# CƠ SỞ DỮ LIỆU

**GIÁO VIÊN:** Đỗ Thị Mai Hường

**BỘ MÔN:** Các Hệ thống thông tin

**KHOA:** Công nghệ thông tin

Email: dohuong@gmail.com

# Chương 5 Ngôn ngữ T-SQL

1. Giới thiệu T-SQL
2. Stored Procedure
3. Function
4. Trigger

# Mục đích

- Nắm vững các khái niệm lô (batch) và xử lý theo lô
- Viết các câu lệnh SQL thể hiện logic của ứng dụng
- Định nghĩa và gán giá trị cho các biến
- Nắm vững và dùng được các lệnh điều khiển cấu trúc lập trình
- Nắm cách dùng biến con trỏ
- Viết được các thủ tục cơ bản đáp ứng yêu cầu qt csdl
- Viết được và Sử dụng được hàm SQL trong truy vấn
- Tạo được các trigger cơ bản

# Giới Thiệu Transact SQL (T-SQL)

- Transact-SQL là ngôn ngữ SQL mở rộng dựa trên SQL chuẩn của ISO (International Organization for Standardization) và ANSI (American National Standards Institute) được sử dụng trong SQL Server T-SQL được chia làm 3 nhóm:

  Data Definition Language (DDL):lệnh dùng để quản lý các thuộc tính của một database như định nghĩa các hàng hoặc cột của một table, hay vị trí data file của một database...thường có dạng

  Create object_Name
  Alter object_Name
  Drop object_Name
  Trong đó object_Name có thể là một table, view, stored procedure, indexes...

  Ví dụ:

  Lệnh Create sau sẽ tạo ra một table tên Importers với 3 cột CompanyID,CompanyName,Contact

  USE Northwind
- CREATE TABLE Importers(
  CompanyID int NOT NULL,
  CompanyName varchar(40) NOT NULL,
  Contact varchar(40) NOT NULL
  )

# Giới Thiệu Transact SQL (T-SQL)

- Data Control Language (DCL):

Đây là những lệnh quản lý các quyền truy cập lên từng object (table, view, stored procedure...). Thường có dạng sau: Grant, Revoke, Deny
Ví dụ:

Lệnh sau sẽ cho phép user trong Public Role được quyền Select đối với table Customer trong database Northwind (Role là một khái niệm giống như Windows Group sẽ được bàn kỹ trong phần Security)

```
USE Northwind
GRANT SELECT
ON Customers
TO PUBLIC
```

Lệnh sau sẽ từ chối quyền Select đối với table Customer trong database Northwind của các user trong Public Role

```
USE Northwind
DENY SELECT
ON Customers
TO PUBLIC
```

Lệnh sau sẽ xóa bỏ tác dụng của các quyền được cho phép hay từ chối trước đó

```
USE Northwind
REVOKE SELECT
ON Customers
TO PUBLIC
```

# Giới Thiệu Transact SQL (T-SQL)

- Data Manipulation Language (DML):

  Đây là những lệnh phổ biến dùng để xử lý data như Select, Update, Insert, Delete

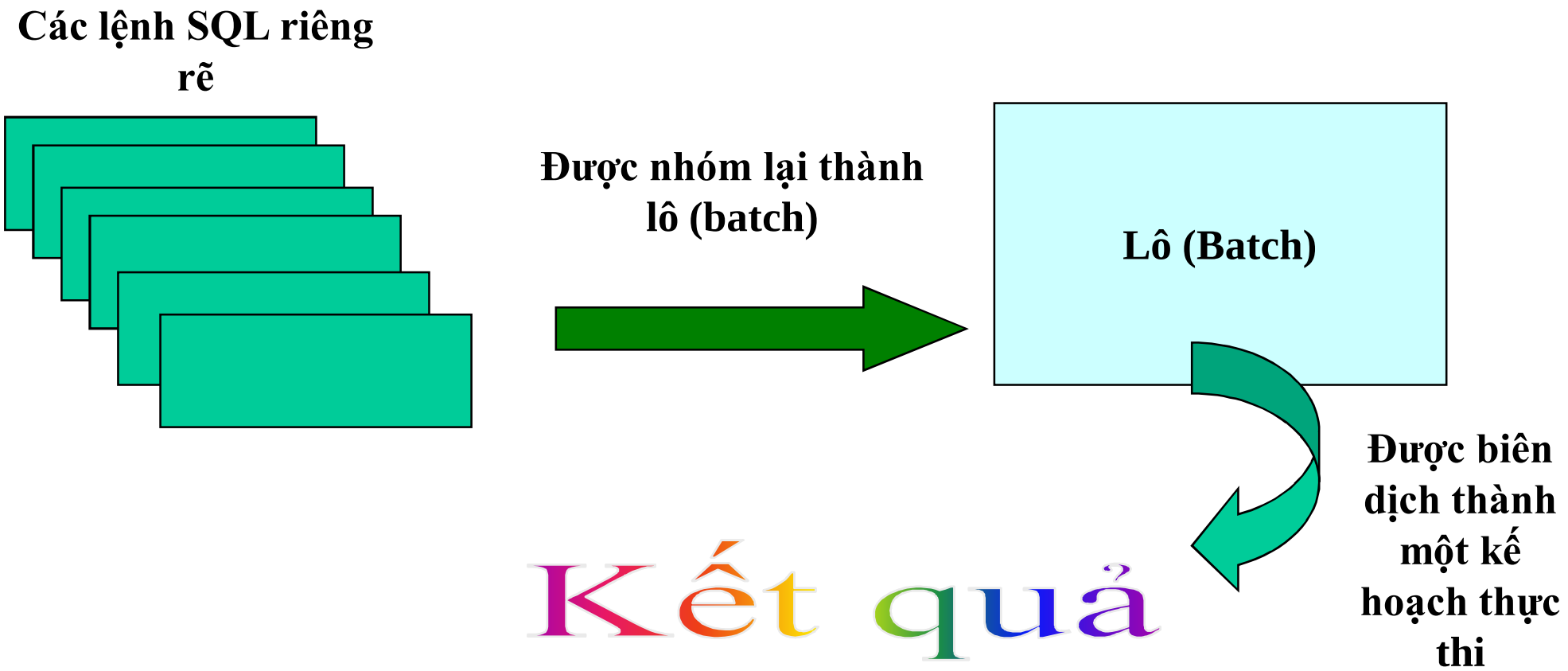
# Giới Thiệu Transact SQL (T-SQL)

- Vd: USE qlysv

```
DECLARE @stt INT
CREATE TABLE sv
        (stt          INT,
        masv        NVARCHAR(10)       NOT NULL
                CONSTRAINT pk_sinhvien PRIMARY KEY,
      hoten         NVARCHAR(50)       NOT NULL ,
        ngaysinh                SMALLDATETIME    NULL ,
        gioitinh        BIT             NULL ,
        noisinh                 NVARCHAR(100)    NULL ,
        malop                   NVARCHAR(10)       NULL
)

SELECT @stt = @@IDENTITY -- giá trị identity gần nhất được sinh ra ( có gt <> NULL nếu lấy sau câu lệnh insert dữ liệu bảng có chưa cột lấy giá trị identity
INSERT INTO sv
VALUES(@stt,'SV01CDT2K8', 'Trần Thu Thuỷ', '11/10/1987',0,'Thái bình','CDT001K008')
SELECT * FROM SV
```

# Giới thiệu về xử lý theo lô (SQL Batch Processing)



# Định nghĩa

Quá trình trong đó một tập lệnh được xử lý cùng lúc được gọi là

**Xử lý theo lô**

# Ví dụ về một lô (batch)

```
Use QlSach
Select * from tacgia
Update tacgia
set phone= ‘098890 4566‘
where tentg = ‘Trung'
Go
```



**Lệnh báo hiệu kết thúc lô**

# Mục đích: Hay dùng phân tách nhóm để thực hiện độc lập

- VDụ: Xét kịch bản

```
CREATE DaTaBASE qlbanhang
USE qlbanhang
CREATE TABLE ktra
(
        A INT,
        B INT
)
SELECT * FROM ktra
```

- --Sẽ bị báo lỗi

# Cần sửa lại

- CREATE DaTaBASE qlbanhang
  GO
  USE qlbanhang
  CREATE TABLE ktra
  (
  A INT,
  B INT
  )
  GO
  SELECT * FROM ktra
  GO

# Chú thích trong một lô xử lý

## Chú thích

- Các chuỗi ký tự trong mã lệnh chương trình (còn được gọi là chú thích) không được xử lý bởi trình biên dịch.
- Dùng để giải thích cho mã lệnh hay vô hiệu hóa tạm thời các thành phần câu lệnh T-SQL đang xử lý
- Giúp việc bảo trì mã lệnh dễ dàng hơn.
- Chú thích thường được sử dụng để ghi lại tên chương trình, tên tác giả và ngày tháng thực hiện thay đổi mã lệnh.
- Chú thích có thể được dùng để mô tả các phép tính toán phức tạp hay giải thích về phương pháp lập trình.

# Các hình thức chú thích

**SQL Server hỗ trợ hai hình thức chú thích:**

**1) --(hai ghạch ngang)**

**Ví dụ:**

```
USE Qlsach
GO
-- Đây là chú thích.
```

**2) /* ... */ (cặp dấu ghạch chéo và dấu sao)**

**Ví dụ:**

```
SELECT * FROM nhanvien /*Đây là chú thích*/
```

# Chú thích nhiều dòng

- Chú thích nhiều dòng /* */ không thể vượt quá một lô. Một chú thích hoàn chỉnh phải nằm trong một lô xử lý.
- Ví dụ, trong công cụ Query Analyzer, lệnh GO báo hiệu kết thúc lô. Khi gặp lệnh GO trên dòng lệnh nó sẽ gửi tất cả các mã lệnh sau từ khóa GO cuối cùng lên máy chủ SQL trong một lô xử lý.
- Nếu lệnh GO xuất hiện trên một dòng giữa /* và */ thì Query Analyzer sẽ gửi đi một đoạn chú thích có các ký tự đánh dấu sai trong mỗi lô và sẽ gây ra lỗi cú pháp.

# *Chú ý:*

- *Đối với các lệnh CREATE như là:* ***CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER, CREATE VIEW*** *không được phép kết hợp với các lệnh khác trong cùng một lô.*

# Biến cục bộ

**Biến là một đối tượng có thể chứa dữ liệu**

**Dữ liệu có thể được đưa vào các câu lệnh SQL dùng biến cục bộ**

**Tên của các biến cục bộ phải bắt đầu bằng '@'**

**Từ khóa SET hay SELECT được dùng để gán giá trị cho biến cục bộ**

***Ví dụ :*** DECLARE @cust VARCHAR(20)

SET @cust= 'FRANK'

# Khai báo biến

- DECLARE @Tên_biến  Kiểu_dữ_liệu [, ...]
- Kiểu dữ liệu text, ntext hoặc image không được chấp nhận khi khai bao bien
- 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**

# Ví du:

- *Để gán giá trị là ngày 25/03/2002 vào biến ngày xuất hàng ta sử dụng lệnh SET như sau:*
- DECLARE @Ngayxh DATETIME

  SET @Ngayxh='2002-03-25'
- ***Chú ý:****Đối với kiểu dữ liệu dạng ngày trong Microsoft SQL Server thường sử dụng theo* ***định dạng yyyy-mm-dd*** *để gán giá trị vào biến hoặc vào trong cơ sở dữ liệu.*

# Ví dụ:

- *Để tính lương lớn nhất, lương nhỏ nhất, tổng lương trong phòng ban có mã là 'PB01'. Sử dụng lệnh SELECT như sau:*

```
DECLARE @LuongLN INT,@LuongNN int, @Tongluong int
SELECT @LuongLN=MAX(Luong), @LuongNN=Min(luong),
@Tongluong=Sum(luong)
FROM Nhanvien
WHERE MaPB='PB01'
```

# Xem giá trị hiện hành của biến

- PRINT @Tên_biến | Biểu_thức_chuỗi
- *Để tính lương lớn nhất, lương nhỏ nhất, tổng lương trong phòng ban có mã là 'PB01'. Sử dụng lệnh SELECT và lệnh Print như sau:*

```
DECLARE @LuongLN INT,@luongnn int, @tongluong int
  SELECT @LuongLN=MAX(Luong), @luongnn=Min(luong),
  @tongluong=Sum(luong)
   FROM Nhanvien
   WHERE MaPB='PB01'
   Print 'Luong LN là ' + convert(varchar(10),@luongLN)
  Print 'Luong NN là ' + convert(varchar(10),@luongnn)
  Print 'Tổng lương là ' + cast(@tongluong  as varchar)
```

# Các loại biến

SQL Server hỗ trợ hai loại biến sau trong T-SQL:

Biến toàn cục

Biến cục bộ

# Các biến toàn cục

Biến toàn cục trong SQL Server bắt đầu bằng 2 ký tự @. Ta có thể truy xuất giá trị của các biến này bằng truy vấn SELECT đơn giản

```
SELECT @@VERSION AS SQL_SERVER_VERSION_DETAILS
```

| | SQL_SERVER_VERSION_DETAILS |
|---|---|
| 1 | Microsoft SQL Server 2000 - 8.00.194 (Intel X86) Aug 6 2000 00:57:48 ... |

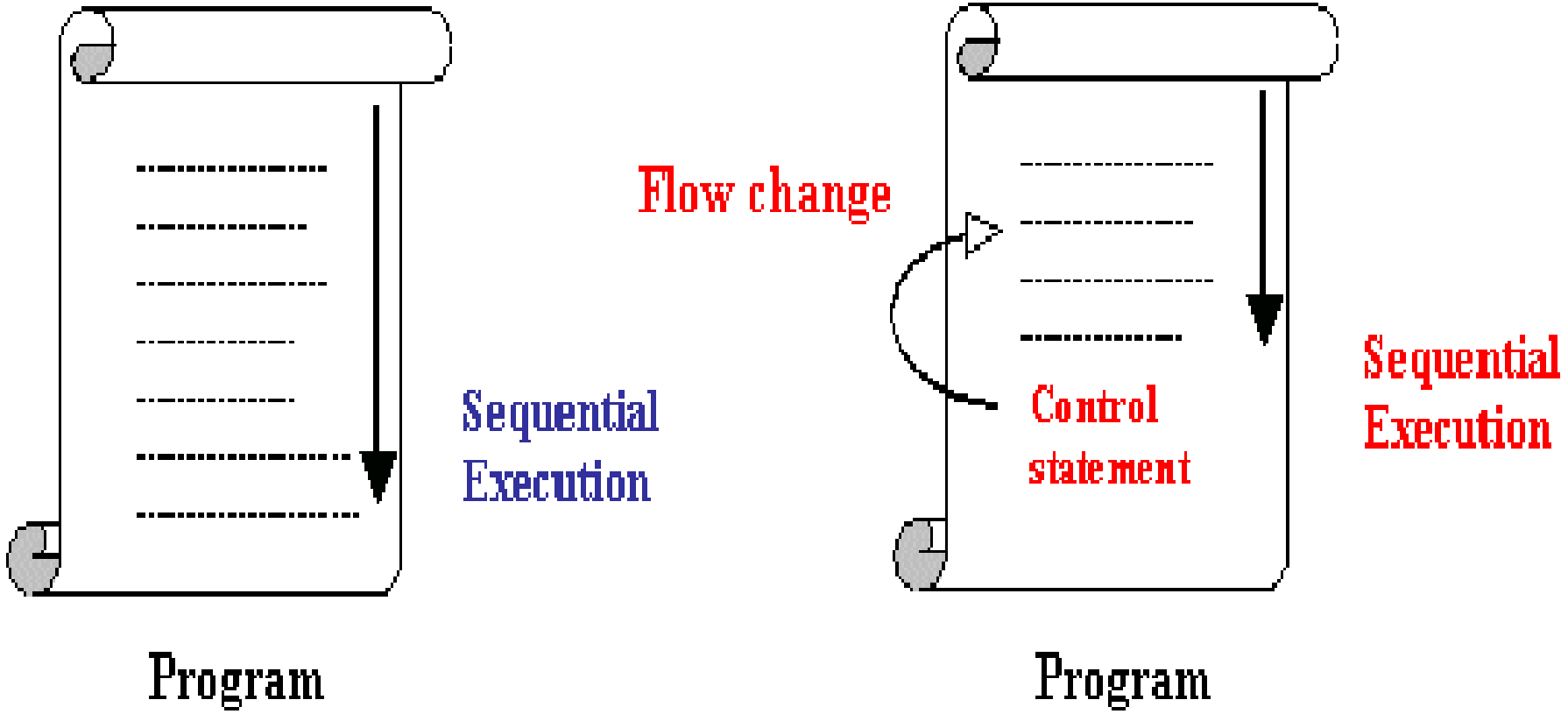
# Danh sách các biến toàn cục

| Các biến | Ý nghĩa |
|---|---|
| @@CONNECTIONS | Số các kết nối đên máy chủ từ lần khởi động cuối. |
| @@CPU_BUSY | Số milliseconds (một phần ngìn giây) hệ thống đã xử lý từ khi SQL Server được khởi động |
| @@CURSOR_ROWS | Số bản ghi trong cursor mở gần nhất. |
| @@DATEFIRST | Giá trị hiện tại của tham số trong lệnh SET DATEFIRSTquyết định ngày đầu tiên của tuần. |
| @@ERROR | Mã lỗi của lỗi xảy ra gần nhất |
| @@FETCH_STATUS | 0 nếu trạng thái lần truy xuất cuối thành công. -1 nếu có lỗi |

# Danh sách các biến toàn cục(tiếp…)

| Các biến | Ý nghĩa |
|---|---|
| @@IDENTITY | Giá trị identity gần nhất được sinh ra |
| @@LANGUAGE | Tên của ngôn ngữ đang được sử dụng. |
| @@MAX_CONNECTIONS | Số kết nối tối đa có thể. |
| @@ROWCOUNT | Số bản ghi bị tác động bởi câu lệnh SQL gần nhất. |
| @@SERVERNAME | Tên của máy chủ |
| @@SERVICENAME | Tên của dịch vụ SQL trên máy chủ |
| @@TIMETICKS | Số milliseconds trong một tick trên máy chủ |
| @@TRANSCOUNT | Số giao dịch đang hoạt động trên kết nối hiện tại |
| @@VERSION | Thông tin về phiên bản của SQL Server |

# Các lệnh điều khiển



# Các lệnh điều khiển(tiếp..)

| Control Keyword | Purpose |
|---|---|
| BEGIN...END | To create a statement block. |
| GOTO label | To transfer the flow to the specified label. |
| IF...ELSE | To execute different set of statement/s based on the specified condition. |
| WHILE | To repeat statement/s while a specified condition holds TRUE. |
| BREAK | To break the flow of execution and come out of the WHILE loop. |
| CONTINUE | To restart a WHILE loop. |
| WAITFOR | To set a delay for statement execution. |
| RETURN | To exit unconditionally. |

# BEGIN..END

BEGIN...END : Một tập lệnh SQL được thực thi sẽ được đặt trong BEGIN..END.

Cú pháp:

```
BEGIN
        <lệnh> | <đoạn lệnh>
END
```

# IF..ELSE

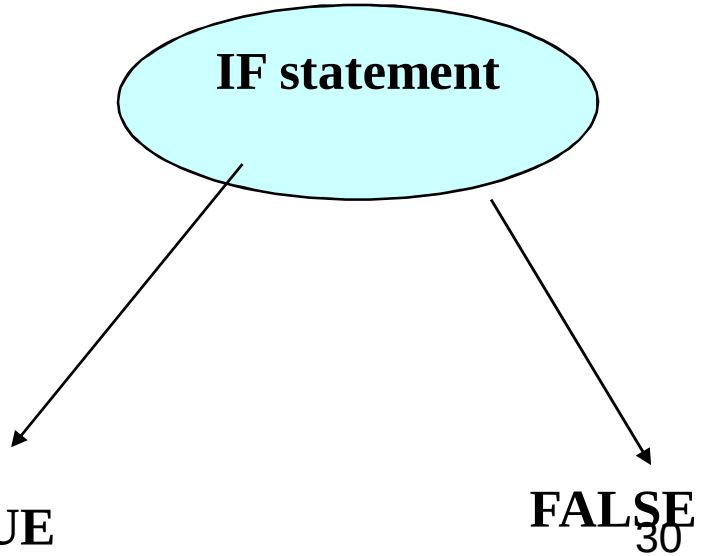
**IF...ELSE: Chúng ta có thể thực thi các tập lệnh SQL khác nhau dựa vào các điều kiện khác nhau.**

**Cú pháp:**

IF <điều kiện>

< lệnh sql1> | <tập lệnh1>

[ ELSE
< lệnh sql2>|< tập lệnh2> ]



# Ví dụ về IF

```
IF (SELECT COUNT(ORDERID) FROM ORDERS)>1
   BEGIN
        GOTO X
   END
ELSE
   BEGIN
        SELECT * FROM CUSTOMERS
   END
SELECT * FROM ORDERS
X:
SELECT * FROM SHIPPERS
```

| | ShipperID | CompanyName | Phone |
|---|---|---|---|
| 1 | 1 | Speedy Express | (503) 555-9831 |
| 2 | 2 | United Package | (503) 555-3199 |
| 3 | 3 | Federal Shipping | (503) 555-9931 |

# **IF** có kết hợp từ khóa **EXISTS**

- Để kiểm tra sự tồn tại của các dòng dữ liệu bên trong bảng
- IF EXISTS (Câu_lệnh_SELECT)

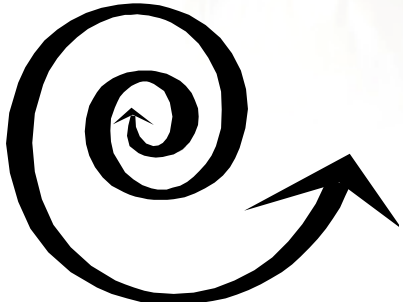
  Câu_lệnh1 | Khối_lệnh1

  [ ELSE

  Câu_lệnh2 | Khối_lệnh2 ]

# Cấu trúc WHILE

**WHILE: C**ó thể thực thi một lệnh SQL hay một tập lệnh dựa vào điều kiện nào đó. Các câu lệnh được thực thi nhiều lần khi nào điều kiện vẫn còn đúng.



**Cú pháp:**

```
WHILE  <Điều kiện>
 BEGIN
          Các_lệnh_lặp
 END
```

# BREAK và CONTINUE

**Chúng ta có thể dùng từ khóa *CONTINUE* và *BREAK* trong vòng lặp while để điều khiển phần thực thi của các câu lệnh.**

```
USE pubs
GO
WHILE (SELECT AVG(price) FROM titles) < $30
BEGIN
UPDATE titles
        SET price = price * 2
SELECT MAX(price) FROM titles
IF (SELECT MAX(price) FROM titles) > $50
        BREAK
ELSE
        CONTINUE
END
PRINT 'Too much for the market to bear'
```

# Sơ đồ đầy đủ:

- WHILE Biểu_thức_logic
  BEGIN
  Các_lệnh_nhóm_lặp1
  [ IF Biểu_thức_lặp_tiếp
  CONTINUE ]
  [ IF Biểu_thức_thoát
  BREAK ]
  Các_lệnh_nhóm_lặp2
  END
  Các_lệnh_khác

# Từ khóa GOTO

GOTO:

Có thể thay đổi dòng thực thi của chương trình đến một điểm (còn gọi là nhãn).

Các lệnh sau từ khóa GOTO sẽ được bỏ qua và tiến trình thực thi tiếp tục ở vị trí nhãn chỉ ra trong mệnh đề GOTO.

Cú pháp:

GOTO <nhãn>

# RETURN

RETURN: Ta có thể dùng RETURN bất cứ lúc nào để thoát khỏi một đoạn lệnh hay một thủ tục. Các lệnh sau từ khóa RETURN sẽ không được thực thi.

Cú pháp:

RETURN [số nguyên]

# Hàm CASE

**Cú pháp 1:**

```
CASE <input_expression>
     WHEN when_expression THEN       result_expression
     [WHEN …]
     [  ELSE else_result_expression  ]
  END
```

**Cú pháp 2:**

```
CASE
     WHEN Boolean_expression THEN result_expression
     [WHEN ... ]
     [       ELSE else_result_expression     ]
  END
```

# Hàm CASE (2)

```
declare @st varchar(100)
declare @i float

set @i=RAND()

 SELECT   @st =
     CASE
           WHEN @i<0.2 THEN 'Gia tri nho hon 0.2'
            WHEN @i<0.4 THEN 'Gia tri nho hon 0.4'
    ELSE
           'Cac gia tri khac'
     END
  print @st
```

# Hàm CASE (3)

```
USE AdventureWorks;
GO
SELECT   ProductNumber, Category =
      CASE ProductLine
         WHEN 'R' THEN 'Road'
         WHEN 'M' THEN 'Mountain'
         WHEN 'T' THEN 'Touring'
         WHEN 'S' THEN 'Other sale items'
         ELSE 'Not for sale'
      END,   Name
FROM Production.Product
ORDER BY ProductNumber;GO
```

# Hàm CASE (4)

```
USE AdventureWorks;GO
SELECT   ProductNumber, Name, [Price Range] =
     CASE
       WHEN ListPrice =  0 THEN 'Mfg item - not for resale'
       WHEN ListPrice < 50 THEN 'Under $50'
       WHEN ListPrice >= 50 and ListPrice < 250 THEN 'Under $250'
       WHEN ListPrice >= 250 and ListPrice < 1000 THEN 'Under $1000'
       ELSE 'Over $1000'
     END
FROM Production.Product
ORDER BY ProductNumber ;
GO
```

# Con trỏ

- Một con trỏ là một đối tượng csdl, được sử dụng để thao tác với từng hàng dữ liệu
- Với con trỏ ta có thể:
  - Cho phép định vị các hàng chỉ định của tập kết quả.
  - Nhận về một hàng đơn hoặc tập hợp các hàng từ vị trí hiện tại của tập kết quả.
  - Hỗ trợ sửa đổi dữ liệu của hàng ở vị trí hiện tại trong tập kết quả.
  - Hỗ trợ quan sát đối với các thay đổi được tạo ra bởi các người dùng khác trên các dữ liêu của tập kết quả.

# Tạo con trỏ

- Lệnh DECLARE dùng để tạo một con trỏ.
- Lệnh này chứa các lệnh SELECT để bao gồm các bản ghi từ bảng. Cú pháp là:

```
DECLARE  <Tên con trỏ> CURSOR
[LOCAL | GLOBAL]
[FORWARD ONLY | SCROLL]
[STATIC | KEYSET | DYNAMIC ]
[READ_ONLY | SCROLL_LOCKS]
        FOR <Lệnh SELECT>
[FOR UPDATE [OF <Tên cột> [,....N]]]
```

# Các bước sử dụng con trỏ

- Mở con trỏ
  OPEN <Cursor_name>
- Nhận về các bản ghi
  FETCH <Cursor_name>
- Đóng con trỏ
  CLOSE <Cursor_name>
- Xoá các tham chiếu tới con trỏ
  DEALLOCATE <Cursor_name>

# Truy xuất và duyệt con trỏ

FETCH [NEXT | PRIOR | FIRST | LAST
| ABSOLUTE n | RELATIVE n]
FROM Tên_cursor
[INTO Danh_sách_biến]

FETCH FIRST: Truy xuất hàng đầu tiên.

FETCH NEXT: Truy xuất hàng tiếp theo

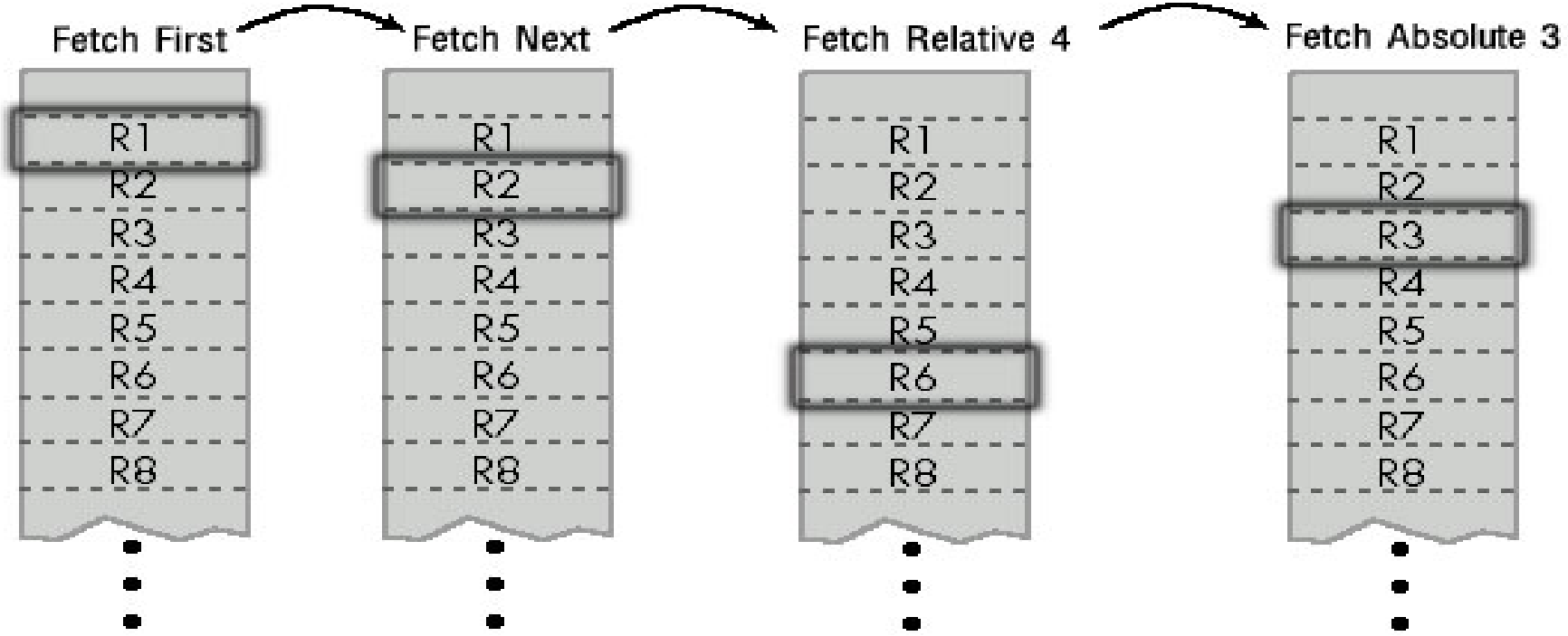
FETCH PRIOR: Truy xuất hàng trước hàng truy xuất trước đó.

FETCH LAST: Truy xuất hàng cuối cùng.

FETCH ABSOLUTE *n*: Nếu *n* là một số nguyên dương, truy xuất hàng n trong con trỏ. Nếu *n* là một số nguyên âm, hàng n trước hàng cuối cùng trong con trỏ được truy xuất. Nếu *n* bằng 0, không hàng nào được truy xuất.

# Truy xuất và duyệt con trỏ

- FETCH RELATIVE *n*: Truy xuất *n* hàng từ hàng truy xuất trước đó, nếu *n* là số dương. Nếu *n* là số âm, *n* hàng trước hàng truy xuất trước đó được truy xuất. Nếu *n* bằng 0, hàng hiện tại được nhận về.



# Các biến toàn cục của lệnh FETCH

- @@FETCH _STATUS: Biến này trả về một số nguyên biểu diễn kết quả của lệnh truy xuất cuối cùng của con trỏ.
  - 0 Truy xuất thành công
  - -1 lỗi
- @@CURSOR_ROWS: Biến này trả về tổng số hàng hiện tại trong con trỏ đang mở.

# Ví dụ tạo con trỏ

```
DECLARE Pub_Cursor CURSOR SCROLL
        FOR SELECT * FROM publishers ORDER BY pub_name
OPEN Pub_Cursor
FETCH FIRST FROM Pub_Cursor
WHILE @@FETCH_STATUS = 0
BEGIN
        FETCH NEXT FROM Pub_Cursor
END
```

| | pub_id | pub_name | city | state | country |
|---|---|---|---|---|---|
| 1 | 1389 | Algodata Infosystems | Berkeley | CA | USA |

| | pub_id | pub_name | city | state | country |
|---|---|---|---|---|---|
| 1 | 0877 | Binnet & Hardley | Washington | DC | USA |

| | pub_id | pub_name | city | state | country |
|---|---|---|---|---|---|
| 1 | 1622 | Five Lakes Publishing | Chicago | IL | USA |

| | pub_id | pub_name | city | state | country |
|---|---|---|---|---|---|
| 1 | 9901 | GGG&G | München | NULL | Germany |

| | pub_id | pub_name | city | state | country |
|---|---|---|---|---|---|

Grids | Messages

# Ví dụ tạo con trỏ

```
Begin
create table danhsach(sobd nchar(10),manv nchar(10), hoten nvarchar(50), ngaysinh
    datetime)
declare @ma nchar(10), @ten nvarchar(50),@ns datetime,@i int
set @i=1
DECLARE cur_tro CURSOR FORWARD_ONLY FOR SELECT manv,hoten,ngaysinh from
    nhanvien
 OPEN cur_tro
    WHILE 0=0--@@FETCH_STATUS=0
    BEGIN
      FETCH NEXT FROM cur_tro
            INTO @ma,@ten,@ns
            IF @@FETCH_STATUS<>0
                  BREAK
            insert into danhsach  values('SBD'+convert(nchar(7),@i),@ma,@ten,@ns)
      set @i=@i+1
   end
CLOSE cur_tro
DEALLOCATE cur_tro
End
```

Kiểm tra:

```
Select * from danhsach
```

# Con trỏ

## Bài tập:

Thêm trường tongsogio vào bảng Duan

Sử dụng con trỏ cập nhật lại giá trị cho trường tongsogio.

Hướng dẫn:

Tongsogio trong bảng Duan bằng tổng của số giờ của các nhân viên tham gia mã dự án này trong bảng Phancong

**Cách 1:** Sử dụng con trỏ duyệt từng bản ghi trong bảng Duan để lấy ra số mã dự án. Sau đó vào bảng phancong tính tổng số giờ của mã dự án này. Cuối cùng quay lại bảng Duan để cập nhật lại Tongsogio

**Cách 2:** Sử dụng con trỏ duyệt từng bản ghi trong bảng phancong lấy ra mã dự án và tổng số giờ của dự án này. Sau đó sang bảng Duan cập nhật lại tongsogio

# Con trỏ

```
DECLARE @ma nchar(10),@tsg int
 DECLARE tro CURSOR  FORWARD_ONLY
 FOR  Select mada,sum(sogio)
        From Phancong
        Group by mada
OPEN tro
FETCH FIRST FROM tro @ma,@tsg
WHILE @@FETCH_STATUS=0
BEGIN
update duan    set tongsogio =@tsg    where mada=@ma
   Print ‘ Đang cập nhật mã dự án '+@ma
FETCH NEXT FROM tro  INTO @ma,@tsg
END
```

# Tổng kết

- Con trỏ được tạo bằng lệnh DECLARE. Đầu tiên con trỏ được khai báo và tạo ra trong bộ nhớ. Sau đó nó mới được mở.
- Lệnh OPEN mở con trỏ. Việc nhận về các bản ghi từ một con trỏ được gọi là fetching. Một người dùng chỉ có thể nhận về một bản ghi tại một thời điểm.
- Lệnh FETCH được sử dụng để đọc các bản ghi từ con trỏ.
- Ngầm định, một con trỏ là forward only. Nó có thể truy xuất tuần tự các bản ghi từ bản ghi đầu tiên đến bản ghi cuối cùng.

# Thủ tục lưu trữ

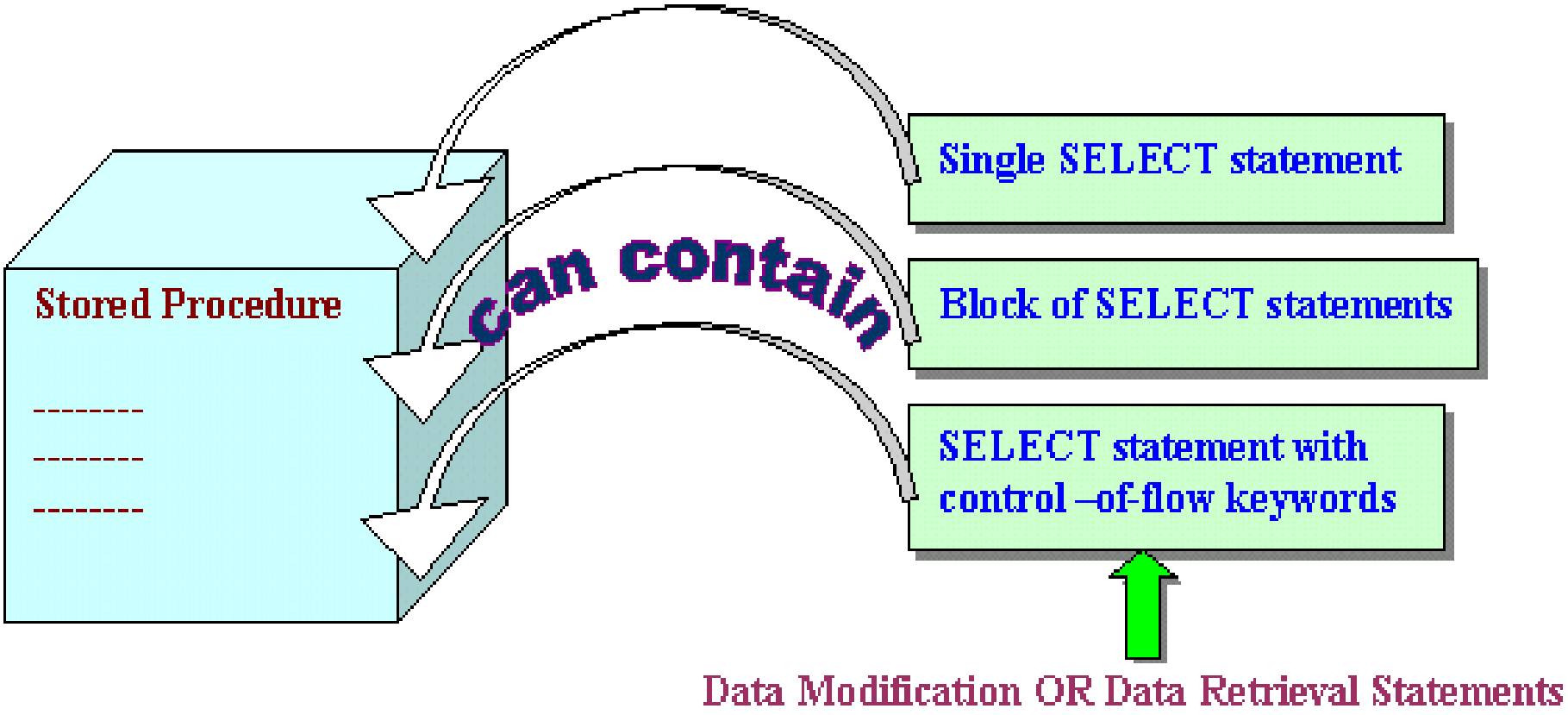
# Mục tiêu

- Định nghĩa các thủ tục lưu trữ.
- Giải thích quá trình tạo lập, sửa và thực thi các thủ tục lưu trữ do người dùng định nghĩa.
- Sử dụng các tham số và các biến trong thủ tục lưu trữ.
- Thực hiện cài đặt thủ tục trên ví dụ
- Chọn các tuỳ chọn biên dịch lại phù hợp.
- Tìm hiểu báo lỗi trong thủ tục lưu trữ.

# Thủ tục lưu trữ

- Tập hợp biên dịch các câu lệnh T-SQL được lưu trữ với một tên xác định
- Sử dụng để thực hiện các nhiệm vụ quản trị, hoặc áp dụng các luật giao dịch phức tạp
- Có hai loại thủ tục lưu trữ:
  - Thủ tục lưu hệ thống đề cập đến phương pháp quản trị dữ liệu và cập nhật thông tin vào các bảng (thường bắt đầu bằng sp_).
  - Thủ tục lưu do người dùng định nghĩa.

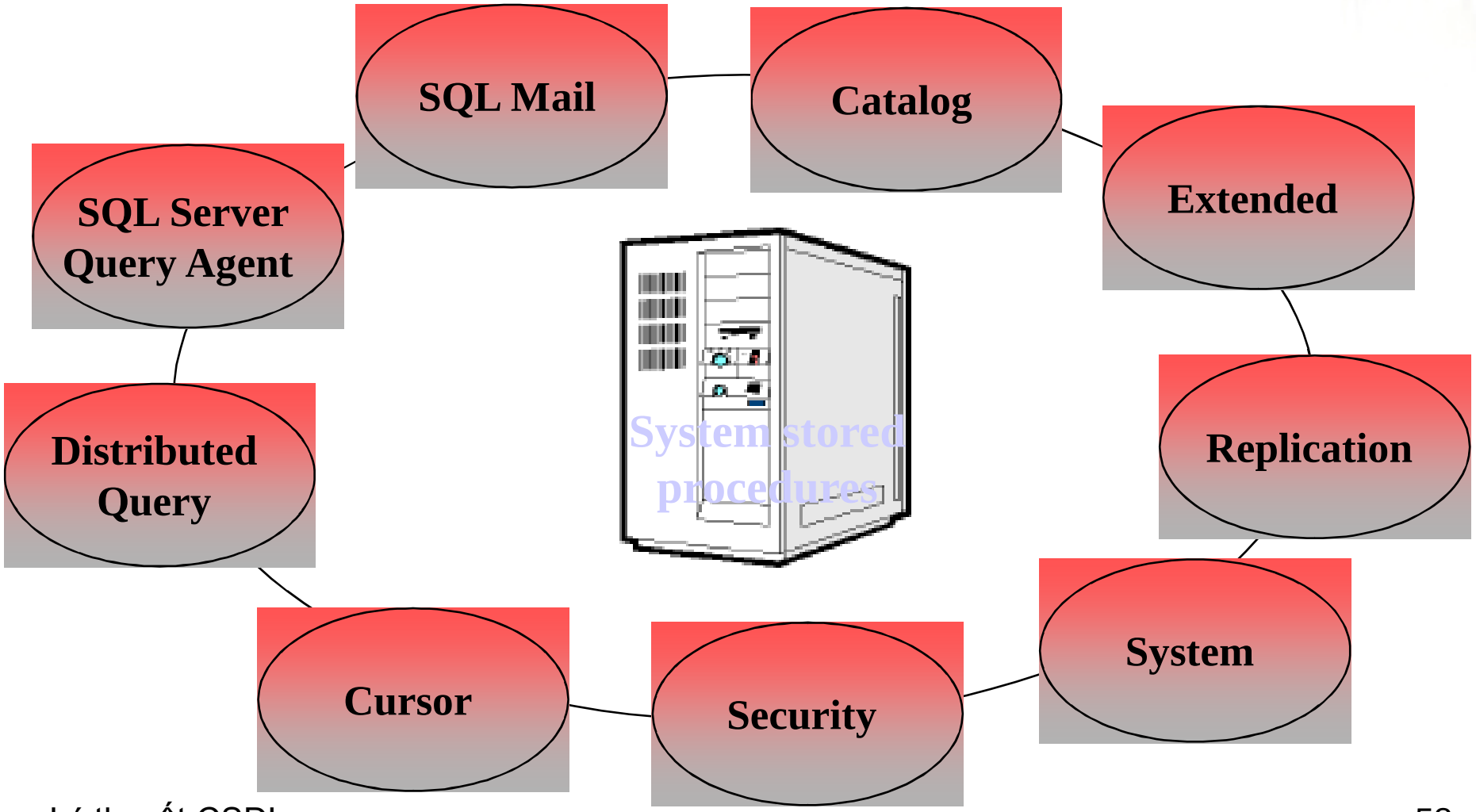
# Thủ tục lưu trữ < tiếp tục…>



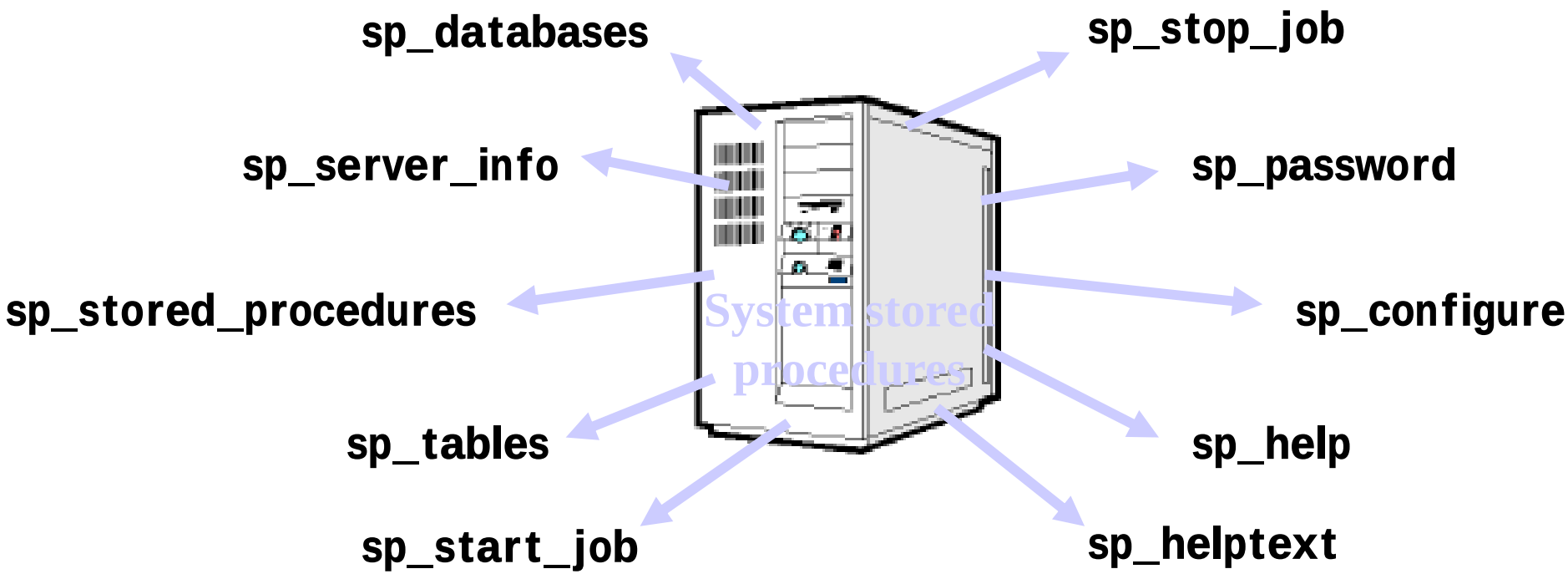
# Lợi ích của thủ tục

- Tăng tôc độ thực hiện : Các thủ tục được tối ưu hóa lần đầu tiên khi chúng biên dịch ->cho phép thực thi với chi phí it hơn so với T-SQL thông thường.
- Tốc độ truy nhập dữ liệu nhanh hơn: SQl không phải lựa chọn cách tốt nhất để xử lý các lệnh SQL và truy suất 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 đông 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 quyên thực thi cho các thủ tục vì vậy nó thực hiện đúng tác vụ người dùng.

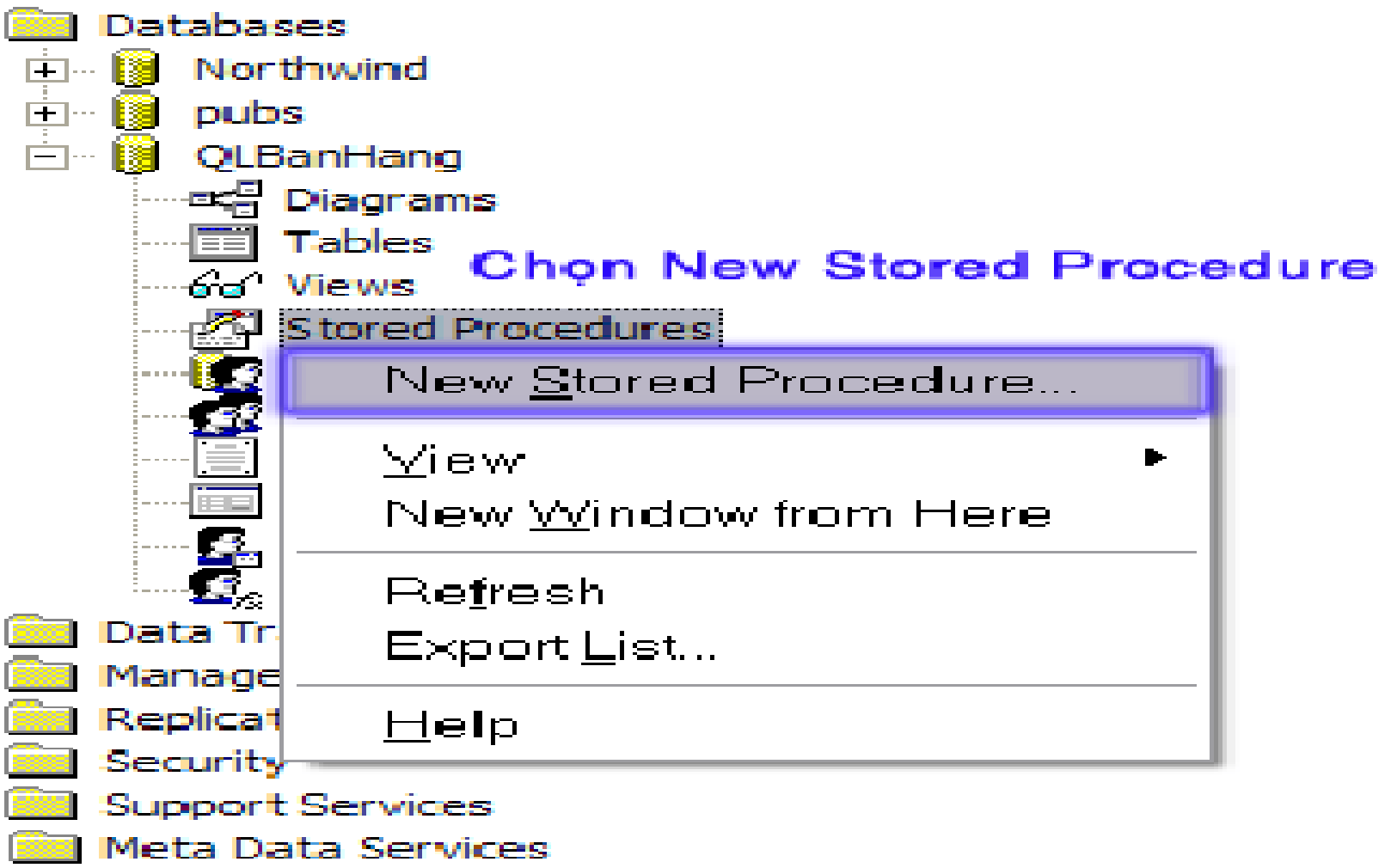
# Các danh mục của thủ tục lưu trữ hệ thống



# Ví dụ về hệ thống thủ tục lưu trữ

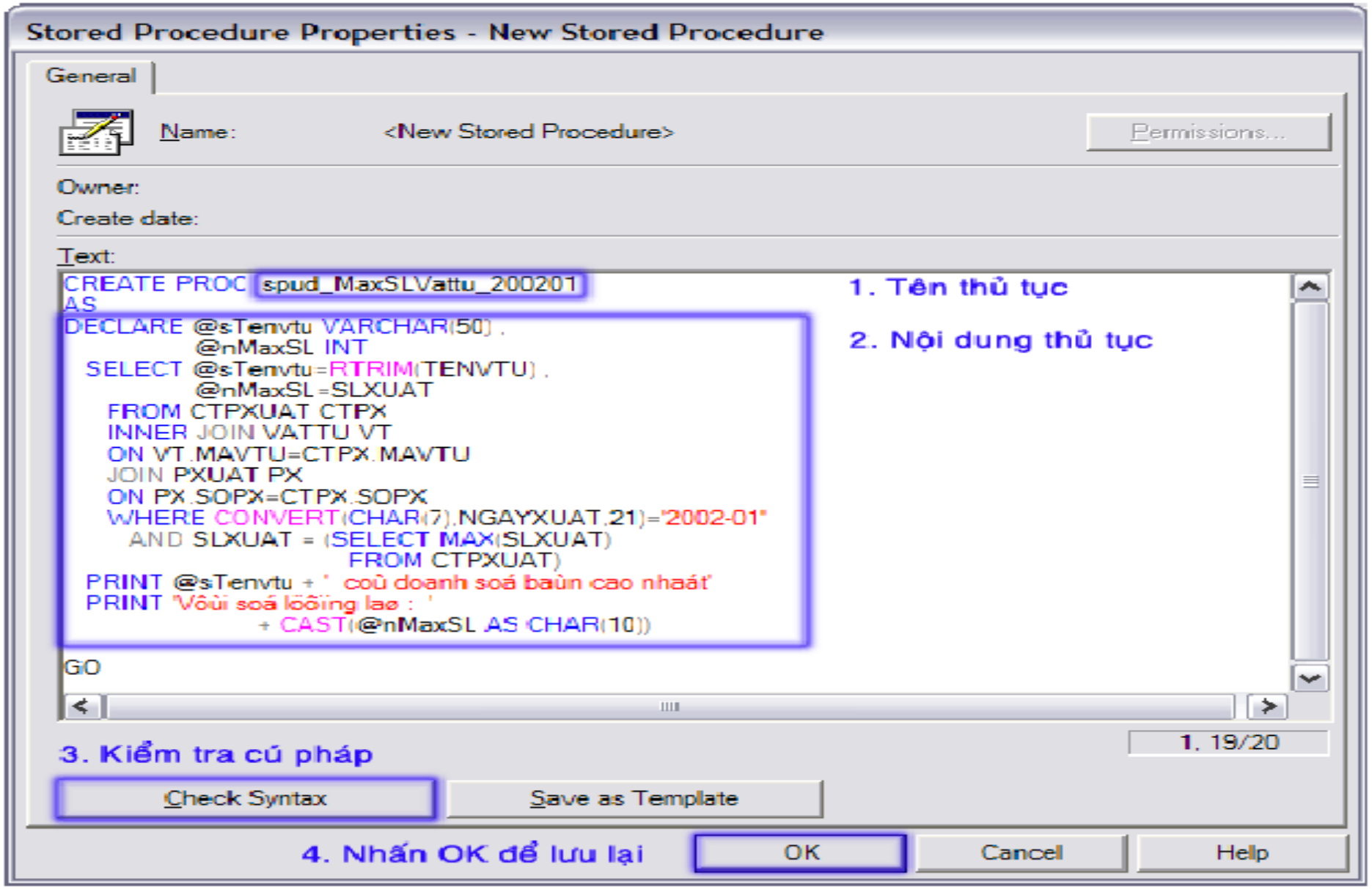


# Định nghĩa thủ tục lưu trữ bằng EM: Bước 1:




5-1. Chọn New Stored Procedure để tạo thủ tục nội

# Bước 2:


Stored Procedure Properties - New Stored Procedure
General
Name:
<New Stored Procedure>
Permissions...
Owner:
Create date:
Text:
CREATE PROC spud_MaxSLVattu_200201
AS
DECLARE @sTenvtu VARCHAR(50),
@nMaxSL INT
SELECT @sTenvtu=RTRIM(TENVTU),
@nMaxSL=SLXUAT
FROM CTPXUAT CTPX
INNER JOIN VATTU VT
ON VT.MAVTU=CTPX.MAVTU
JOIN PXUAT PX
ON PX.SOPX=CTPX.SOPX
WHERE CONVERT(CHAR(7),NGAYXUAT,21)='2002-01'
AND SLXUAT = (SELECT MAX(SLXUAT)
FROM CTPXUAT)
PRINT @sTenvtu + ' coù doanh soá baùn cao nhaát'
PRINT 'Vôùi soá löôïng laø : '
+ CAST(@nMaxSL AS CHAR(10))
GO
1. Tên thủ tục
2. Nội dung thủ tục
1, 19/20
3. Kiểm tra cú pháp
Check Syntax
Save as Template
4. Nhấn OK để lưu lại
OK
Cancel
Help

# Tạo thủ tục lưu trữ bằng T-SQL

- Cú pháp:

  CREATE PROC[EDURE] <tên thủ tục> [(<DSách tham số>) ]

  [WITH RECOMPILE|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 so bắt đầu bởi @, cần chỉ ra kiểu dliệu của tham số
- Dung lượng tối đa của thủ tục lưu trữ là 128 MB.

# Thực thi các thủ tục người dùng

Lời gọi thủ tục có dạng:

<tên_thủ_tục> [<danh_sách_các_đối_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.

Trường hợp 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:

EXEC[UTE] <tên_thủ_tục> [<danh_sách_các_đối_số>]

vdụ: EXECUTE MaxSLhang_200201

*Kết quả trả về* : Số lượng 10

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>

# Tạo thủ tục lưu trữ bằng T-SQL

Ví dụ: Thủ tục có tham số

```
CREATE PROC THEMPB(@MA NCHAR(10),@TEN NVARCHAR(50))
AS
  begin
   insert into phongban(mapb,tenpb)
   values (@mapb,@tenpb)
  end
```

Thực hiện thủ tục: Thempb ‘PB50’,N’Kỹ thuật’

# Ví dụ đánh số báo danh tự động

```
create proc danhmatudong
as
begin
create table danhsach(sobd nchar(10),manv nchar(10), hoten nvarchar(50), ngaysinh datetime)
declare @ma nchar(10), @ten nvarchar(50),@ns datetime,@stt int,@i int
set @i=1
DECLARE cur_tro CURSOR FORWARD_ONLY FOR SELECT manv,hoten,ngaysinh from nhanvien
 OPEN cur_tro
       WHILE 0=0--@@FETCH_STATUS=0
       BEGIN
        FETCH NEXT FROM cur_tro
                  INTO @ma,@ten,@ns
                  IF @@FETCH_STATUS<>0
                              BREAK
                  insert into danhsach  values('SBD'+convert(nchar(7),@i),@ma,@ten,@ns)
        set @i=@i+1
    end
CLOSE cur_tro
DEALLOCATE cur_tro
End
```

Gọi thủ tục;

```
danhmatudong
```

Kiểm tra;

```
 Select * from danhsach
```

# Tạo thủ tục lưu trữ bằng T-SQL

- Ví dụ:
  - Tạo thủ tục Thêm một dự án mới
  - Cho tất cả các nhân viên thuộc phòng ‘P01’ tham gia dự án này
- Ví dụ:
  - Tạo thủ tục Thêm một dự án mới
  - Cho tất cả các nhân viên thuộc phòng ‘Triển khai dự án’ tham gia dự án này

# Tạo thủ tục lưu trữ bằng T-SQL

```
Create  proc sp_XemLuong (@Ten nvarchar(50))
as
begin
 DECLARE @mA CHAR(10),@LuongLN INT,@luongnn int, @tongluong int

SELECT @MA=Mapb from phongban where tenpb=@ten

SELECT @LuongLN=MAX(Luong), @luongnn=Min(luong),
     @tongluong=Sum(luong)
     FROM Nhanvien
     WHERE MaPB =@Ma
     Print 'Luong LN là ' + convert(varchar(10),@luongLN)
Print 'Luong NN là ' + convert(varchar(10),@luongnn)
Print N'Tổng lương là ' + cast(@tongluong  as varchar)
end
```

# Thủ tục

- *Tính mặt hàng nào có số lượng bán cao nhất trong tháng 01/2002.*

# Thủ tục

```
CREATE PROC MaxSLhang_200201 AS
DECLARE @sTenhang VARCHAR(100), @smahang char(4), @nMaxSL INT
SELECT
    @smahang=a.mahang,@sTenhang=tenhang,@nMaxSL=Sum(b.soluong)
FROM mathang a INNER JOIN chitietdathang b ON a.Mahang=b.Mahang
JOIN dondathang c ON b.SOhd=c.SOhd
WHERE CONVERT(CHAR(7),ngaydathang,21)="2002-01“
GROUP BY a.mahang, tenhang
HAVING sum(soluong)>=ALL
```

# Thủ tục

```
(SELECT sum(soluong)
 FROM (Select mathang a INNER JOIN chitietdathang b ON
    a.Mahang=b.Mahang
JOIN dondathang c ON b.SOhd=c.SOhd
WHERE CONVERT(CHAR(7),ngaydathang,21)="2002-01“
GROUP BY a.mahang, tenhang
)
PRINT @sTenhang + " có doanh số bán cao nhất,"
PRINT “Với số lượng: " + CAST(@nMaxSL AS CHAR(10))
GO
```

# VD sử dụng biến trong thủ tục:

```
CREATE PROC sp_Vidu(@malop1 NVARCHAR(10),
@malop2 NVARCHAR(10))
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, ta 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, ta 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
```

- 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 với giá trị mặc định:

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

  @<tên_tham_số> <kiểu_dữ_liệu>  =  <giá_trị_mặc_định>

```
CREATE PROC sp_TestDefault(
AS
BEGIN

@tenlop NVARCHAR(30)=NULL,
@noisinh NVARCHAR(100)='Huế')
```

```
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
```

- Cho biết họ tên của các sinh viên sinh tại *Huế*:
  sp_testdefault
- Cho biết họ tên của các sinh viên lớp *Tin K24* sinh tại *Huế*:
  sp_testdefault @tenlop='Tin K24'
- 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 *Tin K26* sinh tại Đà Nẵng:
  sp_testdefault @tenlop='Tin K26',@noisinh='Đà Nẵng'

# Biên dịch lại các thủ tục lưu trữ

- Các thủ tục lưu trữ được biên dịch lại để phản ánh sự thay đổi tới các chỉ số.
- Có ba cách để biên dịch lại các thủ tục:
  - Sử dụng thủ tục hệ thống sp_recompile system
  - Chỉ rõ WITH RECOMPILE với lệnh CREATE PROCEDURE
  - Chỉ rõ WITH RECOMPILE với lệnh EXECUTE

# Sửa các thủ tục lưu trữ

- Câu lệnh ALTER PROCEDURE được sử dụng để sửa chữa một thủ tục lưu trữ
- Cú pháp giống như lệnh CREATE PROCEDURE
- Sự thay đổi này vẫn giữ lại các quyền người dùng
- Cp:

ALTER PROCEDURE *tên_thủ_tục* [(*danh_sách_tham_số*)]
[WITH RECOMPILE|ENCRYPTION|
RECOMPILE,ENCRYPTION] AS

*<Các_câu_lệnh_Của_thủ_tục>*

# Các thông báo lỗi

- Trả về các mã hoặc lệnh RAISERROR có thể được dùng để đưa ra các lỗi của người dùng
- Trả về mã trong thủ tục lưu trữ là các giá trị nguyên
- Lệnh RAISERROR statement ghi các lỗi và gán các cấp độ nghiêm trọng của lỗi

# Xoá thủ tục

- Sử dụng câu lệnh DROP PROCEDURE với cú pháp như sau:

  DROP PROCEDURE <tên_thủ_tục >

# Tóm tắt

- 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.

# Tóm tắt <tiếp theo…>

- Có hai kiểu thủ tục lưu trữ:
  - Các thủ tục lưu trữ yêu cầu các cơ chế đối với CSDL quản trị, và cập nhật các bảng.
  - Các thủ tục người dùng định nghĩa.
- Câu lệnh CREATE PROCEDURE được sử dụng để tạo lập một thủ tục lưu trữ người dùng dịnh nghĩa.
- Câu lệnh EXECUTE được sử dụng để chạy thủ tục lưu trữ.
- Các tham số có thể được sử dụng để truyền các giá trị vào và ra từ thủ tục lưu trữ.

# Tóm tắt <tiếp theo…>

- Có ba cách để biên dịch lại các thủ tục lưu trữ:
  - Sử dụng thủ tục hệ thống sp_recompile
  - Chỉ rõ WITH RECOMPILE với lệnh CREATE PROCEDURE
  - Chỉ rõ  WITH RECOMPILE với lện EXECUTE
- Câu lệnh ALTER PROCEDURE được sử dụng để sửa chữa một thủ tục lưu trữ.
- Trả về các mã hoặc lệnh RAISERROR được sử dụng để đưa ra các lỗi của người sử dụng.

# Hàm

- Hàm là đối tượng cơ sở dữ liệu tương tự như thủ tục.
- Điểm khác biệt giữa hàm và thủ tục: Hàm trả về một giá trị thông qua tên hàm còn thủ tục thì không.
- Có thể sử dụng hàm như là một thành phần của một biêu thức (chẳng hạn, trong dsách chọn của lệnh SELECT).
- Có hàm do HQT CSDL cung cấp sẵn
- Người sử dụng có thể định nghĩa các hàm nhằm phục vụ cho mục đích riêng của mình

# Hàm hệ thống

Hàm bao gồm 3 loại

- Các hàm thao tác tập bản ghi

- Các hàm tập hợp

- Các hàm vô hướng

# Ba loại hàm

- Các hàm thao tác với tập bản ghi có thể được dùng thay cho tên các bảng trong SQL.
- Các hàm tập hợp tính toán cho ra kết quả là một giá trị đơn nhất (ví dụ tính tổng hay trung bình).
- Các hàm vô hướng thao tác trên một giá trị và trả về một giá trị. Các hàm này có thể được dùng trong các biểu thức.

# Các hàm chuyển đổi

Hàm chuyển đổi được dùng để chuyển 1 giá trị từ một kiểu dữ liệu sang kiểu dữ liệu khác. Ngoài ra nó còn được dùng để định dạng ngày tháng. SQL Server cung cấp cho ta hàm chuyển đổi duy nhất là CONVERT().

Cú pháp:

CONVERT(datatype[(length)], expression [,style])

**Ví dụ:**

SELECT 'EMP ID:' + CONVERT (CHAR(4), EMPLOYEEID) FROM EMPLOYEES

# Date Parts

| DatePart | Từ viết tắt | Giá trị |
|---|---|---|
| Hour | hh | 0-23 |
| Minute | Mi | 0-59 |
| Second | Ss | 0-59 |
| Millisecond | Ms | 0-999 |
| Day of year | Dy | 1-366 |
| Day | Dd | 1-31 |

# Date Parts (tiếp…)

| Datepart | Từ viết tắt | | Giá trị |
|---|---|---|---|
| Week | wk | | 1-53 |
| Weekday | dw | | 1-7 |
| Month | mm | | 1-12 |
| Quarter qq | | 1-4 | |
| Year | yy | | 1753-9999 |

# Các hàm ngày tháng và số học

**Các hàm ngày tháng**

**GETDATE()**
**DATEADD(datepart,number,date)**
**DATEDIFF(datepart,date1,date2)**
**DATENAME(datepart,date)**
**DATEPART(datepart,date)**

**Các hàm số học**

**ABS(num_expr)**
**CEILING(num_expr)**
**FLOOR(num_expr)**
**POWER(num_expr,y)**
**ROUND(num_expr,length)**
**Sign(num_expr)**
**Sqrt(float_expr)**

# Các hàm hệ thống

**Hàm**

DB_ID([‘database_name’])
DB_NAME([database_id])
HOST_ID()
HOST_NAME()
ISNULL(expr,value)
OBJECT_ID(‘obj_name’)
OBJECT_NAME(object_id)
SUSER_SID([‘login_name’])
SUSER_ID([‘login_name’])
SUSER_SNAME([server_user_id])
SUSER_NAME([server_user_id])
USER_ID([‘user_name’])
USER_NAME([user_id])

# Các hàm tập hợp

| **Hàm** | **Giá trị trả về** |
|---|---|
| Sum(col_name) | Trả về giá trị tổng. |
| Avg(col_name) | Trả về giá trị trung bình. |
| COUNT(*) | Hàm đếm các bản ghi trong bảng thỏa mãn điều kiện |
| Max(col_name) | Trả về giá trị lớn nhất trong một tập giá trị. |
| Min(col_name) | Trả về giá trị nhỏ nhất trong một tập hợp. |

# Định nghĩa và sử dụng hàm

Cú pháp:

```
CREATE FUNCTION tên_hàm ([danh_sách_tham_số]) RETURNS
   (kiểu_trả_về_của_hàm)
AS BEGIN
   các_câu_lệnh_của_hàm
END
```

Sửa hàm:

```
ALTER FUNCTION tên_hàm ([danh_sách_tham_số]) RETURNS
   (kiểu_trả_về_của_hàm)
AS BEGIN
   các_câu_lệnh_của_hàm
END
```

# VD: Định nghĩa hàm tính ngày trong tuần (thứ) của một giá trị kiểu ngày

```
CREATE FUNCTION thu(@ngay DATETIME) RETURNS NVARCHAR(10)
AS
   BEGIN
DECLARE @st NVARCHAR(10)
      SELECT @st=CASE DATEPART(DW,@ngay)
                  WHEN 1 THEN 'Chu 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
```

# Sử dụng hàm

- Sử dụng như hàm do hqt csdl cung cấp:
- SELECT masv,hodem,ten,dbo.thu(ngaysinh),ngaysinh
  FROM sinhvien
  WHERE malop=’C24102’

# Hàm với giá trị trả về là “dữ liệu kiểu bảng”

```
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"

## Các qui tắc:

- 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).

# Hàm với giá trị trả về là "dữ liệu kiểu bảng"

## VD: Định nghĩa hàm *func_XemSV*

```
CREATE FUNCTION func_XemSV(@khoa SMALLINT) RETURNS TABLE
AS
    RETURN(SELECT masv,hodem,ten,ngaysinh
   FROM sinhvien INNER JOIN lop
            ON sinhvien.malop=lop.malop
   WHERE khoa=@khoa)
```

# Hàm với giá trị trả về là “dữ liệu kiểu bảng”

## Dùng hàm đã định nghĩa:

- Để biết danh sách các sinh viên khoá 25,  ta sử dụng câu lệnh như sau:

```
SELECT * FROM dbo.func_XemSV(25)
```

- Còn câu lệnh dưới đây cho ta biết được danh sách sinh viên khoá 26

```
SELECT * FROM dbo.func_XemSV(26)
```

# Hàm với giá trị trả về là "dữ liệu kiểu bảng"

- Khi cần phải sử dụng nhiều câu lệnh trong phần thân hàm, cú pháp đnghĩa hàm:
- CREATE FUNCTION <*tên_hàm*>([<*danh_sách_tham_số*>]) RETURNS @<*biến_bảng*> TABLE <*định_nghĩa_bảng*>

AS

BEGIN

<*các_câu_lệnh_trong_thân_hàm*>

RETURN

END

# Hàm với giá trị trả về là "dữ liệu kiểu bảng"

## 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"

Ví dụ:

Tạo hàm có một tham số là @maphongban nchar(10), thực hiện thống kê số nhân viên theo mã phòng ban nhập vào, nếu giá trị biến @maphongban nhập vào là kí tự trống hoặc Null thì thống kê nhân viên theo từng phòng ban

```
CREATE FUNCTION Func_TongNV(@mapb nchar(10))
    RETURNS @bangthongke TABLE
    (
        mapb    NCHAR(10),
        tongsonv    INT
    ) AS
```

# Hàm với giá trị trả về là “dữ liệu kiểu bảng”

```
 BEGIN
  if (@mapb is null) or (@mapb='' )
  INSERT INTO @bangthongke
          SELECT mapb,COUNT(manv)
          FROM nhanvien
          GROUP BY mapb
else
      INSERT INTO @bangthongke
       SELECT mapb,COUNT(manv)
          FROM nhanvien
          WHERE mapb=@mapb
          GROUP BY mapb
   RETURN /*Trả kết quả về cho hàm*/
   END
```

# Hàm với giá trị trả về là “dữ liệu kiểu bảng”

SELECT * FROM dbo.func_TongNV(‘PB01’)

Sẽ cho kết quả thống kê tổng số nhân viên của phòng ban ‘PB01’

Còn câu lệnh:

SELECT * FROM dbo.func_TongNV(‘’)

Cho ta biết tổng số nhân viên của mỗi phòng ban

# Hàm

- Bài tập 1:

  Tạo hàm có một tham số là @tenphongban nvarchar(50), thực hiện thống kê số nhân viên theo tên phòng ban nhập vào, nếu giá trị biến @tenphongban nhập vào là kí tự trống hoặc Null thì thống kê nhân viên cho từng phòng ban

- Bài tập 2:

  Tạo hàm thống kê số nhân viên tham gia dự án cho một phòng ban có tên là gì đó. Nếu không nhập vào tên thì thống kê cho từng phòng ban

# Các trigger

# Sử dụng Trigger

- So sánh kiểu dữ liệu.
- Đọc dữ liệu từ các bảng nằm trong cơ sở dữ liệu khác.
- Thay đổi theo tầng hoặc xoá liên tục các bảng liên quan trong một cơ sở dữ liệu
- Huỷ bỏ các thay đổi không đúng
- Tuân theo ràng buộc mà việc bắt lỗi bằng ràng buộc CHECK khó thực hiện
-

# Khái niệm về trigger

- Trigger là một kiểu thủ tục được kích hoạt tự động theo các sự kiện (events).
- Có 02 lọai triggers:
  + Data Modification Language –DML (For | After triggers, Instead-of triggers)
  + Data Definition Language - DDL triggers (For | After triggers)

# Data Definition Language (DDL) Trigger

Cú pháp:

CREATE TRIGGER trigger_name ON { ALL SERVER | DATABASE } [ WITH [ ENCRYPTION ] | [ EXECUTE AS CALLER | SELF | 'user_login' ] { FOR | AFTER } { event_type | event_group } [ ,...n ] AS { sql_statement [ ; ] [ ...n ] }

Execute As Caller là option mặc định.

Execute As User = 'user'

Cú pháp: DROP TRIGGER *trigger_name* [ **,**...*n* ] ON { DATABASE | ALL SERVER }

DISABLE TRIGGER { [ *schema* . ] *trigger_name* [ **,***...n* ] | ALL } ON {DATABASE | ALL SERVER } [ ; ]

DDL triggers là các triggers được tự động gọi sau khi máy thực hiện các lệnh sau:

Create Table, Drop Table, Alter Procedure, Drop Schema, Create Login, …

(Xem BOL: DDL triggers, events used for firing)

# DML Triggers

- **Cú Pháp:**

  -Tạo  CREATE TRIGGER <trigger_name> ON
  <table_name>|<view_name>

  [With encryption|EXECUTE AS { CALLER | SELF | '*user_name*' } ]

  {[FOR| AFTER] | Instead of [insert],[update],[delete] }
  AS Transact-SQL statements

  - Xoá Drop Trigger <trg_name>
  - DISABLE TRIGGER { *trigger_name* [ ,*...n* ] | ALL } ON  *object_name*

    ON: Chỉ ra rằng Trigger đang được viết cho bảng hoặc view nào.

    With encryption: nội dung của trigger sẽ được mã hóa.

# Các dạng hoạt động của DML trigger

- AFTER (FOR): các câu lệnh bên trong trigger sẽ được thực hiện sau khi các sự kiện tạo nên trigger đã xảy ra rồi.
- INSTEAD OF: sẽ bỏ qua sự kiện đã kích hoạt trigger mà thay vào đó sẽ thực hiện các dòng lệnh SQL bên trong Trigger
- *Ví dụ:* ta có một Update trigger trên một table với câu INSTEAD OF: Khi đó nếu ta thực hiện việc update dữ liệu trong bảng thì thay vì update dữ liệu, SQL Server sẽ thực hiện các lệnh đã được viết sẵn bên trong trigger.

# Các kiểu Trigger

- Trigger Insert: Trigger được phát biểu bởi For insert. Trigger được thực hiện khi tiến hành thêm một mẫu tin vào bảng. Mẫu tin cần thêm sẽ được lưu trong một bảng tạm có tên là Inserted.
- Trigger Delete: Trigger được phát biểu bởi For delete. Trigger được thực hiện khi tiến hành xóa một mẫu tin trong bảng. Mẫu tin bị xóa sẽ được lưu trong một bảng tạm có tên là deleted.
- Trigger Update: Trigger được phát biểu bởi For update. Trigger được thực hiện khi tiến hành sửa một mẫu tin trong bảng. Mẫu tin bị thay đổi sẽ được lưu trong 2 bảng tạm có tên là Inserted (chứa giá trị mới) và Deleted (chứa giá trị cũ).

# Chú ý

- Trigger không thể được tạo ra trên bảng tạm thời hay bảng hệ thống. Trigger chỉ có thể được kích hoạt một cách tự động bởi một trong các event Insert, Update, Delete. Có thể áp dụng trigger cho View.
- Inserted và Deleted  là 2 table tạm chỉ chứa trên bộ nhớ và chỉ có giá trị bên trong trigger mà thôi (nghĩa là chỉ nhìn thấy được trong trigger mà thôi). Ta có thể dùng thông tin trong 2 table này để so sánh dữ liệu cũ và mới hoặc kiểm tra xem dữ liệu mới.

# Trigger dạng For, Update

--Tạo trigger trên bảng nhanvien cho su kien
--insert, trigger thuc hien thong bao manv vua them

```
CREATE TRIGGER THEMNV ON NHANVIEN FOR
INSERT
AS
 DECLARE @MA NCHAR(10)
BEGIN
 SELECT @MA=MANV FROM INSERTED
 PRINT 'Ma nhan vien vua them la '+@ma
END
```

# Trigger dạng For, Update

Ví dụ: Tạo trigger trên bảng Nhanvien cho sự kiện insert. Để khi thêm nhân viên thì tự động cho nhân viên tham gia tất cả các dự án.

```
Create TRIGGER DBO.ADDNV ON NHANVIEN FOR INSERT
AS
 DECLARE @MANV NCHAR(10)
BEGIN
    SELECT @MANV=MANV FROM INSERTED
    INSERT INTO PHANCONG(MADA,MANV) SELECT MADA,@MANV FROM
    DUAN
END
```

Bài tập: Thêm 1 nhân viên mới, cho nhân viên này tham gia tất cả các đề án mà phòng ban của nhân viên này phụ trách.

# Trigger dạng INSTEAD OF – Thay thế

Dạng INSTEAD OF sẽ bỏ qua sự kiện đã kích hoạt trigger mà thay vào đó sẽ thực hiện các dòng lệnh SQL bên trong Trigger

INSTEAD OF được chia làm 3 loại nhỏ:

INSTEAD OF INSERT, INSTEAD OF UPDATE và INSTEAD OF DELETE.

# Trigger dạng INSTEAD OF – Thay thế

Ví dụ: Viết trigger để khi xóa nhân viên, thực hiện xóa thông tin tham gia dự án của nhân viên

```
Create TRIGGER XOANV ON NHANVIEN INSTEAD OF
DELETE
AS
 DECLARE @MANV NCHAR(10)
BEGIN
        SELECT @MANV=MANV FROM DELETED

        DELETE PHANCONG
        WHERE  MANV = @MA

        DELETE NHANVIEN
        WHERE MANV=@MA
END
```

# Trigger dạng FOR

```
CREATE TRIGGER Ktra_DonGia1 ON [dbo].[Products]
AFTER INSERT
AS
  If exists( Select * from inserted i where
    i.dongiamua>i.dongiaban)
Begin
        RollBack tran
        RaisError ('Khong hop le', 16,1)
End
```

# Hàm Update() trong các trigger

```
CREATE TRIGGER Ktra_DonGia2 ON [dbo].[Products]
FOR  UPDATE AS
If Update(dongiamua) or Update(dongiaban)
Begin
    If exists( Select 'true' from inserted i where
    i.dongiamua>i.dongiaban)
    Begin
        RollBack tran
        RaisError ('Khong hop le', 16,1)
   End
End
```

# Debug trigger

Để gỡ rối một trigger chúng ta phải viết một procedure có gọi các thao tác làm phát sinh sự kiện để thực thi trigger. Ví dụ để kiểm tra các trigger Ktra_DonGia1 và Ktra_DonGia2 chúng ta viết thủ tục sau

```
CREATE PROCEDURE dbo.DebugTrigger  as
   insert into Products(Masp, Tensp, Mota, Donvitinh, Dongiamua, Dongiaban, VAT)
  Values ( 'SP-01', ‘San Pham-01 ' , '@Mota' ,'@donvi', 20,10 , 1 )
GO
```

# Tổng kết

- Các trigger là các thủ tục lưu mà được thực hiện tự động để tương tác với các tác vụ thêm, cập nhật và xoá trên một bảng.
- Các trigger thường được sử dụng để thực hiện các quy tắc nghiệp vụ đòi hỏi.
- Lệnh CREATE TRIGGER được sử dụng để tạo một trigger.
- Các trigger truy nhập tới các bảng logic **Inserted** và **Deleted**. Các bảng này chứa các hình ảnh của dữ liệu trước đó, và sau quá trình cập nhật.

# Tổng kết

- Các kiểu trigger:
  - INSERT: Thực hiện mỗi khi xuất hiện việc thêm mới dữ liệu vào bảng. Các trigger này đảm bảo rằng dữ liệu được chèn vào bảng là hợp lệ.
  - UPDATE: Thực hiện khi một tác vụ cập nhật xảy ra trên một bảng. Các trigger này có thể được thi hành ở mức bảng hoặc mức cột.
  - DELETE: Thực hiện khi dữ liệu được xoá khỏi một bảng.

# Thảo luận

Câu hỏi:

1. Cấu trúc lệnh trong T_SQL?
2. Khi nào sử dụng từ khóa Go?
3. Khi nào sử dụng con trỏ? Khai báo, định nghĩa, sử dụng con trỏ?
4. Khi nào sử dụng thủ tục Store Procedure? Khai báo, định nghĩa, lời gọi?
5. Khi nào sử dụng thủ tục hàm Function? Khai báo, định nghĩa, lời gọi?
6. Khi nào sử dụng thủ tục Trigger? Khai báo, định nghĩa, kích hoạt trigger?

# Thảo luận

- Thảo luận nhóm:

  Thảo luận phần lập trình T_SQL trên bài tập lớn của nhóm

# Ôn tập

Ôn tập các nội dung trong chương 1, 2, 3, 4, 5,