Chương 4

Lập trình với T-SQL

Nội dung

- Kỹ thuật thi hành lệnh T-SQL
- Các cấu trúc điều khiển
- Stored Procedure (thủ tục)
- Function (hàm)

Cơ bản về lập trình bằng T_SQL

- Khai báo và sử dụng biến
- Các cấu trúc lệnh
- RaiseError

IDENTIFIER (Định danh)

- Tên của các đối tượng đều được gọi là định danh. Trong SQL Server, có các định danh như Server, Databases, Object of Database as Table, View, Index, Constraint,…
- Qui tắc định danh:
  - Tối đa 128 ký tự.
  - Bắt đầu là một ký tự từ A-Z
  - Bắt đầu là một ký hiệu @, # sẽ có một ý nghĩa khác.
  - Những định danh nào có dấu khoảng trắng ở giữa thì phải kẹp trong dấu [] hoặc “ “
  - Đặt định danh sao cho ngắn gọn, đầy đủ ý nghĩa, phân biệt giữa các đối tượng với nhau, không trùng lặp, không trùng với từ khóa của T-SQL.

Tham chiếu đến các đối tượng trong SQL Server

Cú pháp:

Server.Database.Owner.Object

Hay:

Server.Database..Object

Ví dụ:

Create Table Northwind.dbo.Customers

Create Table Northwind..Customers

Data type, Batch, Script

Kiểu dữ liệu (Data type) : có hai loại

- Kiểu dữ liệu hệ thống: Do hệ thống cung cấp
- Kiểu dữ liệu do người dùng định nghĩa (User – defined data types.)

Gói lệnh (Batch)

- Bao gồm các phát biểu T-SQL và kết thúc bằng lệnh GO.
- Các lệnh trong gói lệnh sẽ được biên dịch và thực thi cùng một lúc.
- Nếu một lệnh trong batch bị lỗi thì batch cũng xem như lỗi
- Các phát biểu Create bị ràng buộc trong một batch đơn.

Ex :
```
Use  Northwind
Select  * from Customers
GO
```

Kịch bản (Script)

- Một kịch bản là một tập của một hay nhiều bó lệnh được lưu lại thành một tập tin .SQL

Biến – Biến cục bộ

- Biến là một đối tượng dùng để lưu trữ dữ liệu. Biến phải được khai báo trước khi dùng.
- Có 2 loại biến: cục bộ và toàn cục
- Biến cục bộ:
  - Được khai báo trong phần thân của một bó lệnh hay một thủ tục.
  - Phạm vi hoạt động của biến bắt đầu từ điểm mà nó được khai báo cho đến khi kết thúc một bó lệnh, một thủ tục hay một hàm mà nó được khai báo.
  - Tên của biến bắt đầu bằng @

Sử dụng biến cục bộ

- Khai báo

```
DECLARE @var_name = expression
SELECT {@var_name = expression}[,...n]
```

Ví dụ

```
DECLARE @makh CHAR(5)
SET @makh = 'ANTON'
SELECT * FROM Customers
        WHERE Customerid = @makh
```

Sử dụng biến cục bộ

VD :

```
DECLARE @manv int
SET @manv = 2
SELECT  *  FROM Employees
        WHERE Employeeid = @manv
```

```
DECLARE @manv int, @country nvarchar(15)
SET @manv = 3
SET @country ='Usa'
SELECT * FROM Employees
        WHERE Employeeid = @manv
                and country =@country
```

Sử dụng biến cục bộ

VD

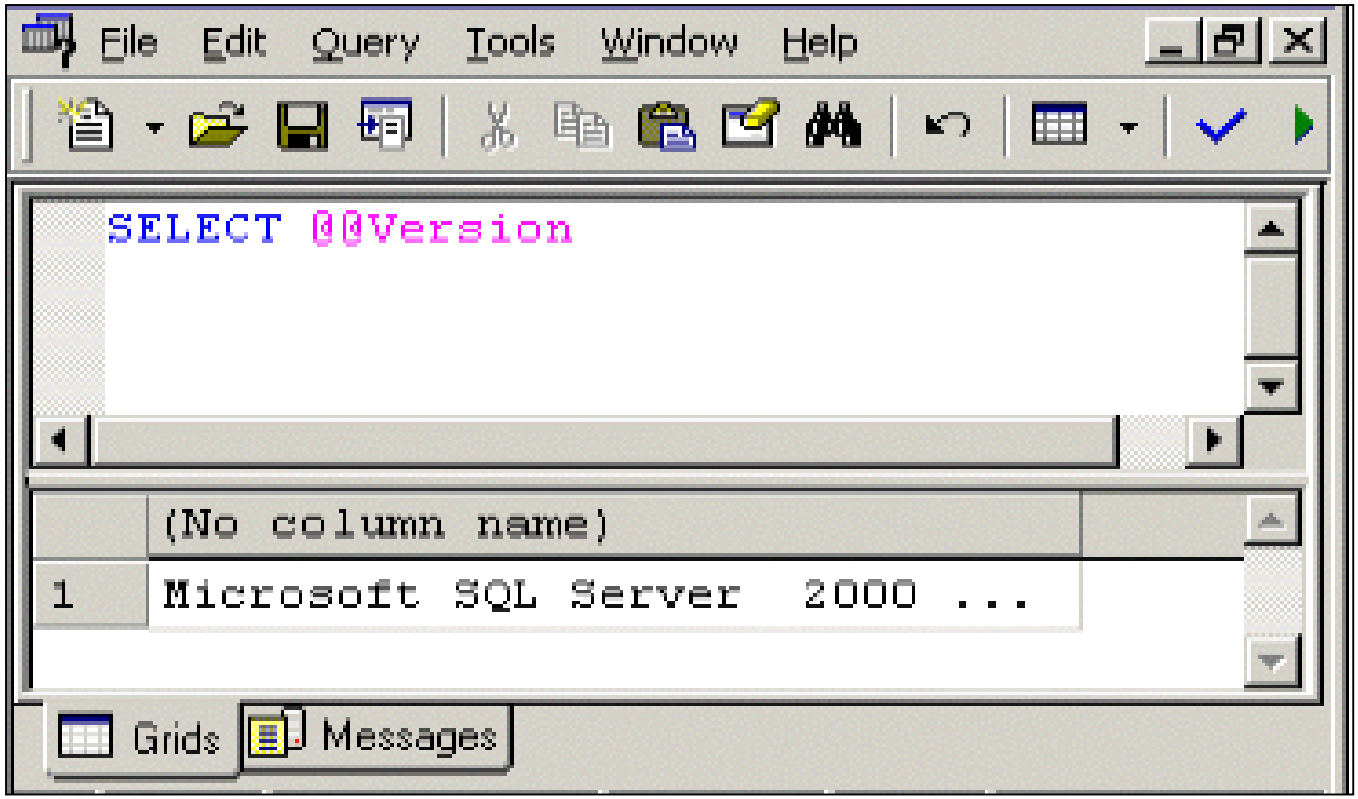
```
DECLARE @tong int
Select  @tong = Sum(quantity * Unitprice) From
[Order details]
SELECT @tong as tongtien
Print 'Tong tien = '+ convert(varchar(20),@tong)
```

```
DECLARE @masp int
Select @masp = productid  From
Northwind..Products
Select @masp –của dòng cuối trong bảng
```

```
DECLARE @Masp int
Select @masp = Productid From Northwind..Products
Order by  Productid DESC
Select @masp as 'Ma SP'
```

Biến toàn cục

- Biến toàn cục được định nghĩa như hàm hệ thống. Các biến này không có kiểu.
- Tên bắt đầu bằng @@



Các biến toàn cục

Variable	Return value
@@Trancount	Number of transactions currently open on the connection
@@Servername	Name of local servers running SQL Server
@@Rowcount	Number of rows affected by the latest SQL statement
@@Identity	Return last Number Identity
@@Error	Return order number Error when SQL exculate, return 0 when The command completed successfully
@@Fetch_status	Return status of Fetch command of pointer variable (0 :Success, -1 : Mistake or exceed range, -2 : Unsuccess

Các biến toàn cục

VD:

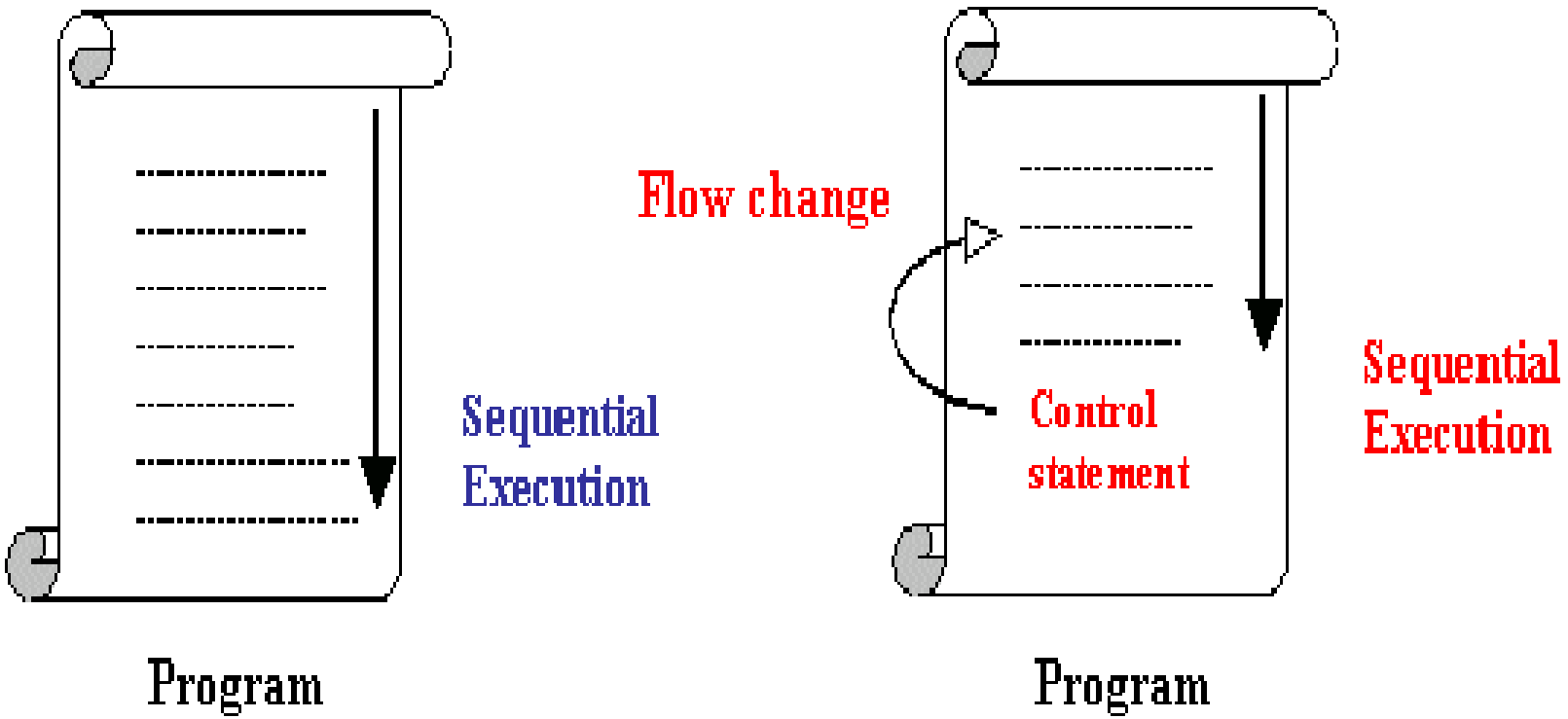
```
--How many are transaction opening
If (@@Trancount>0)
Begin
        Raiserror ('Take can not be executed within a transaction',10,1)
        Return
End
```

```
--Number of rows affected by the latest SQL statement
 Use Northwind
   Update Employees Set LastName ='Brooke' where LastName ='Lan'
        If (@@RowCount =0)
        Begin
                Print  'Warning : No rows were updated'
                Return
        End

 Update Customers  Set Phone ='030' + Phone
        Where Country = 'German'
    Print @@Rowcount
```

Các biến toàn cục

```
--Tra ve so Identitity phat sinh sau cung
Create table hd (Mahd int identity Primary key, Ghichu
varchar(20))

Create table cthd(Mahd int,Masp char(10), Soluong int)

Insert into hd Values ('Record 1')
Insert into hd Values ('Record 2')

Declare @maso int
Set @maso = @@identity
Insert into cthd Values (@maso,'sp001',5)
Insert into cthd Values (@maso,'sp002',12)

Select * from hd
Select * from cthd
```

Cấu trúc điều khiển



Cấu trúc điều khiển

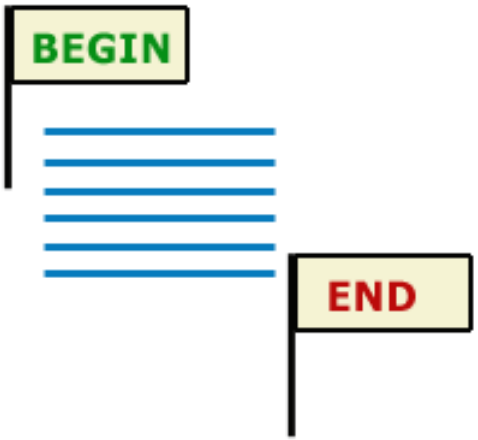
- Khối **BEGIN...END:** Nếu nhiều phát biểu cần thực thi có liên quan với nhau thì đặt các phát biểu này trong Begin…End

  Cú pháp:

  BEGIN

  statement | statement_block

  END



- **RETURN:** Trả về một giá trị, lệnh này nằm trong một block (khối) hay procedure. Nếu gặp phát biểu Return, quá trình xử lý kết thúc

  Cú pháp

  Return [Integer_expression]

Cấu trúc điều khiển

- Lệnh PRINT: Dùng để in thông tin ra màn hình kết quả của SQL.

  Cú pháp:

  PRINT 'any ASCII text'|@local_variable|@@Function |string_expr

  Ví dụ:

```
Print 'Hello'
Print N'Chào bạn'
Print getdate()
Print @@version

Declare @ten nvarchar(15)
Set @ten ='Nguyen Minh'
Print @ten
```

Cấu trúc điều khiển

- Cấu trúc điều khiển IF...ELSE: Cho phép thực thi một hay nhiều lệnh tùy thuộc vào một điều kiện nào đó.

- Cú pháp:

If Condition

statements

[Else [Condition 1]

statements]

- Ví dụ :

```
If  (Select Count(*) From Customers
        Where  Country ='Germany')>0
                print 'Co khach hang o Germany'
Else
                print 'Khong co khach hang o Germany'
```

Cấu trúc điều khiển

- Ví dụ:

```
Declare @msg varchar(100)
    If (Select Count(Unitprice)  From Products
        Where QuantityPerunit like  '%box%')>0
            Begin
                    Set NOCOUNT ON
                    Set @msg = 'Co vai sp co don vi tinh co chu box.
                                    Cac  sp do la :'
                    Select @msg
                    Select ProductName From Products
                            where QuantityPerunit like '%box%'
            End
    Else
                    print 'Khong co sp nao co dvt co chu box'
```

Cấu trúc điều khiển

- Ví dụ:

```
If (Select Avg(Unitprice)  From Products
      where QuantityPerUnit like '%box%' )>0
        Begin
              Set NOCOUNT on
              Declare @msg nvarchar(30)
              Set @msg = 'Co vai sp có don vi tinh co chu box.
                          Cac sp do la:'
              Select @msg
              Select ProductName From Products
                    where QuantityPerUnit like '%box%'
        End
        Else
              print 'Khong co sp nao co dvt co chu box'
```

Cấu trúc điều khiển

- **CASE :** là một biểu thức điều kiện được áp dụng bên trong một phát biểu khác. Case trả về các giá trị khác nhau tùy vào điều kiện hay một điều khiển nào đó.
- Cú pháp 1 :

```
Case  input_expression
        When when_expression Then resulf_expression[...n]
        [ELSE else_result_expression]
End
```

- Cú pháp 2 :

```
Case Boolean_expression
     When Boolean_expression Then resulf_expression[...n]
     [
     ELSE else_result_expression
      ]
End
```

Cấu trúc điều khiển

- **Example 1 :**

```
Declare @a int, @b int, @Hieu int
Set @a = 15
Set @b =27
Set @hieu = Case
                    When @a<@b then @b-@a
                    When @a>@b then @a-@b
                   Else 0
              End
Print  'Hieu='+convert(varchar(20),@hieu)
```

Example 2 :

```
Select ProductName, Unitprice,
        'Classification'=CASE
            When Unitprice<10 then 'Low price'
            When Unitprice Between 10 and 20 then 'Moderately Price'
            When Unitprice>20 then 'Expensive'
            Else 'Unknown'
        END
 From Products
```

Cấu trúc điều khiển

```
Select Productid, Quantity, UnitPrice, [discount%]=
    CASE
        When Quantity <=5 then 0.05
        When Quantity between 6 and 10 then 0.07
        When Quantity between 11 and 20 then 0.09
        Else 0.1
    END
From [Order Details]
Order by Quantity, Productid
```

Cấu trúc điều khiển

- **GOTO**: chuyển thực thi chương trình đến vị trí nhãn (label)
- Example

```
Declare @a int, @b int, @Hieu int
Set @a = 39
Set @b =10
hieu_loop:
        if @a>@b
        begin
                Set @hieu =@A-@B
                print 'a= '+convert(varchar(20),@a)
                print 'b= '+convert(varchar(20),@b)
                print 'hieu= '+convert(varchar(20),@hieu)
                Set @a =@hieu
                Goto hieu_loop
        end
print 'a='+convert(varchar(20),@a)
print 'b='+convert(varchar(20),@b)
print 'hieu='+convert(varchar(20),@hieu)
```

Cấu trúc điều khiển

- **Phát biểu lặp WHILE**: Vòng lặp sẽ thực thi cho đến khi biểu thức điều kiện (Boolean expression) trong While mang giá trị False.

- Cú pháp 1 :

```
WHILE Boolean_expression
        {sql_statement | statement_block}
        [BREAK]
        {sql_statement | statement_block}
        [CONTINUE]
```

Cấu trúc điều khiển

- Example :

```
Use  Northwind
While (Select  Avg(unitprice) From [Order Details]) <$50
Begin
    Update [Order Details]
        SET Unitprice = Unitprice *2
    Select Max(Unitprice) From [Order Details]
    If (Select Max(Unitprice) From [Order Details])>$50
        BREAK
    Else
        CONTINUE
End
Print 'Too much for the market to bear'
```

Cấu trúc điều khiển

- WAITFOR: SQL Server tạm dừng một thời gian trước khi xử lý tiếp các phát biểu sau đó.
- Cú pháp :

  WAITFOR {DELAY 'time' |TIME 'time'}

  Time : hh:mm:ss

  Deplay 'time': hệ thống tạm dừng trong khoảng thời gian time

  TIME 'time': hệ thống tạm dừng trong khoảng thời gian time chỉ ra

Ví dụ:

```
WAITFOR DELAY '00:00:02'
SELECT EmployeeID FROM Northwind.dbo.Employees
```

Cấu trúc điều khiển

- Lệnh **RAISERROR**: phát sinh lỗi của người dùng

Cú pháp

RAISERROR ({msg_id | msg_str}{, severity, state}
[WITH option[,...n]]

- **Msg_id:** Là thông báo, được lưu trong bảng sysmessage. Mã thông báo của người dùng phải bắt đầu từ trên 50000
- **Msg_str:** Nội dung thông báo, tối đa 400 ký tự.
- Để truyền tham số vào trong thông báo thì dùng dạng %<Loại ký tự>
- Loại ký tự là d,I,o,x,X hay u
- Lưu ý: số **float**, double, char không được hỗ trợ

Cấu trúc điều khiển

- Lệnh **RAISERROR**: phát sinh lỗi của người dùng

Các ký tự	Mô tả
d hoặc I	Biểu hiện là số nguyên (integer)
O	Octal không dấu
P	Con trỏ
S	Chuỗi
U	Số nguyên không dấu
x or X	Hexadecimal không dấu

Cấu trúc điều khiển

Severity Levels: Mức lỗi của một thông báo lỗi cung cấp một sự biểu thị loại vấn đề mà SQL Server gặp phải.

- Mức lỗi **10** là lỗi về thông tin và biểu thị nguyên nhân do thông tin nhập vào.
- Mức lỗi từ **11 đến 16** thì thông thường là do các user.
- Mức từ **17 đến 25** do lỗi phần mềm hoặc phần cứng. Bạn nên báo cho nhà quản trị hệ thống bất cứ khi nào sự cố xảy ra. Nhà quả trị hệ thống phải giải quyết sự cố đó và theo dõi chúng thường xuyên. Khi mức lỗi 17,18,19 xảy ra, bạn có thể tiếp tục làm việc, mặc dù bạn không thể thực thi lệnh đặc biệt.

Cấu trúc điều khiển

- **Mức lỗi 17:** Những thông báo này cho biết rằng câu lệnh SQL Server cạn kiệt tài nguyên (Ví dụ như lock hoặc thiếu không gian đĩa cho CSDL) hoặc vượt quá tập giới hạn bởi nhà quản trị
- Người quản trị hệ thống nên giám sát tất cả các sự cố xảy ra có mức trầm trọng từ **17 đến 25** và in ra giải thích lỗi bao gồm các thông tin để quay lại từ lỗi.
- Mức lỗi từ **20 đến 25** chỉ ra sự cố hệ thống. Đó là lỗi không tránh được, có nghĩa là tiến trình không còn đang chạy. Tiến trình tê liệt trước khi nó dừng, ghi nhận thông tin về cái gì xảy ra, và sau đó kết thúc.

Cấu trúc điều khiển

- *State*: Là một số nguyên tùy ý từ 1 đến 127 mô tả thông tin diễn giải về trạng thái lỗi.
- *Argument:* Là tham số dùng trong việc thay thế cho biến để định nghĩa thông báo lỗi hoặc thông báo tương ứng với mã lỗi msg_id. Có thể không có hoặc có nhiều tham số, tuy nhiên, không được quá 20. Mỗi tham số thay thế có thể là một biến local hoặc một trị bất kỳ trong các kiểu dữ liệu int, char, varchar, binary, varbinary. Các kiểu khác không được cung cấp.

Cấu trúc điều khiển

- Thêm một thông báo lỗi mới do người dùng định nghĩa

Cú pháp:

Sp_AddMessage msg_id, severity,
'msg'[,'language'][,'with_log'][,'replace']

- Xóa một thông báo lỗi mới do người dùng định nghĩa

Cú pháp:

Sp_DropMessage msg_id

Cấu trúc điều khiển

- ***msg_id***: là mã số của lỗi mới, là một số int, không được trùng các mã đã có sẵn, bắt đầu là 50001.
- ***severity***: là mức nghiêm trọng của lỗi, là một số smallint. Mức hợp lệ là từ 1 đến 25. Chỉ có người quản trị CSDL mới có thể phát sinh thêm một thông báo lỗi mới từ 19 đến 25.
- **'*msg*':** là một chuỗi thông báo lỗi, tối đa 255 ký tự.
- **'*language*':** là ngôn ngữ của thông báo lỗi, mặc định là ngôn ngữ của phiên kết nối.

Cấu trúc điều khiển

- '***with_log***': thông báo lỗi có được ghi nhận vào nhật ký của ứng dụng khi nó xảy ra hay không, mặc định là FALSE. Nếu là **true**, thì lỗi luôn luôn được ghi vào nhật ký ứng dụng. Chỉ có những thành viên có vai trò **sysadmin** mới có thể sử dụng tham số này.
- '***replace***': nếu được chỉ định chuỗi **REPLACE**, thì thông báo lỗi đã tồn tại được ghi đè bởi chuỗi thông báo mới và mức lỗi mới. Tham số này phải chỉ định nếu *msg_id* đã có.
- Lưu ý: *nếu trả về 0 tức là thêm vào thành công, trả về 1 là thất bại.*

Cấu trúc điều khiển

Ví dụ: exec sp_...

sp_addmessage 50001,10,'Khong tim thay mau tin %d trong %ls'

sp_addmessage 50002,16,'Khong xoa duoc %s vi %s co ton tai trong %ls'

sp_addmessage 50003,16,'Mot lop chi co toi da %d hoc sinh'

sp_addmessage 50004,16,'Don gia ban phai lon hon don gia goc'

--xem thong bao loi vua xay dung

use master

sp_helptext sysmessages

Select * From sysmessages Where error =50002

sp_dropmessage 50002

Select * From sysmessages

Phát biểu RAISERROR

```
--XAY DUNG CAU THONG BAO LOI BANG RAISERROR
use Northwind
RAISERROR (50001,10,1,4,'SANPHAM')
DECLARE @@MA INT
DECLARE @@TEN NVARCHAR
SET @@TEN ='SANPHAM'
SET @@MA =8
SELECT  productid FROM products
            WHERE productid=@@MA
IF (@@ROWCOUNT=0)
  BEGIN
     RAISERROR (50001,10,1,@@MA,@@TEN)
  END
GO
```

Thủ tục (Stored Procedure)

- Khái niệm về thủ tục
- Các thao tác cơ bản với thủ tục
- Tham số bên trong thủ tục
- Một số vấn đề khác trong thủ tục

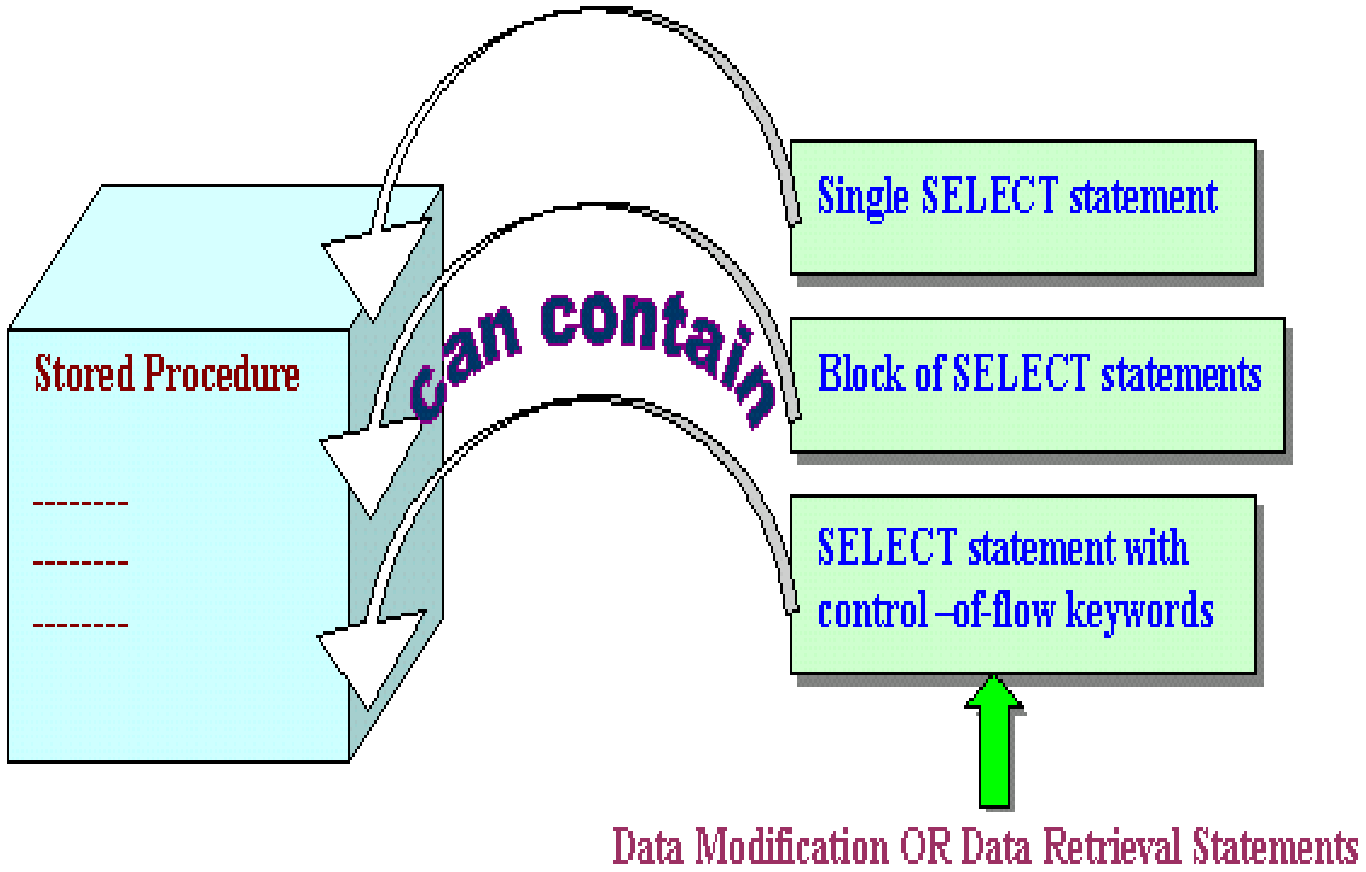
Khái niệm về thủ tục

- Một thủ tục là một đối tượng trong cơ sở dữ liệu, bao gồm một tập nhiều câu lệnh SQL được nhóm lại với nhau thành một nhóm với những khả năng sau:
- Có thể bao gồm các cấu trúc điều khiển (IF, WHILE, FOR).
- Bên trong thủ tục lưu trữ có thể sử dụng các biến nhằm lưu giữ các giá trị tính toán được, các giá trị truy xuất được từ cơ sở dữ liệu.

Khái niệm về thủ tục

- Một thủ tục có thể nhận các tham số truyền vào cũng như có thể trả về các giá trị thông qua các tham số.
- Khi một thủ tục lưu trữ đã được định nghĩa, nó có thể được gọi thông qua tên thủ tục, nhận các tham số truyền vào, thực thi các câu lệnh SQL bên trong thủ tục và có thể trả về các giá trị sau khi thực hiện xong.

Khái niệm về thủ tục



Lợi ích của thủ tục

- Đơn giản hoá các thao tác trên cơ sở dữ liệu nhờ vào khả năng module hoá các thao tác này.
- Thủ tục lưu trữ được phân tích, tối ưu khi tạo ra; nên việc thực thi chúng nhanh hơn nhiều so với việc phải thực hiện một tập rời rạc các câu lệnh SQL tương đương theo cách thông thường.
- Cho phép thực hiện cùng một yêu cầu bằng một câu lệnh đơn giản thay vì phải sử dụng nhiều dòng lệnh SQL→ làm giảm thiểu sự lưu thông trên mạng.
- Có thể cấp phát quyền cho người sử dụng thông qua các thủ tục lưu trữ, nhờ đó tăng khả năng bảo mật đối với hệ thống.

Phân loại thủ tục

- Các loại Procedures
  - User-defined
  - System
  - Temporary
  - Remote
  - Extended

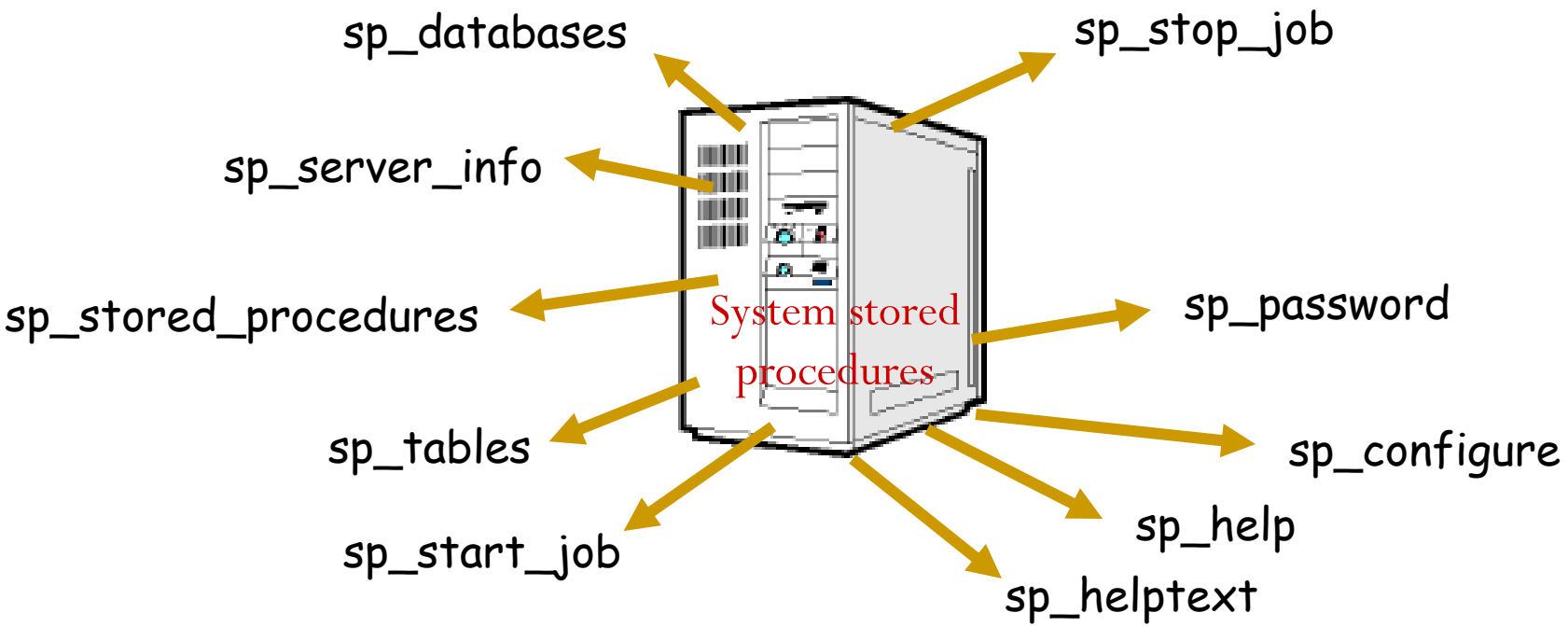
Phân loại thủ tục

- Sp = Stored procedure
- System sp: được lưu trữ trong CSDL master. Các thủ tục có tên bắt đầu là sp. Chúng đóng vai trò khác nhau của các tác vụ được cung cấp trong SQL Server.
- Local sp: được lưu trữ trong các CSDL người dùng, nó thực thi các tác vụ trong CSDL chứa nó. Được người sử dụng tạo hay từ các sp hệ thống.

Phân loại thủ tục

- Temporary sp: giống local sp nhưng nó chỉ hiện hữu cho đến khi kết nối tạo ra nó bị đóng. Nó được lưu trong CSDL TempDB. Có 3 loại temporary sp: Local (private), Global, sp tạo trực tiếp trong TempDB.
- Extended sp: là một thủ tục được tạo từ các ngôn ngữ lập trình khác (không phải SQL Server) và nó được triển khai tính năng của một thủ tục trong SQL Server. Các thủ tục này có tên bắt đầu là xp.
- Remote sp: là một thủ tục được gọi thực thi từ một server từ xa.

Ví dụ về thủ tục



Ví dụ về thủ tục

System Store Procedure	Description
Sp_databases	Lists all the databases available on the server.
Sp_server_info	Lists server information, such as, character set, version, and sort order.
Sp_stored_procedures	Lists all the stored procedures avaible in the current environment.
Sp_tables	Lists all the objects that can be queried in the current environment.
Sp_start_job	Starts an automated task immediately
Sp_stop_job	Stops an automated task that is running

Ví dụ về thủ tục

System Store Procedure	Description
Sp_password	Change the password for a login account
Sp_configue	Changes the SQL Server global configuration option. When used without options, display the current server settings.
Sp_help	Displays information about any database object.
Sp_helptext	Displays the actual text for a rule, a default, or an un-define function, trigger or view

User-defined Stored Procedures

- Được tạo bởi người sử dụng trong CSDL hiện hành.
- Các thủ tục có thể được tạo trước các đối tượng mà thủ tục tham chiếu đến.

Cách 1 : Dùng Enterprise Manager

Right Click at **Database**, expand **Programmability**, expand **Stored Procedures**, then right click **New Stored Procedure**

User-defined Stored Procedures

<u>Cách 2 :</u> Dùng phát biểu CREATE PROCEDURE

```
CREATE PROC [ EDURE ] procedure_name [ ; number ]
  [ { @parameter data_type }
     [ VARYING ] [ = default ] [ OUTPUT ]
  ] [,...n ]
[ WITH  { RECOMPILE | ENCRYPTION | RECOMPILE,
ENCRYPTION } ]
[ FOR REPLICATION ]
AS sql_statement [ ...n ]
```

Kiểm tra sự tồn tại của thủ tục:

sp_helptext 'Procedure_*name*'
sp_help 'Procedure_*name*'
sp_depends 'Procedure_*name*'
sp_stored_procedures

User-defined Stored Procedures

- RECOMPILE: Thông thường, thủ tục sẽ được phân tích, tối ưu và dịch sẵn ở lần gọi đầu tiên. Nếu tuỳ chọn WITH RECOMPILE được chỉ định, thủ tục sẽ được dịch lại mỗi khi được gọi.

- ENCRYPTION: Thủ tục sẽ được mã hoá nếu tuỳ chọn WITH ENCRYPTION được chỉ định. Nếu thủ tục đã được mã hoá, ta không thể xem được nội dung của thủ tục.

User-defined Stored Procedures

VD: Thủ tục không có tham số

```
CREATE PROC Tong
as
        Declare @a int, @b int
        Set @a =7
        Set @b =3
        Print 'Tong = '+convert(varchar(10),@a+@b)
        Print 'Hieu = '+convert(varchar(10),@a-@b)
        Print 'Tich = '+convert(varchar(10),@a*@b)
        If @b<>0
        Print 'Thuong = '+convert(varchar(10),@a/@b)
        Else
                Print 'Khong chia duoc'
--Thuc thi
EXEC Tong
```

User-defined Stored Procedures

VD: Thủ tục có tham số

```
CREATE PROC Tong1(@a int, @b int)
as
        Print 'Tong = '+convert(varchar(10),@a+@b)
        Print 'Hieu = '+convert(varchar(10),@a-@b)
        Print 'Tich = '+convert(varchar(10),@a*@b)
        If @b<>0
        Print 'Thuong ='+convert(varchar(10),@a/@b)
        Else
                Print 'Khong chia duoc'
--Thực thi
EXEC Tong1 5,7
```

User-defined Stored Procedures

Ví dụ

```
CREATE PROCEDURE London_KH AS
        SELECT * FROM Customers WHERE City=
        'London'
--Thuc thi
Exec LonDon_KH

CREATE PROCEDURE TP_KH (@TP nvarchar(15))
AS
SELECT * FROM Customers WHERE City=@TP
--Thuc thi
Exec TP_KH  'London'
```

Thực thi một Stored Procedure

Cú pháp:

[EXEC [UTE]]
{ [*@return_status* =]
{ *procedure_name* [;*number*] | *@procedure_name_var*
}
[[*@parameter* =] { *value* | *@variable* [OUTPUT] | [DEFAULT]]
[,...*n*]
[WITH RECOMPILE]

Ví dụ 1:

```
EXECUTE TP_KH  'London'
GO
EXECUTE TP_KH  'Paris'
GO
DECLARE @City nvarchar(15), @Return_Value tinyint
SET @City='LonDon'
EXECUTE @Return_Value=TP_KH @City
PRINT @Return_Value
GO
```

Sử dụng tham số

- Có 2 loại tham số:
  - Input parameter: tham số nhập, đưa giá trị của tham số để thông báo cho thủ tục nên làm gì trong CSDL
  - Output parameter: tham số xuất chứa giá trị trả về của thủ tục.
- Khai báo tham số:

  {@parameter data_type} [= default|NULL][varying] [OUTPUT]

Sử dụng tham số

Cú pháp

```
CREATE PROCEDURE procedure_name
@Parameter_name data_type
AS
....
```

Tạo thủ tục với tham số

Ví dụ

```
CREATE PROCEDURE city_KH
@KH_city varchar(15)
AS
    SELECT * FROM Customers
                WHERE City = @KH_city
```

Thực thi thủ tục:

Exec city_KH 'London'

Tạo thủ tục với tham số

Ví dụ:

```
CREATE PROC prcListCustomer @City char(15)
   AS
   BEGIN
    PRINT 'List of Customers'
    SELECT CustomerID,CompanyName,Address,Phone
    FROM Customers  WHERE City = @City
   END

EXEC  prcListCustomer 'LonDon'
```

Tạo thủ tục với tham số

Ví dụ:

```
CREATE PROCEDURE CustOrderHist @CustomerID nchar(5)
AS
SELECT ProductName, Total=SUM(Quantity)
FROM Products P, [Order Details] OD, Orders O, Customers C
WHERE C.CustomerID = @CustomerID
AND C.CustomerID = O.CustomerID AND O.OrderID =
OD.OrderID AND OD.ProductID = P.ProductID
GROUP BY ProductName

Exec CustOrderHist  'NORTS'
```

Thủ tục có trị trả về

- Trị trả về là giá trị kiểu integer.
- Mặc định giá trị trả về là 0

Cú pháp

```
DECLARE @return_variable_name   data_type
EXECUTE @return_variable_name  =
procedure_name
```

Ví dụ tạo thủ tục có giá trị trả về

VD:

```
CREATE PROC Tinhtoan
@a int, @b int , @tong int output, @hieu int output, @tich int output,
 @thuong real output
as
Begin
        Set @tong =@a +@b
        Set @hieu = @a -@b
        Set @tich = @a *@b
        if @b<>0
        begin
                Set @thuong = @a/@b
                Print 'Thuong = '+convert(varchar(10),@thuong)
        end
        Else
                Print 'Khong chia duoc'

        Set @b = @b *100
 End
```

Ví dụ tạo thủ tục có giá trị trả về

```
--Thực thi:
        Declare @tong int, @hieu int, @tich int, @thuong real,
                @a int, @b int
        Set @a= 8
        Set @b=5
        Print 'a = '+convert(varchar(10),@a)
        Print 'b = '+convert(varchar(10),@b)

        EXEC tinhtoan  @a, @b, @tong OUTPUT,@hieu OUTPUT,
         @tich output, @thuong output
        Print 'a = '+convert(varchar(10),@a)
        Print 'b = '+convert(varchar(10),@b)
        Print 'Tong = '+convert(varchar(10),@tong)
        Print 'Hieu = '+convert(varchar(10),@hieu)
        Print 'Tich = '+convert(varchar(10),@tich)
        Print 'Thuong = '+convert(varchar(10),@thuong)
         Go
```

Ví dụ tạo thủ tục có giá trị trả về

VD:

```
CREATE PROCEDURE prcGetUnitPrice_UnitsInStock  @ProductID int,
@Unitprice Money OUTPUT, @UnitsInStock smallint OUTPUT
AS
    BEGIN
        IF EXISTS (SELECT * FROM Products
                    WHERE ProductID = @ProductID)
            BEGIN
               SELECT
                 @Unitprice=Unitprice,@UnitsInStock=UnitsInStock
               FROM Products
               WHERE ProductID=@ProductID
               RETURN 0
            END
        ELSE
            RETURN 1
    END
```

Ví dụ tạo thủ tục có giá trị trả về

```
--Thực thi:
Declare @Unitprice Money, @UnitsInStock smallint

EXEC prcGetUnitPrice_UnitsInStock  1, @Unitprice
   OUTPUT, @UnitsInStock  OUTPUT

Select @Unitprice AS Gia, @UnitsInStock  AS
   SoLuongTon
```

Ví dụ tạo thủ tục có giá trị trả về

Ví dụ

```
CREATE PROCEDURE KH_city
@KH_city VARCHAR(15) AS
DECLARE @KH_return int
SELECT @KH_return=COUNT(*) FROM CUSTOMERS
WHERE City = @KH_city
RETURN @KH_return+1

--Thuc thi
Declare @SoKH int
EXEC @SoKH=KH_city 'LonDon'
Print 'So KH la '+convert(varchar(4),@SoKH)
```

Ví dụ tạo thủ tục có giá trị trả về

VD:

```
CREATE PROCEDURE prcDisplayUnitPrice_UnitsInStock @ProductID int
AS
BEGIN
    DECLARE @UnitPrice Money, @UnitsInStock smallint
    DECLARE @ReturnValue Tinyint
    EXEC @ReturnValue = prcGetUnitPrice_UnitSInStock @ProductID,
            @UnitPrice output, @UnitsInStock output
    IF (@ReturnValue = 0)
     BEGIN PRINT 'The Status for product: '+ Convert(char(10),
           @ProductID)
        PRINT 'Unit price : '  + CONVERT( char(10), @Unitprice)
        PRINT 'Current Units In Stock:' + CONVERT (char(10),
           @UnitsInStock)
      END
     ELSE  PRINT 'No records for the given productID ' +
           Convert(char(10), @ProductID)
END
```

Ví dụ tạo thủ tục có giá trị trả về

```
--Thực thi:
EXECUTE prcDisplayUnitPrice_UnitsInStock 1222
GO
EXECUTE  prcDisplayUnitPrice_UnitsInStock 1
```

Sửa một thủ tục - Stored Procedure

```
ALTER { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
    [ VARYING ] [ = default ] [ [ OUT [ PUT ]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS
     { <sql_statement> [ ...n ] | <method_specifier> }
```

ALTER PROCEDURE KH_city
AS

SELECT * FROM dbo.Customers;

Exec KH_city

Sửa một thủ tục - Stored Procedure

- Example:

```
ALTER PROC prcListCustomer @City char(15)=NULL
AS
    BEGIN
    IF @city is NULL
    BEGIN
      PRINT 'Usage: prcListCustomer <City>'
      RETURN
    END
      PRINT 'List of Customers'
     SELECT CustomerID,CompanyName,Address,Phone
     FROM Customers
     WHERE City = @City
    END
```

Sửa một thủ tục - Stored Procedure

- VD:

```
ALTER PROC prcListCustomer @City char(15)
AS
BEGIN
  IF EXISTS (SELECT * FROM Customers WHERE City=@city)
     BEGIN
    PRINT 'List of Customers'
    SELECT CustomerID,CompanyName,Address,Phone
    FROM Customers  WHERE City = @City
    RETURN 0
      END
   ELSE
      BEGIN
    PRINT 'No Records Found for given city'
    RETURN 1
      END
END
```

Xóa một Stored Procedure

DROP PROCEDURE *proc_name*

Ví dụ:

DROP PROCEDURE City_KH

HÀM (Function)

- Khái niệm về Hàm
- Các loại hàm
- Các loại giá trị trả về của UDFs
- Tạo và quản lý hàm UDFs
- Scalar Function
- Table-valued Function
- Sử dụng hàm UDFs

Khái niệm về Hàm

- Hàm tương tự thủ tục bao gồm các phát biểu T-SQL và một số cấu trúc điều khiển được lưu với một tên và được xử lý như một đơn vị độc lập. Hàm được biên dịch trước, không cần kiểm tra và biên dịch lại.
- Điểm khác biệt giữa hàm và thủ tục là hàm trả về một giá trị thông qua tên hàm còn thủ tục thì không.

Khái niệm về Hàm

- **Hàm được dùng trong:**
  - Lệnh Print hay lệnh Select để hiển thị giá trị trả về của hàm.
  - Danh sách chọn của một câu lệnh Select để cho ra một giá trị.
  - Một điều kiện tìm kiếm của mệnh đề Where trong các câu lệnh T-SQL.

Ưu điểm của Hàm

Người gởi chỉ gởi một câu lệnh đơn và SQL Server chỉ kiểm tra một lần sau đó tạo ra một execute plan và thực thi. Cú pháp của các câu lệnh SQL đã được SQL Sever kiểm tra trước khi save nên nó không cần kiểm lại khi thực thi→giảm nghẽn mạng

Bảo trì (maintainability) dễ dàng hơn do việc tách rời giữa business rules và database. Nếu có một sự thay đổi nào đó về mặt logic thì ta chỉ việc thay đổi code bên trong hàm mà thôi

Security: có thể được encrypt (mã hóa) để tăng cường tính bảo mật.

Các loại Hàm

- **Có hai loại:**
  - **Built-in functions:** Hoạt động như là một định nghĩa trong T-SQL và không thể hiệu chỉnh. Chỉ được tham chiếu trong các câu lệnh T-SQL. Trị trả về là một tập các dòng(Rowset), vô hướng(scalar) và aggregate(thống kê).
  - **User-define functions** hay còn gọi là UDFs: do người dùng tự định nghĩa để đáp ứng một mục tiêu nào đó. Các tham số truyền vào không được mang thuộc tính OUTPUT, do đó giá trị trả về cho hàm bằng phát biểu RETURN. Giá trị trả về là giá trị vô hướng (Scalar valued) hay bảng (Table – valued).

Các loại giá trị trả về của UDFs

- Scalar Function: Một hàm vô hướng trả về một giá trị đơn và có thể được dùng bất cứ nơi nào của biểu thức hay có thể được dùng câu lệnh SELECT, mệnh đề SET của lệnh UPDATE,... Một hàm vô hướng có thể được xem như kết quả của vài phép toán hay hàm chuỗi.
- Table-valued Function : Một hàm có giá trị trả về là một tập kết quả và có thể được dùng bất cứ nơi nào mà bảng hay view được dùng. Hàm giá trị bảng có thể được tham chiếu trong mệnh đề FROM của câu lệnh SELECT.
- **Tên và những thông tin về Function khi được tạo ra sẽ chứa trong SysObjects table còn phần text của nó chứa trong SysComments table.**

Các lệnh tạo và quản lý UDF

- **Tạo Hàm**

  CREATE FUNCTION <TenHam>…

- **Sửa Hàm**

  ALTER FUNCTION <TenHam>…

- **Xóa Hàm**

  DROP FUNCTION statement

- **Thực thi Hàm**

  Dùng lệnh **Print**

  Dùng Lệnh **Select**

- **Xem các lệnh của UDFs**

  Sp_helptext TenHam

Scalar Function

1. Scalar Function Không có tham số

- **Là hàm không nhận giá trị từ bên ngoài truyền vào.**
- Cú pháp:

```
CREATE FUNCTION
    [Owner_name.]function_name
RETURNS scalar_return_data_type
[WITH { ENCRYPTION | SCHEMABINDING } ]
[ AS ]
    BEGIN
            function_body
            RETURN scalar_expression
    END
```

Scalar Function – Tạo Hàm

- **Ví dụ 1 : Hàm trả về tổng 2 số 4 và 6**

```
Create Function tong2so()
Returns int
as
Begin
   Declare @so1 int, @so2 int
   Set @so1 = 4
   Set @so2 =6
   Return @so1+@so2
End
--thuc hien
Print 'Tong = ' +convert(char(10),dbo.tong2so())
Select dbo.tong2so() as Tong
```

Scalar Function – Sửa và Xóa Hàm

```
Alter  Function tong2so()
Returns int
With Encryption
as
Begin
        Declare @so1 int, @so2 int
        Set @so1 = 4
        set @so2 =6
        Return @so1+@so2
End
--Xem lệnh
        sp_helptext tong2so
--Thực hiện
        print 'Tong = ' +convert(char(10),dbo.tong2so())
        select dbo.tong2so() as Tong
--Xóa hàm
            Drop function Tong2so
```

Scalar Function

Ví dụ 2 : Hàm trả về tổng tiền của khách hàng có mã là TOMSP

```
Create function Tongtien()
Returns money
AS
Begin
        Declare @tong money
        Select @tong = sum(unitprice*Quantity) from orders o,
                        [Order Details] d
        where o.orderid = d.orderid and customerid = 'TOMSP'
        Return @tong
End
print 'Tong = ' +convert(char(10),dbo.tongtien())
select dbo.tongtien() as [Tong Tien Cua Khach Hang TOMSP]
```

Scalar Function

2. Scalar Function Có tham số

- **Là hàm nhận các giá trị từ bên ngoài truyền vào.**
- Cú pháp:

```
CREATE FUNCTION [owner_name.]function_name
([{@parameter_name [AS] data_type [=default]} [
    ,...n ]])
RETURNS scalar_return_data_type
[WITH { ENCRYPTION | SCHEMABINDING } ]
[ AS ]
    BEGIN
            function_body
            RETURN scalar_expression
    END
```

Scalar Function

Ví dụ : Hàm trả về tổng của hai số bất kỳ

```
Create function tong(@so1 int, @so2 int)
Returns int
as
Begin
        Return @so1+@so2
End
-- Thuc hien ham
Declare @a int, @b int
Set @a = 4
Set @b =6
Print 'Tong cua '+convert(char(5),@a) +'  '+

        convert(char(5),@b)+'='+convert(char(5),dbo.tong(@a,@b))
Select dbo.tong(@a,@b) as tong
```

Scalar Function

VD: Hàm trả về tổng tiền của khách hàng nào đó

```
Create function TongtienTS(@makh nchar(5))
Returns money
AS
Begin
    Declare @tong money
    Select @tong = sum(unitprice*Quantity) From orders o,  [Order
    Details] d
    Where o.orderid = d.orderid and customerid = @makh
    Return @tong
End
Declare @ma nchar(5)
Set @ma = 'TOMSP'
Print 'Tong = ' +convert(char(10),dbo.tongtients(@ma))
Select dbo.tongtients(@ma) as Tong
```

Scalar Function

Bài tập áp dụng : Hàm trả về thứ bằng tiếng Việt

```
Create function thu(@ngay datetime)
Returns varChar(10)
As
Begin     Declare @t varchar(10), @d tinyint
          Set @d = datepart(dw,@ngay)
          Set @t = case
                    When @d = 1 then 'Chu Nhat'
                    When @d = 2 then 'Hai'
                    When @d = 3 then 'Ba'
                    When @d = 4 then 'Tu'
                    When @d = 5 then 'Nam'
                    When @d = 6 then 'Sau'
                    When @d = 7 then 'Bay'
                    end
          Return @t
End
```

Scalar Function

--Thực thi

```
Declare @ngaysinh datetime
Set @ngaysinh = getdate()
Print 'Ban sinh vao Thu '+dbo.thu(@ngaysinh) +
 ' Ngay '+ convert(char(3),day(@ngaysinh)) + ' thang ' +
Convert(char(3), month(@ngaysinh))+' nam '
+convert(char(5),year(@ngaysinh))
```

--Thuc hien voi cau lenh Select

```
Select employeeid, LastName +'  '+FirstName as Hoten, thu =
dbo.thu(birthdate) from Employees
Select employeeid, LastName +' '+FirstName as Hoten, [Thu Ngay
Thang Nam Sinh] ='Thu '+dbo.thu(birthdate) +' Ngay '+
convert(char(2),day(birthdate)) + ' thang ' + Convert(char(2),
month(birthdate))+ ' nam ' +convert(char(4),year(birthdate))
from Employees
```

Scalar Function

BT2 : Hàm trả về Tổng tiền của các sản phẩm

```
Create function TotalAmount
(@Unitprice money, @quantity Smallint,@Discount real)
Returns Money
As
Begin
        Return (@Unitprice * @Quantity)*(1-@discount)
End
--Su dung
Select Productid, Total =
dbo.TotalAmount(Unitprice,Quantity,Discount)
From [Order Details]
Where Orderid =10250
```

Scalar Function

BT tự làm:

Viết hàm trả về chiết khấu của sản phẩm dựa vào số lượng lập hoá đơn và theo quy định sau:

-Nếu số lượng <=5 thì chiết khấu là 0.05

-Nếu số lượng từ 6 đến 10 thì chiết khấu 0.07

-Nếu số lượng từ 11 đến 20 thì chiết khấu là 0.09

ngược lại thì chiết khấu là 0.1

The table-valued UDFs

- The table-valued UDFs**: được chia thành hai loại là inline và multistatement table-valued.**
- *Inline table-valued UDF:*
  - *Được xem như là một View có tham số*. Thực thi một câu lệnh Select như trong một view nhưng có thể bao gồm các tham số giống thủ tục
  - Cú pháp:

CREATE FUNCTION [*owner_name.*]*function_name*

([{*@parameter_name* [AS] *data_type* [=*default*]} [,...n]])

RETURNS TABLE

[WITH { ENCRYPTION | SCHEMABINDING }]

[AS]

RETURN [(] select-statement [)]

The table-valued UDFs

Ví dụ 1: Cho biết tổng số hóa đơn của khách hàng bất kỳ.

```
CREATE FUNCTION  CountOrderCust (@cust  varchar(5))
RETURNS TABLE
AS
RETURN (Select CustomerID, count(orderid) as countOrder
        From orders
        Where customerID like @cust
        Group by customerID )
```

- Thi hành (không cần tên đầy đủ)

```
Select  *  From CountOrderCust('A%' )
```

-
```
Declare @ma nvarchar(5)
Set @ma='A%'
Select * from CountOrderCust(@ma)
```

The table-valued UDFs

Ví dụ 2 : trả về tổng số lượng của từng sản phẩm theo loại hàng nào đó.

```
CREATE FUNCTION SalesByCategory(@Categoryid Int)
RETURNS TABLE
AS
RETURN
(SELECT c.CategoryName, P. ProductName,
                SUM(Quantity) AS TotalQty
  FROM Categories c
  INNER JOIN Products p ON c.CategoryID= p. CategoryID
  INNER JOIN [Order Details] od ON p.ProductID = od.ProductID
  WHERE c.CategoryID= @Categoryid
  GROUP BY c. CategoryName,p.ProductName)
```

- Thực thi

```
SELECT * FROM SalesByCategory (1)
```

The table-valued UDFs

- Multistatement Table-valued UDF: là dạng phức tạp nhất. Loại hàm này xây dựng tập kết quả từ một hay nhiều câu lệnh Select
- Cú pháp:

```
CREATE FUNCTION [owner_name.]function_name
([{@parameter_name [AS] data_type [=default]} [ ,...n ]])
RETURNS @return_variable
TABLE ({column_definition | table_constraint} [ ,...n ])
[WITH { ENCRYPTION | SCHEMABINDING } ]
[AS]
BEGIN
        function_body
RETURN
END
```

The table-valued UDFs

Ví dụ 1

```
CREATE FUNCTION  CountOrderCust()
RETURNS  @fn_CountOrderCust  TABLE
(OrderIdent  tinyint Not null, Cust varchar(5) )
AS
Begin
     Insert @fn_CountOrderCust
     Select Count(orderid),CustomerId From Orders
      Group by Customerid
     Return
End
--Thi hành
Select * From CountOrderCust()
```

The table-valued UDFs

Ví dụ 2

```
CREATE FUNCTION Contacts(@suppliers bit=0)
RETURNS @Contacts TABLE (ContactName nvarchar(30), Phone nvarchar(24),
    ContactType nvarchar(15))
AS BEGIN
    INSERT @Contacts
    SELECT ContactName, Phone, 'Customer' FROM Customers
    INSERT @Contacts
    SELECT FirstName + ' ' + LastName, HomePhone, 'Employee'
    FROM Employees
    IF @Suppliers=1
        INSERT @Contacts
    SELECT ContactName, Phone, 'Supplier'  FROM Suppliers
     RETURN
END
-- Thực thi
SELECT * FROM CONTACTS(1) ORDER BY ContactName
```

Sử dụng UDFs

- Scalar UDF: khi gọi luôn luôn theo cú pháp: *owner.functionname.*

  Ví dụ:

  ```
  SELECT ProductID, Total=dbo.TotalAmount(UnitPrice, Quantity,
  Discount)
  FROM [Order details]
  WHERE OrderID=10250
  ```

- Scalar UDF có thể được sử dụng trong biểu thức, trong câu lệnh SELECT hay lệnh CREATE TABLE

  ```
  CREATE TABLE [Order Details] (
  OrderID int NOT NULL , ProductID int NOT NULL ,
  UnitPrice money NOT NULL DEFAULT (0),
  Quantity smallint NOT NULL DEFAULT (1),
  Discount real NOT NULL DEFAULT (0),
  Total AS dbo.TotalAmount(UnitPrice, Quantity, Discount))
  ```

Sử dụng UDFs

- A table-valued UDF: Có thể được gọi theo cú pháp *owner.functionname hay functionname*

  SELECT * FROM Contacts(1) ORDER BY ContactName

- Nếu table-valued function không có tham số, bạn phải sử dụng dấu()
- Nếu tham số có giá trị mặc định, bạn phải truyền giá trị vào mặc dù bạn có sử dụng từ khóa DEFAULT

  SELECT * FROM Contacts() ORDER BY ContactName

Sử dụng UDFs

- **Bài tập tự làm:**

  Viết hàm trả về danh sách các hoá đơn đã lập của một khách hàng nào đó trong một tháng năm nào đó. Thông tin gồm: Makh, TenKh, Diachi, mahd, ngaylapHD, Noichuyen, LoaiHD. Trong đó, LoaiHD được hiển thị rõ là Nhập hoặc Xuất.