêm vào một chuỗi con khác tại vị trí vừa huỷ bỏ.</p> <pre>PRINT STUFF('1234567', 3, 4, 'ABC')</pre> <p>→ '12ABC7'</p>

## 4. CÁC HÀM THÔNG DỤNG



### HÀM XỬ LÝ CHUỖI

Tên hàm	Giá trị trả về
REPLACE(chuỗi nguồn, chuỗi tìm, chuỗi thay thế)	<p>Cho phép tìm và thay thế (nếu có) một chuỗi con trong chuỗi nguồn bằng một chuỗi khác.</p> <pre>PRINT REPLACE('MON HOC THCB', 'THCB', 'CSDL') → 'MON HOC CSDL'</pre>
CHAR(số nguyên)	Trả về là một ký tự tương ứng trong bảng mã ASCII.
ASCII(ký tự)	Trả về là một số nguyên tương ứng với ký tự trong bảng mã ASCII.

# 4. CÁC HÀM THÔNG DỤNG



## HÀM TOÁN HỌC

Tên hàm	Giá trị trả về
ABS(biểu thức số)	Lấy trị tuyệt đối của một số
PI()	Hằng số PI trong toán học
POWER(biểu thức số, số mũ)	Tính lũy thừa của một số nào đó theo một số mũ
RAND([số nguồn])	Trả về là một số thực ngẫu nhiên.
ROUND(biểu thức số, Vị trí làm tròn)	Làm tròn một số

## 4. CÁC HÀM THÔNG DỤNG



### HÀM TOÁN HỌC

Tên hàm	Giá trị trả về
SIGN(biểu thức số)	Trả về là một số qui định dấu của một biểu thức số: <b>1</b> : biểu thức dương, <b>-1</b> : biểu thức âm, <b>0</b> : biểu thức bằng 0.
SQRT(biểu thức số)	Tính căn bậc 2 của một số.



## 4. CÁC HÀM THÔNG DỤNG



### HÀM TOÁN HỌC

Tên hàm	Giá trị trả về
FLOOR(số nguồn)	Trả về số nguyên lớn nhất nhỏ hơn hoặc bằng số nguồn. $\text{Floor}(3.3) \rightarrow 3$
CEILING(số nguồn)	Trả về số nguyên nhỏ nhất lớn hơn hoặc bằng số nguồn. $\text{Ceiling}(3.3) \rightarrow 4$

## 4. CÁC HÀM THÔNG DỤNG



### HÀM XỬ LÝ NGÀY GIỜ

Các hàm này thường có tham số vào là kiểu dữ liệu ngày giờ và giá trị trả về của chúng có thể là kiểu dữ liệu số, chuỗi hoặc ngày giờ.

## 4. CÁC HÀM THÔNG DỤNG



### HÀM XỬ LÝ NGÀY GIỜ

Bảng mô tả viết tắt của các đơn vị thời gian:

Từ viết tắt	Ý nghĩa	Miền giá trị
yy	Năm	1900 – 9999
qq	Quý	1 – 4
mm	Tháng	1 – 12
dd	Ngày trong tháng	1 – 31
dy	Ngày trong năm	1 – 366
wk	Tuần	1 – 53
dw	Ngày trong tuần	1 – 7

## 4. CÁC HÀM THÔNG DỤNG



### HÀM XỬ LÝ NGÀY GIỜ

Bảng mô tả viết tắt của các đơn vị thời gian:

Từ viết tắt	Ý nghĩa	Miền giá trị
hh	Giờ trong ngày	0 – 23
mi	Phút trong giờ	0 – 59
ss	Giây trong phút	0 – 59
ms	Phần trăm mili giây	0 - 999

## 4. CÁC HÀM THÔNG DỤNG



### HÀM XỬ LÝ NGÀY GIỜ

Tên hàm	Giá trị trả về
DATEADD(đơn vị thời gian, số nguyên, ngày chỉ định)	<p>Cộng hoặc trừ một ngày chỉ định một đơn vị thời gian và trả về một ngày mới theo yêu cầu.</p> <pre>PRINT DATEADD(dd, 14, '03/16/2007')</pre> <p>→ <b>Mar 30 2007 12:00AM</b></p> <pre>PRINT DATEADD(mm, 14, '03/16/2007')</pre> <p>→ <b>May 16 2008 12:00AM</b></p>

## 4. CÁC HÀM THÔNG DỤNG



### HÀM XỬ LÝ NGÀY GIỜ

Tên hàm	Giá trị trả về
DATEDIFF(đơn vị thời gian, ngày 1, ngày 2)	<p>Tính khoảng cách (hiệu) của 2 ngày bất kỳ và trả về là một số nguyên.</p> <pre>PRINT DATEDIFF(dd, '12/05/2007', '02/06/2008') → 63.</pre> <pre>PRINT DATEDIFF(mm, '12/05/2008', '02/06/2008') → -10.</pre> <pre>PRINT DATEDIFF(mm, '12/05/2008', '02/06/2009') → 2.</pre>

## 4. CÁC HÀM THÔNG DỤNG



### HÀM XỬ LÝ NGÀY GIỜ

Tên hàm	Giá trị trả về
GETDATE()	Lấy ngày giờ hiện tại theo ngày giờ của hệ thống.
DATENAME(đơn vị thời gian, ngày chỉ định)	Trả về <u>một chuỗi</u> thời gian đại diện của một ngày chỉ định theo một đơn vị thời gian bất kỳ. <code>PRINT DATENAME(dw, '03/13/2013')</code> → Wednesday

## 4. CÁC HÀM THÔNG DỤNG



### HÀM XỬ LÝ NGÀY GIỜ

Tên hàm	Giá trị trả về
DATEPART(đơn vị thời gian, ngày chỉ định)	Trả về <u>một số nguyên</u> thời gian đại diện của một ngày chỉ định theo một đơn vị thời gian bất kỳ.  PRINT DATEPART(dw, '03/13/2013')  → 4
DAY(ngày chỉ định) MONTH(ngày chỉ định) YEAR(ngày chỉ định)	



## 4. CÁC HÀM THÔNG DỤNG



### HÀM CHUYỂN ĐỔI KIỂU DỮ LIỆU

- Được dùng để chuyển đổi qua lại các kiểu dữ liệu tương thích nhau trong Microsoft SQL Server.
- Trong các xử lý, chúng ta thường chuyển đổi các kiểu dữ liệu số hoặc kiểu dữ liệu ngày giờ sang kiểu dữ liệu chuỗi để hiển thị ra màn hình.
- Các kiểu dữ liệu **image**, **text**, **ntext** rất hạn chế trong việc chuyển đổi qua lại các kiểu dữ liệu khác.

## 4. CÁC HÀM THÔNG DỤNG



### HÀM CHUYỂN ĐỔI KIỂU DỮ LIỆU

Tên hàm	Giá trị trả về
CAST (biểu thức <b>AS</b> kiểu dữ liệu)	<p>Chuyển đổi một biểu thức nào đó sang một kiểu dữ liệu bất kỳ.</p> <pre>PRINT      'Số nguyên tố: ' + CAST(123 AS VARCHAR(3))</pre> <p>→ Số nguyên tố 123</p>

## 4. CÁC HÀM THÔNG DỤNG



### HÀM CHUYỂN ĐỔI KIỂU DỮ LIỆU

Tên hàm	Giá trị trả về
CONVERT (kiểu dữ liệu, biểu thức [, định dạng])	<p>Chuyển đổi một biểu thức nào đó sang một kiểu dữ liệu bất kỳ và có thể theo một định dạng nào đó.</p> <pre>PRINT CONVERT (VARCHAR(11), GETDATE(), 101)</pre> <p>→ 03/11/2013</p>

Định dạng (yy)	Định dạng (yyyy)	Dạng hiển thị dữ liệu
1	101	mm/dd/yy
2	102	yy.mm.dd
3	103	dd/mm/yy
4	104	dd.mm.yy
5	105	dd-mm-yy
6	106	dd mon yy
7	107	mon dd yy
8	108	hh:mm:ss
9	109	mon dd yyyy hh:mm:ss
10	110	mm-dd-yy
11	111	yy/mm/dd
12	112	Yymmdd
13	113	dd mon yyyy hh:mm:ss
14	114	hh:mm:ss:mmm
	21 hoặc 121	vvvv-mm-dd hh:mm:ss mmm

## 4. CÁC HÀM THÔNG DỤNG



### HÀM CHUYỂN ĐỔI KIỂU DỮ LIỆU

Tên hàm	Giá trị trả về
STR(số thực, số ký tự [,số lẻ])	Chuyển đổi kiểu dữ liệu số sang kiểu dữ liệu chuỗi.  <code>PRINT STR(12.5, 7, 2)</code>  <code>→ '__ 12.50'</code>

# BÀI TẬP 3.1



```
1. Declare @a int, @b float, @c int
2. Set a = 2
3. Select b = 2.4
4. set c = 2.5
5. print 'a=' + @a + 'b=' + @b + ' c=' + @c
6. If @a > @b
7. select @tmp = @b
8. if @b > @c
9. set @tmp = @c
10. if @c > @a
11. set @tmp = @a
12. print 'Số nhỏ nhất là: ' + @tmp
```

**Tìm lỗi**

## BÀI TẬP 3.2



**Cho CSDL:**

**SinhVien**(MaSV, HoTen, NgaySinh)

**DiemThi**(MaSV, MaMH, Diem)

Tính điểm trung bình của mỗi sinh viên. Nếu điểm trung bình lớn hơn 5.0 thì “Đậu”, ngược lại “Trượt”. In dưới dạng bảng sau:

**Ví dụ:**

MaSV	HoTen	Điểm TB	Kết quả
0912033	Nguyễn Kim Trọng	4.5	Trượt

## BÀI TẬP 3.2



```
select MaSV, HoTen,  
       AVG(Diem) as DiemTB,  
       case  
         when AVG(Diem) > 5.0 then N'Đầu'  
         else N'Trượt'  
       end  
from SinhVien hv, KetQua kq  
where hv.MaSV = kq.MaSV  
group by MaSV, HoTen
```



# HB2\_6.9.2024



- ❖ Đã xong bài tập 3.1 và 3.2
- ❖ Tuần sau tiếp LT và BT 3.3

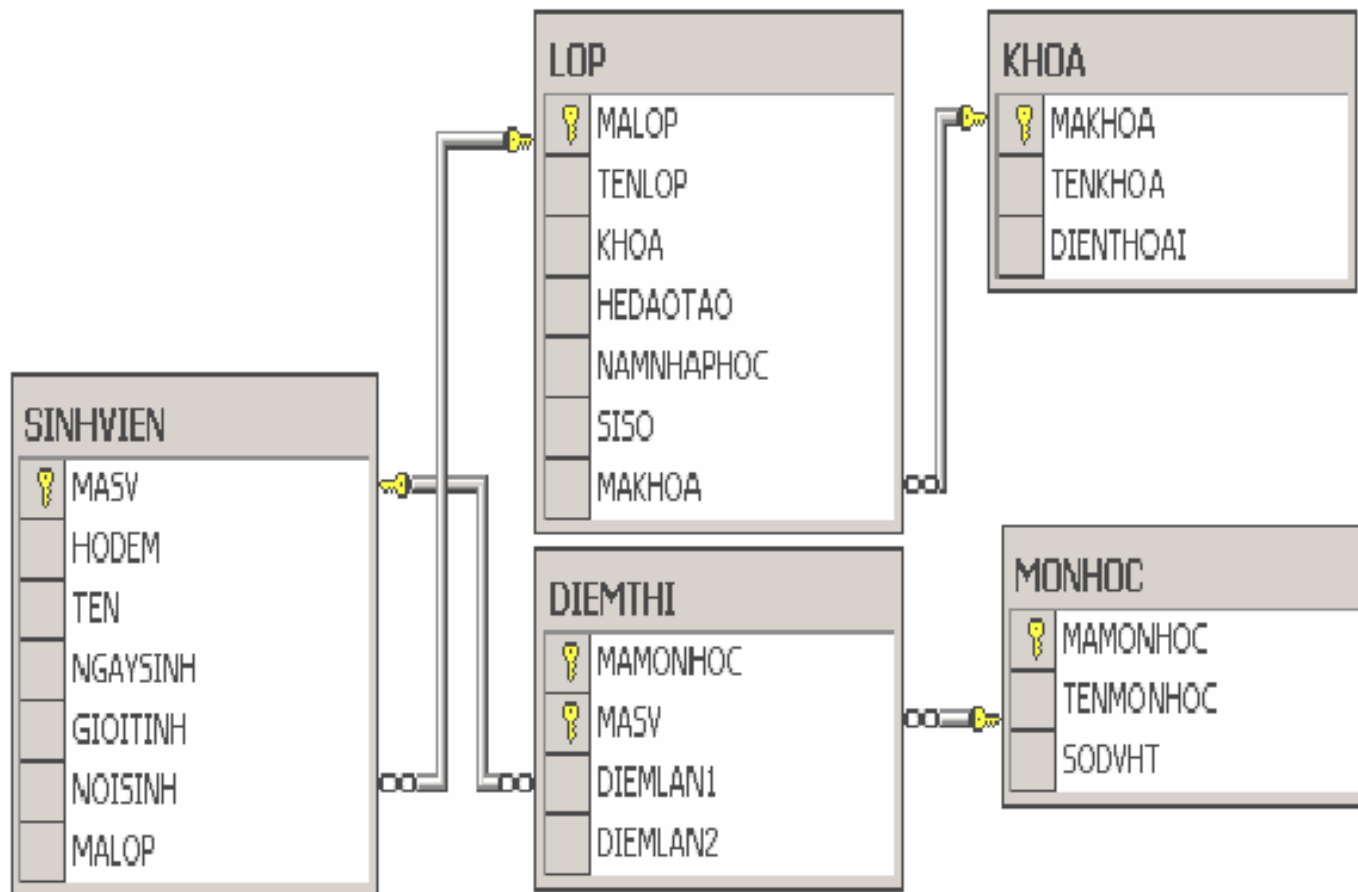


# THỦ TỤC THƯỜNG TRÚ (STORE PROCEDURE – SPs)

## 5. STORED PROCEDURE – SPs



Cho CSDL quản lý điểm thi như sau:



## 5. STORED PROCEDURE – SPs



Giả sử 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*).

## 5. STORED PROCEDURE – SPs

- 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,masv)

```
SELECT 'CST005', masv
```

```
FROM sinhvien
```

```
WHERE malop = 'CDT002'
```

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

## 5. STORED PROCEDURE – SPs



Thay vì viết 2 câu lệnh như trên, ta có thể định nghĩa 1 thủ tục với các tham số: @mamh, @tenmh, @sosvht, @malop để nhập dữ liệu cho một môn học bất kỳ do người dùng truyền vào khi gọi thủ tục.

## 5. STORED PROCEDURE – SPs



```
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
```

## 5. STORED PROCEDURE – SPs

- Công cụ rất cần thiết cho các hệ quản trị cơ sở dữ liệu
- Là “Chương trình con” của SQL SERVER
- Chứa các lệnh T-SQL...
- Có thể gọi thủ tục nội tại ở trong SQL hay trong các ứng dụng được xây dựng bằng VB.NET, C#...
- Có thể được tạo ra từ công cụ và câu lệnh.
- Có thể chứa những câu lệnh thực hiện dữ liệu (DML) hoặc những câu lệnh truy vấn dữ liệu (SELECT).



## 5. STORED PROCEDURE – SPs

### Đặc tính và những thuận lợi của SPs:

- Chấp nhận những tham số vào và trả về những giá trị được chứa trong các tham số ra.
- Người dùng có thể chỉ tạo SPs một lần, lưu trữ trong database một lần nhưng trong chương trình có thể gọi nó với số lần bất kỳ.
- SPs cho phép thực thi nhanh hơn: một đoạn source code khá lớn thực thi lặp đi lặp lại thì SPs thực hiện sẽ nhanh hơn.
- ...

## 5. STORED PROCEDURE – SPs



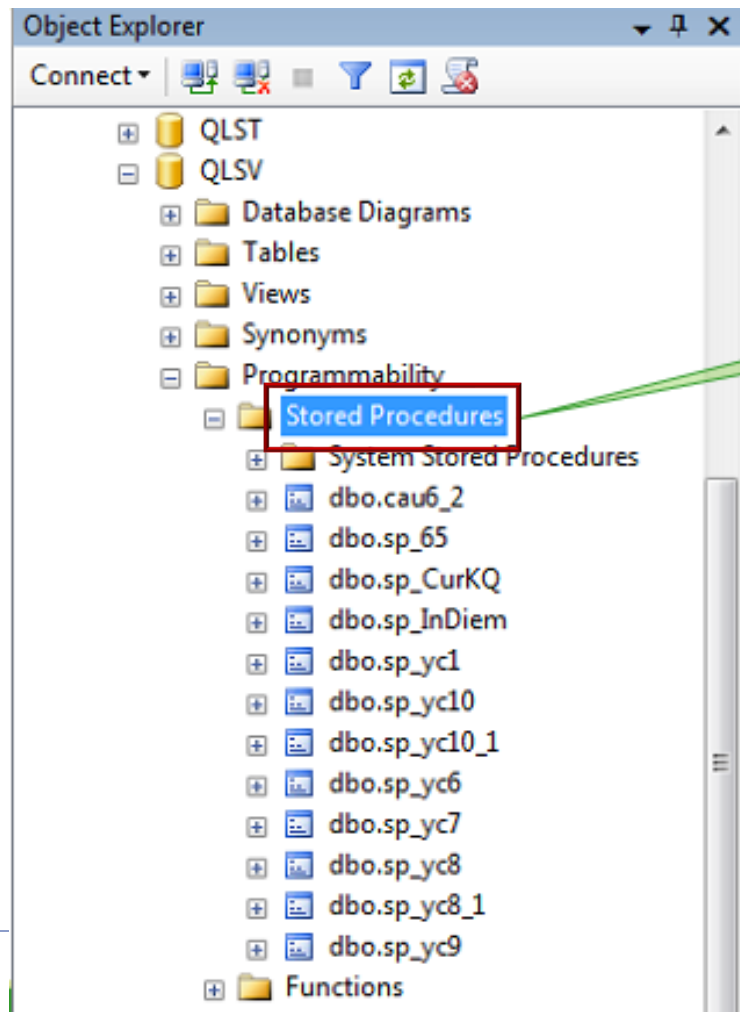
□ Có hai loại thủ tục lưu trữ:

- Thủ tục hệ thống – System Stored Procedures (thường bắt đầu bằng **sp\_**).
- Thủ tục do người sử dụng tự viết gọi là **User Stored Procedures**.

## 5. STORED PROCEDURE – SPs



- ❑ Trong SQL Server, thủ tục được lưu trữ trong thư mục như sau:



Thư mục chứa thủ tục

## 5. STORED PROCEDURE – SPs



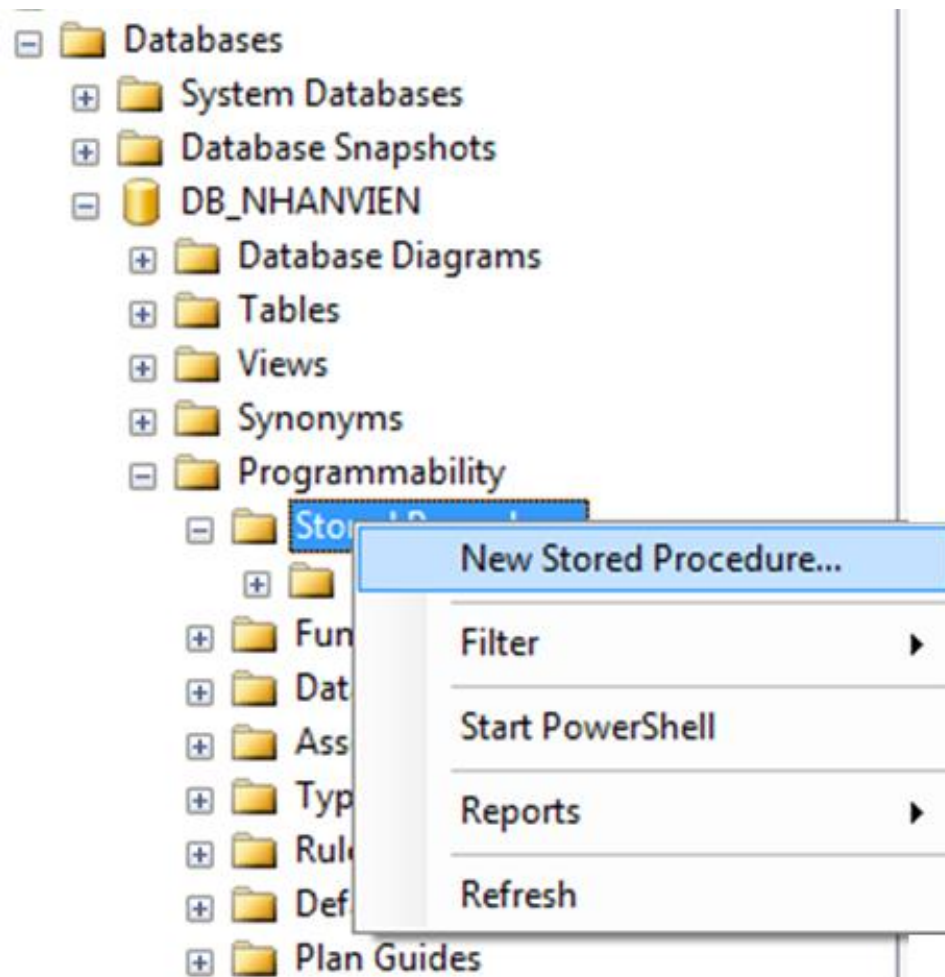
**Một SPs được định nghĩa gồm những thành phần chính sau:**

- ☐ Tên của SPs
- ☐ Các tham số
- ☐ Thân của SPs: bao gồm các câu lệnh T-SQL dùng để thực thi SPs

# 5. STORED PROCEDURE – SPs



❑ Tạo SPs bằng SQL Server Management Studio:

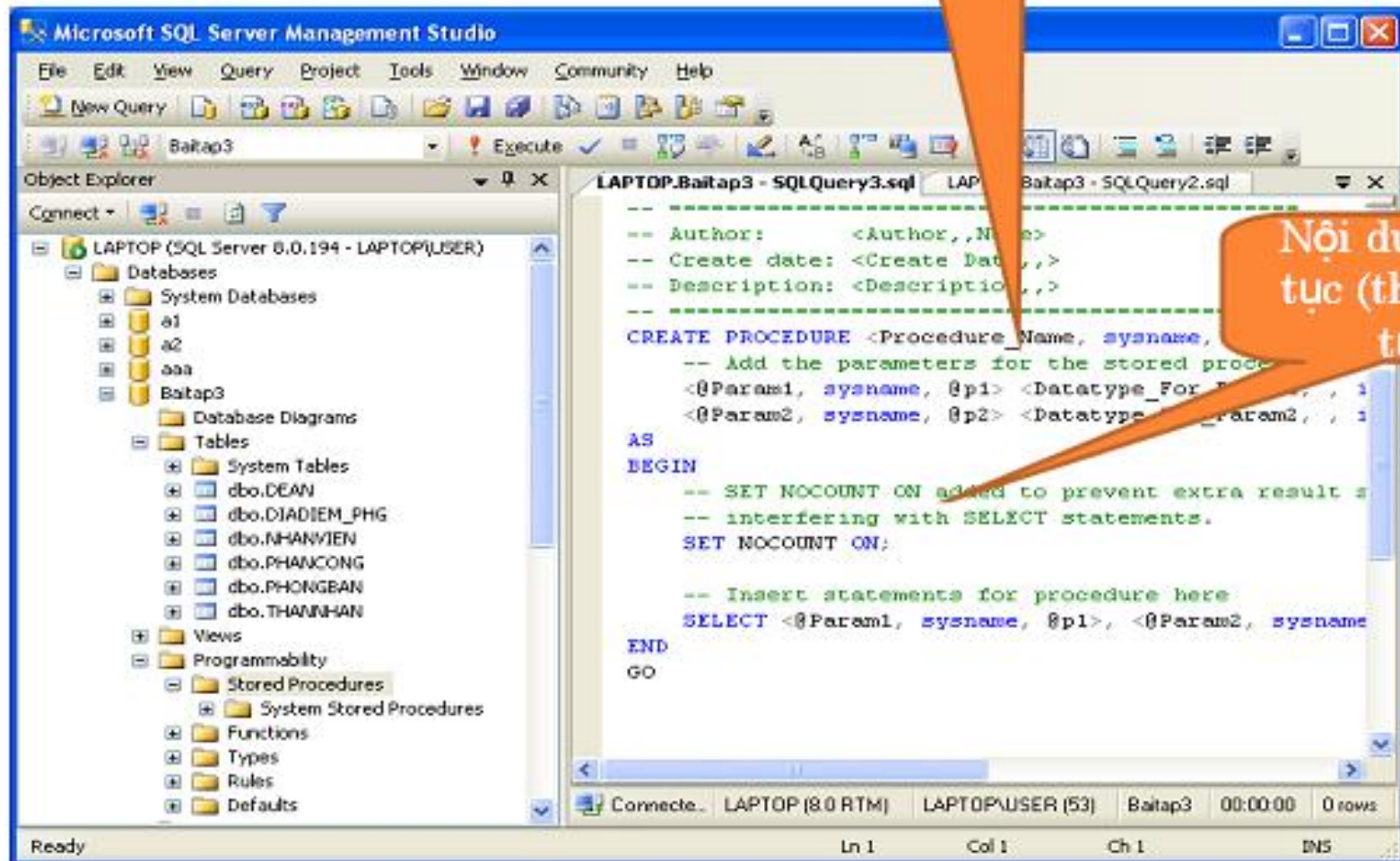


# 5. STORED PROCEDURE – SPs



Tên thủ tục

Nội dung thủ  
tục (thân thủ  
tục)



## 5. STORED PROCEDURE – SPs



### TẠO THỦ TỤC BẰNG T-SQL

```
CREATE PROC [EDURE] Tên_thủ_tục  
[ (Danh_sách_tham_số) ]  
[ WITH RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION ]  
AS  
  
[Begin]  
  
    [Declare <biến cục bộ>]  
    <Các_câu_lệnh_của_thủ_tục>  
  
[End]
```

## 5. STORED PROCEDURE – SPs



### TẠO THỦ TỤC BẰNG T-SQL

**Trong đó:**

- Tên\_thủ\_tục: Là tên thủ tục được tạo mới, phải là duy nhất trong một CSDL và đặt theo quy tắc đặt tên.
- Danh\_sách\_tham\_số: Là các tham số dùng để nhận các giá trị bên ngoài đưa vào. Các tham số này được khai báo như sau: @⟨tên\_tham\_số⟩ ⟨kiểu\_dữ\_liệu⟩ [độ\_dài].
- Số tham số tối đa trong một thủ tục là 255.



## 5. STORED PROCEDURE – SPs



### TẠO THỦ TỤC BẰNG T-SQL

**Trong đó :**

- **WITH RECOMPILE:** SQL Server sẽ biên dịch lại thủ tục lưu trữ mỗi khi được gọi.
- **WITH ENCRYPTION:** SQL Server sẽ mã hóa thủ tục lưu trữ. Khi thủ tục đã được mã hóa, không thể xem được nội dung của thủ tục.

## 5. STORED PROCEDURE – SPs



### TẠO THỦ TỤC BẰNG T-SQL

#### Lưu ý:

- Trong SQL Server, có thể ghi tắt **một số** từ khóa mà tên có chiều dài hơn 4 ký tự. Ví dụ: Create Proc
- Tên SPs, tên biến trong SQL Server không phân biệt chữ hoa chữ thường.

## 5. STORED PROCEDURE – SPs



### TẠO THỦ TỤC BẰNG T-SQL

#### Ví dụ 1:

```
CREATE PROCEDURE XINCHAO
```

```
AS
```

```
Print N'Xin chào bạn đến với Stored Procedure'
```

## 5. STORED PROCEDURE – SPs



### TẠO THỦ TỤC BẰNG T-SQL

**Ví dụ 2:** Tạo một thủ tục in ra danh sách những môn học (MAMH, TENMH) có số tiết nhiều hơn 45.

```
CREATE PROCEDURE ds_mh
```

```
AS
```

```
    SELECT  MAMH,  TENMH
```

```
    FROM    MONHOC
```

```
    WHERE   SOTIET > 45
```

## 5. STORED PROCEDURE – SPs



### TẠO THỦ TỤC BẰNG T-SQL

Lưu ý:

- CREATE PROCEDURE không chứa những câu lệnh sau: CREATE VIEW, CREATE TRIGGER, CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE.
- Số biến hệ thống và số biến cục bộ chỉ bị giới hạn bởi khả năng bộ nhớ.

## 5. STORED PROCEDURE – SPs



### TẠO THỦ TỤC BẰNG T-SQL

Lưu ý:

- Có thể có 2100 biến trong Stored Procedure.
- Kích thước tối đa cho Stored Procedure là 128 MB.

## 5. STORED PROCEDURE – SPs



### BIÊN DỊCH VÀ GỌI THỰC THI THỦ TỤC

**Biên dịch:** Chọn toàn bộ mã lệnh tạo Stored Procedure

→ Nhấn **F5**

**Thực thi thủ tục (gọi thủ tục):**

**Exec[ute]** <tên\_thủ\_tục>

**Ví dụ:**

`Exec ds_mh --Gọi thực thi thủ tục tên ds_mh`

## 5. STORED PROCEDURE – SPs



### HỦY BỎ THỦ TỤC

Cú pháp:

**Drop Proc[edure]** <tên thủ tục>

Ví dụ:

Drop Proc ds\_mh --Hủy thủ tục tên ds\_mh



## 5. STORED PROCEDURE – SPs



### THAY ĐỔI NỘI DUNG THỦ TỤC

Cú pháp:

```
Alter Proc[edure] <tên thủ tục>
```

```
As
```

```
[Begin]
```

```
    [Declare <biến cục bộ>]
```

```
    <Các lệnh>
```

```
[End]
```

## 5. STORED PROCEDURE – SPs



### THAM SỐ TRONG THỦ TỤC

- Một SPs có thể không có, có một hay nhiều tham số.
- Các tham số chỉ có nghĩa cục bộ trong SPs.
- Tên tham số duy nhất, nên đặt gọi nhớ.
- Một SPs cho phép tối đa 1024 tham số.

## 5. STORED PROCEDURE – SPs



### THAM SỐ TRONG THỦ TỤC

Có **hai** loại tham số:

- Tham số đầu vào
- Tham số đầu ra

## 5. STORED PROCEDURE – SPs



### THAM SỐ TRONG THỦ TỤC

**Tham số đầu vào:**

- Là tham số được dùng để **nhận giá trị từ người dùng** truyền vào cho thủ tục.

## 5. STORED PROCEDURE – SPs



### THAM SỐ TRONG THỦ TỤC - Tham số đầu vào:

Cú pháp:

```
Create Procedure <tên thủ tục>  
    @<tên tham số đầu vào> <kdliệu> [độ dài] [, ...]  
As  
    [Declare <biến cục bộ>]  
    <Các lệnh>
```

## 5. STORED PROCEDURE – SPs



**THAM SỐ TRONG THỦ TỤC - Tham số đầu vào:**

**Ví dụ 3.4:** Tạo thủ tục **TIMSV** in ra thông tin của sinh viên có MaSV do người dùng truyền vào.

```
CREATE PROC TIMSV @MASV CHAR(10)
```

```
AS
```

```
SELECT *
```

```
FROM SINHVIEN
```

```
WHERE MASV = @MASV
```

## 5. STORED PROCEDURE – SPs



**THAM SỐ TRONG THỦ TỤC - Tham số đầu vào:**

**Ví dụ 3.4:** Tạo thủ tục **TIMSV** in ra thông tin của sinh viên có MaSV do người dùng truyền vào

**Gọi thực hiện thủ tục:**

```
Exec TIMSV 'SV01'
```

## 5. STORED PROCEDURE – SPs



**THAM SỐ TRONG THỦ TỤC - Tham số đầu vào:**

**Ví dụ 3.5:** Tạo thủ tục INBANGDIEM in ra bảng điểm (MAMH, TENMH, DIEM) của sinh viên có MaSV do người dùng truyền vào.

```
CREATE PROC INBANGDIEM @MASV CHAR(10)
AS
    SELECT mh.MAMH, TENMH, DIEM
    FROM KETQUA kq, MONHOC mh
    WHERE kq.MAMH = mh.MAMH AND MASV = @MASV
```



## 5. STORED PROCEDURE – SPs



**THAM SỐ TRONG THỦ TỤC** - Tham số đầu vào:

**Ví dụ 3.6:** Tạo thủ tục THEMMONHOC để thêm mới một môn học.

```
CREATE PROC THEMMONHOC @MAMH CHAR(10),  
    @TENMH NVARCHAR(30), @SOTIET INT  
  
AS  
  
    INSERT INTO MONHOC  
    VALUES (@MAMH, @TENMH, @SOTIET)
```

```
EXEC THEMMONHOC 'M13', 'HE QTCSDL', 30
```

## 5. STORED PROCEDURE – SPs



### THAM SỐ TRONG THỦ TỤC

Tham số đầu ra:

- Nhận kết quả trả về từ thủ tục (giữ lại giá trị của đối số sau khi kết thúc thủ tục) và hiển thị cho người dùng.
- Được sử dụng kết hợp với từ khóa **OUTPUT** hay **OUT**.

## 5. STORED PROCEDURE – SPs



**THAM SỐ TRONG THỦ TỤC** - Tham số đầu ra:

Cú pháp:

```
Create Procedure <tên thủ tục>  
                @<tên tham số> <kdliệu> OUTPUT [...]
```

**As**

```
[Declare <biến cục bộ>]  
<Các lệnh>
```

## 5. STORED PROCEDURE – SPs



### THAM SỐ TRONG THỦ TỤC - Tham số đầu ra:

Lưu ý:

- Khi gọi thủ tục có chứa tham số đầu ra phải chỉ tường minh tham số đầu ra với từ khóa **OUTPUT** hay **OUT**.

## 5. STORED PROCEDURE – SPs



**THAM SỐ TRONG THỦ TỤC** - Tham số đầu ra:

**Ví dụ 3.7:** Tạo thủ tục trả về **điểm một môn học** của một sinh viên.

- Tham số đầu vào: Mã sinh viên, Mã môn học.
- Tham số đầu ra: **Điểm của môn học**.

## 5. STORED PROCEDURE – SPs



**THAM SỐ TRONG THỦ TỤC** - Tham số đầu ra:

**Ví dụ 3.7:** Tạo thủ tục cho biết điểm một môn học của một sinh viên.

```
CREATE PROC XEMDIEM @MASV CHAR(10),  
                @MAMH CHAR(10), @DIEM INT OUTPUT  
AS  
SET @DIEM = (SELECT DIEM FROM KETQUA  
                WHERE MASV = @MASV  
                AND MAMH = @MAMH)
```

```
SELECT @DIEM = DIEM FROM KETQUA
WHERE MASV = @MASV AND MAMH = @MAMH
```

## 5. STORED PROCEDURE – SPs



**THAM SỐ TRONG THỦ TỤC** - Tham số đầu ra:

**Ví dụ 3.7:** Tạo thủ tục cho biết **điểm một môn học** của một sinh viên.

```
-- Gọi thực thi thủ tục
```

```
DECLARE @DIEM_SV INT
```

```
EXEC XEMDIEM 'SV01', 'M001', @DIEM_SV OUTPUT
```

```
PRINT @DIEM_SV
```



## 5. STORED PROCEDURE – SPs



### THAM SỐ TRONG THỦ TỤC - Tham số đầu ra:

**Ví dụ 3.8:** Tạo thủ tục nhập vào MASV, xuất ra điểm TB của sinh viên đó và xếp loại SV theo DTB như sau:

- $DTB < 4$  : Yếu
- $4 \leq DTB < 6.5$  : Trung bình
- $6.5 \leq DTB < 8.5$  : Khá
- $DTB \geq 8.5$  : Giỏi

## 5. STORED PROCEDURE – SPs



```
CREATE PROC XEP_LOAI @MASV CHAR(10), @DTB FLOAT
    OUTPUT, @XEPLOAI NVARCHAR(30) OUTPUT
AS
    SELECT @DTB = AVG(DIEM)
    FROM KETQUA
    WHERE MASV = @MASV
    if @DTB < 4
        Set @XEPLOAI = N'Yếu'
    else if @DTB < 6.5
        Set @XEPLOAI = N'TB'
    else if @DTB < 8.5
        Set @XEPLOAI = N'Khá'
    else
        Set @XEPLOAI = N'Giỏi'
```

## 5. STORED PROCEDURE – SPs



### THAM SỐ TRONG THỦ TỤC

**Ví dụ 3.8:** Tạo thủ tục nhập vào MASV, xuất ra điểm TB của sinh viên đó và xếp loại SV theo DTB.

```
-- Gọi thực thi thủ tục  
declare @dtbsv float, @xl nvarchar(30)  
exec xep_loai 'sv05', @dtbsv output, @xl output  
print @dtbsv  
print @xl
```

LOP(MALOP, TENLOP)

SINHVIEN(MASV,HOTEN,TUOI, PHAI, MALOP)

MONHOC(MAMH,TENMH,SOTC)

KETQUA(MASV,MAMH, DIEM)

a/ Viết thủ tục truyền vào tham số mã sinh viên, trả về tên lớp của sinh viên đó

b/ Viết thủ tục truyền vào mã lớp, trả về tổng số sinh viên có trong lớp đó

c/ Viết thủ tục truyền vào mã sinh viên trả về họ tên và tổng số tín chỉ mà sinh viên đó đã học

LOP(MALOP, TENLOP)

SINHVIEN(MASV,HOTEN,TUOI, PHAI, MALOP)

MONHOC(MAMH,TENMH,SOTC)

KETQUA(MASV,MAMH, DIEM)

b/ Viết thủ tục truyền vào mã lớp, trả về tổng số sinh viên có trong lớp đó

c/ Viết thủ tục truyền vào mã sinh viên trả về họ tên và tổng số tín chỉ mà sinh viên đó đã học

LOP(MALOP, TENLOP)

SINHVIEN(MASV,HOTEN,TUOI, PHAI, MALOP)

MONHOC(MAMH,TENMH,SOTC)

KETQUA(MASV,MAMH, DIEM)

c/ Viết thủ tục truyền vào mã sinh viên trả về họ tên và tổng số tín chỉ mà sinh viên đó đã học

d/ Viết thủ tục truyền vào mã môn học, in ra danh danh sách những sinh viên có điểm  $\geq 5$

e/ Viết thủ tục truyền vào mã sinh viên và mã môn học, trả về *Đạt* nếu sinh viên có điểm  $\geq 5$ , ngược lại trả về *Không đạt*. Trường hợp sinh viên không học môn đó thì trả về *Chưa học*

## 5. STORED PROCEDURE – SPs

**Ví dụ:** (Gọi thủ tục)

*--1. Truyền trị*

**Exec** usp\_ThemDangKy '001', 'HP01'

*--2. Truyền trị có tên biến*

**Exec** usp\_ThemDangKy @MaHP = 'HP01', @MaSV = '001'

*--3. Truyền trị có tên biến*

**Exec** usp\_ThemDangKy @MaHP, @MaSV

*--4. Có output*

**Declare** @SiSo **int**

**Exec** usp\_ThemDangKy '001', 'HP01', @SiSo **output**

## 5. STORED PROCEDURE – SPs



### MỘT SỐ VẤN ĐỀ KHÁC TRONG SPs

- Mã hóa nội dung thủ tục
- Biên dịch lại thủ tục
- Thủ tục lồng nhau
- Lệnh **return** trong SPs



## 5. STORED PROCEDURE – SPs



### MÃ HÓA NỘI DUNG TRONG SPs

**Mục đích:** Không cho phép người dùng khác xem mã lệnh trong SPs.

```
Create Procedure <tên_thủ_tục>  
    @<tên_tham_số> <kdliệu> [Output] [...]
```

**With Encryption**

**As**

```
[Declare <biến_cục_bộ>]  
    <Các_lệnh>
```

## 5. STORED PROCEDURE – SPs



### BIÊN DỊCH LẠI THỦ TỤC TRONG SPs

**Mục đích:** Mỗi lần có người dùng gọi thủ tục thì bản thân nó sẽ biên dịch lại.

```
Create Proc[edure] <tên_thủ_tục>  
    @<tên_tham_số> <kdliệu> [Output] [...]
```

**With Recompile**

**As**

```
[Declare <biến_cục_bộ>]  
    <Các lệnh>
```



**Cách 1**

## 5. STORED PROCEDURE – SPs



### BIÊN DỊCH LẠI THỦ TỤC TRONG SPs

**Mục đích:** Mỗi lần có người dùng gọi thủ tục thì bản thân nó sẽ biên dịch lại.

**EXEC** <tên\_thủ\_tục> [<các tham số>] **With Recompile**

**Lưu ý:** Cách này dùng với việc tạo thủ tục không dùng tùy chọn With Recompile.

A dark blue starburst graphic with multiple points, containing the text 'Cách 2' in white.

Cách 2

## 5. STORED PROCEDURE – SPs



### THỦ TỤC LỒNG NHAU

- SQL cho phép các thủ tục lồng vào nhau (gọi lẫn nhau).
- SQL cho phép lồng tối đa 32 cấp.

## 5. STORED PROCEDURE – SPs



### THỬ TỤC LÒNG NHAU

```
Create proc A  
AS  
Begin  
    -- Các lệnh  
End
```

```
Create proc B  
AS  
Begin  
    EXEC A  
    -- Các lệnh  
End
```

## 5. STORED PROCEDURE – SPs

### LỆNH RETURN TRONG SPs

#### Cú pháp

**RETURN** [**⟨số nguyên⟩**]

#### Khi đó cách thức gọi thủ tục

**EXEC** @⟨biến⟩ = ⟨Tên\_thủ\_tục⟩[⟨các tham số⟩] [...]

**Lưu ý:** Khi gặp câu lệnh **RETURN** ngay lập tức **SPs** sẽ kết thúc.

# LỆNH RETURN TRONG SPs

--Ví dụ 3.9: Tao SPs nhập vào ten mon hoc, tra ve 1 neu co sinh vien hoc mon hoc do, nguoc lai tra ve 0.

```
Create Proc Ktra_SV_MH @tenmh varchar(30) AS
    If exists (select *
               from   KetQua kq, MonHoc mh
               where  kq.MaMH = mh.MaMH And TenMH =
@tenmh)
    Begin
        return 1
    End
Else return 0
```

# LỆNH RETURN TRONG SPs

## --Lời gọi thủ tục

```
declare @kq int, @tenmh varchar(30)
set @tenmh = 'Triet hoc'
Exec @kq = Ktra_SV_MH @tenmh
if @kq = 0
print 'Khong ton tai sinh vien nao hoc mon ' +
@tenmh
```



# LỆNH RETURN TRONG SPs

## --Ví dụ 3.10: Tạo thủ tục trả về tuổi của một sinh viên – C1

```
CREATE PROCEDURE tinhhtuoi @masv char(10),  
                           @tuoi int OUTPUT  
  
AS  
  
    SET @tuoi = (SELECT datediff(yy, ngsinh, getdate())  
                FROM sinhvien WHERE masv = @masv)
```

-----Gọi thực hiện -----

```
DECLARE @tuoisv int  
EXEC tinhhtuoi 'sv05', @tuoisv OUTPUT  
PRINT @tuoisv
```

# LỆNH RETURN TRONG SPs

## --Ví dụ 3.10: Tạo thủ tục trả về tuổi của một sinh viên – C2

```
CREATE PROCEDURE kttuoi @masv CHAR(10)
AS
    DECLARE @tuoi INT
    SET @tuoi = (SELECT datediff(yy, ngsinh, getdate())
                FROM sinhvien WHERE masv = @masv)
    RETURN @tuoi
```

-----Gọi thực hiện

```
DECLARE @tuoisv INT
EXEC @tuoisv = kttuoi 'sv05'
PRINT @tuoisv
```

# LỆNH RETURN TRONG SPs

--Ví dụ 3.11: Tạo thủ tục KTTUOI của một SV, nếu tuổi lớn hơn 30 return 1, ngược lại return 0.

```
Create Procedure kttuoi @masv char(10)
as
    declare @tuoi int
    set @tuoi = (select datediff(yy,ngsinh,getdate())
                from sinhvien where masv = @masv)
    if @tuoi > 30
        return 1
    else
        return 0
    print 'da kiem tra xong'
```

# LỆNH RETURN TRONG SPs

--Ví dụ 3.11: Tạo thủ tục KTTUOI của một SV, nếu tuổi lớn hơn 30 return 1, ngược lại return 0.

```
-----  
DECLARE    @k1    INT  
EXEC @k1 = kttuoi    'SV09'  
PRINT @k1
```

## 5. STORED PROCEDURE – SPs



### SỬ DỤNG GIÁ TRỊ MẶC ĐỊNH CHO THAM SỐ

- Trong lệnh gọi thủ tục có tham số đầu vào, nếu không truyền giá trị cho tham số tương ứng thì hệ thống sẽ báo lỗi.
- Đôi khi gọi thủ tục, chúng ta muốn bỏ qua giá trị truyền vào cho tham số. Để thực hiện điều này ta sử dụng giá trị mặc định cho tham số theo cú pháp:

@<Tên tham số> <Kiểu dữ liệu> = <Giá trị mặc định>

## 5. STORED PROCEDURE – SPs



### SỬ DỤNG GIÁ TRỊ MẶC ĐỊNH CHO THAM SỐ

```
CREATE PROC TT_SV @MASV CHAR(10)='SV01'  
AS  
    SELECT *  
    FROM SINHVIEN  
    WHERE MASV = @MASV  
GO
```

**Khi gọi thực hiện thủ tục mà không cung cấp mã sinh viên thì giá trị mặc định sẽ được sử dụng là ‘SV01’**

## 5. STORED PROCEDURE – SPs



### SỬ DỤNG GIÁ TRỊ MẶC ĐỊNH CHO THAM SỐ

**Ví dụ 3.12:** Tạo thủ tục truyền vào mã lớp sẽ trả về tổng số sinh viên trong lớp đó. Trường hợp không truyền tham số thì trả về giá trị 0.

## 5. STORED PROCEDURE – SPs



### SỬ DỤNG GIÁ TRỊ MẶC ĐỊNH CHO THAM SỐ

```
CREATE PROC TONGSV @MaLop char(10) = '0'
AS
    IF (@MaLop = '0')
        RETURN 0
    ELSE
        RETURN (SELECT COUNT(MASV)
                FROM SINHVIEN
                WHERE MALOP = @MALOP)
```



## 5. STORED PROCEDURE – SPs



### SỬ DỤNG GIÁ TRỊ MẶC ĐỊNH CHO THAM SỐ

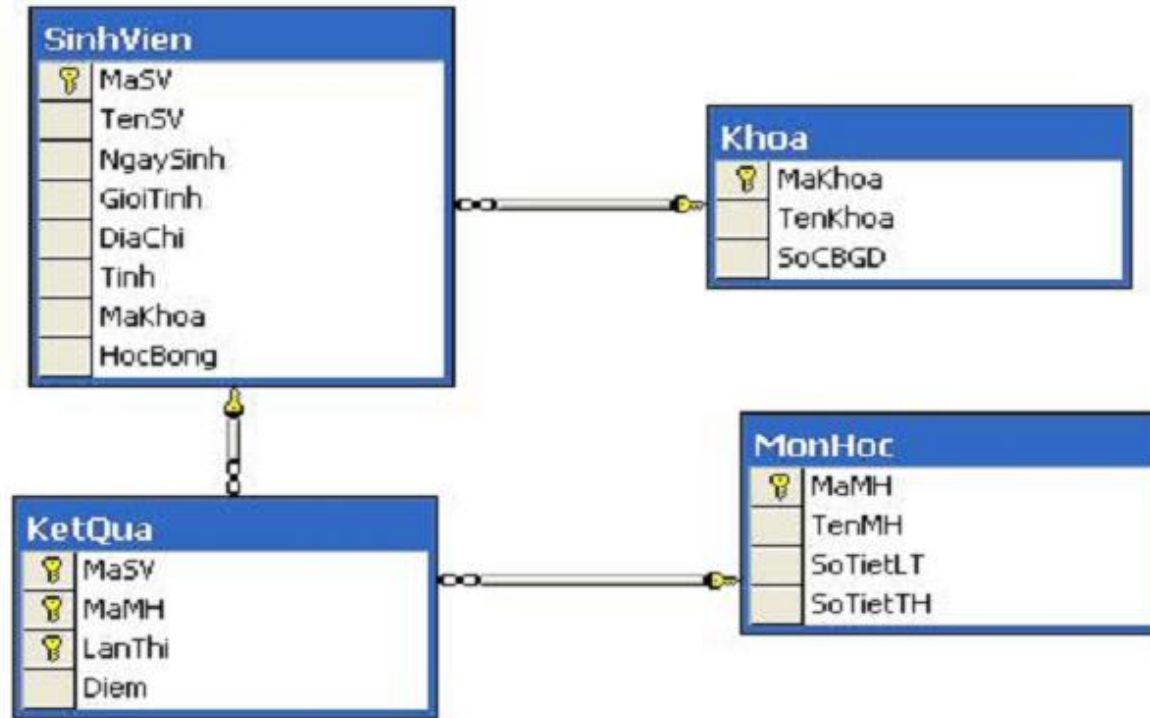
```
declare @SL int  
exec @sl = TONGSV 'L001'  
print @SL
```

```
declare @SL int  
exec @SL = TONGSV  
print @SL
```

# BÀI TẬP 3.3



Xét các bảng dữ liệu có cấu trúc như sau:



Viết SPs nâng điểm của sinh viên có **MASV** là [input], **TenMH** [input], **LanThi** [input], **So\_diem\_duoc\_nang** [input]

- ❖ Làm hết BT 3.3\_3.4 phần Store Procedure
- ❖ Tuần sau học LT Function và BT Function



# HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



- Hàm là đối tượng cơ sở dữ liệu, tương tự như thủ tục.
- Có thể sử dụng hàm như một thành phần của một biểu thức (VD: Trong danh sách chọn của lệnh **SELECT**)
- Hàm có thể do HQT CSDL cung cấp sẵn và người sử dụng có thể định nghĩa các hàm cá nhân.

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



### □ Giống Stored Procedure

- Là mã lệnh có thể tái sử dụng.
- Chấp nhận các tham số input.
- Dịch một lần và sau đó có thể gọi khi cần.

### □ Khác Stored Procedure

- Chấp nhận **nhiều kiểu dữ liệu** giá trị trả về thông qua tên hàm.
- **Không chấp nhận tham số output.**
- Khác về cách gọi thực hiện.

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



- Hàm gồm 2 loại:
  - Giá trị trả về là kiểu dữ liệu cơ sở (int, varchar, float, datetime, ...) → **Scalar Value Function.**
  - Giá trị trả về là **Table** có được từ một câu truy vấn → **Table value Function.**

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Programmability' folder is expanded, and the 'Functions' folder is selected, with 'dbo.XEM\_TEN' highlighted. A red circle is drawn around the 'Functions' folder and its sub-items. The right pane shows the SQL script for creating the function:

```
CREATE FUNCTION XEM_TEN()  
RETURNS VARCHAR(20)  
AS  
BEGIN  
    DECLARE @Hoten varchar(20)  
    SET @Hoten=(Select HOTEN  
                From SINHVIEN  
                Where MASV='3001100218')  
    RETURN @Hoten  
END
```

At the bottom, the 'Messages' pane shows the status: 'Command(s) completed successfully.'



## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



**ĐỊNH NGHĨA HÀM** (hàm vô hướng – Scalar Value Function – Trả về duy nhất một giá trị)

**CREATE FUNCTION** <ten\_ham> ([ds\_tham\_so])

**RETURNS** <kieu\_du\_lieu\_tra\_ve>

**AS**

**BEGIN**

<cau\_lenh>

**END**

Dù thân function chỉ có 1 lệnh cũng phải đặt giữa Begin và End

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



### Ví dụ 3.13: Tìm số lớn nhất trong 3 số a, b, c

```
Create function UF_SoLonNhat (@a int, @b int, @c int)
Returns int
As
Begin
    Declare @max int
    Set @max = @a
    If @b > max set @max = @b
    If @c > max set @max = @c
    Return @max
End
```

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



**Ví dụ 3.14:** Viết hàm nhận vào giá trị một ngày và trả về một chuỗi có giá trị là thứ của ngày đó trong tuần.

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



```
CREATE FUNCTION thu (@ngay DATE)
RETURNS nvarchar(10)
AS
BEGIN
    DECLARE @st nvarchar(10)
    SET @st = CASE DATEPART(dw, @ngay)
                WHEN 1 THEN 'Chu nhật'
                WHEN 2 THEN 'Thu hai'
                ...
    END
    RETURN @st
END
```

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



### GỌI HÀM

```
DECLARE @st NVARCHAR(10)
SET @st = dbo.thu (GETDATE())
PRINT @st
```

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



**HÀM** với giá trị trả về là dữ liệu kiểu **BẢNG**

**Cú pháp 1:**

```
CREATE FUNCTION <ten_ham>([ds_tham_so])
```

```
RETURNS TABLE
```

```
AS
```

```
    RETURN    <cau_lenh_SELECT>
```

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



**HÀM** với giá trị trả về là dữ liệu kiểu **BẢNG**

**Lưu ý:**

- 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, không đặt trong cặp **Begin .. End**).

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



**GỌI HÀM** với giá trị trả về là dữ liệu kiểu **BẢNG**

```
SELECT ten_cot FROM dbo.Ten_ham(doi_so)
```



## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



Ví dụ:

```
CREATE FUNCTION NOITUYEN (@TUOI INT)
```

```
RETURNS TABLE
```

```
AS
```

```
    RETURN (SELECT * FROM SINHVIEN
```

```
            WHERE TUOI > @TUOI)
```

```
-- Gọi hàm
```

```
    SELECT * FROM dbo.NOITUYEN(23)
```

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



**HÀM** với giá trị trả về là dữ liệu kiểu **BẢNG**

Khi cần sử dụng nhiều câu lệnh (khác câu lệnh **SELECT**) trong phần thân hàm, sử dụng cú pháp 2 như sau:

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



### Cú pháp 2:

```
CREATE FUNCTION <ten_ham> ([ds_tham_so])  
RETURNS @Ten_bang_tra_ve TABLE  
        (Ten_cot Kieu_du_lieu[,...])  
AS  
BEGIN  
    <Cau_lenh>  
    Insert into @Ten_bang_tra_ve  
    RETURN  
END
```

## 6. HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA



### Lưu ý:

- Cấu trúc bảng trả về bởi hàm được xác định dựa vào định nghĩa của bảng trong mệnh đề **RETURNS**.
- Biến **@Ten\_bang\_tra\_ve** 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 **@Ten\_bang\_tra\_ve**

## Ví dụ:



```
CREATE FUNCTION Func_TongSV (@lop SMALLINT)
RETURNS @bangthongke TABLE (
    Malop NVARCHAR(5),
    Tenlop NVARCHAR(50),
    Tongsosv INT
)

AS
BEGIN
    IF @lop = 0
        INSERT INTO @bangthongke
        SELECT lop.Malop, Tenlop, Count(Masv)
        FROM Lop, Sinhvien
        WHERE Sinhvien.Malop = Lop.Malop
        GROUP BY lop.Malop, Tenlop
```

## Ví dụ:

```
CREATE FUNCTION Func_TongSV(@lop SMALLINT)
RETURNS @bangthongke TABLE (
    Malop NVARCHAR(5),
    Tenlop NVARCHAR(50),
    Tongso sv INT
)

AS
BEGIN
    ELSE
        INSERT INTO @bangthongke
        SELECT lop.Malop, Tenlop, Count(Masv)
        FROM Lop, Sinhvien
        WHERE Sinhvien.Malop = Lop.Malop and
            lop.malop = @lop
        GROUP BY lop.Malop, Tenlop

    RETURN
END
```

## Ví dụ:

Câu lệnh:

```
SELECT * FROM dbo.func_TongSV(2)
```

→ Kết quả: thống kê tổng số sinh viên lớp có mã là 2.

Còn câu lệnh:

```
SELECT * FROM dbo.func_TongSV(0)
```

→ Cho biết tổng số sinh viên tất cả các lớp.

## BÀI TẬP 3.4



**Cho CSDL:**

**SinhVien**(MaSV, HoTen, NgaySinh)

**DiemThi**(MaSV, MaMH, Diem)

1. Viết hàm trả về điểm trung bình của một sinh viên.
2. Viết hàm trả về mã SV có điểm trung bình cao nhất.
3. Viết hàm xuất danh sách các sinh viên có điểm trung bình bé hơn 5 (Thông tin gồm MASV, HOTEN)



LOP(MALOP, TENLOP)

SINHVIEN(MASV,HOTEN,TUOI, PHAI, MALOP)

MONHOC(MAMH,TENMH,SOTC)

KETQUA(MASV,MAMH, DIEM)

- 1/ Viết hàm truyền vào mã sinh viên, trả về tên lớp của sinh viên đó.
- 2/ Viết hàm truyền vào tham số mã sinh viên và mã môn học trả về điểm có sinh viên học môn đó.
- 3/ Viết hàm truyền vào mã sinh viên trả về tổng số tín chỉ tích lũy của sinh viên đó. Biết rằng chỉ tích lũy đối với môn học có điểm  $\geq 5$

4/ Viết hàm truyền vào tham số mã môn học, trả về bảng chứa thông tin những sinh viên học môn đó gồm (Mã sinh viên, họ tên, điểm, tên lớp)

5/ Viết hàm truyền vào mã lớp, trả về danh sách những sinh viên học lớp đó gồm: mã sinh viên, họ tên, điểm trung bình.

LOP(MALOP, TENLOP, SISO)

SINHVIEN(MASV, HOTEN, NGSINH, GIOITINH, QUEQUAN, MALOP, DIEMTB, XEPLOAI)

MONHOC(MAMH, TENMH, SOTC, BATBUOC)

KETQUA(MASV, MAMH, HOCKY, DIEMTHI)



- e) Viết hàm truyền vào tham số mã sinh viên và học kỳ, trả về tổng số tín chỉ đã đạt của sinh viên đó, biết rằng sinh viên đạt một môn nếu điểm thi  $\geq 5$ .
- f) Viết hàm truyền vào tham số mã lớp, trả về bảng chứa danh sách những sinh viên (mã sv, họ tên, ngày sinh) học lớp đó.
- g) Viết hàm truyền vào tham số mã môn học và học kỳ, trả về bảng chứa danh sách những sinh viên (mã sv, họ tên, ngày sinh, tên lớp) có điểm  $< 5$ .
- h) Viết hàm truyền vào tham số mã môn học, trả về bảng chứa danh sách những sinh viên (mã sv, họ tên, ngày sinh) chưa học môn đó.
- i) Viết hàm truyền vào tham số mã sinh viên, trả về bảng chứa danh sách những môn học (mã môn, tên môn, điểm cao nhất) mà sinh viên học. Biết rằng trường hợp sinh viên học một môn học nhiều lần thì chỉ hiển thị số điểm cao nhất.

## BÀI TẬP 3.4



1. Viết hàm tính điểm trung bình của một sinh viên.

```
CREATE FUNCTION FUNC_DTB (@MASV CHAR(10))  
RETURNS NUMERIC(5,1)  
AS  
    BEGIN  
        DECLARE @DTB NUMERIC(5,1)  
        SET @DTB = (SELECT AVG(DIEM)  
                    FROM DIEMTHI  
                    WHERE MASV = @MASV)  
        RETURN @DTB  
    END  
GO;
```

## BÀI TẬP 3.4



1. Viết hàm tính điểm trung bình của một sinh viên.

--GỌI HÀM

```
DECLARE @DTB NUMERIC(5,1)
```

```
SET @DTB = dbo.FUNC_DT_B('SV03')
```

```
PRINT @DTB
```

```
GO;
```

## BÀI TẬP 3.4



2. Viết hàm tìm mã của sinh viên có điểm trung bình cao nhất.

```
CREATE FUNCTION FUNC_TIMSV()  
RETURNS CHAR(10)  
AS  
    BEGIN  
        DECLARE @MASV CHAR(10)  
        SET @MASV = (SELECT MASV  
                     FROM DIEMTHI  
                     GROUP BY MASV  
                     HAVING AVG(DIEM) >= ALL(SELECT AVG(DIEM)  
                                             FROM DIEMTHI  
                                             GROUP BY MASV))  
        RETURN @MASV  
    END  
GO;
```

## BÀI TẬP 3.4



2. Viết hàm tìm mã của sinh viên có điểm trung bình cao nhất.

```
--GỌI HÀM  
DECLARE @MASV CHAR(10)  
SET @MASV = dbo.FUNC_TIMSV()  
PRINT @MASV
```

## BÀI TẬP 3.4



3. Viết hàm xuất danh sách các sinh viên có điểm trung bình bé hơn 5 (Thông tin gồm MASV, HOTEN)

```
CREATE FUNCTION FUNC_DSSVDUOITB()  
RETURNS TABLE  
AS  
    RETURN (SELECT SINHVIEN.MASV, HOTEN  
              FROM SINHVIEN, DIEMTHI  
              WHERE SINHVIEN.MASV = DIEMTHI.MASV  
              GROUP BY SINHVIEN.MASV, HOTEN  
              HAVING AVG(DIEM) <5)  
GO;
```



## BÀI TẬP 3.4



3. Viết hàm xuất danh sách các sinh viên có điểm trung bình bé hơn 5 (Thông tin gồm MASV, HOTEN)

```
--GỌI HÀM  
SELECT * FROM DBO.FUNC_DSSVDUOITB()  
GO;
```

## BÀI TẬP 3.4



3. Viết hàm xuất danh sách các sinh viên có điểm trung bình bé hơn 5 (Thông tin gồm MASV, HOTEN)

```
CREATE FUNCTION FUNC_DSSVDUOITB2 ()
RETURNS @T_KQ TABLE (
    MASV CHAR(10),
    TENSX NVARCHAR(50))
AS
BEGIN
    INSERT INTO @T_KQ
        SELECT SINHVIEN.MASV, HOTEN
        FROM SINHVIEN, DIEMTHI
        WHERE SINHVIEN.MASV = DIEMTHI.MASV
        GROUP BY SINHVIEN.MASV, HOTEN
        HAVING AVG(DIEM) <5
    RETURN
END
GO;
```



A blurred, grayscale background image showing a person in a dark suit walking up a modern staircase with glass railings. The image is out of focus, creating a sense of motion and depth.

# TRIGGER

## 7. TRIGGER



- **Trigger** là một kiểu thủ tục được lưu đặc biệt, chúng được tự động gọi khi sửa đổi dữ liệu mà trigger được thiết kế để bảo vệ.
- **Trigger** được thiết kế giúp đảm bảo sự toàn vẹn dữ liệu bằng cách ngăn không cho những thay đổi không nhất quán được thực hiện
- **Ví dụ:** Mỗi sinh viên chỉ được đăng ký không quá 20TC/1 học kỳ.

## Mục tiêu

- ✓ Cài đặt các RBTV phức tạp
- ✓ Các quy tắc nghiệp vụ

## Đặc điểm

- ✓ Là một thủ tục đặc biệt
- ✓ Không có tham số
- ✓ Thuộc duy nhất một bảng

## Hoạt động

- ✓ Được kích hoạt tự động thông qua các thao tác (**I**nsert, **U**ppdate, **D**elete, ...)
- ✓ Dựa trên bảng tạm: **Inserted, Deleted.**

# 7. TRIGGER



## CÁC BẢNG TRUNG GIAN

### ■ Inserted

- Chứa dữ liệu được thêm mới trong hành động **Insert/Update**.
- Cấu trúc bảng giống với bảng thực sự được cập nhật dữ liệu

### ■ Deleted

- Chứa dữ liệu bị xóa trong hành động **Delete/Update**.
- Cấu trúc bảng giống với bảng thực sự được cập nhật dữ liệu.

## 7. TRIGGER



### CÁC BẢNG TRUNG GIAN

- **Hành động Update trong SQL Server**
  - Xóa dòng dữ liệu cũ
  - Thêm dòng dữ liệu mới với thông tin đã cập nhật.



## 7. TRIGGER



### CƠ CHẾ HOẠT ĐỘNG CỦA TRIGGER

- Trigger tương ứng với hành động Insert, phát sinh bảng **Inserted**.
- Trigger tương ứng với hành động Delete, phát sinh bảng **Deleted**.
- Trigger tương ứng với hành động Update, phát sinh bảng **Inserted và Deleted**.

## 7. TRIGGER



### Một số gợi ý trước khi tạo trigger

- Xác định tên của trigger.
- Chỉ định table hoặc view gắn trigger.
- Chỉ định biến cố Insert, Update, Delete.
- Các câu lệnh tương ứng với nhiệm vụ mà trigger sẽ thực hiện.

# 7. TRIGGER



## TẠO MỚI

### Cú pháp:

```
CREATE TRIGGER <Ten_Trigger> ON <Ten_Bang|Ten_View>  
FOR|AFTER|InStead of <INSERT [, UPDATE, DELETE]>  
AS
```

```
[DECLARE bien_cuc_bo]  
  
<Cau_lenh>
```

## 7. TRIGGER



### XÓA

Cú pháp:

```
DROP    TRIGGER    <Ten_Trigger>
```

## 7. TRIGGER



### Lưu ý:

- Các lệnh sau **không được dùng** trong Trigger:
  - Alter
  - Create
  - Drop
  - Restore, ...

## 7. TRIGGER



### **AFTER:**

- Trigger được thực thi sau khi các thao tác insert/update/delete đã thực hiện thành công.
- Không thể định nghĩa AFTER Trigger cho View

## 7. TRIGGER



### **Insert Trigger:**

Khi một hành động thêm (INSERT) dữ liệu vào bảng xảy ra thì Insert trigger trên bảng này sẽ được kích hoạt.

**Ví dụ 3.15:** Tạo trigger kiểm tra khi nhập dữ liệu vào bảng

**SACH (MASACH , TENSACH , NAMXB , NHAXB , . . . )** thì NAMXB phải lớn hơn 1990.

## 7. TRIGGER



### Insert Trigger:

```
CREATE TRIGGER KT_NamXB ON SACH
FOR INSERT
AS
    IF (SELECT NamXB FROM Inserted) > 1990
        Commit Tran      -- cho phép thêm vào
    ELSE
    BEGIN
        Print 'Nam XB khong hop le'
        Rollback Tran    -- không cho thêm vào
    END
```



## 7. TRIGGER



### Delete Trigger:

Khi xoá - DELETE dữ liệu trên bảng thì Delete Trigger trên bảng đó sẽ được kích hoạt.

**Ví dụ 3.16:** Giả sử có một bảng dữ liệu có tên là **GIAODICH** lưu trữ thông tin khách hàng giao dịch với ngân hàng như sau:

MAKH	TENKH	NGAYGD
K0034	Tran Thanh	12/02/2013
K0036	Nguyen Thu Trang	15/04/2013
K0015	Le Van Khanh	23/05/2013

## 7. TRIGGER



### Delete Trigger:

**Ví dụ 3.16:** Viết Trigger kiểm tra chỉ cho xóa những khách hàng có **thời gian giao dịch trước tháng 4 năm 2013.**

**CREATE TRIGGER** Xoa\_KH **ON** GIAODICH  
**FOR DELETE**  
**AS**

```

Declare @thang int, @nam int
Set @thang = (Select month(NGAYGD)
              From Deleted)
Set @nam = (Select year(NGAYGD) From Deleted)
if(@nam < 2013)
    Commit tran
else if (@nam = 2013 && @thang <4)
    Commit tran
else
    Rollback tran

```

## 7. TRIGGER



### **Update Trigger:**

Khi sửa đổi (UPDATE) dữ liệu trên một bảng thì Update Trigger trên bảng đó sẽ được kích hoạt.

**Hành động sửa dữ liệu tương ứng với hai hành động: xoá dữ liệu cũ sau đó thêm dữ liệu mới.**

## 7. TRIGGER



### Update Trigger:

Ví dụ 3.17: Xét bảng **HANG** như sau

MAHG	TENHG	DONGIA
A001	X	50000
A002	Y	100000
A003	Z	60000

Viết trigger chỉ cho phép cập nhật đơn giá mới thêm không quá 10% so với đơn giá cũ.

## 7. TRIGGER



### Update Trigger:

```
CREATE TRIGGER KT_dongia ON HANG
FOR UPDATE
AS
    IF (SELECT DONGIA
        From Inserted) <= 1.1 * (SELECT DONGIA
                                FROM Deleted)
        Commit tran
    ELSE
    BEGIN
        Print 'Don gia khong hop le'
        Rollback tran
    END
```

## 7. TRIGGER



### Trigger nhiều hành động:

Một trigger khi được tạo có thể khai báo cho nhiều hành động khác nhau: Insert, Delete, Update thay vì phải viết 3 trigger tương ứng với mỗi hành động.

## 7. TRIGGER



### Trigger nhiều hành động:

```
CREATE TRIGGER KT_SOTIET
ON MONHOC
FOR INSERT, UPDATE
AS
    IF (SELECT SOTC FROM INSERTED) < 0
        ROLLBACK TRAN
GO
```

Trigger này sẽ được kích hoạt với một trong 2 hành động là insert và update



## 7. TRIGGER



### **INSTEAD OF Trigger:**

- Đoạn lệnh trong Instead of trigger được thực thi **thay cho thao tác insert/update/delete tương ứng**. Có nghĩa là nó được kích hoạt trước khi xảy ra sự thay đổi trong cơ sở dữ liệu.
- Các ràng buộc không được kiểm tra trước khi trigger kích hoạt.
- Các bảng tạm **inserted** và **deleted** vẫn được tạo ra.
- **Thường được dùng để xử lý cập nhật trên View**

## 7. TRIGGER



**INSTEAD OF trigger:** thường dùng cho View với các chức năng:

- Cập nhật nhiều bảng cùng một lúc trong 1 View
- Tăng điều kiện ràng buộc trên các thuộc tính so với CHECK

# 7. TRIGGER

## INSTEAD OF trigger

VD1:

ACER\LAMMI_SQLS...thu - dbo.SV_LOP		ACER\LAMMI_SQLS...R.thu - dbo.LOP	
	MALOP	TENLOP	SOSO
*	NULL	NULL	NULL

ACER\LAMMI_SQLSERVER.thu - dbo.SV		ACER\LAMMI_SQLS...R.thu - dbo.LOP		SQLQuery
	MASV	TENSV	DIACHI	MALOP
*	NULL	NULL	NULL	NULL

```
INSERT INTO SV
VALUES ('SV001', 'A', 'TN', 'L101')
```

Messages

Msg 547, Level 16, State 0, Line 1  
The INSERT statement conflicted with the FOREIGN KEY constraint "FK\_SV\_LOP".  
The conflict occurred in database "thu", table "dbo.LOP", column 'MALOP'.  
The statement has been terminated.

# 7. TRIGGER



## INSTEAD OF trigger

**VD1:**

```
CREATE TRIGGER VD1 ON SINHVIEN
INSTEAD OF INSERT
AS
    DECLARE @MALOP CHAR(10)
    SET @MALOP = (SELECT MALOP FROM INSERTED)
    IF NOT EXISTS (SELECT * FROM LOP
                   WHERE MALOP = @MALOP)
        INSERT INTO LOP
        VALUES (@MALOP, 'CHUA BIET', 0)
GO

-----

INSERT INTO SINHVIEN
VALUES ('SV001', 'A', 'TN', 'L101')
```

## 7. TRIGGER



### INSTEAD OF trigger

**VD2:** View SV\_LOP lấy thông tin từ 2 bảng SINHVIEN và LOP

```
CREATE VIEW SV_LOP
AS
    SELECT MASV, TENSX, S.MALOP, TENLOP
    FROM SINHVIEN S, LOP L
    WHERE S.Malop = L.MALOP
GO
```

## 7. TRIGGER

### INSTEAD OF trigger

**VD2:** Muốn thêm 1 dòng vào view SV\_LOP, sử dụng câu lệnh sau:

```
INSERT INTO SV_LOP  
VALUES ('SV11', 'NGUYEN VAN A', 'L100', '12CDTH1')
```

### LỖI

View or function 'SV\_LOP' is not updatable because the modification affects multiple base tables.

## 7. TRIGGER



```
CREATE TRIGGER VD2 ON SV_LOP INSTEAD OF INSERT
AS
IF NOT EXISTS (SELECT * FROM LOP
                WHERE MALOP = (SELECT MALOP FROM INSERTED))
AND NOT EXISTS (SELECT * FROM SINHVIEN
                WHERE MASV = (SELECT MASV FROM INSERTED))
BEGIN
    INSERT INTO LOP (MALOP, TENLOP)
        SELECT MALOP, TENLOP FROM INSERTED
    INSERT INTO SV (MASV, TENSX, MALOP)
        SELECT MASV, TENSX, MALOP FROM INSERTED
END
ELSE
    PRINT N'SINH VIÊN HOẶC LỚP ĐÃ TỒN TẠI'
GO
```

# 7. TRIGGER



## INSTEAD OF trigger

ACER\LAMMI_SQLS...thu - dbo.SV_LOP		ACER\LAMMI_SQLS...R.thu - dbo.LOP	
	MALOP	TENLOP	SOSO
*	NULL	NULL	NULL

ACER\LAMMI_SQLSERVER.thu - dbo.SV		ACER\LAMMI_SQLS...R.thu - dbo.LOP		SQLQuery
	MASV	TENSV	DIACHI	MALOP
I_SQLSERVER.thu - dbo.SV	NULL	NULL	NULL	NULL

```
INSERT INTO SV_LOP
VALUES ( 'SV11' , 'ABC' , 'L100' , '12CDTH1' )
```



# 7. TRIGGER



## INSTEAD OF trigger

ACER\LAMMI_SQLSERVER.thu - dbo.SV		ACER\LAMMI_SQLS...R.thu - dbo.LOP		SQLQuery
	MASV	TENSV	DIACHI	MALOP
▶	SV11	ABC	<i>NULL</i>	L100
*	<i>NULL</i>	<i>NULL</i>	<i>NULL</i>	<i>NULL</i>

ACER\LAMMI_SQLSERVER.thu - dbo.SV		ACER\LAMMI_SQLS...R.thu - dbo.LOP	
	MALOP	TENLOP	SOSO
▶	L100	12CDTH1	<i>NULL</i>
*	<i>NULL</i>	<i>NULL</i>	<i>NULL</i>

# 7. TRIGGER



## Sử dụng **IF UPDATE** trong Trigger

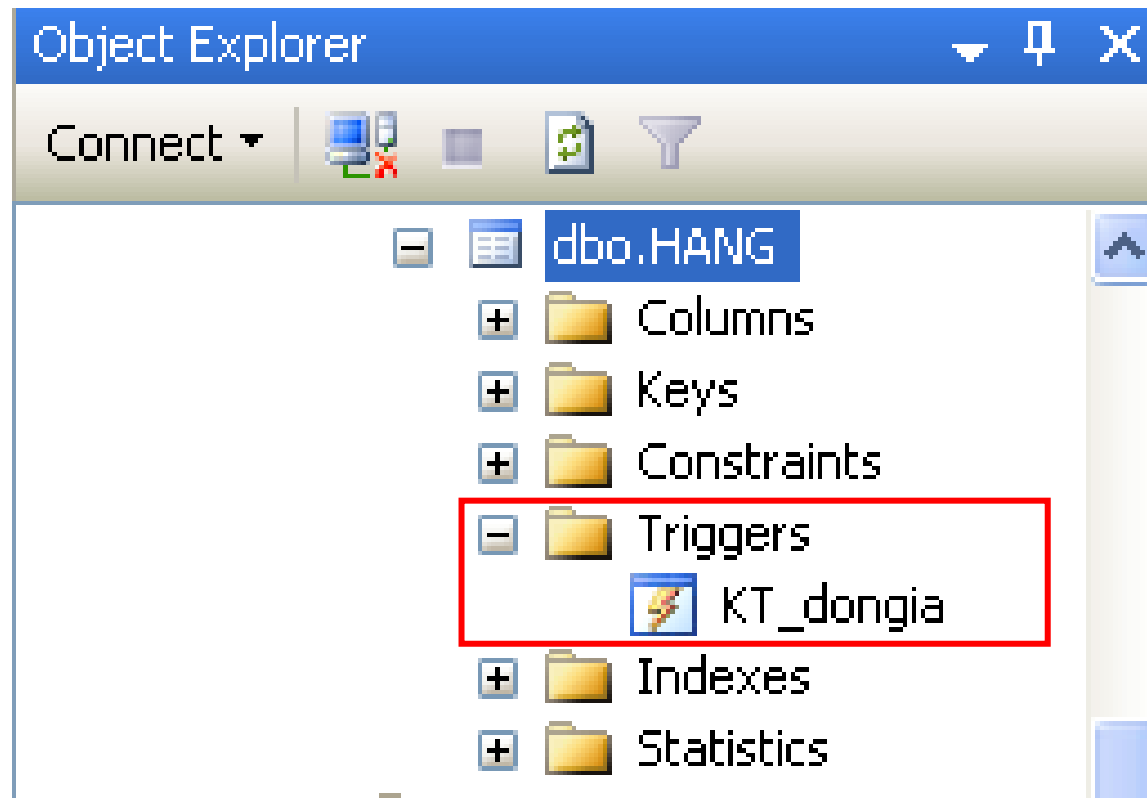
```
CREATE TRIGGER SISO_UPDATE ON SINHVIEN FOR
UPDATE
AS
    IF UPDATE (MALOP)
    BEGIN
        UPDATE LOP
        SET SISO = SISO + 1
        WHERE MALOP = (SELECT MALOP FROM INSERTED)

        UPDATE LOP
        SET SISO = SISO - 1
        WHERE MALOP = (SELECT MALOP FROM DELETED)
    END
```

# 7. TRIGGER



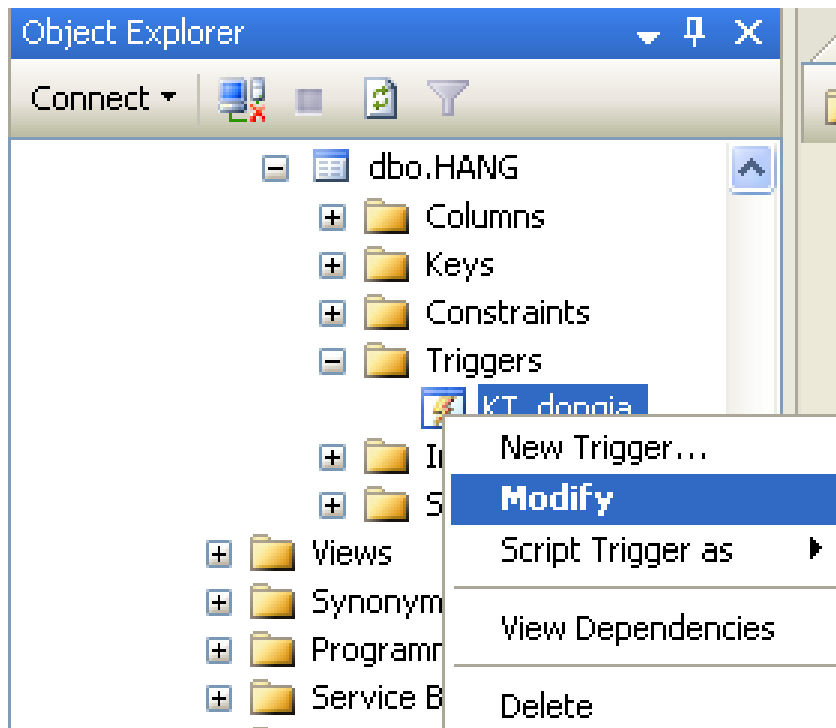
## QUẢN LÝ TRIGGER



# 7. TRIGGER



## XEM NỘI DUNG TRIGGER



`sp_helptrigger <tên_trigger>`

## 7. TRIGGER

### **BÀI TẬP 3.5:** Ràng buộc liên bộ

Xét LĐQH

**KETQUA** (MaSV, MaMH, LanThi, Diem)

**Phát biểu:** “Sinh viên chỉ được thi tối đa 2 lần cho một môn học”

**Bảng tầm ảnh hưởng:**

	Insert	Delete	Update
KetQua	+	-	+ (MASV, MAMH)

## 7. TRIGGER



```
CREATE TRIGGER KTRA_DKY_SV ON KETQUA
FOR INSERT, UPDATE
AS
    DECLARE @SOLT INT
    SET @SOLT = (SELECT COUNT (*)
                  FROM KETQUA K, INSERTED
                  WHERE K.MASV = I.MASV
                       AND K.MAMH =
I.MAMH)
    IF @SOLT > 2
        ROLLBACK TRAN
```

## 7. TRIGGER



**BÀI TẬP 3.6:** Ràng buộc liên thuộc tính – liên quan hệ  
Cho CSDL

**DATHANG** (MaPDH, NgayDH, ...)

**GIAOHANG** (MaPGH, MaPDH, NgayGH, ...)

**Phát biểu:** “Ngày giao hàng không thể nhỏ hơn ngày đặt hàng tương ứng”

**Bảng tầm ảnh hưởng:**

	Thêm	Xóa	Sửa
DatHang	-	-	+ (NgayDH)
GiaoHang	+	-	+ (NgayGH, MaPDH)

## 7. TRIGGER



**BÀI TẬP 3.6:** Ràng buộc liên thuộc tính – liên quan hệ  
Cho CSDL

**DATHANG** (MaPDH, NgayDH, ...)

**GIAOHANG** (MaPGH, MaPDH, NgayGH, ...)

**Phát biểu:** “Ngày giao hàng không thể nhỏ hơn ngày đặt hàng tương ứng”

**Yêu cầu:** Cài trigger cho thao tác thêm/sửa trên bảng **GIAOHANG**.



## 7. TRIGGER



```
CREATE TRIGGER KTRA_NGAYGH ON GIAOHANG
FOR INSERT, UPDATE
AS
    IF EXISTS (SELECT * FROM INSERTED I, DATHANG
D
                WHERE I.MAPDH = D.MAPDH
                AND I.NGAYGH < D.NGAYDH)
BEGIN
    PRINT 'NGAY GH KHONG NHO HON NGAY DH'
    ROLLBACK TRAN
END
```

## 7. TRIGGER



### BÀI TẬP 3.7: Cho CSDL

MATHANG (MAHANG, TENHANG, SOLUONG)  
NHATKYBANHANG (STT, NGAY, NGUOIMUA,  
MAHANG, SOLUONG, GIABAN)

**Yêu cầu:** Định nghĩa trigger có chức năng tự động giảm số lượng hàng hiện có khi một mặt hàng nào đó được bán (tức là khi câu lệnh INSERT được thực thi trên bảng NHATKYBANHANG). Gọi thực hiện trigger.

## 7. TRIGGER



```
create trigger trg_nkbh_insert
on nhattybanhang
for insert
AS
    Update mathang
    Set soluong = mathang.soluong - inserted.soluong
    From mathang, inserted
    Where mathang.mahang = inserted.mahang
Go

select * from mathang where mahang = 'H1'
insert into nhattybanhang(ngay, nguoi mua,
                           mahang, soluong, giaban)
values ('2012/10/23', 'khoa', 'H1', 3, 5000)
```



## 8. KIỂU DỮ LIỆU CURSOR



**KIỂU DỮ LIỆU CURSOR** hay còn gọi là Kiểu dữ liệu con trỏ.

Tại sao phải sử dụng Cursor?

Vì các lệnh trong SQL như:  
SELECT, UPDATE, DELETE, ...  
đều thao tác trên nhiều dòng dữ liệu  
thỏa mãn điều kiện where cùng một  
lúc mà không thể thao tác trên từng  
dòng dữ liệu cụ thể

## 8. KIỂU DỮ LIỆU CURSOR



### Khái niệm:

- ❑ Cursor là kiểu dữ liệu cơ bản dùng để duyệt qua từng dòng dữ liệu trả về từ câu truy vấn SELECT, giúp ta có thể có những xử lý khác nhau cho từng dòng dữ liệu cụ thể.
- Cursor cho phép cập nhật / xóa dữ liệu (dữ liệu thật sự trong CSDL) tương ứng với vị trí hiện hành của cursor.

## 8. KIỂU DỮ LIỆU CURSOR



### Đặc điểm:

- ❑ Cho phép thao tác trên từng dòng lệnh trả về từ câu lệnh SELECT.
- Do phải duyệt qua từng dòng dữ liệu nên xử lý hơi chậm.

## 8. KIỂU DỮ LIỆU CURSOR

Khi sử dụng cursor trong SQL Server cần lưu ý:

- **Cursor cần phải khai báo** và các thuộc tính của nó cũng cần được xác định.
- Khi sử dụng đến Cursor thì phải **mở cursor**
- **Đọc và xử lý từng dòng lệnh** bên trong cursor.
- Dữ liệu trong dòng hiện hành có thể được hiệu chỉnh nếu cần thiết.
- Tạm thời không dùng cursor thì **phải đóng cursor**.
- Cursor **cần phải được giải phóng** khi không cần dùng nữa.



## 8. KIỂU DỮ LIỆU CURSOR



### **Định nghĩa biến kiểu cursor**

Có thể khai báo theo cú pháp chuẩn hoặc cú pháp mở rộng của T-SQL.

## 8. KIỂU DỮ LIỆU CURSOR

### Cú pháp chuẩn

```
DECLARE <Tên_cursor> [Insensitive] [Scroll] CURSOR  
FOR  
    <Câu lệnh SELECT>  
[ FOR {Read Only | UPDATE [OF <cột 1> [, ..., cột n]]}]
```

**Tên\_Cursor:** Chiều dài 128 kí tự

## 8. KIỂU DỮ LIỆU CURSOR



**VD:**

```
DECLARE my_cur CURSOR
```

```
FOR SELECT MSSV, TenSV FROM SINHVIEN
```

Hoặc, có thể khai báo biến Cursor rồi sau đó mới gán lệnh SELECT vào Cursor

```
DECLARE @cur CURSOR
```

```
SET @cur = CURSOR
```

```
FOR SELECT MSSV, TenSV  
FROM SINHVIEN
```

## 8. KIỂU DỮ LIỆU CURSOR



### Cú pháp mở rộng

**DECLARE** <Tên\_cursor> **CURSOR**

[ **LOCAL** | **GLOBAL** ]

[ **FORWARD\_ONLY** | **SCROLL** ]

[ **STATIC** | **DYNAMIC** ]

[ **READ\_ONLY** ]

**FOR**

<Câu lệnh SELECT>

[ **FOR UPDATE** [ **OF** <cột 1> [, ..., cột n] ] ] }

## 8. KIỂU DỮ LIỆU CURSOR



**Câu lệnh Select:** Là câu lệnh truy vấn để định nghĩa tập kết quả của cursor.

**Phạm vi:**

- **LoCal:** chỉ sử dụng trong phạm vi khai báo
- **Global:** sử dụng chung cho cả kết nối.

**Mặc định là Local.**

## 8. KIỂU DỮ LIỆU CURSOR



### Di chuyển:

- **ForWard\_Only**: chỉ di chuyển một hướng theo chiều đi tới (**mặc định**).
- **Scroll**: di chuyển tùy ý (tiến lùi giữa các dòng trong Cursor)

### Xử lý:

- **Read\_Only**: chỉ đọc, không thể sử dụng cursor để cập nhật dữ liệu trong các bảng liên quan.

## 8. KIỂU DỮ LIỆU CURSOR



### Trạng thái:

- **Insensitive| Static (Cursor tĩnh):** dữ liệu trên Cursor không thay đổi mặc dù dữ liệu trong bảng nguồn thay đổi.
- **Dynamic (Cursor động):** dữ liệu trên Cursor sẽ thay đổi khi dữ liệu trong bảng nguồn thay đổi (**mặc định**).

Khi khai báo cursor với thuộc tính tĩnh (STATIC) thì dữ liệu trong cursor xem như là chỉ đọc.

## 8. KIỂU DỮ LIỆU CURSOR



- **UPDATE [OF <cột 1> [,...n]:** Quy định cột cho phép được **update** khi dùng cursor. Nếu **OF danh sách cột [, ...n]** được chỉ định rõ ràng thì chỉ có các cột được chỉ định mới được cho phép hiệu chỉnh, nếu không có thì **tất cả các cột có thể update.**



## 8. KIỂU DỮ LIỆU CURSOR



**Ví dụ:**

```
DECLARE my_cur CURSOR  
GLOBAL  
SCROLL  
DYNAMIC  
FOR SELECT MSSV, TenSV  
FROM SINHVIEN
```

## 8. KIỂU DỮ LIỆU CURSOR



**Mở cursor:** Cursor được mở sau khai báo để sử dụng

**OPEN** <Tên\_cursor|@Biến\_cursor>

## 8. KIỂU DỮ LIỆU CURSOR



**Sử dụng cursor:** Dùng lệnh **Fetch** để duyệt tuần tự qua cursor

**FETCH**

**NEXT** | PRIOR | FIRST | LAST |

ABSOLUTE n | REALTIVE n ]

**FROM** <Tên\_cursor|@Biến\_cursor>

[INTO @biến [,...n]]

Biến chứa giá trị của cursor. Số lượng biến phải = số cột trả ra của câu select khi gán cursor

## 8. KIỂU DỮ LIỆU CURSOR



### Hướng di chuyển:

- **NEXT**: Di chuyển về sau
- **PRIOR**: Di chuyển về trước
- **FIRST**: Di chuyển về đầu
- **LAST**: Di chuyển về cuối
- **ABSOLUTE n**: di chuyển đến mẫu tin thứ n tính từ mẫu tin đầu tiên (Nếu  $n < 0$ : tính từ mẫu tin cuối)
- **RELATIVE n**: giống ABSOLUTE nhưng tính từ mẫu tin hiện hành.

## 8. KIỂU DỮ LIỆU CURSOR



Cursor	FETCH NEXT	FETCH RELATIVE 3	FETCH ABSOLUTE 4	FETCH LAST
1 ←	1	1	1	1
2	2 ←	2	2	2
3	3	3	3	3
4	4	4	4 ←	4
5	5	5 ←	5	5
6	6	6	6	6 ←

## 8. KIỂU DỮ LIỆU CURSOR



- ❑ Mặc định: **Fetch Next**
- ❑ Đối với cursor dạng **forward\_only**, chỉ có thể **fetch next**.
- ❑ Biến hệ thống **@@Fetch\_status** cho biết lệnh **Fetch** vừa thực hiện có thành công hay không.

## 8. KIỂU DỮ LIỆU CURSOR



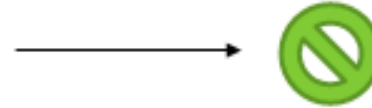
Truy xuất từng dòng dữ liệu. Kiểm tra phạm vi con trỏ bằng **@@Fetch\_status**, có giá trị trả về như sau:

- **0**: Đọc dữ liệu thành công.
- **-1**: Đọc dữ liệu thất bại (dòng vượt quá kết quả gán)
- **-2**: Đọc dữ liệu thất bại (dòng dữ liệu truy cập không tồn tại)

## 8. KIỂU DỮ LIỆU CURSOR




Trước lệnh `fetch` đầu tiên:  
`@@fetch_status` không xác định



Fetch next lần đầu tiên:  
`@@fetch_status` = 0 (*thành công*)

...

**Object**




`@@ fetch_status`  $\neq$  0 



## 8. KIỂU DỮ LIỆU CURSOR



### ❖ Đóng Cursor:

**Close** <Tên\_cursor|@Biến\_cursor>

Kết thúc hành động của Cursor cho 1 lần mở. Cursor vẫn hiện hữu cho đến khi gặp một lệnh Open khác.

### ❖ Giải phóng Cursor khỏi bộ nhớ

**Deallocate** Tên\_cursor

Giải phóng Cursor ra khỏi bộ nhớ. Sau lệnh này nếu có dòng lệnh nào truy cập đến Cursor đều gây lỗi.

## 8. KIỂU DỮ LIỆU CURSOR



### Thứ tự các thao tác khi xử lý dữ liệu trên CurSor

1. Định nghĩa biến Cursor
2. Mở Cursor bằng lệnh `Open`
3. Duyệt và xử lý dữ liệu trên Cursor

`Fetch (Next, ...)`

- Dùng lệnh `Into` để đưa giá trị của cursor vào biến
- Nếu không có lệnh `Into`, giá trị của cursor sẽ hiển thị ra màn hình kết quả sau lệnh `Fetch`.
- Có thể sử dụng vị trí hiện tại như là điều kiện cho mệnh đề `Where` của câu `Delete/ Update` (nếu cursor không là `Read_only`)

## 8. KIỂU DỮ LIỆU CURSOR



### Thứ tự các thao tác khi xử lý dữ liệu trên CurSor

4. Lặp lại việc duyệt và sử dụng cursor, có thể sử dụng biến `@@fetch_status` để biết đã duyệt qua hết cursor hay chưa.
5. Đóng CurSor bằng lệnh `Close <Tên_cursor>`
6. Giải phóng Cursor `Deallocate <Tên_cursor>`

⇒ *Sau khi đóng, vẫn có thể mở lại nếu cursor chưa bị hủy*

## 8. KIỂU DỮ LIỆU CURSOR



### Ví dụ 1:

SINHVIEN (MaSV, HoTen, MaKhoa)

- Định nghĩa biến kiểu cursor có tên là cs\_sinhvien gồm 2 cột dữ liệu trong bảng sinh viên là MASV, HOTEN.
- Mở và xử lý in các dòng dữ liệu.

## 8. KIỂU DỮ LIỆU CURSOR



SINHVIEN (MaSV, HoTen, MaKhoa)

**B1:** Định nghĩa một Cursor `cs_sinhvien` lấy giá trị của hai thuộc tính `MASV`, `HOTEN`

```
DECLARE cs_sinhvien CURSOR
FOR
    SELECT MASV, HOTEN
    FROM SINHVIEN
```

## 8. KIỂU DỮ LIỆU CURSOR



### B2: Mở cursor

```
OPEN cs_sinhvien
```

## 8. KIỂU DỮ LIỆU CURSOR



### B3: Xử lý dữ liệu trong cursor

```
FETCH NEXT FROM cs_sinhvien
WHILE (@@FETCH_STATUS = 0)
BEGIN
    --Đọc tiếp các dòng dữ liệu nếu thành công
    FETCH NEXT FROM cs_sinhvien
END
```

## 8. KIỂU DỮ LIỆU CURSOR



### B4: Đóng cursor

```
CLOSE MY_CUR
```

### B5: Hủy cursor

```
DEALLOCATE MY_CUR
```



## 8. KIỂU DỮ LIỆU CURSOR



### Ví dụ 2:

SINHVIEN (MaSV, HoTen, MaKhoa)

KHOA (MaKhoa, TenKhoa)

- Tạo cursor gồm: MaSV, MaKhoa
- Duyệt và đọc giá trị từ cursor
- Cập nhật lại giá trị

$\text{MaSV} = \text{MaKhoa} + \text{MaSV}$  hiện tại

- Áp dụng cho tất cả sinh viên

## 8. KIỂU DỮ LIỆU CURSOR



SINHVIEN (MaSV, HoTen, MaKhoa)

KHOA (MaKhoa, TenKhoa)

**B1:** Định nghĩa một Cursor lấy giá trị của hai thuộc tính  
MASV, MAKHOA

```
DECLARE  MY_CUR CURSOR
FOR
    SELECT MASV, MAKHOA
    FROM SINHVIEN
FOR UPDATE OF MASV
```

## 8. KIỂU DỮ LIỆU CURSOR



### B2: Mở cursor

```
OPEN MY_CUR
```

## 8. KIỂU DỮ LIỆU CURSOR

**B3:** Khai báo 2 biến @MASV, @MAKHOA để lấy dữ liệu đọc từ Cursor và cập nhật bảng SINHVIEN

```
DECLARE @MASV CHAR(20), @MAKHOA NCHAR(10)
--Đọc dữ liệu từ cursor đưa vào 2 biến
FETCH NEXT FROM MY_CUR INTO @MASV, @MAKHOA
WHILE (@@FETCH_STATUS = 0) -- đọc thành công
BEGIN
    UPDATE SINHVIEN
    SET MASV = @MAKHOA + @MASV
    WHERE MASV = @MASV -- CURRENT OF MY_CUR
    FETCH NEXT FROM MY_CUR INTO @MASV, @MAKHOA
END
```

## 8. KIỂU DỮ LIỆU CURSOR



### **B4:** Đóng cursor

```
CLOSE    MY_CUR
```

### **B5:** Hủy cursor

```
DEALLOCATE    MY_CUR
```

# Kết hợp Cursor và Thủ tục/ Trigger



Để dễ dàng sử dụng và quản lý cursor, ta cài đặt cursor lồng bên trong thủ tục/trigger. Như vậy, mỗi lần gọi thủ tục/trigger thì cursor sẽ được mở hoặc tạo mới, khi kết thúc thủ tục/trigger thì cursor sẽ được đóng lại hoặc huỷ đi.

**Tham khảo bài giảng/ trang 113.**



**Thank You !**