Bài 5. Thao tác dữ liệu, tạo khung nhìn và lập trình hàm trong SQL Server

- Mục đích, yêu cầu: Cung cấp cho sinh viên kiến thức về thao tác dữ liệu, tạo khung nhìn và lập trình hàm trong SQL Server.

- Hình thức tổ chức dạy học: Lý thuyết, trực tiếp + trực tuyến + tự học

- Thời gian: Lý thuyết (trên lớp: 3; online: 3) Tự học, tự nghiên cứu: 12

- Nội dung chính:

Thao tác dữ liệu, tạo khung nhìn và lập trình hàm trong SQL Server

I. Thao tác dữ liệu

1.1. Bổ sung, cập nhật và xoá dữ liệu

Các câu lệnh thao tác dữ liệu trong SQL không những chỉ sử dụng để truy vấn dữ liệu mà còn để thay đổi và cập nhật dữ liệu trong cơ sở dữ liệu. So với câu lệnh SELECT, việc sử dụng các câu lệnh để bổ sung, cập nhật hay xoá dữ liệu đơn giản hơn nhiều.

Trong phần còn lại của chương này sẽ đề cập đến 3 câu lệnh:

• Lệnh INSERT

• Lệnh UPDATE

• Lệnh DELETE

1.2. Bổ sung dữ liệu

Dữ liệu trong các bảng được thể hiện dưới dạng các dòng (bản ghi). Để bổ sung thêm các dòng dữ liệu vào một bảng, ta sử dụng câu lệnh INSERT. Hầu hết các hệ quản trị CSDL dựa trên SQL cung cấp các cách dưới đây để thực hiện thao tác bổ sung dữ liệu cho bảng:

• Bổ sung từng dòng dữ liệu với mỗi câu lệnh INSERT. Đây là các sử dụng thường gặp nhất trong giao dịch SQL.

• Bổ sung nhiều dòng dữ liệu bằng cách truy xuất dữ liệu từ các bảng dữ liệu khác.

Bổ sung từng dòng dữ liệu với lệnh INSERT

Để bổ sung một dòng dữ liệu mới vào bảng, ta sử dụng câu lệnh INSERT với cú pháp như sau:

INSERT INTO *tên_bảng*[(*danh_sách_cột*)] VALUES(*danh_sách_trị*)

Trong câu lệnh INSERT, danh sách cột ngay sau tên bảng không cần thiết phải chỉ định nếu giá trị các trường của bản ghi mới được chỉ định đầy đủ trong danh sách trị. Trong trường hợp này, thứ tự các giá trị trong danh sách trị phải bằng với số lượng các trường của bảng cần bổ sung dữ liệu cũng như phải tuân theo đúng thứ tự của các trường như khi bảng được định nghĩa.

Câu lệnh dưới đây bổ sung thêm một dòng dữ liệu vào bảng KHOA

```
INSERT INTO khoa VALUES('DHT10','KhoaLuật','054821135')
```

Trong trường hợp chỉ nhập giá trị cho một số cột trong bảng, ta phải chỉ định danh sách các cột cần nhập dữ liệu ngay sau tên bảng. Khi đó, các cột không được nhập dữ liệu sẽ nhận giá trị mặc định (nếu có) hoặc nhận giá trị NULL (nếu cột cho phép chấp nhận giá trị NULL). Nếu một cột không có giá trị mặc định và không chấp nhận giá trị NULL mà không đuợc nhập dữ liệu, câu lệnh sẽ bị lỗi.

Câu lệnh dưới đây bổ sung một bản ghi mới cho bảng SINHVIEN

```
INSERT INTO sinhvien(masv,hodem,ten,gioitinh,malop)
VALUES('0241020008','Nguyễn Công','Chính',1,'C24102')
```

câu lệnh trên còn có thể được viết như sau:

```
INSERT INTO sinhvien VALUES('0241020008','Nguyễn
Công','Chính', NULL,1,NULL,'C24102')
```

Bổ sung nhiều dòng dữ liệu từ bảng khác

Một cách sử dụng khác của câu lệnh INSERT được sử dụng để bổ sung nhiều dòng dữ liệu vào một bảng, các dòng dữ liệu này được lấy từ một bảng khác thông qua câu lệnh SELECT. Ở cách này, các giá trị dữ liệu được bổ sung vào bảng không được chỉ định tường minh mà thay vào đó là một câu lệnh SELECT truy vấn dữ liệu từ bảng khác.

Cú pháp câu lệnh INSERT có dạng như sau:

```
INSERTINTO tên_bảng[(danh_sách_cột)] câu_lệnh_SELECT
```

Giả sử ta có bảng LUUSINHVIEN bao gồm các trường HODEM, TEN, NGAYSINH. Câu lệnh dưới đây bổ sung vào bảng LUUSINHVIEN các dòng dữ liệu có được từ câu truy vấn SELECT:

```
INSERT INTO luusinhvien SELECThodem,ten,ngaysinh FROM
sinhvien WHERE noisinh like '%Hà Nội%'
```

Khi bổ sung dữ liệu theo cách này cần lưu ý một số điểm sau:

- Kết quả của câu lệnh `SELECT` phải có số cột bằng với số cột được chỉ định trong bảng đích và phải tương thích về kiểu dữ liệu.
- Trong câu lệnh `SELECT` được sử dụng mệnh đề `COMPUTE ... BY`

1.3. Cập nhật dữ liệu

Câu lệnh `UPDATE` trong SQL được sử dụng để cập nhật dữ liệu trong các bảng. Câu lệnh này có cú pháp như sau:

```
UPDATE tên_bảng SETtên_cột = biểu_thức [, ..., tên_cột_k =
biểu_thức_k] [FROM danh_sách_bảng] [WHERE điều_kiện]
```

Sau `UPDATE` là tên của bảng cần cập nhật dữ liệu. Một câu lệnh `UPDATE` có thể cập nhật dữ liệu cho nhiều cột bằng cách chỉ định các danh sách tên cột và biểu thức tương ứng sau từ khoá `SET`. Mệnh đề `WHERE` trong câu lệnh `UPDATE` thường được sử dụng để chỉ định các dòng dữ liệu chịu tác động của câu lệnh (nếu không chỉ định, phạm vi tác động của câu lệnh được hiểu là toàn bộ các dòng trong bảng)

Câu lệnh dưới đây cập nhật lại số đơn vị học trình của các môn học có số đơn vị học trình nhỏ hơn 2

```
UPDATE monhoc
SET sodvht = 3
WHERE sodvht = 2
```

Sử dụng cấu trúc CASE trong câu lệnh UPDATE

Cấu trúc CASE có thể được sử dụng trong biểu thức khi cần phải đưa ra các quyết định khác nhau về giá trị của biểu thức

Giả sử ta có bảng NHATKYPHONG sau đây

SOPHONG	LOAIPHONG	SONGAY	TIENPHONG
101	A	5	
202	B	5	
101	A	2	
102	C	3	

Sau khi thực hiện câu lệnh:

```
UPDATE nhatkyphong SET tienphong=songay*CASE WHEN
loaiphong='A' THEN 100 WHEN loaiphong='B' THEN 70 ELSE 50
END
```

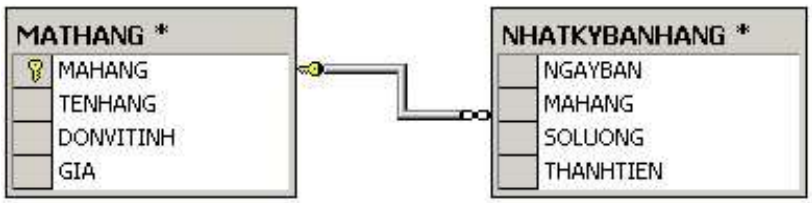
Dữ liệu trong bảng sẽ là

SOPHONG	LOAIPHONG	SONGAY	TIENPHONG
101	A	5	500
202	B	5	350
101	A	2	200
102	C	3	150

Điều kiện cập nhật dữ liệu liên quan đến nhiều bảng

Mệnh đề `FROM` trong câu lệnh `UPDATE` được sử dụng khi cần chỉ định các điều kiện liên quan đến các bảng khác với bảng cần cập nhật dữ liệu. Trong trường hợp này, trong mệnh đề WHERE thường có điều kiện nối giữa các bảng.

Giả sử ta có hai bảng MATHANG và NHATKYBANHANG như sau:



Câu lệnh dưới đây sẽ cập nhật giá trị trường THANHTIEN của bảng NHATKYBANHANG theo công thức THANHTIEN = SOLUONG × GIA

```
UPDATE nhatkybanhang SET thanhtien = soluong*gia
FROM mathang
WHERE nhatkybanhang.mahang = mathang.mahang
```

Câu lệnh UPDATE với truy vấn con

Tương tự như trong câu lệnh SELECT, truy vấn con có thể được sử dụng trong mệnh đề WHERE của câu lệnh UPDATE nhằm chỉ định điều kiện đối với các dòng dữ liệu cần cập nhật dữ liệu.

Câu lệnh ở trên có thể được viết như sau:

```
UPDATE nhatkybanhang SETthanhtien = soluong*gia
FROM mathang
WHERE mathang.mahang =(SELECT mathang.mahang FROM mathang
WHERE mathang.mahang=nhatkybanhang.mahang)
```

1.4. Xoá dữ liệu

Để xoá dữ liệu trong một bảng, ta sử dụng câu lệnh `DELETE` . Cú pháp của câu lệnh này như sau:

```
DELETE FROM tên_bảng [FROM danh_sách_bảng] [WHERE
điều_kiện]
```

Trong câu lệnh này, tên của bảng cần xoá dữ liệu được chỉ định sau DELETE FROM. Mệnh đề WHERE trong câu lệnh được sử dụng để chỉ định điều kiện đối với các dòng dữ liệu cần xoá. Nếu câu lệnh DELETE không có mệnh đề WHERE thì toàn bộ các dòng dữ liệu trong bảng đều bị xoá.

Câu lệnh dưới đây xoá khỏi bảng SINHVIEN những sinh viên sinh tại Huế

```
DELETE FROM sinhvien WHERE noisinh LIKE '%Huế%'
```

Xoá dữ liệu khi điều kiện liên quan đến nhiều bảng

Nếu điều kiện trong câu lệnh DELETE liên quan đến các bảng không phải là bảng cần xóa dữ liệu, ta phải sử dụng thêm mệnh đề FROM và sau đó là danh sách tên các bảng đó. Trong trường hợp này, trong mệnh đề WHERE ta chỉ định thêm điều kiện nối giữa các bảng

Câu lệnh dưới đây xoá ra khỏi bảng SINHVIEN những sinh viên lớp *Tin K24*

```
DELETE FROM sinhvien FROM lop WHERE
lop.malop=sinhvien.malop AND tenlop='Tin K24'
```

Sử dụng truy vấn con trong câu lệnh DELETE

Một câu lệnh SELECT có thể được lồng vào trong mệnh đề WHERE trong câu lệnh DELETE để làm điều kiện cho câu lệnh tương tự như câu lệnh UPDATE.

Câu lệnh dưới đây xoá khỏi bảng LOP những lớp không có sinh viên nào học

```
DELETE FROM lop WHERE malop NOT IN (SELECT DISTINCT malop
FROM sinhvien)
```

Xoá toàn bộ dữ liệu trong bảng

Câu lệnh `DELETE` không chỉ định điều kiện đối với các dòng dữ liệu cần xoá trong mệnh đề `WHERE` sẽ xoá toàn bộ dữ liệu trong bảng. Thay vì sử dụng câu lệnh `DELETE` trong trường hợp này, ta có thể sử dụng câu lệnh `TRUNCATE` có cú pháp như sau:

```
TRUNCATE TABLE tên_bảng
```

Câu lệnh sau xoá toàn bộ dữ liệu trong bảng *diemthi*:

```
DELETE FROM diemthi có tác dụng tương tự với câu lệnh
TRUNCATE TABLE diemthi
```

II. Làm việc với View (khung nhìn)

Khái niệm view (Khung nhìn)

Các bảng trong cơ sở dữ liệu đóng vai trò là các đối tượng tổ chức và lưu trữ dữ liệu. Như vậy, ta có thể quan sát được dữ liệu trong cơ sở dữ liệu bằng cách thực hiện các truy vấn trên bảng dữ liệu. Ngoài ra, SQL còn cho phép chúng ta quan sát được dữ liệu thông qua việc định nghĩa các khung nhìn.

Một khung nhìn (view) có thể được xem như là một bảng "ảo" trong cơ sở dữ liệu có nội dung được định nghĩa thông qua một truy vấn (câu lệnh `SELECT`). Như vậy, một khung nhìn trông giống như một bảng với một tên khung nhìn và là một tập bao gồm các dòng và các cột. Điểm khác biệt giữa khung nhìn và bảng là khung nhìn không được xem là một cấu trúc lưu trữ dữ liệu tồn tại trong cơ sở dữ liệu. Thực chất dữ liệu quan sát được trong khung nhìn được lấy từ các bảng thông qua câu lệnh truy vấn dữ liệu.

Hình dưới đây minh hoạ cho ta thấy khung nhìn có tên DSSV được định nghĩa thông qua câu lệnh `SELECT` truy vấn dữ liệu trên hai bảng SINHVIEN và LOP:

```
SELECT masv,hodem,ten, DATEDIFF(YY,ngaysinh,GETDATE()) AS
tuoi,tenlop
FROM sinhvien,lop
WHERE sinhvien.malop=lop.malop
```

Khi khung nhìn DSSV đã được định nghĩa, ta có thể sử dụng câu lệnh SELECT để truy vấn dữ liệu từ khung nhìn như đối với các bảng. Khi trong câu truy vấn xuất hiện khung nhìn, hệ quản trị cơ sở dữ liệu sẽ dựa vào đinh nghĩa của khung nhìn đểchuyển yêu

cầu truy vấn dữ liệu liên quan đến khung nhìn thành yêu cầu tương tự trên các bảng cơ sở và việc truy vấn dữ liệu được thực hiện bởi yêu cầu tương đương trên các bảng.

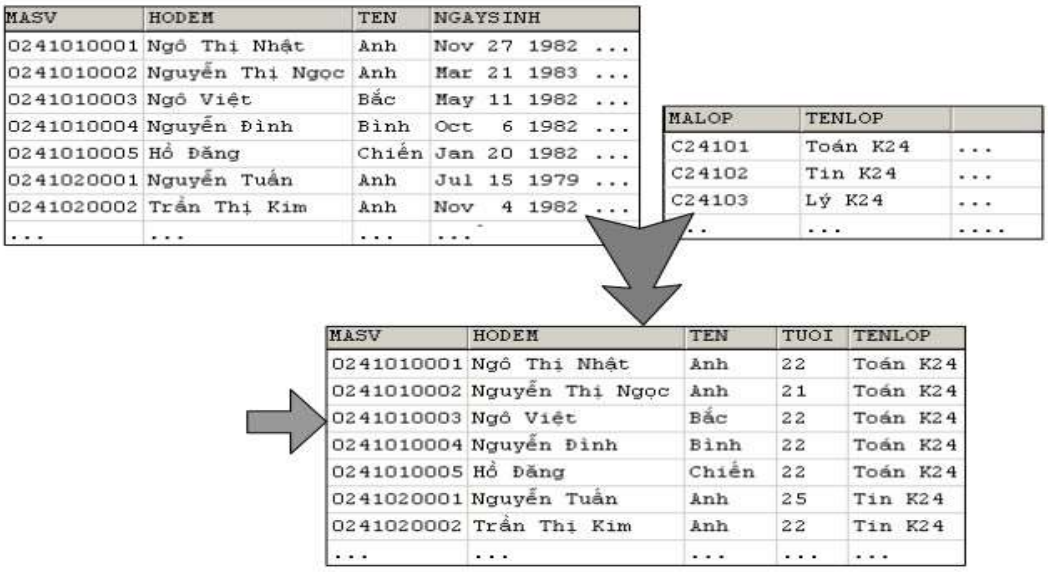
Việc sử dụng khung nhìn trong cơ sở dữ liệu đem lại các lợi ích sau đây:

- **Bảo mật dữ liệu :**Người sử dụng được cấp phát quyền trên các khung nhìn với những phần dữ liệu mà người sử dụng được phép. Điều này hạn chế được phần nào việc người sử dụng truy cập trực tiếp dữ liệu.

- **Đơn giản hoá các thao tác truy vấn dữ liệu:**Một khung nhìn đóng vai trò như là một đối tượng tập hợp dữ liệu từ nhiều bảng khác nhau vào trong một "bảng". Nhờ vào đó, người sử dụng có thể thực hiện các yêu cầu truy vấn dữ liệu một cách đơn giản từ khung nhìn thay vì phải đưa ra những câu truy vấn phức tạp.

- **Tập trung và đơn giản hoá dữ liệu :**Thông qua khung nhìn ta có thể cung cấp cho người sử dụng những cấu trúc đơn giản, dễ hiểu hơn về dữ liệu trong cơ sở dữ liệu đồng thời giúp cho người sử dụng tập trung hơn trên những phần dữ liệu cần thiết.

- **Độc lập dữ :** Một khung nhìn có thể cho phép người sử dụng có được cái nhìn về dữ liệu độc lập với cấu trúc của các bảng trong cơ sở dữ liệu cho dù các bảng cơ sở có bị thay đổi phần nào về cấu trúc.



MASV	HODEM	TEN	NGAYSINH
0241010001	Ngô Thị Nhật	Anh	Nov 27 1982 ...
0241010002	Nguyễn Thị Ngọc	Anh	Mar 21 1983 ...
0241010003	Ngô Việt	Bắc	May 11 1982 ...
0241010004	Nguyễn Đình	Bình	Oct 6 1982 ...
0241010005	Hồ Đăng	Chiến	Jan 20 1982 ...
0241020001	Nguyễn Tuấn	Anh	Jul 15 1979 ...
0241020002	Trần Thị Kim	Anh	Nov 4 1982 ...
...	...	...	...

MALOP	TENLOP	
C24101	Toán K24	...
C24102	Tin K24	...
C24103	Lý K24	...
..	...	....

MASV	HODEM	TEN	TUOI	TENLOP
0241010001	Ngô Thị Nhật	Anh	22	Toán K24
0241010002	Nguyễn Thị Ngọc	Anh	21	Toán K24
0241010003	Ngô Việt	Bắc	22	Toán K24
0241010004	Nguyễn Đình	Bình	22	Toán K24
0241010005	Hồ Đăng	Chiến	22	Toán K24
0241020001	Nguyễn Tuấn	Anh	25	Tin K24
0241020002	Trần Thị Kim	Anh	22	Tin K24
...	...	...	...	...

Khung nhìn DSSV với dữ liệu được lấy từ bảng SINHVIEN và LOP

Tuy nhiên, việc sử dụng khung nhìn ũc ng tồn tại một số nhược điểm sau:

- Do hệ quản trị cơ sở dữ liệu thực hiện việc chuyển đổi các truy vấn trên khung nhìn thành những truy vấn trên các bảng cơ sở nên nếu một khung nhìn được định nghĩa bởi một truy vấn phức tạp thì sẽ dẫn đến chi phí về mặt thời gian khi thực hiện truy vấn liên quan đến khung nhìn sẽ lớn.
- Mặc dù thông qua khung nhìn có thể thực hiện được thao tác bổ sung và cập nhật dữ liệu cho bảng cơ sở nhưng chỉ hạn chế đối với những khung nhìn đơn giản. Đối với những khung nhìn phức tạp thì thường không thực hiện được , hay nói cách khác là dữ liệu trong khung nhìn là chỉ đọc.

Tạo khung nhìn

Câu lệnh CREATE VIEW được sử dụng để tạo ra khung nhìn và có cú pháp như sau:

```
CREATE VIEW tên_khung_nhìn[(danh_sách_tên_cột)] AS
câu_lệnh_SELECT
```

Câu lệnh dưới đây tạo khung nhìn có tên DSSV từ câu lệnh SELECT truy vấn dữ liệu từ hai bảng SINHVIEN và LOP

```
CREATE VIEW dssv
AS
SELECT masv,hodem,ten,
DATEDIFF(YY,ngaysinh,GETDATE()) AS tuoi,tenlop
FROM sinhvien,lop
WHERE sinhvien.malop=lop.malop
```

và nếu thực hiện câu lệnh:

```
SELECT * FROM dssv
```

ta có được kết quả như sau:

MASV	HODEM	TEN	TUOI	TENLOP
0241010001	Ngô Thị Nhật	Anh	22	Toán K24
0241010002	Nguyễn Thị Ngọc	Anh	21	Toán K24
0241010003	Ngô Việt	Bắc	22	Toán K24
0241010004	Nguyễn Đình	Bình	22	Toán K24

Nếu trong câu lệnh CREATE VIEW, ta không chỉ định danh sách các tên cộtcho khung nhìn, tên các cột trong khung nhìn sẽ chính là tiêu đề các cột trong kết quả của câu

lệnh SELECT. Trong trường hợp tên các cột của khung nhìn đươc chỉ định, chúng phải có cùng số lượng với số lượng cột trong kết quả của câu truy vấn.

Câu lệnh dưới đây tạo khung nhìn từ câu truy vấn tương tự như ví dụ trên nhưng có đặt tên cho các cột trong khung nhìn:

```
CREATE VIEW dssv(ma,ho,ten,tuoi,lop)
AS
SELECT masv,hodem,ten,
DATEDIFF(YY,ngaysinh,GETDATE()),tenlop
FROM sinhvien,lop
WHERE sinhvien.malop=lop.malop
```

và câu lệnh

```
SELECT * FROM dssv
```

trong trường hợp này có kết quả như sau:

MA	HO	TEN	TUOI	LOP
0241010001	Ngô Thị Nhật	Anh	22	Toán K24
0241010002	Nguyễn Thị Ngọc	Anh	21	Toán K24
0241010003	Ngô Việt	Bắc	22	Toán K24
0241010004	Nguyễn Đình	Bình	22	Toán K24
0241010005	Hồ Đăng	Chiến	22	Toán K24

Khi tạo khung nhìn với câu lệnh CREATE VIEW, ta cần phải lưu ý một số nguyên tắc sau:

- Tên khung nhìn và tên cột trong khung nhìn, cũng giống như bảng, phải tuân theo qui tắc định danh.
- Không thể qui định ràng buộc và tạo chỉ mục cho khung nhìn.
- Câu lệnh SELECT với mệnh đề COMPUTE ... BY không được sử dụng để định nghĩa khung nhìn.
- Phải đặt tên cho các cột của khung nhìn trong các trường hợp sau đây:

Trong kết quả của câu lệnh SELECT có ít nhất một cột được sinh ra bởi một biểu thức (tức là không phải là một tên cột trong bảng cơ sở) và cột đó không được đặt tiêu đề. Tồn tại hai cột trong kết quả của câu lệnh SELECT có cùng tiêu đề cột.

Câu lệnh dưới đây là câu lệnh sai do cột thứ 4 không xác định được tên cột

```
CREATE VIEW tuoisinhvien
AS
SELECT masv,hodem,ten,DATEDIFF(YY,ngaysinh,GETDATE()) FROM
sinhvien
```

Xoá khung nhìn

Khi một khung nhìn không còn sử dụng, ta có thể xoá nó ra khỏi cơ sở dữ liệu thông qua câu lệnh:

```
DROP VIEW tên_khung_nhìn
```

Nếu một khung nhìn bị xoá, toàn bộ những quyền đã cấp phát cho người sử dụng trên khung nhìn cũng đồng thời bị xoá. Do đó, nếu ta tạo lại khung nhìn thì phải tiến hành cấp phát lại quyền cho người sử dụng.

Câu lệnh dưới đây xoá khung nhìn VIEW_LOP ra khỏi cơ sở dữ liệu

```
DROP VIEW view_lop
```

II. Hàm

Hàm do người dùng định nghĩa

Hàm là đối tượng cơ sở dữ liệu tương tự như thủ tục. Điểm khác biệt giữa hàm và thủ tục là hàm trả về một giá trị thông qua tên hàm còn thủ tục thì không. Điều này cho phép ta sử dụng hàm như là một thành phần của một biêu thức (chẳng hạn trong danh sách chọn của câu lệnh SELECT).

Ngoài những hàm do hệ quản trị cơ sở dữ liệu cung cấp sẵn, người sử dụng có thể định nghĩa thêm các hàm nhằm phục vụ cho mục đích riêng của mình.

Định nghĩa và sử dụng hàm

Hàm được định nghĩa thông qua câu lệnh CREATE FUNCTION với cú pháp như sau:

```
CREATE FUNCTION tên_hàm ([danh_sách_tham_số])
RETURNS(kiểu_trả_về_của_hàm)
AS
```

```
BEGIN các_câu_lệnh_của_hàm END
```

Câu lệnh dưới đây định nghĩa hàm tính ngày trong tuần (thứ trong tuần) của một giá trị kiểu ngày

```
CREATE FUNCTION thu(@ngay DATETIME)
RETURNS NVARCHAR(10)
AS BEGIN
DECLARE @st NVARCHAR(10)
SELECT @st=CASE DATEPART(DW,@ngay)
WHEN 1 THEN 'Chu nhật'
WHEN 2 THEN 'Thứ hai'
WHEN 3 THEN 'Thứ ba'
WHEN 4 THEN 'Thứ tư'
WHEN 5 THEN 'Thứ năm'
WHEN 6 THEN 'Thứ sáu'
ELSE 'Thứ bảy'
END
RETURN (@st) /* 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)

Câu lệnh SELECT dưới đây sử dụng hàm đã được định nghĩa ở ví dụ trước:

```
SELECT masv,hodem,ten,dbo.thu(ngaysinh),ngaysinh FROM
sinhvien WHERE malop='C24102'
```

có kết quả là:

MASV	HODEM	TEN		NGAYSINH
0241020001	Nguyễn Tuấn	Anh	Chủ nhật	1979-07-15 00:00:00
0241020002	Trần Thị Kim	Anh	Thứ năm	1982-11-04 00:00:00
0241020003	Võ Đức	Ân	Thứ hai	1982-05-24 00:00:00
0241020004	Nguyễn Công	Bình	Thứ tư	1979-06-06 00:00:00
0241020005	Nguyễn Thanh	Bình	Thứ bảy	1982-04-24 00:00:00
0241020006	Lê Thị Thanh	Châu	Thứ ba	1982-05-25 00:00:00
0241020007	Bùi Đình	Chiến	Thứ ba	1981-04-07 00:00:00
0241020008	Nguyễn Công	Chính	Chủ nhật	1981-11-01 00:00:00

Hàm với giá trị trả về là "dữ liệu kiểu bảng"

Ta đã biết được chức năng cũng như sự tiện lợi của việc sử dụng các khung nhìn trong cơ sở dữ liệu. Tuy nhiên, nếu cần phải sử dụng các tham số trong khung nhìn (chẳng hạn các tham số trong mệnh đề WHERE của câu lệnh SELECT) thì ta lại không thể thực hiện được. Điều này phần nào đó làm giảm tính linh hoạt trong việc sử dụng khung nhìn.

Xét khung nhìn được định nghĩa như sau:

```
CREATE VIEW sinhvien_k25
AS
SELECT masv,hodem,ten,ngaysinh
FROM sinhvien INNER JOIN lop ON sinhvien.malop=lop.malop
WHERE khoa=25
```

với khung nhìn trên, thông qua câu lệnh:

```
SELECT * FROM sinhvien_K25
```

ta có thể biết được danh sách các sinh viên khoá 25 một cách dễ dàng nhưng rõ ràng không thể thông qua khung nhìn này để biết được danh sách sinh viên các khoá khác do không thể sử dụng điều kiện có dạng `KHOA = @thamso` trong mệnh đề `WHERE` của câu lệnh `SELECT` được.

Nhược điểm trên của khung nhìn có thể khắc phục bằng cách sử dụng hàm với giá trị trả về dưới dạng bảng và được gọi là *hàm nội tuyến* (inline function). Việc sử dụng hàm loại này cung cấp khả năng như khung nhìn nhưng cho phép chúng ta sử dụng được các tham số và nhờ đó tính linh hoạt sẽ cao hơn.

Một hàm nội tuyến được định nghĩa bởi câu lệnh CREATE TABLE với cú pháp như sau:

```
CREATE FUNCTION tên_hàm ([danh_sách_tham_số])
RETURNS TABLE AS RETURN (câu_lệnh_select)
```

Cú pháp của hàm nội tuyến phải tuân theo các qui tắc sau:

•Kiểu trả về của hàm phải được chỉ định bởi mệnh đề RETURNS TABLE.

•Trong phần thân của hàm chỉ có duy nhất một câu lệnh RETURN xác định giá trị trả về của hàm thông qua duy nhất một câu lệnh SELECT. Ngoài ra, không sử dụng bất kỳ câu lệnh nào khác trong phần thân của hàm.

Ta định nghĩa hàm *func_XemSV* như sau

```
CREATE FUNCTION func_XemSV(@khoa SMALLINT)
RETURNS TABLE
AS RETURN(SELECT masv,hodem,ten,ngaysinh FROM sinhvien
INNER JOIN lop ON sinhvien.malop=lop.malop WHERE
khoa=@khoa)
```

hàm trên nhận tham số đầu vào là khóa của sinh viên cần xem và giá trị trả về của hàm là tập các dòng dữ liệu cho biết thông tin về các sinh viên của khoá đó. Các hàm trả về giá trị dưới dạng bảng được sử dụng như là các bảng hay khung nhìn trong các câu lệnh SQL.

Với hàm được định nghĩa như trên, để biết danh sách các sinh viên khoá 25, ta sử dụng câu lệnh như sau:

```
SELECT * FROM dbo.func_XemSV(25)
```

còn câu lệnh dưới đây cho ta biết được danh sách sinh viên khoá 26

```
SELECT * FROM dbo.func_XemSV(26)
```

Đối với hàm nội tuyến, phần thân của hàm chỉ cho phép sự xuất hiện duy nhất của câu lệnh RETURN. Trong trường hợp cần phải sử dụng đến nhiều câu lệnh trong phần thân của hàm, ta sử dụng cú pháp như sau để định nghĩa hàm:

```
CREATE FUNCTION tên_hàm([danh_sách_tham_số])
RETURNS @biến_bảng TABLE định_nghĩa_bảng
AS
BEGIN
các_câu_lệnh_trong_thân_hàm
```

```
RETURN END
```

Khi định nghĩa hàm dạng này cần lưu ý một số điểm sau:

- Cấu trúc của bảng trả về bởi hàm được xác định dựa vào định nghĩa của bảng trong mệnh đề RETURNS. Biến *@biến_bảng* trong mệnh đề RETURNS có phạm vi sử dụng trong hàm và được sử dụng như là một tên bảng.
- Câu lệnh RETURN trong thân hàm không chỉ định giá trị trả về. Giá trị trả về của hàm chính là các dòng dữ liệu trong bảng có tên là *@biếnbảng*được định nghĩa trong mệnh đề RETURNS

Cũng tương tự như hàm nội tuyến, dạng hàm này cũng được sử dụng trong các câu lệnh SQL với vai trò như bảng hay khung nhìn. Ví dụ dưới đây minh hoạ cách sử dụng dạng hàm này trong SQL.

Ta định nghĩa hàm *func_TongSV* như sau:

```
CREATE FUNCTION Func_Tongsv(@khoa SMALLINT)
RETURNS @bangthongke TABLE (makhoa NVARCHAR(5),tenkhoa
NVARCHAR(50),tongsosv INT )
AS
BEGIN
IF @khoa=0
   INSERT INTO @bangthongke
   SELECT khoa.makhoa,tenkhoa,COUNT(masv)
   FROM (khoa INNER JOIN lop ON khoa.makhoa=lop.makhoa)
   INNER JOIN sinhvien on lop.malop=sinhvien.malop
   GROUP BY khoa.makhoa,tenkhoa
ELSE
   INSERT INTO @bangthongke
   SELECT khoa.makhoa,tenkhoa,COUNT(masv) FROM (khoa
   INNER JOIN lop ON khoa.makhoa=lop.makhoa) INNER JOIN
   sinhvien ON lop.malop=sinhvien.malop
   WHERE khoa=@khoa
   GROUP BY khoa.makhoa,tenkhoa
   RETURN /*Trả kết quả về cho hàm*/
```

```
END
```

Với hàm được định nghĩa như trên, câu lệnh:

```
SELECT * FROM dbo.func_TongSV(25)
```

Sẽ cho kết quả thống kê tổng số sinh viên khoá 25 của mỗi khoa:

MAKHOA	TENKHOA	TONGSOSV
DHTO1	Khoa Toán cơ - Tin học	5
DHTO2	Khoa Công nghệ thông tin	6
DHTO3	Khoa Vật lý	6
DHTO5	Khoa Sinh học	8

MAKHOA	TENKHOA	TONGSOSV
DHTO1	Khoa Toán cơ - Tin học	15
DHTO2	Khoa Công nghệ thông tin	19
DHTO3	Khoa Vật lý	13
DHTO5	Khoa Sinh học	13

Còn câu lệnh:

```
SELECT * FROM dbo.func_TongSV(0)
```

Cho ta biết tổng số sinh viên hiện có (tất cả các khoá) của mỗi khoa

Tài liệu tham khảo:

[1]. Giáo trình SQL Server – Đỗ Ngọc Sơn, Phan Văn Viên - Tài liệu lưu hành nội bộ của Trường Đại học Công nghiệp Hà Nội, 2015.

[2]. Giáo trình hệ quản trị cơ sở dữ liệu - Đỗ Ngọc Sơn; Phan Văn Viên; Nguyễn Phương Nga - NXB Khoa học Kỹ thuật

[3]. Bài tập Hệ quản trị Cơ sở dữ liệu – Phạm Văn Hà, Trần Thanh Hùng, Đỗ Ngọc Sơn, Nguyễn Thị Thanh Huyền – Trường Đại học Công nghiệp Hà Nội, 2020.