VIEW, PROCEDURE, FUNCTION & TRIGGER

Lương Trần Hy Hiến

Nội dung

- View
- Stored procedure
- Function
- Trigger
- Transaction

View

- Một câu lệnh SQL SELECT được đặt tên để đơn giản hóa việc sử dụng.
- Với những câu lệnh truy vấn phức tạp (kết nối nhiều bảng, nhiều điều kiến, truy vấn con…) chúng ta nên tạo một view chứa nó để khi sử dụng được đơn giản hơn.

TẠO VIEW

```
CREATE VIEW <Tên View>
AS
      <CÂU LỆNH SELECT SQL>
```

SỬ DỤNG VIEW

```
SELECT * FROM <Tên View>
```

Ví dụ View

- View sau tạo một view tên HangHoaView chứa tất cả hàng hóa có cả tên loại và tên công ty cung cấp chúng.

TẠO VIEW

```
CREATE VIEW HangHoaView
AS
    SELECT hh.*, TenLoai, TenCongTy FROM HangHoa hh
        JOIN Loai lo ON lo.MaLoai=hh.MaLoai
            JOIN NhaCungCap ncc ON ncc.MaNCC=hh.MaHH
```

SỬ DỤNG VIEW

```
SELECT * FROM HangHoaView
```

Results | Messages

	MaHH	TenHH	MaLoai	DonGia	MaNCC	DacBiet	SoLuong	TenLoai	TenCongTy
1	1001	Aniseed Syrup	1000	190	AP	1	55	Clocks	Apple
2	1002	Change	1000	19	AP	0	100	Clocks	Apple
3	1003	Aniseed Syrup	1001	10	AP	0	100	Laptops	Apple
4	1004	Chef Anton's Cajun Seasoning	1001	22	AP	0	100	Laptops	Apple
5	1005	Chef Anton's Gumbo Mix	1001	21.35	AP	0	100	Laptops	Apple
6	1006	Grandma's Boysenberry Spread	1001	25	AP	1	100	Laptops	Apple

STORE PROCEDURE (PROC)

- Thủ tục lưu là một mô-đun code thực hiện một nhiệm vụ cụ thể. Mô đun hóa và tận dụng khả năng lập trình của SQL là mục đích chính của PROC.
- Tham số của thủ tục lưu gồm 2 loại IN và OUT
  - IN: là tham số truyền vào - dữ liệu đầu vào cần thiết cho tham số
  - OUT: là tham số ra được thủ tục lưu thiết lập giá trị để truyền ra ngoài.
- Chúng ta cũng có thể đặc giá trị mặc định cho tham số IN. Khi gọi thủ tục ma không truyền thì thủ tục sẽ sử dụng các giá trị mặc định này.

```
CREATE PROCEDURE <tên thủ tục lưu>
        [(<danh sách tham số>)]
AS
        <Các câu lệnh của thủ tục>
```

Example


```
CREATE PROCEDURE sp_HangHoa
(
    @MaLoai INT,
    @MaNCC NVARCHAR(50)
)
AS
BEGIN
    SELECT * FROM HangHoa
        WHERE MaLoai=@MaLoai AND MaNCC=@MaNCC
END
```

CREATE PROCEDURE

```
EXECUTE sp_HangHoa 1005, 'SS'
```

CALL PROCEDURE

THAM SỐ PROC

```
CREATE PROCEDURE sp_ThamSo
(
     @SoBanGhi INT OUTPUT, -- tham số ra
     @MaLoai INT=1000 -- tham số vào, mặc định
)
AS
BEGIN
     SELECT * FROM HangHoa WHERE MaLoai=@MaLoai
     --thiết lập giá trị cho tham số ra
     SET @SoBanGhi=@@ROWCOUNT
END
```

Không truyền tham số @MaLoai cho PROC

```
DECLARE @x INT
EXECUTE sp_ThamSo @x OUT
PRINT @x
```

Results | Messages

	MaHH	TenHH	MaLoai	MoTaDonVi
1	1001	Aniseed Syrup	1000	10 boxes x 2
2	1002	Change	1000	10 boxes x 2
3	1024	Guaranaj Fantajstica	1000	12 - 355 ml c
4	1034	Sasquatch Ale	1000	24 - 12 oz bc
5	1025	Steeleye Stout	1000	24 - 12 oz bc

Results | Messages

```
(14 row(s) affected)
14
```

PHÂN TRANG BÊN TRONG CSDL

```
ALTER PROC sp_FindByName     --tìm theo tên, có phân trang
(
    @RowCount INT OUTPUT,    --nhận tổng số bản ghi truy vấn được
    @Name NVARCHAR(100)='', --tên hàng hóa cần tìm
    @StartRow INT=1,         --bắt đầu từ bản ghi thứ
    @Length INT=20           --số lượng bản ghi cần lấy
)
AS
BEGIN
    WITH TempTable AS --khai báo bảng chứa dữ liệu tạm có chỉ số hàng
    (
        SELECT *, ROW_NUMBER() OVER (ORDER BY MaHH) AS RowIndex
            FROM HangHoa WHERE TenHH LIKE '%'+@Name+'%'
    )
    --chỉ nhận các bảng ghi từ vị trí @StartRow đến @StartRow + @Length
    SELECT * FROM TempTable
        WHERE RowIndex BETWEEN @StartRow AND @StartRow + @Length - 1
    --thiết lập giá trị cho tham số ra @RowCount
    SELECT @RowCount=COUNT(*) FROM HangHoa WHERE TenHH LIKE '%'+@Name+'%'
END
```

- Tìm các hàng hóa có chứa “on” và lấy 7 mặt hàng tính từ bản ghi thứ 3.
- Đoạn mã cũng xuất ra tổng số bản ghi được tìm thấy

```
DECLARE @TSHH INT
EXEC sp_FindByName @TSHH OUT, 'on' ,3, 7
PRINT 'TONG SO HANG HOA: ' + CAST(@TSHH AS VARCHAR(50))
```

Results | Messages

	MaHH	TenHH	MaLoai	MoTaDonVi
1	1013	Konbu	1007	2 kg box
2	1017	Alice Mutton	1005	20 - 1 kg tins
3	1018	Carnarvon Tigers	1007	16 kg pkg.
4	1021	Sir Rodney's Scones	1002	24 pkgs. x 4 pieces
5	1031	Gorgonzola Telino	1003	12 - 100 g pkgs
6	1032	Mascarpone Fabioli	1003	24 - 200 g pkgs.
7	1040	Boston Crab Meat	1007	24 - 4 oz tins

Results | Messages

```
(7 row(s) affected)
TONG SO HANG HOA: 12
```

User-defined function

- Chúng ta đã biết một số hàm đã xây dựng sẵn về xử lý chuỗi, ngày, tổng hợp số liệu…
- Thông thường hàm do người dùng định nghĩa được viết ra để thực hiện việc tính toán và cho kết quả và được sử dụng như các hàm dựng sẵn.

```
CREATE FUNCTION <tên hàm>([danh sách tham số])
RETURNS <kiểu dữ liệu trả về>
AS
BEGIN
    <câu lệnh thực hiện>
    RETURN <giá trị trả về>
END
```

Ví dụ hàm


```
CREATE FUNCTION fnDoanhSo
(
    @MaHH INT
)
RETURNS FLOAT
AS
BEGIN
    DECLARE @DoanhSo FLOAT

    SELECT @DoanhSo=SUM(SoLuong*DonGia*(1-GiamGia))
        FROM ChiTietHD WHERE MaHH=@MaHH

    RETURN @DoanhSo
END
```

```
SELECT MaHH, dbo.fnDoanhSo(MaHH) FROM HangHoa
```

Trigger

- Trigger là các thủ tục lưu đặc biệt
  - Không có tham số
  - Kết hợp với từng bảng cụ thể
  - Thực hiện tự động khi xảy ra các sự kiện INSERT, UPDATE, DELETE
- Trong trigger tồn tại ***2 bảng đặc biệt***
  - INSERTED chứa các bản ghi vừa được chèn vào bảng
  - DELETED chứa các bản ghi vừa bị xóa khỏi bảng
- Hành động UPDATE ảnh hưởng đến cả 2 bảng đặc biệt. Nhưng các hành động INSERT và DELETE chỉ ảnh hưởng đến một trong hai bảng mà thôi.

```
CREATE TRIGGER <tên trigger>  ON  <tên bảng sở hữu>
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
        <xử lý sự kiện>
END
```

Ví dụ Trigger

- Tạo trigger để cập nhật số lượng tồn kho của các mặt hàng được mua. Tức khi khách chọn mua thì số hàng bị giảm xuống với số lượng tương ứng.
- Mặt hàng chọn mua sẽ được chèn vào bảng ChiTietHD nên trigger phải kết hợp với sự kiện INSERT của bảng ChiTietHD.

```
CREATE TRIGGER tg_CapNhatSoLuongTonKho
   ON  ChiTietHD
   AFTER INSERT
AS
BEGIN
    DECLARE @SoLuong INT, @MaHH INT
    --lấy số lượng và mã hàng hóa vừa được chọn mua
    SELECT @SoLuong=SoLuong, @MaHH=MaHH FROM INSERTED
    --cập nhật số lượng tồn kho
    UPDATE HangHoa SET SoLuong=SoLuong-@SoLuong WHERE MaHH=@MaHH
END
```

Ví dụ Trigger

```
CREATE TRIGGER tg_CapNhatThanhTien
    ON  ChiTietHD
    AFTER INSERT, UPDATE, DELETE
AS
BEGIN
     DECLARE @MaHD INT, @ThanhTien FLOAT;
     -- lấy mã hàng hóa vừa thêm, sửa hay xóa
     WITH BangTam AS
     (
          SELECT MaHD FROM INSERTED
          UNION
          SELECT MaHD FROM DELETED
     )
     SELECT @MaHD=MaHD FROM BangTam
     --tính tổng tiền của hóa đơn trên bảng ChiTietHD
     SELECT @ThanhTien=SUM(SoLuong*DonGia*(1-GiamGia))
          FROM ChiTietHD WHERE MaHD=@MaHD
     --cập nhật thành tiền của hóa đơn
     UPDATE HoaDon SET ThanhTien=@ThanhTien
          WHERE MaHD=@MaHD
END
```

Hủy hành động thao tác dữ liệu

- Sử dụng lệnh ROLLBACK TRANSACTION để hủy thao tác dữ liệu vừa thực hiện.
- Ví dụ sau không cho phép xóa các bản ghi trong bảng mathang

```
CREATE TRIGGER trg_mathang_delete
ON mathang FOR DELETE AS
ROLLBACK TRANSACTION
```

Transaction

- "None or All" (được ăn cả, ngã về không) là khẩu hiệu của transaction. Nghĩa là khi bạn thực hiện một ***khối lệnh*** SQL, nếu bạn muốn sai một trong các lệnh trong khối thì hủy bỏ tất cả các lệnh khác và ngược lại thì mới chấp nhận kết quả thì hãy áp dụng transaction.
- Các ứng dụng
  - Chuyển khoản
  - Tạo đơn hàng

```
BEGIN TRAN <tên transaction>;
BEGIN TRY
    <khối lệnh sql>
    COMMIT TRAN <tên transaction>;
END TRY
BEGIN CATCH
    ROLLBACK TRAN <tên transaction>;
END CATCH
```

Ví dụ transaction

- Khối mã lệnh sau sẽ không thực hiện được bất kỳ một lệnh nàovì MaKhoa (khóa chính) bị trùng ở hàng màu đỏ

```
BEGIN TRAN TRS_KHOA;
BEGIN TRY
  INSERT INTO Khoa(MaKhoa, TenKhoa, DienThoai) VALUES('K0001', 'Kinh Te', '0913745789');
  INSERT INTO Khoa(MaKhoa, TenKhoa, DienThoai) VALUES('K0002', 'CNTT', '0918355888');
  INSERT INTO Khoa(MaKhoa, TenKhoa, DienThoai) VALUES('K0003', 'Dien Tu', '0917060606');
  INSERT INTO Khoa(MaKhoa, TenKhoa, DienThoai) VALUES('K0001', 'Hoa Hoc', '0918696969');
  INSERT INTO Khoa(MaKhoa, TenKhoa, DienThoai) VALUES('K0005', 'Nano', '0987456789');
  COMMIT TRAN TRS_KHOA;
END TRY
BEGIN CATCH
  ROLLBACK TRAN TRS_KHOA;
END CATCH
```

Bài tập

Lập trình với T_SQL

- Khai báo biến:

  DECLARE @Tên_Biến Kiểu_Dữ_Liệu
- Ví dụ:

  DECLARE @Tuoi int

  DECLARE @MSSV varchar(5)

  DECLARE @numCount int
- Tên biến: Bắt đầu bởi @
- Kiểu dữ liệu của biến: Lấy kiểu dữ liệu hệ thống, trừ kiểu text, ntext, image

Gán giá trị cho biến

- Cách 1:

```
SET @Tên_Biến = Giá_Trị
```

- Ví dụ:

```
DECLARE @HoTen nvarchar(20)
SET @HoTen = N‘Nguyễn Hằng Nga’
```

Gán giá trị cho biến

- Cách 2:

```
SELECT @Tên_Biến = Giá_Trị
```

- Ví dụ:

```
DECLARE @HoTen nvarchar(20)
SELECT @HoTen = N‘Nguyễn Hằng Nga’
```

Gán giá trị cho biến

- Cách 3:

```
SELECT @Tên_Biến = Tên_cột
  FROM Tên_Bảng
```

- Ví dụ: Tìm lương lớn nhất của tất cả nhân viên:

```
DECLARE @MaxSalary decimal(18,2)
SELECT @MaxSalary = MAX(Luong)
FROM NhanVien
```

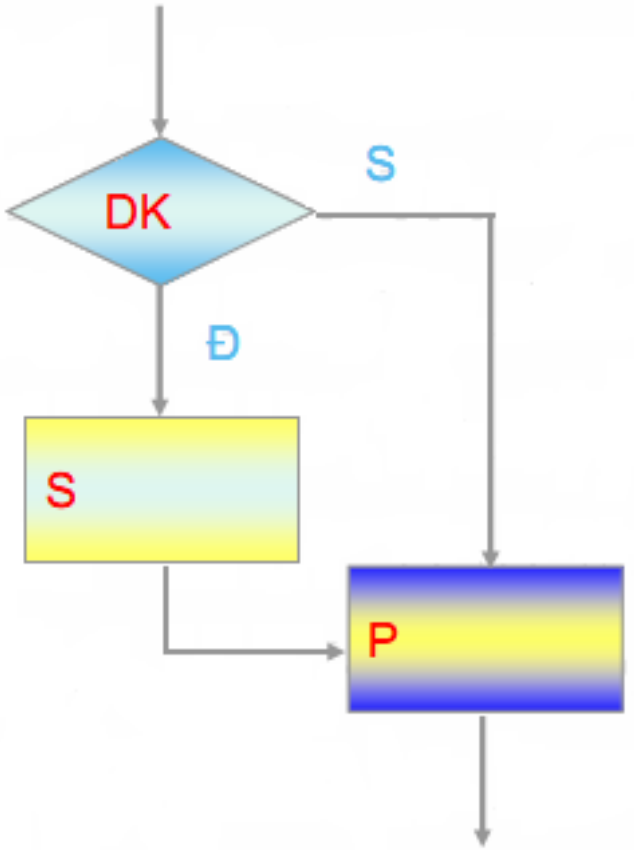
Cấu trúc điều khiển

- Cấu trúc điều kiện:

Nếu (biểu thức điều kiện) thì

Lệnh/Khối lệnh

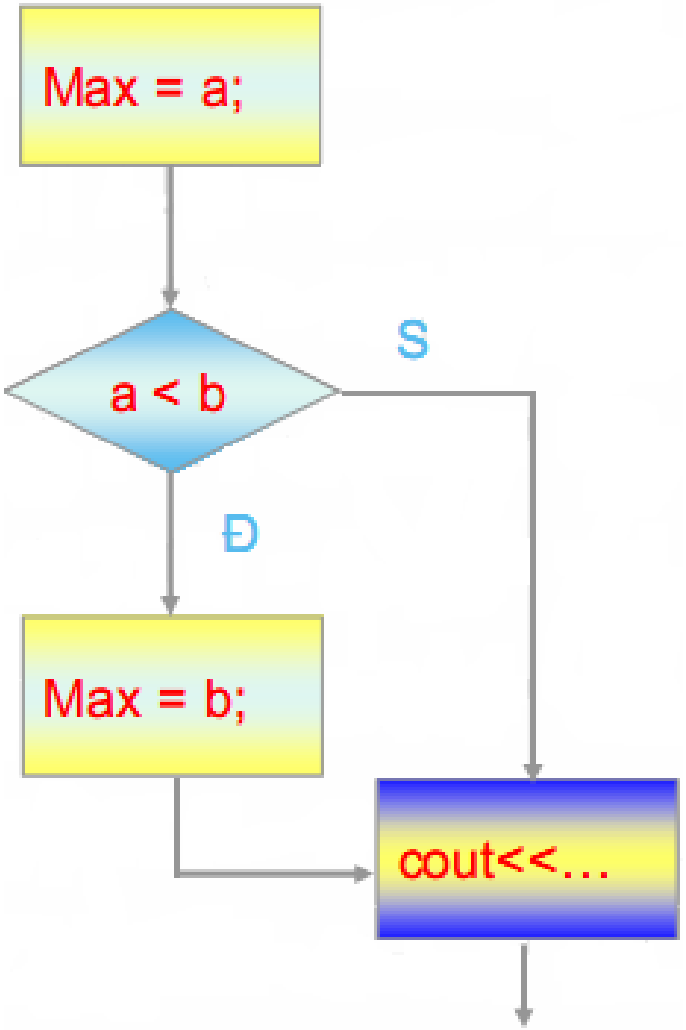
```
IF (biểu thức điều kiện)
BEGIN
   Lệnh/Khối lệnh S
        hoặc SQL Statement
END
   Lệnh/Khối lệnh P
   hoặc SQL Statement
```



Cấu trúc điều kiện

- Tìm Max 2 số

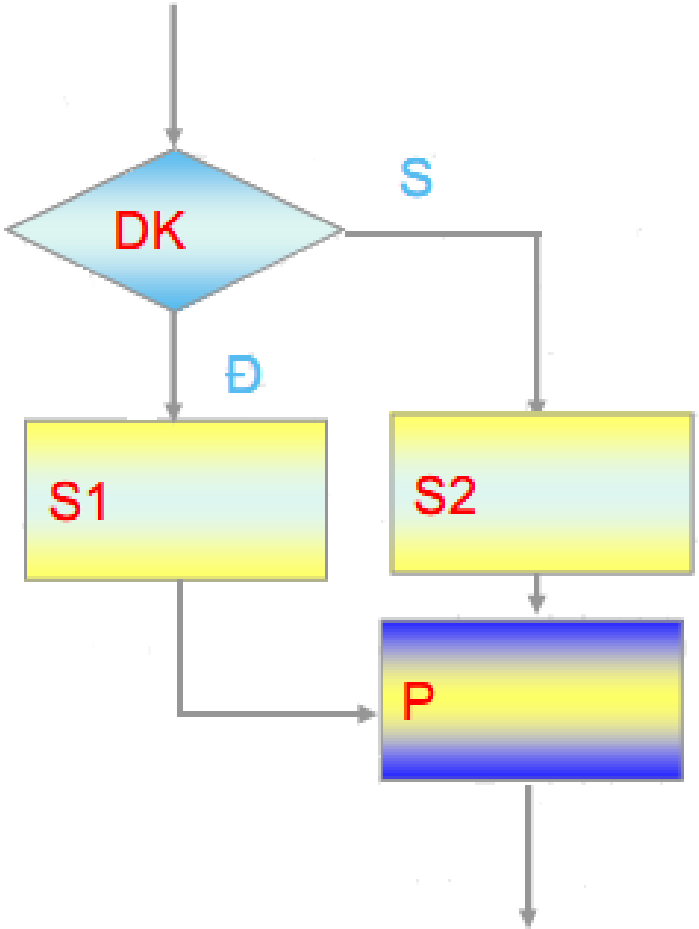
```
DECLARE @a, @b, @Max int
SET @Max = @a
IF (@a < @b)
BEGIN
   SET @Max = @b
END
Print @Max
```



Cấu trúc điều kiện

- Nếu (biểu thức điều kiện) thì
  Lệnh/Khối lệnh S1
- Ngược lại
  Lệnh/Khối liệnh S2

```
IF (biểu thức điều kiện)
BEGIN
   Lệnh/Khối lệnh S1
END
ELSE
BEGIN
   Lệnh/Khối lệnh S
END
Lệnh/Khối lệnh P
```



Cấu trúc điều kiện

- Tìm Max 2 số

```
DECLARE @a, @b, @Max int
IF (@a < @b)
BEGIN
   SELECT @Max = @b
END
ELSE
BEGIN
   SELECT @Max = @a
END
Print @Max
```

Cấu trúc CASE

- Cho phép kiểm tra điều kiện và xuất thông tin theo từng trường hợp
- Cú pháp 1

```
CASE <tên cột>/<biểu thức>
    WHEN <giá trị> THEN <biểu thức>
    WHEN <giá trị> THEN <biểu thức>
    …
    [ELSE <biểu thức>]
END
```

Cấu trúc CASE

- Cú pháp 2

```
CASE   WHEN <giá trị> THEN <biểu thức>
    WHEN <giá trị> THEN <biểu thức>
    …
    [ELSE <biểu thức>]
END
```

Cấu trúc lặp


```
WHILE (bieuthuc logic)
BEGIN
     --Lenh/Khoi lenh
END
```

- Viết chương trình tính tổng s = 1 + 2 + .. + n

```
DECLARE @i INT
DECLARE @N INT
DECLARE @S INT
SELECT @i=1, @s=0
WHILE (@i <= @N)
BEGIN
     SELECT @S = @S + @i
     SELECT @i = @i + 1
END
PRINT @S
```

Cấu trúc lặp

- BREAK: Thoát khỏi vòng lặp WHILE
- CONTINUE: Thực hiện lần lặp mới

Một số biến toàn cục

- @@ERROR: giá trị lỗi về trong biến toàn cục
  - @@ERROR= 0: không xảy ra lỗi
  - @@ERROR <> 0: xảy ra lỗi với mã lỗi là @@ERROR
- @@IDENTITY
- @@ROWCOUNT