

Chương 2: Ngôn ngữ lập trình Transaction – SQL

Thời lượng: 6 tiết

Nội dung

- 1. Khai báo và sử dụng biến trong SQL**
- 2. Cấu trúc điều khiển**
- 3. Stored Procedure**
- 4. Trigger**
- 5. Cursor**
- 6. Function**

Khai báo biến

Dùng từ khoá declare để khai báo biến

DECLARE {@local_variable} data_type} [,...n]

Gán giá trị cho biến

SET @*local_variable_name* = *expression*

Ví dụ

```
DECLARE @vLastName char(20),  
@vFirstName varchar(11)
```

```
SET @vLastName = 'Dodsworth'
```

```
SELECT @vFirstName = FirstName  
FROM Northwind.Employees  
WHERE LastName = @vLastName
```

```
PRINT @vFirstName + ' ' + @vLastName
```

Gán
giá trị
cho
biến
bằng
tù
khoá
set

hoặc
bằng
câu
lệnh
select

Ví dụ

❖ Khai báo biến

- **DECLARE và bắt đầu với ký hiệu@**

DECLARE @limit money

DECLARE @min_range int, @hi_range int

❖ Gán giá trị cho biến dùng SET

SET@min_range = 0, @hi_range = 100

SET@limit = \$10

❖ Gán giá trị cho biến dùng SELECT

SELECT @price = price

FROM titles

WHERE title_id = 'PC2091'

Data Type (1)

- Integers
 - Bigint: 8 bytes
 - Int: 4 bytes
 - Smallint: 2 bytes
 - Tinyint: 1 byte, từ 0 -> 255.
- bit
 - Bit: 1 hoặc 0 value.
- decimal and numeric
 - Decimal từ $-10^{38+1} \rightarrow 10^{38-1}$.
 - Numeric: giống decimal.
- money and smallmoney
 - Money: 8 bytes
 - Smallmoney: 4 bytes
- Approximate Numerics
 - Float: từ $-1.79E + 308 \rightarrow 1.79E + 308$.
 - Real: từ $-3.40E + 38 \rightarrow 3.40E + 38$.

Data Type (2)

- **datetime and smalldatetime**
 - **Datetime**: từ 1/1/1753-> 31/12/9999.
 - **Smalldatetime** từ 1/1/1900, -> 6/6/2079.
- **Character Strings**
 - **Char**: Fixed-length non-Unicode character, <= 8,000 ký tự
 - **Varchar**: Variable-length non-Unicode , <= 8,000 ký tự
 - **Text**: Variable-length non-Unicode <= $2^{31} - 1$
(2,147,483,647) ký tự
- **Unicode Character Strings**
 - **nchar**Fixed-length Unicode , <=4,000 characters.
 - **nvarchar**Variable-length Unicode, <=4,000 characters
 - **Ntext** Variable-length Unicode <= $2^{30} - 1$ (1,073,741,823)
characters.
- **Other Data Type**
 - **Cursor**: là một tham chiếu đến một cursor.
- Một biến không thể có kiểu là text, ntext, hoặc image

Toán tử (operators)

- Các loại toán tử
 - Số học: *, /, %, - , +
 - So sánh: =, <>, >, >=, <, <=
 - Nối chuỗi: +
 - Luận lý: AND, OR, NOT

Thứ tự ưu tiên các toán tử

Type	Operator	Symbol
Grouping	Primary grouping	()
Arithmetic	Multiplicative	* / %
Arithmetic	Additive	- +
Other	String concatenation	+
Logical	NOT	NOT
Logical	AND	AND
Logical	OR	OR

Functions (1)

- Aggregate functions: tính toán trên một nhóm và trả về một giá trị. Ví dụ:

```
SELECT AVG(UnitPrice) FROM Products
```

Products

28.8663

(1 row(s) affected)

Functions (2)

- **Scalar functions:** Tác động lên một giá trị và trả về một giá trị. Có thể sử dụng hàm trong các biểu thức.
- Chúng ta có thể nhóm các scalar function theo nhóm sau:

Configuration	Trả về các thông tin về configuration
Cursor	Trả về các thông tin về Cursor
DateTime	Hàm tác động lên giá trị dateTime nhập vào và trả về một giá trị là string, numeric, hoặc datetime
Mathematical	Hàm số học
Metadata	Thông tin về database
String	Các hàm chuỗi

Functions (3) _ Ví dụ

```
SELECT DB_NAME() AS 'database'
```

Database

Northwind

(1 row(s) affected)

```
SET DATEFORMAT dmy
```

```
GO
```

```
DECLARE @vdate datetime
```

```
SET @vdate = '29/11/00'
```

```
SELECT @vdate
```

2000-11-29 00:00:00.000

Mathematical Functions

<u>ABS</u>	<u>DEGREES</u>	<u>RAND</u>
<u>ACOS</u>	<u>EXP</u>	<u>ROUND</u>
<u>ASIN</u>	<u>FLOOR</u>	<u>SIGN</u>
<u>ATAN</u>	<u>LOG</u>	<u>SIN</u>
<u>ATN₂</u>	<u>LOG₁₀</u>	<u>SQUARE</u>
<u>CEILING</u>	<u>PI</u>	<u>SQRT</u>
<u>COS</u>	<u>POWER</u>	<u>TAN</u>
<u>COT</u>	<u>RADIANS</u>	

Aggregate Functions

<u>AVG</u>	<u>MAX</u>
<u>BINARY_CHECKSUM</u>	<u>MIN</u>
<u>CHECKSUM</u>	<u>SUM</u>
<u>CHECKSUM_AGG</u>	<u>STDEV</u>
<u>COUNT</u>	<u>STDEVP</u>
<u>COUNT_BIG</u>	<u>VAR</u>
<u>GROUPING</u>	<u>VARP</u>

DateTime functions

Function	Determinism
<u>DATEADD</u>	Deterministic
<u>DATEDIFF</u>	Deterministic
<u>DATENAME</u>	Nondeterministic
<u>DATEPART</u>	Deterministic except when used as DATEPART (dw, date). dw, the weekday datepart, depends on the value set by SET DATEFIRST, which sets the first day of the week.
<u>DAY</u>	Deterministic
<u>GETDATE</u>	Nondeterministic
<u>GETUTCDATE</u>	Nondeterministic
<u>MONTH</u>	Deterministic
<u>YEAR</u>	Deterministic

String functions

<u>ASCII</u>	<u>NCHAR</u>	<u>SOUNDEX</u>
<u>CHAR</u>	<u>PATINDEX</u>	<u>SPACE</u>
<u>CHARINDEX</u>	<u>REPLACE</u>	<u>STR</u>
<u>DIFFERENCE</u>	<u>QUOTENAME</u>	<u>STUFF</u>
<u>LEFT</u>	<u>REPLICATE</u>	<u>SUBSTRING</u>
<u>LEN</u>	<u>REVERSE</u>	<u>UNICODE</u>
<u>LOWER</u>	<u>RIGHT</u>	<u>UPPER</u>
<u>LTRIM</u>	<u>RTRIM</u>	

Cast và Convert

- **CAST (*expression AS data_type*)**
- **CONVERT (*data_type [(length)] , expression [, style]*)**

Cấu trúc điều khiển

- BEGIN...END
- IF...ELSE
- WHILE
- CASE ... WHEN
- TRY...CATCH
- BREAK / CONTINUE
- GOTO
- RETURN

BEGIN...END

BEGIN...END: định nghĩa một khối lệnh

BEGIN

sql_statement | statement_block

END

If .. else

IF Boolean_expression

 { *sql_statement* | *statement_block* }

 [**ELSE**

 { *sql_statement* | *statement_block* }]

```
if (select count(*) from customers where  
    country='canada') > 0
```

```
begin
```

```
    print 'There are many Canada customers'
```

```
end
```

```
else
```

```
    print 'Welcome'
```

Demo If.... Else

```
DECLARE @MaritalStatus NCHAR(2)
```

```
SELECT @MaritalStatus = MaritalStatus  
FROM HumanResources.Employee WHERE EmployeeID = 1
```

```
IF (RTRIM(@MaritalStatus) = 'M')  
    PRINT 'This is a man.'  
ELSE  
    PRINT 'This is a woman.'
```

While

WHILE Boolean_expression

{ *sql_statement* | *statement_block* }

[**BREAK**]

{ *sql_statement* | *statement_block* }

[**CONTINUE**]

- **BREAK:** thoát ra khỏi vòng while
- **CONTINUE:** restart lại vòng lặp, bỏ qua các lệnh sau CONTINUE.

Demo While

```
USE AdventureWorks;
GO
WHILE (SELECT AVG(ListPrice) FROM Production.Product) < $300
BEGIN
    UPDATE Production.Product
        SET ListPrice = ListPrice * 2
    SELECT MAX(ListPrice) FROM Production.Product
    IF (SELECT MAX(ListPrice) FROM Production.Product) > $500
        BREAK
    ELSE
        CONTINUE
END
PRINT 'Too much for the market to bear';
```

CASE...WHEN

Cấu trúc điều khiển: CASE...WHEN

- **Ước lượng một danh sách các điều kiện và trả về một trong nhiều biểu thức kết quả có thể.**
- Cú pháp:

CASE input_expression

WHEN when_expression THEN result_expression

[WHEN when_expression THEN result_expression_n]

[ELSE else_ result_ expression]

END

Demo CASE...WHEN

USE AdventureWorks;

GO

SELECT

ProductNumber ,

CASE ProductLine

WHEN 'R' THEN 'Road'

WHEN 'M' THEN 'Mountain'

WHEN 'T' THEN 'Touring'

WHEN 'S' THEN 'Other sale items'

ELSE 'Not for sale'

END Category ,

Name

FROM Production.Product

ORDER BY ProductNumber;

GO

GOTO

GOTO LabelName

```
IF (SELECT SYSTEM_USER()) = 'payroll'
```

```
    GOTO calculate_salary
```

-- Other program code would appear here.

-- When the IF statement evaluates to TRUE,

-- the statements between the GOTO and

-- the calculate_salary label are ignored.

-- When the IF statement evaluates to FALSE the

-- statements following the GOTO are executed.

```
calculate_salary:
```

-- Statements to calculate a salary would appear after the

GOTO

GOTO LabelName

- Thay đổi luồng thi hành đến một nhãn. Lệnh Transact-SQL hoặc các lệnh sau GOTO được bỏ qua và tiếp tục được xử lý tại nhãn
- Cú pháp:

Define the label:

label :

Alter the execution:

GOTO label

Demo GOTO

```
DECLARE @Counter int;
SET @Counter = 1;
WHILE @Counter < 10
BEGIN
    SELECT @Counter
    SET @Counter = @Counter + 1
    IF @Counter = 4 GOTO Branch_One --Jumps to the first branch.
    IF @Counter = 5 GOTO Branch_Two --This will never execute.
END
Branch_One:
    SELECT 'Jumping To Branch One.'
    GOTO Branch_Three; --This will prevent Branch_Two from executing.
Branch_Two:
    SELECT 'Jumping To Branch Two.'
Branch_Three:
    SELECT 'Jumping To Branch Three.'
```

Lệnh RAISERROR

- Trả về một thông báo lỗi do user định nghĩa
- Cú pháp:

```
RAISERROR ( { msg_id | msg_str }  
    { , severity , state } )  
    [ WITH option [ ,...n ] ]
```

Tham số

- *msg_id* là mã lỗi được lưu trong sysmessages table.
- *msg_str*: là chuỗi thông báo lỗi được định dạng giống như lệnh printf trong lập trình C
- Severity: mức độ nghiêm trọng của lỗi. Có giá trị từ 0->18 được dùng bởi user, từ 19 -> 25 được dùng bởi sysadmin (dùng với WITH LOG).
- State: số nguyên từ 1 ->127 mô tả mức độ cần thiết của lỗi.

Các cách thực hiện các lệnh T-SQL

- Các lệnh có cấu trúc động
- Dùng Batch
- Dùng Scripts
- Dùng Transactions
- Dùng XML

Dùng cấu trúc lệnh động

- Dùng lệnh EXECUTE với các hằng chuỗi và biến
- Sử dụng khi ta phải gán giá trị cho biến tại thời điểm execute
- Các biến và table tạm chỉ tồn tại trong thời gian thực thi lệnh.

```
DECLARE @dbname varchar(30), @tblname varchar(30)
SET @dbname = 'Northwind'
SET @tblname = 'Products'
EXECUTE
('USE ' + @dbname + ' SELECT * FROM ' + @tblname)
```

Sử dụng khối (batch)

- Một hoặc nhiều lệnh T- SQL được submit cùng lúc với nhau.
- Sử dụng lệnh GO để kết thúc một khối
- Các biến không thể tham chiếu sau lệnh GO
- Không thể dùng các lệnh sau đây trong batch:
 - CREATE PROCEDURE
 - CREATE VIEW
 - CREATE TRIGGER
 - CREATE RULE
 - CREATE DEFAULT

Ví dụ lệnh khôi lệnh batch hợp lệ

CREATE DATABASE ...

CREATE TABLE ...

GO

CREATE VIEW1 ...

GO

CREATE VIEW2 ...

GO

Ví dụ về
khối lệnh
batch
không
hợp lệ

CREATE DATABASE ...

~~CREATE TABLE ...~~

CREATE TRIGGER ...

CREATE TRIGGER ...

GO

Sử dụng script

- Script là một tập tin có phần mở rộng là .sql, có nội dung là các lệnh T-SQL, được tạo bởi bất kỳ một Text editor nào.
- Được thực hiện trong công cụ SQL Query Analyzer hoặc osql Utility
- Được dùng lại khi cần.

Dùng Transactions

- Được xử lý giống một Batch
- Bảo đảm tính toàn vẹn dữ liệu
- Toàn bộ các lệnh trong transaction sẽ thành công hoặc không thành công
- Một transaction có thể có nhiều batch
- Transaction được bắt đầu bằng lệnh

BEGIN TRANSACTION

Và kết thúc bằng lệnh

COMMIT TRANSACTION

Hoặc

ROLLBACK TRANSACTION

Dùng XML

- **Cho phép Client Browser định dạng dữ liệu:** Dùng mệnh đề FOR XML trong lệnh SELECT để trả kết quả dạng XML
- Dùng **FOR XML AUTO**
- Hoặc **FOR XML RAW**
- Khi dùng mệnh đề FOR XML trong lệnh SELECT, ta không thể dùng:
 - Lệnh SELECT lồng nhau
 - Mệnh đề INTO .
 - Mệnh đề COMPUTE BY.
 - Dùng Stored procedures mà được gọi trong lệnh INSERT
 - Định nghĩa view hoặc user-defined function để trả về một rowset.

Dùng XML (2)

Use QLVT

```
SELECT makh, tenkh FROM khachhang  
FOR XML RAW
```

```
<row makh="KH01" tenkh="NGUYEN THI BE"/>  
<row makh="KH02" tenkh="LE HOANG NAM"/>  
<row makh="KH03" tenkh="TRAN THI CHIEU"/>  
<row makh="KH04" tenkh="MAI THI QUE ANH"/>  
<row makh="KH05" tenkh="LE VAN SANG"/>  
<row makh="KH06" tenkh="TRAN HOANG KHAI"/>
```

Stored Procedure

Mục tiêu bài học

- Hiểu được stored procedure là gì, procedure hoạt động như thế nào.
- Quản lý procedure: Tạo, xoá, sửa và thực thi
- Tham số trong store procedure
- Bài tập áp dụng

Nội dung bài học

- 1. Giới thiệu Stored procedures**
- 2. Tạo, thực thi, cập nhật và xoá stored procedures**
- 3. Bài tập thực hành**
- 4. Truyền tham số trong stored procedures**
- 5. Điều khiển lỗi**
- 6. Một số lưu ý**

Giới thiệu Stored Procedure

- **Stored procedure là một tập các lệnh Transact SQL được đặt tên và lưu trữ trong database server**
- **Có thể nhận tham số vào và tham số trả giá trị về**
- **Trả về trạng thái thực thi của procedure là thành công hay không thành công**

Xử lý Stored procedure (1)_Initial

Execution: Khi lần thứ nhất mà procedure được thực hiện hoặc khi procedure phải recompile, query processor sẽ đọc procedure trong process được gọi là resolution.

Sau giai đoạn resolution, SQL Server sẽ tạo ra một sơ đồ thực hiện và đặt sơ đồ này trong procedure cache

Creation: Các lệnh trong procedure sẽ được phân tích cú pháp theo cú pháp của T-SQL. Nếu thành công, tên của stored procedure được lưu trong SysObjects table, còn text của procedure lưu trong SysComments.

Delayed Name Resolution: cho phép stored procedure tham chiếu đến các đối tượng chưa tồn tại trong lúc procedure được tạo. Delayed Name resolution được thực hiện trong lúc procedure được thực hiện

xử lý stored procedure (2) Subsequent

Execution Plan Retrieved

Execution Plan

```
SELECT *
FROM
dbo.member
WHERE
member_no = ?
```

Execution Context

Connection 1

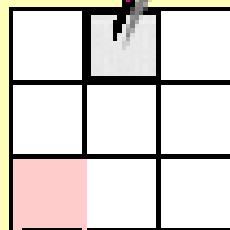
8082

Connection 2

24

Connection 3

1003



Unused plan is aged out

Lợi ích của stored procedure

- Cho phép lập trình theo hướng modular (modular programming)
- Thực thi nhanh hơn, giảm được việc chiếm dụng đường truyền mạng
- Bảo mật
- Xử lý các chức năng và chia sẻ với các ứng dụng khác

Cú pháp tạo procedure

```
CREATE PROCEDURE procedure_name
    [ { @parameter data_type }
    [ VARYING ] [ = default ] [ OUTPUT ]
    ] [ ,...n ]
    [ WITH
        { RECOMPILE | ENCRYPTION }]
AS
    sql_statement [ ...n ]
```

Lưu ý

- Có thể tham chiếu đến các tables, view, procedure khác cũng như các temporary table
- Để tạo một procedure, user phải có quyền CREATE PROCEDURE (sysadmin, hoặc database owner)
- Kích thước của procedure tối đa là 128 MB
- Có thể lồng 32 cấp procedure
- Dùng procedure sp_helptext để hiển thị nội dung text của stored procedure mà user đã tạo
- Không thể kết hợp lệnh create procedure với các lệnh SQL khác để tạo thành một bó lệnh (batch)
- Chỉ có thể tạo procedure trong database hiện hành.

Thực thi procedure

Lệnh để thực thi một stored procedure:

EXECUTE

[*@return_status =*] *procedure_name*

[[*@parameter =*] { *value* | *@variable*

 [**OUTPUT**] | [**DEFAULT**]]

 [,...n]

 [**WITH RECOMPILE**]

Ví dụ:

- Procedure không có tham số
- Procedure có tham số
- Procedure có tham số có giá trị default
- Procedure dùng output parameter

Tham số có kiểu là cursor

- Chỉ dùng cho tham số OUTPUT.
- Nếu kiểu của tham số là cursor thì VARYING và OUTPUT là bắt buộc.
- Nếu VARYING được chỉ định cho một tham số thì kiểu dữ liệu của tham số phải là cursor và từ khoá OUTPUT phải được chỉ định.

Function

- Trong SQL Server ta có thể viết hàm và lấy giá trị trả về. Các dạng hàm có thể viết như sau:
 - Hàm trả về giá trị vô hướng (scalar value) : varchar , int,
 - Hàm trả về giá trị là bảng tạm (inline table-valued) : table

Function

Cú pháp

CREATE FUNCTIONS function_name

([@parameter_name parameter_data_type])

RETURNS [return Data-type] /*Returns có 's' */

AS

Begin

return [scalar value/select command]

End

Function

- Viết hàm tính tuổi của người có năm sinh là@ns :

--Xóa hàm nếu đã có

```
if object_id('fTuoi','FN') is not null drop function fTuoi
```

go

--Tạo hàmfTuoi

```
Create function fTuoi (@ns int)
```

Returns int

As

Begin

```
return year(getdate()) - @ns
```

end

go

--Biên dịch hàm với F5

--Kiểm tra thử hàm

```
print dbo.fTuoi(1982) --phải có dbo.
```

Kiểu dữ liệu Cursor (1)

- Được dùng trong procedure hoặc trigger
- Chứa result set column, record
- Xử lý cursor:
 - Khai báo biến kiểu cursor chứa dữ liệu trả về
 - Kết hợp cursor với câu lệnh select bằng lệnh DECLARE CURSOR
 - Dùng lệnh OPEN để mở cursor
 - Dùng lệnh FETCH INTO để đổ một record hiện hành vào các biến tương ứng với từng column.
 - Dùng lệnh CLOSE để đóng cursor

Kiểu dữ liệu Cursor (2)

- **sp_cursor_list** để lấy ra danh sách các cursor hiện có
- **sp_describe_cursor**, **sp_describe_cursor_column** và **sp_describe_cursor_tables** để xem đặc tính của cursor
- Sau khi cursor mở, hàm **@@CURSOR_ROWS** tra về số lượng record
- Sau lệnh FETCH, hàm **@@FETCH_STATUS** để phản ánh trạng thái fetch sau cùng (0,-1)

System stored procedure	Description
sp_cursor_list	Returns a list of cursors currently visible on the connection and their attributes.
sp_describe_cursor	Describes the attributes of a cursor, such as whether it is a forward-only or scrolling cursor.
sp_describe_cursor_columns	Describes the attributes of the columns in the cursor result set.
sp_describe_cursor_tables	Describes the base tables accessed by the cursor.

cú pháp khai báo cursor

```
DECLARE cursor_name CURSOR  
[ LOCAL | GLOBAL ]  
[ FORWARD_ONLY | SCROLL ]  
[ STATIC | KEYSET | DYNAMIC |  
FAST_FORWARD ]  
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]  
FOR select_statement  
[ FOR UPDATE [ OF column_name [ ,...n ] ] ]
```

Ví dụ 1

```
DECLARE customer_cursor CURSOR  
FOR SELECT * FROM customers  
OPEN customer_cursor -- mở cursor  
FETCH NEXT FROM customer_cursor
```

Ví dụ 2 (1)

```
DECLARE @customerId varchar(11), @CompanyName  
varchar(30), @message varchar(80)  
PRINT "----- Customer report -----"  
DECLARE customer_cursor CURSOR  
FOR SELECT customerId, companyName FROM  
customers WHERE country = "USA"  
OPEN customer_cursor  
FETCH NEXT FROM customer_cursor INTO  
@customerId, @CompanyName
```

Ví dụ 2 (2)

While @@FETCH_STATUS = 0

begin

 print 'Customer ID: ' + @customerID

 print 'Company Name: ' + @companyName

 Fetch next from customer_cursor into
 @customerid, @CompanyName

end

Close customer_cursor

Deallocate customer_cursor

go

Sử dụng OUTPUT cursor parameter

```
USE northwind
```

```
CREATE PROCEDURE customer_cursor
    @customer_cursor CURSOR VARYING OUTPUT AS
SET @customer_cursor = CURSOR FORWARD_ONLY
    STATIC
FOR
SELECT * FROM CUSTOMERS
OPEN @customer_cursor
GO
```

Sử dụng tham số cursor trả về

```
DECLARE @MyCursor CURSOR  
EXEC customer_cursor @customer_cursor =  
    @MyCursor OUTPUT  
WHILE (@@FETCH_STATUS = 0)  
BEGIN  
    FETCH NEXT FROM @MyCursor  
END  
CLOSE @MyCursor  
DEALLOCATE @MyCursor
```

Bài tập ứng dụng

- Tạo procedure và thực thi để in ra company name có số lượng orders nhiều nhất
- Tạo proc p1 để trả về doanh thu của năm truyền vào, nếu user không truyền ngày vào thì lấy năm hiện hành
- Khai báo một procedure CustomersOfCountryCursor để lấy ra một cursor chứa các record của table customers có country bằng giá trị truyền vào. Thực thi CustomersOfCountryCursor và in ra các dữ liệu có trong cursor trả về.

Bài tập áp dụng (database QLVT)

1. Tạo procedure P1 để lấy ra danh sách các hoá đơn gồm các thông tin: MAHD, NGAY, TENKH, TONGTG
2. Tạo procedure P2 để xoá các chi tiết hoá đơn của hoá đơn có mã là tham số truyền vào
3. Tạo procedure P3 để tính tổng doanh thu của năm với năm là tham số truyền vào và trả về giá trị là tổng doanh thu đã tính được.

Tóm tắt nội dung buổi học

- Stored procedure trong SQL Server giống procedure trong các ngôn ngữ lập trình
- Xử lý nhanh hơn batch
- Procedure có thể có các tham số input và output
- thực thi một stored procedure dùng lệnh execute

Q & A

- Tạo proc lấy ra danh sách khách hàng có địa chỉ là tham số truyền vào:

```
CREATE PROC P2 @DC VARCHAR(50)
```

```
AS
```

```
select * from khachhang where diachi=@dc
```

- Create proc P1

As

```
Select hd.mahd, tenkh, ngay, sum(sl*giaban) as tongtg  
From hoадон hd, khachhang kh, chitiethoadon cthd  
Where (hd.mahd=cthд.mahd) and  
      (kh.makh = hd.makh)  
Group by hd.mahd, tenkh, ngay
```

- Create proc p2 @mahd nvarchar(10)

As

```
delete * from chitiethoadon where mahd=@mahd
```

Go

-- thuc thi

Exec p2 'hd002'

go

- Create proc p3 @nam int, @dt int OUTPUT
As

```
select @dt=sum(sl*giaban)
From chitiethoadon cthd, hoadon hd
Where (hd.mahd=cthdd.mahd) AND
      year(ngay)= @nam
```

Go

```
Declare @dt int
Exec p3 2000,@dt OUTPUT
Print 'Doanh thu nam 2000 la ' + str(@dt,8)
```

Trong QLVT

- Tạo procedure để lấy ra tên của các khách hàng đã mua hàng trong tháng Và năm (tham số input). Danh sách này được trả về trong một kiểu cursor.
- Thực thi P4 để lấy ra danh sách các khách hàng của tháng 6 năm 2000 và in ra tên của các khách hàng đó.

Trong NorthWind

- Khai báo một procedure `CustomersOfCountryCursor` để lấy ra một cursor chứa các record của table `customers` có country bằng giá trị truyền vào. Thực thi `CustomersOfCountryCursor` và in ra các dữ liệu có trong cursor trả về.

TRIGGER

Sau bài học này, sinh viên có thể:

- Hiểu được trigger là gì, công dụng của nó
- Tạo trigger
- Xoá trigger
- Thay đổi trigger

Nội dung

- 1. Trigger là gì ?**
- 2. Khi nào ta cần sử dụng Trigger**
- 3. Đặc điểm của Trigger**
- 4. Tạo Một Trigger như thế nào?**
- 5. Các ví dụ Trigger**
- 6. Các lưu ý**

Trigger là gì ?

- Trigger là một stored procedure đặc biệt **được gọi tự động** khi user cập nhật dữ liệu trên một table
- Được kết hợp với table: Được định nghĩa trên một table cụ thể .
- Được gọi tự động: Khi có một thao tác cập nhật dữ liệu trên table (insert, update, hoặc delete) thì trigger của thao tác tương ứng được tự động thực hiện.
- Khác với procedure, trigger không thể được gọi trực tiếp, **không nhận tham số**
- Là một phần của transaction: những lệnh trong trigger được xem là một single transaction, có thể được roll back từ bất kỳ chỗ nào trong trigger

Khi nào ta cần sử dụng trigger

- Ta chỉ sử dụng trigger khi mà các biện pháp bảo đảm data intergrity khác như Constraints không thể thỏa mãn yêu cầu của ứng dụng.
- Thực hiện cascade updates và cascade deletes qua các table quan hệ trong database
- Ép buộc tính toàn vẹn dữ liệu phức tạp: Thực hiện các ràng buộc có tham chiếu đến các column trong nhiều table.
- Định nghĩa Custom Error Messages: Dùng trigger để trả về các chuỗi thông báo trạng thái của một hàng động nào đó.
- Bảo trì các dữ liệu không được chuẩn hóa:!

Đặc điểm của Trigger

- Trigger được thực hiện tự động sau khi lệnh **INSERT**, **UPDATE**, hoặc **DELETE** được thực hiện trên một table mà trigger đó được định nghĩa. Còn các constraints và **INSTEAD OF** trigger sẽ được kiểm tra trước khi lệnh **INSERT**, **UPDATE**, hoặc **DELETE** thực hiện.
- **Constraints** sẽ được kiểm tra trước trigger.
- Một table có thể có nhiều Triggers cho một action. Một trigger có thể được định nghĩa cho nhiều action.

Đặc điểm của Trigger

- Khi có nhiều trigger trong một table, thì table owner có thể dùng procedure hệ thống `sp_settriggerorder` để chỉ định trigger đầu và trigger cuối để thực thi. Thứ tự của các trigger còn lại không thể sắp xếp được.
- User phải có quyền để thực hiện tất cả các lệnh mà được định nghĩa trong Triggers
- Table Owners không thể tạo ra các Triggers trên Views hoặc Temporary Tables nhưng có thể tham chiếu đến view và temporary.

Đặc điểm của Trigger

- Triggers không trả kết quả về.
- Triggers có thể điều khiển multi-row actions: một hành động **INSERT**, **UPDATE**, hoặc **DELETE** gọi một trigger có thể ảnh hưởng lên nhiều dòng dữ liệu, Ta có thể chọn:
 - Xử lý tất cả các dòng cùng với nhau trong trường hợp các dòng ảnh hưởng phải thỏa điều kiện của trigger.
 - Xử lý từng dòng thỏa điều kiện.

Logic tables

- Khi có action Insert, table logic **inserted** sinh ra, có cấu trúc giống với cấu trúc table được insert, có dữ liệu là record đang được insert
- Khi có action delete, table **deleted** sinh ra, có cấu trúc giống với cấu trúc table bị deleted, có dữ liệu là record đang bị xoá
- Khi có action update, có 2 table **inserted** và **deleted**

Tạo trigger

```
CREATE TRIGGER trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{
  {{ FOR | AFTER | INSTEAD OF }
    {[ INSERT ][,][ UPDATE ][,][ DELETE] }
    [ WITH APPEND ]
  AS
    [ { IF UPDATE ( column )
        [ { AND | OR } UPDATE ( column ) ] [...n ]
      }
    ]
    sql_statement [...n ]
  }
}
```

Tham số (1)

- **Table | view** : tên view/table mà trigger được thực hiện khi có action tương ứng.
- **WITH ENCRYPTION**: mã hoá nội dung text của lệnh create trigger trong table syscomments.
- **AFTER**: Trigger sẽ được gọi chỉ khi tất cả các hành động đã thực hiện xong. Các kiểm tra constrain và cascade sẽ được kiểm tra hoàn thành trước khi trigger thực hiện. Default là AFTER nếu chỉ có từ khoá FOR được chỉ định. AFTER trigger không thể định nghĩa trên view.
- **INSTEAD OF**: chỉ định trigger được thực hiện thay cho action của trigger. INSTEAD OF triggers không cho phép cập nhật dữ liệu trên view có WITH CHECK OPTION.

Tham số (2)

- { [DELETE] [,] [INSERT] [,] [UPDATE] } : chỉ định action gắn với trigger. Đối với INSTEAD OF triggers, action **DELETE** không cho phép trên table mà có relationship mà chỉ định **CASCADE ON DELETE**. Tương tự, action **UPDATE** không cho phép trên table có relationships mà **CASCADE ON UPDATE**.
- Table **deleted** và **inserted** là logical tables. Chúng có **cấu trúc giống** với table mà trigger **được định nghĩa, chứa các dòng giá trị cũ hoặc mới** mà có thể thay đổi bởi action của user. Ta có truy xuất dữ liệu trong 2 table này trong định nghĩa trigger.

Tham số (3)

- Các giá trị kiểu **text**, **ntext**, hoặc **image** trong table **inserted** và **deleted** không truy xuất được.
- Khi trigger ở mức 65, giá trị null sẽ được trả về cột có kiểu **text**, **ntext**, hoặc **image** trong table **inserted** hoặc **deleted** nếu cột cho phép null; chuỗi zero-length được trả về nếu cột có thể null.
- IF UPDATE (*column*): **kiểm tra action update trên cột được chỉ định, không dùng cho action delete.**
- With Append: Chèn thêm trigger này vào các trigger đã có trước đó

Ví dụ

```
Use Northwind
GO
CREATE TRIGGER Empl_Delete ON Employees
FOR DELETE
AS
IF (SELECT COUNT(*) FROM Deleted) > 1
BEGIN
    RAISERROR(
        'You cannot delete more than one employee at a time.',
        16, 1)
    ROLLBACK TRANSACTION
END
```

Những lệnh sau đây không được dùng trong định nghĩa trigger

- ALTER DATABASE
- CREATE DATABASE
- DISK INIT
- DISK RESIZE
- DROP DATABASE
- LOAD DATABASE
- LOAD LOG
- RECONFIGURE
- RESTORE DATABASE
- RESTORE LOG

Ví dụ (1)

USE pubs

```
IF EXISTS (SELECT name FROM sysobjects WHERE name =  
    'reminder' AND type = 'TR')
```

```
DROP TRIGGER reminder
```

```
GO
```

```
CREATE TRIGGER reminder ON titles
```

```
    FOR INSERT, UPDATE
```

```
    AS
```

```
        RAISERROR (50009, 16, 10)
```

```
GO
```

Alter trigger

```
ALTER TRIGGER trigger_name ON table
    [WITH ENCRYPTION]
    {{FOR {[,] [DELETE] [,] [UPDATE] [,][INSERT]}}
AS
sql_statement [...n] }
|      {FOR {[,] [INSERT] [,] [UPDATE]}}
AS
IF UPDATE (column)
[ {AND | OR} UPDATE (column) [...n]]
sql_statement [...n] }
}
```

Xoá trigger

DROP TRIGGER *trigger_name*

Bài tập áp dụng (QLVT)

- Tạo trigger để khi insert một record vào trong table CHITIETHOADON, thì cập nhật lại SLTON của vật tư đó trong table VATTU
- Tạo trigger để không cho phép một hoá đơn có nhiều hơn 4 chi tiết hoá đơn
- Tạo trigger không cho phép hai vật tư trùng tên
- Tạo trigger để không cho phép xoá cùng lúc nhiều hơn một khách hàng

- Tạo trigger để không cho phép xoá một vật tư mà đã có ít nhất một chi tiết hoá đơn của vật tư đó.
- Tạo trigger để kiểm tra số lượng bán ra của một vật tư phải nhỏ hơn số lượng tồn trong kho

```
create trigger t1
on chitiethoadon1
for insert
as
declare @sl int, @mavt varchar(10)
select @sl = sl, @mavt = mavt from inserted
update vattu1 set slt=slt- @sl where mavt =@mavt
```