Bài 2

NGÔN NGỮ T-SQL

Mục tiêu bài học:

- Biết các đặc điểm chính của ngôn ngữ T-SQL
- Mô tả biến, kiểu dữ liệu, và các chú thích trong T-SQL.
- Tìm hiểu hàm và biểu thức trong T-SQL
- Mô tả ngôn ngữ DML, DDL, DCL

I. Giới thiệu ngôn ngữ T-SQL

- Ngôn ngữ SQL là ngôn ngữ phổ biến nhất được sử dụng trong thế giới CSDL.

- T-SQL là việc thực thi ngôn ngữ truy vấn cấu trúc chuẩn của Microsoft.

- Ngôn ngữ T-SQL có thể dùng để định nghĩa bảng, chèn, xoá, cập nhật và truy cập dữ liệu trong bảng.

- T-SQL là ngôn ngữ khá mạnh có đề cập đến kiểu dữ liệu, đối tượng tạm thời, các thủ tục hệ thống và các thủ tục mở rộng.

- T-SQL còn có khả năng xử lý trên mẫu tin, xử lý có điều kiện, điều khiển giao tác, xử lý lỗi và biệt lệ.

1. Biến trong T-SQL

- Biến là một đối tượng có thể lưu giữ một giá trị dữ liệu. Dữ liệu có thể được chuyển đến câu lệnh T-SQL bằng việc sử dụng biến cục bộ. Biến có thể được phân thành 2 loại: biến cục bộ và toàn cục.

a. <u>Biến cục bộ</u>:

Trong T-SQL biến cục bộ được tạo và được sử dụng cho việc lưu trữ tạm thời trong khi câu lệnh SQL được thực hiện. Tên của biến cục bộ phải bắt đầu với dấu '@'

<u>Cú pháp:</u>

```
DECLARE
{
        @local_variable [AS] data_type
}
```

Trong đó:

@local_variable: xác định tên của biến, tên của biến phải bắt đầu với 1 dấu '@'.

Data_type: là kiểu dữ liệu được định nghĩa bởi người sử dụng hoặc hệ thống.

Câu lệnh **SET** hoặc **SELECT** được sử dụng để gán giá trị đến cho biến cục bộ.

Cú pháp:

```
SET @local_variable=value
OR
SELECT  @local_variable=value
```

Ví dụ:

```
DECLARE @empID int
SET @empID = 1
        SELECT FirstName, LastName
        FROM Employee
        WHERE EmployeeID=@empID
```

b. Biến toàn cục

Biến toàn cục là biến được định nghĩa và xử lý bởi hệ thống. Biến toàn cục trong SQL Server được bắt đầu với 2 dấu '@'. Giá trị của các biến này có thể được truy lục bằng câu truy vấn SELECT đơn giản.

Ví dụ:

```
SELECT @@VERSION AS sqlServerVersionDetails
```

2. Kiểu dữ liệu trong T-SQL

Kiểu dữ liệu là một thuộc tính định nghĩa dữ liệu mà một đối tượng có thể chứa. T-SQL bao gồm một số kiểu dữ liệu cơ bản như varchar, text, và int. Tất cả các dữ liệu được lưu trữ trong SQL Server phải tương thích với một trong các kiểu dữ liệu cơ bản đó.

Các đối tượng sau đây có kiểu dữ liệu:

- Cột trong table và trong view
- Tham số trong Stored Procedure
- Biến
- Hàm T-SQL trả về một hoặc nhiều giá trị dữ liệu của một kiểu dữ liệu xác định.
- Thủ tục có câu lệnh return luôn có kiểu dữ liệu integer.

Kiểu dữ liệu	**Kích thước**	**Miền giá trị dữ liệu lưu giữ**
Các kiểu dữ liệu dạng số nguyên		
Int	4 Bytes	Từ -2.147.483.648 đến +2.147.483.647
Smallint	2 bytes	Từ -32.768 đến +32.767
Tinyint	1 byte	Từ 0 đến 255
Bit	1 byte	0,1 hoặc NULL
Các kiểu dữ liệu dạng số thập phân		
Decimal, Numeric	17 bytes	Từ -10^38 đến +10^38
Các kiểu dữ liệu dạng số thực		
Float	8 bytes	Từ -1.79E +308 đến +1.79E +308
Real	4 bytes	Từ -1.79E +308 đến +1.79E +308
Các kiểu dữ liệu dạng chuỗi có độ dài cố định (fixed)		

Char	N bytes	Từ 1 đến 8000 ký tự, mỗi ký tự là 1 bytes
Các kiểu dữ liệu dạng chuỗi có độ dài biến đổi (variable)		
Varchar	N bytes	Từ 1 đến 8000 ký tự, mỗi ký tự là 1 bytes
Text	N bytes	Từ 1 đến 2.147.483.647 ký tự, mỗi ký tự là 1 bytes
Các kiểu dữ liệu dạng chuỗi dùng font chữ Unicode (national)		
Nchar	2*n bytes	Từ 1 đến 4000 ký tự, mỗi ký tự là 2 bytes
Nvarchar	2*n bytes	Từ 1 đến 4000 ký tự,mỗi ký tự là 2 bytes
Ntext	2*n bytes	Từ 1 đến 1.073.741.823 ký tự, mỗi ký tự là 2 bytes
Các kiểu dữ liệu dạng tiền tệ		
Money	8 bytes	Từ -922.337.203.685.477.5808 đến +922.337.203.685.477.5807
Smallmoney	4 bytes	Từ -214.748.3648 đến +214.748.3647
Các kiểu dữ liệu dạng ngày và giờ		
Datetime	8 bytes	Từ 1/1/1753 đến 31/12/9999
Smalldatetime	4 bytes	Từ 1/1/1900 đến 6/6/2079
Các kiểu dữ liệu dạng chuỗi nhị phân (binary string)		
Binary	N bytes	Từ 1 đến 8000 bytes
Varbinary	N bytes	Từ 1 đến 8000 bytes
Image	N bytes	Từ 1 đến 2.147.483.647 bytes

3. Ghi chú trong T-SQL

Microsoft SQL Server hỗ trợ 2 kiểu ghi chú:

-- (double hyphens). Ví dụ:

```
USE pubs
--bang Employee chua chi tiet cua nhan vien
--Cau lenh nay truy luc tat ca các hàng cua bang Employee
SELECT * FROM Employee
```

/*…….*/ (forward slash-asterisk character pairs) Ví du:

```
/* bang Employee chua chi tiet cua nhan vien. Cau lenh nay truy luc
tat ca cac hang cua bang Employee */
SELECT * FROM Employee
```

II. Hàm và biểu thức trong T-SQL

Hàm là một tập của các câu lệnh T-SQL được sử dụng để thực hiện một vài tác vụ

Biểu thức là sự kết hợp của định danh, giá trị và toán tử.

1. Hàm trong T-SQL

T-SQL bao gồm một số lượng lớn các hàm. Các hàm đó có thể hữu ích khi dữ liệu được tính toán và được xử lý.

a. <u>Các hàm tập hợp</u>

Các hàm tập hợp như **SUM, AVG, COUNT, MAX, MIN** tạo ra các giá trị tổng hợp trong kết quả truy vấn.

SQL Server hỗ trợ các hàm tập hợp sau:

- **SUM**

  Hàm SUM trả về tổng số của tất cả các giá trị của trường dữ liệu trong biểu thức. Ta có thể dùng DISTINCT với SUM để tính tổng cho các giá trị duy nhất của trường dữ liệu trong biểu thức. Các giá trị NULL được bỏ qua. SUM chỉ có thể được dùng với các trường dữ liệu kiểu số (numeric).

  ```
  Cú pháp: SUM(biểu thức)
  ```

Ví dụ: *để tìm tổng số học viên đăng ký cho khoá học có CourseCode là 1 ta dùng hàm SUM trong câu lệnh SELECT như sau:*

```
SELECT SUM(NoOfStudentsEnrolled) AS 'Enrolled Students' FROM Batch
WHERE CourseCode=1
```

Các hàm tập hợp còn có ý nghĩa khác khi dùng với mệnh đề GROUP BY. Chúng ta sẽ bàn về mệnh đề GROUP BY trong phần sau.

- AVG

Hàm AVG trả về giá trị trung bình của tất cả các giá trị của trường dữ liệu được chỉ ra trong biểu thức. AVG chỉ có thể được dùng với các trường số (numeric) và có thể tự loại bỏ các giá trị NULL.

```
Cú pháp:
AVG ([ALL | DISTINCT] ) biểu thức)
```

ALL: là giá trị mặc định, có tác dụng với tất cả các giá trị.

DISTINCT: Chỉ ra rằng AVG chỉ làm việc với một giá trị duy nhất của trường được chỉ ra, bất kể các giá trị này xuất hiện bao nhiêu lần.

Biểu thức: có thể là bất kỳ biểu thức SQL Server nào, thường là tên trường dữ liệu.

Ví dụ: *để tìm trung bình giá trị các hoá đơn trong bảng Invoice ta dùng câu truy vấn sau:*

```
SELECT AVG(Amount) AS 'Average Amount' FROM Invoice;
```

- COUNT

Hàm **COUNT** đếm được các giá trị khác NULL trong biểu thức. Nếu dùng với từ khoá DISTINCT, COUNT đếm được các giá trị duy nhất. Hàm COUNT có thể được

dùng với các trường số và ký tự. Các trường khoá chính và khoá ngoại dùng tốt nhất với hàm COUNT vì các trường này không chứa giá trị NULL.

Ta cũng có thể dùng ký tự * thay cho biểu thức trong hàm COUNT. Với cách dùng này ta sẽ đếm tất cả các mẫu tin mà không quan tâm đến bất kỳ trường dữ liệu nào.

```
Cú pháp: COUNT (biểu thức) hay COUNT(*)
```

Ví dụ: *để đếm số khoá học trong bảng Course ta dùng:*

```
SELECT COUNT(CourseCode) AS 'Total Course' FROM Course;
```

- MAX

Hàm MAX trả về giá trị lớn nhất trong biểu thức. Hàm MAX có thể được dùng với các kiểu dữ liệu số, chuỗi, và ngày tháng. MAX trả về giá trị lớn nhất trong toàn bộ giá trị sau khi đã đối chiếu. MAX bỏ qua các giá trị NULL.

```
Cú pháp: MAX(biểu thức)
```

Ví dụ: *để tìm giá trị hoá đơn lớn nhất trong bảng Invoice ta dùng:*

```
SELECT MAX(Amount) AS 'Maximum Invoice Amount' FROM Invoice;
```

- MIN

Hàm MIN trả về giá trị nhỏ nhất trong biểu thức. Hàm này có thể được dùng với các trường số, chuỗi và ngày tháng. Khi MIN được dùng với các trường kiểu chuỗi, MIN trả về giá trị nhỏ nhất trong danh sách so sánh. MIN bỏ qua giá trị NULL.

```
Cú pháp: MIN(biểu thức)
```

Ví dụ: *để tìm giá trị hoá đơn nhỏ nhất trong bảng Invoice ta dùng truy vấn sau:*

```
SELECT MIN(Amount) AS ‘Minimum Invoice Amount’ FROM Invoice
```

b. <u>Các hàm xử lý chuỗi</u>

- Hàm ASCII: hàm trả về giá trị mã ASCII của ký tự bên trái của chuỗi.

Ví dụ:

Print ASCII(‘TOI’)

Kết quả trả về như sau:

84

- Hàm char: hàm này chuyển đổi kiểu mã ASCII từ số nguyên sang dạng chuỗi.

Ví dụ:

Print char(35)

Kết quả trả về như sau:

#

- Hàm UPPER: hàm này chuyển đổi chuỗi sang kiểu chữ hoa.

Ví dụ:

Print UPPER(‘Quyen’)

Kết quả trả về như sau:

QUYEN

- Hàm LOWER: hàm này chuyển đổi chuỗi sang kiểu chữ thường.

Ví dụ:

Print LOWER(QUYEN)

Kết quả trả về như sau:

quyen

- Hàm len: hàmnày trả về chiều dài của chuỗi.

Ví dụ:

Print len('Ly Tu Trong')

Kết quả trả về như sau:

11

- Hàm LTRIM: loại bỏ khỏang trắng bên trái của chuỗi.

Ví dụ:

Print LTRIM(' Trong')

Kết quả trả về như sau:

'Trong'

- Hàm RTRIM: loại bỏ khoảng trắng bên phải của chuỗi.

Ví dụ:

Print RTRIM('LyTuTrong ')

Kết quả trả về như sau:

'LyTuTrong'

- Hàm Left: trả về chuỗi bên trái tính từ đầu cho đến vị trí thứ n.

Ví dụ:

Print left('Quyen',3)

Kết quả trả về như sau:

'Quy'

- Hàm Right: trả về chuỗi bên phải tính từ cuối cho đến vị trí thứ n.

Ví dụ:

Print Right('QUyen',4)

Kết quả trả về như sau:

'Uyen'

- Hàm CHARINDEX: trả về vị trí chuỗi bắt đầu của chuỗi con trong chuỗi xét.

Ví dụ:

Print CHARINDEX('Trong','Ly Tu Trong')

Kết quả trả về như sau:

7

c. <u>Các hàm về xử lý thời gian</u>

- Hàm getDate(): trả về ngày tháng năm của hệ thống

Ví dụ:

Select 'Today is '=getDate()

Kết quả trả về như sau:

Today is

2007-10-17 14:55:20

- Hàm DATEPART: trả về một phần giá trị của một chuỗi dạng ngày tháng đầy đủ.

Ví dụ 1:

Select 'Hom nay ngay: '=datepart(d,getDate())

Kết quả trả về như sau:

Hom nay ngày:

17

Ví dụ 2:

Select 'Hom nay tuan: '=datepart(w,getDate())

Kết quả trả về như sau:

Hom nay tuan:

- - - - - - - - - - - - - - -

4

Hàm này bao gồm các hệ số như trong bảng sau:

Hàm DATEPART	**Tham số**
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
Miliecond	ms

- Hàm DATEDIFF: trả về số ngày trong khỏang thời gian giữa ngày.

Ví dụ:

Select 'So ngay giua ngay thu tien den hom nay: '=datediff(d, ReceiptDate, getDate())

From tblReceipt

Kết quả trả về như sau:

So ngay giua ngay thu tien den hom nay:

- - - - - - - - - - - - - - - - - - -- - - -- - - - - - - -

74

72

- Hàm day: trả về ngày thứ mấy trong tháng.

Ví dụ:

Select 'Hom nay ngay: '=day(getDate())

Kết quả trả về như sau:

21

- Hàm month: trả về tháng thứ mấy trong năm.

Ví dụ:

Select 'Hom nay thang:'=month(getDate())

Kết quả trả về như sau:

10

- Hàm year: trả về năm

Ví dụ:

Select 'Nam nay la:'=year(getDate())

Kết quả trả về như sau:

2007

d. <u>Các hàm về toán học</u>

- Hàm square: trả về bình phương của một biểu thức

Ví dụ:

Print square(4)

Kết quả trả về như sau:

16

- Hàm sqrt: trả về căn bậc hai của một biểu thức

Ví dụ:

Print sqrt(4)

Kết quả trả về như sau:

2

- Hàm round: trả về số làm tròn của một biểu thức.

Ví dụ:

Print round(748.58, -1)

Kết quả trả về như sau:

750.00

e. Các hàm về chuyển đổi

- Hàm cast: trả về giá trị có kiểu dữ liệu theo định nghĩa

Ví dụ:

Print cast(getDate() as varchar(11))

Kết quả trả về như sau:

Oct 17 2007

Kết quả bao gồm 11 ký tự.

- Hàm convert: chuyển đổi giá trị có kiểu dữ liệu này sang kiểu dữ liệu khác nếu cho phép.

Ví dụ:

Print convert(int, '12')

Kết quả trả về la số nguyên có gia trị như sau:

12

Hoặc chúng ta có thể dùng hàm này để lấy thời gian từ biến hay cột, giá trị có dạng ngày tháng như sau:

Print convert(char(10), getDate(), 108)

Kết quả trả về như sau:

12 : 07 : 46

2. Biểu thức trong T-SQL

a. Biểu thức điều kiện

Như chúng ta đã biết T-SQL là một tập hợp của việc lập trình mở rộng từ Microsoft. Có có thể lập trình các chức năng trong CSDL quan hệ được cung cấp bởi SQL Server 2000.

Mệnh đề WHERE được sử dụng như một phần của một câu lệnh T-SQL để lọc dữ liệu.Trong mệnh đề WHERE thường sử dụng các toán tử so sánh và các biểu thức điều kiện. Biểu thức là sự kết hợp của các toán tử và toấn hạng. Các biểu thức đơn giản có thể là một biến đơn, hằng đơn, cột đơn,… Toán tử có thể được sử dụng để kết hợp 2 hay nhiều biểu thức đơn thành một biểu thức phức tạp.

Một biểu thức có thể bao gồm một hay nhiều loại như sau:

- Hằng: thể hiện một giá trị dữ liệu đơn và giá trị dữ liệu đó phải cụ thể.
- Cột: Tên của cột trong một bảng. Chỉ tên của cột được cho phép trong một biểu thức.
- Toán tử một ngôi: Toán tử này chỉ có một toán hạng tham gia vào.
- Toán tử hai ngôi: toán tử này định nghĩa cách thức mà 2 biểu thức kết hợp tạo ra một kết quả đơn. Toán tử 2 ngôi có thể là một toán tử số học, toán tử gán (=), toán tử bitwise, toán tử so sánh, toán tử logic, …

Bảng dưới đây liệt kê các toán tử so sánh được sử dụng trong câu lệnh T-SQL trong mệnh đề WHERE

Toán tử	**Mô tả**
=	Toán tử bằng
>	Lớn hơn
<	Nhỏ hơn
>=	Lớn hơn và bằng
<=	Nhỏ hơn và bằng
< >	Không bằng
!	Phủ định

Ví dụ:

PRICE >100

NAME LIKE 'DAVID'

GRADE < > 'FAIL'

Các ký tự đại diện trong câu lệnh T-SQL

Ký tự đại diện	**Mô tả**	**Ví dụ**
'-'	Một ký tự đơn	SELECT udesc FROM usertypes WHERE udesc LIKE 'C_'
%	chiều dài bất kỳ một chuỗi	SELECT * FROM users WHERE username LIKE "AL%"
[ ]	Một ký tự đơn trong phạm vi một cặp dấu ngoặc vuông	SELECT * FROM coursematerial WHERE yearsem LIKE 'SEM[1-2]'
[^]	Nhiều ký tự đơn mà không nằm trong phạm vi cặp dấu ngoặc đơn.	SELECT * FROM coursematerial WHERE yearsem LIKE 'sem[^ 1-2]'

b. <u>Toán tử luận lý</u>

Các toán tử luận lý được hỗ trợ trong T-SQL là AND, OR, NOT

Các toán tử AND và OR được sử dụng để giúp tìm kiếm các điều kiện trong mệnh đề WHERE. NOT là toán tử phủ định. AND kết nối hai điều kiện và trả về TRUE chỉ khi cả 2 điều kiện là TRUE. OR kết nối 2 điều kiện nhưng nó chỉ trả về TRUE khi một trong 2 điều kiện là TRUE.

Độ ưu tiên của toán tử luận lý là: NOT, AND, OR

III. Ngôn ngữ DDL

1. Câu lệnh CREATE TABLE

Một bảng có thể được tạo bằng việc sử dụng một trong 2 cách sau đây:

- Enterprise Manager
- Các câu lệnh T-SQL trong Query Analyzer

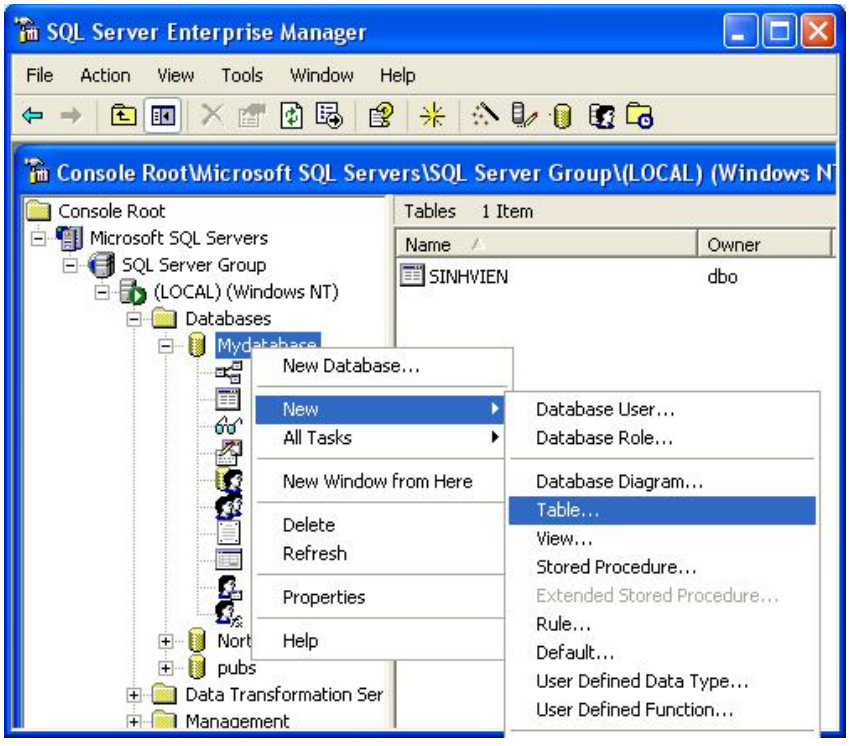
Cú pháp để tạo bảng bằng T-SQL là:

```
CREATE TABLE <Table_Name>
(<Column_Name>  <Data_Type>)
```

Ví dụ:

```
CREATE  TABLE  SinhVien
( MSSV  Char(4), HOTENSV  varchar(15))
```

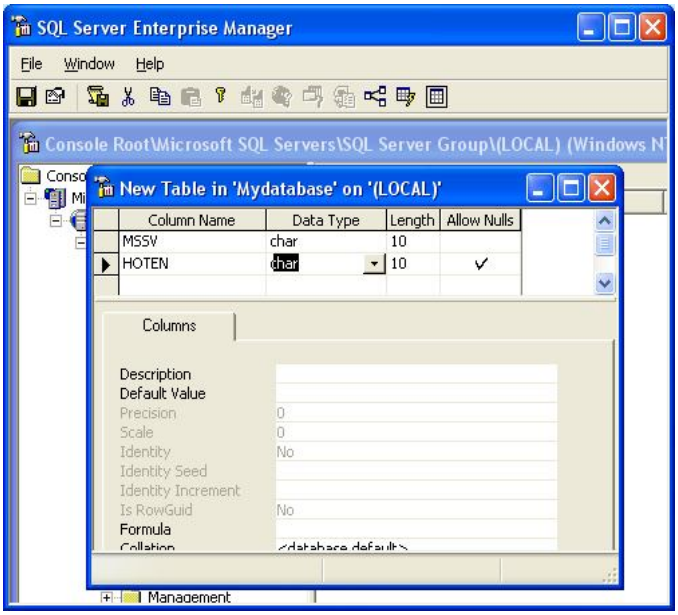
Chúng ta có thể tạo một bảng mới với Enterprise Manager bằng việc chọn CSDL và sau đó chọn tuỳ chọn Action, New và Table.



Khi chọn tuỳ chọn Table, cửa sổ thiết kế (Design) đổ xuống, chúng ta có thể gõ vào trong cấu trúc bảng này.

Ví dụ:

Chúng ta muốn tạo bảng tên là SINHVIEN có 2 field MSSV và HOTENSV. Chúng ta gõ tên field, kích thước, và kiểu dữ liệu cho 2 field đó và sau đó lưu bảng với tên là SINHVIEN.



- Tạo primary key ngay sau khi khai báo kiểu dữ liệu của cột đó trong khi tạo bảng

```
CREATE TABLE SINHVIEN
(
        MSSV char(4) NOT NULL primary key, HOTEN varchar(20)
)
```

- Tạo ràng buộc khóa ngoại trong khi tạo bảng

```
CREATE TABLE SINHVIEN
(
        MSSV char(4) NOT NULL primary key, HOTEN varchar(20), MALOP
```

```
char(4) foreign key (MALOP) references LOP(MALOP)
)
```

- Tạo ràng buộc Default trong khi tạo bảng

```
CREATE TABLE SINHVIEN
(
    MSSV char(4), HOTEN varchar(20), NGAYSINH datetime, PHAI bit DEFAULT 0
)
```

- Tạo ràng buộc duy nhất trong khi tạo bảng

```
CREATE TABLE SINHVIEN
(
    MSSV char(4), HOTEN varchar(20), DIENTHOAI varchar(15) NOT NULL UNIQUE
)
```

2. Câu lệnh ALTER TABLE

- Lệnh Alter table được sử dụng để sửa đổi việc định nghĩa một bảng bằng việc thay đổi, thêm, hoặc xoá các cột và các ràng buộc.

Cú pháp:

```
ALTER TABLE <table name>
    ALTER COLUMN [<Column name> <New_data_type>] | ADD [<Column_name> <Data_type>] | DROP COLUMN [<Column_Name>]
```

Ví dụ 1: **Thêm cột DIACHI vào bảng SINHVIEN**

```
ALTER TABLE SINHVIEN
```

```
ADD DIACHI varchar(20) NULL
```

Ví dụ 2: **Thay đổi kiểu dữ liệu mới cho cột HOTEN của bảng SINHVIEN**

```
ALTER TABLE SINHVIEN
ALTER COLUMN  HOTEN nvarchar(30) NULL
```

Ví dụ 3: **Xóa cột DIACHI trong bảng SINHVIEN**

```
ALTER TABLE SINHVIEN
DROP COLUMN  DIACHI
```

- Tạo primary key cho cột trong bảng đã tồn tại

```
ALTER TABLE SINHVIEN
ADD CONSTRAINT PK_MSSV
PRIMARY KEY (MSSV)
```

- Tạo ràng buộc khóa ngoại trong bảng đã tồn tại

```
ALTER table SINHVIEN
Add constraint FK_MALOP
foreign key (MALOP) references LOP(MALOP)
```

- Tạo ràng buộc check trong bảng có dữ liệu

```
ALTER table SINHVIEN
ADD CONTRAINT CK_NGAYSINH
CHECK (NGAYSINH<=getDate())
```

- Thêm giá trị mặc nhiên trong bảng tồn tại dữ liệu

```
ALTER TABLE SINHVIEN
ADD CONSTRAINT DEF_PHAI
```

```
DEFAULT 0 for PHAI
```

- Tạo ràng buộc duy nhất trong bảng đã tồn tại

```
ALTER TABLE SINHVIEN
(       ADD CONSTRAINT UK_DIENTHOAI
        UNIQUE(DIENTHOAI)
)
```

3. Câu lệnh DROP TABLE

Câu lệnh DROP TABLE xoá việc định nghĩa một bảng và tất cả các dữ liệu, chỉ mục, trigger, ràng buộc, và việc xác định các quyền cho bảng đó.

Cú pháp:

DROP TABLE <table_name>

Ví dụ:

DROP TABLE Employee

Chú ý: Câu lệnh DROP TABLE không thể được dùng để xoá một bảng mà bảng này đang có mối quan hệ với bảng khác. Trước khi xoá phải quan tâm đến ràng buộc tham chiếu khoá ngoại.

IV. Ngôn ngữ DML và DCL

1. Câu lệnh SELECT, INSERT, DELETE, UPDATE

a. Dùng T-SQL để Insert

Chúng ta có thể thêm dữ liệu mới vào một bảng hoặc chúng ta cũng có thể thêm dữ liệu từ một bảng đang tồn tại vào một bảng mới được tạo ra.

Cú pháp câu lệnh Insert để thêm dữ liệu vào một bảng :

```
INSERT [INTO] <table_name> VALUES <values>
```

Trong đó:

[INTO] là một từ khoá

<table_name> tên của bảng nơi mà record được chèn vào.

<values> xác định các giá trị cho cột của bảng.

Ví dụ:

```
INSERT INTO jobs VALUES ('Graphics Artist', 25, 100)
```

Bảng jobs bao gồm 3 fields: job_desc, min_lvl, max_lvl.

Cú pháp để thêm dữ liệu từ một bảng vào một bảng khác

```
INSERT <table_name> SELECT <column list> FROM <tablename2>
```

Trong đó: table name là tên của bảng nơi mà chúng ta có thể thêm dữ liệu vào , column list là danh sách các cột được chèn vào từ một bảng đang tồn tại và tablename2 là tên của bảng đang tồn tại.

Ví dụ:

```
CREATE TABLE author_details (au_id varchar(11), au_lname varchar(40))
GO
INSERT author_details SELECT authors.au_id, authors.au_lname FROM authors
```

b. Dùng T-SQL để update

Chúng ta có thể thay đổi, cập nhật dữ liệu trong một bảng bằng câu lệnh UPDATE. Việc cập nhật thì cần thiết khi dữ liệu cần được thay đổi.

Cú pháp cho việc cập nhật một hàng đơn trong một bảng :

```
UPDATE <table_name>
SET <column_name=value>
[WHERE <search condition>]
```

Tham số được giải thích như sau:

<table_name> là tên của bảng nơi mà mẫu tin được cập nhật.

<value> xác định các giá trị mới cho các cột được thay đổi.

```
UPDATE titles SET price=price+(25/100*price) WHERE title_id='TC7777'
```

Cập nhật nhiều hàng của một bảng

```
UPDATE publishers
SET city='Atlanta', state='GA'
```

c. Dùng T-SQL để delete:

Cú pháp để xoá một hàng từ một bảng

```
DELETE FROM <table_name> [WHERE <search_condition>]
```

Trong đó:

<table_name> là tên của bảng chứa mẫu tin muốn xoá.

mệnh đề WHERE được sử dụng để xác định điều kiện

Ví dụ :

```
DELETE FROM pub_info WHERE pub_id=9999
```

d. Dùng T-SQL để SELECT:

Có 5 phần cơ bản để trình bày một câu lệnh SELECT đơn giản:

Cú pháp:

```
SELECT <columnname(s)> FROM <tablename>;
```

Trong đó:

- SELECT là từ đầu tiên trong câu query.
- <columnname(s)> phải là thành phần kế tiếp trong câu query. Nếu muốn truy lục từ 2 hay nhiều cột trong câu SELECT thì các columnname phải được cách nhau bởi dấu chấm và các cột đó phải tồn tại trong bảng mà chúng ta đang truy lục.
- FROM phải là từ kế tiếp trong câu query của chúng ta.
- <tablename> là tên bảng mà chúng ta đang truy lục.

- Dấu ";" là thành phần cuối cùng trong câu query. Dấu ";" nói cho SQL biết rằng câu query đã hoàn thành và bây giờ nên thực thi.

Câu lệnh SELECT được thông dịch như sau:

SELECT (cái gì?)	Một hay nhiều tên cột . Tên cột phải được tách ra bởi dấu phẩy. Chú ý rằng tên cột cuối cùng không có dấu phẩy đi kèm.
FROM (đâu?)	Tên bảng.

```
SELECT * FROM <tablename>
```

SELECT có nhiều chức năng. Chúng ta chỉ vừa nhìn thấy một dạng cơ bản mà thôi. Có một số các chức năng khác như sau:

- **WHERE:** Đây là một phần tùy chọn của query. Phần này xác định một điều kiện. Nếu một query không có mệnh đề WHERE thì tất cả các hàng trong bảng sẽ được hiển thị. Câu lệnh so sánh cũng có thể được xác định trong mệnh đề này. Ví dụ: để hiển thị tên của sinh viên khi biết mã số sinh viên ta làm như sau:

```
SELECT hosv
tensv FROM sinhvien WHERE masv="ltt200603";
```

- **GROUP BY:** Đây là một chức năng khác của câu query. Nó chỉ được sử dụng khi kết quả của query phải được nhóm dựa trên điều kiện. Ví dụ: nếu chúng ta muốn hiển thị thông tin của sinh viên từ những mã số sinh viên khác nhau được nhóm lại bằng mệnh đề GROUP BY như sau:

```
SELECT masv, hosv, tensv
FROM sinhvien
 GROUP BY masv;
```

- **ORDER BY:** Chức năng này của câu query cho phép sắp xếp các hàng theo thứ tự. Ví dụ: nếu chúng ta muốn hiển thị thông tin của sinh viên có mã số "ltt200603" trong bảng sinhvien được sắp xếp theo họ của sinh viên, ta làm như sau:

```
SELECT * FROM sinhvien WHERE masv="ltt200603" ORDER BY hosv;
```

Câu lệnh SELECT đơn giản

Câu lệnh đơn giản nhất để thể hiện tất cả các cột trong một bảng được ghi như sau:

```
SELECT * FROM sinhvien;
```

Câu lệnh trên sẽ hiển thị tất cả các hàng và cột trong bảng sinhvien. cột sẽ hiển thị theo thứ tự như ban đầu khi chúng ta thiết kế bảng.

Các bảng trong CSDL thì thường lớn và chứa nhiều hàng dữ liệu. Thật vậy, đôi lúc chúng ta không cần thiết phải hiển thị tất cả các hàng trong một bảng như là kết quả của câu query trên. Chúng ta có thể sử dụng mệnh đề **WHERE** để lọc các dữ liệu theo một điều kiện cho trước.

Điều kiện được xác định trong mệnh đề WHERE được gọi là một thuộc tính. Ví dụ: Xét câu query sau:

```
SELECT * FROM sinhvien WHERE phai="Nu";
```

Mệnh đề **WHERE** được sử dụng để giới hạn kết quả của query, chỉ hiển thị những bản ghi có cột **phái ="Nu"**. Khi câu lệnh được thực thi, mỗi mẫu tin trong bảng sẽ được so sánh với giá trị trong điều kiện. Chỉ những mẫu tin có giá trị thoả mãn điều kiện mới

được hiển thị. Cần chú ý rằng cột dữ liệu được dùng trong mệnh đề điều kiện **WHERE** không nhất thiết phải được hiển thị trong kết quả truy vấn.

Cú pháp:

```
SELECT <columnname> FROM <tablename>
WHERE <điều kiện>
```

Mệnh đề điều kiện có thể chứa bất kỳ kiểu dữ liệu nào như ký tự, số hay ngày tháng. Các toán tử logic như **AND**, **OR**, và **NOT** có thể được dùng kết hợp trong mệnh đề điều kiện. ***Ví dụ, để truy xuất thông tin sinh viên thuộc phái nữ và có nơi sinh="TPHCM", ta viết câu lệnh như sau:***

```
SELECT masv,hosv, tensv, sodt, phai, noisinh FROM sinhvien
WHERE phai="Nu" and noisinh="TPHCM";
```

Các toán tử quan hệ cũng có thể được dùng trong mệnh đề điều kiện để xây dựng điều kiện chi tiết hơn. Một mệnh đề điều kiện có thể là sự kết hợp của các loại toán tử như logic và toán tử quan hệ.

Nếu chúng ta muốn các mẫu tin trong bảng hiển thị theo thứ tự cụ thể, ta dùng mệnh đề **ORDER BY** sắp xếp kết quả truy vấn. Nó có thể sắp xếp kết quả tăng dần (ASC) hay giảm dần (DESC). Mặc định các mẫu tin được sắp xếp tăng dần.

Câu lệnh SELECT sau sắp xếp dữ liệu trong bảng sinh viên theo trường mã số sinh viên.

```
SELECT * FROM sinhvien
ORDER BY masv;
```

Trong ví dụ dưới đây, nếu ta muốn liệt kê danh sách các sinh viên có nơi sinh là TPHCM và sắp xếp theo mã sinh viên từ bảng sinh viên, ta dùng truy vấn sau:

```
SELECT masv, hosv, tensv, noisinh
FROM sinhvien
WHERE noisinh=”TPHCM”
ORDER BY masv;
```

Thứ tự sắp xếp có thể là tăng dần hay giảm dần. Dùng các tham số **DESC** hay **ASC** ta có thể sắp xếp các mẫu tin theo thứ tự yêu cầu. Nếu không có tham số nào trong câu lệnh truy vấn, thứ tự sắp xếp là tăng dần.

Ta có thể kết hợp các trường dữ liệu với các hằng số kiểu chuỗi để có kết quả dễ đọc và định dạng hợp lý. Thông thường các hằng số này không tồn tại như một trường dữ liệu trong kết quả truy vấn mà được kết hợp với các trường khác để xây dựng kết quả hiển thị cho truy vấn.

Ví dụ: để thêm vào các ký tự “:” và “->” trong kết quả truy vấn dữ liệu từ bảng titles ta tạo truy vấn như sau:

```
SELECT title_id +’:’ + title + ‘->’ +type
FROM titles;
```

Chú ý: khi dùng toán tử + trong danh sách truy vấn, ta cần chú ý đến kiểu dữ liệu của các trường. Các trường là toán hạng của toán tử này phải có cùng kiểu dữ liệu, nếu không SQL Server sẽ báo lỗi.

Mệnh đề AS có thể được dùng để thay đổi tiêu đề của trường dữ liệu hay gán tiêu đề cho trường phát sinh trong kết quả của câu truy vấn. Các trường hiển thị trong kết quả của câu truy vấn thường có tiêu đề là tên trường trong bảng. Để có tiêu đề dễ hiểu hơn ta dùng mệnh đề AS để tuỳ biến.

Ví dụ: để hiển thị tiêu đề “ROLL NUMBER” thay cho trường RollNo trong bảng Students ta dùng câu lệnh SELECT sau:

```
SELECT RollNo AS 'ROLL NUMBER' FROM Students
```

Các câu lệnh SELECT có thể được dùng với các ràng buộc.

Ví dụ: ràng buộc IDENTITY có thể được sử dụng trong truy vấn như sau:

```
SELECT IDENTITY (datatype, seed, increment) AS ID_Number
INTO table2 FROM table1
```

Trong câu truy vấn trên **table1** là bảng có sẵn và **table2** là bảng mà ta muốn tạo ra và đưa các giá trị tự tăng vào.

Datatype là kiểu dữ liệu của trường tự tăng. Trường này chỉ nhận các kiểu dữ liệu integer hay kiểu dữ liệu decimal.

Seed là giá trị gán cho mẫu tin đầu tiên của bảng. Các mẫu tin tiếp theo được gán các giá trị tăng dần, các giá trị này được tính bằng tổng các giá trị IDENTITY cuối cùng cộng với giá trị tăng.

Increment là giá trị tăng được cộng thêm cho cac mẫu tin tiếp theo của bảng dữ liệu.

Từ khoá DISTINCT loại bỏ các mẫu tin trùng lắp từ truy vấn. Nếu không có từ khoá DISTINCT, truy vấn sẽ trả về tất cả các mẫu tin bao gồm cả các mẫu tin trùng lắp.

Ví dụ: nếu chúng ta truy xuất trường MaterialType trong bảng Material không dùng từ khoá DISTINCT ta sẽ nhận được một danh sách dài các giá trị MaterialType trùng lắp. Nếu dùng DISTINCT trong truy vấn SQL Server sẽ chỉ trả về các giá trị MaterialType duy nhất. Câu truy vấn này như sau:

```
SELECT DISTINCT MaterialType FROM Material
```

Mệnh đề TOP được dùng để hạn chế số mẫu tin trả về trong kết quả truy vấn.

Câu lệnh SELECT này có cú pháp như sau:

```
SELECT TOP n <tên trường> FROM <tên bảng>
```

Ví dụ: Để hiển thị 3 mẫu tin đầu tiên trong bảng Students ta dùng:

```
SELECT TOP 3 * FROM Students
```

Ký tự * trong câu truy vấn truy xuất tất cả các trường.

Tương tự như vậy ta có thể truy xuất số mẫu tin theo tỷ lệ phần trăm từ bảng dữ liệu.

Ví dụ:

```
SELECT TOP 40 PERCENT * FROM Students
```

sẽ truy vấn 40% mẫu tin đầu tiên của bảng Students.

2. Câu lệnh GRANT, REVOKE, DENY

a. Câu lệnh GRANT

Câu lệnh GRANT được sử dụng khi cơ sở dữ liệu được chia sẻ với những người sử dụng khác. Câu lệnh GRANT được dùng để cấp quyền truy cập dữ liệu cho người sử dụng.

Cú pháp:

```
GRANT
{ALL | statement [, ….]}
ON table_name
TO Security_Account[, ……]
```

Trong đó:

ALL chuyển tất cả các quyền cho đối tượng khi ALL được sử dụng để cấp quyền cho người sử dụng trên một đối tượng.

Statement đề cập đến câu lệnh mà quyền được cho bởi người sử dụng.

TO Security_Account xác định quyền cho một người sử dụng đặc biệt nào đó. Người sử dụng đó có thể là Microsoft SQL Server, một SQL Server role, một Windows OS….

Table_name đề cập đến tên của bảng mà được cấp quyền.

Ví dụ:

```
GRANT SELECT ON Employee TO QUYEN
```

b. Câu lệnh REVOKE

Câu lệnh revoke được dùng để huỷ quyền người sử dụng được cấp bởi câu lệnh grant

Cú pháp:

```
REVOKE
{ALL | statement[,…]}
ON table_name
FROM Security_Account[,….]
```

Ví dụ:

```
REVOKE SELECT ON Employee FROM QUYEN
```

c. Câu lệnh DENY

Cú pháp:

```
DENY
{ALL | statement[,…]}
ON table_name
TO Security_Account[,….]
```

Ví dụ:

```
DENY SELECT ON Employee TO QUYEN
```