## CHƯƠNG 8. TRIGGER AND USER DEFINED FUNCTION (UDF)

**A. TRIGGER.**

Trigger là 1 loại Stored Procedures đặc biệt được thực hiện 1 cách tự động khi user thực hiện việc cập nhật (insert, update, delete) dữ liệu trên table. Trigger nhằm mục đích đảm bảo sự an toàn về ràng buộc toàn vẹn dữ liệu. Mỗi table có thể có nhiều trigger tương ứng với các hành động insert, delete, update trên table.

Ta gõ vào tên trigger thay cho TRIGGER NAME, và gõ vào các câu lệnh sau từ khóa

AS.

* Ěể kiểm tra cú pháp của các lệnh trong Trigger, ta click nút lệnh **Check Syntax**
* Ěể xóa trigger, ta chọn tên trigger, sau đó click nút **Delete**

## CÚ PHÁP:

CREATE TRIGGER *trigger\_name* ON { *table* | view}

{FOR BEFORE | AFTER | INSTEAD OF }

{ [DELETE] [**,**] [INSERT] [**,**] [UPDATE] } [NOT FOR REPLICATION]

AS

*sql\_statement* [...*n*]

}

hoặc là

{FOR { [INSERT] [**,**] [UPDATE] } *-- không có DELETE nếu có UPDATE(column)*

[NOT FOR REPLICATION] AS

{ IF UPDATE **(***column***)**

[{AND | OR} UPDATE **(***column***)**] [...*n*]

| IF (COLUMNS\_UPDATED**()** {*bitwise\_operator*} *updated\_bitmask*)

{ *comparison\_operator*} *column\_bitmask* [...*n*]

}

*sql\_statement* [ ...*n*]

}

}

## Các tham số:

* *trigger\_name* : là tên của trigger. Tên trigger phải tuân thủ quy tắc như danh hiệu, và là duy nhất trong cơ sở dữ liệu.
* *table* , view : là tên của table hoặc view mà trên đó trigger được thực hiện .
* BEFORE : Trigger sẽ hoạt động trước khi thao tác lệnh xảy ra
* FOR, AFTER : Trigger sẽ hoạt động sau khi thao tác lệnh xảy ra
* INSTEAD OF : Trigger sẽ được thực thi thay thế cho cho các thao tác INSERT,

UPDATE hoặc DELETE

* { [DELETE] [**,**] [INSERT] [**,**] [UPDATE] } | { [INSERT] [**,**] [UPDATE]} : là các từ

khóa cho biết trigger sẽ tự động hoạt động theo lệnh nào.

* NOT FOR REPLICATION : trigger sẽ không hoạt động khi tiến trình nhân bản có thay đổi dữ liệu trên table có trigger.

\* Một số table đặc biệt được dùng trong lệnh CREATE TRIGGER: **deleted** and **inserted** là các table logic. Chúng có cấu trúc tương tự như table mà trigger đang hoạt động, và các table này lưu giữ giá trị cǜ hay giá trị mới của các mẫu tin đã được thay đổi bởi hành động của user. Lệnh Update được xem như thao tác xóa dữ liệu cǜ, và tạo dữ liệu mới nên bảng deleted chứa dữ liệu trước khi update, và bảng inserted chứa dữ liệu mới.

* IF UPDATE **(***column***)** : kiểm tra cột nào đang chịu sự tác động của lệnh INSERT hoặc UPDATE, UPDATE(column) không được sử dụng với lệnh DELETE. Ta có thể kiểm tra nhiều cột
* IF (COLUMNS\_UPDATED**()) :** kiểm tra cột nào đang chịu sự tác động của lệnh INSERT hoặc UPDATE. COLUMNS\_UPDATED() trả về 1 trị kiểu varbinary để cho ta biết các cột nào trong table đã được chèn hay được hiệu chỉnh dữ liệu.

## Ví dụ: Dùng trigger để nhắc nhở:

Ví dụ này sẽ in 1 thông báo đến client khi có 1 ai đó đang thêm hay thay đổi dữ liệu

trong table Nhanvien.

USE QLVT

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

DROP TRIGGER reminder GO

CREATE TRIGGER reminder ON NhanVien AFTER INSERT, UPDATE

AS RAISERROR (‘Khong duoc them nhan vien moi hoac hieu chinh’, 16, 10) GO

## TRIGGER KIỂM TRA CÁC THAO TÁC CẬP NHẬT DỮ LIỆU:

* 1. **Trigger kiểm tra thao tác thêm mẫu tin** :Trigger loại này dùng để kiểm tra mẫu tin thêm vào phải tuân thủ các ràng buộc về khoá chính, và khóa ngoại.

Ví dụ: Tạo trigger Test\_ThemSV để kiểm tra khi ta thêm 1 sinh viên mới thì mã lớp của sinh viên đó phải có trước trong table Lop. Nếu mã lớp này chưa có trong table Lop, thì báo lỗi và bỏ qua việc thêm sinh viên đó.

CREATE TRIGGER Test\_ThemSV ON dbo.Sinhvien

## FOR INSERT

AS

Declare @Loi int=1

If exists( Select \* from Lop , inserted

where Lop.malop=inserted.malop) Set @Loi=0

If @Loi=1

raiserror( 'Khong the them sinh vien vi ma lop chua ton tai ben table LOP’,16,1)

* 1. **Trigger cho thao tác xóa mẫu tin:** Trigger loại này thường được dùng để đảm bảo ràng buộc toàn vẹn. Ví dụ khi ta xóa 1 vật tư trong trong chi tiết phiếu nhập thì giảm số lượng tồn trong bảng VATTU

ALTER TRIGGER [dbo].[XOA\_CTPN] ON [dbo].[CTPN] AFTER DELETE

AS BEGIN

SET NOCOUNT ON; UPDATE VATTU

SET SOLUONGTON= SOLUONGTON - (SELECT SOLUONG FROM deleted) WHERE MAVT = (SELECT MAVT FROM deleted)

END

* 1. **Trigger cho thao tác hiệu chỉnh dữ liệu:** Ví dụ ta viết TRIGGER để cập nhật số lượng tồn trong bảng VATTU khi ta thay đổi field số lượng của 1 vật tư nhập trong bảng CTPN.

CREATE TRIGGER TR\_AfterUpdate\_SOLUONG\_CTPN ON CTPN AFTER UPDATE

AS BEGIN

IF (UPDATE(SOLUONG)) BEGIN

UPDATE VATTU

SET SOLUONGTON= SOLUONGTON -(SELECT SOLUONG FROM deleted) +

(SELECT SOLUONG FROM inserted) WHERE MAVT = (SELECT MAVT FROM inserted)

END

-- Trường hợp hiệu chỉnh field ??? sẽ ảnh hưởng tới số lượng tồn

END

Bài tập: TƯƠNG TỰ CHO CÁC THAO TÁC INSERT, UPDATE, DELETE TREN TABLE CTPX

**B.** USER DEFINED FUNCTION **(HAØM DO NGÖÔØI DUØNG ÑÒNH NGHÓA)**

## KHAÙI NIEÄM:

Những hàm trong ngôn ngữ lập trình là những chương trình con được dùng để đóng gói những đoạn lệnh thực hiện một nhiệm vụ.

Microsoft SQL Server hỗ trợ 2 loại hàm:

* + Những hàm được cài đặt sẵn (Built-in Functions): được định nghƿa sẵn trong Transact-SQL Reference và không thể bổ sung. Những hàm này có thể được tham chiếu trong những phát biểu Transact-SQL sử dụng cú pháp được định nghƿa trong Transact-SQL Reference.
  + Những hàm người dùng định nghƿa (User-defined Functions): cho phép định nghƿa những hàm Transact-SQL của chính ta qua phát biểu CREATE FUNCTION.

Tên của “hàm người dùng định nghƿa” (database\_name.owner\_name.function\_name)

không được trùng nhau; Tên của UDF được lưu trong table Sys.sysobjects với xtype

=’FN’

* Ta phải được cấp quyền trong CREATE FUNCTION để được phép hiệu chỉnh, xoá UDF. Người dùng (ngoài người tạo) phải được cấp quyền hợp lệ trên hàm mới có thể dùng hàm đó trong câu lệnh SQL.
* Ěể tạo hay hiệu chỉnh UDF có tham chiếu đến Tables trong ràng buộc CHECK, mệnh đề DEFAUL, hay định nghƿa 1 calculated column... ta phải dùng quyền REFERENCES trong hàm.
* Trong Trigger hay Stored Procedure, nếu câu lệnh nào đó bị lỗi thì câu lệnh tiếp theo trong cùng module sẽ được thực hiện (theo mặc định). Nhưng trong hàm, lỗi sẽ làm dừng hàm, và làm cho câu lệnh gọi hàm bị huỷ bỏ.
* UDF không nhận tham số hoặc nhận nhiều tham số (tối đa 1024 tham số) và trả về một trị vô hướng hoặc một bảng. Khi tham số đầu vào của hàm có giá trị mặc định, ta phải dùng từ khoá DEFAULT khi gọi hàm (khác với tham số có giá trị mặc định trong Stored Procedure sẽ bị bỏ đi). UDF không sử dụng tham biến
* UDF trả về bảng có thể thay cho View. Một UDF trả về bảng cǜng được dùng thay cho bảng hay View trong truy vấn SQL. View bị giới hạn với một câu lệnh SELECT, nhưng UDF có thể chứa thêm các câu lệnh hiệu quả hơn View.
* UDF có 2 loại: Scalar trả về 1 giá trị đơn, hoặc Table-valued trả về 1 tập kết quả dưới dạng bảng

## UDF trả về một tập kết quả có thể thay cho các Stored Procedure trả về một tập kết quả. Bảng trả về có thể được tham chiếu trong mệnh đề FROM của câu truy vấn SQL, nhưng điều này không thể thực hiện được đối với Stored Procedure.

**II . Cú pháp:**

**CREATE FUNCTION Tên\_Hàm**

**( @parameter type (=default) [, @parameter …] )**

**RETURNS {scalar\_type | TABLE] AS {block | RETURN (select\_stmt)}**

* **@parameter type:** tham số gởi cho hàm để xử lý
* **RETURNS {scalar\_type | TABLE:** chỉ ra hàm trả về 1 giá trị đơn hay 1 tập kết quả dưới dạng table
* **Block:** khối lệnh BEGIN … END chứa cài đặt của hàm. Khối lệnh phải có lệnh return trả về kết quả. Trong khối lệnh có thể có các lệnh:
  + Lệnh DECLARE để khai báo biến và con trỏ cục bộ.
  + Lệnh SET dùng để gán giá trị cho cho các biến và các biến table ở dạng cục bộ.
  + Thao tác con trỏ tham chiếu đến con trỏ cục bộ được khai báo, mở, đóng, và cấp lại trong hàm. Lệnh FETCH gán giá trị cho biến cục bộ dùng mệnh đề INTO.
  + Cấu trúc While, If
  + Lệnh SELECT để gán giá trị cho các biến cục bộ.
  + Lệnh UPDATE, INSERT, và DELETE tác động lên biến table cục bộ.

**Nhóm quyền được tạo hàm:** db\_owner, db\_ddladmin

**Ví dụ, Hàm CubicVolume đơn giản trả về một giá trị decimal:**

## CREATE FUNCTION CubicVolume

(@CubeLength decimal(4,1), @CubeWidth decimal(4,1),

@CubeHeight decimal(4,1) )

## RETURNS decimal(12,3)

AS BEGIN

RETURN ( @CubeLength \* @CubeWidth \* @CubeHeight ) END

Hàm này có thể dùng ở những nơi cho phép sử dụng một biểu thức số, như trong một cột tính toán của một Table:

CREATE TABLE Bricks (

BrickPartNmbr int PRIMARY KEY, BrickColor nchar(20), BrickHeight decimal(4,1), BrickLength decimal(4,1), BrickWidth decimal(4,1),

BrickVolume AS

(

dbo.CubicVolume(BrickHeight,

BrickLength, BrickWidth)

)

)

**CREATE FUNCTION [dbo].[FN\_CHUANHOA]**

(

@S NVARCHAR(100) -- Add the parameters for the function here

)

**RETURNS NVARCHAR(100)** -- KIEU TRA VE AS

BEGIN

SET @S= LTRIM(RTRIM(@S))

WHILE PATINDEX('% %', @S) <> 0 -- tìm 2 khoảng trắng

BEGIN

SET @S=STUFF(@S,PATINDEX('% %', @S), 1,'') END

**RETURN @S**

END

SQL Server cǜng hỗ trợ hàm UDF trả về kiểu dữ liệu table:

- Hàm có thể khai báo một tham số table, chèn thêm những hàng vào tham số và sau đó trả về dữ liệu trong tham số table (multistatement table-valued function).

- Hàm do user định nghƿa được biết đến như những hàm inline, trả về tập kết quả qua

phát biểu SELECT

Những hàm này có thể được sử dụng ở những vị trí mà biểu thức Table có thể được chỉ định.

Một hàm UDF mà trả về một table có thể dùng ở những nơi mà biểu thức Table hoặc View được phép trong những câu truy vấn Transact-SQL.

Ví dụ hàm fn\_EmployeesInDept là một hàm UDF trả về một table và có thể đươc gọi bởi phát biểu SELECT:

SELECT \*

FROM tb\_Employees AS E,

**dbo.fn\_EmployeesInDept('shipping')** AS EID WHERE E.EmployeeID = EID.EmployeeID

Ví dụ về một hàm trong cơ sở dữ liệu QLDSV trả về một table:

# CREATE FUNCTION FN\_DSSVLOP (@MALOP NVARCHAR (20))

RETURNS TABLE -- Inline Function AS

RETURN

SELECT MASV, HO, TEN FROM SINHVIEN WHERE MALOP=@MALOP

Lệnh gọi hàm: SELECT \* FROM FN\_DSSVLOP ('D08-HTTT')

Dưới đây là hàm có giá trị trả về là table @mytable; Trong hàm, ta insert các dòng vào

@mytable, và cuối cùng trả về kết quả lưu trong @mytable.

ALTER FUNCTION [dbo].[FN\_TEST] (@NIENKHOA NVARCHAR(11), @GV\_COHUU BIT)

RETURNS @mytable TABLE –- multistatement table-valued function (MA\_GV NCHAR(15), HO NVARCHAR(100), TEN NVARCHAR(10))

AS BEGIN

IF (@GV\_COHUU = 'TRUE')

INSERT INTO @mytable( MA\_GV, HO, TEN )

SELECT MA\_GV, HO, TEN FROM GIANG\_VIEN\_CO\_HUU WHERE NIENKHOA=@NIENKHOA

ELSE

INSERT INTO @mytable( MA\_GV, HO, TEN )

SELECT MA\_GV, HO, TEN FROM GIANG\_VIEN\_THINH\_GIANG WHERE NIENKHOA=@NIENKHOA

RETURN END

## Date and Time Functions

These scalar functions perform an operation on a date and time input value and return a string, numeric, or date and time value.

This table lists the date and time functions and their determinism property.

|  |  |
| --- | --- |
| Function | Determinism |
| DATEADD | Deterministic |
| DATEDIFF | Deterministic |
| DATEPART | 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. |
| DAY | Deterministic |
| GETDATE | Nondeterministic |
| GETUTCDATE | Nondeterministic |
| MONTH | Deterministic |
| YEAR | Deterministic |

Returns a new datetime value based on adding an interval to the specified date.

Syntax

DATEADD ( *datepart* , *number*, *date* )

Arguments

*datepart*

Is the parameter that specifies on which part of the date to return a new value. The table lists the dateparts and abbreviations recognized by Microsoft® SQL Server™.

|  |  |
| --- | --- |
| Datepart | Abbreviations |
| Year | yy, yyyy |
| quarter | qq, q |
| Month | mm, m |
| dayofyear | dy, y |
| Day | dd, d |
| Week | wk, ww |

|  |  |  |
| --- | --- | --- |
| Hour | hh |  |
| minute | mi, n |
| second | ss, s |
| millisecond | ms |

*number*

Is the value used to increment *datepart*. If you specify a value that is not an integer, the fractional part of the value is discarded. For example, if you specify day for *datepart* and1.75 for *number*, *date* is incremented by 1.

*date*

Is an expression that returns a datetime or smalldatetime value, or a character string in a date format. For more information about specifying dates, see datetime and smalldatetime.

If you specify only the last two digits of the year, values less than or equal to the last two digits of the value of the two digit year cutoff configuration option are in the same century as the cutoff year. Values greater than the last two digits of the value of this option are in the century that precedes the cutoff year. For example, if two digit year cutoff is 2049 (default), 49 is interpreted as 2049 and 2050 is interpreted as 1950. To avoid ambiguity, use four-digit years.

Return Types

Returns datetime, but smalldatetime if the *date* argument is smalldatetime.

## DATEDIFF

Returns the number of date and time boundaries crossed between two specified dates.

Syntax

DATEDIFF ( *datepart* , *startdate* , *enddate* )

Arguments

*datepart*

Is the parameter that specifies on which part of the date to calculate the difference. The table lists dateparts and abbreviations recognized by Microsoft® SQL Server™.

|  |  |
| --- | --- |
| Datepart | Abbreviations |
| Year | yy, yyyy |
| quarter | qq, q |
| Month | mm, m |

|  |  |  |
| --- | --- | --- |
| dayofyear | dy, y |  |
| Day | dd, d |
| Week | wk, ww |
| Hour | hh |
| minute | mi, n |
| second | ss, s |
| millisecond | ms |

*startdate*

Is the beginning date for the calculation. *startdate* is an expression that returns a datetime or smalldatetime value, or a character string in a date format.

Because smalldatetime is accurate only to the minute, when a smalldatetime value is used, seconds and milliseconds are always 0.

If you specify only the last two digits of the year, values less than or equal to the last two digits of the value of the two digit year cutoff configuration option are in the same century as the cutoff year. Values greater than the last two digits of the value of this option are in the century that precedes the cutoff year. For example, if the two digit year cutoff is 2049 (default), 49 is interpreted as 2049 and 2050 is interpreted as 1950. To avoid ambiguity, use four-digit years.

For more information about specifying time values, see Time Formats. For more information about specifying dates, see datetime and smalldatetime.

*enddate*

Is the ending date for the calculation. *enddate* is an expression that returns a datetime or smalldatetime value, or a character string in a date format.

Return Types

integer

Remarks

*startdate* is subtracted from *enddate*. If *startdate* is later than *enddate*, a negative value is returned.

DATEDIFF produces an error if the result is out of range for integer values. For milliseconds, the maximum number is 24 days, 20 hours, 31 minutes and 23.647 seconds. For seconds, the maximum number is 68 years.

The method of counting crossed boundaries such as minutes, seconds, and milliseconds makes the result given by DATEDIFF consistent across all data types. The result is a signed integer value equal to the number of *datepart* boundaries crossed between the first and second

date. For example, the number of weeks between Sunday, January 4, and Sunday, January 11, is 1.

## DATEPART

Returns an integer representing the specified datepart of the specified date.

Syntax

DATEPART ( *datepart* , *date* )

Arguments

*datepart*

Is the parameter that specifies the part of the date to return. The table lists dateparts and abbreviations recognized by Microsoft® SQL Server™.

|  |  |
| --- | --- |
| Datepart | Abbreviations |
| year | yy, yyyy |
| quarter | qq, q |
| month | mm, m |
| dayofyear | dy, y |
| day | dd, d |
| week | wk, ww |
| weekday | dw |
| hour | hh |
| minute | mi, n |
| second | ss, s |
| millisecond | ms |

The week (wk, ww) datepart reflects changes made to SET DATEFIRST. January 1 of any year defines the starting number for the week datepart, for example: DATEPART(wk, 'Jan 1, xxxx') = 1, where xxxx is any year.

The weekday (dw) datepart returns a number that corresponds to the day of the week, for example: Sunday = 1, Saturday = 7. The number produced by the weekday datepart depends on the value set by SET DATEFIRST, which sets the first day of the week.

*date*

Is an expression that returns a datetime or smalldatetime value, or a character string in a date format. Use the datetime data type only for dates after January 1, 1753. Store dates as character data for earlier dates. When entering datetime values, always enclose them in quotation marks. Because smalldatetime is accurate only to the minute, when a smalldatetime value is used, seconds and milliseconds are always 0.

If you specify only the last two digits of the year, values less than or equal to the last two digits of the value of the two digit year cutoff configuration option are in the same century as the cutoff year. Values greater than the last two digits of the value of this option are in the century that precedes the cutoff year. For example, if two digit year cutoff is 2049 (default), 49 is interpreted as 2049 and 2050 is interpreted as 1950. To avoid ambiguity, use four-digit years.

For more information about specifying time values, see Time Formats. For more information about specifying dates, see datetime and smalldatetime.

Return Types

int

Remarks

The DAY, MONTH, and YEAR functions are synonyms for DATEPART(dd, *date*), DATEPART(mm, *date*), and DATEPART(yy, *date*), respectively.

## GETDATE

Returns the current system date and time in the Microsoft® SQL Server™ standard internal format for datetime values.

Syntax

GETDATE ( )

Return Types

datetime

Remarks

Date functions can be used in the SELECT statement select list or in the WHERE clause of a query.

In designing a report, GETDATE can be used to print the current date and time every time the report is produced. GETDATE is also useful for tracking activity, such as logging the time a transaction occurred on an account.

Examples

## Mathematical Functions

These scalar functions perform a calculation, usually based on input values provided as arguments, and return a numeric value.

|  |  |  |
| --- | --- | --- |
| ABS | DEGREES | RAND |
| ACOS | EXP | ROUND |
| ASIN | FLOOR | SIGN |
| ATAN | LOG | SIN |
| ATN2 | LOG10 | SQUARE |
| CEILING | PI | SQRT |
| COS | POWER | TAN |
| COT | RADIANS |  |

Note Arithmetic functions, such as ABS, CEILING, DEGREES, FLOOR, POWER, RADIANS, and SIGN, return a value having the same data type as the input value. Trigonometric and other functions, including EXP, LOG, LOG10, SQUARE, and SQRT, cast their input values to float and return a float value.

## String Functions

These scalar functions perform an operation on a string input value and return a string or numeric value.

|  |  |  |
| --- | --- | --- |
| ASCII | NCHAR | SOUNDEX |
| CHAR | PATINDEX | SPACE |
| CHARINDEX | REPLACE | STR |
| DIFFERENCE | QUOTENAME | STUFF |
| LEFT | REPLICATE | SUBSTRING |
| LEN | REVERSE | UNICODE |
| LOWER | RIGHT | UPPER |
| LTRIM | RTRIM |  |

All built-in string functions, except for CHARINDEX and PATINDEX, are deterministic. They return the same value any time they are called with a given set of input values.