Lập Trình SQL

Mục tiêu

- Cách tạo và sử dụng biến cục bộ
- Tạo và kiểm tra script và batch
- Các lệnh rẽ nhánh
- Tạo, thử nghiệm và điều hành các thủ tục stored procedures.

Batch và script

- *Batch* là một tập hợp các lệnh tạo thành một nhóm và được gửi đi dưới dạng 1 nhóm lệnh thực thi.
- SQL Server biên dịch các lệnh của 1 batch thành 1 bảng thực thi (execution plan). Các lệnh trong bảng thực thi được thực thi cùng lúc.
- Lỗi biên dịch (compile error) ngăn không cho biên dịch bảng thực thi, vì vậy không có lệnh nào trong batch được thực thi.
- Lỗi thực thi (run-time error) gây ra 1 trong 2 ảnh hưởng sau:
  - Hầu hết các lỗi thực thi đều dừng lệnh đang chạy và các lệnh tiếp sau nó trong batch.
  - Một vài lỗi thực thi như vi phạm ràng buộc, chỉ làm dừng lệnh hiện hành. Tất cả các lệnh còn lại trong batch vẫn được thực thi.

Batch và script

- Giả sử có 10 lệnh trong 1 batch.
  - Nếu lệnh thứ 5 có lỗi cú pháp, không 1 lệnh nào trong batch được thực thi.
  - Nếu batch được biên dịch, và lệnh thứ hai bị lỗi trong lúc thực thi, kết quả của lệnh đầu tiên không bị ảnh hưởng vì đã thực thi rồi.

Batch và script

- *Script* là 1 chuỗi các batch được kết hợp và lưu trữ trong cùng 1 file.
- Lệnh *GO* được dùng để xác định việc kết thúc của 1 batch bên trong 1 script.
- Các quy luật dành cho batch:
  - Để chạy các thủ tục (stored procedure) sau lệnh đầu tiên trong batch thì phải chạy tường minh lệnh EXECUTE
  - Các biến cục bộ (local variable) chỉ có tác dụng trong phạm vi của batch tạo ra nó → không thể tham chiếu các biến này sau lệnh.

Ví dụ về script

- Script sau có 3 batch

```
USE pubs
GO /* đánh dấu kết thúc batch thứ 1*/
CREATE VIEW auth_titles AS
   SELECT * FROM authors
GO /* đánh dấu kết thúc batch thứ 2 */
SELECT * FROM auth_titles
GO /* đánh dấu kết thúc batch thứ 3 */
```

Ví dụ

```
USE pubs
GO
DECLARE @MyMsg VARCHAR(50)
SELECT @MyMsg = 'Hello, World.'
GO -- Biến @MyMsg không còn hợp lệ sau lệnh GO
   kết thúc batch thứ hai.

PRINT @MyMsg – Báo lỗi vì @MyMsg không được
   khai báo trong batch thứ 3
GO
SELECT @@VERSION;
 sp_who  -- Báo lỗi phải chạy lệnh EXEC sp_who
GO
```

Biến - Variable

- Trong Transact SQL, có 2 loại biến:
  - Biến cục bộ (Local variable hay user-defined variables) : là các biến được khai báo bên trong 1 batch và sẽ bị xoá khi batch kết thúc
  - Biến toàn cục (Global variables) là các biến được khai báo và gán giá trị bởi chính server. Biến toàn cục luôn bắt đầu bằng @@.

Global variables

Variable name	*Returns*
@@version	Date, version and other information on the current version
@@servername	Name of SQL server
@@spid	Server process Id number of the current process
@@procid	Stored process ID of the currently-executing procedure
@@error	0 if the last transaction succeeded, otherwise the last error number
@@rowcount	Number of rows affected by the last query, 0 if no rows are affected
@@connections	Sum of the number of connections and the attempted connections since Microsoft SQL Server was started
@@trancount	Number of currently-active transactions for a user
@@max_connections	Maximum number of simultaneous connections
@@total_errors	Total number of errors that have occurred during the current SQL server session

Biến người dùng
User defined Variables

- Biến được dùng để lưu trữ giá trị tạm thời (temporary value) khi chạy chương trình
- Cách khai báo biến:

  DECLARE @variable_name data_type

- Để gán giá trị vào biến: dùng lệnh SET hoặc SELECT
- Ví dụ:

```
DECLARE @Charge int
SET @Charge = 100
Or
SELECT @Charge =10
```

Biến người dùng
User defined Variables

- Lệnh PRINT được dùng để hiển thị thông báo của người dùng hoặc nội dung của biến ra màn hình
- Ví dụ

```
USE Northwind
GO
DECLARE @MyObject NVARCHAR(128)
SET @MyObject = 'Products'
PRINT 'Object Name: ' + @MyObject
PRINT ' Object ID: ' + STR(123456)
```

Lập trình trong SQL Server

- Các lệnh điều khiển (Control-of-Flow) dùng để điều khiển thứ tự thực hiện các lệnh trong batch, stored procedure, trigger và transaction
  - Lệnh IF…ELSE
  - Lệnh CASE
  - Lệnh WHILE

Lệnh IF…ELSE

- ***Lệnh này có thể được dùng để thực thi có điều kiện các lệnh của SQL.***
- ***Cú pháp***

  IF boolean_expression

  {sql_statement | statement_block}

  [ELSE boolean_expression

  {sql_statement | statement_block}]

Khối lệnh BEGIN…END

- ***Nếu cần nhóm các lệnh lại thành 1 khối → đặt nhóm lệnh đó bên trong khối lệnh BEGIN và END***
- ***Cú pháp***

BEGIN

{sql_statement | statement_ block}

END

Lệnh CASE

- Nếu có nhiều điều kiện cần được đánh giá
  - Dùng các lệnh IF lồng nhau
  - Dùng lệnh CASE
- Lệnh CASE có 2 dạng như sau:
  - Dạng đơn giản: lệnh sẽ so sánh 1 biểu thức ***input_expression*** với 1 bộ các biểu thức đơn giản để chọn ra nhánh cần thực hiện.
  - Dạng dò tìm: lệnh sẽ đánh giá 1 bộ các biểu thức điều kiện để chọn nhánh cần thực hiện.

CASE Statement

- **Dạng đơn giản:**

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

- **Searched CASE function:**

CASE
WHEN ***Boolean_expression*** **THEN** ***result_expression***
[...*n*]
[
ELSE ***else_result_expression***
]
END

Ví dụ lệnh case dạng đơn giản

```
USE pubs
GO
SELECT Category = CASE type
WHEN 'popular_comp' THEN 'Popular Computing'
WHEN 'mod_cook' THEN 'Modern Cooking'
WHEN 'business' THEN 'Business'
WHEN 'psychology' THEN 'Psychology'
WHEN 'trad_cook' THEN 'Traditional Cooking'
ELSE 'Not yet categorized'
END,
CAST(title AS varchar(25)) AS 'Shortened Title',
price AS Price
FROM titles
WHERE price IS NOT NULL
ORDER BY type, price COMPUTE AVG(price) BY type
GO
```

Ví dụ lệnh case dạng dò tìm

```
USE pubs
GO
SELECT 'Price Category' = CASE
WHEN price IS NULL THEN 'Not yet priced'
WHEN price < 10 THEN 'Very Reasonable Title'
WHEN price >= 10 and price < 20 THEN 'Coffee Table
   Title'
ELSE 'Expensive book!'
END,
CAST(title AS varchar(20)) AS 'Shortened Title'
FROM titles ORDER BY price
GO
```

Lệnh WHILE

- Lệnh WHILE có thể được dùng trong batch, stored procedure, trigger, hay cursor để cho phép 1 khối lệnh T-SQL được thực hiện lặp lại chừng nào mà điều kiện vẫn còn TRUE
- Cú pháp

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

BREAK và CONTINUE

- BREAK : thoát khỏi vòng lặp và bắt đầu thực hiện lệnh kế tiếp sau vòng lặp.
- CONTINUE quay về lại đầu vòng lặp, bỏ qua các lệnh nằm sau continue.

Ví dụ

```
DECLARE @count int
SELECT @count=11
WHILE @count > 0
   BEGIN
        SELECT @count=@count-1
        IF @count=4
        BEGIN
                BREAK
        END
        IF @count=6
        BEGIN
                CONTINUE
        END
   PRINT @count
   END
```


???

Các lệnh điều khiển thứ tự thực hiện

- Lệnh RETURN được dùng để thoát khỏi 1 query, batch, hay stored procedure một cách không điều kiện

Lệnh Execution

- EXEC[UTE] được dùng để thực thi các batch, stored procedure
- Cú pháp để thực thi 1 stored procedure:

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

Lệnh Execution

- Ví dụ

```
DECLARE @tab_name varchar(40)
SET @tab_name=‘inventory’
EXEC sp_help @tab_name
```

Stored procedures

- Các thủ tục hệ thống (system stored procedure) luôn bắt đầu với ký hiệu sp_ và được lưu trữ trong database master, và có thể được thực thi trong bất kỳ database nào.
- Các thủ tục của người dùng (User-defined stored procedures) được tạo ra bởi người dùng.

Stored procedures

- Một thủ tục tạm thời cục bộ (local temporary stored procedure) có tên bắt đầu với ký hiệu # và chỉ có tác dụng trong phiên làm việc của người dùng
- Một thủ tục tạm thời toàn cục (global temporary stored procedure) có tên bắt đầu với biểu tượng ## và thủ tục có tác dụng cho tất cả phiên làm việc của ngươì dùng

Stored procedures

- Thủ tục chứa các lệnh T-SQL để thực thi 1 nhiệm vụ (task). Thủ tục được thiết kế, mã hóa, kiểm tra và biên dịch thành mã thực thi (single execution plan) đặt ngay server. Các ứng dụng (application) khi cần thực thi nhiệm vụ, chỉ cần gọi thủ tục. Server sẽ chạy execution plan và trả kết quả về lại client.
  - ➔Thực thi nhanh hơn
  - ➔Giảm lưu lượng mạng (network traffic)

Lợi ích khi dùng thủ tục

- Dùng thủ tục (stored procedure) trong SQL Server tốt hơn dùng các chương trình T-SQL được lưu trữ cục bộ trên client:
  - Cho phép lập trình thủ tục: có thể tạo thủ tục 1 lần, lưu trữ nó trong database, và gọi nó bất kỳ lúc nào cần dùng trong chương trình.
  - Cho phép thực thi nhanh hơn: thủ tục được kiểm tra lỗi và tối ưu khi chúng được tạo và được biên dịch thành 1 execution plan ở lần chạy đầu tiên. Các lần sau đó khi client có yêu cầu chạy thủ tục thì server sẽ chạy từ execution plan đã biên dịch sẵn.
  - Có thể giảm lưu lượng mạng: để thực thi 1 tác vụ, thay vì phải gửi có lúc lên đến hàng trăm lệnh về server, chỉ cần dùng 1 lệnh gọi thủ tục là đủ
  - Có thể được dùng như cơ chế bảo mật: người dùng được cấp quyền để chạy thủ tục thậm chí ngay cả khi họ không có quyền thực thi từng lệnh đơn lẻ trong thủ tục.

Stored procedure

- Khi 1 thủ tục được tạo ra, tên thủ tục được viết vào bảng hệ thống sysobjects, nội dung của thủ tục thì được lưu trong bảng hệ thống syscomments.
- Thủ tục khi chạy lần đầu hay được biên dịch lại sẽ theo 3 bước sau:
  - Resolution
  - Optimization
  - Compilation

Biên dịch lại - Recompilation

- Khi database thay đổi, các execution plan gốc cũng cần tối ưu hoá lại bằng cách biên dịch lại.
- SQL Server cung cấp 3 cách để biên dịch lại (recompile) thủ tục:
  - Dùng thủ tục hệ thống **sp_recompile** để biên dịch lại thủ tục.

sp_recompile 'object'

Object có thể là stored procedure, trigger, table

  - Dùng lệnh tạo thủ tục (CREATE PROCEDURE) với tùy chọn WITH RECOMPILE để quy định SQL Server không cần lưu vào bộ nhớ execution plan của thủ tục này, mà thủ tục cần được biên dịch lại mỗi lần thủ tục được gọi.
  - Dùng lệnh EXECUTE với tuỳ chọn WITH RECOMPILE để gọi 1 thủ tục

Một số hướng dẫn khi tạo và sử dụng thủ tục

- Mỗi thủ tục chỉ nên hoàn thành 1 nhiệm vụ.
- Lệnh tạo thủ tục (CREATE PROCEDURE) không thể chứa các lệnh CREATE VIEW, CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, hay CREATE TRIGGER.
- Các lệnh CREATE TABLE và CREATE INDEX có thể được tạo và tham chiếu ngay trong cùng 1 thủ tục.

Một số hướng dẫn khi tạo và sử dụng thủ tục

- Một thủ tục không thể tạo/xóa 1 đối tượng rồi lại tham chiếu đến đối tượng đó.
- Các thủ tục có thể tham chiếu đến các bảng tạm thời.
- Các bảng tạm (Temporary table) có thể được tạo ra bên trong thủ tục và được tự động xoá khi thủ tục kết thúc.
- Có thể tham chiếu đến các đối tượng từ các CSDL khác và server từ xa.
- Cho phép các thủ tục đệ quy (recursive) – thủ tục có thể gọi chính nó..

Lệnh tạo stored procedures

- Cú pháp:

```
CREATE PROC[EDURE]
  [owner.]procedure_name
[@parameter data_type [=default]
  [OUTPUT][ ,...n ]
[WITH RECOMPILE]
AS
  sql_statements
```

Ví dụ thủ tục không có tham số

```
CREATE PROC shownullcust
AS
SELECT custName, city, region
FROM customers
WHERE country IS NULL
```

Tham số - Parameters

- ***Tham số là nơi mà thủ tục dùng để nhận giá trị của người dùng mỗi khi thủ tục được thực thi***
- Hai loại thủ tục:
  - *Thủ tục vào (Input parameter)*
  - *Thủ tục ra (Output parameter)*
- Có thể có tới 2100 tham số được dùng trong thủ tục.
- Thủ tục vào (Input parameter) để đưa giá trị vào thủ tục.
- Thủ tục ra (Output parameter) dùng để đưa giá trị ra khỏi thủ tục.
- Mỗi thủ tục trả về 1 mã số nguyên (integer return code) cho người gọi. Mặc định mã trả về là 0

Thủ tục vào - Input parameters

- Cú pháp

  @parameter_name data_type [=default]

- ***Có thể dùng tham số mặc định để đưa giá trị vào thủ tục trong trường hợp không có giá trị nào được đưa vào khi gọi thủ tục.***
- ***Giá trị mặc định phải là hằng hay NULL***
- Ví dụ:

```
CREATE PROC prcMultiplyNumber(
                @Number1 int, @Number2 int=5)
AS
PRINT @Number1*@Number2
```

Gọi thủ tục với tham số đầu vào

- Để gọi thủ tục

a. prcMultiplyNumber 4,30

→ The output would be 120

b. prcMultiplyNumber 2

→The output would be 10 (2*5).

c. prcMultiplyNumber

→ An error message specifies that you have not passed the value to @Number1.

Thủ tục hệ thống

- Xem thông tin thông qua các thủ tục hệ thống:
  - sp_help
  - sp_helptext
  - sp_depends
  - sp_stored_procedures

Sửa đổi thủ tục

- Cú pháp
  - ALTER PROC [EDURE] *procedure_name*
    [{ *@parameter data_type* }
    [= *default*] [OUTPUT]
    ] [,...*n*]
    [WITH RECOMPILE]
    AS
    sql_statement [...*n*]

Cú pháp tương tự như lệnh CREATE PROCEDURE

Lệnh xoá thủ tục

- Được dùng để xoá thủ tục khỏi database
- Cú pháp

  DROP PROCEDURE proc_name
- Không thể khôi phục 1 thủ tục ngay khi nó bị xoá

Tóm tắt các câu lệnh

-Tạo Procedure:

CREATE PROCEDURE tên thủ tục

(@tên tham số kiểu dữ liệu, @tên tham số kiểu dữ liệu, …)

AS

BEGIN

….

END

-Tham số đầu ra:

@Tên tham số kiểu dữ liệu OUTPUT

Tóm tắt các câu lệnh

-Khai báo biến:

DECLARE @tên biến kiểu dữ liệu

-Đặt giá trị cho biến:

SET @tên biến = giá trị hoặc SELECT @tên biến = giá trị

-In ra màn hình:

PRINT ‘Ket qua …’

PRINT STR(3.456)

Tóm tắt các câu lệnh

IF (điều kiện) khối lệnh
ELSE khối lệnh

CASE biểu thức
    WHEN giá trị 1 THEN khối lệnh
    WHEN giá trị 2 THEN khối lệnh
    ELSE khối lệnh
END

WHILE (điều kiện)
    Khối lệnh

Tóm tắt các câu lệnh

-Thực thi thủ tục:

EXECUTE tên thủ tục

EXECUTE tên thủ tục tham số 1, tham số 2,…

- Ví dụ:

EXECUTE InTenMonHoc ‘THT01’

Ví dụ: Viết Stored Procedure in ra tên môn học với mã môn học là tham số đầu vào

```
CREATE PROCEDURE InTenMonHoc  (@maMon varchar(10))
AS
BEGIN
        --Khai báo biến @tenMon
        DECLARE @tenMon nvarchar (100)
        --Tìm tên môn học ứng với @maMon, sau đó gán giá trị vừa tìm được cho @tenMon
        SELECT @tenMon  =  tenMonHoc
        FROM MonHoc
        WHERE  Ma = @maMon
        --In tên môn học ra màn hình
        PRINT N‘Tên môn học là ’+ @tenMon
END
```

Bài tập

- Viết các stored procedure sau:

1. In danh sách các sinh viên của 1 lớp học (mã lớp là tham số đầu vào)
2. Nhập vào 2 sinh viên, 1 môn học, tìm xem sinh viên nào có điểm thi môn học đó lần đầu tiên là cao hơn.
3. Nhập vào 1 môn học và 1 mã sv, kiểm tra xem sinh viên có đậu môn này trong lần thi đầu tiên không, nếu đậu thì xuất ra là "Đậu", không thì xuất ra "Không đậu"
4. Nhập vào 1 khoa, in danh sách các sinh viên (mã sinh viên, họ tên, ngày sinh) thuộc khoa này.

Bài tập

5. Nhập vào 1 sinh viên và 1 môn học, in điểm thi của sinh viên này của các lần thi môn học đó.
6. Nhập vào 1 sinh viên, in ra các môn học mà sinh viên này phải học.
7. Nhập vào 1 môn học, in danh sách các sinh viên đậu môn này trong lần thi đầu tiên.
8. In điểm các môn học của sinh viên có mã số là maSinhVien được nhập vào.
   (Chú ý: điểm của môn học là điểm thi của lần thi sau cùng) Chỉ in các môn đã có điểm