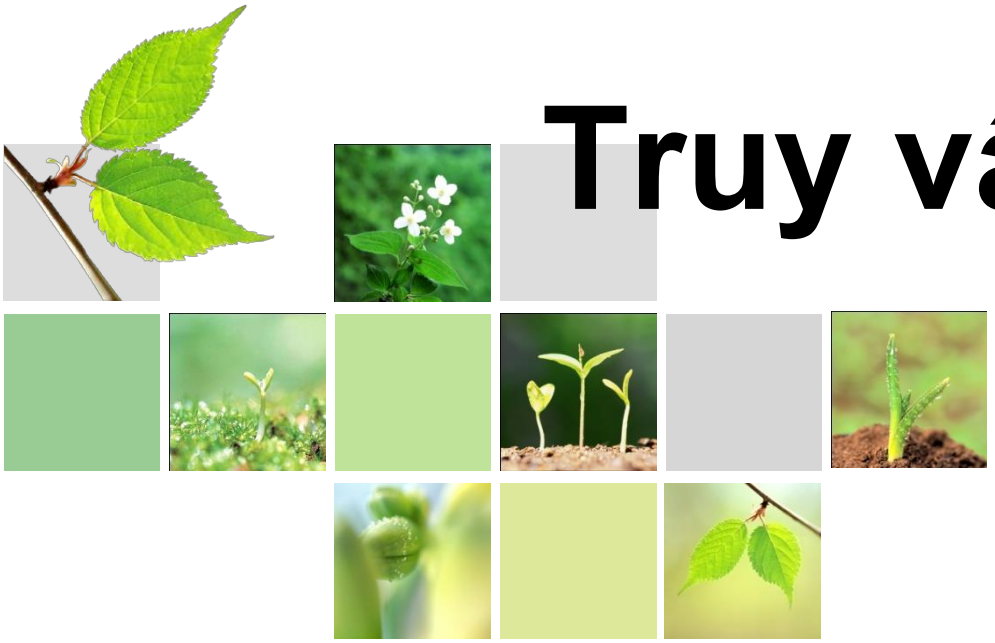
Chương 3

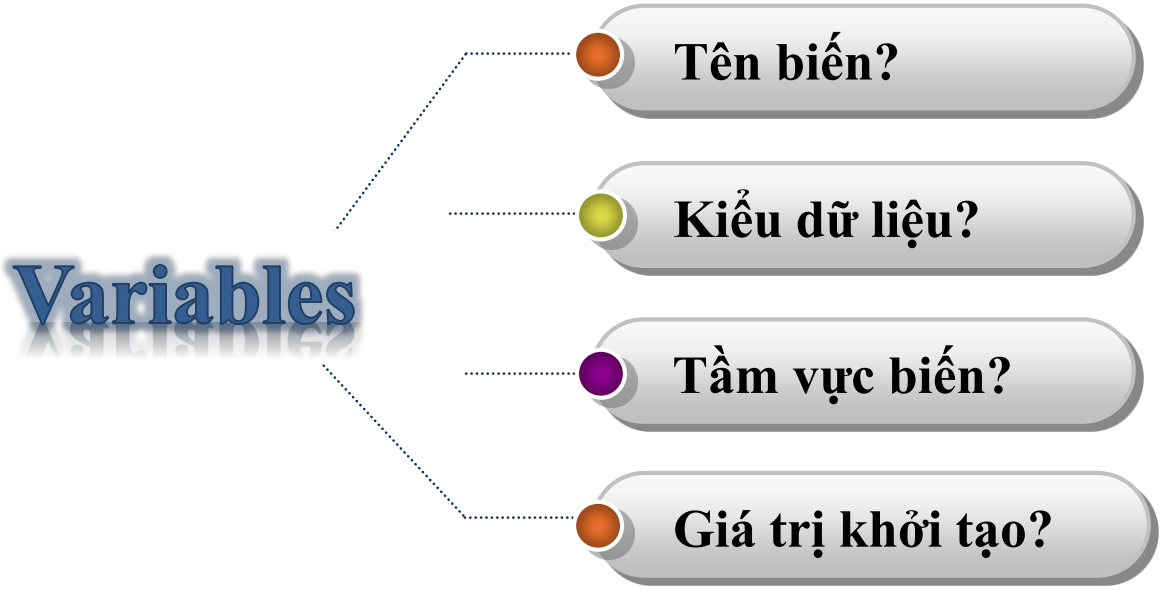
Truy vấn nâng cao



Nội dung

1. Cấu trúc lệnh
2. Thủ tục thường trú
3. Kiểu dữ liệu cursor
4. Hàm người dùng

Khai báo biến



Khai báo biến

❖Cú pháp

- Declare *Var_name* Datatype
- Lưu ý: Tên biến phải bắt đầu bằng 1 ký tự @

❖Ví dụ

- Declare @MaSinhVien nvarchar(10)
- Declare @TienLuong float
- Declare @Sum float, @Count int
- Declare @temp TABLE (ma int,
  ten nvarchar(10))

Khai báo biến

❖Tầm vực biến

- Biến cục bộ có ý nghĩa trong một *query batch* hay một thủ tục thường trú hoặc một hàm người dùng
- Biến hệ thống có ý nghĩa trên cả hệ thống. Tên của chúng bắt đầu bằng @@. Các biến này là read-only.
- Ví dụ biến hệ thống : @@*fetch_status*, @@*rowcount*, @@*trancount*…

Lệnh gán

Set @TenBien = GiaTri

Set @TenBien = TenBien

Set @TenBien = BieuThuc

Select @TenBien = (KetQuaTruyVan)

❖Ví dụ :

Set @MaLop = 'TH2001'

Set @SoSV = (select count(*) from SinhVien)

Set @MaLop = 'TH' + CAST

(Year(@NgayTuyenSinh) AS char(4))

Lệnh gán

- Cũng có thể gán giá trị cho biến bằng câu truy vấn thay vì chỉ thị **set**
- Ví dụ :

SV(<u>MaSV</u>, HoTen, Tuoi)

```
Select @Var2 = HoTen, @Var1 = Tuoi
from  SV
where MaSV = 1
```

Kiểu dữ liệu phải tương ứng. Nếu câu truy vấn trả về nhiều dòng thì các biến chỉ nhận giá trị từ dòng đầu tiên

Lệnh gán

- Cũng có thể gán giá trị cho biến bằng câu truy vấn thay vì chỉ thị **set**
- Ví dụ :

NhanVien(MaNV, HoTen, NgaySinh)

```
Declare @Var1 datetime
Select @Var1 = NgaySinh
from  NhanVien
where MaNV = 1
If (year(getdate()) – year(@Var1) > 50)
        .......
```

Cấu trúc điều khiển

Cú Pháp

If <logial expression>

[Begin]

    Code block

[End]

Else

[Begin]

    Code block

[End]

Có thể chứa các câu truy vấn phức tạp tùy ý

- Khai báo biến
- Các tính toán trên biến
- Các câu truy vấn phức tạp tùy ý
- …

Optional

Cấu trúc điều khiển

```
If logial expression
[Begin]
    Code block
[End]
[Else if logial expression
[Begin]
    Code block
[End]
[,…n]]
Else
[Begin]
    Code block
[End]
```

Có thể lặp lại nhiều lần tùy ý. Mô phỏng cấu trúc case

Cấu trúc điều khiển

WHILE *<Logical_expression>*

[Begin]

{ *sql_statement* | *statement_block* }

[BREAK] — Thoát vòng lặp

{ *sql_statement* | *statement_block* }

[CONTINUE] — Bỏ qua đoạn lệnh sau

[End]

Cấu trúc điều khiển

- Tính tổng các số nguyên chẵn từ 1 đến n
- Đếm và in ra số lượng các số nguyên chia hết cho 3 nằm trong đoạn từ 1 đến n
- Cho 3 số a, b, c. Tìm số lớn nhất. In giá trị của a, b, c. Xuất thông báo "Số lớn nhất là :"

Cấu trúc điều khiển

❖ ***Ví dụ***

HocPhan(MaHP, TenHP, SiSo)

DangKy(MaSV, MaHP)

Viết lệnh để thêm một đăng ký mới cho sinh viên có mã số 001 vào học phần HP01 (giả sử học phần này đã tồn tại trong bảng HocPhan). Qui định sỉ số lớp cho mỗi học phần không quá 50 sv

Cấu trúc điều khiển

❖Ví dụ

NhanVien (MaNV: int, HoTen: nvarchar(30))

Viết lệnh xác định một ma nhan vien mới theo qui định: mã nhan vien tăng dần, nếu có chỗ trống thì mã mới xác định sẽ chèn vào chỗ trống đó

Vd: 1,2,3,7 → mã nhan vien mới: 4

Cấu trúc điều khiển

❖Ví dụ

```
Declare @STT int
Set @STT = 1
While exists (select * from SV
                    where MaSV = @STT)
              set @STT = @STT+1
Insert into SV(MaSV, HoTen)
values(@STT, 'Nguyen Van A')
```

Cấu trúc điều khiển

```
CASE [input_expression]
    WHEN when_expression THEN result_expression
    [ ...n ]
    [ ELSE else_result_expression ]
END
```

Có thể là giá trị hoặc biểu thức điều kiện

Cấu trúc điều khiển

❖ ***Ví dụ***:

NHAN_VIEN(MaNV, HoTen, NgaySinh, CapBac, Phai)

Cho biết những nhân viên đến tuổi về hưu (tuổi về hưu của nam là 60, của nữ là 55)

Cấu trúc điều khiển

```
Select * From NHAN_VIEN
Where datediff(yy, NgaySinh, getdate())
 > = Case Phai
          when ‘Nam’ then 60
          when ‘Nu’ then 55
     End
```

Cấu trúc điều khiển

- Cho biết mã NV, họ tên và loại nhân viên (cấp bậc <=3:bình thường, cấp bậc = null: chưa xếp loại, còn lại: cấp cao)

```
Select MaNV, HoTen, 'Loai' =  Case
    when CapBac<=3 then 'Binh Thuong'
    when CapBac is null then 'Chua xep loai'
    else 'Cap Cao'    End
From NhanVien
```

Bài tập 1

Cho CSDL:

SinhVien (MaSV, Hoten, DiemTB)

Tìm sinh viên có điểm trung bình lớn nhất và xuất thông báo theo yêu cầu sau:

- **Nếu điểm TB >= 8.0**
  - **[MaSV] - Điểm trung bình [DiemTB] – Xếp loại : Giỏi**
- **Nếu điểm TB >= 6.5**
  - **[MaSV] - Điểm trung bình [DiemTB] – Xếp loại : Khá**
- **Nếu điểm TB >= 5.0**
  - **[MaSV] - Điểm trung bình [DiemTB] – Xếp loại : Trung bình**
- **Ngược lại**
  - **[MaSV] - Điểm trung bình [DiemTB] – Xếp loại : Yếu**

Bài tập 2

Cho CSDL:

SinhVien(MaSV, HoTen, NgaySinh)

Tìm sinh viên có MaSV = '0912033' với định dạng như sau:

Mã SV : 0912033

Họ tên : Nguyễn Kim Ái

Ngày sinh : 20/9/1990

Bài tập 3

Cho CSDL:

SinhVien(MaSV, HoTen, NgaySinh)

DiemThi(MaSV, MaMH, Diem)

Tính điểm trung bình của một sinh viên. Nếu sinh viên có điểm trung bình > 5.0 thì in là 'đậu' ngược lại 'rớt'. In dưới dạng bảng.

Ví dụ:

MaSV	HoTen	Điểm TB	Kết quả
0912033	Nguyễn Kim Ái	4.5	Rớt

Bài tập 4

Cho CSDL:

SinhVien(MaSV, HoTen, NgaySinh)

DiemThi(MaSV, MaMH, Diem)

Kiểm tra MaSV = 0912003 có tồn tại chưa

- **Nếu chưa tồn tại xuất thông báo [MaSV] chưa tồn tại.**
- **Ngược lại, xuất thông báo [MaSV] sinh viên đã tồn tại.**

Bài tập 5

Cho CSDL:

MonHoc(MaMH, TenMH,SoChi)

Kiểm tra MaMH đã tồn tại chưa?

- **Nếu tồn tại rồi xuất thông báo "[MaMH] đã tồn tại"**
- **Ngược lại, phát sinh MaMH mới và in thông báo "Mã MH mới là [MaMHmoi]"**

Ví dụ:

Tìm được MaMH lớn nhất là : MH008

Phát sinh MaMH mới = MH009

Nội dung

1. Cấu trúc lệnh
2. Thủ tục thường trú
3. Kiểu dữ liệu cursor
4. Hàm người dùng

Thủ tục thường trú

- ***Thủ tục***:
  - Chứa các lệnh T_SQL
  - Tương tự như một thủ tục trong các ngôn ngữ lập trình: có thể truyền tham số, có tính tái sử dụng
- ***Thường trú***:
  - Được dịch và lưu trữ thành một đối tượng trong CSDL

STORE PROCEDURES (THỦ TỤC)

- **Stored Procedure (Thủ tục)** là tập hợp một hoặc nhiều câu lệnh T-SQL thành một nhóm đơn vị xử lý logic và được lưu trữ trên Database Server.
- Có tham số hoặc không tham số
- Khi một câu lệnh gọi chạy *Stored Procedure* lần đầu tiên thì HQTCSDL sẽ thực thi và lưu trữ vào bộ nhớ đệm (plan cache), những lần tiếp theo HQTCSDL sẽ sử dụng lại plan cache → tốc độ xử lý tối ưu

Thủ tục thường trú

Stored-Procedure

- Tên thủ tục?
- Tham số vào?
- Tham số ra?
- Giá trị trả về?
- Yêu cầu xử lí?

Ý nghĩa

- Tính tái sử dụng
- Tối ưu hóa khi biên dịch
- Giảm lượng thông tin trao đổi
- Đảm bảo an CSDL an toàn hơn
- Đơn giản hóa việc lập báo cáo

Cú pháp

Create {**proc** | **procedure**} *proc_name*

Parameter DataType [**output**] [,…n]

As

Code block

[**return** [*return_value*]]

Go

Tên của stored .Nên bắt đầu với **USP**

Kiểu DL của tham số

Giá trị trả ra nếu có thì dùng một (hay một số) tham số output

Tên tham số (đặt như tên biến)

Thân sủa SP, viết như thế nào là tùy vào từng bài toán cụ thể

Chỉ trả về giá trị int

Ví dụ

```
SELECT
sp_Ten, sp_Gia
FROM
 SanPham
ORDER BY
  sp_Ten;
```

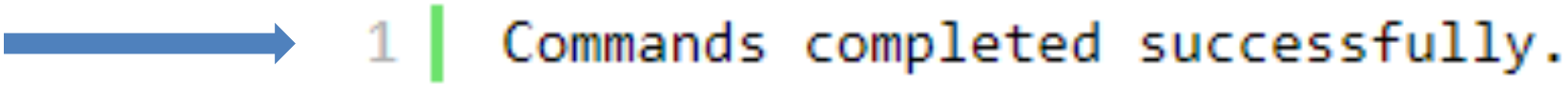
Truy vấn lấy ds sản phẩm

```
CREATE PROCEDURE DS_sanPham
AS
BEGIN
     SELECT sp_Ten, sp_Gia
     FROM
          SanPham
     ORDER BY
          sp_Ten;
END;
```

Tạo proc lấy ds sản phẩm

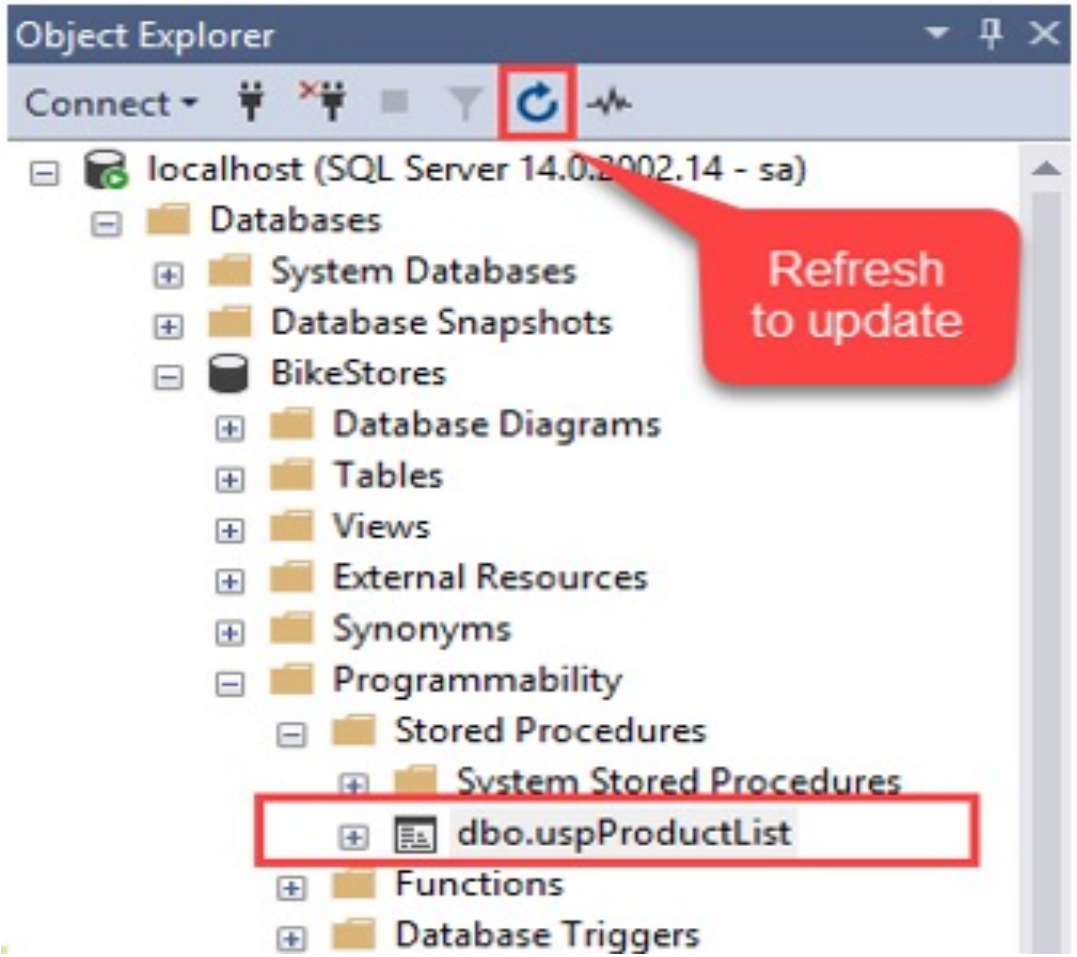
Thực thi

- Gọi thực thi thủ tục: chạy câu SQL tạo stored bằng cách click vào nút Execute ở trên thanh công cụ



Thực thi

- Xem danh sách Proc đã tạo



Thực thi

- Thực thi bằng câu lệnh

```
EXECUTE DS_sanPham;
```

Hoặc

```
EXEC DS_sanPham;
```

Trong MYSQL

```
Call DS_sanPham;
```

Chỉnh sửa/ Thay đổi

- Sử dụng lệnh ALTER PROCEDURE và tên của stored sẽ nằm phía sau.
  - Ví dụ: Thay đổi cách sắp xếp kết quả trả về từ sp_ten thành sp_gia

```
ALTER PROCEDURE uspProductList
    AS
    BEGIN
        SELECT
            sp_Ten,
            sp_gia
        FROM
            SanPham
        ORDER BY
            sp_gia
    END;
```

Xoá

- Thực thi bằng câu lệnh

```
DROP PROCEDURE DS_sanPham;
```

Hoặc

```
DROP PROC DS_sanPham;
```

Truyền tham số

- Lấy danh sách sản phẩm có giá lớn hơn một giá cụ thể nào đó khi thực thi Stored Proc

```
CREATE PROCEDURE ds_SanPham(@min_giaSP AS
DECIMAL)
AS
BEGIN
    SELECT
        sp_Ten,sp_Gia
    FROM
        SanPham
    WHERE
        sp_Gia >= @min_giaSP
    ORDER BY
         sp_Gia;
END;
```

Truyền tham số

- Tham số sẽ nằm trong cặp dấu ngoặc ( )
- Mỗi tham số phải khai báo tên và kiểu dữ liệu tương ứng thông qua từ khóa as.

- Sử dụng tham số ở bên trong phần thân của Stored Proc: @min_giaSP

```
CREATE PROCEDURE ds_SanPham
     (@min_giaSP AS DECIMAL)
AS
BEGIN …
          WHERE
          sp_Gia >= @min_giaSP
     …
END;
```

Thực thi

- Thực thi bằng câu lệnh

```
EXECUTE DS_sanPham 20000;
```

Hoặc

```
EXEC DS_sanPham 20000;
```

Truyền tham số

- Lấy danh sách sản phẩm có giá nằm trong khoảng min và max.

```
ALTER PROCEDURE TimSP(
     @min_giaSP AS DECIMAL
     ,@max_giaSP AS DECIMAL)
AS
BEGIN
     SELECT
          sp_Ten, sp_gia
     FROM
          SanPham
     WHERE
          sp_gia>= @min_giaSP AND
          sp_gia <= @max_giaSP
     END;
```

Thực thi

- Gán giá trị mặc định

```
ALTER PROCEDURE TimSP(
    @min_giaSP AS DECIMAL = 20000,
    @max_giaSP AS DECIMAL = 100000)
AS
BEGIN
    SELECT
    …
END;
```

Thực thi

- Thực thi: lấy danh sách sản phẩm có giá nằm trong khoảng **min** và **max**:

```
EXECUTE TimSP 20000, 100000
```

Scalar input parameters

- **Unamed**

```
CREATE PROC USP_XemSV
        @MaSV Char(10) = NULL
AS
BEGIN
        IF @MaSV is NULL
                SELECT * FROM SINHVIEN
        ELSE
                SELECT *
                FROM SINHVIEN
                WHERE MaSV = @MaSV
END
```

```
EXEC USP_XemSV
EXEC USP_XemSV '0912311'
```

Scalar input parameters

- **Named**

```
CREATE PROC USP_XemSV
        @MaSV Char(10)
AS
BEGIN
        IF @MaSV is NULL
                SELECT * FROM SINHVIEN
        ELSE
                SELECT *
                FROM SINHVIEN
                WHERE MaSV = @MaSV
END
```

EXEC USP_XemSV '0912311'

Scalar output parameters

Thống kê doanh thu của mỗi sản phẩm

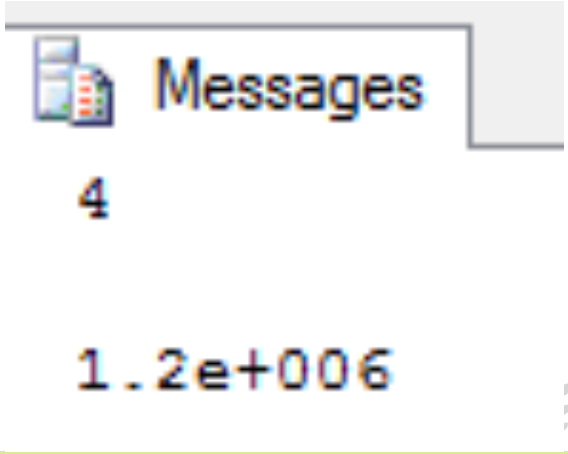
```
CREATE PROC USP_ThongKe
        @MaSP Char(10),
        @TongSLBan int output,
        @TongDoanhThu float output
AS
```

Scalar output parameters

```
BEGIN
        --Tính tổng số lượng
        SET @TongSLBan = (SELECT SUM(SoLuong)
                                FROM CHITIETPHIEUDAT
                                WHERE MaSanPham = @MaSP)
        --Tính tổng doanh thu
        SET @TongDoanhThu =
                                (SELECT SUM(SoLuong * DonGia)
                                FROM CHITIETPHIEUDAT
                                WHERE MaSanPham = @MaSP)
END
```

Scalar output parameters

```
--Gọi thực thi
DECLARE @TongSL int, @TongDT float
EXEC USP_ThongKe      'SP00000001',
                      @TongSL output,
                      @TongDT output
PRINT CAST(@TongSL AS Char(3)) + Char(13)
PRINT @TongDT
```



Bài tập 1

Cho CSDL:

SinhVien(MaSV, HoTen, NgaySinh)

Viết thủ tục tìm kiếm thông tin sinh viên với tham số truyền vào là mã số sinh viên

Đáp án BT1

```
CREATE PROCEDURE GetStudent
        @masv INT
AS
BEGIN
      SELECT MaSV, HoTen, NgaySinh Position
      FROM SinhVien WHERE MASV = @masv;
END;
GO
```

Bài tập 2

Cho CSDL:

SinhVien(MaSV, HoTen, NgaySinh)

Viết thủ tục thêm thông tin sinh viên mới khi đã kiểm tra nếu sinh viên chưa tồn tại?

Đáp án BT2

```
CREATE PROCEDURE CheckAndInsertStudent
        @masv char(20), @hoten nvarchar(50),
        @ngaysinh date
AS
BEGIN
        NOT EXISTS (SELECT * FROM SinhVien WHERE masv = @masv)
        BEGIN
                INSERT INTO SinhVien VALUES (@masv, @hoten, @ngaysinh)
                PRINT 'Đã thêm thông tin thành công';
        END
        ELSE BEGIN
                PRINT 'Đã tồn tại sinh viên này';
        END
END;
```

Bài tập 3

Cho CSDL:

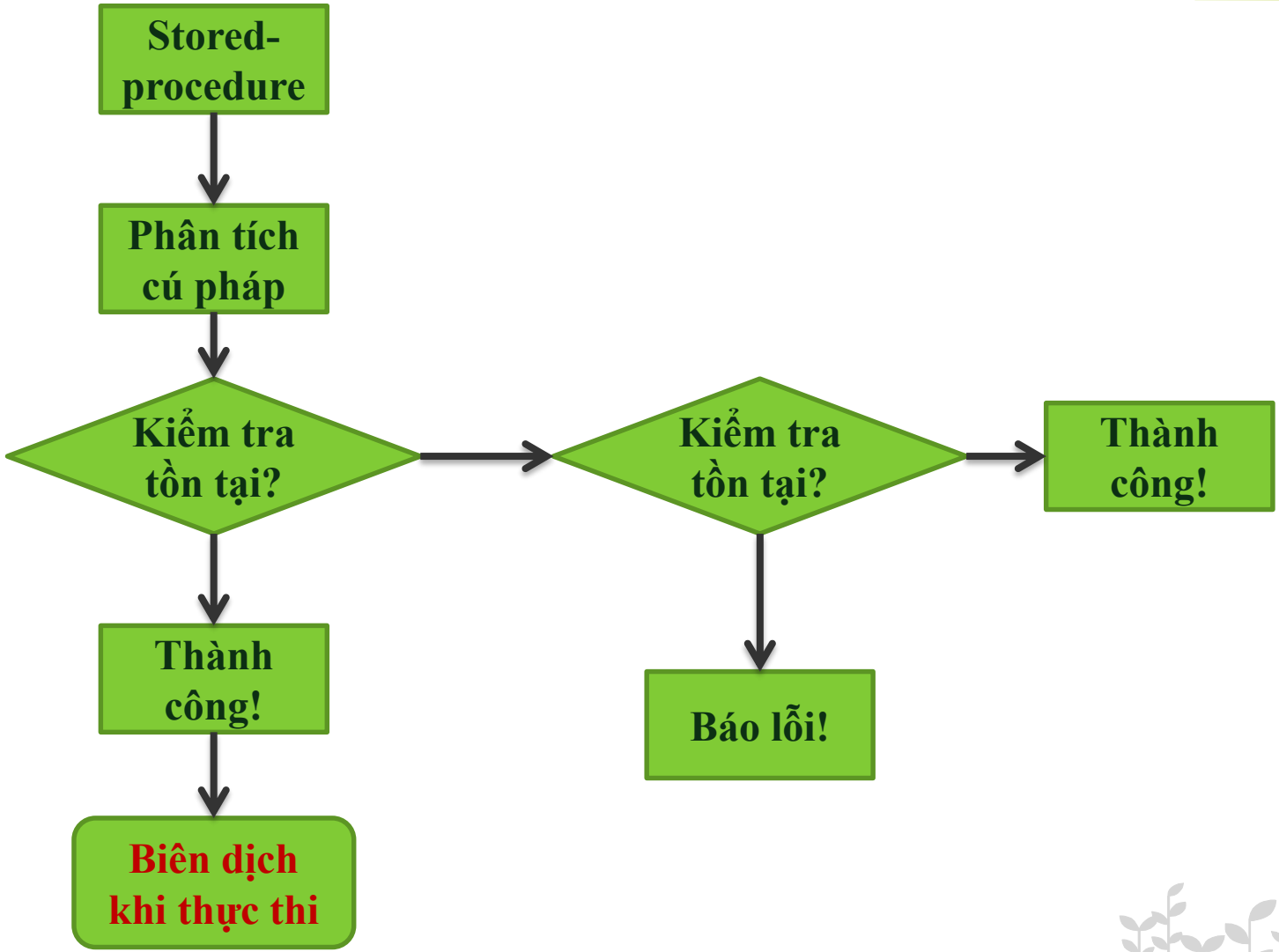
SACH (masach, ten, loai, tacgia, nxb, tinhtrang)

THEMUON (masach, madocgia, ngaymuon, ngaytra)

Viết thủ tục tiến hành cho mượn sách

- **Kiểm tra tình trạng sách sẵn có trước khi cho mượn**
- **Thêm thông tin vào Themuon và cập nhật lại tình trạng sách**

Stored-Procedure



Thủ tục lồng nhau

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

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

Nội dung

1. Cấu trúc lệnh
2. Thủ tục thường trú
3. Kiểu dữ liệu cursor
4. Hàm người dùng

Cursor – Khái niệm

- Là một cấu trúc dữ liệu ánh xạ đến một tập các dòng dữ liệu là kết quả của một câu truy vấn (select)
- Cho phép duyệt tuần tự qua tập các dòng dữ liệu và đọc giá trị từng dòng.

Cursor – khái niệm

- Vị trí hiện hành của *cursor* có thể được dùng như điều kiện trong mệnh đề *where* của lệnh *update* hoặc *delete*
  - Cho phép cập nhật / xoá dữ liệu (dữ liệu thật sự trong CSDL) tương ứng với vị trí hiện hành của cursor

Cursor – khai báo

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

    Declare *cur_name* **Cursor**

    For *select_statement*

    [**For** {**Read only**| **Update** [**of** *column_name* [,…n]] }]

Cursor – Khai báo

- *Cú pháp mở rộng*

  Declare cursor_name **Cursor**

  [**Local** | **Global**]

  [**Forward_only**| **Scroll**]

  [**Static**| **Dynamic**]

  [**Read_only**]

  For select_statement

  [**For Update** [**of** column_name [,…n]]]

Cursor – Khai báo

- ***Cursor_name***:
  - Chiều dài 128 kí tự
  - Có 2 cách khai báo
    - ✓***Tên cursor*** – Tên tỉnh mô tả cho một đối tượng cursor. Tên *cursor* sẽ được gán bằng đối tượng *cursor* thông qua câu lệnh Declare.

    VD:

    DECLARE ***cur*** CURSOR

    FOR SELECT MSSV, TenSV FROM SINHVIEN

Cursor – Khai báo

- ✓**Biến *cursor*** – *cursor* được khai báo như một biến kiểu CURSOR, khi gán giá trị cho biến *cursor* thông qua lệnh SET thì biến này sẽ trỏ tới đối tượng *cursor*.

  VD:

  ```
  DECLARE @cur  CURSOR
  SET @cur = CURSOR
  FOR SELECT MSSV, TenSV FROM SINHVIEN
  ```

  HOẶC

  ```
  DECLARE @cur  CURSOR
  SET @cur = my_cur
  ```

Cursor – Khai báo

- Ý nghĩa các tham số tùy chọn:
  - ***Insensitive / static:*** nội dung của cursor không thay đổi trong suốt thời gian tồn tại, trong trường hợp này cursor chỉ là read only.
  - ***Dynamic:*** trong thời gian tồn tại, nội dung của cursor có thể thay đổi nếu dữ liệu trong các bảng liên quan có thay đổi.

Cursor – Khai báo

- ***Local:*** cursor cục bộ, chỉ có thể sử dụng trong phạm vi một khối (query batch) hoặc một thủ tục/ hàm
- ***Global:*** cursor toàn cục (tồn tại trong suốt connection hoặc đến khi bị hủy tường minh)

Cursor – Khai báo

- ***Forward_only:*** cursor chỉ có thể duyệt một chiều từ đầu đến cuối
- ***Scroll:*** có thể duyệt lên xuống cursor tùy ý
- ***Read only:*** chỉ có thể đọc từ cursor, không thể sử dụng cursor để update dữ liệu trong các bảng liên quan (ngược lại với “for update…”)

Cursor – Khai báo

- **Mặc định**:
  - Global
  - Forward_only
  - For update
  - Dynamic

Ví dụ khai báo Cursor

- Ví dụ: : khai báo con trỏ gắn với các bản ghi của bảng SinhVien

```
declare con_tro_SV cursor
DYNAMIC SCROLL
for
     Select * from SinhVien
```

Mở Cursor

- Cú pháp:

Open Cursor_Name

- Ví dụ:

```
Open con_tro_SV;
```

Cursor – Duyệt cursor

❖ ***Dùng lệnh Fetch để duyệt tuần tự qua cursor***

Fetch

[[**Next**| **Prior**| **First**| **Last**| **Absolute** n| **Relative** n]

From] Tên_cursor

[**Into** @Tên_biến [,…n]]

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

Đọc và xử lý từng dòng lệnh FETCH

- Ví dụ: đếm số lượng sinh viên

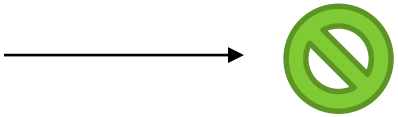
```
declare @SoSV int;
set @SoSV = 0;
FETCH FIRST from con_tro_SV
while (@@FETCH_STATUS=0)
Begin
     set @SoSV = @SoSV + 1
     FETCH NEXT from con_tro_SV
End
print N'Số sinh viên: ' + cast(@SoSV as char(4));
```

Cursor - Duyệt cursor

- **Mặc định** : *fetch next*
- Đối với cursor dạng forward_only, chỉ có thể fetch next
- Biến hệ thống @@fetch_status cho biết lệnh fetch vừa thực hiện có thành công hay không

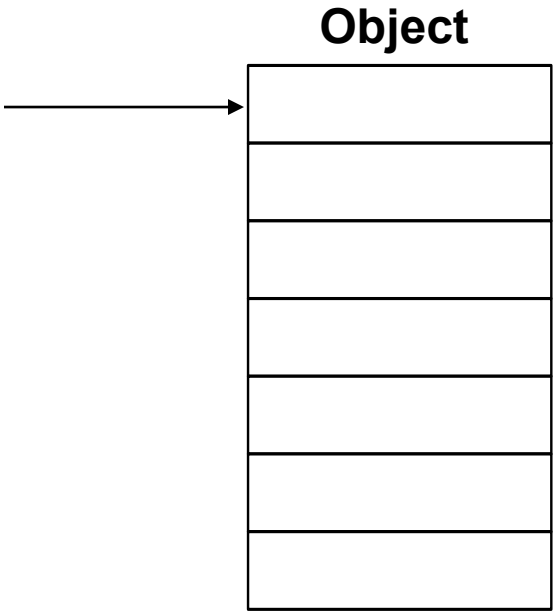
@@fetch_status

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



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

…



@@ fetch_status <> 0

Đóng Cursor

- Giải phóng dữ liệu tham chiếu bên trong Cursor

```
CLOSE Tên_con_trỏ
```

- Giải phóng Cursor ra khỏi bộ nhớ

```
DEALLOCATE Tên_con_trỏ
```

- Ví dụ:

```
Close con_tro_SV;
DeAllocate con_tro_SV;
```

Trình tự sử dụng

- Khai báo cursor
- "Mở" cursor bằng lệnh ***Open***

  Open *tên_cursor*
- Fetch (next,…) cursor để chuyển đến vị trí phù hợp
  - Dùng lệnh ***INTO*** để đưa giá trị của cursor vào biến
  - Nếu không có lệnh ***INTO***, giá trị của cursor sẽ hiển thị ra màn hình kết quả sau lệnh *fetch*
  - Có thể sử dụng vị trí hiện tại như là điều kiện cho mệnh đề where của câu delete/ update (nếu cursor không là read_only)

Trình tự sử dụng

- Lặp lại việc duyệt và sử dụng cursor, có thể sử dụng biến @@fetch_status để biết đã duyệt qua hết cursor hay chưa.
- Đóng cursor bằng lệnh ***Close***

  Close *Tên_cursor*
- Hủy cursor bằng lệnh ***deallocate***

  Deallocate *Tên_cursor*

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

Ví dụ

SINHVIEN (MaSV, HoTen, MaKhoa)

KHOA (MaKhoa, TenKhoa)

Ví dụ 1:

⇨Duyệt và đọc giá trị từ cursor

⇨Cập nhật lại giá trị

MaSV = MaKhoa + MaSV hiện tại

Áp dụng cho tất cả sinh viên

```
--1. Khai báo
Declare cur_DSKhoa Cursor
        For Select MaKhoaFrom Khoa
--2. Mở cursor
Open cur_DSKhoa
Declare @MaKhoa int
--3. Nạp cursor lần 1
Fetch Next From cur_DSKhoa into @MaKhoa
```

```
--4. Fetch lần 2…n
While @@fetch_status = 0
   Begin
        update   SinhVien
        set MaSV = MaKhoa +MaSV
        Where MaKhoa = @MaKhoa
        Fetch Next From cur_DSKhoa into @MaKhoa
    End
```

--5. Đóng cursor

Close cur_DSKhoa

--6. Hủy cursor

Deallocate cur_DSKhoa

Ví dụ

```
declare con_tro_SV cursor
DYNAMIC SCROLL
for
    Select MaSV, HoTen, GioiTinh from SinhVien

Open con_tro_SV;

declare @HoTen nvarchar(50), @MaSV int, @GioiTinh nvarchar(3);
FETCH NEXT from con_tro_SV into @MaSV, @HoTen, @GioiTinh
while (@@FETCH_STATUS=0)
Begin
    print cast(@MaSV as char(4)) + '    ' + @HoTen + '          ' + @GioiTinh
    FETCH NEXT from con_tro_SV into @MaSV, @HoTen, @GioiTinh
End

Close con_tro_SV;
DeAllocate con_tro_SV;
```

Nội dung

1. Cấu trúc lệnh
2. Thủ tục thường trú
3. Kiểu dữ liệu cursor
4. Hàm người dùng

FUNCTION (Hàm)

- **Function**: gom các câu lệnh SQL thành một nhóm và có thể sử dụng lại nhiều lần

- **Cú pháp**

```
CREATE FUNCTION function_name (parameter_list)
RETURN data_type
AS
BEGIN
    statements
    RETURN value
END
```

Hàm người dùng

- ***Giống stored procedure:***
  - Là mã lệnh có thể tái sử dụng
  - Chấp nhận các tham số input
  - Dịch một lần và từ đó có thể gọi khi cần
- ***Khác stored procedure***
  - Chấp nhận nhiều kiểu giá trị trả về (chỉ một giá trị trả về)
  - Không chấp nhận tham số output
  - Khác về cách gọi thực hiện

Ví dụ 1

- Viết hàm tính tổng hai số bất kỳ

```
CREATE FUNCTION tinh_tong(@a INT,@b INT)
RETURNS INT
AS
BEGIN
    RETURN @a + @b;
END;
```

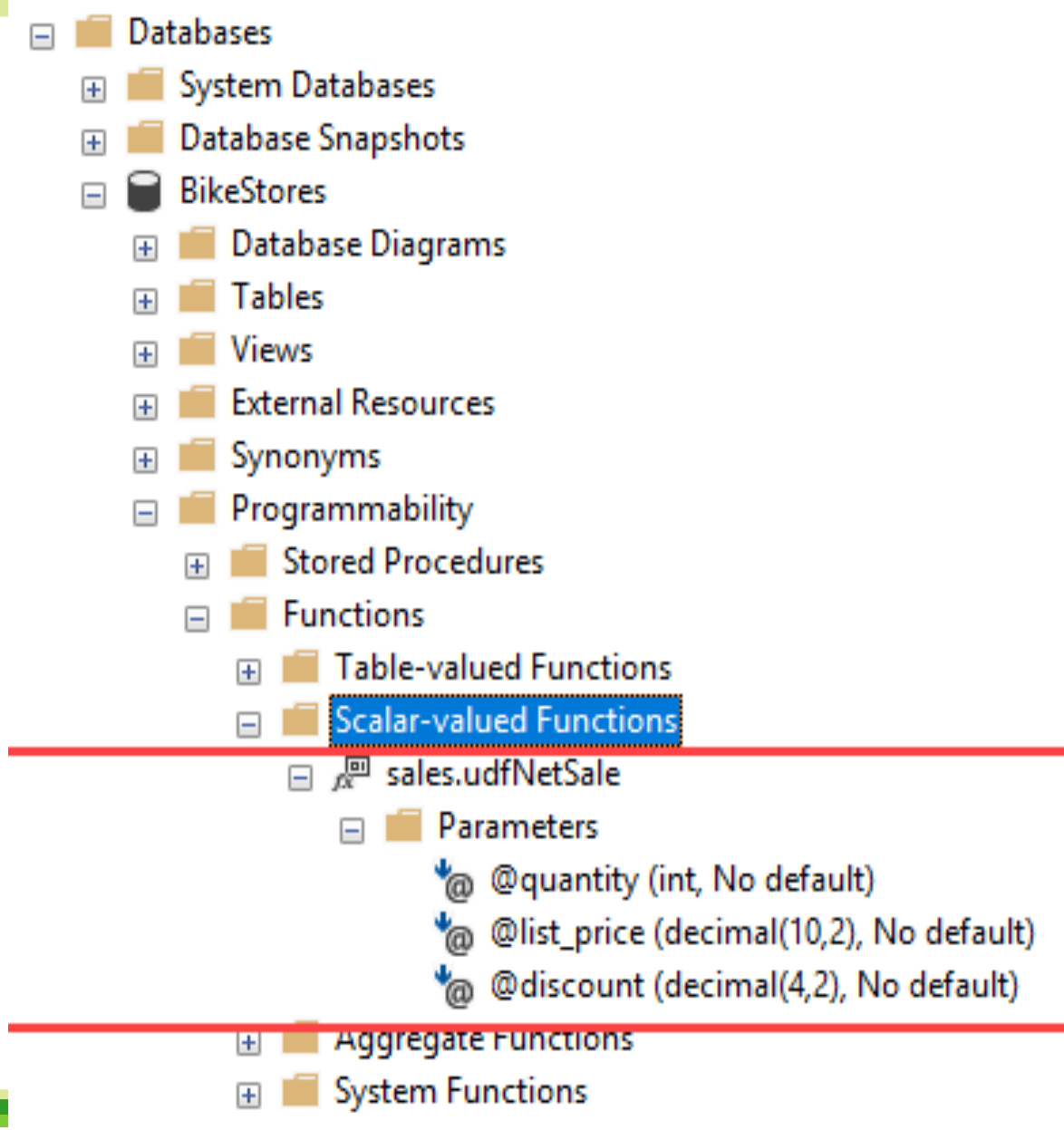
Ví dụ 2

- Viết hàm tính giá sản phẩm

```
CREATE FUNCTION GiaSP(
      @SoLuong INT,
      @DonGia DEC(10,2),
      @GiaGiam DEC(4,2)
)
RETURNS DEC(10,2)
AS
BEGIN
      RETURN @Soluong*@Dongia-@GiaGiam;
END;
```

FUNCTION (Hàm)

- Hàm sau khi được tạo sẽ được lưu trong: Programmability > Functions > Scalar-valued Functions.



Thực thi

- Gọi hàm

```
SELECT  tinh_tong(10, 20)

SELECT  GiaSP(10,2000,300)
```

Chỉnh sửa

```
ALTER FUNCTION function_name (parameter_list)
    RETURN data_type AS
    BEGIN
        statements
        RETURN value
    END
```

Xoá

```
DROP FUNCTION function_name;
```

Hàm người dùng

- ***Phân loại*** : gồm 3 loại
  - Giá trị trả về là *kiểu dữ liệu cơ sở* (int, varchar, float, datetime…) → thư mục *Scalar value function*
  - Giá trị trả về là *Table* có được từ một câu truy vấn → thư mục *Table value function*
  - Giá trị trả về là *table* mà dữ liệu có được nhờ tích lũy dần sau một chuỗi thao tác xử lý và insert. → thư mục *Table value function*

Hàm người dùng

❖**Loại 1**: **Giá trị trả về là kiểu dữ liệu cơ sở**

```
Create function func_name
    ( {parameter_name  DataType [= default ] }
    [,…n])
Returns DataType
As
Begin
        …
        Return {value | variable | expression}
End
```

Dù không có tham số cũng phải ghi cặp ngoặc rỗng

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

Ví dụ

Tìm số lớn nhất trong 3 số a, b, c

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

Hàm người dùng

❖ **Loại 2: Giá trị trả về là Table có được từ một câu truy vấn**

```
Create function func_name
  ( {parameter_name  DataType [= default ]
  [,…n])
Returns Table
As
    Return [ ( ]select_statement [ ) ]
Go
```

Thân function luôn chỉ có một lệnh, không đặt trong cặp Begin -End

Hàm người dùng

- **<u>Loại 3</u>: Giá trị trả về là table mà dữ liệu có được nhờ tích lũy dần sau một chuỗi thao tác xử lý và insert.**

```
Create function func_name
( {parameter_name  DataType [= default ] } [,…n])
Returns TempTab_name Table(Table_definition)
As
Begin
    …
    Return
End
```

Ví dụ

```
Create function uf_DanhSachLop
Returns @DS
Table(MaLop varchar(10),SoSV int)
As
    Declare cur_L cursor for Select Ma From Lop
    Declare @Ma varchar(10)
    Open cur_L
    Fetch next from cur_L into @Ma
    While @@fetch_status=0
    Begin
      ....
    End
    Close cur_L
    Deallcocate cur_L
Return
Go
```

```
Insert into @DS
Values (@Ma, (select count(*) from
SinhVien where Lop=@Ma))
Fetch next from cur_L into @Ma
```

Một số lưu ý

- Mỗi function có thể sử dụng ở bất cứ đâu trong câu lệnh T-SQL và nằm trong phạm vi database.
- Có thể có nhiều tham số, tuy nhiên chỉ trả về được một giá trị duy nhất, bắt buộc phải return.
- Có thể sử dụng mọi câu lệnh T-SQL bên trong function.
- Function này có thể sử dụng function khác

Hàm người dùng

- Ngoài các hàm do người dùng định nghĩa, SQL Server còn cung cấp các hàm xây dựng sẵn của hệ thống
- Các hàm này cung cấp tiện ích như xử lý chuỗi, xử lý thời gian, xử lý số học…
- Sinh viên tìm hiểu thêm về các hàm này trong Books on-line và các tài liệu tham khảo

Bài tập

1. Viết hàm tính điểm trung bình của sinh viên.
2. Viết hàm tìm mã sinh viên có điểm trung bình cao nhất.
3. Viết hàm xuất danh sách các sinh viên có điểm < 5.
4. Viết thủ tục xếp loại cho sinh viên (gọi hàm câu 1).

Q & A