SQL Server – Function và Trigger

Posted by [*khoaba*](https://khoaba.wordpress.com/author/khoaba/) on [*06/09/2010*](https://khoaba.wordpress.com/2010/09/06/sql-server-function-va-trigger/)

Posted in: [Cơ sở dữ liệu](https://khoaba.wordpress.com/category/cong-ngh%e1%bb%87-thong-tin/th%e1%ba%bf-gi%e1%bb%9bi-l%e1%ba%adp-trinh/c%c6%a1-s%e1%bb%9f-d%e1%bb%af-li%e1%bb%87u/). [4 phản hồi](https://khoaba.wordpress.com/2010/09/06/sql-server-function-va-trigger/#comments)

Trong bài trước mình đã giới thiệu về [Stored Procedure](https://khoaba.wordpress.com/2010/09/06/stored-procedure/), Trong bài này chúng ta sẽ tìm hiểu về  Function và Trigger – một phần cũng không kém phần quan trong trong lập trình với cơ sở dữ liệu

**1. Hàm – Functions**

Cũng giống như [Stored Procedure](http://hmweb.com.vn/343/pro/71/61/cntt.aspx) Hàm là một đối tượng trong cơ sở dữ liệu bao gồm một tập nhiều câu lệnh SQL được nhóm lại với nhau thành một nhóm. Điểm khác biệt giữa hàm và thủ tục là hàm trả về một giá trị thông qua tên hàm. Điều này cho phép ta sử dụng hàm như là một thành phần của một biểu thức chẳng hạn như trong các câu lệnh truy vấn hay các câu lệnh thực hiện cập nhật dữ liệu

Trong SQL có rất nhiều các hàm được định nghĩa sẵn (Được chia theo nhóm – Trong 1 Database bạn chọn Programmability/Functions/System Functions) như các hàm về chuỗi (String Functions), các hàm về ngày tháng (Date and Time Functions), Các hàm toán học (Mathematical Function), … Ngoài những hàm do hệ quản trị cơ sở dữ liệu cung cấp sẵn, bạn có thể tự xây dựng các hàm nhằm phục vụ cho mục đích riêng của mình – Các hàm do người dùng định nghĩa. Các hàm do người dùng định nghĩa thường có 2 loại: Loại 1 là **Hàm với giá trị trả về là “dữ liệu kiểu bảng”** – Table-valued Functions; Loại 2 là **Hàm với giá trị trả về là một giá trị** – Scalar-valued Functions và các hàm này cũng sẽ được Hệ quản trị phân thành 2 nhóm.

Các hàm sẵn có của SQL bạn tự tìm hiểu và sử dụng, trong bài viết này mình giới thiệu qua về những hàm “Do người dùng định nghĩa”.

Cú pháp của hàm như sau:

*CREATE FUNCTION Ten\_Ham ( [Danh\_Sach\_Cac\_Tham\_So] )   
RETURNS Kieu\_Du\_Lieu\_Tra\_Ve\_Cua\_Ham  
AS   
BEGIN  
   Cac\_Cau\_Lenh\_Cua\_Ham  
END*

*– Ten\_Ham:* Tên của hàm cần tạo. Tên phải tuân theo qui tắc định danh và không trùng với tên của các hàm hệ thống có sắn.  
*– Danh\_Sach\_Cac\_Tham\_So:* Các tham số của hàm được khai báo ngay sau tên hàm và được bao bởi cặp dấu (), Danh sách các tham số này có thể không có – trường hợp này thì sau tên hàm bạn cần có cặp dấu (). Nếu hàm có nhiều tham số thì các khai báo phân cách nhau bởi dấu phẩy và phải bao hồm 2 phần: Tên tham số được bắt đầu bởi dấu @, Kiểu dữ liệu của tham số  
*– Cac\_Cau\_Lenh\_Cua\_Ham:* Tập hợp các câu lệnh sử dụng trong nội dung hàm để thực hiện các yêu cầu của hàm.

Giờ chúng ta sẽ xem các ví dụ cụ thể để hiểu hơn về hàm nhé:  
**Ví dụ 1:** Mình sẽ lấy một ví dụ thật đơn giản là hàm không có *Danh\_Sach\_Cac\_Tham\_So –*trả về giá trị là năm hiện hành (Theo giờ hệ thống trên máy Database server):

CREATE  FUNCTION dbo.fuGetCurrYear ()   
RETURNS int  
AS   
BEGIN  
 RETURN   YEAR(getdate())  
END

Xem ví dụ trên bạn sẽ thấy nó rất đơn giản nhưng qua đây bạn cũng đã biết được việc viết hàm trong SQL như thế nào.

**Ví dụ 2:** Tiếp theo mình sẽ viết một ví dụ nữa để bạn hiểu và có thể viết cho mình các hàm tự định nghĩa:  
Ví dụ này sẽ có 2 tham sô trong *Danh\_Sach\_Cac\_Tham\_So*. Hàm sẽ trả về số ngày của tháng, năm do bạn truyền vào; Bạn biết khi lập trình với Pascal bạn đã quen với bài toán tính số ngày của thàng – Với năm nhuận thì tháng 2 có 29 ngày, các năm khác có 28 ngày. (Qua hàm này bạn cũng sẽ hiểu hơn về điều khiển IF (Xem thêm bài viết [Kỹ thuật phân trang bằng Store Procedure](http://hmweb.com.vn/279/pro/71/61/cntt.aspx) để hiểu hơn về cách sử dụng IF trong SQL) và sử dụng Case – ([Xem bài viết về Hàm Case trong SQL](http://hmweb.com.vn/182/pro/71/61/cntt.aspx)để hiểu hơn về Case)

 CREATE  FUNCTION dbo.fuDaysInMonth (

 @Thang Int,

 @Nam  Int

)

RETURNS int

AS

BEGIN

 DECLARE @Ngay Int

 IF @Thang = 2

  BEGIN

   IF ((@Nam % 4 = 0 AND @Nam % 100 <> 0)

    OR (@Nam % 400 = 0))

    SET @Ngay = 29

   ELSE

    SET @Ngay = 28

  END

 ELSE

  SELECT @Ngay =

   CASE @Thang

    WHEN 1 THEN 31

    WHEN 3 THEN 31

    WHEN 5 THEN 31

    WHEN 7 THEN 31

    WHEN 8 THEN 31

    WHEN 10 THEN 31

    WHEN 12 THEN 31

    WHEN 4 THEN 30

    WHEN 6 THEN 30

    WHEN 9 THEN 30

    WHEN 11 THEN 30

   END

 RETURN @Ngay

END

**Ví dụ 3:**Bạn xem tiếp ví dụ sau để xác định thứ trong tuần của một giá trị kiểu ngày

CREATE FUNCTION fuThu  
(  
 @ngay DATETIME  
)  
RETURNS NVARCHAR(10)  
AS  
     BEGIN  
          DECLARE @KetQua NVARCHAR(10)  
          SELECT @KetQua=CASE DATEPART(DW,@ngay)  
                        WHEN 1 THEN N’Chủ nhật’  
                        WHEN 2 THEN N’Thứ hai’  
                        WHEN 3 THEN N’Thứ ba’  
                        WHEN 4 THEN N’Thứ tư’  
                        WHEN 5 THEN N’Thứ năm’  
                        WHEN 6 THEN N’Thứ sáu’  
                        ELSE N’Thứ bảy’  
                      END     
          RETURN (@KetQua)  /\* Trị trả về của hàm \*/  
END

Một hàm khi đã được định nghĩa có thể được sử dụng như các hàm do hệ quản trị cơ sở dữ liệu cung cấp (thông thường trước tên hàm ta phải chỉ định thêm tên của người sở hữu hàm bằng dbo.) như ví dụ dưới đây:

SELECT e.FirstName, e.LastName,  
dbo.fuThu(e.BirthDate) AS ThuOfBirth   
FROM Employees e

Bạn có thể tham khảo thêm bài viết [Xử lý từ khóa tìm kiếm cho bài viết](http://hmweb.com.vn/329/pro/71/61/cntt.aspx)

Tiếp theo mình sẽ nói về **Hàm với giá trị trả về là “dữ liệu kiểu bảng”**  
Nếu đã biết về SQL chắc hẳn bạn đã biết cách tạo View từ các bảng trong CSDL, Nhưng với View bạn không thể truyền các tham số được, điều này phần nào đó làm giảm tính linh hoạt trong việc sử dụng View. Vậy nên khi bạn cần sử dụng dữ liệu dạng View mà có các tham số thì việc sử dụng hàm là một giải pháp hợp lý nhất.  
**Ví dụ 4:** Giả sử Mình tạo 1 View như sau:

 CREATE VIEW vProducts

 as

 SELECT

 Categories.CategoryID,

 Categories.CategoryName,

 Products.ProductName,

 Products.QuantityPerUnit,

 Products.UnitPrice

FROM

 Categories INNER JOIN

 Products ON Categories.CategoryID = Products.CategoryID

 WHERE Categories.CategoryID=1

Bạn xem ví dụ bạn thấy rằng mình tạo ra 1 View vProducts có  Categories.CategoryID=1 và bạn muốn truy vấn các trường của bảng Products ừng với CategoryID=1 và bạn chỉ cần câu lệnh Select \* from vProducts là bạn đã có kết quả như ý. Nhưng với những CategoryID khác thì View vProducts  không làm được trừ phi bạn. Vậy bạn thử sử dụng hàm sau để làm minh họa nhé

**Ví dụ 5:** Tạo một hàm trả về dữ liệu dạng bảng tùy theo giá trị của biến @CategoryID truyền vào:

CREATE FUNCTION fuGetProducts  
(  
 @CategoryID int  
)RETURNS TABLE  
 AS  
  RETURN  
  (  
   SELECT      
  Categories.CategoryID,  
  Categories.CategoryName,  
  Products.ProductName,  
  Products.QuantityPerUnit,  
  Products.UnitPrice  
 FROM          
  Categories INNER JOIN  
  Products ON Categories.CategoryID = Products.CategoryID  
  WHERE [Categories.CategoryID=@CategoryID)](mailto:Categories.CategoryID=@CategoryID)

chạy thử hàm trên(Chú ý là khi hàm trả về dạng bảng bạn cũng coi đó như 1 table hoặc 1 View và bạn có thể truy vấn theo 1 hay nhiều trường của hàm) như sau:

SELECT CategoryID,  
CategoryName,  
ProductName,  
QuantityPerUnit,  
UnitPrice  
FROM  dbo.fuGetProducts(1)

Sẽ tra về dữ liệu chính là Select \* from vProducts ở trên. Nếu muốn lầy Theo CategoryID=2 bạn dùng câu lệnh Select \* from dbo.fuGetProducts(2)…

Trên đây chỉ là một ví dụ nhỏ về hàm trả lại dữ liệu kiểu bảng hy vọng bạn sẽ hiểu phần nào về loại hàm này. Trong thực tế chúng ta sẽ cần nó để thực hiện các yêu cầu phức tạp hơn tùy vào dữ liệu thiết kế, quan hệ dữ liệu và yêu cầu mà bạn viết hàm để sử dụng

**2. Trigger**

Cũng tương tự như thủ tục lưu trữ ([Stored Prodedure](http://hmweb.com.vn/343/pro/71/61/cntt.aspx)), một trigger là một đối tượng chứa một tập các câu lệnh SQL và tập các câu lệnh này sẽ được thực thi khi trigger được gọi. Điểm khác biệt giữa thủ tục lưu trữ và trigger là: Các thủ tục lưu trữ được thực thi khi người sử dụng có lời gọi đến chúng còn các trigger lại được “gọi” tự động khi xảy ra những giao tác làm thay đổi dữ liệu trong các bảng.

Mỗi một trigger được tạo ra được gắn liền với một bảng nào đó trong cơ sở dữ liệu của bạn. Khi dữ liệu trong bảng bị thay đổi (Là khi xảy ra các sự kiện INSERT, UPDATE hay DELETE) thì trigger sẽ được tự đông kích hoạt. Để xem các Trigger của một bảng trong SQL 2005 bạn chọn bảng đó, chọn Triggers

Sử dụng trigger một cách hợp lý trong cơ sở dữ liệu sẽ có tác động rất lớn trong việc tăng hiệu năng của cơ sở dữ liệu. Các  trigger thực sự hữu dụng với những khả năng sau:

* Một trigger có thể nhận biết, ngăn chặn và huỷ bỏ được những thao tác làm thay đổi trái phép dữ liệu trong cơ sở dữ liệu.
* Các thao tác trên dữ liệu (xoá, cập nhật và bổ sung) có thể được trigger phát hiện ra và tự động thực hiện một loạt các thao tác khác trên cơ sở dữ liệu nhằm đảm bảo tính hợp lệ của dữ liệu.
* Thông qua trigger, ta có thể tạo và kiểm tra được những mối quan hệ phức tạp hơn giữa các bảng trong cơ sở dữ liệu mà bản thân các ràng buộc không thể thực hiện được.

Khi xảy ra ra một sự kiện thao tác dữ liệu một bản ghi trong CSDL nó sẽ lưu ra một bản ghi trong Trigger nó có tên là **inserted**đối với các thao tác Insert hay Update và **deleted**đối với Delete   
Cú pháp chung để tạo một Trigger như sau:

CREATE TRIGGER Ten\_Trigger  
ON Ten\_Bang  
FOR {[INSERT] | [UPDATE] | [DELETE]}  
AS  
BEGIN  
 Cac\_Cau\_Lenh\_Cua\_Trigger  
END

Như vậy khi tạo ra một trigger ta phải chỉ rõ là tạo ra trigger trên table nào và được trigger khi nào (insert, update hay delete. Sau chữ AS là các câu lệnh SQL xử lý công việc, có thể dùng cặp Begin … End hoặc không). Bạn có thể tham khảo thêm bài viết về Trigger [Xây dựng cơ sở dữ liệu – TRIGGER](http://hmweb.com.vn/143/pro/71/61/cntt.aspx)

Giờ ta sẽ tìm hiều ví dụ để hiều hơn về Trigger nhé.  
Vẫn với CSDL Northwind giả sử trong bảng Employees mình thêm 1 trường là Age – là tuổi của Employees. Giờ mình sẽ viết 1 trigger gắn với bảng Employees để khi thay đổi BirthDay thì trường Age sẽ tự động được cập nhật. Bạn xem ví dụ sau.

CREATE TRIGGER trigCalcAge  
   ON Employees  
  FOR  UPDATE, Insert  
AS  
BEGIN  
 DECLARE @age int  
 DECLARE @EmployeeID int  
 SELECT  
  @age=YEAR(GETDATE())-year(BirthDate),  
  @EmployeeID=EmployeeID  
 FROM inserted  
 IF UPDATE (BirthDate)  
 UPDATE Employees  
  SET Age = @age  
 WHERE [EmployeeID=@EmployeeID](mailto:EmployeeID=@EmployeeID)  
END

Xem ví dụ trên bạn thấy khi có thay đổi nó sẽ tạo 1 bản ghi inserted và chúng ta có thể lấy các giá trị của bản ghi đó. Khi thay đổi dữ liệu bạn sẽ không cần cập nhật trường Age.  
Tương tự như vậy với trường hợp Xóa dữ liệu.  
Một điều chú ý là với Trigger nó chỉ thực hiện với sự thay đổi dữ liệu của từng bản ghi. Với trường hợp cập nhật dữ liệu theo bó thì khi đó bạn cần các kỹ thuật xử lý phức tạp hơn.  
Ví dụ câu lệnh sau: Update Employees Set BirthDate=’12/12/1990 12:00:00 AM’ Câu lệnh này sẽ update toàn bộ dữ liệu (Cập nhật theo bó) của bảng Employees  Nhưng khi đó Trigger của ta chỉ update trường Age ở bản ghi đầu tiên. Để xử lý trường hợp này có nhiều cách chẳng hạn như dùng vòng lặp (dùng con trỏ).