

Chương 4

1

Lập trình trong SQL

Khái niệm lập trình trong SQL

- **Lập trình CSDL:** Giao tiếp với chương trình ứng dụng
 - Chương trình bao gồm: Biến (variable), câu lệnh SQL và cấu trúc điều khiển.
- Các khái niệm cơ bản:
 - Định danh (Identifiers)
 - Batch (tập các câu lệnh T-SQL liên tiếp kết thúc bằng lệnh GO)
 - Script

Kiểu dữ liệu trong T-SQL

- Kiểu dữ liệu của hệ thống (System - supplied data type)
- Kiểu dữ liệu do người dùng định nghĩa (User-defined data type)

Kiểu dữ liệu của hệ thống

Data type name	Class	Size in Bytes
SmallDateTime	Date/Time	4
Date	Date/Time	3
Cursor	Special Numeric	1
Char	Character	Varies
VarChar	Character	Character
Text	Character	Varies
NChar	Unicode	Varies
NVarChar	Unicode	Varies

Kiểu dữ liệu của hệ thống

Data type name	Class	Size in Bytes
Ntext	Unicode	Varies
Binary	Binary	Varies
VarBinary	Binary	Varies
Image	Binary	Varies
Table	Other	Special
XML	Character	Varies

Kiểu dữ liệu người dùng định nghĩa

- Tạo kiểu dữ liệu từ các kiểu dữ liệu của SQL Server

```
CREATE TYPE DBO.NAME  
FROM datatype_name
```

- Tạo kiểu dữ liệu TABLE

```
CREATE TYPE DeptType AS TABLE (  
    DeptId INT, DeptName VARCHAR(30))
```

Các toán tử dùng trong T-SQL

- Toán tử toán học : $*/ + - \%$
- Toán tử so sánh: $=, !=, <>, >, <, >=, <=, !<, !>$
- Toán tử luận lý: ALL, AND, ANY, BETWEEN, EXISTS, IN, LIKE, NOT, OR, IS NULL, UNIQUE

Hàm trong T-SQL

- Hàm có sẵn trong SQL
- Hàm do người dùng định nghĩa

Hàm có sẵn trong SQL

- ▶ Transact-SQL gồm các nhóm hàm sau:
 - ▶ Hàm thống kê (Aggregate Functions)
 - ▶ Hàm chuyển đổi (Conversion Functions)
 - ▶ Hàm ngày giờ (Date Functions)
 - ▶ Hàm toán học (Mathematical Functions)
 - ▶ Hàm xử lý chuỗi

Hàm thống kê (Aggregate Functions)

Function	Return Value	Example
SUM(col_name)	Trả về tổng giá trị trong cột	SELECT SUM (soluong) AS Tong FROM chitietHD
AVG(col_name)	Trả về giá trị trung bình trong cột	SELECT AVG (Dongia) AS DogiaTB FROM ChitietHD
COUNT	Trả về tổng số record trong table thỏa điều kiện cho trước	SELECT COUNT (*) AS tongsoHD FROM ChitietHD WHERE Soluong > 100
MAX(col_name)	Trả về giá trị lớn nhất trong dãy giá trị	SELECT MAX (Soluong * Dongia) AS TTMax FROM ChitietHD
MIN(col_name)	Trả về giá trị nhỏ nhất trong dãy giá trị	SELECT MIN (Sooluong * Dongia) AS TTMin FROM ChitietHD

Hàm chuyển đổi (Convert Functions)

- Dùng chuyển đổi giá trị của một kiểu dữ liệu này sang kiểu dữ liệu khác.
- Cú pháp

CONVERT(datatype[(length)], expression [,style])

Hàm chuyển đổi (Convert Functions)

➤ Trong đó:

- *datatype*: kiểu dữ liệu của SQL Server.
- *length*: tùy chọn chiều dài cho kiểu char, varchar, binary or varbinary, tối đa là 255
- *Expression*: xác định tên cột, hằng, hàm, biến hay một subquery
- *Style*: tùy chọn xác định dạng hiển thị ngày khi chuyển từ datetime/smallerdatetime sang dạng ký tự.

Hàm ngày giờ (Date Functions)

Datepart	Abbreviation	Values
Hour	hh	0-23
Minute	Mi	0-59
Second	Ss	0-59
Millisecond	Ms	0-999
Day of Year	Dy	1-366
Day	Dd	1-31
Week	wk	1-53
Weekday	dw	1-7
Month	mm	1-12
Quarter	qq	1-4
Year	yy	1753-9999

Hàm ngày giờ (Date Functions)

► Các hàm được dùng với DatePart

Hàm	Giá trị trả về	VD
GETDATE()	Trả về ngày tháng hiện hành	SELECT GETDATE()
DATEADD(datepart, number, date)	Cộng thêm một số (number) vào một ngày (date) tương ứng với datepart	SELECT DATEADD(mm,4,'01/01/99') - returns 05/01/99 in the current date format
DATEDIFF(datepart, date1, date2)	Trả về số chênh lệch giữa 2 ngày tương ứng với datepart	SELECT DATEDIFF(mm,'01/01/99','05/01/99') - returns 4
DATENAME(datepart, date)	Trả về tên thứ của ngày (date)	SELECT DATENAME(dw,'01/01/2000') - returns Saturday
DATEPART(datepart, date)	Trả về số tương ứng trong datepart.	SELECT DATEPART(day,'01/15/2000') - returns 15

Hàm toán học

Hàm	Giá trị trả về	VD
ABS(num_expr)	Trị tuyệt đối của một số	SELECT ABS(-43) → 43
CEILING(num_expr)	Làm tròn về số nguyên gần nhất (lớn hơn) một số	SELECT CEILING(43.5) → 44
FLOOR(num_expr)	Làm tròn về xuống nguyên (nhỏ hơn) một số	SELECT FLOOR(43.5) → 43
POWER(num_expr,y)	Lũy thừa của một số	SELECT POWER(5,2) → 25
ROUND(num_expr,length)	Làm tròn đến số lẻ mong muốn	SELECT ROUND(43.543,1) → 43.500
SIGN(num_expr)	Trả về 1 nếu num_expr > 0, -1 nếu num_expr < 0 và 0 nếu num_expr = 0	SELECT SIGN(-43) → -1
SQRT(float_expr)	Tính bình phương của một số	SELECT SQRT(9) → 3

Hàm xử lý chuỗi

Function	Description
LEFT(string, int)	Trích ra các ký tự từ bên trái của chuỗi
RIGHT(string, int)	Trích ra các ký tự từ bên trái của chuỗi
LEN(string)	Trả về chiều dài của chuỗi
LOWER(string)	Chuyển sang chữ thường
LTRIM(string)	Loại bỏ ký tự khoảng trắng
NCHAR(int	Trả về số ký tự UNICODE

Biến

- **Biến cục bộ (Local variable)**

- **Cú pháp khai báo:**

DECLARE@ VariableName var_type

- **Ví dụ: DECLARE @EmpIDVar int**

Biến

- **Gán giá trị cho biến:** Khi một biến được khai báo thì giá trị của nó là Null

SET @VariableName = expression

or

SELECT{@VariableName=expression} [...n]

Biến

► Ví dụ:

```
DECLARE @TenTP varchar(10)  
SET @ TenTP = 'HCM'  
SELECT * FROM KhachHang  
WHERE ThanhPho= @TenTP
```

Biến toàn cục (Global Variables)

- Biến toàn cục trong SQL là một hàm hệ thống.
 - Giá trị trả về của hàm được hiển thị bởi câu lệnh **SELECT @@Variablename.**
 - Không gán giá trị cho biến toàn cục.
 - Biến toàn cục không có kiểu
 - Tên biến được bắt đầu với @@.

Biến toàn cục (Global Variables)

➤ Một số biến toàn cục thông dụng

- **@@SERVERNAME**: trả về tên của server
- **@@ROWCOUNT**: số dòng chịu tác dụng của câu lệnh cuối cùng.
- **@@ERROR**: trả về chỉ số index của lỗi
- **@@IDENTITY**: trả về định danh .

Biến toàn cục (Global Variables)

Ví dụ:

Update Nhanvien

set TenNV= 'Hoang'

Where HoNV='Nguyen'

If(@@rowcount =0)

begin

print 'Khong co record nao duoc Update'

return

end

Cấu trúc điều khiển

➡ IF ... ELSE

```
IF boolean_expression  
    {sql_statement | statement_block}  
[ELSE boolean_expression  
    {sql_statement | statement_block}]
```

Cấu trúc điều khiển

➡ BEGIN ...END

BEGIN

{sql_statement / statement_block}

END

Cấu trúc điều khiển

► Ví dụ:

```
if (select sum(soluong)
    from chitiethoadon
    where masach='s001')<20
begin
    print N'Còn sách trong kho'
end
else
    print N'Đã hết'
```

Cấu trúc điều khiển

➔ WHILE

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

Cấu trúc điều khiển

Ví dụ 1:

```
DECLARE @counter INT
SET @counter=0
WHILE (@counter<20)
BEGIN
    INSERT INTO nhomsach1
VALUES ('N00'+CAST(@counter as char(2)),
        N'Sách_'+CAST(@counter as char(2)))
    SET @counter=@counter+1
END
```

Cấu trúc điều khiển

➡ CASE

➡ Cú pháp CASE đơn giản

CASE *input_expression*

WHEN *when_expression* **THEN** *result_expression* [...*n*]

[**ELSE** *else_result_expression*]

END

Cấu trúc điều khiển

► Cú pháp CASE đầy đủ:

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

Cấu trúc điều khiển

► Ví dụ

```
SELECT Masach, SLton, Dongia, [Giamgia]=  
CASE  
    WHEN SLton <=5 THEN 0.05  
    WHEN SLton BETWEEN 6 and 10 THEN 0.07  
    WHEN SLton BETWEEN 11 and 20 THEN 0.09  
    ELSE  
        0.1  
    END  
from danhmucsach  
order by MaSach
```

Thủ tục (Stored Procedures) và Hàm (Function)

Thủ tục (Stored procedure - SP)

- **Thủ tục:** là một đoạn mã chứa các khai báo hoặc các câu lệnh SQL.
 - Stored procedure được lưu trữ trong danh mục CSDL server và nó có thể được gọi từ trigger, một thủ tục khác hoặc một ứng dụng phía client.
 - Stored procedures có thể được tạo trong CSDL và tái sử dụng.
 - Thủ tục nhận tham số đầu vào và trả về một kết quả.

Loại thủ tục

- **System SP (sp):** được lưu trữ trong CSDL Master, nhưng có thể thực thi ở bất kỳ CSDL nào.
 - **sp_helptext:** In nội dung của rule, a default, an unencrypted stored procedure, user-defined function, trigger, computed column, or view.
 - **sp_help:** Xuất thông tin về một đối tượng của CSDL.
 - **sp_depends:** Hiển thị thông tin về đối tượng của CSDL phụ thuộc vào view(s), trigger(s), và procedure(s) trong CSDL.

Loại thủ tục

- ❑ **Extended SP (xp):** được tạo từ ngôn ngữ khác (C++,...) và được sử dụng như một thủ tục của SQL Server.
- ❑ **User_defined:**
 - *Local sp*: là đối tượng trong CSDL dùng để thực thi các tác vụ, có thể tạo trong CSDL master.
 - *Temporary sp*: local (tên bắt đầu với #) và global (tên bắt đầu với ##).

Tạo thủ tục

➡ Cú pháp:

```
CREATE PROC [EDURE ] procedure_name  
[ ; number ] [ { @parameter data_type }  
[ VARYING ] [ = default ] [ OUTPUT ] ] [,...n ]  
[ WITH { RECOMPILE | ENCRYPTION | RECOMPILE,  
ENCRYPTION } ]  
[FOR REPLICATION ]  
AS sql_statement [ ...n ]
```

Tạo thủ tục

Ví dụ: *tạo thủ tục tính tổng tiền của mỗi nhân viên*

```
CREATE PROCEDURE TongtienHD
AS
    SELECT MaNV, Tongtien=SUM(Soluong*Dongiaban)
    FROM Hoadon h JOIN Chitiethoadon c
    ON (h.MaHD= c.MaHD)
    GROUP BY MaNV
```

Thực thi thủ tục

- **Execute**: Dùng để thực thi
 - User-defined function.
 - System procedure.
 - User-defined stored procedure,
 - Extended stored procedure.

Thực thi thủ tục

- Cú pháp:

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

Thực thi thủ tục

➡ Hoặc:

```
EXECUTE ProductName [ ; number ]  
[<parameter>[, ...n][ OUTPUT ]]
```

Ví dụ: `exec TongtienHD`

Hiệu chỉnh thủ tục

➡ **Cú pháp:**

```
ALTER PROC[EDURE] procedure_name  
[WITH option]  
AS  
    sql_statement [...n]
```

Xóa thủ tục

➤ **Cú pháp:**

```
DROP PROC owner.stored_procedure_name
```

Tham số trong thủ tục

➔ Input parameter:

```
CREATE PROCEDURE procedure_name  
[@parameter_name data_type] [=default_value]  
[WITH option]  
AS  
sql_statement [...n]
```

Tham số trong thủ tục

Ví dụ1: *Tính tổng tiền hóa đơn của một nhân viên tùy ý*

```
CREATE PROCEDURE TongtienHD_TS @manv char(5)
AS
    SELECT h.MaHd, Tongtien=SUM(Soluong*Dongiaban)
    FROM Hoadon h JOIN Chitiethoadon c
    ON (h.MaHD= c.MaHD)
    where manv=@manv
    GROUP BY h.MaHd
```

Thực thi

```
exec TongtienHD_ts 'nv005'
```

Tham số trong thủ tục

➡ Output parameter

```
CREATE PROCEDURE procedure_name  
[@parameter_name data_type] [=default_value]  
OUTPUT  
[WITH option]  
AS  
sql_statement [...n]
```

Tham số trong thủ tục

► Ví dụ:

```
CREATE PROC count_row  
    @movie_count int OUTPUT  
AS  
SELECT @movie_count = COUNT(*) FROM Movie  
GO
```

Khi thực thi output parameter: cần khai báo biến để lưu trữ giá trị trả về của output parameter

```
DECLARE @num int  
EXEC count_row @num OUTPUT  
SELECT @num
```

Hàm (FUNCTION)

➤ Hàm hệ thống (System function):

- **Aggregate function:** avg(), count(), count(*), sum(), max(), min(),...
- **Other function:** getdate(), month(), upper(), user_name(), @@rowcount,...

Hàm (FUNCTION)

- **Hàm do người dùng định nghĩa (User-defined function):**
 - Định nghĩa hàm T-SQL có thể nhận một hoặc nhiều tham số đầu vào và trả về một giá trị đơn hoặc một bảng giá trị
- **Có 3 loại hàm do người dùng định nghĩa:**
 - **Scalar:** trả về một giá trị đơn dựa trên giá trị đầu vào.
 - **Multi-statement Table-valued:** trả về một tập các dòng.
 - **Inline Table-valued:** trả về một tập các dòng.

Hàm (FUNCTION)

➤ Scalar function:

```
CREATE FUNCTION [ owner_name. ] function_name  
    ( [ { @parameter_name [AS] scalar_parameter_data_type  
        [= default ] } [ ,...n ] ] )  
RETURNS scalar_return_data_type.  
[WITH < function_option > [ [ , ] ...n ] ]  
[AS ]  
    BEGIN  
        function_body  
        RETURN scalar_expression  
    END
```

Hàm (FUNCTION)

- Ví dụ: viết hàm tính tổng số hóa đơn trong một tháng tùy ý

```
CREATE FUNCTION TongsoHD (@thang tinyint )
RETURNS tinyint
AS
BEGIN
    DECLARE @SoHD tinyint
    SELECT @SoHD = count(MaHD)
    FROM Hoadon
    WHERE month(NgayBan)= @thang
    RETURN @SoHD
END
```

Hàm (FUNCTION)

► Thực thi hàm:

- Hàm được gọi trong câu lệnh SQL

Ví dụ: `select [dbo].[TongsoHD] (7)`

Hàm (FUNCTION)

❑ Table-valued Functions

```
CREATE FUNCTION [ owner_name. ] function_name  
    ([{ @parameter_name  
        [AS] scalar_parameter_data_type [= default ] } [,...n ] ] )  
RETURNS TABLE  
    [WITH < function_option > [ [,] ...n ] ]  
    [AS ]  
    RETURN [(] select-stmt [)]
```

Hàm (FUNCTION)

- ➡ Ví dụ: viết hàm tính điểm trung bình của sinh viên

```
create function trungbinh1()  
returns table  
as  
    return  
        select s.masv, tensv, tb=round(avg(diemlan1),1)  
        from SINHVIEN s inner join KETQUA k on s.masv=k.masv  
        group by s.masv, tensv  
go
```

- ➡ Thực thi `select *from [dbo].[trungbinh1]()`

Hàm (FUNCTION)

► Multistatement Table-valued

```
CREATE FUNCTION [owner_name.]function_name  
([{@parameter_name [AS] data_type [=default]} [ ,...n ]])  
RETURNS @return_variable  
TABLE ({column_definition | table_constraint} [ ,...n ])  
[WITH { ENCRYPTION | SCHEMABINDING } [ [,] ...n ] ]  
[AS]  
    BEGIN  
        function_body  
    RETURN  
END
```

Hàm (FUNCTION)

► Ví dụ:

```
create function tbinh(@masv char(5))
returns @tbinh table(masv char(5), Tensv varchar(20), DiemTB float)
as
begin
    insert into @tbinh
        select s.masv, TenSV, diemtb=round(avg(diemlan1),1)
        from SINHVIEN s, KETQUA k
        where s.MaSV=k.MaSV and s.masv=@masv
        group by s.masv, TenSV
    return
end
```

Bài tập

- Câu 1: Viết hàm đếm số sản phẩm có từ 'box' ở trong quy cách sản phẩm (Quantity per Unit)
- Câu 2: cải tiến hàm ở câu 1 để đếm số sản phẩm có chứa chuỗi con bất kỳ trong quy cách sản phẩm
- Câu 3: Tính tổng doanh thu bán hàng do các nhân viên ở cùng 1 thành phố lập

Bài tập (tt)

- Câu 4: Viết hàm `f_capnhatsp(p1, p2, m, s)` – thực hiện – kiểm tra nếu sản phẩm có mã P1 được bán trong tháng @m có số lượng > @s thì giảm đơn giá bán xuống p2 (%). =>Trả về số lượng sản phẩm có điều chỉnh giá
- Câu 5: Tương tự câu 4 – nhưng hàm trả về danh sách các sản phẩm có điều chỉnh giá

57

TRIGGERS

Trigger

- **Trigger** là một loại thủ tục đặc biệt, tự động thực thi khi một sự kiện xảy ra trong database server.
- **DML triggers:** thực thi khi người dùng cố hiệu chỉnh dữ liệu thông qua một sự kiện ngôn ngữ thao tác dữ liệu (DML).
- DML events gồm: các câu lệnh
 - INSERT
 - UPDATE,
 - DELETE

Tạo trigger

➡ Cú pháp tạo trigger:

```
CREATE TRIGGER [ schema_name . ] trigger_name  
ON { table | view }  
[ WITH ENCRYPTION ]  
{  
  { { FOR | AFTER | INSTEAD OF }  
    { [DELETE] [,] [INSERT] [,] [UPDATE] }  
    [ WITH APPEND ]  
    [ NOT FOR REPLICATION ]  
    AS  
    sql_statement [ ...n ]  
  }  
}
```

Tạo trigger

- *Schema_name*: tên của lược đồ chứa trigger.
- *Table | view*: tên bảng hoặc View mà trên đó trigger thực thi.
- *WITH ENCRYPTION*: mã hóa trigger.
- *FOR | AFTER | INSTEAD OF*: loại trigger
- **[DELETE] [,] [INSERT] [,] [UPDATE]**: các hành động kích hoạt trigger.

Loại trigger

➤ **AFTER trigger** hoặc **FOR trigger** :

- Thực thi sau khi thực hiện insert hoặc delete các dòng trong table, gọi là reactive, chỉ tạo trên table.
- Khi tạo trigger nếu không chỉ định rõ thì mặc định là **AFTER Trigger**

Loại trigger

➤ **INSTEAD OF triggers:**

- Kiểm tra trước khi thực hiện Insert hoặc Delete, gọi là proactive, tạo trên *table* và *view*.
- Table1 có trigger1, table2 có trigger2, nếu thao tác trên table1 có liên quan đến table2 thì trigger2 tự động thực thi, gọi là trigger lồng (Nested Trigger)

Loại trigger

For/After	Instead of
- Chỉ áp dụng cho table	- Áp dụng cho table, view
- Có thể định nghĩa nhiều trigger trên một hành động I/U/D	- Chỉ định nghĩa một Trigger trên một hành động I/U/D
Thực thi sau khi : <ul style="list-style-type: none">+ Xử lý ràng buộc+ Thực hiện xong hành động I/U/D phát sinh trigger	- Thi hành trước khi: <ul style="list-style-type: none">+ Xử lý ràng buộc+ Thay thế hành động phát sinh trigger
	- Không xây dựng được trên table có áp dụng cascade D/U

Hành động kích hoạt trigger

- **[DELETE] [,] [INSERT] [,] [UPDATE]** : thao tác mà khi thực hiện thì trigger tự động thực thi.
 - **Khi Insert** mẫu tin mới vào bảng thì mẫu tin mới đó cũng lưu trong bảng **INSERTED**
 - **Khi Delete** mẫu tin trong bảng thì mẫu tin bị xóa được di chuyển sang bảng **DELETED**.
 - **Khi Update** mẫu tin trong bảng thì bảng được cập nhật và bảng **INSERTED** đều chứa mẫu tin mới, còn **DELETED** chứa mẫu tin cũ.

Thuộc tính của trigger

➤ WITH APPEND

- Chỉ định thêm một trigger
- WITH APPEND không được dùng với INSTEAD OF triggers.

➤ NOT FOR REPLICATION

- Trigger sẽ không thực hiện khi bảng có liên quan đến kỹ thuật sao chép nhân bản (rellication)
- ***sql_statement***: câu lệnh SQL chứa điều kiện và hành động của trigger.

Tạo trigger

Ví dụ:

```
CREATE TRIGGER NoDelete
ON Product
FOR DELETE AS
IF(SELECT ProductID FROM Deleted )=12
BEGIN
    Print 'You cannot delete the Productid=12'
    RollBack transaction
END
```

Tạo trigger

Ví dụ:

```
CREATE TRIGGER NoUpdate
ON Product
FOR Update
IF Update(ProductID)
BEGIN
    PRINT 'You cannot update Productid'
    RollBack Transaction
END
```

Tạo trigger

Ví dụ:

```
CREATE TRIGGER NoupdareOrders
ON Orders
FOR Update AS
IF (Select OrderDate from Deleted) > Getdate()
BEGIN
    Print 'Ngày lap hoa don <=ngay hien hanh'
    RollBack Transaction
END
```

Ví dụ

```
CREATE TRIGGER trInsNV  
ON NHANVIEN  
FOR INSERT  
AS
```

```
RAISERROR('%d hang da duoc them vao bang  
NHANVIEN', 0, 1, @@rowcount)
```

☞ Trigger này tự động được thực hiện mỗi khi có bản ghi mới được chèn vào bảng NHANVIEN

Ví dụ

```
CREATE TRIGGER trDeINV  
ON NHANVIEN  
FOR DELETE  
AS
```

```
RAISERROR('%d hàng bị xóa trong bảng  
NHANVIEN', 0, 1, @@rowcount)
```

☞ Trigger này tự động được thực hiện mỗi khi có một hoặc nhiều bản ghi bị xóa ở bảng NHANVIEN

Ví dụ

```
CREATE TRIGGER trUpNV  
ON NHANVIEN  
FOR UPDATE  
AS  
RAISERROR('%d %d hàng đã được sửa ở bảng bảng  
NHANVIEN', 0, 1, @@rowcount)
```

- ☞ Trigger này tự động được thực hiện mỗi khi có một câu lệnh Update được thực hiện trên bảng NHANVIEN.

Hiệu chỉnh Triggers

➡ Cú pháp:

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

Ví dụ: Đảm bảo ràng buộc toàn vẹn dữ liệu

```
CREATE TRIGGER trDelINV
ON NHANVIEN
FOR DELETE
AS
RAISERROR('%d hàng bị xóa trong bảng NHANVIEN', 0,
1,@@rowcount)
```

```
-----
CREATE TRIGGER trDelPhong
ON PHONG
FOR DELETE
AS
DELETE      NHANVIEN      FROM      DELETED      WHERE
      DELETED.MAPHONG =NHANVIEN.MAPHONG
```

Một số chú ý khi dùng trigger

- Một bảng có nhiều trigger
- Mỗi một trigger có tên duy nhất
- Trong trigger thường dùng mệnh đề **IF EXISTS**

Ví dụ

*Tạo một trigger kiểm tra khóa ngoại **Manhom** khi nhập dữ liệu vào bảng **Danhmucsach***

```
create trigger ktkhoangoai
on dmsach for insert
as
    if exists(select * from inserted
              where manhom not in(select manhom from nhomsach))
    begin
        print 'Ma nhom khong ton tai'
        rollback transaction
    end
```

Hiệu chỉnh Triggers

Ví dụ:

```
ALTER TABLE [Order Details]
```

```
DISABLE TRIGGER ALL
```

➡ **Xóa trigger:**

➡ **DROP TRIGGER Trigger_Name**