



## **Chương 3: LẬP TRÌNH TRÊN SQL SERVER**

Email: [vienthanhnha@tlu.edu.vn](mailto:vienthanhnha@tlu.edu.vn)  
Ths. Viên Thành Nhã

# Giới thiệu ngôn ngữ T-SQL

## 3.1 Giới thiệu ngôn ngữ T-SQL:

### 3.1.1 Khái niệm:

Transaction SQL (T-SQL) là ngôn ngữ phát triển nâng cao của ngôn ngữ SQL chuẩn. Nó là ngôn ngữ dùng để giao tiếp giữa ứng dụng và SQL Server. T-SQL có các khả năng của ngôn ngữ định nghĩa dữ liệu - DDL và ngôn ngữ thao tác dữ liệu – DML của SQL chuẩn cộng với một số hàm mở rộng, các store procedure hệ thống và cấu trúc lập trình (như IF, WHILE,...) cho phép lập trình trên SQL Server được linh động hơn.

Trong các chương trước ta đã giới thiệu ngôn ngữ SQL chuẩn và làm quen với các câu lệnh T-SQL dùng để định nghĩa dữ liệu, thao tác dữ liệu như: Tạo CSDL, tạo bảng, tạo View, tạo Index, chèn dữ liệu,.v.v... Trong chương này ta sẽ tìm hiểu thêm về T-SQL.

# Giới thiệu ngôn ngữ T-SQL

## 3.1.2 Phát biểu truy vấn dữ liệu nâng cao:

### a) Mệnh đề TOP:

Mệnh đề TOP chỉ định tập hợp các dòng đầu tiên được trả về trong truy vấn. Tập hợp các dòng đó có thể là một con số hoặc theo tỷ lệ phần trăm (PERCENT) các dòng dữ liệu. Mệnh đề TOP được sử dụng trong các khối câu lệnh Select, Insert, Update và Delete. Cú pháp:

```
[     TOP (expression) [PERCENT]
      [ WITH TIES ]
]
```

Trong đó:

expression: Là biểu thức trả về giá trị kiểu số

PERCENT: Chỉ định số dòng trả về là expression phần trăm trong tập kết quả.

WITH TIES: TOP ...WITH TIES chỉ được chỉ định trên khối câu lệnh SELECT và có mệnh đề ORDER BY. Chỉ định thêm các dòng từ tập kết quả cơ sở có cùng giá trị với các cột trong mệnh đề ORDER BY xuất hiện như là dòng cuối cùng của TOP n (PERCENT).

# Giới thiệu ngôn ngữ T-SQL

Ví dụ: Sử dụng mệnh đề TOP

- Trong câu lệnh Insert

```
INSERT TOP (2) INTO LOP  
    SELECT * FROM DMLOP ORDER BY Khoa
```

- Trong câu lệnh Select

```
INSERT INTO LOP  
    SELECT TOP (2) WITH TIES * FROM DMLOP  
        ORDER BY Khoa
```

# Giới thiệu ngôn ngữ T-SQL

b) Điều kiện kết nối - JOIN Trong khối câu lệnh SELECT, ở mệnh đề FROM ta có thể sử dụng phát biểu JOIN để kết nối các bảng có quan hệ với nhau.

Mệnh đề kết nối Join được phân loại như sau:

- Inner joins (toán tử thường dùng để kết nối thường là các toán tử so sánh = hoặc <>). Inner joins sử dụng một toán tử so sánh để so khớp các dòng từ hai bảng dựa trên các giá trị của các cột so khớp của mỗi bảng. Kết quả trả về của Inner Join là các dòng thỏa mãn điều kiện so khớp.
- Outer joins. Outer joins có thể là left, right, hoặc full outer join.
  - + LEFT JOIN hoặc LEFT OUTER JOIN : Kết quả của left outer join không chỉ bao gồm các dòng thỏa mãn điều kiện so khớp giữa hai bảng mà còn gồm tất cả các dòng của bảng bên trái trong mệnh đề LEFT OUTER. Khi một dòng ở bảng bên trái không có dòng nào của bảng bên phải so khớp đúng thì các giá trị NULL được trả về cho tất cả các cột ở bảng bên phải.
  - + RIGHT JOIN or RIGHT OUTER JOIN: Right outer join là nghịch đảo của left outer join. Tất cả các dòng của bảng bên phải được trả về. Các giá trị Null cho bảng bên trái khi bất cứ một dòng nào bên phải không có một dòng nào bảng bên trái so khớp đúng
  - + FULL JOIN or FULL OUTER JOIN: full outer join trả về tất cả các dòng trong cả hai bảng bên trái và phải. Bất kỳ một dòng không có dòng so khớp đúng của bảng còn lại thì bảng còn lại nhận các giá trị NULL. Khi có sự so khớp đúng giữa các bảng thì tập kết quả sẽ chứa dữ liệu các bảng cơ sở đó.

# Giới thiệu ngôn ngữ T-SQL

- Cross joins: Trả về tất cả các dòng của bảng bên trái và mỗi dòng bên trái sẽ kết hợp với tất cả các dòng của bảng bên phải. Cross joins còn được gọi là tích Đề các (Cartesian products).

Ví dụ: Sử dụng Join

- Inner Joins:

```
SELECT MONHOC.MaMH, MONHOC.TenMH, MONHOC.SDVHT,
       DIEM.MaSV, DIEM.DiemL1
  FROM DIEM INNER JOIN MONHOC
    ON DIEM.MaMH = MONHOC.MaMH
```

- Left Joins:

```
SELECT MONHOC.MaMH, MONHOC.TenMH, MONHOC.SDVHT,
       DIEM.MaSV, DIEM.DiemL1
  FROM MONHOC LEFT JOIN DIEM
    ON MONHOC.MaMH= DIEM.MaMH
```

- Right Joins:

```
SELECT MONHOC.MaMH, MONHOC.TenMH, MONHOC.SDVHT,
       DIEM.MaSV, DIEM.DiemL1
  FROM DIEM Right JOIN MONHOC
    ON DIEM.MaMH= MONHOC.MaMH
```

- Full Joins:

```
SELECT MONHOC.MaMH, MONHOC.TenMH, MONHOC.SDVHT,
       DIEM.MaSV, DIEM.DiemL1
  FROM DIEM Full JOIN MONHOC
    ON DIEM.MaMH= MONHOC.MaMH
```

- Cross Joins:

```
SELECT MONHOC.MaMH, MONHOC.TenMH, MONHOC.SDVHT,
       DIEM.MaSV, DIEM.DiemL1
  FROM MONHOC CROSS JOIN DIEM
```

# Giới thiệu ngôn ngữ T-SQL

## c) Truy vấn Cross tab:

Trong một số trường hợp thống kê, ta cần phải xoay bảng kết quả, do đó có các cột được biểu diễn theo chiều ngang và các dòng được biểu diễn theo chiều dọc (được gọi là truy vấn cross tab).

Ví dụ ta có một view tính tổng giá trị của một hóa đơn View\_Order (OrderID, OrderDate, Month, Year, Total). Ta cần thống kê doanh thu theo từng tháng của các năm.

```
SELECT Year,
       SUM(CASE Month WHEN 1 THEN Total ELSE 0 END) AS Jan,
       SUM(CASE Month WHEN 2 THEN Total ELSE 0 END) AS feb,
       SUM(CASE Month WHEN 3 THEN Total ELSE 0 END) AS mar,
       SUM(CASE Month WHEN 4 THEN Total ELSE 0 END) AS apr,
       SUM(CASE Month WHEN 5 THEN Total ELSE 0 END) AS may,
       SUM(CASE Month WHEN 6 THEN Total ELSE 0 END) AS jun,
       SUM(CASE Month WHEN 7 THEN Total ELSE 0 END) AS jul,
       SUM(CASE Month WHEN 8 THEN Total ELSE 0 END) AS aug,
       SUM(CASE Month WHEN 9 THEN Total ELSE 0 END) AS sep,
       SUM(CASE Month WHEN 10 THEN Total ELSE 0 END) AS oct,
       SUM(CASE Month WHEN 11 THEN Total ELSE 0 END) AS nov,
       SUM(CASE Month WHEN 12 THEN Total ELSE 0 END) AS dec
  FROM View_Order
 GROUP BY Year
```

Kết quả:

	Year	Jan	feb	mar	apr	may	jun	jul	aug	sep	oct	nov	dec
1	2007	522.00	779.40	3488.40	180.00	7234.20	1323.00	190.00	1245.00	70338.5999	80720.9999	661.50	5678.00
2	2008	832.50	2341.00	1234.00	1246.00	890.00	3738.50	392.00	540.00	1503.00	0.00	0.00	0.00

# Giới thiệu ngôn ngữ T-SQL

- Sử dụng toán tử PIVOT và UNPIVOT:

SQL Server 2005 đưa ra các toán tử đơn giản hơn cho việc tạo truy vấn cross tab, đó là toán tử PIVOT và UNPIVOT trong mệnh đề FROM của khối câu lệnh SELECT.

+ Toán tử PIVOT thực hiện xoay một biểu thức giá trị bảng (table valued expression) thành một bảng khác bằng việc đưa các giá trị duy nhất của một cột thành các cột và thực hiện các hàm thống kê trên các cột còn lại.

+ Toán tử UNPIVOT thực hiện quá trình ngược lại với quá trình thực hiện của toán tử PIVOT, xoay các cột của biểu thức bảng thành giá trị của một cột.

Cú pháp:

```
FROM { <table_source> } [ ,...n ]
<table_source> ::= 
{
    <pivoted_table>
    | <unpivoted_table> [ ,...n ]
}

<pivoted_table> ::=
    table_source PIVOT <pivot_clause> table_alias

<pivot_clause> ::=
    ( aggregate_function( value_column )
      FOR pivot_column
      IN ( <column_list> )
    )

<unpivoted_table> ::=
    table_source UNPIVOT <unpivot_clause>
table_alias

<unpivot_clause> ::=
    ( value_column FOR pivot_column IN (
<column_list> ) )

<column_list> ::=
    column_name [ , ... ]
```

# Giới thiệu ngôn ngữ T-SQL

- Trong đó:
  - + table\_source PIVOT : Chỉ định bảng table\_source được xoay dựa trên cột pivot\_column. table\_source là một bảng hoặc biểu thức bảng. Output là một bảng chứa tất cả các cột của table\_source trừ cột pivot\_column và value\_column. Các cột của table\_source, trừ pivot\_column và value\_column, được gọi là các cột phân nhóm của toán tử pivot.
  - + aggregate\_function: Là một hàm thống kê của hệ thống hoặc do người dùng định nghĩa. Hàm COUNT(\*) không được phép sử dụng trong trường hợp này
  - + value\_column: Là cột giá trị của toán tử PIVOT. Khi sử dụng với toán tử UNPIVOT, value\_column không được trùng tên với các cột trong bảng input table\_source.
  - + FOR pivot\_column : Chỉ định trục xoay của toán tử PIVOT. pivot\_column là có kiểu chuyển đổi được sang nvarchar(). Không được là các kiểu image hoặc rowversion. Khi UNPIVOT được sử dụng, pivot\_column là tên của cột output được thu hẹp lại từ table\_source. Tên cột này không được trùng với một tên nào trong table\_source.
  - + IN ( column\_list ) : Trong mệnh đề PIVOT, danh sách các giá trị trong pivot\_column sẽ trở thành tên các cột trong bảng output. Danh sách này không được trùng với bất kỳ tên cột nào tồn tại trong bảng input table\_source mà đang được xoay. Trong mệnh đề UNPIVOT, danh sách các cột trong table\_source sẽ được thu hẹp lại thành một cột pivot\_column.
  - + table\_alias: Là tên bí danh của bảng output. pivot\_table\_alias phải được chỉ định.
  - + UNPIVOT < unpivot\_clause > : Chỉ định bảng input được thu hẹp bằng các cột trong column\_list trở thành một cột gọi là pivot\_column

# Giới thiệu ngôn ngữ T-SQL

\*Hoạt động của toán tử PIVOT:

Toán tử PIVOT thực hiện theo tiến trình sau:

- + Thực hiện GROUP BY dựa vào các cột phân nhóm trên bảng input\_table và kết quả là ứng với mỗi nhóm cho một dòng output trên bảng kết quả.
- + Sinh các giá trị ứng với các cột trong danh sách column list cho mỗi dòng output bằng việc thực thi như sau:
  - Nhóm các dòng được sinh từ việc GROUP BY ở bước trước dựa trên cột pivot\_column.

Đối với mỗi cột output trong *column\_list*, chọn một nhóm con thỏa mãn điều kiện:

```
pivot_column=CONVERT(<data type of  
pivot_column>, 'output_column')
```

- aggregate\_function định giá trị dựa tên cột value\_column trong nhóm con này và kết quả được trả về của nó tương ứng là giá trị của cột output\_column. Nếu nhóm con là rỗng thì SQL Server sinh giá trị NULL cho cột output\_column đó. Nếu hàm thống kê là COUNT thì nó sinh giá trị 0.

# Giới thiệu ngôn ngữ T-SQL

Ví dụ ta có một view tính tổng giá trị của một hóa đơn View\_Order (OrderID, OrderDate, Month, Year, Total). Ta cần thống kê doanh thu theo từng tháng của các năm.

```
SELECT Year,[1]AS Jan,[2]AS feb, [3]AS mar,[4] AS apr,[5]
AS may,[6] AS jun,[7] AS jul,[8] AS aug,[9] AS sep,
[10]AS oct,[11] AS nov,[12] AS dec
FROM
    (SELECT Year, Month,Total
     FROM View_Order) p
PIVOT
    (Sum(Total) FOR Month IN
        ([1], [2], [3], [4], [5], [6], [7], [8], [9],
        [10], [11], [12]))
)AS pvt
```

# Giới thiệu ngôn ngữ T-SQL

## Ví dụ: Sử dụng PIVOT

```
USE AdventureWorks
GO
SELECT VendorID, [164] AS Emp1, [198] AS Emp2, [223] AS
Emp3, [231] AS Emp4, [233] AS Emp5
FROM
(SELECT PurchaseOrderID, EmployeeID, VendorID
FROM Purchasing.PurchaseOrderHeader) p
PIVOT
(
COUNT (PurchaseOrderID)
FOR EmployeeID IN
( [164], [198], [223], [231], [233] )
) AS pvt
ORDER BY VendorID;
```

## Ví dụ: Sử dụng UNPIVOT

```
CREATE TABLE pvt (VendorID int, Emp1 int, Emp2 int,
Emp3 int, Emp4 int, Emp5 int)
GO
INSERT INTO pvt VALUES (1,4,3,5,4,4)
INSERT INTO pvt VALUES (2,4,1,5,5,5)
INSERT INTO pvt VALUES (3,4,3,5,4,4)
INSERT INTO pvt VALUES (4,4,2,5,5,4)
INSERT INTO pvt VALUES (5,5,1,5,5,5)
GO
--Unpivot the table.
SELECT VendorID, Employee, Orders
FROM
(SELECT VendorID, Emp1, Emp2, Emp3, Emp4, Emp5
FROM pvt) p
UNPIVOT
(Orders FOR Employee IN
(Emp1, Emp2, Emp3, Emp4, Emp5)
) AS unpvt
```

# Giới thiệu ngôn ngữ T-SQL

d) UNION và UNION ALL :

Toán tử UNION [ALL] dùng để hợp kết quả của hai hoặc nhiều câu truy vấn tương thích với nhau. Hai câu truy vấn tương thích là hai câu có cùng cấu trúc, tức là có cùng số cột và tập các cột tương ứng có cùng kiểu dữ liệu hoặc có các kiểu dữ liệu tương thích nhau. Cú pháp của câu lệnh:

```
select_statement UNION [ALL] select_statement
```

Tên của các cột trong phép toán UNION là tên các cột trong tập kết quả của khối câu lệnh SELECT thứ nhất trong UNION.

Theo mặc định phép toán UNION chỉ lấy đại diện cho tập các dòng trùng nhau. Nếu ta sử dụng từ khóa ALL, thì tất cả các dòng được cho vào bảng kết quả và các dòng trùng nhau sẽ không loại bỏ các dòng trùng nhau.

# Giới thiệu ngôn ngữ T-SQL

## Ví dụ Sử dụng UNION

```
SELECT * from LOP  
UNION ALL  
SELECT * from DMLOP
```

# Giới thiệu ngôn ngữ T-SQL

## 3.1.3 Lập trình cấu trúc trong SQL Server

### a) Các toán tử:

- *Toán tử gán*: Ký hiệu là dấu '=' được dùng để gán giá trị cho một biến hoặc một cột.

```
DECLARE @intValue int  
SELECT @intValue = 1  
PRINT @intValue
```

hoặc

```
DECLARE @intValue int  
SET @intValue = 1  
PRINT @intValue
```

- *Toán tử số học*: Đó là các phép toán cộng (+), trừ (-), nhân (\*), chia (/) và chia modul (%).

$$12+4=16$$

$$12-4=8$$

$$12*4=48$$

$$12/4=3$$

$$15\%2=1$$

# Giới thiệu ngôn ngữ T-SQL

- Toán tử so sánh: Đó là các phép toán so sánh giữa hai biểu thức và trả về giá trị TRUE hoặc FALSE. Đó là các phép so sánh: = (bằng), <> (khác), > (lớn hơn), >= (lớn hơn hoặc bằng), < (nhỏ hơn), <= (nhỏ hơn hoặc bằng).
- Toán tử logic: Kiểm tra điều kiện đúng của hai biểu thức, chúng thường được sử dụng cùng với các toán tử so sánh để trả về giá trị TRUE hoặc FALSE. Các toán tử logic được cho trong bảng sau.

Toán tử	Ý nghĩa	Ví dụ
ALL	So sánh một giá trị vô hướng với một tập các giá trị của một cột được lấy từ một câu truy vấn con. ALL trả về giá trị TRUE nếu tất cả các giá trị trong cột trả về giá trị TRUE ngược lại trả về	5 > ALL (SELECT * FROM sales)

# Giới thiệu ngôn ngữ T-SQL

	giá trị FALSE.			
AND	Kết hợp và so sánh giữa hai biểu thức Boolean, nếu cả hai biểu thức đều TRUE thì nó trả về giá trị TRUE và ngược lại nó trả về giá trị FALSE.	5 > 7 AND 6 < 15	NOT	Dùng để phủ định một biểu thức Boolean.
ANY	So sánh một giá trị vô hướng với một tập các giá trị của một cột được lấy từ một câu truy vấn con. Nó sẽ trả về giá trị TRUE nếu có bất cứ giá trị nào trong cột trả về giá trị TRUE. Nếu không có một giá trị nào trả về giá trị TRUE thì nó trả về giá trị FALSE. ANY tương tự như toán tử SOME.	5 > ANY (SELECT qty FROM sales)	OR	Kết hợp và so sánh giữa hai biểu thức Boolean, nếu một trong hai biểu thức là TRUE thì nó trả về giá trị TRUE và ngược lại nó trả về giá trị FALSE.
BETWEEN	Kiểm tra giá trị có nằm giữa phạm vi được chỉ định hay không. Trả về giá trị TRUE nếu nó nằm trong khoảng giá trị đó và ngược lại trả giá trị FALSE.	5 BETWEEN (3 AND 10)	SOME	So sánh một giá trị vô hướng với một tập các giá trị của một cột được lấy từ một câu truy vấn con. Nó sẽ trả về giá trị TRUE nếu có bất cứ giá trị nào trong cột trả về giá trị TRUE. Nếu không có một giá trị nào trả về giá trị TRUE thì nó trả về giá trị FALSE. SOME tương tự như toán tử ANY.
EXISTS	Kiểm tra xem có giá trị nào trả về khi thực hiện một câu truy vấn. Nếu có các giá trị trả về thì toán tử cho giá trị TRUE, ngược lại trả về giá trị FALSE.	EXISTS (SELECT * FROM test)		5 > SOME (SELECT * FROM sales)
IN	Kiểm tra xem một giá trị có tồn tại trong một tập các giá trị hay không. Nếu giá trị mà thuộc tập giá trị đó thì toán tử trả về giá trị TRUE, ngược lại trả về giá trị FALSE.	5 IN (SELECT qty FROM sales)		
LIKE	Dùng để so khớp các giá trị với một mẫu theo từ khóa LIKE. Nó sẽ trả về giá trị TRUE nếu khớp với mẫu ngược lại trả về giá trị FALSE. Ký tự % đại diện cho một dãy ký tự bất kỳ, _ đại diện cho một ký tự bất kỳ.	SELECT name WHERE name LIKE '%s%'		

# Giới thiệu ngôn ngữ T-SQL

- Toán tử ghép chuỗi (+): Dùng để ghép hai chuỗi với nhau thành một chuỗi. Toán tử ghép chuỗi được dùng với các kiểu dữ liệu char, varchar, nchar, nvarchar, text, và ntext.

`SELECT 'This' + ' is a test.'`

- Toán tử bit: Thực hiện thao tác với các bit-level với các kiểu dữ liệu Integer. Các toán tử đó được cho trong bảng

Toán tử	Ý nghĩa	Ví dụ
&	Thực hiện AND giữa các bit tương ứng giữa hai biểu diễn nhị phân của hai số integer.	$7 \& 51 = 3$ ( 7=111, 51=110011, 3=11)
	Thực hiện OR giữa các bit tương ứng giữa hai biểu diễn nhị phân của hai số integer.	$7   51 = 55$
^	Thực hiện XOR giữa các bit tương ứng giữa hai biểu diễn nhị phân của hai số integer. (hai bit giống nhau trả về bit 0, khác nhau trả về bit 1)	$7 ^ 51 = 52$
~	Thực hiện NOT của biểu thức biểu diễn nhị phân của một số nguyên	$\sim 7 = -8$

# Giới thiệu ngôn ngữ T-SQL

## b) Cấu trúc lặp:

SQL Server cung cấp hai cấu trúc lặp đó là: cấu trúc WHILE và GOTO.

- Cấu trúc lặp WHILE: Câu lệnh WHILE sẽ kiểm tra điều kiện trước khi thực hiện lệnh. Một khối lệnh là một tập các câu lệnh được bao trong cặp từ khóa BEGIN ...END. Cú pháp:

```
WHILE Boolean_expression
    {sql_statement| statement_block}
    [BREAK]
    {sql_statement| statement_block}
    [CONTINUE]
```

# Giới thiệu ngôn ngữ T-SQL

- Trong đó:
  - + Boolean\_expression: Là biểu thức điều kiện để kiểm tra điều kiện lặp. Vòng lặp sẽ được thực hiện khi biểu thức trả về giá trị True và kết thúc vòng lặp khi trả về giá trị False.
  - + sql\_statement|statement\_block: Đó là câu lệnh SQL hoặc khối các câu lệnh SQL sẽ được lặp lại trong câu lệnh While. Khối các câu lệnh SQL được bao trong cặp từ khóa BEGIN ... END
  - + BREAK: Từ khóa dùng để chỉ định dừng việc thực thi vòng lặp hiện tại. Tất cả các câu lệnh sau từ khóa BREAK và trước từ khóa END sẽ bị bỏ qua.
  - + CONTINUE: Từ khóa dùng để restart lại vòng lặp hiện tại tại ví trí bắt đầu. Tất cả các câu lệnh sau từ khóa CONTINUE và trước từ khóa END sẽ bị bỏ qua.

# Giới thiệu ngôn ngữ T-SQL

- Ví dụ: Sử dụng cấu trúc lặp WHILE đơn giản.

```
Use pubs
go
CREATE TABLE WhileLoopTest
(
    LoopID      INT,
    LoopValue   VARCHAR(32)
)
GO
SET NOCOUNT ON
DECLARE @intCounter      INT
DECLARE @vchLoopValue    VARCHAR(32)

SELECT @intCounter = 1
WHILE (@intCounter <= 100)
BEGIN
    SELECT @vchLoopValue = 'Loop Iteration #' +
        CONVERT(VARCHAR(4), @intCounter)
    INSERT INTO WhileLoopTest(LoopID, LoopValue)
        VALUES (@intCounter, @vchLoopValue)
    SELECT @intCounter = @intCounter + 1
END
```

# Giới thiệu ngôn ngữ T-SQL

- Cấu trúc lặp GOTO: Tương tự như cấu trúc WHILE, GOTO có thể cho phép lặp một chuỗi câu lệnh cho đến khi điều kiện được thỏa mãn.

\*Chú ý: Câu lệnh GOTO không nhất thiết phải sử dụng trong các vòng lặp mà có thể sử dụng để thoát khỏi vòng lặp khác.

Để sử dụng câu lệnh GOTO, trước hết ta phải định nghĩa một nhãn. Nhãn là một câu lệnh chỉ định vị trí mà câu lệnh GOTO sẽ nhảy đến. Để tạo nhãn ta sử dụng cú pháp sau:

LABLE:

Để nhảy đến nhãn trong code ta sử dụng câu lệnh GOTO theo cú pháp sau:

GOTO LABLE

Trong đó: LABLE là nhãn đã được định nghĩa ở trước đó trong code. Bằng việc sử dụng GOTO, ta có thể nhảy đến một vị trí bất kỳ trong code.

# Giới thiệu ngôn ngữ T-SQL

Ví dụ:Sử dụng cấu trúc lặp GOTO đơn giản.

```
Use pubs
Go
CREATE TABLE GotoLoopTest
(
    GotoID      INT,
    GotoValue   VARCHAR(32)
)
GO

SET NOCOUNT ON

DECLARE      @intCounter      INT
DECLARE      @vchLoopValue   VARCHAR(32)

SELECT @intCounter = 0

LOOPSTART:
    SELECT @intCounter = @intCounter + 1
    SELECT @vchLoopValue = 'Loop Iteration #' +
        CONVERT(VARCHAR(4), @intCounter)
    INSERT INTO GotoLoopTest(GotoID, GotoValue) VALUES
(@intCounter, @vchLoopValue)
    IF (@intCounter <= 1000)
        BEGIN
            GOTO LOOPSTART
        END
```

# Giới thiệu ngôn ngữ T-SQL

c) Cấu trúc rẽ nhánh:

Cấu trúc IF...ELSE: Cấu trúc IF...ELSE là một khối các câu lệnh dùng để rẽ nhánh dựa trên các tham số được cung cấp. Cú pháp của khối câu lệnh IF như sau:

```
IF expression
BEGIN
    sql_statements
END
[ELSE
BEGIN
    sql_statements
END]
```

Chú ý: Ta có thể sử dụng các cấu trúc IF lồng nhau.

# Giới thiệu ngôn ngữ T-SQL

## Ví dụ:Sử dụng cấu trúc rẽ nhánh IF.

```
Use pubs
Go

CREATE PROCEDURE uspCheckNumber
    @intNumber      INT
AS
IF @intNumber < 1
    BEGIN
        PRINT 'Number is less than 1.'
        RETURN
    END
ELSE IF @intNumber = 1
    BEGIN
        PRINT 'One'
        RETURN
    END
```

```
ELSE IF @intNumber = 2
    BEGIN
        PRINT 'Two'
        RETURN
    END
ELSE IF @intNumber = 3
    BEGIN
        PRINT 'Three'
        RETURN
    END
ELSE IF @intNumber = 4
    BEGIN
        PRINT 'Four'
        RETURN
    END
ELSE IF @intNumber = 5
    BEGIN
        PRINT 'Five'
        RETURN
    END
ELSE IF @intNumber = 6
    BEGIN
        PRINT 'Six'
        RETURN
    END
ELSE IF @intNumber = 7
    BEGIN
        PRINT 'Seven'
        RETURN
    END
ELSE IF @intNumber = 8
    BEGIN
        PRINT 'Eight'
        RETURN
    END
ELSE IF @intNumber = 9
    BEGIN
        PRINT 'Nine'
        RETURN
    END
ELSE IF @intNumber = 10
    BEGIN
        PRINT 'Ten'
        RETURN
    END
ELSE
    BEGIN
        PRINT 'Number is greater than 10.'
        RETURN
    END
```

# Giới thiệu ngôn ngữ T-SQL

- Cấu trúc CASE: Cấu trúc này được dùng để đánh giá một biểu thức và trả về một hoặc một số các kết quả dựa vào giá trị của biểu thức.
- Có 2 kiểu cấu trúc CASE khác nhau như sau:

- Simple CASE: Với cấu trúc này, một biểu thức sẽ được dùng để so sánh với một tập các giá trị để xác định kết quả. Cú pháp như sau:

```
CASE case_expression  
    WHEN expression THEN result  
    [...]  
    [ELSE else_result]  
END
```

- Searched CASE: Đánh giá tập các biểu thức Boolean để xác định kết quả. Cú pháp của nó như sau:

```
CASE  
    WHEN Boolean_expression THEN result  
    [...]  
    [ELSE else_result]  
END
```

# Giới thiệu ngôn ngữ T-SQL

- Trong đó:
  - +case\_expression: Biểu thức dùng để SQL Server đánh giá giá trị trong câu lệnh Simple CASE.
  - +Expression: Giá trị dùng để so sánh với biểu thức case\_expression nếu đúng thì nó sẽ trả về kết quả.
  - +Result: Kết quả sẽ được trả về nếu như giá trị biểu thức case\_expression so với Expression là đúng.
  - +Boolean\_expression: SQL Server dùng biểu thức Boolean để rẽ nhánh, nếu biểu thức nhận giá trị True thì sẽ thực hiện kết quả Result.
  - +else\_result: Thực hiện các kết quả sau ELSE.

# Giới thiệu ngôn ngữ T-SQL

Ví dụ: Sử dụng cấu trúc rẽ nhánh CASE dùng trong cả hai trường hợp Simple Case và

Searched Case.

```
Use pubs
Go
CREATE PROCEDURE uspCheckNumberCase
    @chrNumber      CHAR (2)
AS
IF  (CONVERT(INT,  @chrNumber) < 1)  OR  (CONVERT(INT,
@chrNumber) > 10)
    BEGIN
        SELECT CASE
            WHEN CONVERT(INT,  @chrNumber) < 1 THEN 'Number
is less than 1.'
            WHEN CONVERT(INT,  @chrNumber) > 10 THEN 'Number
is greater than 10.'
        END
        RETURN
    END
SELECT CASE CONVERT(INT,  @chrNumber)
    WHEN 1 THEN 'One'
    WHEN 2 THEN 'Two'
    WHEN 3 THEN 'Three'
    WHEN 4 THEN 'Four'
    WHEN 5 THEN 'Five'
    WHEN 6 THEN 'Six'
    WHEN 7 THEN 'Seven'
    WHEN 8 THEN 'Eight'
    WHEN 9 THEN 'Nine'
    WHEN 10 THEN 'Ten'
END
```

# Giới thiệu ngôn ngữ T-SQL

## d) Cấu trúc WAITFOR:

Cấu trúc WaitFor được dùng để ngăn việc thực thi một lô, thủ tục, hay một giao dịch cho đến một thời điểm nào đó hoặc sau một khoảng thời gian nào đó. Cú pháp của WAITFOR như sau:

```
WAITFOR { DELAY 'time' | TIME 'time' }
```

Trong đó:

- + DELAY: Chỉ định khoảng thời gian phải chờ. Tối đa là 24 giờ.
- + TIME: Chỉ định thời điểm thực thi một lô, thủ tục, hay một giao dịch

Ví dụ: Sử dụng cấu trúc WAITFOR để chờ đến lúc 21h 30 thì thực hiện xóa bản ghi.

```
BEGIN
    WAITFOR TIME '21:30'
    DELETE FROM DMLOP WHERE MALOP='TH6A'
END
```

# Giới thiệu ngôn ngữ T-SQL

Ví dụ: Xây dựng thủ tục time\_delay để chờ trong một khoảng thời gian nào đó và đưa ra thông báo khoảng thời gian đã chờ đó.

```
CREATE PROCEDURE time_delay @DELAYLENGTH char(9)
AS
DECLARE @RETURNINFO varchar(255)
BEGIN
    WAITFOR DELAY @DELAYLENGTH
    SELECT @RETURNINFO = 'A total time of ' +
        SUBSTRING(@DELAYLENGTH, 1, 2) +
        ' hours, ' +
        SUBSTRING(@DELAYLENGTH, 4, 2) +
        ' minutes, and ' +
        SUBSTRING(@DELAYLENGTH, 7, 2) +
        ' seconds ' +
        'has elapsed! Your time is up.';
    PRINT @RETURNINFO;
END;
GO
-- This next statement executes the time_delay procedure.
EXEC time_delay '00:05:00'
GO
```

# Các store procedure – Các thủ tục

## 3.2 Các store procedure – Các thủ tục:

### 3.2.1 Các khái niệm:

Store Procedure là một tập các phát biểu T-SQL mà SQL Server biên dịch thành một kế hoạch thực thi đơn. Lần đầu tiên khi SQL Server thực thi store procedure thì nó biên dịch store procedure thành kế hoạch và lưu trong bộ nhớ đệm. Mỗi khi gọi thực hiện store procedure này thì nó sử dụng lại kế hoạch này mà không phải biên dịch lại lần nữa.

T-SQL store procedure tương tự như các ngôn ngữ lập trình khác, chúng chấp nhận các tham số nhập, trả về giá trị xuất thông qua tham số hoặc trả về thông điệp cho biết thủ tục thành công hay thất bại.

# Các store procedure – Các thủ tục

## 3.2.2 Tạo store procedure:

Cú pháp:

```
CREATE {PROC|PROCEDURE} [schema_name.]  
procedure_name [ ; number ]  
[{@parameter [type_schema_name.] data_type }  
[VARYING] [= default ][[ OUT|OUTPUT ]  
[,...n ]  
  
[ WITH <procedure_option> [ ,...n ]  
AS { [ BEGIN ] statements [ END ] }  
[;]  
  
<procedure_option> ::=  
[ ENCRYPTION ]  
[ RECOMPILE ]
```

# Các store procedure – Các thủ tục

- Trong đó:
  - + procedure\_name: là tên của store procedure sẽ được tạo.
  - + parameter: Là các tham số truyền vào store procedure, ta phải định nghĩa chúng trong phần khai báo của store procedure. Khai báo bao gồm tên của tham số (trước tên tham số sử dụng tiền tố @), kiểu dữ liệu của tham số và một số chỉ định đặc biệt thuộc vào mục đích sử dụng của tham số đó.
  - + ; number: Là số nguyên tùy chọn được sử dụng trong nhóm các thủ tục có cùng tên.
  - + data type: Kiểu của tham số trong phần khai báo.
  - + [VARYING]: Đây là tùy chọn được chỉ định khi cursor trả về như một tham số.
  - + [= default] : Gán giá trị mặc định cho tham số. Nếu không gán giá trị mặc định thì tham số nhận giá trị NULL.
  - + OUTPUT: Đây là từ khóa chỉ định tham số đó là tham số xuất. Tham số xuất không dùng được với kiểu dữ liệu Text và image.
  - + [...n]: Chỉ định rằng có thể khai báo nhiều tham số.
  - + RECOMPILE: Chỉ định Database Engine không xây dựng kế hoạch cho thủ tục này và thủ tục sẽ được biên dịch tại thời điểm thực thi thủ tục.
  - + ENCRYPTION: Chỉ định SQL Server sẽ mã hóa bản text lệnh CREATE PROCEDURE. Users không thể truy cập vào các bảng hệ thống hoặc file dữ liệu để truy xuất bản text đã mã hóa.

# Các store procedure – Các thủ tục

- Thực thi store procedure trong SQL Server: Để thực thi một thủ tục trong SQL Server ta sử dụng cú pháp sau:

```
{ EXEC | EXECUTE }  
  { module_name [ ;number ] }  
    [ [ @parameter = ] { value  
                        | @variable [ OUTPUT ]  
                        | [ DEFAULT ]  
                      }  
    ]  
  [ ,...n ]  
  [ WITH RECOMPILE ]  
+ module_name: Là tên thủ tục cần thực hiện.
```

- + ;number: Chỉ định thủ tục trong nhóm thủ tục cùng tên.
- + @parameter: Tên tham số trong thủ tục.
- + @variable: Chỉ định biến chứa các tham số hoặc trả về tham số
- + DEFAULT: Chỉ định lấy giá trị mặc định của biến.

# Các store procedure – Các thủ tục

Ví dụ: Xây dựng thủ tục XemDSSV.

```
Use QLDiemSV
Go
IF EXISTS(Select name from sysobjects
           where name ='p_DSSV' and type='p')
DROP PROCEDURE p_DSSV
GO
CREATE PROCEDURE p_DSSV
AS
SELECT MaSV, Hodem + ' ' +TensV as Hoten, Ngaysinh, MaLop
      From HOSOSV
GO
```

- Thực thi thủ tục p\_DSSV

```
Use QLDiemSV
Go
EXEC p_DSSV
```

# Các store procedure – Các thủ tục

\*Truyền tham số nhập vào trong store procedure

Ví dụ: Xây dựng thủ tục pp\_DSSV để hiển thị danh sách sinh viên theo tham số mã lớp. Mã lớp được truyền vào khi thủ tục được thực hiện.

```
Use QLDiemSV
Go
IF EXISTS(Select name from sysobjects
where name ='p_DSSV' and type='P')
DROP PROCEDURE p_DSSV
GO
CREATE PROCEDURE p_DSSV
@parMaLop Varchar(10)='TH%'
AS
SELECT MaSV, Hodem + ' ' +TensV as Hoten, Ngaysinh
From HOSOSV Where MaLop like @parMaLop
GO
```

Gọi thực thi thủ tục trên với truyền giá trị cho tham số nhập như sau:

```
EXEC p_DSSV 'TH03A'
EXEC p_DSSV @parMaLop=DEFAULT
```

# Các store procedure – Các thủ tục

\*Sử dụng tham số xuất trong store procedure.

Ví dụ: Xây dựng thủ tục pp\_Siso để xuất giá trị sĩ số của một lớp theo tham số mã lớp. Mã lớp được truyền vào khi thủ tục được thực hiện.

```
Use QLDiemSV
Go
IF EXISTS(Select name from sysobjects where name
='pp_Siso' and type='p')
DROP PROCEDURE pp_Siso
GO
CREATE PROCEDURE pp_Siso
@parMaLop Char(10), @parSiso Int OUTPUT
AS
SELECT @parSiso=count(*)
From HOSOSV Where MaLop=@parMaLop
GO
```

```
DECLARE @sido int
exec pp_Siso 'TH03A',@parSiso=@sido OUTPUT
Print 'Si so lop TH03A la :'+ convert(varchar(3),@sido)
Go
Kết quả thực hiện chương trình:
Si so lop TH03A là :12
```

# Các store procedure – Các thủ tục

## \*Sử dụng biến cục bộ:

Các biến cục bộ được sử dụng trong bô lệnh, trong chương trình gọi (Script) hoặc trong thủ tục.Biến cục bộ thường được giữ các giá trị sẽ được kiểm tra trong phát biểu điều kiện và giữ giá trị sẽ được trả về bởi lệnh RETURN. Phạm vi của biến cục bộ trong store procedure là từ điểm biến đó được khai báo cho đến khi thoát store procedure. Ngay khi store procedure kết thúc thì biến đó không được tham chiếu nữa. Cú pháp khai báo biến cục bộ:

```
DECLARE <parameter> [AS] <data type>
```

Giống như khai báo các biến ở trên, trước tên biến phải có tiền tố @. Giá trị khởi tạo ban đầu của biến là NULL. Để thiết lập giá trị của biến ta sử dụng cú pháp:

```
SET <parameter> = <expression>
SELECT <parameter> = <expression>
```

## Các store procedure – Các thủ tục

\* Câu lệnh PRINT: Dùng để hiển thị chuỗi thông báo tới người sử dụng. Chuỗi thông báo này có thể dài tới 8000 ký tự. Cú pháp của lệnh PRINT như sau:

```
PRINT < messages>
```

# Các store procedure – Các thủ tục

\* Sử dụng SELECT để trả về giá trị: Ta có thể trả về giá trị bằng việc sử dụng SELECT trong thủ tục hoặc trả về kết quả thiết lập từ truy vấn SELECT.

Ví dụ. Xây dựng thủ tục pp\_Siso để xuất giá trị số lượng sinh viên của một lớp theo tham số mã lớp ra ngoài. Mã lớp được truyền vào khi thủ tục được thực hiện.

Use QLDiemSV

Go

```
IF EXISTS(Select name from sysobjects where name  
='pp_Siso' and type='p')  
DROP PROCEDURE pp_Siso  
GO  
CREATE PROCEDURE pp_Siso  
@parMaLop Char(10), @parSiso Int OUTPUT  
AS  
SELECT @parSiso=count(*) From HOSOSV Where  
MaLop=@parMaLop  
GO  
DECLARE @siso int  
exec pp_Siso 'TH03A',@parSiso=@siso OUTPUT  
SELECT 'Số lượng lớp TH03A là :='= @siso  
Go
```

# Các store procedure – Các thủ tục

- Lệnh RETURN: Ta có thể sử dụng lệnh RETURN để thoát không điều kiện khỏi thủ tục. Khi lệnh RETURN được thực thi trong thủ tục, khi đó các câu lệnh sau RETURN trong thủ tục sẽ bị bỏ qua và thoát khỏi thủ tục để trở về dòng lệnh tiếp theo trong chương trình gọi. Ngoài ra, ta có thể sử dụng lệnh RETURN để trả về giá trị cho chương trình gọi, giá trị trả về phải là một số nguyên, nó có thể là một hằng số hoặc một biến. Cú pháp như sau:

```
RETURN [ integer_expression ]
```

- Ví dụ: Cho CSDL pubs. Xây dựng thủ tục usp\_4\_31 kiểm tra một chủ đề có tồn tại trong bảng titles hay không? Nếu tồn tại một chủ đề thì hiển thị chủ đề đó. Nếu không tồn tại chủ đề đó thì thủ tục trả về giá trị 1 hoặc có nhiều hơn một chủ đề đó thì trả về giá trị 2.

# Các store procedure – Các thủ tục

```
Use pubs
Go
IF EXISTS(Select name from sysobjects
            where name ='usp_4_31' and type='p')
DROP PROCEDURE usp_4_31
GO
CREATE PROCEDURE usp_4_31
    @vchTitlePattern    VARCHAR(80) = '%'
AS
SELECT @vchTitlePattern = '%' + @vchTitlePattern + '%'
IF (SELECT COUNT(*) FROM titles
        WHERE title LIKE @vchTitlePattern) < 1
BEGIN
    RETURN 1
END
IF (SELECT COUNT(*) FROM titles
        WHERE title LIKE @vchTitlePattern) > 1

BEGIN
    RETURN 2
END
SELECT title, price FROM titles
    WHERE title LIKE @vchTitlePattern
RETURN 0
GO
DECLARE @intReturnValue      INT
EXEC @intReturnValue = usp_4_31 'Tin hoc'
IF (@intReturnValue = 1)
BEGIN
    PRINT 'There are no corresponding titles.'
END
IF (@intReturnValue = 2)
BEGIN
    PRINT 'There are multiple titles that match this
          criteria. Please narrow your
          search.'
END
GO
```

# Các store procedure – Các thủ tục

3.2.3 Thay đổi, xóa, xem nội dung store procedure:

a) Thay đổi store procedure

Cú pháp:

```
ALTER {PROC|PROCEDURE} [schema_name.]  
procedure_name [ ; number ]  
[{@parameter [type_schema_name.] data_type }  
[VARYING] [= default ][[ OUT|OUTPUT ]  
[,...n ]  
[ WITH <procedure_option> [ ,...n ]  
AS { [ BEGIN ] statements [ END ] }  
[ ; ]  
  
<procedure_option> ::=  
[ ENCRYPTION ]  
[ RECOMPILE ]
```

# Các store procedure – Các thủ tục

Ví dụ: Ta sửa lại thủ tục p\_DSSV như sau:

```
Use QLDiemSV
Go
ALTER PROCEDURE p_DSSV
@parMaLop Char(10)
AS
SELECT MaSV, Hodem, TensV, Ngaysinh
      From HOSOSV Where MaLop=@parMaLop
GO
```

b) Xóa store procedure

Cú pháp: `DROP { PROC | PROCEDURE } { [schema_name.] procedure }`

```
Use QLDiemSV
Go
DROP PROCEDURE p_DSSV
Go
```

# Các store procedure – Các thủ tục

c) Xem nội dung store procedure Để xem nội dung của thủ tục ta sử dụng thủ tục hệ thống sp\_helptext.

Ví dụ: Xem nội dung thủ tục pp\_DSSV:

```
Use QLDiemSV  
Go  
Exec sp_helptext pp_DSSV  
Go
```

Kết quả như hình sau:

The screenshot shows a SQL query window in SSMS. The query is:

```
Use QLDiemSV  
Go  
Exec sp_helptext pp_DSSV  
go
```

The results pane displays the script for the pp\_DSSV stored procedure:

Text
1 CREATE PROCEDURE pp_DSSV
2 @parMaLop Char(10)
3 AS
4 SELECT MaSV, Hodem + ' ' +TensV as Hoten, Ngaysinh From HOSOSV Where MaLop=@parMaLop

At the bottom, the status bar shows "Query batch completed." and the session information: THUHuong (8.0) | THUHuong\chuhuong (52) | QLDiemSV | 0:00:00 | 4 rows | Ln 7, Col 1.

# Các hàm trong SQL Server

## 3.3 Các store function – Các hàm

### 3.3.1 Các khái niệm

Tất cả các ngôn ngữ lập trình, bao gồm cả T-SQL, việc có các hàm tạo cho các ứng dụng trở lên mạnh mẽ. Ngoài ra, người lập trình có thể tự tạo một hàm riêng cho mình làm cho hệ thống dễ được mở rộng. Một hàm - function - trong SQL Server được định nghĩa là một thủ tục đơn giản bao gồm một nhóm các câu lệnh SQL

### 3.3.2 Tạo các hàm

Một hàm do người dùng định nghĩa được tạo bằng cách sử dụng câu lệnh CREATE FUNCTION theo cú pháp sau:

# Các hàm trong SQL Server

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

# Các hàm trong SQL Server

Trong đó:

- + [owner\_name.]: Chỉ định tên đối tượng sẽ sở hữu. Ta không phải bắt buộc chỉ định tên người sẽ tạo đối tượng sở hữu nó.
- + function\_name: tên của hàm ta sẽ tạo.
- + parameter\_name: Là các tham số Input cho hàm. Các tham số này xây dựng cũng tương tự như trong stored procedure.
- + scalar\_data\_type: Là kiểu dữ liệu vô hướng của tham số. Một hàm có thể nhận bất kỳ kiểu dữ liệu nào như là tham số trừ các kiểu timestamp, cursor, text, ntext, image.
- + default: Chỉ định giá trị mặc định cho tham số, tương tự như trong stored procedure.
- + [...n]: Chỉ định một hàm có thể tạo nhiều tham số. Một hàm trong SQL Server có thể chứa tối 1024 tham số.

# Các hàm trong SQL Server

- + RETURNS: từ khóa này chỉ định kiểu dữ liệu hàm sẽ trả về. Kiểu dữ liệu của hàm có thể là một kiểu dữ liệu vô hướng hoặc một bảng.
- + scalar\_data\_type: Ta sẽ chỉ định kiểu dữ liệu nếu như hàm trả về một giá trị vô hướng. Ở đây ta phải chỉ định kiểu độ dài dữ liệu.
- + TABLE: Đây là kiểu dữ liệu cho phép hàm có thể trả về nhiều dòng dữ liệu.
- + column\_definition: Định nghĩa các cột cho kiểu dữ liệu TABLE. Các cột này được định nghĩa tương tự như định nghĩa các cột trong bảng.
- + table\_constraint: Định nghĩa các ràng buộc trong kiểu dữ liệu TABLE này.
- + [...n]: Chỉ định có thể có nhiều cột và nhiều ràng buộc trong bảng.

# Các hàm trong SQL Server

- + WITH ENCRYPTION: Từ khóa chỉ định code của hàm sẽ được mã hóa trong bảng syscomments.
- + SCHEMABINDING: Từ khóa này chỉ định hàm được tạo để buộc vào tất cả các đối tượng mà nó tham chiếu.
- + [...n]: Chỉ định có thể có nhiều từ khóa khác ngoài hai từ khóa trên.
- + AS: Từ khóa cho biết code của hàm bắt đầu.
- + BEGIN: Đi cùng với END để tạo thành bao khói bao các câu lệnh trong thân hàm.
- + function\_body: thân của hàm.
- + END: Đi cùng với BEGIN để tạo thành bao khói bao các câu lệnh trong thân hàm.
- + RETURN: Từ khóa này sẽ gửi giá trị tới thủ tục gọi hàm.
- + select\_statement: đi kèm với RETURN để gửi giá trị tới thủ tục gọi hàm.

# Các hàm trong SQL Server

## 3.3.3 Các ví dụ tạo hàm

Ví dụ: Xây Dựng một hàm fncGetThreeBusinessDays trả về ngày làm việc thứ 3 tính từ ngày bắt đầu @dtmDateStart.

```
CREATE FUNCTION fncGetThreeBusinessDays
    (@dtmDateStart DATETIME)
    RETURNS DATETIME
AS
BEGIN
    IF DATEPART(dw, @dtmDateStart) = 4
        BEGIN
            RETURN (DATEADD(dw, 5, @dtmDateStart))
        END
    ELSE IF DATEPART(dw, @dtmDateStart) = 5
        BEGIN
            RETURN (DATEADD(dw, 5, @dtmDateStart))
        END
    ELSE IF DATEPART(dw, @dtmDateStart) = 6
        BEGIN
            RETURN (DATEADD(dw, 5, @dtmDateStart))
        END
END
```

# Các hàm trong SQL Server

```
ELSE IF DATEPART(dw, @dtmDateStart) = 7
BEGIN
    RETURN (DATEADD(dw, 4, @dtmDateStart))
END
RETURN (DATEADD(dw, 3, @dtmDateStart))
END
```

Thực hiện thử nghiệm hàm trên, ta xây dựng script sau:

```
DECLARE @dtmDate      DATETIME
SELECT      @dtmDate = '1/10/2008'
SELECT      DATENAME(dw, @dtmDate)
SELECT      DATENAME(dw, dbo.
                                fncGetThreeBusinessDays(@dtmDate))
```

# Các hàm trong SQL Server

**Ví dụ:** Sử dụng hàm fncGetThreeBusinessDays trên để tính toán trên một cột trong một bảng.

```
CREATE TABLE OrderInfo
(
    OrderID          INT NOT NULL,
    ShippingMethod   VARCHAR(16) NOT NULL,
    OrderDate        DATETIME NOT NULL DEFAULT GETDATE(),
    ExpectedDate     AS (
        dbo.fncGetThreeBusinessDays(OrderDate)
    )
)
GO

INSERT OrderInfo VALUES (1, 'UPS GROUND', GETDATE())
INSERT OrderInfo VALUES (2, 'FEDEX STANDARD',
                        DATEADD(dd, 2, GETDATE()))
INSERT OrderInfo VALUES (3, 'PRIORITY MAIL',
                        DATEADD(dd, 4, GETDATE()))
GO

SELECT OrderID, ShippingMethod, CONVERT(VARCHAR(12),
    OrderDate, 1) + ' (' + DATENAME(dw, OrderDate) + ') '
AS 'OrderDate', CONVERT(VARCHAR(12), ExpectedDate, 1) +
' (' + DATENAME(dw, ExpectedDate) + ')' AS 'ExpectedDate'
FROM OrderInfo
```

# Các hàm trong SQL Server

**Ví dụ:** Xây dựng hàm trả về các dòng dữ liệu gồm thông tin về điểm của các môn học theo Mã lớp.

```
CREATE FUNCTION fncBangDiem(@MaLop CHAR(10))
    RETURNS @TableName TABLE
    (
        MaSV      CHAR(10),
        Hoten     nvarchar(100),
        TenMH     NVARCHAR(50),
        DiemL1    INT,
        DiemL2    INT
    )
AS
BEGIN
    INSERT INTO @TableName
        SELECT Di.MaSV, Ho.HoDem + ' ' + Ho.TenSV, Mo.TenMH,
               Di.DiemL1, Di.DiemL2
        FROM DIEM Di
            JOIN HOSOSV Ho ON (Di.MaSV = Ho.MaSV)
            JOIN MONHOC Mo ON (Di.MaMH = Mo.MaMH)
        WHERE Ho.MaLop = @MaLop
    RETURN
END
GO
```

# Các hàm trong SQL Server

Thử nghiệm gọi hàm này trong đoạn script chương trình sau và kết quả cho trong hình : Select \* from fncBangDiem('TH03A')

The screenshot shows a Microsoft SQL Server Management Studio (SSMS) window titled "Query - THUHuong.QLDiemSV.THUHuong\chuhuong - E:\Bai giang\SQL...". The query entered is "Select \* from fncBangDiem('TH03A')". The results are displayed in a grid:

	MaSV	Hoten	TenMH	DiemL1	DiemL2
1	20040970	Phạm Đức Hiếu	Hệ quản trị CSDL	7	NULL
2	20040970	Phạm Đức Hiếu	Lập trình Web	6	NULL
3	20040971	Nguyễn Thế Bảo	Hệ quản trị CSDL	4	6
4	20040971	Nguyễn Thế Bảo	Lập trình Web	8	NULL
5	20071004	Nguyễn Tiến Đạt	Hệ quản trị CSDL	6	NULL
6	20071004	Nguyễn Tiến Đạt	Lập trình Web	3	NULL
7	SV-0149	Vũ Thị Thanh Hoa	Hệ quản trị CSDL	8	NULL
8	SV-0232	Trần Thị Hà Chính	Hệ quản trị CSDL	4	5
9	SV-0248	Lê Thanh Hà	Hệ quản trị CSDL	3	NULL

Gọi hàm fncBangDiem

# Các hàm trong SQL Server

**Ví dụ:** Xây dựng hàm đưa ra danh sách sinh viên của một lớp phụ thuộc tham số mã lớp đưa vào.

```
CREATE FUNCTION fncDSSV(@MaLop CHAR(10))  
RETURNS TABLE  
AS  
RETURN  
SELECT MaSV, HoDem + ' ' + TenSV AS Hoten, NgaySinh  
FROM HOSOSV  
WHERE MaLop=@MaLop  
GO
```

Gọi hàm này trong đoạn script chương trình sau:

```
Select * from fncDSSV('TH03A')
```

# Các hàm trong SQL Server

## 3.3.4 Thay đổi, xóa, xem nội dung store function

a) Thay đổi các hàm Để thay đổi các hàm ta dùng câu lệnh ALTER FUNCTION.

```
ALTER FUNCTION [owner_name.]function_name
(
    [ @parameter_name scalar_data_type [= default] ]
    [,...n] ]
)
RETURNS scalar_data_type |
TABLE(column_definition | table_constraint
[,...n])

[WITH ENCRYPTION | SCHEMABINDING [,...n] ]
[AS]
[BEGIN function_body END] | RETURN [()
select_statement ()]
```

# Các hàm trong SQL Server

**Ví dụ:** Thay đổi hàm fncDSSV

```
ALTER FUNCTION fncDSSV (@MaLop CHAR(10))  
RETURNS TABLE  
AS  
RETURN  
SELECT MaSV, HoDem + ' ' + TenSV AS Hoten  
FROM HOSOSV  
WHERE MaLop=@MaLop  
GO
```

# Các hàm trong SQL Server

## b) Xóa hàm

Để xóa hàm ta dùng câu lệnh DROP FUNCTION.

```
DROP FUNCTION { [ schema_name. ] function_name }
```

### Ví dụ 4.43. Xóa hàm fncDSSV

```
DROP FUNCTION fncDSSV
```

# Trigger

## 3.4 Trigger:

### 3.4.1 Các khái niệm:

Trigger là một kiểu stored procedure đặc biệt được kích nổ (thực thi) một cách tự động khi xảy ra một sự kiện trên Database server và không thể thực thi bằng tay. Trigger ược chia ra làm hai nhóm: DML và DDL trigger.

+ DML triggers (hay Standart triggers) thực thi khi một người sử dụng cố gắng sửa đổi dữ liệu thông qua sự kiện thao tác dữ liệu (data manipulation language - DML) INSERT, UPDATE, hoặc DELETE trên bảng hoặc view. DML triggers thường được sử dụng trong chính sách đảm bảo các quy tắc thương mại hoặc đảm bảo tính toàn vẹn dữ liệu.

+ DDL triggers được thực thi đáp ứng các sự kiện định nghĩa lược đồ dữ liệu (data definition language - DDL). Nhóm lệnh chính của các lệnh định nghĩa lược đồ dữ liệu là CREATE, ALTER, và DROP. Nhóm DDL trigger là nhóm trigger mới được bổ sung trong SQL Server 2005 Database Engine.

# Trigger

- DML triggers có hai kiểu, chúng được xử lý khác nhau. Kiểu phổ biến với mọi người nhất đó là kiểu AFTER trigger và kiểu thứ hai đó là INSTEAD OF trigger.
  - AFTER Triggers: Khi các câu lệnh thay đổi dữ liệu được thực hiện trên một bảng có định nghĩa trigger, một vài xử lý xuất hiện trước khi trigger thực sự nổ. Đầu tiên bộ truy vấn sẽ kiểm tra trên bảng có bắt cứ các ràng buộc nào hay không. Nếu có, SQL Server sẽ tiến hành kiểm tra tính hợp lệ của dữ liệu trên mọi ràng buộc nào. Nếu dữ liệu đang thêm vào hoặc đang sửa đổi mà không thỏa các ràng buộc thì bộ truy vấn sẽ dừng câu lệnh trên và khi đó trigger sẽ không được kích nổ. Ngược lại, khi kiểm tra thành công thì các trigger sẽ được thực thi.

# Trigger

Trước khi bắt cứ câu lệnh nào chứa trigger thực sự được thực hiện, SQL Server tạo ra hai bảng đặc biệt lưu trong bộ nhớ có cùng cấu trúc với bảng mà trên đó trigger được tạo.

- + Table DELETED chứa các giá trị đang bị xóa từ bảng.
- + Table INSERTED chứa các giá trị đang được Add vào bảng.

Nếu trigger được định nghĩa là INSERT trigger thì SQL Server sẽ chỉ tạo bảng INSERTED, còn nếu là DELETE trigger thì SQL Server sẽ chỉ tạo bảng DELETED.

Cuối cùng là nếu trigger được định nghĩa là UPDATE trigger thì SQL Server sẽ tạo cả hai bảng vì INSERTED sẽ chứa ảnh của hàng sau khi thay đổi còn bảng DELETED chứa ảnh của hàng trước khi thay đổi.

# Trigger

\*INSTEAD OF trigger: Là một đặc điểm thêm vào của SQL Server 2000. Như tên gọi đã ám chỉ, INSTEAD OF trigger được kích nổ thay cho hành động được sử dụng để kích nó.

Nghĩa là, nếu một trigger được định nghĩa là INSTEAD OF INSERT trigger thì trigger này sẽ được nổ khi câu lệnh Insert được thực hiện trên bảng. Sau khi câu lệnh sửa đổi dữ liệu được gửi đi thì INSTEAD OF trigger được kích nổ và hiện lập tức ngay lập tức. Các ràng buộc không được kiểm tra trước khi trigger được kích nổ, mặc dù các bảng INSERTED, DELETED vẫn được tạo. Sau khi các bảng này được tạo thì quá trình xử lý trigger tương tự như quá trình xử lý của stored procedure. INSTEAD OF trigger có thể được tạo trên bảng hoặc trên view.

# Trigger

## 3.4.2 Tạo trigger:

a) Dùng T-SQL để tạo trigger Ta dùng T-SQL để tạo trigger theo cú pháp sau:

- Tạo DML Trigger:

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

# Trigger

- Trong đó:
  - + trigger\_name: tham số chỉ định tên của trigger sẽ tạo.
  - + ON: Chỉ định lấy tên của bảng hoặc view mà ta sẽ xây dựng trigger trên đó.
  - + table | view: Tên của bảng hoặc của view.
  - + WITH ENCRYPTION: Dùng để chỉ định code của trigger mã hóa lưu trữ trên bảng syscomments mà người khác không thể xem code của trigger đó.
    - + FOR: Định nghĩa hành động mà trigger được kích nổ và kiểu của trigger ta đang tạo. AFTER trigger sẽ là trigger mặc định nếu chỉ có chỉ định FOR.
    - + AFTER: Chỉ định AFTER trigger. Tùy chọn này chỉ định riêng không lẫn với từ khóa INSTEAD OF. AFTER triggers không được định nghĩa trên view.
    - + INSTEAD OF: Từ khóa chỉ định đây là INSTEAD OF trigger. Tùy chọn này chỉ định để không lẫn với từ khóa AFTER.
    - + DELETE: Chỉ định rằng trigger ta đang tạo sẽ được kích nổ đáp ứng với hành động DELETE của bảng hoặc view.
    - + INSERT: Chỉ định rằng trigger ta đang tạo sẽ được kích nổ đáp ứng với hành động INSERT của bảng hoặc view.
    - + UPDATE: Chỉ định rằng trigger ta đang tạo sẽ được kích nổ đáp ứng với hành động UPDATE của bảng hoặc view.

# Trigger

- + NOT FOR REPLICATION: Chỉ định rằng bất cứ bản sao nào của các hành động chạy ngầm dưới bảng này đều không được kích nổ trigger này.
- + AS: Chỉ định phần code của trigger bắt đầu từ đây.
- + IF UPDATE (column): Được dùng trong các trigger INSERT, UPDATE. Cấu trúc này được dùng để kiểm tra các sửa đổi trên cột chỉ định và sau đó là các hành động trên nó.
  - + {AND | OR} UPDATE (column): Chỉ định rằng ta có thể sử dụng chuỗi các cấu trúc UPDATE với nhau để kiểm tra một vài cột tại cùng một thời điểm.
  - + ...n: Chỉ định ta có thể lặp lại các cấu trúc trên nếu cần thiết.
  - + IF(COLUMNS\_UPDATED()): Chỉ dùng trong các trigger INSERT, UPDATE. Hàm trả về một bit chỉ định cột bị chỉnh sửa trong quá trình INSERT, UPDATE trên bảng cơ sở.
  - + bitwise\_operator: Được dùng để so sánh với bit trả về của hàm COLUMNS\_UPDATED()
  - + updated\_bitmask: Được sử dụng để kiểm cột nào thực sự được Update trong câu lệnh Insert hoặc Update.
  - + sql\_statement: Các câu lệnh T-SQL
  - + ... n: Chỉ định lặp lại các câu lệnh T-SQL.

# Trigger

- Tạo DDL Trigger:

```
CREATE TRIGGER trigger_name  
ON { ALL SERVER | DATABASE }  
[ WITH ENCRYPTION ]  
{ FOR |AFTER } {event_type } [,....n ]  
AS { sql_statement [ ; ] [ ...n ] }
```

+ **DATABASE**: Chỉ định phạm vi của DDL trigger là database hiện thời. Trigger sẽ kích nổ khi sự kiện event\_type xảy ra trên database hiện thời.

+ **ALL SERVER**: Chỉ định phạm vi của DDL trigger là server hiện thời. Trigger sẽ kích nổ khi sự kiện event\_type xảy ra trên server hiện thời.

+ **event\_type**: Là tên của sự kiện mà là nguyên nhân kích nổ DDL trigger. Các sự kiện này có thể là: CREATE\_FUNCTION, CREATE\_INDEX, GRANT\_DATABASE, CREATE\_TABLE, ALTER\_VIEW, ALTER\_TABLE, DROP\_TABLE, DROP\_VIEW, .v.v...

# Trigger

Ví dụ :Tạo một AFTER INSERT Trigger.

Trong ví dụ này ta định nghĩa AFTER INSERT Trigger trên bảng TriggerTableChild. Trigger này có nhiệm vụ kiểm tra xem có sự tương ứng với các dòng của bảng TriggerTableParent hay không. Nếu không có sự tương ứng nó sẽ thực hiện roll back (cuốn lại) lại giao dịch đó và dòng thông báo lỗi hiện lên. Nếu có sự tương ứng thì giao dịch được thực hiện một cách bình thường.

```
CREATE TABLE TriggerTableParent
(
    TriggerID      INT,
    TriggerText    VARCHAR (32)
)
GO
```

# Trigger

```
INSERT INTO TriggerTableParent VALUES (1, 'Trigger Text 1')
INSERT INTO TriggerTableParent VALUES (2, 'Trigger Text 2')
INSERT INTO TriggerTableParent VALUES (3, 'Trigger Text 3')
INSERT INTO TriggerTableParent VALUES (4, 'Trigger Text 4')
INSERT INTO TriggerTableParent VALUES (5, 'Trigger Text 5')
GO

CREATE TABLE TriggerTableChild
(
    TriggerID      INT,
    TriggerSubText  VARCHAR(32)
)
GO

CREATE TRIGGER trTriggerTableChildInsert
ON TriggerTableChild
FOR INSERT
AS
IF (SELECT COUNT(*) FROM TriggerTableParent TTP
    INNER JOIN INSERTED I ON (TTP.TriggerID= I.TriggerID))= 0
BEGIN
    ROLLBACK TRANSACTION
    RAISERROR ('No corresponding record was found in the
                TriggerTableParent table for this insert.', 11, 1)
END
ELSE
    Print 'This was inserted'
GO
SET NOCOUNT ON
INSERT INTO TriggerTableChild VALUES(1, 'Sub Trigger Text 1')
INSERT INTO TriggerTableChild VALUES(2, 'Sub Trigger Text 2')
INSERT INTO TriggerTableChild VALUES(3, 'Sub Trigger Text 3')
INSERT INTO TriggerTableChild VALUES(6, 'Sub Trigger Text 6')
GO
```

# Trigger

## Ví dụ: Dùng bảng INSERTED và DELETED

```
CREATE TRIGGER trTriggerTableParentUpdate1
ON TriggerTableParent1
AFTER UPDATE
AS
SET NOCOUNT ON

PRINT      'Contents of the INSERTED Table:'
SELECT      *
FROM       INSERTED

PRINT      'Contents of the DELETED Table:'
SELECT      *
FROM       DELETED

PRINT      'Contents of the TriggerTableParent Table:'
SELECT      TTP.* FROM    TriggerTableParent1 TTP
INNER JOIN INSERTED I ON
(TTP.TriggerID = I.TriggerID)

ROLLBACK TRANSACTION
GO
UPDATE TriggerTableParent1
SET TriggerText = 'Changed Trigger Text 1'
WHERE TriggerID = 1
```

Khi sử dụng lệnh UPDATE, kết quả cho thấy bảng INSERTED chứa các giá trị mới; bảng DELETED chứa các giá trị cũ và bảng TriggerTableParent1 chứa các giá trị mới.

# Trigger

## Ví dụ. Sử dụng cấu trúc IF UPDATED trong UPDATE Trigger

```
CREATE TRIGGER trTriggerTableChildUpdate
ON TriggerTableChild1
AFTER UPDATE
AS
IF UPDATE(TriggerID)
BEGIN
    IF (SELECT COUNT(*) FROM TriggerTableParent1 TTP
        INNER JOIN INSERTED I ON
        (TTP.TriggerID = I.TriggerID)) = 0
        BEGIN
            RAISERROR ('No parent record exists for this
modification. Transaction cancelled.', 11,
1)
            ROLLBACK TRANSACTION
            RETURN
        END
    END
GO

UPDATE TriggerTableChild1
SET TriggerID = 7
WHERE TriggerID = 1
GO
```

# Trigger

Ví dụ: Tạo INSTEAD OF Trigger. Bởi vì INSTEAD OF Trigger được kích nổ trước khi bắt cứ dữ liệu nào được sửa đổi trong CSDL, do đó vai trò của hai bảng INSERTED và DELETED bị giảm nhẹ hơn.

```
CREATE TABLE TriggerTableParent2
(
    TriggerID      INT,
    TriggerText    VARCHAR(32)
)
GO
INSERT INTO TriggerTableParent2 VALUES(1, 'Trigger Text 1')
INSERT INTO TriggerTableParent2 VALUES(2, 'Trigger Text 2')
INSERT INTO TriggerTableParent2 VALUES(3, 'Trigger Text 3')
INSERT INTO TriggerTableParent2 VALUES(4, 'Trigger Text 4')
INSERT INTO TriggerTableParent2 VALUES(5, 'Trigger Text 5')
GO
CREATE TABLE TriggerTableChild2
(
    TriggerID      INT,
    TriggerSubText  VARCHAR(32)
)
GO
```

# Trigger

```
CREATE TABLE TriggerTableChild2
(
    TriggerID      INT,
    TriggerSubText  VARCHAR(32)
)
GO
CREATE TRIGGER trTriggerTableParent2InsteadOfUpdate
ON TriggerTableParent2
INSTEAD OF UPDATE
AS
SET NOCOUNT ON

PRINT      'Contents of the INSERTED Table:'
SELECT      *
FROM        INSERTED
PRINT      'Contents of the DELETED Table:'
SELECT      *
FROM        DELETED
PRINT      'Contents of the TriggerTableParent Table:'
SELECT      TTP.*
FROM        TriggerTableParent2 TTP
INNER JOIN INSERTED I ON
(TTP.TriggerID = I.TriggerID)

ROLLBACK TRANSACTION
GO

UPDATE      TriggerTableParent2
SET        TriggerText = 'Changed Trigger Text 1'
WHERE      TriggerID = 1
GO
```

Khi thực hiện lệnh Update, ta thấy bảng cơ sở chưa được chèn dữ liệu do trigger đã được thực thi trước khi có sự sửa đổi dữ liệu, đây chính là đặc điểm của INSTEAD OF Trigger. Cụ thể khi sử dụng lệnh UPDATE, kết quả cho thấy bảng INSERTED chứa các giá trị mới; bảng DELETED chứa các giá trị cũ và bảng TriggerTableParent2 chứa các giá trị cũ

# Trigger

Ví dụ: View của các bảng và INSTEAD OF Trigger. Một trong những đặc điểm nổi bật chính của INSTEAD OF Trigger là cho phép người sử dụng thực hiện các câu lệnh thay đổi dữ liệu trên view của nhiều bảng. Trong ví dụ này, ta xây dựng một INSTEAD OF Trigger thực hiện chức năng này.

```
SET NOCOUNT ON  
GO  
CREATE TABLE ViewTable1  
(  
    KeyColumn      INT,  
    Table1Column   VARCHAR(32)  
)  
GO
```

```
GO  
INSERT INTO ViewTable1 VALUES (1, 'ViewTable1 Value 1')  
INSERT INTO ViewTable1 VALUES (2, 'ViewTable1 Value 2')  
INSERT INTO ViewTable1 VALUES (3, 'ViewTable1 Value 3')  
INSERT INTO ViewTable1 VALUES (4, 'ViewTable1 Value 4')  
INSERT INTO ViewTable1 VALUES (5, 'ViewTable1 Value 5')  
INSERT INTO ViewTable1 VALUES (6, 'ViewTable1 Value 6')  
INSERT INTO ViewTable1 VALUES (7, 'ViewTable1 Value 7')  
INSERT INTO ViewTable1 VALUES (8, 'ViewTable1 Value 8')  
INSERT INTO ViewTable1 VALUES (9, 'ViewTable1 Value 9')  
INSERT INTO ViewTable1 VALUES (10, 'ViewTable1 Value 10')  
GO  
  
CREATE TABLE ViewTable2  
(  
    KeyColumn      INT,  
    Table2Column   VARCHAR(32)  
)  
GO  
  
INSERT INTO ViewTable2 VALUES (1, 'ViewTable2 Value 1')  
INSERT INTO ViewTable2 VALUES (2, 'ViewTable2 Value 2')  
INSERT INTO ViewTable2 VALUES (3, 'ViewTable2 Value 3')  
INSERT INTO ViewTable2 VALUES (4, 'ViewTable2 Value 4')  
INSERT INTO ViewTable2 VALUES (5, 'ViewTable2 Value 5')  
INSERT INTO ViewTable2 VALUES (6, 'ViewTable2 Value 6')  
INSERT INTO ViewTable2 VALUES (7, 'ViewTable2 Value 7')  
INSERT INTO ViewTable2 VALUES (8, 'ViewTable2 Value 8')  
INSERT INTO ViewTable2 VALUES (9, 'ViewTable2 Value 9')  
INSERT INTO ViewTable2 VALUES (10, 'ViewTable2 Value 10')  
GO
```

# Trigger

```
CREATE VIEW TestView1
AS
SELECT VT1.KeyColumn, VT1.Table1Column, VT2.Table2Column
FROM ViewTable1 VT1 INNER JOIN ViewTable2 VT2 ON
(VT1.KeyColumn = VT2.KeyColumn)
GO
INSERT INTO TestView1 VALUES (11, 'ViewTable1 Value 11',
'ViewTable2 Value 11')
GO
CREATE TRIGGER trTestView1InsteadOfInsert
ON TestView1
INSTEAD OF INSERT
AS
DECLARE @intKeyColumn      INT
DECLARE @vchTable1Column   VARCHAR(32)
DECLARE @vchTable2Column   VARCHAR(32)
DECLARE @intError          INT

SET NOCOUNT ON
SELECT @intKeyColumn=KeyColumn, @vchTable1Column =
Table1Column, @vchTable2Column = Table2Column
FROM INSERTED
BEGIN TRANSACTION
    INSERT INTO ViewTable1
VALUES (@intKeyColumn, @vchTable1Column)
    SELECT @intError = @@ROWCOUNT
    INSERT INTO ViewTable2 VALUES (@intKeyColumn,
@vchTable2Column)
    SELECT @intError = @intError + @@ROWCOUNT
    IF ((@intError < 2) OR (@intError % 2) <> 0)
    BEGIN
        RAISERROR('An error occurred during the multitable
insert.', 1, 11)
        ROLLBACK TRANSACTION
        RETURN
    END
COMMIT TRANSACTION
GO
INSERT INTO TestView1 VALUES (11, 'ViewTable1 Value
11','ViewTable2 Value 11')
GO
```

Trong đoạn script trên, đầu tiên ta tạo 2 bảng và chèn 10 dòng vào hai bảng đó.

Hai bảng đó có chung một cột là KeyColumn và ta tạo view thể hiện dữ liệu của hai bảng.

Lệnh INSERT sẽ thực hiện chèn dữ liệu vào hai bảng.

Lệnh INSERT đầu tiên sẽ bị lỗi vì có nhiều bảng trong mệnh đề FROM của câu lệnh SELECT tạo view đó.

Lệnh tiếp theo sẽ được thực hiện bởi trigger đã tạo.

# Trigger

## \*Tiến trình xử lý giao dịch - TRANSACTION:

DBMS phải đảm bảo một giao dịch được xử lý như một đơn vị cơ sở. Điều này có nghĩa là khi DBMS đang thực hiện tiến trình xử lý câu lệnh đầu tiên trong một giao dịch thì tiến trình này phải được tiếp tục cho đến khi tất cả các câu lệnh trong giao dịch được thực hiện thành công, giao dịch này không bị bẻ gãy. Tất cả các câu lệnh trong một giao dịch phải được xử lý như một đơn vị công việc. Khi một tiến trình bị bẻ gãy ở giữa một giao dịch, như kết quả của việc re boot, hệ thống bị vỡ, thì DBMS phải đưa database về trạng thái tồn tại trước khi giao dịch được bắt đầu. Một câu lệnh SQL đặc biệt để làm điều này đó là ROLL BACK. Nếu tất cả câu lệnh trong đã được thực hiện thành công thì sự thay đổi Database được thực hiện bởi câu lệnh COMMIT. COMMIT và ROLL BACK là các câu lệnh SQL thao chuẩn ANSI/ISO, giống như các câu lệnh SELECT, INSERT, .v.v...

# Trigger

## Ví dụ: DDL Trigger.

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
    PRINT 'You must disable Trigger "safety" to
drop or alter tables!'
    ROLLBACK
```

# Trigger

## 3.4.3 Các thao tác quản lý trigger

- Xem mã trigger: Dùng thủ tục sp\_helptext.

Ví dụ Xem code của trigger trTestView1InsteadOfInsert

```
Exec sp_helptext trTestView1InsteadOfInsert  
go
```

- Xem những trigger nào đang tồn tại trên một bảng hoặc một view:  
Dùng thủ tục sp\_helptrigger.

Ví dụ. Xem các trigger trên bảng HOSOSV

```
Use QLDiemSV  
Go  
Exec sp_helptrigger HOSOSV  
go
```

# Trigger

\* Thay đổi nội dung trigger: Để thay đổi trigger ta dùng câu lệnh ALTER TRIGGER theo cú pháp sau:

- Sửa DML Trigger:

```
ALTER TRIGGER trigger_name
ON {table | view }
[WITH ENCRYPTION]
{
{{FOR | AFTER | INSTEAD OF} { [DELETE] [,]
[INSERT] [,] [UPDATE]
[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 ]
}
```

- Sửa DDL Trigger:

```
ALTER TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH ENCRYPTION ]
{ FOR |AFTER } {event_type } [,...n ]
AS { sql_statement [ ; ] [ ...n ]}
```

# Trigger

## Ví dụ Thay đổi nội dung trigger trTestView1InsteadOfInsert

```
ALTER TRIGGER trTestView1InsteadOfInsert
ON TestView1
INSTEAD OF INSERT
AS
DECLARE @intKeyColumn      INT
DECLARE @vchTable1Column   VARCHAR(32)
DECLARE @vchTable2Column   VARCHAR(32)
DECLARE @intError          INT

SET NOCOUNT ON

SELECT @intKeyColumn=KeyColumn, @vchTable1Column =
Table1Column,    @vchTable2Column = Table2Column
FROM    INSERTED

BEGIN TRANSACTION
    INSERT INTO ViewTable1
VALUES (@intKeyColumn,@vchTable1Column)
    SELECT @intError = @@ROWCOUNT
    INSERT INTO ViewTable2 VALUES (@intKeyColumn,
@vchTable2Column)
    SELECT @intError = @intError + @@ROWCOUNT
    IF ((@intError < 2) OR (@intError % 2) <> 0)
        BEGIN
            ROLLBACK TRANSACTION
            RETURN
        END
    COMMIT TRANSACTION
GO
```

# Trigger

\* Xóa một trigger: Dùng câu lệnh DROP TRIGGER.

- Xóa DML Trigger:

--

```
DROP TRIGGER schema_name.trigger_name [ ,...n ]
```

- Xóa DDL Trigger:

```
DROP TRIGGER trigger_name [ ,...n ]
  ON { DATABASE | ALL SERVER }
```

Ví dụ Xóa trigger trTestView1InsteadOfInsert

```
DROP TRIGGER trTestView1InsteadOfInsert
```

```
go
```

# Trigger

- Vô hiệu hóa hoặc làm cho có hiệu lực một trigger ta dùng câu lệnh:

```
DISABLE | ENABLE TRIGGER{[ schema . ]  
trigger_name [ ,...n ] | ALL }  
ON { object_name | DATABASE | ALL SERVER } [ ; ]
```

Ví dụ: Vô hiệu hóa trigger trTriggerTableParent2InsteadOfUpdate Trên bảng TriggerTableParent2.

```
ALTER TABLE TriggerTableParent2  
DISABLE TRIGGER trTriggerTableParent2InsteadOfUpdate  
GO.
```

Ví dụ: Vô hiệu hóa trigger [trTriggerTableChildInsert] trên bảng TriggerTableChild.

```
DISABLE TRIGGER [trTriggerTableChildInsert] ON  
[dbo].[TriggerTableChild]
```

Ví dụ: Làm cho trigger trTriggerTableParent2InsteadOfUpdate trên bảng TriggerTableParent2 có hiệu lực.

```
ALTER TABLE TriggerTableParent2  
ENABLE TRIGGER trTriggerTableParent2InsteadOfUpdate  
GO.
```