

CHƯƠNG 4. DATA MANIPULATION LANGUAGE (DML)

Ngôn ngữ thao tác dữ liệu có 4 lệnh chính:

- Select
- Insert
- Update
- Delete

I. LỆNH SELECT : Chọn ra các mẫu tin từ 1 hay nhiều table.

Cú pháp:

Select [Distinct] [Top n / Top n Percent]

danhsach_cot

[INTO # | ## <table ảo>]

From *danhsach_table*

[Where điều kiện]

[Group by cột_nhóm [Having điều kiện]]

[Order By <cột> [DESC] [, <cột> [DESC]]...]

[Compute AggregateFunction(cột)]

*** Các toán tử dùng trong điều kiện:**

> >= (!<) < <= (!>) = <> (!=)

Is Null : Ghichu Is Null

Is Not Null : Ghichu Is Not Null

Between ... And ... : Luong Between 8000000 AND 15000000

In (danhsach các trị) : Loai IN ('N', 'X')

Like : _ đại diện 1 ký tự : Sodat Like '091%5'

% đại diện 1 string

Not And Or

* Một cột trong lệnh Select có thể là: 1 field trong table, 1 hằng, 1 biểu thức, hay 1 hàm aggregate function (Count, Sum, Avg, Max, Min), Select-Stmt

* Một table trong *danhsach_table* của lệnh Select có thể là 1: table hệ thống, user table, table ảo, view, UDF, Select-Stmt.

Ví dụ: Select 'So cac nhan vien trong cong ty ', Count(*)

From Nhanvien

Ví dụ: Hãy chọn ra các phiếu nhập hàng mà nhân viên có mã là 1 đã lập

Select * From Phatsinh

Where Loai='N' and MANV = 1

* **SubQuery:** để thi hành 1 SubQuery ta dùng các toán tử so sánh, và một số toán tử sau: Exists, Not Exists, ALL , ANY, IN

Ví dụ 1: Chọn ra các nhân viên chưa từng lập phiếu trong công ty

Select *

From Nhanvien

Where MANV Not IN (Select MANV

From Phatsinh)

* **Biểu thức Case**: SQL cung cấp cấu trúc Case để thay thế 1 trị có sẵn trong cơ sở dữ liệu bằng 1 biểu thức khác. Cấu trúc Case có dạng sau:

Case

When đk1 **Then** Expr1

When đk2 **Then** Expr2

...

When đkn **Then** Exprn

Else Expr_x

End

Ví dụ 2: Hãy hiển thị thêm cột LoaiNV để xếp loại nhân viên dựa vào doanh số bán hàng trong tháng 3/2016 của từng nhân viên theo quy tắc sau:

Doanh số < 20000000 -> LoaiNV =1

Doanh số >=20000000 và <50000000 -> LoaiNV =2

Doanh số >=50000000 -> LoaiNV =3

SELECT MANV, DOANHSCO=SUM (THANH TIEN)

INTO #TAM

FROM PHATSINH

WHERE LOAI ='X' **AND** MONTH(NGAY) = 3 **AND** YEAR (NGAY) = 2016

GROUP BY MANV

SELECT #TAM.* ,

LOAINV = (Case

When DOANHSCO < 20000000 **Then** 1

When DOANHSCO < 50000000 **Then** 2

Else 3

End)

FROM #TAM

* Lưu ý: Trong trường hợp ta đã tạo các Cluster Index hoặc Noncluster Index, và ta muốn liệt kê các records theo thứ tự đã chỉ định trong các Index thì ta viết câu lệnh Select như sau: Select * from table WITH (INDEX = ten_index)

II. DÙNG DML ĐỂ HIỆU CHỈNH DỮ LIỆU:

1. Lệnh Insert : thêm 1 record mới vào table

Cú pháp:

Insert Into <table> (danh sách field)

Values (Danh sách các giá trị)

Ví dụ 3: Lệnh Insert sau sẽ copy tất cả các record từ 1 version cũ của table Nhanvien vào version mới của Nhanvien (có thêm field NoiSinh với giá trị là ‘ ‘)

Insert Into Nhanvien (MANV, HO, TEN, NOISINH)

Select MANV, HO, TEN, ‘ ‘

From CustOld

Lưu ý: Lệnh Select Into để tạo ra 1 table mới có các mẫu tin lấy từ 1 hoặc nhiều tables.

Ví dụ 4: Đưa các mã nhân viên có doanh số bán hàng từ 50000000 trở lên trong tháng 3/2016 vào 1 bảng riêng tên NV_DOANHSCO_CAO

SELECT *

INTO NV_DOANHSCO_CAO

FROM #TAM **WHERE** DOANHSCO >=50000000

2. Lệnh Update: để thay đổi giá trị của 1 hay nhiều cột trong table

Ví dụ 5: Hãy đổi tên của nhân viên mã 1 sang tên mới Huỳnh Văn Diệp

Update Nhanvien

Set Name ='Huỳnh Văn Diệp',

Where MANV =1

3. Lệnh Delete:

Ví dụ 14: Xóa nhanvien có mã 2

Delete

From Nhanvien Where Manv = 2

Ta có thể xóa toàn bộ dữ liệu trong table:

Delete From <tên Table>

hay

Truncate Table <tên Table>

Lưu ý: Muốn xóa hẳn dữ liệu lẫn cấu trúc của Table: **Drop Table <tên Table>**

*** Lock 1 Table :**

Khi 2 user cùng truy xuất 1 table, có khả năng tiến trình update của user này sẽ xung đột với tiến trình Select hay update của user thứ 2. Chẳng hạn như khi user 1 đang thực hiện lệnh: Select Avg (Discount) From Customer thì user thứ 2 lại đang thực hiện lệnh Update:

Update Customer

Set Discount = 0.01

Where ShipCity = 'Khanh Hoa'

Trong trường hợp này, user 1 có thể lấy giá trị cũ của Discount để tính trung bình. Tập lệnh Transact-SQL cung cấp 1 tùy chọn để khóa table, hạn chế xung đột xảy ra trong quá trình truy xuất dữ liệu.

Lệnh sau đây sẽ bảo vệ khỏi sự xung đột với update:

Select Avg (Discount)

From Customer **With (TabLock)**

TabLock là khóa chung (share lock) trên table – nó ngăn các user khác update dữ liệu trên table đang được sử dụng bởi 1 user nào đó, nhưng vẫn cho phép đọc dữ liệu.

Tương tự, phát biểu update có thể dùng From với **TabLockX** để thiết lập khóa riêng (exclusive lock) – không cho bất cứ loại truy xuất nào trên table đang dùng.

Update Customer

Set Discount = .10

From Customer **With (TabLockX)**

Where ShipCity = 'Campuchia'

III. STORED PROCEDURE:

Một stored procedure là một nhóm các câu lệnh Transact-SQL đã được biên dịch và chứa trong SQL Server dưới một tên nào đó và được xử lý như một đơn vị (chứ không phải nhiều câu lệnh SQL riêng lẻ).

Một stored procedure có thể chứa tất cả các lệnh SQL (ngoại trừ các lệnh CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER, CREATE VIEW, USE). Trong stored procedure có các tham số đầu vào, các tham số đầu ra, biến cục bộ, lệnh gán, các thao tác trên cơ sở dữ liệu và cấu trúc điều khiển việc thực thi.

Các stored procedure của Microsoft® SQL Server™ trả về dữ liệu theo 4 dạng:

- Các **output parameter** có thể là:
 - ✓ dữ liệu (ký tự hay số nguyên)
 - ✓ một biến con trỏ (các con trỏ là các tập kết quả được rút trích mỗi lần một dòng).
- Các **mã trả về**, thường là các số nguyên.
- **Một tập kết quả** cho mỗi câu lệnh SELECT chứa trong stored procedure hay trong những stored procedure khác được gọi bởi stored procedure.
- **Một con trỏ toàn cục** có thể được tham khảo bên ngoài stored procedure.

Ví dụ stored procedure đơn giản sau minh họa 3 phương cách mà stored procedure có thể trả dữ liệu về:

1. Đầu tiên stored procedure cấp một câu lệnh SELECT để trả về một tập kết quả tổng kết hoạt động bán hàng theo từng nhân viên trong bảng **PHATSINH**.
2. Sau đó cấp một câu lệnh SELECT để điền vào tham biến số lượng các phiếu xuất đã tạo.
3. Cuối cùng, stored procedure có một câu lệnh RETURN trả về một số nguyên. Việc trả về các mã thường được dùng để trả về các thông tin kiểm tra lỗi.

USE QLVT

GO

DROP PROCEDURE sp_ThongKe_XuatHang

GO

```

CREATE PROC sp_ThongKe_XuatHang
    @SoCacPX INT OUTPUT
AS
-- SELECT để trả về một tập kết quả tổng kết trị giá của các phiếu xuất theo từng
-- nhân viên . Kết xuất : MaNV      Tong Tri gia
SELECT MANV, SUM(SOLUONG*DONGIA) as TongTriGia
FROM PhatSinh PS, CT_PHATSINH CT
Where Loai ='X' AND PS.PHIEU = CT.PHIEU
GROUP BY MANV
ORDER BY MANV

SELECT @SoCacPX = Count(Phieu) FROM PHATSINH WHERE LOAI ='X'
-- hoặc có thể viết:
-- SET @SoCacPX = (SELECT Count(Phieu) FROM PHATSINH WHERE LOAI ='X')

-- Trả về 0 để báo là thành công
RETURN 0
GO

-- Kiểm tra stored procedure vừa viết: ta dùng
-- Khai báo các tham biên giữ mã trả về và tham số xuất.
DECLARE @Maloi INT
DECLARE @ SoCacPX  INT

-- Thực thi thủ tục, trả về tập kết quả từ câu lệnh SELECT đầu ---- tiên.
EXECUTE @Maloi = sp_ThongKe_XuatHang @SoCacPX OUTPUT

```

1. Ưu Điểm Của Stored Procedure

Stored procedure có một số ưu điểm chính như sau:

- **Hiệu quả thực thi (performance):** Khi thực thi một câu lệnh SQL thì SQL Server phải kiểm tra permission(quyền thực hiện) xem user gửi câu lệnh đó có được phép thực thi câu lệnh hay không đồng thời kiểm tra cú pháp rồi mới tạo ra một kế hoạch thực thi (excute plan) và thực thi. Nếu có nhiều câu lệnh như vậy gửi qua mạng có thể sẽ làm giảm tốc độ làm việc của server. SQL Server sẽ làm việc hiệu quả hơn nếu dùng stored procedure vì người gửi chỉ gửi một tập các câu lệnh đơn và SQL Server chỉ cần kiểm tra một lần sau đó tạo ra một kế hoạch thực thi và thực thi. Nếu stored procedure được gọi nhiều lần thì kế hoạch thực thi có thể sử dụng lại nên stored procedure đã biên dịch do vậy mà hiệu quả làm việc sẽ nhanh hơn. Ngoài ra cú pháp của các câu lệnh SQL đã được SQL Server kiểm tra trước khi save nên không cần kiểm tra lại mỗi lần thực thi.
- **Tạo khung sườn trong lập trình(Programming Framework):** Một khi stored procedure được tạo ra nó có thể được sử dụng lại. Điều này sẽ làm cho việc bảo trì(maintainability) dễ dàng hơn do việc tách rời giữa các luật business (business rules _ những luật thể hiện bên trong stored procedure) và cơ sở dữ liệu. Ví dụ nếu có một sự thay đổi nào đó về mặt logic thì ta chỉ việc thay đổi code bên trong stored procedure mà thôi. Những ứng dụng dùng stored procedure này có thể sẽ không cần

thay đổi mà vẫn tương thích với các business rule mới. Cũng giống như các ngôn ngữ lập trình khác stored procedure cho phép ta đưa vào các tham số đầu vào và trả về các tham số đầu ra đồng thời nó cũng có khả năng gọi các stored procedure khác.

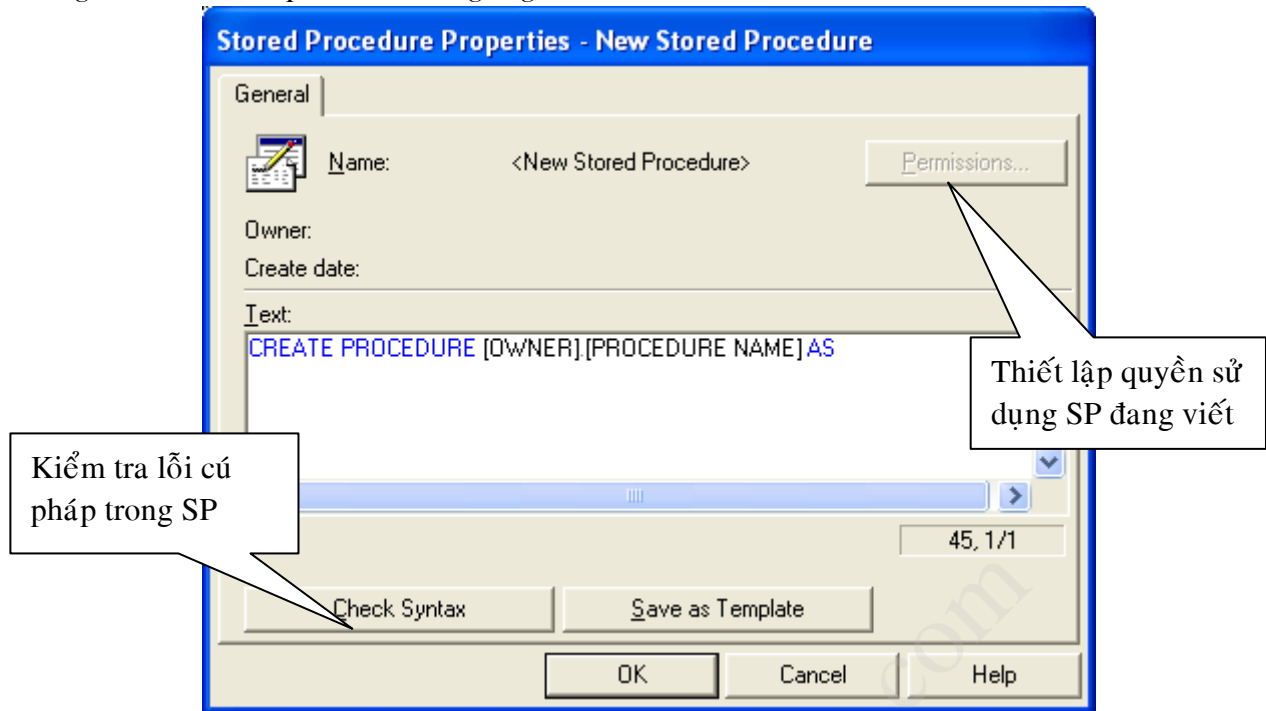
- **Bảo mật:** Giả sử chúng ta muốn giới hạn việc truy xuất dữ liệu trực tiếp của một user nào đó vào một số bảng, ta có thể viết một stored procedure để truy xuất dữ liệu và chỉ cho phép user đó được sử dụng stored procedure đã viết sẵn mà thôi chứ không thể đụng đến các bảng cách trực tiếp. Ngoài ra stored procedure có thể được mã hoá (encrypt) để tăng thêm tính bảo mật.

2. Các loại Stored Procedure:

Stored procedure có thể được chia thành 5 nhóm sau:

- **System stored procedure :** Là những stored procedure chứa trong cơ sở dữ liệu **master** và thường bắt đầu bằng tiếp đầu ngữ **sp_**. Các stored procedure này thuộc loại built-in và chủ yếu dùng trong việc quản trị cơ sở dữ liệu cũng như quản trị bảo mật. Ví dụ bạn có thể kiểm tra tất cả các tiến trình đang sử dụng bởi user DomainName\Administrators nhờ vào câu lệnh EXEC sp_who @lginame='DomainName\Administrators'. Có hàng trăm stored procedure hệ thống trong SQL Server. Và đây cũng là phần nghiên cứu trọng tâm của đề tài này.
- **Local stored procedure :** Đây là loại thường dùng nhất. Chúng được chứa trong cơ sở dữ liệu do user tạo và thường được viết để thực hiện một công việc nào đó. Thông thường người ta nói đến stored procedure là nói đến loại này. Stored procedure cục bộ thường được viết bởi người quản trị hệ cơ sở dữ liệu hoặc lập trình viên.
- **Temporary stored procedure :** Là những stored procedure tương tự như stored procedure cục bộ nhưng chỉ tồn tại cho đến khi kết nối đã tạo ra chúng bị đóng lại hoặc SQL Server shutdown. Các stored procedure này được tạo ra trên cơ sở dữ liệu **tempdb** của SQL Server nên chúng sẽ bị xoá khi kết nối tạo ra chúng bị cắt đứt hay khi SQL Server down. **Temporary stored procedure được chia làm 3 loại: local** (bắt đầu bằng dấu #), **global** bắt đầu bằng dấu ##) và stored procedure được **tạo ra trực tiếp trên cơ sở dữ liệu tempdb**. Loại local chỉ được sử dụng bởi kết nối đã tạo đã chúng và bị xoá khi disconnect, loại global có thể được sử dụng bởi bất kỳ kết nối nào. Quyền thực thi cho loại global mặc định không thay đổi cho nhóm **public**. Loài stored procedure được tạo trực tiếp trên cơ sở dữ liệu tempdb khác với 2 loại trên ở chỗ ta có thể set permission, chúng tồn tại kể cả sau khi kết nối tạo ra chúng bị cắt và chỉ biến mất khi SQL Server shutdown.
- **Extended stored procedure :** Đây là một loại stored procedure sử dụng chương trình ngoại vi (external program) vốn được biên dịch thành một DLL để mở rộng chức năng hoạt động của SQL Server. Loại này thường bắt đầu bằng tiếp đầu ngữ **xp_**. Ví dụ, xp_sendmail dùng để gửi mail cho một người nào đó hay xp_cmdshell dùng để chạy một DOS command ... (xp_cmdshell 'dir:c\'). Nhiều loài stored procedure mở rộng được xem như stored procedure hệ thống và ngược lại.
- **Remote stored procedure :** Những stored procedure gọi stored procedure ở server khác.

3. Tạo 1 Stored Procedure: Right click Stored Procedures / New Stored Procedure



Cú pháp: `CREATE PROCEDURE procedure_name`
`[{ @parameter data_type } [VARYING] [= default] [OUTPUT]]`
`[WITH { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION }]`
`[FOR REPLICATION]`
`AS sql_statement`

Các đối số:

- ✓ *Procedure_name*: là tên của stored procedure mới. Các procedure tạm cục bộ và toàn cục được tạo bằng cách thêm dấu # phía trước tên như *#procedure_name* đối với stored procedure tạm cục bộ, *##procedure_name* đối với các stored procedure tạm toàn cục. Tên đầy đủ của stored procedure bao gồm cả dấu # không được vượt quá 128 ký tự. Việc chỉ định người tạo stored procedure là tùy chọn.
- ✓ *@parameter*: Là tham số trong stored procedure. Có một hay nhiều hơn các tham số được khai báo trong câu lệnh CREATE PROCEDURE. Giá trị của mỗi tham số được khai báo phải được cấp bởi người dùng khi stored procedure được thực thi (nếu tham số không được định nghĩa giá trị mặc định). Một stored procedure có thể lên đến tối đa 2100 tham số.

Xác định một tên tham số bằng cách thêm vào ký hiệu @ trước ký tự đầu tiên. Tên tham số phải phù hợp với các luật dành cho những định danh. Các tham số cục bộ trong procedure, các tên tham số giống nhau có thể được dùng ở những stored procedure khác nhau.

- ✓ *data_type*: Là kiểu dữ liệu của tham số.
- ✓ *VARYING*: Chỉ định tập kết quả được hỗ trợ như một tham số xuất.
- ✓ *default*: là giá trị mặc định dành cho tham số. Giá trị mặc định phải là một hằng và có thể là NULL. Nó có thể chứa các ký tự như %, _, [], and [^] nếu procedure sử dụng tham số với từ khóa LIKE.

✓ OUTPUT: Chỉ định tham số là tham số trả về.

✓ {RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}

RECOMPILE cho biết SQL Server không lưu lại kế hoạch dành cho stored procedure này và stored procedure này phải được tái biên dịch lại tại mỗi thời điểm chạy.



ENCRYPTION cho biết SQL Server mã hoá bộ trong bảng **syscomments** chứa văn bản của câu lệnh CREATE PROCEDURE . Sử dụng ENCRYPTION để ngăn procedure publish stored procedure như một phần của việc nhân bản SQL Server.

✓ FOR REPLICATION: cho biết stored procedure được tạo ra để nhân bản không thể thực thi trên Subscriber. Tùy chọn này không thể sử dụng với tùy chọn WITH RECOMPILE .

✓ AS: chỉ định các hành động mà procedure sẽ thực hiện.

✓ *Sql_statement*: các lệnh trong Sql

Note : Tạo một stored procedure sử dụng công cụ **QUERY ANALYZER**

- 1) Kết nối với Server có chứa cơ sở dữ liệu cần tạo stored procedure.
- 2) Trong cửa sổ Query, đánh câu lệnh Transact-SQL tạo một stored procedure mới.
- 3) Click nút kiểm tra cú pháp câu lệnh () hay ấn tổ hợp phím Ctrl-F5 để kiểm tra cú pháp.
- 4) Click nút thực thi () hay ấn phím F5 để tạo một stored procedure mới.

Note: Sau khi tạo 1 Stored Procedure, để thi hành nó ta vào cửa sổ Query Analyze và dùng lệnh Execute:

Thực thi một stored procedure bằng cách sử dụng câu lệnh Transact-SQL EXECUTE. Bạn cũng có thể thực thi một stored procedure mà không cần từ khoá EXECUTE nếu stored procedure là lệnh đầu tiên trong khối.

Cú pháp: EXEC

```
[[ @return_status = ] { procedure_name @procedure_name_var }
[ [ @parameter = ] { value | @variable [ OUTPUT ] | [ DEFAULT ] ]
[ ,...n ]
[ WITH RECOMPILE ]
```

Các đối số:

✓ *@return_status*

Là một tham biến kiểu int tùy chọn lưu trữ trạng thái trả về của một stored procedure. Tham biến này phải được khai báo trong stored procedure trước khi dùng trong một câu lệnh EXECUTE.

✓ *procedure_name*

Là tên stored procedure cần thực thi.

✓ *@procedure_name_var*

Tên tham biến cục bộ đã được định nghĩa thể hiện tên stored procedure.

✓ *@parameter*

Là tham số dành cho stored procedure , như định nghĩa trong phần câu lệnh CREATE PROCEDURE. Khi được dùng dưới dạng *@parameter_name = value* , các tên tham số và các hằng không cần theo thứ tự tham số đã khai báo trong câu lệnh CREATE PROCEDURE. Tuy nhiên, nếu dạng *@parameter_name = value* được dùng cho bất kỳ một tham số, nó phải sử dụng cho tất cả các tham số tiếp theo.

✓ *value*

Là giá trị tham số cho thủ tục. Nếu tên tham số không được xác định, các giá trị tham số phải được cấp theo đúng thứ tự đã khai báo trong câu lệnh CREATE PROCEDURE.

✓ *@variable*

Là tham biến lưu một tham số hay một tham số trả về.

✓ OUTPUT

Chỉ định stored procedure trả về một tham số. Sử dụng từ khoá này khi các tham biến con trỏ chính là các tham số.

✓ DEFAULT

Cung cấp giá trị mặc định cho tham số đã định nghĩa trong thủ tục. Khi thủ tục yêu cầu một giá trị của tham số không có mặc định đã định nghĩa sẵn và tham số cũng không được chỉ định giá trị, một lỗi sẽ xảy ra.

✓ WITH RECOMPILE

Bắt buộc một kế hoạch mới được biên dịch. Sử dụng tùy chọn này nếu tham số bạn đang cấp không đúng kiểu hay dữ liệu vừa thay đổi cách đặc biệt. Kế hoạch được thay đổi được dùng trong các thực thi tiếp sau. Tùy chọn này được dùng cho các stored procedure mở rộng. Khuyến nghị sử dụng tùy chọn này cách hạn chế.

Ví dụ:

```
CREATE PROCEDURE List_Cust      -- tên stored procedure
  @MinDiscount Dec(5,3)      -- tham số
AS                             -- bắt đầu thân của Stored Procedure
  Select *
    From Customer Where Discount >= @MinDiscount
```

Để thi hành Stored Procedure trên, ta nhập vào: **Execute List_Cust 0.01**

a. Tham số : 1 Stored Procedure có thể có tới 2100 tham số; mỗi tham số có dạng:

@ten_tham_so kiểu_dữ_liệu

Ta có thể định nghĩa 1 giá trị input mặc định (default input value) trong trường hợp khi gọi 1 Stored Procedure mà lại không cung cấp giá trị cho tham số hình thức.

Cú pháp : *@ten_tham_so kiểu_dữ_liệu = giá_trị*

Ví dụ: Trong ví dụ trên ta tạo 1 tham số mặc định như sau:

@MinDiscount Dec(5,3) = 0.001

Tất cả tham số được xem như là input; để 1 tham số đóng vai trò là **output**, ta thêm từ khóa Output vào sau khai báo của nó.

Ví dụ: Tạo 1 Stored Procedure tên GetCustDiscount để trả về mức giảm giá của 1 khách hàng có mã do ta gửi vào.

```
CREATE PROCEDURE GetCustDiscount
```

```
  @MaKH Int,
```

```
  @GiamGia Dec (5,3) Output
```

```
AS
```

```
  Set @Giamgia = (Select Discount From Customer Where CustID = @MaKH)
```

Khi gọi Stored Procedure trên:

Execute GetCustDiscount 246900, @GiamGia Output

với @GiamGia là 1 biến

Ví dụ áp dụng: Tạo Stored Procedure tên ListLowHighDiscount cho biết các khách hàng có mức giảm giá <0.01 và mức giảm giá >=0.10

c. Phát biểu Return: kết thúc procedure và trả về 1 trị nguyên cho người gọi.

Ví dụ:

```
CREATE PROCEDURE ListCustWithDiscount
```

```
    @MinDiscount Dec(5,3) = 0.0001
```

```
AS
```

```
    If (@MinDiscount >1) Return (1)
```

```
    Select * From Customer Where Discount >=@MinDiscount
```

```
    Return (0)
```

Lệnh gọi:

```
Execute @Status = ListCustWithDiscount 0.03
```

Trong Stored Procedure có các lệnh sau:

- Khai báo biến :

```
    Declare @dem int
```

- Lệnh gán :

```
    Set @dem = 1
```

Ví dụ:

```
CREATE Procedure GetNameAndDiscount
```

```
    @CustID int, @Name varchar (30) Output , @Discount Dec(5,3) Output
```

```
As
```

```
    Select @Name = Name, @Discount = Discount
```

```
    From Customer
```

```
    Where CustID = @CustID
```

```
    Print Cast ('Ten khách hàng :'+ @Name as varchar (30) )
```

```
    Print ('Muc giam gia :'+ Str( @discount,5,3) )
```

- Khối lệnh : Trong 1 cấu trúc nếu có nhiều lệnh thì đặt các lệnh đó trong

Begin

```
    lệnh1
```

```
    lệnh 2
```

End

4. Cấu trúc trong SP:

- Cấu trúc If :

```
    If <dieukien>
```

```
        Lệnh thực hiện khi điều kiện đúng
```

```
    Else
```

```
        Lệnh thực hiện khi điều kiện sai
```

Ví dụ:

```
If (Select Avg(Diem) From Diem Where Masv='95Q10001') < 5
    PRINT 'Khong duoc thi tot nghiep'
Else
    PRINT 'Duoc thi tot nghiep'
```

- Vòng lặp While :

```
WHILE <đk>
BEGIN
    Lệnh
    [BREAK]
    [CONTINUE]
END
```

- Label và Goto:

Ví dụ:

SkipNextStep:

....

Goto SkipNextStep

* **Giao tác (Transaction):** Khi thay đổi dữ liệu trên nhiều table hay nhiều records trên 1 table, ta phải đảm bảo tính nhất quán về dữ liệu trên cơ sở dữ liệu. Ví dụ ta đang thực hiện việc tăng mức giảm giá cho các customer thì tiến trình đang thi hành bị ngắt quãng vì 1 nguyên nhân nào đó (mất nguồn). Như vậy, rõ ràng là chỉ có 1 số khách hàng được tăng Discount, còn 1 số khác thì không ; điều này sẽ dẫn đến không nhất quán về dữ liệu.

Để tránh tình trạng này xảy ra, SQL Server cung cấp 1 khả năng cho phép ta phục hồi lại dữ liệu cũ nếu công việc đang thi hành bị lỗi : giao tác .

Một công việc của ta có khả năng là 1 lệnh hay nhiều lệnh SQL tác động lên nhiều Table; một công việc như vậy ta gọi là 1 giao tác. Để bắt đầu 1 giao tác, ta dùng : Begin Transaction; xác nhận 1 giao tác đã hoàn thành : Commit; hủy bỏ giao tác và trả lại dữ liệu cũ : RollBack.

Ví dụ: Trong ngân hàng, 1 giao tác là việc chuyển số tiền từ tài khoản tiết kiệm của khách hàng có tài khoản @TKCHUYEN qua tài khoản @TKNHAN với số tiền @SOTIEN :

```
CREATE PROC [dbo].[SP_CHUYENTIENTIEN]
    @TKCHUYEN NVARCHAR (10) , @TKNHAN NVARCHAR (10) , @SOTIEN BIGINT
AS
SET XACT_ABORT ON
BEGIN TRAN
BEGIN TRY
    UPDATE TAIKHOAN
        SET SODU = SODU+ @SOTIEN
        WHERE SOTK= @TKNHAN
```

```
UPDATE TAIKHOAN
SET SODU = SODU - @SOTIEN
WHERE SOTK= @TKCHUYEN
```

```
COMMIT
END TRY
BEGIN CATCH
ROLLBACK
RAISERROR('Chuyển tiền không được ', 16, 1)
END CATCH
```

* Ghi chú Về tùy chọn **XACT_ABORT**: Đây là tùy chọn ở mức kết nối, chỉ có tác dụng trong phạm vi kết nối của ta. XACT_ABORT nhận hai giá trị ON và OFF (OFF là giá trị mặc định). Khi tùy chọn này được đặt là OFF, SQL Server sẽ chỉ hủy bỏ lệnh gây ra lỗi trong transaction và vẫn cho các lệnh khác thực hiện tiếp, nếu lỗi xảy ra được đánh giá là không nghiêm trọng. Còn khi XACT_ABORT được đặt thành ON, SQL Server mới cư xử đúng như mong đợi – khi gặp bất kỳ lỗi nào nó hủy bỏ toàn bộ transaction và quay lại trở lại như lúc ban đầu.

Bài tập:

Cho cơ sở dữ liệu QLVT với các table sau trong SQL Server :

- **Nhanvien** (MANV, HOTEN, DIACHI, NGAYSINH, LUONG, GHICHU)
- **Kho** (MAKHO, TENKHO,)
- **Vattu** (MAVT, TENVT, DVT)
- **Phatsinh** (PHIEU, NGAY, LOAI, HOTEN, THANHTIEN, MANV)
- **CT_Phatsinh** (PHIEU, MAVT, SOLUONG, DONGIA, MAKHO)

1. Viết 1 SQL Script để tạo cấu trúc của csdl QLVT
2. Tạo 1 SP để hủy phiếu có số phiếu do ta gửi vào.
3. Tính số lượng tồn kho của từng vật tư