Lập trình trên SQL Server

LẠI HIỀN PHƯƠNG

EMAIL: LHPHUONG@TLU.EDU.VN

Cấu trúc điều khiển trong T-SQL

Cấu trúc rẽ nhánh IF … ELSE

- **Cú pháp:**

```
IF biểu_thức_điều_kiện
        Lệnh | khối_lệnh khi điều kiện đúng
[ELSE
        Lệnh | khối lệnh khi điều kiện sai
        ]
```

Với khối lệnh gồm nhiều câu, cần đặt giữa cặp từ khóa BEGIN … END

```
BEGIN
        Câu lệnh 1
        …
        Câu lệnh n
END
```

Ví dụ cấu trúc IF … ELSE

- **Ví dụ**: Từ bảng SinhVien và bảng KetQua, tính điểm trung bình của ‘Nguyễn Văn A’ và hiển thị ‘Đạt’ nếu điểm trung bình lớn hơn hoặc bằng 3.5

```
Declare @Diem as real;
select @Diem = AVG(Diem) from KETQUA, SinhVien
where KETQUA.MaSV = SinhVien.MaSV
and SinhVien.HoTen = N'Nguyễn Văn A';

if (@Diem >= 3.5)
    print N'Sinh viên Nguyễn Văn A có kết quả: Đạt'
else
    print N'Sinh viên Nguyễn Văn A có kết quả: Không Đạt'
```

Ví dụ cấu trúc IF … ELSE

- **Các cấu trúc IF … ELSE có thể lồng nhau**

```
Declare @Diem as real;
select @Diem = AVG(Diem) from KETQUA, SinhVien
where KETQUA.MaSV = SinhVien.MaSV
and SinhVien.HoTen = N'Nguyễn Văn A';

print N'Điểm trung bình là: ' + cast(@Diem as char(4));
if (@Diem < 3.5)
    print N'Xếp loại: Không Đạt'
else if (@Diem < 5)
    print N'Xếp loại: D'
else if (@Diem < 6.5)
    print N'Xếp loại: C'
else if (@Diem < 8)
    print N'Xếp loại: B'
else
    print N'Xếp loại: A'
```

Cấu trúc lựa chọn CASE

- **CASE** trong SQL dùng để đánh giá một danh sách các điều kiện và trả về 1 trong các biểu thức kết quả thỏa mãn điều kiện đánh giá
- **CASE** có 2 định dạng:
  - CASE đơn giản (Simple CASE): so sánh một biểu thức với một bộ các biểu thức đơn giản để xác định kết quả
  - CASE tìm kiếm (Searched CASE): đánh giá một bộ các biểu thức Boolean để xác định kết quả

Cú pháp Simple CASE

```
CASE biểu_thức_đầu_vào
   WHEN biểu_thức_1 THEN biểu_thức_kết_quả_1
   WHEN biểu_thức_2 THEN biểu_thức_kết_quả_2
   …
   WHEN biểu_thức_n THEN biểu_thức_kết_quả_n
   ELSE biểu_thức_kết_quả_khác
END
```

Ví dụ Simple CASE

- **Ví dụ**: hiện ra màn hình tên tháng hiện tại

```
declare @thangHienTai nvarchar(20);
select @thangHienTai =
(
CASE month(getdate())
    When 1 then N'Tháng một'
    When 2 then N'Tháng hai'
    ...
    When 11 then N'Tháng mười một'
    Else N'Tháng mười hai'
END
)
print @thangHienTai
```

Cú pháp Searched CASE

```
CASE
   WHEN biểu_thức_điều_kiện_1 THEN biểu_thức_kết_quả_1
   WHEN biểu_thức_điều_kiện_2 THEN biểu_thức_kết_quả_2
   …
   WHEN biểu_thức_điều_kiện_n THEN biểu_thức_kết_quả_n
   ELSE biểu_thức_kết_quả_khác
END
```

Ví dụ: Searched CASE

- **Ví dụ**: Viết lại ví dụ xếp loại SV Nguyễn Văn A

```
declare @Diem real;
select @Diem = AVG(Diem) from KETQUA, SinhVien
where KETQUA.MaSV = SinhVien.MaSV
and SinhVien.HoTen = N'Nguyễn Văn A';
declare @XepLoai nvarchar(10);

select @XepLoai =
(CASE
    when @Diem < 3.5 then N'Không đạt'
    when @Diem < 5 then N'D'
    when @Diem < 6.5 then N'C'
    when @Diem < 8 then N'B'
    Else N'A'
End)
print @XepLoai
```

Cấu trúc lặp WHILE

- **Cú pháp:**

```
WHILE
biểu_thức_điều_kiện
    BEGIN
        Khối lệnh 1
        [BREAK]
        Khối lệnh 2
        [CONTINUE]
        Khối lệnh 3
    END
```

- **BREAK:** thoát khỏi vòng lặp WHILE, tất cả các lệnh sau từ khóa BREAK và trước từ khóa END sẽ bị bỏ qua.
- **CONTINUE**: bỏ qua các câu lệnh sau từ khóa CONTINUE và trước từ khóa END để nhảy đến vòng lặp tiếp theo của vòng lặp WHILE.

Ví dụ cấu trúc WHILE

- **Ví dụ**: Hiển thị các số từ 1 đến 9

```
declare @bienDem int;
set @bienDem = 1;
while (@bienDem < 10)
    Begin
        print N'Số hiện tại: ' + cast(@bienDem as char(2));
        set @bienDem = @bienDem + 1
    End
```

Một số toán tử đặc biệt

- **Một số toán tử đặc biệt dùng trong các biểu thức điều kiện:**

Toán tử	Ý nghĩa	Ví dụ
ALL	Tất cả	3.5 <= ALL (SELECT Diem from KETQUA)
ANY	Một vài (ít nhất 1)	3.5 > ANY (SELECT Diem from KETQUA)
SOME	Tương tự ANY	3.5 > SOME (SELECT Diem from KETQUA)
BETWEEN	Nằm giữa phạm vi	@Diem BETWEEN (3 and 5)
EXISTS	Tồn tại	EXISTS (SELECT Diem from KETQUA)
IN	Kiểm tra xem một giá trị có tồn tại trong một tập cho trước không	@GT in (N'Nam', N'Nữ')

Ví dụ các toán tử đặc biệt

- **Ví dụ:** Truy vấn hiển thị MaSV, HoTen, KetQua của tất cả các sinh viên trong bảng SinhVien với KetQua = 'Còn nợ môn' với sinh viên có môn thi chưa đạt và 'Đã qua hết' với sinh viên đã qua hết các môn

```
select MaSV, HoTen, GioiTinh, KetQua =
(
    CASE
        when exists
        (select * from KETQUA
        where KETQUA.MaSV = SinhVien.MaSV
        and KETQUA.Diem < 3.5)
        then N'Còn nợ môn'

        else N'Đã qua hết'
    END
)
from SinhVien
```

Ví dụ các toán tử đặc biệt

- **Ví dụ:** cách khác dùng ALL

```
select MaSV, HoTen, GioiTinh, KetQua =
(
    CASE
        when 3.5 <= all
        (select Diem from KETQUA
        where KETQUA.MaSV = SinhVien.MaSV)
        then N'Đã qua hết'

        else N'Còn nợ môn'
    END
)
from SinhVien
```

Ví dụ các toán tử đặc biệt

- **Ví dụ:** cách khác dùng ANY

```
select MaSV, HoTen, GioiTinh, KetQua =
(
    CASE
        when 3.5 > any
        (select Diem from KETQUA
        where KETQUA.MaSV = SinhVien.MaSV)
        then N'Còn nợ môn'

        else N'Đã qua hết'
    END
)
from SinhVien
```

Ví dụ các toán tử đặc biệt

- **Ví dụ:** cách khác dùng IN

```
select MaSV, HoTen, GioiTinh, KetQua =
(
    CASE
        when MaSV in
        (select MaSV from KETQUA
        where KETQUA.MaSV = SinhVien.MaSV and KETQUA.Diem<3.5)
        then N'Còn nợ môn'

        else N'Đã qua hết'
    END
)
from SinhVien
```

Sử dụng biến kiểu dữ liệu Cursor

Khái niệm về cursor

- Các lệnh của SQL Server làm việc trên một nhóm nhiều bản ghi
- Cursor là cấu trúc giúp làm việc với từng bản ghi tại một thời điểm
  - Khai báo cursor như một câu lệnh SELECT
  - Có thể di chuyển giữa các bản ghi trong cursor để làm việc
  - Có thể dùng cursor để cập nhật dữ liệu

Các bước sử dụng kiểu dữ liệu cursor

- Định nghĩa biến kiểu cursor bằng lệnh DECLARE
- Sử dụng lệnh OPEN để mở ra cursor đã định nghĩa trước đó
- Đọc và xử lý trên từng dòng lệnh bên trong cursor
- Đóng cursor lại bằng lệnh CLOSE và DEALLOCATE

Định nghĩa biến kiểu Cursor

- Cú pháp:

```
DECLARE    Tên_cursor    CURSOR
[LOCAL | GLOBAL]
[FORWARD_ONLY | SCROLL]
[STATIC | DYNAMIC | KEYSET]
[READ_ONLY | SCROLL_LOCK]
FOR Câu_lệnh_SELECT
[FOR UPDATE [OF Danh_sách_cột_cập_nhật]]
```

Phạm vi

- LOCAL | GLOBAL: phạm vi hoạt động của biến
  - LOCAL: biến cục bộ
  - GLOBAL: biến toàn cục (tham chiếu đến bất kỳ thủ tục nào của kết nối tạo ra biến cursor đó)
  - Mặc định là LOCAL

Phương pháp di chuyển trong Cursor

- FORWARD_ONLY: duyệt các bản ghi từ đầu đến cuối, theo chiều đi tới
- SCROLL: cursor được phép di chuyển tới lui, qua lại các dòng bản ghi trong cursor

Kiểu cursor

- STATIC (cursor tĩnh) : khi có sự thay đổi bên dưới dữ liệu gốc thì các thay đổi đó không được cập nhật tự động trong dữ liệu của cursor
- DYNAMIC (cursor động): khi có sự thay đổi bên dưới dữ liệu gốc thì các thay đổi đó được cập nhật tự động trong dữ liệu của cursor
- KEYSET: gần giống DYNAMIC. Những thay đổi trên cột không là khóa chính trong bảng gốc sẽ tự đọng cập nhật trong dữ liệu cursor. Tuy nhiên, những mẩu tin vừa thêm mới hoặc vừa hủy bỏ sẽ không hiển thị trong dữ liệu cursor

Các tùy chọn khác

- READ_ONLY : không thể cập nhật dữ liệu
- SCROLL_LOCK: Chỉ định SQL_SERVER khóa các mẩu tin cần thay đổi giá trị hoặc bị hủy bỏ bên trong bảng gốc nhằm đảm bảo hành động cập nhật luôn thành công
- Câu lệnh SQL: chỉ định danh sách các cột sẽ được truy cập bởi cursor.
- UPDATE [OF Danh sách cột cần cập nhật]:
  - nếu được chỉ định thì chỉ những cột trong danh sách được sửa
  - nếu không được chỉ định thì tất cả các cột được sửa trừ khi con trỏ kiểu READ_ONLY

Ví dụ: định nghĩa biến 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 Tên_con_trỏ

- Ví dụ:

```
Open con_tro_SV;
```

Đọc và xử lý từng dòng lệnh FETCH

- **Cú pháp**:

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

  - Absolute n: Đọc dòng thứ n trong cursor
  - Relative n: Đọc dòng thứ n kể từ vị trí hiện hành

Kiểm tra việc đọc dữ liệu

`@@FETCH_STATUS` : biến hệ thống để kiểm tra đọc dữ liệu thành công hay thất bại

Giá trị trả về	Mô tả
0	Câu lệnh FETCH thành công
-1	Câu lệnh FETCH thất bại hoặc dòng đã vượt quá kết quả gán
-2	Dòng truy cập bị xóa

Đọc và xử lý từng dòng lệnh FETCH

- **Ví dụ**: Đếm số 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));
```

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;
```

Ví dụ: đầy đủ các bước với cursor

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

Open con_tro_SV;

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));

Close con_tro_SV;
DeAllocate con_tro_SV;
```

Ví dụ: sử dụng cursor hiển thị danh sách MaSV, HoTen, GioiTinh

```
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;
```

Xử lý lỗi

Khối lệnh TRY … CATCH

- Ý nghĩa : Thực hiện các lệnh trong khối TRY, nếu gặp lỗi sẽ chuyển qua xử lý bằng các lệnh trong khối CATCH
- Cú pháp:

```
BEGIN TRY
        { các câu lệnh }
END TRY
BEGIN CATCH
        { các câu lệnh}
END CATCH
```

Khối lệnh TRY … CATCH

- Lưu ý :
  - TRY và CATCH phải cùng lô lệnh (nằm giữa hai từ khóa GO GO)
  - Sau khối TRY phải là khối CATCH
  - Có thể lồng nhiều cấp

Khối lệnh TRY … CATCH

- Ví dụ :

```
declare @i float;
begin try
    set @i = 2/0;
end try
begin catch
    select ERROR_NUMBER() as errornumber, ERROR_MESSAGE() as errormessage;
end catch
```

- Trả về

	eromumber	erormessage
1	8134	Divide by zero error encountered.

Mộ số hàm ERROR thường dùng

ERROR_NUMBER() : Trả về mã số của lỗi

ERROR_MESSAGE() Trả về chuỗi lỗi

ERROR_SEVERITY() returns the error severity.

ERROR_STATE() returns the error state number.

ERROR_LINE() : Trả về dòng gây ra lỗi

ERROR_PROCEDURE() Trả về tên thủ tục/ trigger gây ra lỗi

Thủ tục RAISERROR

- Ý nghĩa : Trả về thông báo lỗi cho ứng dụng
- **Cú pháp**:
  - Raiserror(tbao_lỗi, mức_độ, trạng_thái [, các_tham_số])
  - Tbao_lỗi: có thể là:
    - Chuỗi thông báo lỗi bất kỳ
    - Mã thông báo lỗi do người dùng định nghĩa trước bằng sp_addmessage và được lưu trong sys.messages. Giá trị phải > 50000
  - Mức_độ: số có giá trị từ 0 đến 25 thể hiện mức độ nghiêm trọng của lỗi
  - Trạng_thái: số từ 1-127 để xác định vị trí lỗi khi sử dụng cùng một thông báo lỗi tại nhiều điểm khác nhau
  - Các_tham_số: hỗ trợ các thông báo lỗi cần tham số

Thủ tục RAISERROR (Ví dụ)

```
declare @phanTram float;
set @phanTram = 101;
declare @sErrMsg nvarchar(50);
if (@phanTram not between 0 and 100)
begin
    Set @sErrMsg = N'Tỷ lệ phần trăm phải nằm trong khoảng [0,100]'
    Raiserror(@sErrMsg,16,1)
end
```