FPT Education

FPT POLYTECHNIC

cdio™

Conceive Design Implement Operate

QUẢN TRỊ CƠ SỞ DỮ LIỆU VỚI SQL SERVER

BÀI 4: ĐIỀU KIỆN & VÒNG LẶP

THỰC HỌC – THỰC NGHIỆP

http://www.poly.edu.vn

OBJECTIVES

- Điều kiện
- Vòng lặp
- Quản lý lỗi

AGENDA

Điều kiện

- Câu lệnh If….else
- Câu lệnh Case

Vòng lặp

- Câu lệnh While
- Break và Continue

Quản lý lỗi

- Try…Catch
- RAISERROR

PHẦN 1

CÁC XỬ LÝ ĐIỀU KIỆN

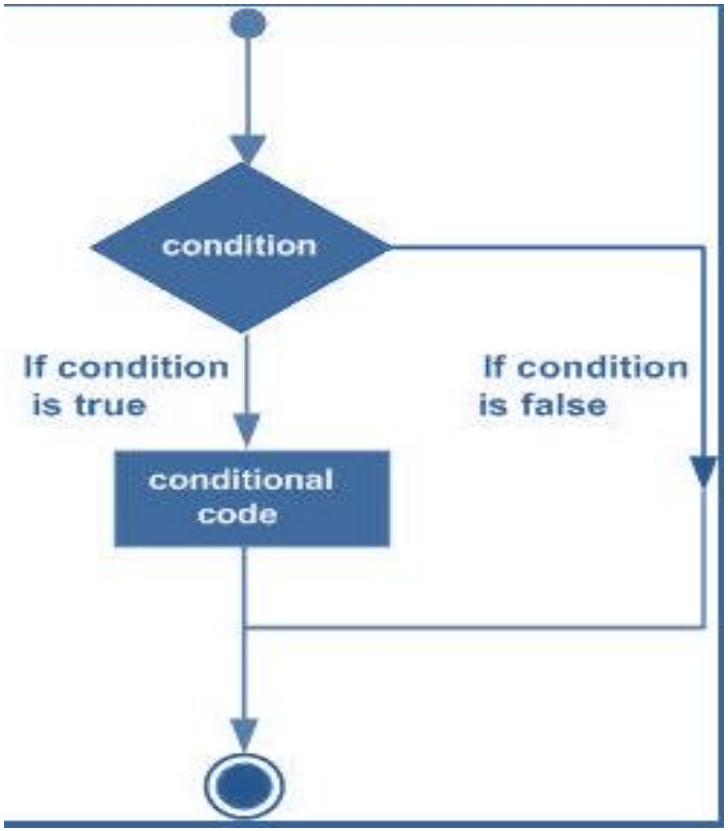
- Giả sử chúng ta viết chương trình xếp loại kết quả học tập dựa vào điểm trung bình khoá học theo tiêu chí sau:
  - Nếu điểm trung bình (sau đây gọi là dtb) nhỏ hơn 5, xếp loại "Yếu"
  - Nếu dtb lớn hơn hoặc bằng 5 và nhỏ hơn 6.5, xếp loại "Trung bình"
  - Nếu dtb lớn hơn hoặc bằng 6.5 và nhỏ hơn 8, xếp loại "Khá"
    - Nếu **dtb < 5** thì **"yếu"**
    - Ngược lại nếu **dtb < 6.5** thì **"Trung bình"**
    - Ngược lại nếu **dtb < 8** thì **"Khá"**

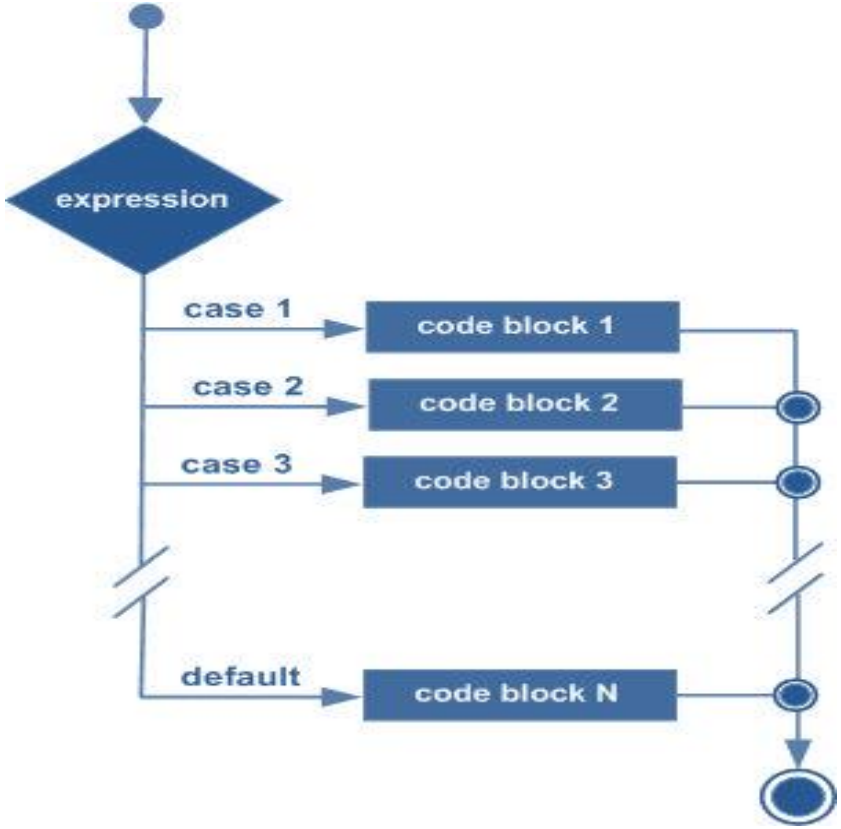
CÁC XỬ LÝ ĐIỀU KIỆN

- Viết chương trình nhập vào số nguyên, hiển thị chức năng cho phép người dùng lựa chọn:
  - Nhấn phím số 1: Thực hiện phép cộng
  - Nhấn phím số 2: Thực hiện phép trừ
  - Nhấn phím số 3: Thực hiện phép nhân
  - Nhấn phím số 4: Thực hiện phép chia
    - Nếu biến **pheptinh = 1** thì thực hiện **phép cộng**
    - Nếu biến **pheptinh = 2** thì thực hiện **phép trừ**
    - Nếu biến **pheptinh = 3** thì thực hiện **phép nhân**
    - Nếu biến **pheptinh = 4** thì thực hiện **phép chia**

CÁC XỬ LÝ ĐIỀU KIỆN

- Sử dụng đến câu lệnh if-else/case là là câu lệnh điều kiện được sử dụng khi cần đưa ra một quyết định nào đó





Câu lệnh IF - ELSE

Cú pháp

```
IF <biểu thức điều kiện>
   {<Câu lệnh>|BEGIN...END}
[ELSE
   {<Câu lệnh>|BEGIN...END}]
```

Chú ý:

Nếu thực thi hai hoặc nhiều câu lệnh trong mệnh đề IF hoặc ELSE, bạn cần bao các câu lệnh này trong khối **BEGIN...END**

❑ Câu lệnh IF - ELSE

```
IF 1 = 1
   BEGIN
        PRINT N'1 = 1 là đúng';
   END
ELSE
   BEGIN
        PRINT 'Sai';
   END;
```

```
%
Messages
1 = 1 là đúng
```

```
DECLARE @dbt float;
SET @dbt = 6.5;
IF @dbt < 5
 PRINT 'Yeu';
ELSE
 PRINT 'Trung binh';

GO
```

❑ Câu lệnh IF - ELSE

```
IF(SELECT COUNT(*) FROM NHANVIEN WHERE LUONG > 300000)>0
BEGIN
    PRINT 'Danh sach nhan vien IT co luong > 30000'
    SELECT HONV,TENNV
       FROM NHANVIEN
       WHERE LUONG>3000
END
ELSE
    PRINT ' Khong co ai lam IT ma luong >30000'
```

Results | Messages

	HONV	TENNV
1	Đinh	Như
2	Trần	Tâm
3	Nguyễn	Hùng
4	Nguễn	Tùng
5	Phạm	Vinh
6	Bùi	Hành
7	Trần	Quang
8	Đinh	Tiên

```
DECLARE @dbt float;
SET @dbt = 6.5;
IF @dbt < 5
 PRINT 'Yeu';
ELSE
    BEGIN
        IF @dbt < 6.5
            PRINT 'Trung bình';
        ELSE
            PRINT 'Kha';
    END
GO
```

Lệnh If Exists

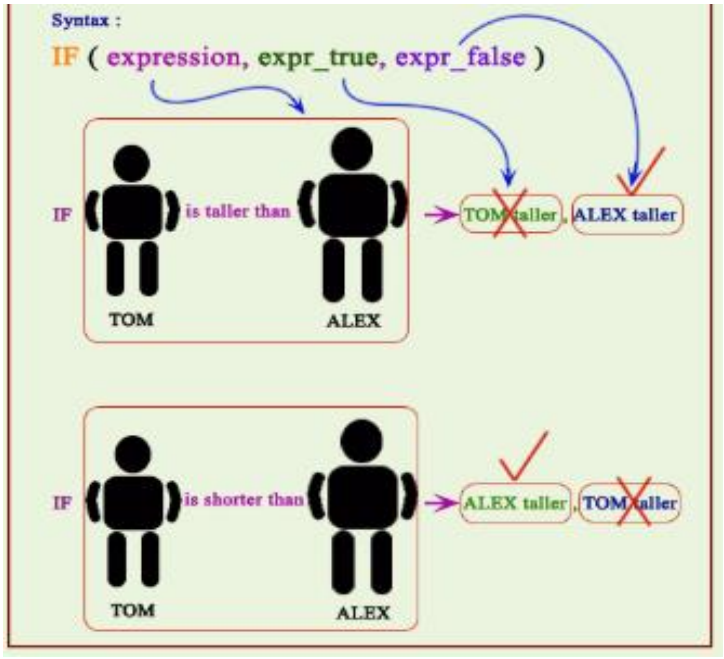
```
IF EXISTS <biểu thức điều kiện>
    {<Câu lệnh>|BEGIN...END}
[ELSE
    {<Câu lệnh>|BEGIN...END}]
```

```
IF EXISTS(SELECT * FROM NHANVIEN WHERE LUONG > 300000)
BEGIN
     PRINT 'Danh sach nhan vien IT co luong > 30000'
     SELECT HONV,TENNV
        FROM NHANVIEN
        WHERE LUONG>3000
END
ELSE
     PRINT ' Khong co ai lam IT ma luong >30000'
```

CÁC XỬ LÝ ĐIỀU KIỆN

Sử dụng IIF Function

IIF(Expression, expr_true, expr_false)




```
SELECT IIF( LUONG>30000 ,'Truong phong','NhanVien')
as ChucVu, TENNV, LUONG
FROM NHANVIEN
```

Results | Messages

	ChucVu	TENNV	LUONG
1	Truong phong	Như	43000
2	NhanVien	Tâm	25000
3	Truong phong	Hùng	38000
4	Truong phong	Tùng	40000
5	Truong phong	Vinh	55000
6	NhanVien	Hành	25000
7	NhanVien	Quang	25000
8	NhanVien	Tiên	30000

Hàm CASE trong SQL Server

- Hàm CASE kiểm định giá trị dựa trên danh sách điều kiện đưa ra, sau đó trả về một hoặc nhiều kết quả.
- CASE rất đa dạng, linh hoạt và rất hữu ích, ứng dụng trong nhiều trường hợp.
- CASE có 2 định dạng:
  - Simple CASE là so sánh một biểu thức với một bộ các biểu thức đơn giản để xác định kết quả
  - Searched CASE là đánh giá một bộ các biểu thức Boolean để xác định kết quả

CÁC XỬ LÝ ĐIỀU KIỆN

Hàm CASE trong SQL Server

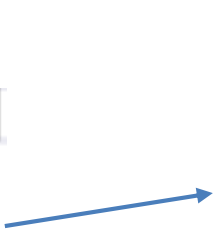
Cú pháp hàm CASE đơn giản (**Simple CASE**)

```
CASE <biểu thức>
    WHEN <điều kiện 1> THEN <biểu thức kết quả 1>
    [WHEN <điều kiện 2> THEN <biểu thức kết quả 2>]
      ...
    [ELSE <biểu thức kết quả mệnh đề else>]
END
```

Hàm CASE trong SQL Server

Ví dụ Simple CASE

```
-- Thêm tiền tố Mr hoặc Ms tùy vào phái là nam hay nữ
Select TenNV = case PHAI
 when 'nam' then 'Mr. '+[TENNV]
 when N'Nữ' then 'Ms. '+[TENNV]
 end
from NHANVIEN
```



	TenNV
1	Ms. Như
2	Mr. Tâm
3	Mr. Hùng
4	Mr. Tùng
5	Ms. Vinh
6	Mr. Hành
7	Mr. Quang
8	Mr. Tiến

```
-- Thêm tiền tố Mr hoặc Ms tùy vào phái là nam hay nữ
Select TenNV = case PHAI
 when 'nam' then 'Mr. '+[TENNV]
 when N'Nữ' then 'Ms. '+[TENNV]
 else 'FreeSex. ' + [TENNV]--Sử dụng với ELSE
 end
from NHANVIEN
```

CÁC XỬ LÝ ĐIỀU KIỆN

Hàm CASE trong SQL Server

Cú pháp hàm CASE tìm kiếm (**Searched CASE**)

```
CASE
    WHEN <biểu thức đk 1> THEN <biểu thức kết quả 1>
    WHEN <biểu thức đk 2> THEN <biểu thức kết quả 2>
      ...
    WHEN <biểu thức đk n> THEN <biểu thức kết quả n>
    ELSE <biểu thức kết quả mệnh đề else>
END
```

Hàm CASE trong SQL Server

Ví dụ Searched CASE:

```
--Tạo thêm cột thuế dựa vào mức lương
Select TENNV,LUONG,Thue = case
 When LUONG between 0 and 25000 then LUONG*0.1
 When LUONG between 25000 and 30000 then LUONG*0.12
 When LUONG between 30000 and 40000 then LUONG *0.15
 When LUONG between 40000 and 50000 then LUONG *0.2
 else LUONG*0.25 end
 from NHANVIEN
```

	TENNV	LUONG	Thue
1	Như	43000	8600
2	The	30000	3600
3	Tâm	25000	2500
4	Hùng	38000	5700
5	Tùng	40000	6000
6	Vinh	55000	13750
7	Hành	25000	2500
8	Quang	25000	2500
9	Tiên	30000	3600

So sánh Simple CASE và Searched CASE

```
--Simple CASE
Select TenNV = case PHAI
when 'nam' then 'Mr. '+[TENNV]
when N'Nữ' then 'Ms. '+[TENNV]
else 'FreeSex. ' + [TENNV]--Sử dụng với ELSE
end
from NHANVIEN
```

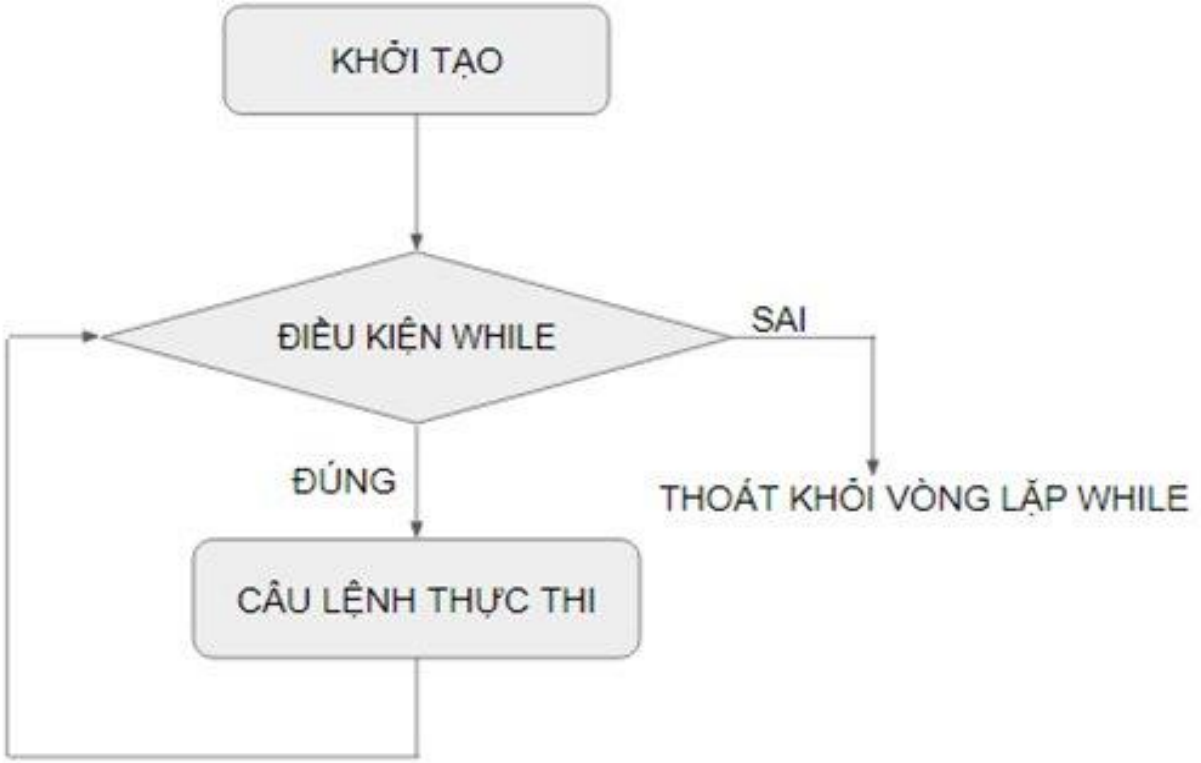
```
--Searched CASE
Select TenNV = case
when Phai like 'nam' then 'Mr. '+[TENNV]
when PHAI like N'Nữ' then 'Ms. '+[TENNV]
else 'FreeSex. ' + [TENNV]--Sử dụng với ELSE
end
from NHANVIEN
```

- Viết câu truy vấn đếm số lượng nhân viên trong từng phòng ban, nếu số lượng nhân viên nhỏ hơn 3 ➔hiển thị "Thiếu nhân viên", ngược lại <5 hiển thị "Đủ Nhan Vien", ngược lại hiển thị"Đông nhân viên"
- Viết câu truy vấn hiển thị TenNV và thêm cột thuế dựa vào mức lương: trong khoảng 0 and 25000 thì Thuế= LUONG*0.1, trong khoảng 25000 and 30000 thì LUONG*0.12, trong khoảng 30000 and 40000 thì LUONG *0.15, trong khoảng 40000 and 50000 thì LUONG *0.2, còn lại LUONG*0.25

PHẦN 2

VÒNG LẶP

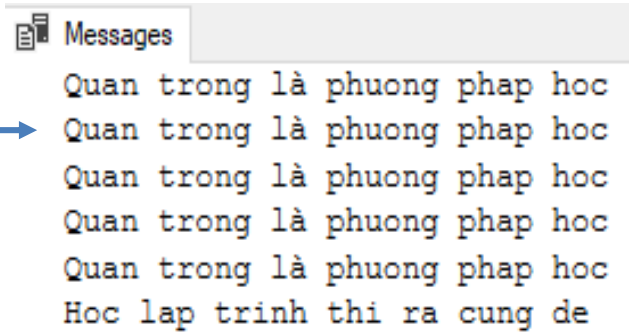
- Vòng lặp được sử dụng nếu muốn chạy lặp đi lặp lại một đoạn mã khi điều kiện cho trước trả về giá trị là TRUE



❑ Cú pháp:

```
WHILE <biểu thức điều kiện>
    {<Câu lệnh>|BEGIN...END}
[BREAK]
[CONTINUE]
```

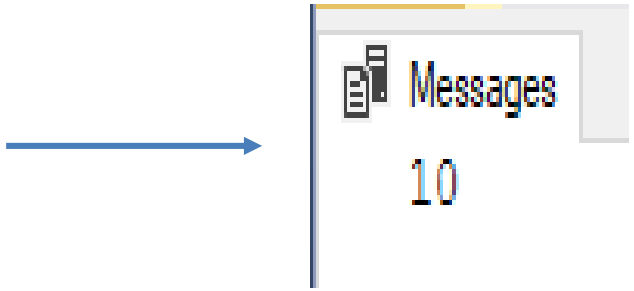
```
DECLARE @dem INT = 0;
WHILE @dem < 5
BEGIN
    PRINT 'Quan trong là phuong phap hoc';
    SET @dem = @dem + 1;
END;
PRINT 'Hoc lap trinh thi ra cung de';
GO
```



Lệnh **Break** (Ngắt điều khiển)

- Dùng để thoát khỏi vòng lặp
- Không có tham số và đối số nào nằm trong câu lệnh BREAK
- Nếu trong đoạn code có WHILE LOOP lồng nhau, BREAK sẽ chấm dứt vòng lặp WHILE gần nhất

```
DECLARE @Number INT = 1 ;
DECLARE @Total INT = 0 ;
WHILE @Number < = 10
BEGIN
IF @NUMBER = 5
BREAK;
ELSE
 SET @Total = @Total + @Number;
 SET @Number = @Number + 1 ;
END
PRINT @Total;
```



Lệnh **Continue**:

- Thực hiện bước lặp tiếp theo, bỏ qua các lệnh trong bước lặp hiện tại.
- Không có tham số và đối số nào nằm trong câu lệnh CONTINUE

```
--Không sử dụng Continue
DECLARE @Number INT = 1 ;
DECLARE @Total INT = 0 ;
WHILE @Number < = 10
BEGIN
IF @NUMBER = 5
BREAK;
ELSE
SET @Total = @Total + @Number;
SET @Number = @Number + 1 ;
SET @Number = @Number + 1 ;
END;
PRINT @Total;
GO
```

Messages

4

```
--Có sử dụng Continue
DECLARE @Number INT = 1 ;
DECLARE @Total INT = 0 ;
WHILE @Number < = 10
BEGIN
IF @NUMBER = 5
BREAK;
ELSE
SET @Total = @Total + @Number;
SET @Number = @Number + 1 ;
Continue
SET @Number = @Number + 1 ;
END;
PRINT @Total;
GO
```

Messages

10

- Viết chương trình tính tổng các số chẵn từ 1 tới 10.
- Viết chương trình tính tổng các số chẵn từ 1 tới 10 nhưng bỏ số 4.

Xử lý lỗi TRY…CATCH

- Thực hiện các lệnh trong khối TRY, nếu gặp lỗi sẽ chuyển qua xử lý bằng các lệnh trong khối CATCH

```
BEGIN TRY
    {<câu lệnh SQL>|<Khối câu lệnh>}
END TRY
BEGIN CATCH
    {<Câu lệnh SQL>|<Khối câu lệnh>}
END CATCH
```

- Các điểm cần lưu ý
  - TRY và CATCH phải cùng lô xử lý
  - Sau khối TRY phải là khối CATCH
  - Có thể lồng nhiều cấp

Xử lý lỗi **TRY…CATCH**

- Câu lệnh TRY … CATCH chỉ bắt và xử lý được các lỗi có mức nghiêm trọng từ 10-20
- Một số hàm **ERROR** thường dùng trong khối CATCH

Hàm	Mô tả
ERROR_NUMBER()	Trả về mã lỗi.
ERROR_MASSAGE()	Trả về thông điệp lỗi.
ERROR_SEVERITY()	Trả về mức nghiêm trọng của lỗi.
ERROR_STATE()	Trả về trạng thái của lỗi.
ERROR_LINE()	Trả về dòng gây ra lỗi.
ERROR_PROCEDURE()	Trả về tên thủ tục/triger gây ra lỗi.

Xử lý lỗi **TRY…CATCH**

```
-- Bắt và xử lý lỗi chèn dữ liệu vào bảng PhongBan
BEGIN TRY
    INSERT PHONGBAN
    VALUES (799, 'ZXK-799', '2008-07-01', '0197-05-22')
     -- Nếu lệnh chèn thực thi thành công in ra dòng bên dưới
    PRINT 'SUCCESS: Record was inserted.'
END TRY
                -- Nếu có lỗi xảy ra khi chèn dữ liệu in ra dòng thông báo lỗi
BEGIN CATCH
    PRINT 'FAILURE: Record was not inserted.'
    PRINT 'Error ' + CONVERT(varchar, ERROR_NUMBER(), 1)
        + ': ' + ERROR_MESSAGE()
END CATCH
```

Messages

```
(0 rows affected)
FAILURE: Record was not inserted.
Error 245: Conversion failed when converting the varchar value 'ZXK-799' to data type int.
```

Thủ tục **RAISERROR**

- Trả thông báo lỗi về cho ứng dụng

Raiserror(tbao_loi, muc_do, trang_thai [, cac_tham_so])

Trong đó:

- tbao_loi :
  - mã thông báo lỗi do người dùng định nghĩa trước bằng sp_addmessage và được lưu trong sys.messages. Giá trị phải lớn hơn 50000.
  - chuỗi thông báo lỗi bất kỳ.
- muc_do:
  - Số có giá trị từ 0
  - 25 thể hiện mức độ nghiêm trọng của lỗi.
- trang_thai: Số từ 1-127 để xác định vị trí lỗi khi sử dụng cùng một tbao_loi tại nhiều điểm khác nhau
- cac_tham_so : Hỗ trợ cho các tbao_loi cần tham số

Thủ tục **RAISERROR**

```
--Khong dùng RAISERROR
BEGIN TRY
  DECLARE @result INT
--Generate divide-by-zero error
  SET @result = 55/0
END TRY
BEGIN CATCH
 DECLARE
   @ErMessage NVARCHAR(2048),
   @ErSeverity INT,
   @ErState INT
 SELECT
   @ErMessage = ERROR_MESSAGE(),
   @ErSeverity = ERROR_SEVERITY(),
   @ErState = ERROR_STATE()
END CATCH
```

Messages

```
Commands completed successfully.
```

```
--Sử dụng RAISERROR
BEGIN TRY
  DECLARE @result INT
--Generate divide-by-zero error
  SET @result = 55/0
END TRY
BEGIN CATCH
 DECLARE
   @ErMessage NVARCHAR(2048),
   @ErSeverity INT,
   @ErState INT
 SELECT
   @ErMessage = ERROR_MESSAGE(),
   @ErSeverity = ERROR_SEVERITY(),
   @ErState = ERROR_STATE()
 RAISERROR (@ErMessage,
            @ErSeverity,
            @ErState )
END CATCH
```

Messages

```
Msg 50000, Level 16, State 1, Line 46
Divide by zero error encountered.
```

- Demo các ví dụ trong phần Try...Cach và RAISERROR

TRANSACTION - GIAO DỊCH

- Là một nhóm thao tác cơ sở dữ liệu được kết hợp thành một đơn vị logic để:
  - Để đảm bảo tính nhất quán của dữ liệu
  - Khi viết mã hai hay nhiều truy vấn thao tác tác động tới các dữ liệu có liên kết
  - Khi cập nhật tham chiếu khóa ngoại.
  - Khi chuyển hàng từ bảng này sang bảng khác.
  - Khi bạn viết truy vấn SELECT trước một truy vấn thao tác, và giá trị được chèn vào từ truy vấn thao tác này lại dựa trên kết quả của truy vấn SELECT.
  - Khi sự thất bại của tập câu lệnh SQL nào đó sẽ vi phạm tính toàn vẹn dữ liệu.

Transaction- Giao dịch

Transaction có bốn đặc điểm tiêu chuẩn sau:

- **Bảo toàn** - đảm bảo rằng tất cả các câu lệnh trong nhóm lệnh được thực thi thành công. Nếu không, transaction bị hủy bỏ tại thời điểm thất bại và tất cả các thao tác trước đó được khôi phục về trạng thái cũ.
- **Nhất quán** - đảm bảo rằng cơ sở dữ liệu thay đổi chính xác các trạng thái khi một transaction được thực thi thành công.
- **Độc lập** - cho phép các transaction hoạt động độc lập và minh bạch với nhau.
- **Bền bỉ** - đảm bảo rằng kết quả của một transaction được commit vẫn tồn tại trong trường hợp lỗi hệ thống.

❑ Ví dụ:

Một ví dụ kinh điển về transaction là khi bạn cần thực hiện một giao dịch chuyển tiền giữa hai tài khoản ngân hàng. Giả sử bạn có hai tài khoản A và B với số tiền tương ứng là 8 tỷ và 1 tỷ; nay bạn cần chuyển bớt 2 tỷ từ tài khoản A sang tài khoản B. Sẽ có hai phép cập nhật như sau:

- ❖ trừ số tiền hiện có của tài khoản A đi 2 tỷ
- ❖ cộng thêm số tiền hiện có của tài khoản B lên 2 tỷ

❑ Ví dụ (tt):

- Nếu hai lệnh cập nhật trên diễn ra độc lập (không nằm trong một transaction), và vì một lý do nào đó lệnh thứ hai bị lỗi, tài khoản A sẽ còn 6 tỷ và tài khoản B vẫn giữ nguyên 1 tỷ. Điều này không thể chấp nhận được vì 2 tỷ bỗng dưng biến mất!
- Khi thực hiện hai lệnh trên trong một transaction, nó sẽ đảm bảo:
  - ❖ hoặc cả hai lệnh update đều được thực hiện thành công. Cả hai tài khoản được cập nhật với số tiền tương ứng.
  - ❖ hoặc trong trường hợp giao dịch bị lỗi cả hai lệnh đều không được thực hiện. Hai tài khoản giữ nguyên số tiền như trước khi thực hiện transaction.

❑ Mẫu code sử dụng transaction:

```
SET XACT_ABORT ON
BEGIN TRAN
BEGIN TRY
    -- lệnh 1
    -- lệnh 2
   -- ...
COMMIT
END TRY
BEGIN CATCH
    ROLLBACK
    DECLARE @ErrorMessage VARCHAR(2000)
    SELECT @ErrorMessage = 'Lỗi: ' + ERROR_MESSAGE()
    RAISERROR(@ErrorMessage, 16, 1)
END CATCH
```

TRANSACTION - GIAO DỊCH

❑ Ý nghĩa các lệnh dùng để xử lý transaction.

- ✓ **COMMIT** - để lưu các thay đổi.
- ✓ **ROLLBACK** - để khôi phục lại các thay đổi.
- ✓ **SAVEPOINT** - tạo ra các điểm trong transaction để ROLLBACK.
- ✓ **SET TRANSACTION** - thiết lập các thuộc tính cho transaction.

❖ Các lệnh điều khiển transaction chỉ được sử dụng với các lệnh thao tác dữ liệu **DML** như - **INSERT**, **UPDATE** và **DELETE**.

❖ Chúng **<u>không</u>** thể được sử dụng trong lệnh **CREATE TABLE** hoặc **DROP TABLE** vì các hoạt động này được tự động được commit trong cơ sở dữ liệu.

GIAO DỊCH LỒNG

- Giao dịch lồng (nested transaction) là giao dịch được viết bên trong một giao dịch khác.
- Mỗi khi câu lệnh BEGIN TRAN được thực thi, hàm hệ thống @@TRANCOUNT được tăng thêm 1.
- Khi thực thi câu lệnh COMMIT TRAN.
  - Nếu @@TRANCOUNT > 1, các thay đổi sẽ không được commit. Thay vào đó @@TRANCOUNT giảm đi 1.
  - Nếu @@TRANCOUNT = 1, mọi thay đổi đã được thực hiện trên CSDL trong suốt giao dịch sẽ được commit và @@TRANCOUNT được gán bằng 0.
- Câu lệnh ROLLBACK TRAN roll-back toàn bộ các giao dịch đang hoạt động và thiết lập giá trị cho @@TRANCOUNT về 0.

ĐIỂM LƯU TRỮ (SAVE POINT)

- Câu lệnh **ROLLBACK TRAN** có thể quay lui giao dịch tới điểm bắt đầu giao dịch hoặc đến điểm lưu trữ xác định.
- Để câu lệnh **ROLLBACK TRAN** quay lui giao dịch đến điểm lưu trữ xác định, thực hiện như sau
  - Tạo điểm lưu trữ sử dụng câu lệnh **SAVE TRAN**
  - Viết câu lệnh **ROLLBACK TRAN** kèm theo tên điểm lưu trữ
- Nếu câu lệnh **ROLLBACK TRAN** không đi kèm tên điểm lưu trữ. Câu lệnh này sẽ quay lui toàn bộ giao dịch.

SUMARRY

☑ Điều kiện

- ❖ Câu lệnh If….else
- ❖ Câu lệnh Case

☑ Vòng lặp

- ❖ Câu lệnh While
- ❖ Break và Continue

☑ Quản lý lỗi

- ❖ Try…Catch
- ❖ RAISERROR


thank
you!