


Hệ quản trị Cơ sở dữ liệu

Chương 2: T-SQL nâng cao

GV: ThS. Đỗ Thị Minh Phụng


1



Nội dung

- ☐ Một số thành phần trong SQL
- ☐ Stored Procedure
- ☐ Hàm do người dùng tự định nghĩa
- ☐ Trigger
- ☐ Một vài kinh nghiệm với SQL


2



Một số thành phần trong SQL

- ☐ Chú thích
- ☐ Định danh
- ☐ Biến
- ☐ Cấu trúc điều khiển

3



is

Một số thành phần trong SQL

Chú thích

❑ Có 2 cách để người dùng ghi lại những thông tin cần thiết

– Chú thích đoạn

/*

Nhiều dòng chú thích

được đặt ở đây


*/

– Sử dụng hai dấu gạch nối '-'

SELECT ma_docgia, ho, ten

FROM DocGia --Đây là dòng chú thích

4



is

Một số thành phần trong SQL

Định danh

❑ Để định danh các thành phần trong cơ sở dữ liệu người ta đặt tên cho chúng

– Định danh của một đối tượng được tạo ra khi chúng ta định nghĩa đối tượng đó.

– Định danh này sẽ được sử dụng để tham chiếu đến đối tượng đó.

❑ Có 2 loại định danh

– Định danh thường


• Ví dụ: DocGia, NguoiLon, DangKy, CuonSach...

– Định danh có dấu phân cách: định danh được đặt trong cặp dấu nháy đơn (") hoặc cặp ngoặc vuông ([])

• [TreEm]

• [DauSach]

5



is

Một số thành phần trong SQL

Biến

❑ Định nghĩa biến

– Phải có từ khóa DECLARE và bắt đầu với dấu @

DECLARE @tien_datcoc money

DECLARE @isbn int, @ma_tuasach int

❑ Gán giá trị vào biến sử dụng từ khóa SET

SET @isbn = 123456, @ma_tuasach = 10991

SET @tien_datcoc = \$50000


❑ Gán giá trị vào biến sử dụng từ khóa SELECT

SELECT @ma_doc_gia = ma_docgia

FROM Muon

WHERE isbn = 123456

6



Một số thành phần trong SQL


Ví dụ

☐ Ví dụ

--Định nghĩa một biến

DECLARE @isbn int
 /* Truy vấn tìm ra ISBN của tựa sách có mã là 453 và gán vào biến @isbn */
 SELECT @isbn = isbn
 FROM DauSach
 WHERE ma_tuasach = 453

7



Một số thành phần trong SQL

Cấu trúc điều khiển

☐ Cấu trúc điều khiển trong T-SQL gồm những từ khóa sau:

– BEGIN...END

– IF...ELSE

– CASE ... WHEN

– TRY...CATCH


– WHILE

– BREAK / CONTINUE

– GOTO

– RETURN

8



Một số thành phần trong SQL

Cấu trúc điều khiển: BEGIN...END

☐ Định nghĩa một khối lệnh

☐ Định nghĩa tương đương trong các ngôn ngữ khác:

– C#, Java, C: { ... }

– Pascal, Delphi: BEGIN ... END

9

3



Một số thành phần trong SQL

Cấu trúc điều khiển: IF...ELSE

- ☐ Định nghĩa các thao tác khi thỏa một điều kiện và có thể có những thao tác sẽ thực hiện khi điều kiện đó không thỏa
- ☐ Cú pháp:
IF Boolean_expression
SQL_statement | block_of_statements
[ELSE
SQL_statement | block_of_statements]

10



Một số thành phần trong SQL

Cấu trúc điều khiển: IF...ELSE – Ví dụ

☐ Ví dụ
DECLARE @isbn int
SET @isbn = 123456
IF EXISTS (SELECT * FROM DAUSACH WHERE
ISBN = @isbn)
BEGIN
PRINT 'DA TON TAI DAU SACH NAY!'
END
ELSE
PRINT 'KHONG TON TAI DAU SACH NAY!'
END

11




Một số thành phần trong SQL

Cấu trúc điều khiển: CASE...WHEN

- ☐ Đưa ra danh sách các điều kiện và trả về một trong những kết quả phù hợp
- ☐ Cú pháp:
CASE input_expression
WHEN when_expression THEN result_expression
[WHEN when_expression THEN result_expression...n]
[ELSE else_result_expression]
END

12



Một số thành phần trong SQL

Cấu trúc điều khiển: CASE...WHEN – Ví dụ


❑ Ví dụ

```

SELECT ma_docgia, ma_cuonsach,
CASE
  WHEN GETDATE() > DATEADD(day, -10, ngay_hethan) THEN 'Gan
  het han'
  WHEN GETDATE() = ngay_hethan THEN 'Toi han'
  WHEN GETDATE() > ngay_hethan THEN 'Het han'
ELSE 'Con han'
END TinhTrangMuon
FROM Muon

```

13



Một số thành phần trong SQL

Cấu trúc điều khiển: TRY...CATCH

❑ Là câu lệnh bắt lỗi trong T-SQL tương tự như trong ngôn ngữ C# / Java


❑ Cú pháp:

```

BEGIN TRY
  { sql_statement | statement_block }
END TRY
BEGIN CATCH
  [ { sql_statement | statement_block } ]
END CATCH

```

14



Một số thành phần trong SQL

Cấu trúc điều khiển: TRY...CATCH – Ví dụ


❑ Ví dụ

```

BEGIN TRY
  SELECT *
  FROM sys.messages
  WHERE message_id = 21;
END TRY
BEGIN CATCH
  SELECT ERROR_NUMBER() AS ErrorNumber;
END CATCH;
GO

```

15



Một số thành phần trong SQL

Cấu trúc điều khiển: WHILE

Đưa ra điều kiện để thực hiện một khối lệnh lặp lại nhiều lần


- Các câu lệnh trong khối sẽ được thực hiện lặp đi lặp lại khi nào điều kiện còn thỏa.
- Việc thực hiện các câu lệnh trong vòng lặp WHILE có thể được kiểm soát từ bên trong vòng lặp nhờ từ khóa BREAK và CONTINUE.

Cú pháp

WHILE Boolean_expression

{ sql_statement | statement_block | BREAK | CONTINUE }

16




Một số thành phần trong SQL

Cấu trúc điều khiển: WHILE – Ví dụ

Ví dụ

```
--Tong cac so chan tu 1 den 10
DECLARE @i int = 0
DECLARE @tong int = 0
WHILE (@i < 10)
BEGIN
    SET @i += 1
    IF (@i % 2 = 0) SET @tong += @i
END
PRINT STR(@tong)
```

17



Một số thành phần trong SQL

Cấu trúc điều khiển: GOTO

Chuyển việc thực hiện câu lệnh đến vị trí đã đánh dấu bằng nhãn. Các câu lệnh Transact-SQL sau từ khóa GOTO đều bị bỏ qua và tiếp tục tại vị trí của nhãn

Cú pháp:


Định nghĩa nhãn:

label :

Chuyển luồng thực hiện câu lệnh:

GOTO label

18



Một số thành phần trong SQL

Cấu trúc điều khiển: GOTO – Ví dụ


❑ Ví dụ

```

DECLARE @i INT;
SET @i = 0;
WHILE @i <= 10
BEGIN
    IF @i = 2
        GOTO VíDuGOTO;
    SET @i = @i + 1;
END;
VíDuGOTO:
PRINT 'Day la vi du ve GO TO';
GO

```

19



Một số thành phần trong SQL

Cấu trúc điều khiển: RETURN


❑ Thoát vô điều kiện khỏi một truy vấn hay thủ tục

❑ Từ khóa này sẽ được đề cập rõ hơn trong phần Stored Procedure.

❑ Cú pháp

```
RETURN [ integer_expression ]
```

20



Stored Procedure


Tổng quan 1/2

❑ Một stored procedure (SP) là một tập hợp các câu lệnh SQL mà SQL Server sẽ biên dịch để thực hiện cùng lúc.

❑ Bạn có thể đưa các biến đầu vào, SP sẽ trả về giá trị dưới dạng các biến hoặc thông báo về tình trạng thực hiện các câu lệnh thành công/thất bại.

21

7




Stored Procedure

Tổng quan 2/2

- Stored procedures có 4 cách trả về giá trị:
 - Biến đầu ra, có thể là giá trị (chẳng hạn như số nguyên hay ký tự...) hoặc là một biến con trỏ (con trỏ là một tập kết quả mà ta có thể duyệt giá trị lần lượt từng dòng một).
 - Trả về những mã số luôn có giá trị nguyên.
 - Một tập hợp kết quả ứng với mỗi câu lệnh **SELECT** có trong stored procedure hoặc ứng với mỗi stored procedure khác được nó gọi thực thi.
 - Một biến con trỏ toàn cục mà có thể tham chiếu bên ngoài stored procedure.

22




Stored Procedure

Lợi ích khi sử dụng SP 1/2

- Lợi ích khi sử dụng SP
 - Giảm dung lượng giữa server/client:
 - Chỉ có lệnh gọi thực hiện thủ tục được gửi đi trong mạng.
 - Bảo mật tốt hơn
 - Khi gọi thủ tục trong mạng thì chỉ có lệnh thực hiện thủ tục được hiện ra. Do đó, những kẻ xấu không thể thấy được các bảng, các thành phần trong cơ sở dữ liệu, các câu lệnh Transact-SQL bên trong hoặc tìm ra dữ liệu xung đột.

23




Stored Procedure

Lợi ích khi sử dụng SP 2/2

- Lợi ích khi sử dụng SP
 - Tái sử dụng được các câu lệnh:
 - Những đoạn lệnh được sử dụng lặp đi lặp lại nhiều lần có thể được đóng gói trong SP (ví dụ, câu lệnh CẬP NHẬT dữ liệu trong bảng)
 - Cải thiện hiệu năng:
 - SP được lưu trữ ở vùng nhớ tạm khi lần đầu thực hiện nó, do đó chúng ta có thể sử dụng lặp lại nhiều lần. SQL Server sẽ không biên dịch lại mỗi lần chạy.

24




Stored Procedure

Stored Procedure vs. Câu lệnh SQL

Câu lệnh SQL	Stored Procedure
Lần đầu <ul style="list-style-type: none"> Kiểm tra cú pháp Biên dịch Thực hiện Trả về giá trị 	Khởi tạo <ul style="list-style-type: none"> Kiểm tra cú pháp Biên dịch
Lần thứ hai <ul style="list-style-type: none"> Kiểm tra cú pháp Biên dịch Thực hiện Trả về giá trị 	Lần đầu <ul style="list-style-type: none"> Thực hiện Trả về giá trị
	Lần thứ hai <ul style="list-style-type: none"> Thực hiện Trả về giá trị

25




Stored Procedure

Cú pháp – Tạo Stored Procedure

❑ Tạo / Chính sửa SP

```
CREATE PROC[EDURE] procedure_name
[ @parameter_name data_type] [= default]
OUTPUT][,...,n]
AS
SQL_statement_block
```

26



Stored Procedure

Cú pháp – Thực thi, Cập nhật, Xóa SP

❑ Thực thi Stored Procedure:

```
EXEC[UTE] procedure_name
```

❑ Cập nhật Stored Procedure

```
ALTER PROC[EDURE] procedure_name
[ @parameter_name data_type]
[= default] [OUTPUT]
[,...,n]
AS
SQL_statement(s)
```

❑ Xóa Stored Procedure

```
DROP PROC[EDURE] procedure_name
```

27

Stored Procedure

Ví dụ

```

CREATE PROCEDURE UCLN_Euler @a int, @b int
AS
BEGIN
    IF (@a = 0 or @b = 0) PRINT '0'
    ELSE
    BEGIN
        SET @a = ABS(@a)
        SET @b = ABS(@b)
        IF (@b % @a = 0) PRINT STR(@a)
        ELSE
        BEGIN
            SET @b = @b % @a
            EXEC UCLN_Euler @a = @b, @b = @a
        END
    END
END
    
```

28

Stored Procedure

Khuyết điểm

- ☐ Làm tăng cao mức độ xử lý và chiếm dụng bộ nhớ của server
- ☐ Khó viết những thủ tục có logic phức tạp
- ☐ Khó gỡ lỗi
- ☐ Không dễ bảo trì, nâng cấp

29

Hàm tự định nghĩa (UDF)

Hàm là gì?

- ☐ UDF thường nhận các giá trị đầu vào, thực hiện các hành động và trả về giá trị là kết quả của hành động đó. Giá trị trả về có thể là một giá trị vô hướng hoặc là một tập kết quả.
 - Hàm không thể thực hiện những thay đổi lâu dài trong SQL Server như Thêm, Xóa, Sửa trên bảng thực.
- ☐ Có các loại UDF sau:
 - Hàm **Scalar**
 - Hàm **Table-valued**
 - Hàm Inline Table-valued
 - Hàm Multi-statement Table-Valued

30



Hàm tự định nghĩa (UDF)

Cú pháp hàm Scalar

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name data_type [= default ] [ READONLY ] } ]
  ,...n [ ] )
RETURNS return_data_type
[ AS ]
BEGIN
    function_body
    RETURN scalar_expression
END
```

31



Hàm tự định nghĩa (UDF)

Cú pháp hàm Inline Table-valued

```
CREATE FUNCTION [schema_name.]function_name
( [ { @parameter_name data_type [= default ] } ] ,...n [ ] )
RETURNS TABLE
[ WITH <function_option> [ ,...n ] ]
[ AS ]
RETURN ( [ select_statement ] )
```

32




Hàm tự định nghĩa (UDF)

Cú pháp hàm Multi-statement Table-valued

```
CREATE FUNCTION [ schema_name. ] function_name
( ( [ { @parameter_name data_type [= default ] [ READONLY ] } ]
  ,...n [ ] )
)
RETURNS @return_variable TABLE <table_type_definition>
[ WITH <function_option> [ ,...n ] ]
[ AS ]
BEGIN
    function_body
    RETURN
END
```

33



Hàm tự định nghĩa (UDF)

Ví dụ

❑ Ví dụ


- Hàm Scalar

```

CREATE FUNCTION demSoDocGia(@ma_cuonsach smallint)
RETURNS int
AS
BEGIN
    DECLARE @result int;
    SELECT @result = COUNT(@ma_docgia)
    FROM MUON
    WHERE ma_cuonsach = @ma_cuonsach
    RETURN @result;
END;

```

34



Hàm tự định nghĩa (UDF)

Ví dụ

❑ Ví dụ


- Hàm Inline table

```

CREATE FUNCTION getDSDangKy(@isbn int)
RETURNS TABLE
AS
RETURN(
    SELECT *
    FROM DangKy
    WHERE isbn = @isbn
);

```

35



Hàm tự định nghĩa (UDF)

Ví dụ

❑ Ví dụ

- Hàm Multi-statement table-valued

```

CREATE FUNCTION getThongTinDocGiaNguoiLon(@ma_docgia smallint)
RETURNS @ReaderInfo TABLE (
    ma_docgia smallint, hoten nvarchar(50),
    ngaysinh smalldatetime, diachi nvarchar(120)
)
AS
BEGIN
    INSERT INTO @ReaderInfo
    SELECT dg.ma_docgia, dg.ho+ ' '+dg.tenlot+ ' '+dg.ten,
    dg.ngaysinh, nl.sonha+ ' ', ' '+nl.duong+ ' ', ' '+nl.quan
    FROM DocGia dg, NguoiLon nl
    WHERE dg.ma_docgia = @ma_docgia AND dg.ma_docgia = nl.ma_docgia
END
RETURN
END

```

36

Trigger

Trigger là gì?

- ❑ Trigger là một dạng **stored procedure** đặc biệt được tự động thực hiện khi một phần dữ liệu bất kỳ bị thay đổi.
- ❑ Trigger được tạo trên một bảng và hỗ trợ một hay nhiều thao tác gây nên sự thay đổi dữ liệu (INSERT, UPDATE, hay DELETE).
- ❑ Khi một thao tác có cài đặt trigger được thực hiện thì trigger sẽ được tự động chạy.
- ❑ Sau đây là một số ví dụ về việc sử dụng trigger:
 - Duy trì tính chất sao chép và dẫn xuất dữ liệu
 - Ràng buộc phức tạp trên nhiều cột
 - Lan truyền toàn vẹn tham chiếu
 - Tính toán những giá trị mặc định phức tạp
 - Bảo đảm toàn vẹn tham chiếu liên cơ sở dữ liệu.

37

Trigger

Phân loại Trigger

- ❑ Chúng ta quan tâm đến 2 loại Trigger sau:
 - **DML triggers (Standart triggers)**
 - Tự chạy khi người dùng thay đổi dữ liệu trên bảng hay khung nhìn (INSERT, UPDATE, DELETE).
 - Có thể dùng để thực thi các quy tắc và đảm bảo toàn vẹn dữ liệu.
 - **DDL triggers** được thực hiện khi cấu trúc của khung nhìn, bảng,... bị thay đổi (CREATE, ALTER, DROP...).

38

Trigger

DML Trigger

- ❑ DML Trigger bao gồm:
 - AFTER Trigger: được gọi sau khi dữ liệu thay đổi thành công. AFTER là loại trigger mặc định và không thể sử dụng cho khung nhìn.
 - INSTEAD OF Trigger: thực hiện thay cho câu lệnh SQL gọi ra trigger. INSTEAD OF trigger sử dụng được cho cả bảng và khung nhìn.
 - Được dùng để thay thế câu lệnh SQL tương tác với dữ liệu.
 - Rất hữu ích trong việc thay đổi dữ liệu trong khung nhìn khi không thể thực hiện bằng cách thông thường.

39

Trigger

Cú pháp DML Trigger

```

CREATE TRIGGER Trigger_name
ON table | view
[WITH ENCRYPTION]
{ FOR | AFTER | INSTEAD OF }
{ [DELETE] [,] [INSERT] [,] [UPDATE] }
AS Sql_statement

ALTER TRIGGER trigger_name
ON ( table | view )
[ WITH ENCRYPTION ] { ( FOR | AFTER | INSTEAD OF )
{ [ DELETE ] [,] [ INSERT ] [,] [ UPDATE ] }
[ NOT FOR REPLICATION ] AS sql_statement [ ...n ]

DROP TRIGGER { trigger_name }

```

40

Trigger

Cú pháp vô hiệu/kích hoạt

☐
Cú pháp vô hiệu

Disable trigger <trigger_name> ON <table_name>

☐
Cú pháp kích hoạt

Enable trigger <trigger_name> ON <table_name>

41

Trigger

Sử dụng Trigger

☐
Không cần thay đổi bất cứ lệnh nào để thực hiện trigger:

- Tự động
- Thông báo
- Ghi nhận/Kiểm tra
- Ngăn chặn dữ liệu không bình thường

42

Trigger

Bảng Deleted và Inserted

- ❑ Khi tạo ra trigger, bạn có quyền truy cập đến hai bảng tạm (bảng **deleted** và **inserted**). Hai bảng này cũng được coi là bảng nhưng khác với các bảng bình thường. Chúng được lưu trữ trong bộ nhớ, không phải trong đĩa cứng.
- ❑ Khi câu lệnh thêm, xóa hay sửa được thực thi, tất cả dữ liệu được chép vào trong những bảng này với cùng cấu trúc bảng.

Insert

Update

Delete

new

new

old

old

Inserted Table

Deleted Table

- ❑ Giá trị lưu trong hai bảng deleted và inserted chỉ được truy cập bằng trigger mà thôi. Một khi trigger được thực hiện xong thì các bảng này sẽ không còn truy cập được.

43

Trigger

Ví dụ

- ❑ Ví dụ
 - DML Trigger

```

CREATE TRIGGER KiemTraTuaSachTonTai ON TUASACH
FOR INSERT
AS
BEGIN
    IF (SELECT COUNT(TS.TUASACH) FROM TUASACH TS WHERE TS.TUASACH
        IN (SELECT INSERTED.TUASACH FROM INSERTED)) > 1
    BEGIN
        PRINT N'TỰA SÁCH ĐÃ TỒN TẠI!'
        ROLLBACK
    END
END
            
```

44

Trigger


Ví dụ

- ❑ Ví dụ
 - DDL Trigger

```

CREATE TRIGGER safety
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
    PRINT 'Bạn phải vô hiệu Trigger "safety" để xóa hay sửa bảng!'
    ROLLBACK;
            
```

45




Một vài kinh nghiệm với SQL

☐ Bạn nên sử dụng giao tác cho các thao tác DELETE/ INSERT/ UPDATE.

☐ Giao tác sẽ giúp đảm bảo tính ACID của cơ sở dữ liệu.

46



Một vài kinh nghiệm với SQL


Giao tác

☐ Giao tác là một phương pháp giúp cho người dùng xây dựng các tập lệnh một cách logic và phù hợp nhất. Khi hoàn thành giao tác, cơ sở dữ liệu sẽ ở trong trạng thái nhất quán.

☐ Tính chất của giao tác (ACID):

- Nguyên tử (Atomicity): tất cả dữ liệu thay đổi trong giao tác phải được chấp nhận và được chèn vào cơ sở dữ liệu. Nếu không sẽ không có sự thay đổi nào xảy ra.
- Nhất quán (Consistency): Một khi dữ liệu được thay đổi thành công, hoặc không có thay đổi nào xảy ra, tất cả dữ liệu phải được đảm bảo trạng thái nhất quán và dữ liệu phải được đảm bảo sự toàn vẹn.
- Cô lập (Isolation): tất cả những thay đổi diễn ra trong giao tác này phải độc lập với những thay đổi ở các giao tác khác.
- Bền vững (Durability): bất cứ lỗi hệ thống nào (phần cứng hay phần mềm) cũng không thể xóa được những thay đổi đã xảy ra.

47



Một vài kinh nghiệm với SQL


Stored Procedure

☐ Câu lệnh SET NOCOUNT ON: Điều này sẽ giúp giảm lưu lượng đường truyền trong mạng.

☐ Ví dụ:


```
CREATE PROC dbo.ProcName
AS
SET NOCOUNT ON;
--Procedure code here
GO
```

48



Một vài kinh nghiệm với SQL


Stored Procedure

☐ Trình bày thẳng hàng tên biến, kiểu dữ liệu và giá trị mặc định

```

CREATE PROCEDURE dbo.User_Update
@CustomerID      INT,
@FirstName       VARCHAR(32)      = NULL,
@LastName        VARCHAR(32)      = NULL,
@Password        VARCHAR(16)      = NULL,
@EmailAddress    VARCHAR(320)     = NULL,
@Active          BIT              = 1,
@LastLogin       SMALLDATETIME    = NULL
AS
BEGIN
    
```

49




Một vài kinh nghiệm với SQL

Stored Procedure

☐ Luôn cố gắng khai báo biến lúc bắt đầu của SP
☐ Sử dụng IF EXISTS (SELECT 1 ...) thay vì IF EXISTS (SELECT * ...)
☐ Tránh việc sử dụng WHILE: vòng lặp trong SP sẽ dẫn đến vấn đề về hiệu năng.


50



Một vài kinh nghiệm với SQL

☐ Từ khóa **HAVING** có khả năng lọc nhiều dòng sau khi tất cả các dòng đều được chọn. Không được sử dụng HAVING vào mục đích khác.
Cách sử dụng:
 SELECT subject, count(subject)
 FROM student_details
 WHERE subject != 'Science' AND subject != 'Maths'
 GROUP BY subject;
Thay vì:
 SELECT subject, count(subject)
 FROM student_details
 GROUP BY subject
 HAVING subject != 'Science' AND subject != 'Maths';

51



Một vài kinh nghiệm với SQL

❑ Đôi khi bạn cần nhiều truy vấn con trong câu truy vấn chính của bạn. **Hãy cố gắng giảm bớt số lượng truy vấn con này.**

❑ Ví dụ,


Sử dụng:

```
SELECT name FROM employee
WHERE (salary, age) = (
    SELECT MAX (salary), MAX (age)
    FROM employee_details)
AND dept = 'Electronics';
```

Thay vì:

```
SELECT name FROM employee
WHERE salary=(SELECT MAX(salary) FROM employee_details)
AND age = (SELECT MAX(age) FROM employee_details)
AND emp_dept = 'Electronics';
```

52



Một vài kinh nghiệm với SQL

❑ **Sử dụng dữ liệu không phải cột ở cùng một về của điều kiện trong câu truy vấn bởi vì điều này sẽ giúp thực thi nhanh hơn.**


Sử dụng:


```
SELECT id, name, salary
FROM employee
WHERE salary < 25000;
```

Thay vì:

```
SELECT id, name, salary
FROM employee
WHERE salary + 10000 < 35000;
```

53





54
