GIỚI THIỆU

SQL (Structured Query Language) là ngôn ngữ được sử dụng cho các hệ quản trị cơ sổ dữ liệu quan hệ. Ngôn ngữ SQL chuẩn được đưa ra bởi ANSI (American National Standards Institude) và ISO (International Standards Organization) với phiên bản mới nhất hiện nay là phiên bản SQL-92 (phiên bản được đưa ra năm 1992). Mặc dù có nhiều ngôn ngữ khác nhau được đưa ra cho các hệ quản trị CSDL quan hệ, SQL là ngôn ngữ được sử dụng rộng rãi hiện nay trong rất nhiều hệ thống CSDL thương mại như Oracle, SQL Server, DB2, Microsoft Access... Thông qua SQL, người sử dụng có thể dễ dàng định nghĩa được dữ liệu, thao tác với dữ liệu,... Mặt khác, đây là ngôn ngữ có tính khai báo nên nó dễ sử dụng và cũng vì vậy mà trổ nên phổ biến

Giáo trinh này nhằm cung cấp cho bạn tài liệu tham khảo tương đối đầy đủ về các câu lệnh giao tác SQL sử dụng cho hệ quản trị CSDL Microsoft SQL Server. Giáo trình bao gồm bốn chương:

- Chương 1 giới thiệu một số câu lệnh sử dụng trong việc định nghĩa các đối tượng dữ liệu như bảng dữ liệu, khung nhìn và chỉ mục.

- Chương 2 trình bày bốn câu lệnh thao tác dữ liệu là SELECT, INSERT, UPDATE và DELETE; trong đó tập trung chủ yếu ở câu lệnh SELECT.

- Chương 3 đề cập đến hai câu lệnh điều khiển là GRANT và REVOKE sử dụng trong việc cấp phát và huỷ bỏ quyền của người sử dụng CSDL.

- Chương 4 giới thiệu về thủ tục lưu trữ và trigger. Đây là những đối tượng CSDL được sử dụng nhằm tăng hiệu năng khi sử dụng CSDL.

- Trong chương phụ lục, chúng tôi giới thiệu cấu trúc và dữ liệu của các bảng sử dụng trong các ví dụ ở chương 2 để bạn tiện tra cứu và đối chiếu với các ví dụ đã nêu. Ngoài ra trong chương này còn có các hàm thường sử dụng trong SQL Server đề các bạn tham khảo trong thực hành.

Mặc dù đã rất cố gắng nhưng giáo trình không thể tránh được các sai sót. Rất mong nhận được sự góp ý của các bạn để giáo trình ngày càng hoàn thiện hơn.

Chương 1: NGÔN NGỮ ĐỊNH NGHĨA DỮ LIỆU

Ngôn ngữ định nghĩa dữ liệu bao gồm các câu lệnh cho phép người sử dụng định nghĩa CSDL và các đối tượng trong CSDL như các bảng, các khung nhìn, chỉ mục,...

1. Tạo bảng dữ liệu

Dữ liệu bên trong một CSDL được tổ chức lưu trữ trong các bảng. Bên trong các bảng, dữ liệu được tổ chức dưới dạng các dòng và các cột. Mỗi một dòng biểu diễn một bản ghi duy nhất và mỗi một cột biểu diễn cho một trường.

1.1 Các thuộc tính liên quan đến bảng

Khi tạo và làm việc với các bảng dữ liệu, ta cần phải để ý đến các thuộc tính khác trên bảng như: kiểu dữ liệu, các ràng buộc, các khoá, các qui tắc,... Các thuộc tính này được sử dụng nhằm tạo ra các ràng buộc toàn vẹn trên các cột (trường), trên bảng cũng như tạo ra các toàn vẹn tham chiếu giữa các bảng dữ liệu trong CSDL.

a. Kiểu dữ liệu

Mỗi một cột (trường) của một bảng đều phải thuộc vào một kiểu dữ liệu nhất định đã được định nghĩa từ trước. Mỗi một kiểu dữ liệu qui định các giá trị dữ liệu được cho phép đối với cột đó. Các hệ quản trị CSDL thường cung cấp các kiểu dữ liệu chuẩn, ngoài ra còn có thể cho phép người sử dụng định nghĩa các kiểu dữ liệu khác dựa trên các kiểu dữ liệu đã có.

Dưới đây là một số kiểu dữ liệu thường được sử dụng trong giao tác SQL:

Binary	Int	Smallint
Bit	Money	Smallmoney
Char	Nchar	Text
Datetime	Ntext	Tinyint
Decimal	Nvarchar	Varbinary
Float	Real	Varchar
Image	Smalldatetime	

b. Các ràng buộc (CONSTRAINTS)

Trên các bảng dữ liệu, các ràng buộc được sử dụng nhằm các mục đích sau:

- Qui định các giá trị dữ liệu hay khuôn dạng dữ liệu được cho phép chấp nhận trên các cột của bảng (ràng buộc CHECK)
- Qui định giá trị mặc định cho các cột (ràng buộc DEFAULT).

- Tạo nên tính toàn vẹn thực thể trong một bảng dữ liệu và toàn vẹn tham chiếu giữa các bảng dữ liệu trong CSDL (ràng buộc PRIMARY KEY, UNIQUE và FOREIGN KEY).

Chúng ta sẽ tìm hiểu chi tiết hơn về các ràng buộc này ở phần trình bày về câu lệnh CREATE TABLE.

1.2 Tạo bảng bằng truy vấn SQL

Tạo các bảng là một khâu quan trong trọng trong quá trình thiết kế và cài dặt các CSDL. Bên trong các CSDL, mỗi một bảng thường được sử dụng nhằm biểu diễn thông thi về các đối tượng trong thế giới thực và/hoặc biểu diễn mối quan hệ giữa các đối tượng đó. Để có thể tổ chức tốt một bảng dữ liệu, bạn ít nhất cần phải xác định được các yêu cầu sau:

- Bảng được sử dụng nhằm mục đích gì và có vai trò như thế nào bên trong CSDL?
- Bảng sẽ bao gồm những cột nào và kiểu dữ liệu cho các cột đó là gì?
- Những cột nào cho phép chấp nhận giá trị NULL.
- Có sử dụng các ràng buộc, các mặc định hay không và nếu có thì sử dụng ở đâu và nhu thế nào?
- Những cột nào sẽ đóng vai trò là khoá chính, khoá ngoài, khoá duy nhất? Những dạng chỉ mục nào là cần thiết và cần ở đâu.

a. Tạo bảng dữ liệu:

Để tạo một bảng trong CSDL, bạn sử dụng câu lệnh CREATE TABLE có cú pháp như sau:

```
CREATE TABLE  table_name
(
  { colname_1  col_1_properties   [ constraints_1 ]
   [ ,{ colname_2 col_2_properties  [ constraints_2 ]]
    ...
   [ ,{ colname_N col_N_properties  [ constraints_N ]]
   [ table_constraints]
)
```

Trong đó:

- `table_name:` Tên bảng cần tạo. Tên của bảng phải duy nhất trong mỗi CSDL và phải tuân theo các qui tắc về định danh
- `colname_i:` Tên của cột thứ i trong bảng. Các cột trong mỗi

	bảng phải có tên khác nhau và phải tuân theo các qui tắc về định danh. Mỗi một bảng phải có ít nhất một cột
- col_i_properties:	Các thuộc tính của cột thứ i qui định kiểu dữ liệu của cột và chỉ định cột có cho phép chấp nhận giá trị NULL hay không
- constraints_i:	Các ràng buộc (nếu có) trên cột thứ i như các ràng buộc về khoá, các mặc định, các qui định về khuôn dạng dữ liệu.
- table_constraint:	Các ràng buộc trên bảng dữ liệu.

Ví dụ 1.1: Câu lệnh dưới đây thực hiệnh việc tạo bảng NHANVIEN bao gồm các cột MANV, HOTEN, NGAYSINH, DIACHI, DIENTHOAI:

```
CREATE TABLE nhanvien
(
    manv        char(10) not null,
    hoten       char(30) not null,
    ngaysinh    datetime null,
    diachi      char(50) null,
    dienthoai   char(6)  null
)
```

b. Sử dụng các ràng buộc trong bảng dữ liệu

❁ Ràng buộc CHECK

Ràng buộc CHECK được sử dụng để chỉ định các giá trị hay khuôn dạng dữ liệu có thể được chấp nhận đối với một cột. Trên một cột có thể có nhiều ràng buộc CHECK. Để khai báo một ràng buộc CHECK đối với một cột nào đó, ta sử dụng cú pháp như sau:

```
[ CONSTRAINT constraint_name]
  CHECK (expression)
```

Trong đó *expression* là một biểu thức logic qui định giá trị hay khuôn dạng của dữ liệu được cho phép. Khi đó, chỉ những giá trị dữ liệu nào làm cho *expression* nhận giá trị đúng mới được chấp nhận.

Ví dụ 1.2: Để qui định điện thoại của nhân viên phải có dạng '######' (chẳng hạn 826767), câu lệnh ở ví dụ 1.1 được viết như sau:

```
CREATE TABLE nhanvien
(
```

```
manv        char(10) not null,
hoten       char(30) not null,
ngaysinh    datetime null,
diachi      char(50) null,
dienthoai   char(6)  null
            constraint check_dienthoai
            check (dienthoai like '[0-9][0-9][0-9]
                                   [0-9][0-9] [0-9]')
)
```

❁ Ràng buộc DEFAULT

Ràng buộc DEFAULT được sử dụng để qui định giá trị mặc định cho một cột. Giá trị này sẽ tự động được gán cho cột này khi người sử dụng bổ sung một bản ghi mà không chỉ định giá trị cho cột. Trên mỗi cột chỉ có thể có nhiều nhất một ràng buộc DEFAULT (tức là chỉ có thể có tối đa một giá trị mặc định).

Để khai báo một giá trị mặc định cho một cột, ta chỉ định một ràng buộc DEFAULT cho cột bằng cách sử dụng cú pháp sau:

```
[ CONSTRAINT constraint_name]
  DEFAULT { const_expression
          | nonarguments_function
          | NULL}
```

Ví dụ 1.3: Câu lệnh dưới đây chỉ định giá trị mặc định là 'không biết' cho cột DIACHI trong bảng NHANVIEN ở ví dụ 1.1 (*)

```
CREATE TABLE nhanvien
(
    manv        char(10) not null,
    hoten       char(30) not null,
    ngaysinh    datetime null,
    diachi      char(50) default 'không biết',
    dienthoai   char(6)  null
)
```

❁ Ràng buộc PRIMARY KEY

(*) Ở ví dụ này, chúng tôi không đặt tên cho ràng buộc DEFAULT. Khi đó, hệ quản trị CSDL sẽ tự động đặt tên cho ràng buộc này. Tuy nhiên, để dễ dàng cho việc quản trị, bạn nên đặt tên cho các ràng buộc.

Ràng buộc PRIMARY KEY được sử dụng để định nghĩa khoá chính của bảng. Một ràng buộc PRIMARY KEY đảm bảo rằng không có các giá trị trùng lặp được đưa vào trên các cột. Hay nói cách khác, giá trị của khoá chính sẽ giúp cho ta xác định được duy nhất một dòng (bản ghi) trong bảng dữ liệu. Mỗi một bảng chỉ có thể có duy nhất một khoá chính và bản thân khoá chính không chấp nhận giá trị NULL. Ràng buộc PRIMARY KEY là cơ sở cho việc đảm bảo tính toàn vẹn thực thể cũng như toàn vẹn tham chiếu.

Để khai báo một ràng buộc PRIMARY KEY, bạn sử dụng cú pháp sau:

```
[ CONSTRAINT constraint_name ]
  PRIMARY KEY [ CLUSTERED|NONCLUSTERED]
              [ ( colname [ ,colname2 [ ...,colname16]] )]
```

Nếu khoá chính của một bảng chỉ là một cột, khi đó bạn không cần thiết phải chỉ định danh sách các cột (sử dụng ràng buộc ở mức cột). Trong trường hợp khoá chính là một tập hợp từ hai cột trở lên, bạn phải chỉ định danh sách các cột (sử dụng ràng buộc ở mức bảng).

Ví dụ 1.4: tạo bảng NHANVIEN với khoá chính là MANV

```
CREATE TABLE nhanvien
(
    manv        char(10) primary key,
    hoten       char(30) not null,
    ngaysinh    datetime null,
    diachi      char(50) null,
    dienthoai   char(6)  null
)
```

câu lệnh trên có thể viết như sau:

```
CREATE TABLE nhanvien
(
    manv        char(10) not null,
    hoten       char(30) not null,
    ngaysinh    datetime null,
    diachi      char(50) null,
    dienthoai   char(6)  null
                constraint pk_nv primary key(manv)
)
```

❁ Ràng buộc UNIQUE

Thay vì sử dụng khoá chính, bạn có thể sử dụng ràng buộc UNIQUE để đảm bảo tính toàn vẹn thực thể. Sử dụng ràng buộc UNIQUE trên một (hay nhiều) cột bắt buộc các giá trị dữ liệu trên một (hay nhiều) cột này không được trùng lặp nhau. Để khai báo một ràng buộc UNIQUE, bạn sử dụng cú pháp lệnh sau đây:

```
[ CONSTRAINT constraint_name]
 UNIQUE [ CLUSTERED | NONCLUSTERED]
        [ colname1 [ ,colname2 [ ...,colname16]] )]
```

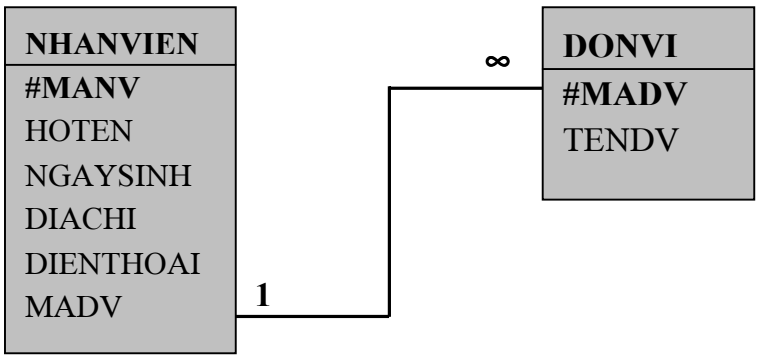
❁ Ràng buộc FOREIGN KEY

Các bảng bên trong một CSDL thường có mối quan hệ với nhau. Các mối quan hệ này được xác định dựa trên tính bằng nhau giữa một hay nhiều trường của bảng này với một hay nhiều trường của bảng khác. Nếu một (hay nhiều) cột nào đó của một bảng có giá trị được xác định từ một (hay nhiều) trường khoá của bảng khác thì các cột đó được gọi là có ràng buộc khoá ngoại (foreign key). Các ràng buộc FOREIGN KEY được sử dụng kết hợp với các ràng buộc PRIMARY KEY và UNIQUE nhằm đảm bảo tính toàn vẹn tham chiếu giữa các bảng được chỉ định.

Để khai báo khoá ngoại, bạn sử dụng cú pháp lệnh như sau:

```
[ CONSTRAINT constraint_name ]
[ FOREIGN KEY (colname [ ,colname2 [ ...,colname16]] )]
 REFERENCES  reference_table [ (ref_colname
                               [ ,ref_colname2
                               [ ...,ref_colname 16]] )]
```

Ví dụ 1.5: Tạo hai bảng NHANVIEN(MANV, HOTEN, NGAYSINH, DIACHI, DIENTHOAI, MADV) và DONVI(MADV, TENDV) theo đồ hình dưới đây:



Hình 1.1

CREATE TABLE donvi

(

```
   madv   char(2)  primary key,
   tendv  char(20) not null
)

CREATE TABLE nhanvien
(
  manv      char(10) primary key,
  hoten     char(20) not null,
  ngaysinh  datetime null,
  diachi    char(50) default 'khong biet',
  dienthoai char(6)  check(dienthoai like '[0-9][0-9][0-9]
                                          [0-9][0-9][0-9]'),
  madv      char(2)  foreign key(madv)
                     references donvi(madv)
)
```

1.3 Sửa đổi bảng

Sau khi đã tạo bảng, bạn có thể tiến hành sửa đổi cấu trúc hay thuộc tính của bảng như bổ sung cột, bổ sung khoá, thay đổi các ràng buộc,... Để có thể sủa đổi bảng, bạn sử dụng câu lệnh ALTER có cú pháp như sau:

```
ALTER TABLE table_name
[ ADD
  { col_name  column_properties  [ column_constraints]
    | [[ ,] table_constraint ] }
     [ ,{ next_col_name|next_table_constraint} ] ...]
[ DROP
  [ CONSTRAINT] constraint_name1
                [ , constraint_name2] ...]
```

Ví dụ 1.6: Tạo một ràng buộc cho bảng DONVI trên cột MADV qui định mã đơn vị phải có dạng hai chữ số (ví dụ 01, 02,...)

```
ALTER TABLE  donvi
ADD  CONSTRAINT check_madv
     CHECK (madv LIKE '[0-9][0-9]')
```

2. Chỉ mục (index)

Các chỉ mục được sử dụng nhằm hỗ trợ việc truy cập đến các dòng dữ liệu được nhanh chóng dựa trên các giá trị của một hay nhiều cột. Chỉ mục được chia ra làm hai loại: chỉ mục tụ nhóm (clustered index) và chỉ mục không tụ nhóm (nonclustered index).

- Một chỉ mục tụ nhóm là một chỉ mục mà trong đó thứ tự logic của các khoá tương tự như thứ tự vật lý của các dòng tương ứng tồn tại trong bảng. Một bảng chỉ có thể có tối đa một chỉ mục tụ nhóm.
- Một chỉ mục không tụ hóm là một chỉ mục mà trong đó thứ tự logic của các khoá không như thứ tự vật lý của các dòng trong bảng. Các chỉ mục tụ nhóm hỗ trợ việc truy cập đến các dòng dữ liệu nhanh hơn nhiều so với các chỉ mục không tụ nhóm.

Khi ta khai báo một khoá chính hay khoá UNIQUE trên một hay nhiều cột nào đó của bảng, hệ quản trị CSDL sẽ tự động tạo chỉ mục trên các cột đó. Bạn có thể tạo thêm các chỉ mục khác bằng cách sử dụng câu lệnh có cú pháp như sau:

```
CREATE [ CLUSTERED|NONCLUSTERED] INDEX index_name
ON  table_name(column_name [ , column_name] ...)
```

Ví dụ 1.7: Câu lệnh dưới đây sẽ tạo một chỉ mục không tụ nhóm trên cột MADV của bảng NHANVIEN

```
CREATE NONCLUSTERED INDEX idx_nhanvien_madv
ON nhanvien(madv)
```

3. Khung nhìn (View)

Một khung nhìn có thể coi như là một "bảng ảo" có nội dung được xác định từ một truy vấn. Một truy vấn (query) là một tập các chỉ dẫn (instruction) nhằm truy xuất và hiển thị dữ liệu từ các bảng trong CSDL. Các truy vấn được thực hiện bằng cách sử dụng câu lệnh SELECT (được đề cập đến trong chương 2).

Một khung nhìn trong giống như một bảng với một tậo các tên cột và các dòng dữ liệu. Tuy nhiên, khung nhìn không tồn tại như là một cấu trúc lưu trữ dữ liệu trong CSDL. Dữ liệu bên trong khung nhìn thực chất là dữ liệu được xác định từ một hay nhiều bảng cơ sổ và do đó phụ thộc vào các bảng cơ sổ.

Các khung nhìn thường được sử dụng bên trong CSDL nhằm các mục đích sau đây:

- Sử dụng khung nhìn để tập trung trên dữ liệu được xác định.

- Sử dụng khung nhìn để đơn giản hoá thao tác dữ liệu.
- Sử dụng khung nhìn để tuỳ biến dữ liệu
- Sử dụng khung nhìn để xuất khẩu (export) dữ liệu.
- Sử dụng khung nhìn để bảo mật dữ liệu.

Để tạo khung nhìn, bạn sử dụng câu lệnh có cú pháp như sau:

```
CREATE VIEW  view_name[(column_name [, column_name]...)]
AS  select_statement
```

Trong đó *select_statement* là một câu lệnh SELECT dùng để truy xuất dữ liệu từ một hay nhiều bảng.

Khi tạo khung nhìn cần lưu ý một số điểm sau:

- Tên khung nhìn phải tuân theo các qui tắc về định danh và phải duy nhất đối với mỗi người sử dụng.
- Không thể ràng buộc các mặc định, các qui tắc cho khung nhìn.
- Không thể xây dựng chỉ mục cho khung nhìn.
- Trong câu lệnh CREATE VIEW không cần thiết phải chỉ định tên cột. Tên của các cột cũng như kiểu dữ liệu của chúng sẽ tương ứng với các cột trong danh sách chọn của câu lệnh SELECT.
- Bạn phải xác định tên cột trong câu lệnh CREATE VIEW trong các trường hợp sau:
  - Mỗi cột của khung nhìn được phát sinh từ một biểu thức số học, một hàm cài sẵn hay một hằng.
  - Hai hay nhiều cột của khung nhìn có trùng tên.
  - Bạn muốn thay đổi tên cột trong khung nhìn khác với tên cột của bảng cơ sở.

Ví dụ 1.8: Câu lệnh dưới đây sẽ bị lỗi do tên của cột thứ 3 không xác định được:

```
CREATE VIEW thongtin_nv
AS
  SELECT manv,hoten,datediff(year,ngaysinh,getdate()),tendv
  FROM nhanvien,donvi
  WHERE nhanvien.madv=donvi.madv
```

Để câu lệnh trên có thể thực hiện được, bạn phải đặt tên cho các cột của khung nhìn như sau:

```
CREATE VIEW thongtin_nv(manv,hoten,tuoi,donvi)
```

```
AS
  SELECT manv,hoten,datediff(year,ngaysinh,getdate()),tendv
  FROM nhanvien,donvi
  WHERE nhanvien.madv=donvi.madv
```

Chương 2: NGÔN NGỮ THAO TÁC DỮ LIỆU

Ngôn ngữ thao tác dữ liệu cung cấp cho người sử dụng khả năng tiến hành các thao tác truy xuất, bổ sung, cập nhật và xoá dữ liệu. Ngôn ngữ thao tác dữ liệu bao gồm các câu lệnh: SELECT, INSERT, UPDATE và DELETE

1. Truy xuất dữ liệu

Để truy xuất dữ liệu từ các dòng và các cột của một hay nhiều bảng, khung nhìn, ta sử dụng câu lệnh SELECT. Câu lệnh này có thể dùng để thực hiện phép chọn (tức là truy xuất một tập con các dòng trong một hay nhiều bảng), phép chiếu (tức là truy xuất một tập con các cột trong một hay nhiều bảng) và phép nối (tức là liên kết các dòng trong hai hay nhiều bảng để truy xuất dữ liệu).

Cú pháp chung của câu lệnh SELECT có dạng như sau:

```
SELECT  [ ALL | DISTINCT ]  select_list
     [ INTO  [ newtable_name ]]
FROM  { table_name | view_name }
                      ..............
      [ ,{ table_name | view_name }]
[ WHERE     clause ]
[ GROUP BY  clause ]
[ HAVING BY clause ]
[ ORDER BY  clause ]
[ COMPUTE   clause ]
```

Chú ý: Các thành phần trong một câu lệnh SELECT phải được sử dụng theo thứ tự được nêu trên.

1.1 Xác định bảng bằng mệnh đề FROM

Mệnh đề FROM trong câu lệnh SELECT được sử dụng nhằm chỉ định các bảng và khung nhìn cần truy xuất dữ liệu. Sau mệnh đề FROM là danh sách tên các bảng và khung nhìn tham gia vào truy vấn (tên của các bảng và khung nhìn được phân cách nhau bởi dấu phẩy).

```
SELECT  select_list
FROM  { table_name | view_name list }
```

Để đơn giản hoá câu hỏi, ta có thể sử dụng các bí danh (alias) cho các bảng hay khung nhìn. Bí danh được gán trong mệnh đề FROM bằng cách chỉ định bí danh sau tên bảng. Ví dụ câu lệnh sau gán bí danh *n1* cho bảng *nhanvien*.

```
SELECT  ten, diachi FROM  nhanvien  n1
```

1.2 Mệnh đề WHERE

Mệnh đề WHERE trong câu lệnh SELECT xác định các điều kiện đối với việc truy xuất dữ liệu. Sau mệnh đề WHERE là một biểu thức logic và chỉ những dòng dữ liệu nào thoả mãn biểu thức sau WHERE mới được hiển thị trong kết quả truy vấn. Trong mệnh đề WHERE thường sử dụng:

- Các toán tử so sánh
- Giới hạn (BETWEEN và NOT BETWEEN).
- Danh sách (IN, NOT IN)
- Khuôn dạng (LIKE và NOT LIKE).
- Các giá trị chưa biết (IS NULL và IS NOT NULL).
- Kết hợp các điều kiện (AND, OR).

Các toán tử so sánh:

Toán tử	ý nghĩa
=	Bằng
>	Lớn hơn
<	Nhỏ hơn
>=	Lớn hơn hoặc bằng
<=	Nhỏ hơn hoặc bằng
<>	Khác
!>	Không lớn hơn
!<	Không nhỏ hơn

Ví dụ 2.1: Truy vấn sau đây cho biết tên, địa chỉ và điện thoại của những nhân viên có hế số lương lớn hơn 1.92:

```
SELECT ten, diachi, dienthoai
FROM nhanvien
WHERE hsluong>1.92
```

Giới hạn (BETWEEN và NOT BETWEEN)

Từ khoá BETWEEN và NOT BETWEEN được sử dụng nhằm chỉ định khoảng giá trị tìm kiếm đối với câu lệnh SELECT. Câu lệnh dưới đây cho biết tên và đia chỉ của những nhân viên có hệ số lương nằm trong khoảng 1.92 đến 3.11

```
SELECT  ten, tuoi, diachi
FROM nhanvien
WHERE  hsluong BETWEEN  1.92 AND 3.11
```

Danh sách (IN và NOT IN)

Từ khoá IN được sử dụng khi ta cần chỉ định điều kiện tìm kiếm dữ liệu cho câu lệnh SELECT là một danh sách các giá trị. Sau IN (hoặc NOT IN) có thể là một danh sách các giá trị hoặc là một câu lệnh SELECT khác.

Ví dụ 2.2: Để hiển thị thông tin về các nhân viên có hệ số lương là 1.86, 1.92 hoặc 2.11, thay vì sử dụng câu lệnh:

```
SELECT * FROM nhanvien
WHERE  hsluong=1.86 OR hsluong=1.92 OR hsluong=2.11
```

Ta có thể sử dụng câu lệnh sau:

```
SELECT * FROM nhanvien
WHERE  hsluong IN (1.86, 1.92, 2.11)
```

Các ký tự đại diện và mệnh đề LIKE

Từ khoá LIKE (NOT LIKE) sử dụng trong câu lệnh SELECT nhằm mô tả khuôn dạng của dữ liệu cần tìm kiếm. Chúng thường được kết hợp với các ký tự đại diện sau đây:

Ký tự đại diện	ý nghĩa
%	Chuỗi ký tự bất kỳ gồm không hoặc nhiều ký tự
-	Ký tự đơn bất kỳ
[]	Ký tự đơn bất kỳ trong giới hạn được chỉ định (ví dụ [a-f]) hay một tập (ví dụ [abcdef])
[^]	Ký tự đơn bất kỳ không nằm trong giới hạn được chỉ định (ví dụ [^a-f] hay một tập (ví dụ [^abcdef]).

Ví dụ 2.3: Câu lệnh dưới đây hiển thị thông tin về các nhân viên có tên là Nam

```
SELECT *  FROM nhanvien
WHERE  hoten LIKE '% Nam'
```

IS NULL và NOT IS NULL

Giá trị NULL có thể được nhập vào một cột cho phép chấp nhận giá trị NULL theo một trong ba cách sau:

- Nếu không có dữ liệu được đưa vào và không có mặc định cho cột hay kiểu dữ liệu trên cột đó.
- Người sử dụng trực tiếp đưa giá trị NULL vào cho cột đó.
- Một cột có kiểu dữ liệu là kiểu số sẽ chứa giá trị NULL nếu giá trị được chỉ định gây tràn số.

Trong mệnh đề WHERE, ta sử dụng IS NULL hoặc IS NOT NULL như sau:

WHERE *col_name* IS [NOT] NULL

Các toán tử logic

Các toán tử logic sử dụng trong mệnh đề WHERE bao gồm AND, OR, NOT. AND và OR được sử dụng để kết hợp nhiều điều kiện trong WHERE.

1.3 Danh sách chọn trong câu lệnh SELECT

*** Chọn tất cả các cột trong bảng**

Khi muốn truy xuất tất cả các cột trong bảng, ta sử dụng câu lệnh SELECT có cú pháp sau:

```
SELECT  *  FROM  table_name
```

Khi sử dụng câu lệnh này, các cột trong kết quả sẽ được hiển thị theo thứ tự mà chúng đã được tạo ra trong câu lệnh CREATE TABLE.

*** Chọn các cột được chỉ định**

Để chọn ra một số cột nào đó trong bảng, ta sử dụng câu lệnh SELECT có cú pháp sau:

```
SELECT  tên_cột [ ,...,tên_cột_n]
FROM  tên_bảng | khung_nhìn
```

Các tên cột trong câu lệnh phải được phân cách nhau bằng dấu phẩy.

Chú ý: Trong câu lệnh SELECT, thứ tự của các cột được nêu ra trong câu lệnh sẽ xác định thứ tự của các cột được hiển thị ra trong kết quả.

*** Đổi tên các cột trong các kết quả**

Khi kết quả được hiển thị, tiêu đề của các cột mặc định sẽ là tên của các cột khi nó được tạo ra trong câu lệnh CREATE TABLE. Tuy nhiên, để các tiêu đề trở nên thân thiện hơn, ta có thể đổi tên các tiêu đề của các cột. Để làm được việc này, ta có thể sử dụng một trong hai cách viết sau:

tiêu_đề_cột = tên_cột

hoặc *tên_cột tiêu_đề_cột*

Ví dụ 2.4: Hai câu lệnh sau sẽ đặt tiêu đề *Họ và tên* cho là *hoten* và *Địa chỉ* cho cột *diachi* khi kết quả được hiển thị cho người sử dụng:

```
SELECT 'Họ và tên'=hoten,
       'Địa chỉ '= diachi
FROM  nhanvien
```

Hoặc:

```
SELECT hoten 'Họ và tên',diachi 'Đia chỉ '
FROM  nhanvien
```

*** Sử dụng cấu trúc CASE để thay đổi dữ liệu trong kết quả**

Trong câu lệnh SELECT, ta có thể sử dụng cấu trúc CASE để thay đổi cách hiển thị kết quả ra màn hình.

Ví dụ 2.5: Câu lệnh sau cho biết họ tên, hệ số lương và xếp loại lương của nhân viên theo hệ số lương:

```
SELECT   'Họ và tên' = ten,  'Hệ số lương' = hsluong,
         'Xếp loại lương' =
          CASE
             WHEN  lsluong=NULL THEN    'Không xác định'
```

```
                WHEN  hsluong<=1.92 THEN     'Luơng thấp'
                WHEN  hsluong<=3.1 THEN   'Lương TB'
                WHEN  hsluong<=5.2 THEN   'Lương cao'
                ELSE  'Lương rất cao'
            END
    FROM   Nhanvien
```

*** Các chuỗi ký tự trong kết quả**

Ta có thể thêm các chuỗi ký tự vào bên trong truy vấn nhằm thay đổi cách thức trình bày dữ liệu.

Ví dụ 2.6: Câu lệnh sau sẽ thêm chuỗi ký tự "Hệ số lương là " vào trước kết quả ở cột mức lương ở từng dòng trong kết quả:

```
SELECT  'Họ và tên' = hoten,
        'Hệ số lương là : ','Hệ số lương'=hsluong
FROM nhanvien
```

Truy vấn trên cho có kết quả có dạng như sau:

```
Họ và tên                          Hệ số lương
---------------    --------------   -------------
Trần Nguyên Phong  Hệ số lương là :    1.92
Lê Hoài Nam        Hệ số lương là :    1.86
Nguyễn Hữu Tình    Hệ số lương là :    1.92
Nguyễn Trung Kiên  Hệ số lương là :    1.86
Nguyễn Thị Hoa     Hệ số lương là :    2.11
Hoàng Nam Phong    Hệ số lương là :    3.21
```

1.4 Tính toán các giá trị trong câu lệnh SELECT

Danh sách chọn trong câu lệnh SELECT có thể có các biểu thức số học. Khi đó kết quả của biểu thức sẽ là một cột trong kết quả truy vấn:

Ví dụ 2.7: Câu lệnh sau cho biết họ tên và lương của các nhân viên

```
SELECT 'Họ và tên'=ten, 'Lương'=hsluong*210000
FROM  nhanvien
```

1.5 Từ khoá DISTINCT

Từ khoá DISTINCT được sử dụng trong câu lệnh SELECT nhằm loại bỏ ra khỏi kết quả truy vấn những dòng dữ liệu có giá trị giống nhau

Ví dụ 2.8: nếu ta sử dụng câu lệnh:

```
SELECT  hsluong FROM nhanvien
```

Ta sẽ có kết quả như sau:

```
hsluong
-----------------
1.92
1.86
1.92
1.86
2.11
3.21
```

Nhưng nếu ta sử dụng câu lệnh:

```
SELECT DISTINCT  hsluong FROM nhanvien
```

kết quả của truy vấn sẽ là:

```
hsluong
----------
1.92
1.86
2.11
3.21
```

1.6 Tạo bảng mới bằng câu lệnh SELECT ... INTO

Câu lệnh SELECT ... INTO có tác dụng tạo một bảng mới có cấu trúc và dữ liệu được xác định từ kết quả của truy vấn. Bảng mới được tạo ra sẽ có số cột bằng số cột được chỉ định trong danh sách chọn và số dòng sẽ à số dòng kết quả của truy vấn

Ví dụ 2.9: Câu lệnh dưới đây tạo mới một bảng có tên là NHANVIEN_LUU bao gồm họ tên và địa chỉ của những nhân viên có hệ số lương lớn hơn 1.92:

```
SELECT hoten, diachi
INTO  nhanvien_luu
FROM  nhanvien
WHERE hsluong>1.92
```

1.7 Sắp xếp kết quả truy vấn bằng ORDER BY

Mệnh đề ORDER BY được sử dụng nhằm sắp xếp kết quả truy vấn theo một hay nhiều cột (tối đa là 16 cột). Việc sắp xếp có thể theo thứ tự tăng tăng (ASC) hoặc giảm (DESC). Nếu không chỉ định rõ thì mặc định là tăng.

Ví dụ 2.10: Câu lệnh sau sẽ sắp xếp các nhân viên theo thứ tự giảm dần của hệ số lương và nếu hệ số lương bằng nhau thì sắp xếp theo thứ tự tăng dần của ngày sinh:

```
SELECT hoten,ngaysinh,hsluong
FROM nhanvien
ORDER BY hsluong DESC, ngaysinh
```

Ta có thể sử dụng các số thay vì sử dụng tên cột sau mệnh đề ORDER BY.

Ví dụ 2.11: câu lệnh dưới đây sắp xếp các nhân viên theo thứ tự tăng dần của ngày sinh:

```
SELECT hoten,ngaysinh,hsluong
FROM nhanvien
ORDER BY 3
```

1.8 Phép hợp và toán tử UNION

Toán tử UNION cho phép ta hợp các kết quả của hai hay nhiều truy vấn thành một tập kết quả duy nhất. Cú pháp của phép hợp như sau:

```
Query_1
[ UNION [ ALL] Query_2 ]
...
[ UNION [ ALL] Query_N ]
[ ORDER BY  clause]
[ COMPUTE  clause]
```

Trong đó:

Query_1 có dạng:

```
SELECT  select_list
[ INTO clause]
[ FROM clause]
[ WHERE clause]
[ GROUP BY clause]
[ HAVING clause]
```

và *Query_i* (i=2,..,N) có dạng:

```
SELECT  select_list
[ FROM clause]
[ WHERE clause]
[ GROUP BY clause]
[ HAVING clause]
```

Ví dụ 2.12: Giả sử ta có hai bảng như sau:

R

A	B	C
abc	3	5
jks	5	7
bdg	10	5
Hht	12	0

S

E	F
edf	15
hht	1
abc	3
adf	2

Thì phép hợp:

```
SELECT A,B FROM R
UNION
```

```
SELECT * FROM S
```

Có kết quả như sau:

```
A     B
----  -----------
abc   3
adf   2
bgd   10
edf   15
hht   1
hht   12
jks   5
```

Theo mặc định, phép toán UNION sẽ loại bỏ những dòng giống nhau trong kết quả. Nếu ta sử dụng tùy chọn ALL thì các dòng giống nhau sẽ không bị loại bỏ. Ta có thể sử dụng các dấu ngoặc để xác định thứ tự tính toán trong phép hợp.

1.8.1 Các nguyên tắc khi xây dựng câu lệnh UNION

Khi xây dựng các câu lệnh UNION, ta cần chú ý các nguyên tắc sau:

- Tất cả các danh sách chọn trong câu lệnh UNION phải có cùng số biểu thức (các tên cột, các biểu thức số học, các hàm gộp,...)
- Các cột tương ứng trong tất cả các bảng, hoặc tập con bất kỳ các cột được sử dụng trong bản thân mỗi truy vấn phải cùng kiểu dữ liệu.
- Các cột tương ứng trong bản thân từng truy vấn của một câu lệnh UNION phải xuất hiện theo thứ tự như nhau. Nguyên nhân là do phép hợp so sánh các cột từng cột một theo thứ tự được cho trong mỗi truy vấn.
- Khi các kiểu dữ liệu khác nhau được kết hợp với nhau trong câu lệnh UNION, chúng sẽ được chuyển sang kiểu dữ liệu cao hơn (nếu có thể được).
- Tiêu đề cột trong kết quả của phép hợp sẽ là tiêu đề cột được chỉ định trong truy vấn đầu tiên.

1.8.2 Sử dụng UNION với các giao tác SQL khác

Các nguyên tắc sau phải được tuân theo khi sử dụng phép hợp với các câu lệnh giao tác SQL khác:

- Truy vấn đầu tiên trong câu lệnh UNION có thể có INTO để tạo một bảng từ kết quả cuối cùng.
- Mệnh đề ORDER BY và COMPUTE dùng để xác định thứ tự kết quả cuối cùng hoặc tính toán các giá trị tóm tắt chỉ được cho phép sử dụng ở cuối của câu lệnh UNION. Chúng không được phép sử dụng trong bất kỳ bản thân truy vấn nào trong phép hợp.
- Mệnh đề GROUP BY và HAVING chỉ có thể được sử dụng trong bản thân từng truy vấn. Chúng không thể được sử dụng để tác động lên kết quả cuối cùng.

- Phép toán UNION cũng có thể được sử dụng bên trong một câu lệnh INSERT.
- Phép toán UNION không thể sử dụng trong câu lệnh CREATE VIEW.

1.9 Phép nối

Phép nối được sử dụng để truy xuất dữ liệu từ hai hay nhiều bảng hoặc khung nhìn trong cùng CSDL hoặc trong các CSDL khác nhau bởi cùng một phép xử lý.

1.9.1 Phép toán nối

Một câu lệnh nối (join statament) thực hiện các công việc sau đây:

- Xác định một cột từ mỗi bảng.
- So sánh các giá trị trong những cột này theo từng dòng.
- Kết hợp các dòng có những giá trị thoả mãn điều kiện thành những dòng mới.

Ví dụ 2.13: Câu lệnh dưới đây cho biết họ tên, địa chỉ của các nhân viên và tên đơn vị mà mỗi nhân viên thực thuộc:

```
SELECT hoten, diachi, tendonvi
FROM nhanvien, donvi
WHERE nhanvien.madonvi = donvi.madonvi
```

Danh sách chọn trong phép nối

Giống như các câu lệnh chọn (selection statment), một câu lệnh nối bắt đầu với từ khóa SELECT. Các cột được chỉ định tên sau từ khoá SELECT là các cột được dưa ra trong kết quả truy vấn. Trong danh sách chọn của phép nối, ta có thể chỉ định một số cột bằng cách chỉ định tên của cột đó hoặc tất cả các cột của những bảng tham gia vào phép nối bằng cách sử dụng dấu sao (*). Danh sách chọn không nhất thiết phải bao gồm các cột của những bảng tham gia phép nối.

Mệnh đề FROM trong phép nối

Mệnh đề FROM của câu lệnh nối xác dịnh các bảng (hay khung nhìn) tham gia vào phép nối. Nếu ta sử dụng dấu * trong danh sách chọn thì thứ tự của các bảng liệt kê trong FROM sẽ ảnh hưởng đến kết quả được hiển thị.

Mệnh đề WHERE trong phép nối

Mệnh đề WHERE xác định điều kiện nối giữa các bảng và các khung nhìn được chỉ định. Nó xác định tên của cột được sử dụng để nối và phép toán nối được sử dụng.

Các toán tử so sánh dưới đây được sử dụng để xác định điều kiện nối

Phép toán	ý nghĩa
=	Bằng
>	Lớn hơn
>=	Lớn hơn hoặc bằng
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
<>	Khác

!>	Không lớn hơn
!<	Không nhỏ hơn.

1.9.2 Các loại phép nối

*** Phép nối bằng và phép nối tự nhiên**

Một *phép nối bằng* (equijoin) là một phép nối trong đó giá trị của các cột được sử dụng để nối được so sánh với nhau dựa trên tiêu chuẩn bằng và tất cả các cột trong các bảng tham gia nối đều được đưa ra trong kết quả.

Ví dụ 2.14:

```
SELECT * FROM nhanvien, donvi
WHERE nhanvien.madonvi = donvi.madonvi
```

Trong kết quả của câu lệnh trên, cột *madonvi* và *tendonvi* xuất hiện hai lần trong kết quả phép nối và như vậy là không cần thiết. Để loại bỏ điều này, ta có thể sử dụng *phép nối tự nhiên* (natural join) bằng cách loại bỏ đi các cột trùng tên với nhau.

*** Phép nối với các điều kiện bổ sung**

Trong mệnh đề WHERE của câu lệnh nối, ta có thể bố sung các điều kiện tìm kiếm khác.

Ví dụ 2.15:

```
SELECT hoten, diachi, tendonvi
FROM nhanvien, donvi
WHERE nhanvien.madonvi = donvi.madonvi AND
      Nhanvien.hsluong>=2.11
```

*** Phép tự nối và các bí danh**

Phép tự nối là phép nối mà trong đó ta so sánh các giá trị bên trong một cột của cùng một bảng.

Ví dụ 2.16: Tìm những nhân viên có cùng địa chỉ với nhân viên '*Trần Nguyên Phong*'

```
SELECT n1.hoten
FROM nhanvien n1, nhanvien n2
WHERE n2.hoten='Trần Nguyên Phong' AND
      n1.diachi = n2.diachi
```

*** Phép nối không dựa trên tiêu chuẩn bằng**

Trong phép nối này, các cột được sử dụng để kết nối được so sánh với nhau không dựa trên điều kiện bằng.

*** Phép nối ngoài (outer join)**

Trong các phép nối đã đề cập ở trên, chỉ những dòng hợp lệ (tức là những dòng có giá trị trong các cột được chỉ định thoả mã điều kiện kết nối) mới được đưa ra trong kết quả. Theo một nghĩa nào đó, những phép nối này loại bỏ thông tin chứa trong những dòng không hợp lệ. Tuy nhiên, đôi khi ta cũng cần giữ lại những thông tin không hợp lệ bằng cách cho phép những dòng không hợp lệ có mặt trong kết quả của

phép nối. Để làm điều này, ta có thể sử dụng *phép nối ngoài*. Giao tác SQL cung cấp hai phép nối ngoài:

- Phép nối ngoài trái (*=) : Phép nối này cho phép lấy tất cả các từ bảng có tên đầu tiên.
- Phép nối ngoài phải (=*) : Phép nối này cho phép lấy tất cả các dòng từ bảng có tên thứ hai.

Ví dụ 2.17: Giả sử ta có hai bảng R và S có nội dung như sau

A	B	C	D
aaa	2	4	aaaa
fff	8	4	f2
ggg	2	7	g4
xxx	2	12	gdf
ggg	2	12	khf
sss	45	0	k3h

Bảng R

E	F	G
aaa	3	(null)
xxx	23	26
abc	3	6
(null)	12	(null)
sss	20	3

Bảng S

Khi đó câu lệnh:

```
SELECT * FROM R,S
WHERE R.A = S.E
```

Cho kết quả là:

A	B	C	D	E	F	G
aaa	2	4	aaaa	aaa	3	(null)
xxx	2	12	gdf	xxx	23	26
sss	45	0	k3h	sss	20	3

Còn câu lệnh:

```
SELECT * FROM R,S
WHERE R.A *= S.E
```

Cho kết quả là:

```
A    B     C     D     E     F       G
---  ----  ----  -----  -----  ------  ------
aaa  2     4     aaaa   aaa   3       (null)
fff  8     4     f2     (null)  (null)  (null)
ggg  2     7     g4     (null)  (null)  (null)
xxx  2     12    gdf    xxx    23      26
ggg  2     12    khf    (null)  (null)   (null)
sss  45    0     k3h    sss    20       3
```

Và câu lệnh

```
SELECT * FROM R,S
WHERE R.A =* S.E
```

Cho kết quả là:

```
A       B       C      D       E       F      G
----    -----   -----  -----  -----   ------ ---------
aaa     2       4      aaaa   aaa    3       (null)
xxx     2       12     gdf    xxx    23      26
(null)(null)  (null)  (null)  abc    3       6
(null)(null)  (null)  (null)  (null)12      (null)
 sss   45       0       k3h    sss    20      3
```

Các giới hạn của phép nối ngoài

Giao tác SQL không cho phép hai phép nối ngoài lồng nhau và phép nối trong lồng vào trong phép nối ngoài. Tuy nhiên, một bảng có thể tham gia trong một phép nối trong và là bảng ngoài (outer table) trong một phép nối ngoài.

Bảng dưới đây cho biết các cách kết hợp hợp lệ và không hợp lệ của phép nối trong và phép nối ngoài:

Hợp lệ	Không hợp lệ
T1 =* T2 *= T3 T1 = T2 *= T3 T1 *= T2 =* T3 T3	T1 *= T2 *= T3 T1 *= T2 = T3 T3

||	||
*	*
T2 =* T1 *= T4	T2 =* T1 =* T4
*	*
||	||
T5	T5
T3	T3
*	||
||	*
T2 *= T1 =* T4	T2 *= T1 *= T4
||	||
*	*
T5	T5

*** Phép nối và các giá trị NULL**

Nếu trong các cột của các bảng tham gia phép nối có các giá trị NULL thì các giá trị NULL được xem như là không bằng nhau. Chẳng hạn ta có hai bảng:

A	B
1	b1
Null	b2
4	b3

Bảng R

C	D
Null	d1
4	d2

Bảng S

Thì câu lệnh:

```
SELECT * FROM  R, S
WHERE A *= C
```

Sẽ cho kết quả là:

A	B	C	D
1	b1	Null	Null
Null	b2	Null	Null
4	b3	4	D2

1.10 Tạo các dòng thống kê dữ liệu với COMPUTE ... BY

Ta sử dụng mệnh đề COMPUTE BY kết hợp với các hàm gộp dòng và mệnh đề ORDER BY để sản sinh ra các các báo cáo (report) nhằm thống kê các giá trị dữ liệu

theo từng nhóm. Những giá trị thống kê này xuất hiện như là những dòng bổ sung trong kết quả truy vấn. Một mệnh đề COMPUTE BY cho phép ta quan sát cả các chi tiết về dữ liệu lẫn các giá trị thống kê. Ta có thể tính các giá trị thống kê cho các nhóm con (subgroups) và ta cũng có thể tính toán nhiều hàm gộp trên cùng một nhóm.

Mệnh đề COMPUTE BY có cú pháp như sau:

```
COMPUTE  row-aggregate(col_name)
         [ ,...,row_aggregate(col_name)]
 BY col_name [ ,...,col_name]
```

Các hàm gộp có thể sử dụng được với COMPUTE BY bao gồm SUM, AVG, MIN, MAX và COUNT.

Ví dụ 2.18: Câu lệnh dưới đây cho biết họ tên, tên đơn vị hệ số lương của nhân viên đồng thời cho biết lương trung bình của các nhân viên trong mỗi đơn vị

```
SELECT hoten,tendonvi,hsluong
FROM nhanvien,donvi
WHERE nhanvien.madonvi=donvi.madonvi
ORDER BY nhanvien.madonvi
COMPUTE AVG(hsluong) BY nhanvien.madonvi
```

Kết quả của truy vấn này sẽ như sau:

```
Hoten              tendonvi          hsluong
---------------    --------------    --------------
Nguyễn Thị Hoa     Phòng kế toán     2.11
                                     avg
                                     ==========
                                     2.11

Lê Hoài Nam        Phòng Tổ chức     1.86
Hoàng Nam Phong    Phòng Tổ chức     3.21
                                     avg
                                     ==========
                                     2.535

Trần Nguyên Phong  Phòng điều hành   1.92
Nguyễn Hữu Tình    Phòng điều hành   1.92
                                     avg
                                     ==========
                                     1.92

Nguyễn Trung Kiên  Phòng tài vụ      1.86
```

```
avg
==========
1.86
```

Khi sử dụng mệnh đề COMPUTE ... BY cần tuân theo các qui tắc dưới đây:

- Từ khóa DISTINCT không cho phép sử dụng với các hàm gộp dòng
- Các cột sử dụng trong mệnh đề COMPUTE phải xuất hiện trong danh sách chọn.
- Không sử dụng SELECT INTO trong một câu lệnh SELECT có sử dụng COMPUTE.
- Nếu sử dụng mệnh đề COMPUTE ... BY thì cũng phải sử dụng mệnh đề ORDER BY. Các cột liệt kê trong COMPUTE BY phải giống hệt hay là một tập con của những gì được liệt kê sau ORDER BY. Chúng phải có cùng thứ tự từ trái qua phải, bắt đầu với cùng một biểu thức và không bỏ qua bất kỳ một biểu thức nào.

Chẳng hạn nếu mệnh đề ORDER BY có dạng:

ORDER BY a, b, c

Thì mệnh đề COMPUTE BY có dạng dưới đây là hợp lệ:

COMPUTE row_aggregate (column_name) BY a, b, c

COMPUTE row_aggregate (column_name) BY a, b

COMPUTE row_aggregate (column_name) BY a

Và các dạng dưới đây là sai

COMPUTE row_aggregate (column_name) BY b, c

COMPUTE row_aggregate (column_name) BY a, c

COMPUTE row_aggregate (column_name) BY c

- Phải sử dụng một tên cột hoặc một biểu thức trong mệnh đề ORDER BY, việc sắp xếp (order) không được thực hiện dựa trên tiêu đề cột.
- Từ khoá COMPUTE có thể được sử dụng mà không có BY và khi đó ORDER BY là tùy chọn.

1.11 Thống kê dữ liệu với GROUP BY và HAVING

Ta có thể sử dụng các mệnh đề GROUP BY và HAVING để thống kê dữ liệu. GROUP BY tổ chức dữ liệu vào các nhóm, HAVING thiết lập các điều kiện lên các nhóm trong kết quả truy vấn. Những mệnh đề này thường được sử dụng kết hợp với nhau (HAVING được sử dụng không kèm với GROUP BY có thể tạo ra những két quả nhầm lẫn).

Các hàm gộp trả về các giá trị tóm lược cho cả bảng hoặc cho các nhóm trong bảng. Do đó, chúng thường được sử dụng với GROUP BY. Các hàm gộp có thể xuất hiện trong một dang sách chọn hay trong mệnh đề HAVING, nhưng không được sử dụng trong mệnh đề WHERE.

Ta có thể sử dụng các hàm gộp dưới đây với GROUP BY (Trong đó *expression* là một tên cột).

Hàm gộp	Chức năng
SUM([ALL | DISTINCT] *expression*)	Tính tổng các giá trị.
AVG([ALL | DISTINCT] *expression*)	Tính trung bình của các giá trị
COUNT([ALL | DISTINCT] *expression*)	Số các giá trị trong biểu thức.
COUNT(*)	Số các dòng được chọn.
MAX(*expression*)	Tính giá trị lớn nhất
MIN(*expression*)	Tính giá trị nhỏ nhất

Trong đó, SUM và AVG chỉ làm việc với những giá trị kiểu số. SUM, AVG, COUNT, MAX và MIN bỏ qua các giá trị null còn COUNT(*) thì không.

Ví dụ 2.19: Câu lệnh dưới đây cho biết hệ số lương trung bình của các nhân viên theo từng đơn vị:

```
SELECT donvi.madonvi,tendonvi, avg(hsluong)
FROM nhanvien,donvi
WHERE nhanvien.madonvi = donvi.madonvi
GROUP BY donvi.madonvi,tendonvi
```

Chú ý: Danh sách các tên cột trong danh sách chọn của câu lệnh SELECT và danh sách các tên cột sau GROUP BY phải như nhau, nếu không câu lệnh sẽ không hợp lệ. Ví dụ câu lệnh dưới đây là không đúng:

```
SELECT donvi.madonvi,tendonvi, avg(hsluong)
FROM nhanvien,donvi
WHERE nhanvien.madonvi = donvi.madonvi
GROUP BY donvi.madonvi
```

Mệnh đề HAVING thiết lập các điều kiện đối với mệnh đề GROUP BY tương tự như cách thức mệnh đề WHERE thiết lập các điều kiện cho câu lệnh SELECT. Mệnh đề HAVING sẽ không có nghĩa nếu như không sử dụng kết hợp với mệnh đề WHERE. Có một điểm khác biệt giữa HAVING và WHERE là trong điều kiện tìm kiếm của WHERE không được có các hàm gộp trong khi HAVING lại cho phép sử dụng các hàm gộp trong điều kiện tìm kiếm của mình. Mệnh đề HAVING có thể tham chiếu đến bất kỳ mục nào trong danh sách chọn và mệnh đề HAVING có thể chứa tối đa 128 điều kiện tìm kiếm.

Ví dụ 2.20: Câu lệnh dưới đây cho biết hệ số lương trung bình của các nhân viên theo từng đơn vị và chỉ hiển thị những đơn vị có hệ số lương trung bình lớn hơn 1.92

```
SELECT donvi.madonvi,tendonvi, avg(hsluong)
FROM nhanvien,donvi
WHERE nhanvien.madonvi = donvi.madonvi
GROUP BY donvi.madonvi,tendonvi
HAVING avg(hsluong)>1.92
```

1.12 Truy vấn con (subquery)

Một truy vấn con là một câu lệnh SELECT được lồng vào bên trong một câu lệnh SELECT, INSERT, UPDATE hay DELETE hoặc bên trong một truy vấn con khác. Câu lệnh truy vấn con có thể tham chiếu đến cùng một bảng với truy vấn ngoài hoặc một bảng khác. Trong giáo tác SQL, một truy vấn con trả về một chỉ giá trị có thể được sử dụng tại những vị trí mà tại đó một biểu thức được cho phép sử dụng.

Cú pháp truy vấn con

Một truy vấn con được lồng vào bên trong một câu lệnh SELECT có cú pháp như sau:

```
(SELECT [ ALL|DISTINCT] subquery_select_list
[ FROM  { table_name|view_name} [ optimizer_hints]
        [[ ,table_name2|view_bame2} [ optimizer_hints]
        [ ...,table_name16|view_name16}[ optimizer_hints]]]
[ WHERE clause]
[ GROUP BY clause]
[ HAVING clause] )
```

Câu lệnh SELECT của truy vấn con luôn nằm trong cặp dấu ngoặc. Nó không được chứa mệnh đề ORDER BY, COMPUTE hoặc FOR BROWSE. Một truy vấn con có thể được lồng vào bên trong mệnh đề WHERE hay HAVING của một câu lệnh SELECT, INSERT hay DELETE, hoặc bên trong truy vấn con khác. Nếu một truy vấn con trả về chỉ một giá trị, nó có thể được sử dụng tại những vị trí mà ở đó một biểu thức được cho phép sử dụng. Một truy vấn con không được phép sử dụng bên trong một danh sách của mệnh đề ORDER BY.

Các câu lệnh chứa truy vấn con thường có một trong số các dạng sau:

```
(1)  WHERE expression [ NOT] IN (subquery)
(2)  WHERE expression comparison_operator [ ANY|ALL] (subquery)
(3)  WHERE [ NOT] EXISTS (subquery)
```

Ví dụ 2.21: Câu lệnh sau đây hiển thi thông tin về các nhân viên làm việc ở những đơn vị có số điện thoại không bắt đầu bởi số 82

```
SELECT *
FROM nhanvien
WHERE madonvi NOT IN ( SELECT madonvi
                       FROM donvi
                       WHERE dienthoai like '82%')
```

2. Bổ sung, cập nhật và xoá dữ liệu

2.1 Bổ sung dữ liệu

Để bổ sung dữ liệu vào trong một bản dữ liệu, ta sử dụng câu lệnh INSERT. Dạng đơn giản nhất của câu lệnh này có cú pháp như sau:

```
INSERT [ INTO]    table_name
VALUES(value1, value2, ...)
```

Trong dó *table_name* là tên của bảng cần thao tác. Số lượng các giá trị được chỉ định phải giống số lượng các cột khi định nghĩa bảng và kiểu dữ liệu của các giá trị này phải phù hợp với kiểu dữ liệu của các cột tương ứng.

Ví dụ 2.22: Câu lệnh dưới đây bổ sung thêm một nhân viên vào bảng *nhanvien*.

```
INSERT INTO nhanvien
VALUES('NV02003','Lê Thị Mai','23/5/72',
       NULL,'523312',1.92,'02')
```

Trong trường hợp chỉ nhập dữ liệu cho một số cột trong bảng, ta phải chỉ định danh sách các cột cần nhập dữ liệu ngay sau tên bảng. Khi đó câu lệnh INSERT có cú pháp như sau:

```
INSERT [ INTO]    table_name(col1,col2,..., colN)
VALUES(value1, value2, ...,valueN)
```

Trong trường hợp này, các cột không được nhập dữ liệu sẽ nhận giá trị mặc định (nếu có) hoặc nhận giá trị NULL. Nếu ta không nhập dữ liệu cho một cột không có ràng buộc DEFAULT và cũng không cho phép chấp nhận giá trị NULL, câu lệnh sẽ bị lỗi.

Ví dụ 2.23:

```
INSERT INTO nhanvien(manv,hoten,diachi)
VALUES('NV03002','Nguyễn Thị Hạnh Dung','56 Trần Phú')
```

Ngoài hai dạng ở trên, câu lệnh INSERT còn cho phép ta nhập dữ liệu cho một bảng bằng cách lấy dữ liệu từ một bảng khác. Hay nói cách khác, câu lệnh INSERT còn cho phép chúng ta sao lưu dữ liệu từ bảng này sang bảng khác.

Ví dụ 2.24: Giả sử ta có bảng *luong_nhanvien* bao gồm hai cột *hoten* và *luong*, câu lệnh dưới đây bổ sung dữ liệu và bảng *luong_nhanvien* bằng cách lấy dữ liệu từ bảng nhân viên:

```
INSERT INTO luong_nhanvien
SELECT  hoten,hsluong*210000 FROM nhanvien
```

2.2 Cập nhật dữ liệu

Câu lệnh UPDATE cho phép người sử dụng thay đổi dữ liệu đã tồn tại bên trong bảng dữ liệu. Câu lệnh này có cú pháp như sau:

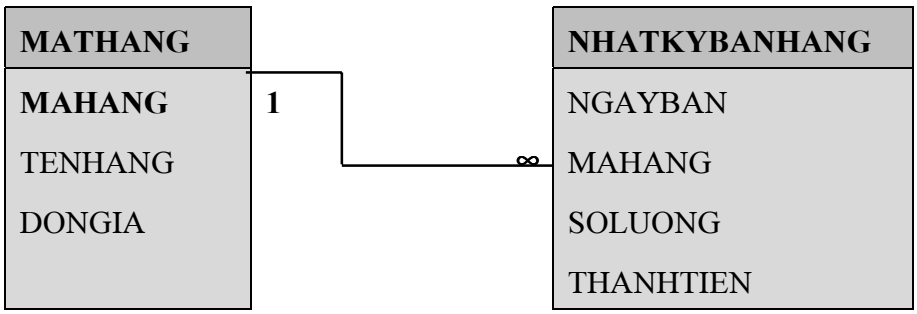
```
UPDATE  updated_table_name
SET  colname = expression
        [ , ..., colname = expression ]
[ FROM  table_name [ ,..., table_name]]
[ WHERE  conditions ]
```

Ví dụ 2.25: Câu lệnh dưới đây tăng lương lên 0.2 cho những nhân viên làm việc tại đơn vị có mã đơn vị là 04

```
UPDATE nhanvien
SET hsluong = hsluong+0.2
WHERE madonvi ='04'
```

Mệnh đề FROM trong câu lệnh UPDATE được sử dụng khi cần chỉ định các điều kiện cập nhật lên quan đến các bảng khác.

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



Câu lệnh dưới đây sẽ cập nhật giá trị cho trường THANHTIEN trong bảng NHATKYBANHANG theo công thức THANHTIEN = SOLUONG × DONGIA

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

2.3 Xoá dữ liệu

Để xoá các bản ghi dữ liệu ra khỏi bảng dữ liệu, ta sử dụng câu lệnh DELETE có cú pháp như sau:

```
DELETE  [ FROM]  delete_table_name
[ FROM table_name [ ,..., table_name]]
[ WHERE conditions]
```

Ví dụ 2.26: Câu lệnh dưới đây xoá khỏi bảng *nhanvien* những nhân viên làm việc tại đơn vị có số điện thoại là '848484'

```
DELETE FROM nhanvien
```

```
FROM donvi
WHERE nhanvien.madonvi = donvi.madonvi AND
      donvi.dienthoai ='848484'
```

Chương 3: NGÔN NGỮ ĐIỀU KHIỂN

Ngôn ngữ điều khiển được sử dụng trong việc cấp phát hay huỷ bỏ quyền của người sử dụng đối với các câu lệnh SQL hoặc trên các đối tượng CSDL.

1. Câu lệnh GRANT

Câu lệnh GRANT được sử dụng nhằm cấp phát quyền cho người sủ dụng trên các đối tượng CSDL hoặc quyền thực thi các câu lệnh SQL. Cú pháp của câu lệnh này có hai dạng như sau:

Dạng 1: Cấp phát quyền đối với các câu lệnh

```
GRANT ALL | statement [ ,...,statementN ]
TO account [ ,...,accountN]
```

Dạng 2: Cấp phát quyền đối với các đối tượng CSDL

```
GRANT ALL | permission [ ,...,permissionN]
    ON table_name |view_name [ (column1 [ ,...,columnN] )]
   |ON stored_procedure
TO account [ ,...,accountN]
```

Trong đó:

- ALL: là từ khoá được sử dụng khi muốn cấp phát tất cả các quyền có thể cho người sử dụng
- *Statement*: là câu lệnh được cấp phát quyền cho người sử dụng. Các câu lệnh có thể cấp phát cho người sử dụng bao gồm:
    - CREATE DATABASE
    - CREATE DEFAULT
    - CREATE PROCEDURE
    - CREATE RULE
    - CREATE TABLE
    - CREATE VIEW
    - BACKUP DATABASE
    - BACKUP LOG

- *account*: là tên tài khoản của người sử dụng khi đăng nhập vào hệ thống.
- *Permission*: là một quyền cấp phát cho người sử dụng trên đối tượng CSDL và được qui dịnh như sau:
    - Các quyền có thể cấp phát trên một bảng hoặc khung nhìn: SELECT, INSERT, DELETE và UPDATE.
    - Các quyền có thể cấp phát trên các cột của bảng hoặc khung nhìn: SELECT và UPDATE.
    - Quyền có thể cấp phát đối với thủ tục lưu trữ: EXCUTE

Ví dụ 3.1:

Cấp phát quyền thực thi câu lệnh CREATE TABLE và CREATE VIEW cho tài khoản có tên là *db_user*:

```
GRANT CREATE TABLE, CREATE VIEW
TO db_user
```

Cấp phát cho các tài khoản có tên là *db_user1* và *db_user2* quyền được xem và cập nhật dữ liệu trên các cột *hoten*, *diachi, dienthoai* và *hsluong* của bảng *nhanvien*

```
GRANT  SELECT, UPDATE
ON nhanvien(hoten,diachi,dienthoai,hsluong)
TO db_user1, db_user2
```

2. Câu lệnh REVOKE

Câu lệnh REVOKE được sử dụng để huỷ bỏ quyền đã được cấp phát cho người sử dụng trên các đối tượng CSDL hoặc câu lệnh SQL. Câu lệnh REVOKE cũng có hai dạng như sau:

Dạng 1: Huỷ bỏ quyền đối với câu lệnh

```
REVOKE ALL | statement [ ,...,statementN]
FROM account [ ,...,accountN]
```

Dạng 2: Huỷ bỏ quyền đối với đối tượng CSDL

```
REVOKE ALL | permission [ ,...,permissionN]}
   ON table_name | view_name [ (column [ ,...,columnN] )]
  | stored_procedure
FROM  account [ ,...,accountN ]
```

Ví dụ 3.2: Huỷ bỏ quyền xem và cập nhật dữ liệu trên cột *hsluong* của bảng *nhanvien* đối với tài khoản có tên là *db_user1*

```
REVOKE SELECT, UPDATE
ON nhanvien(hsluong)
FROM db_user1
```

Huỷ bỏ tất cả các quyền đã cấp phát cho tài khoản có tên là db_user

```
REVOKE ALL
FROM db_user
```

Chương 4: THỦ TỤC LƯU TRỮ VÀ TRIGGER

I. Sử dụng thủ tục lưu trữ (stored procedure)

Các thủ tục lưu trữ là một trong những đối tượng cơ sở dữ liệu. Có thể xem chúng tương tự như những thủ tục trong các ngôn ngữ lập trình. Mỗi một thủ tục lưu trữ có thể có các khả năng sau:

- Nhận các tham số đầu vào, thực thi các câu lệnh bên trong thủ tục và trả về các giá trị.
- Bên trong mỗi thủ tục có thể chứa các câu lệnh nhằm thực hiện các thao tác trên cơ sở dữ liệu (kể cả việc gọi đến các thủ tục lưu trữ khác)
- Trả về một giá trị trạng thái thông qua đó có thể xác định việc thực thi thủ tục là thành công hay bị lỗi.

Việc sử dụng các thủ tục lưu trữ bên trong cơ sở dữ liệu sẽ mang lại những lợi ích sau:

- Thủ tục lưu trữ cho phép module hoá công việc, tạo điều kiện thuận lợi cho việc thực hiện các thao tác trên dữ liệu.
- Thủ tục lưu trữ được phân tích, tối ưu và biên dịch khi tạo ra nên việc thực thi chúng nhanh hơn nhiều so với việc sử dụng một tập các câu lệnh giao tác SQL theo những cách thông thường.
- Thủ tục lưu trữ cho phép chúng ta thực hiện cùng một yêu cầu bằng một câu lệnh đơn giản thay vì phải sử dụng nhiều dòng lệnh SQL. Điều này sẽ làm giảm thiểu sự lưu thông trên mạng.
- Thay vì cấp phát quyền trực tiếp cho người sử dụng trên các câu lệnh SQL, ta có thể cấp phát quyền cho người sử dụng thông qua các thủ tục lưu trữ, nhờ đó tăng khả năng bảo mật đối với hệ thống.

I.1. Tạo các thủ tục lưu trữ

Để tạo một sp, ta sử dụng câu lệnh CREATE PROCEDURE có cú pháp như sau:

```
CREATE PROCEDURE  procedure_name [ ;number]
                [ (parameter1 [ ,parameter2] ...[ parameter255] )]
 AS  sql_statements
```

Ví dụ 4.1:

```
CREATE PROC  sp_list   @bten  char(20)
AS
    SELECT hoten, ngaysinh, diachi
              FROM  nhanvien
    WHERE hoten= @bten
```

Chú ý: Nếu khi gọi thủ tục, chúng ta truyền tham số cho thủ tục dưới dạng:

@tham_số = giá_trị

Thì thứ tự các tham số không cần phải tuân theo thứ tự như khi tạo thủ tục bằng câu lệnh CREATE PROCEDURE. Tuy nhiên, nếu như đã có một tham số được truyền giá trị theo cách trên thì tất cả các tham số còn lại cũng phải được truyền giá trị theo cách đó.

Ta có thể gán một giá trị mặc định cho tham số trong câu lệnh CREATE PROCEDURE. Giá trị này, có thể là hằng bất kỳ, sẽ được lấy làm tham số của thủ tục khi người sử dụng không cung cấp giá trị cho tham số khi gọi thủ tục.

Ví dụ 4.2:

```
CREATE PROC sp_list;2  @bten char(20)='Nguyen Van A'
AS
     SELECT * FROM nhanvien
     WHERE  hoten = @bten
```

Với thủ tục trên, nếu ta gọi *msp_list;2* mà không có tham số thì thủ tục sẽ lấy tham số mặc định là 'Nguyễn Văn A' cho @bten.

Giá trị mặc định có thể NULL. Trong trường hợp này, nếu người sử dụng không cung cấp tham số, SQL Server sẽ thi hành thủ tục theo các tham số khác.

Ví dụ 4.3: Với câu lệnh

```
CREATE PROC sp_list;3 @bten char(20)=NULL,@bluong float
AS
    SELECT * FROM nhanvien
    WHERE hoten=@bten AND hsluong=@bluong
```

Ta thể gọi thủ tục trên như sau:

```
msp_list;3 @btuoi=23
```

mà không bị lỗi.

Mặc định có thể bao gồm các ký tự đại diện (%, _, [], [^]) nếu thủ tục sử dụng tham số với từ khóa LIKE.

Ví dụ 4.4:

```
CREATE PROC  sp_list;4  @bten char(20) ='TrÌn%'
AS
   SELECT * FROM nhanvien
   WHERE hoten LIKE @bten
```

I.2. Thông tin trả về từ các thủ tục lưu trữ

Các giá trị trạng thái trả về:

Các thủ tục có thể trả về một giá trị nguyên được gọi là một trạng thái trả về. Giá trị này chỉ ra cho biết thủ tục được thực hiện thành công hay gặp lỗi và nguyên nhân của lỗi (SQL Server đã định nghĩa sẵn một tập các giá trị trả về, các giá trị này nằm trong khoảng [-99;0]; trong đó giá trị trả về bằng 0 tức là việc thực hiện thủ tục thành công, các giá trị còn lại cho biết nguyên do khi bị lỗi).

Giá trị trả về do người sử dụng định nghĩa

Người sử dụng có thể định nghĩa các giá trị trả về của mình trong các thủ tục lưu trữ bằng cách bổ sung một tham số vào câu lệnh RETURN. Tất cả các số nguyên ngoại trừ các giá trị dành riêng cho hệ thống đều có thể được sử dụng.

Ví dụ 4.5:

```
CREATE PROC sp_exam  @bten  char(20)
AS
  IF  EXISTS (SELECT * FROM nhanvien WHERE hoten = @bten)
      RETURN  1
  ELSE
      RETURN 2
```

Các tham số trả về

Khi cả hai câu lệnh CREATE PROCEDURE và EXECUTE chứa mục chọn OUTPUT cho tên một tham số, thủ tục có thể sử dụng một biến để trả về trị của tham số đó đến người gọi. Bằng việc sử dụng từ khoá OUTPUT, bất cứ sự thay đổi nào đến cũng vẫn còn giữ lại sau khi thủ tục được thực hiện, và các biến có thể được sử dụng trong các câu lệnh SQL bổ sung sau đó trong tập lệnh hay thủ tục được gọi. Nếu từ khoá OUTPUT không được sử dụng, việc thay đổi đến tham số sẽ không được giữ lại sau khi kết thúc thực hiện thủ tục. Ngoài ra, ta còn có thể dùng RETURN để trả về giá trị.

Một thủ tục lưu trữ có thể sử dụng bất kỳ hoặc tất cả khả năng sau để trả về:

- Một hoặc nhiều tập các giá trị.
- Một giá trị trả về rõ ràng (sử dụng câu lệnh RETURN).
- Một tham số OUTPUT.

Nếu chúng ta chỉ định OUTPUT khi thực hiện một thủ tục nhưng tham số tương ứng không được định nghĩa với OUTPUT khi tạo thủ tục thì sẽ bị lỗi. Tuy nhiên nếu ta định nghĩa OUTPUT cho một tham số trong thủ tục nhưng không chỉ định OUTPUT khi thực hiện thì vẫn không bị lỗi (giá trị tham số khi đó sẽ không được trả về).

Ví dụ 4.6:

```
CREATE PROC Chia @sobichia real,@sochia real,
            @kqua  real  OUTPUT
AS
    IF  (@sochia =0)
         Print 'Division by zero'
    ELSE
         SELECT  @kqua = @sobichia / @sochia
```

Khi đó nếu ta thực hiện như sau:

```
DECLARE  @ketqua   real
EXEC  Chia  100, 2, @ketqua  OUT
SELECT  @ketqua
```

Sẽ cho kết quả là:

```
----------------------
50.0
```

Còn nếu thực hiện

```
DECLARE  @ketqua   real
EXEC  Chia  100, 2, @ketqua
SELECT  @ketqua
```

Sẽ cho kết quả là:

```
----------------------
(null)
```

I.3. Các qui tắc sử dụng cho sp

Sau đây là một số qui tắc cần lưu ý khi tạo các thủ tục lưu trữ

- Câu lệnh CREATE PROCEDURE không thể kết hợp với các câu lệnh SQL khác trong một khối lệnh đơn (single batch).
- Bản thân định nghĩa CREATE PROCEDURE có thể bao gồm bất kỳ số lượng cũng như câu lệnh SQL nào ngoại trừ những câu lệnh sau:

  CREATE VIEW CREATE TRIGGER

  CREATE DEFAULT CREATE PROCEDURE

  CREATE RULE
- Các đối tượng CSDL khác có thể được tạo bên trong một thủ tục lưu trữ. Ta có thể tham chiếu một đối tượng được tạo trong cùng thủ tục miễn là nó đã được tạo trước khi tham chiếu.
- Bên trong một thủ tục, ta không thể tạo một đối tượng, xoá nó và sau đó tạo một đối tượng mới với cùng tên.
- Ta có thể tham chiếu các bảng tạm thời bên trong một thủ tục.
- Nếu ta thực thi một thủ tục mà gọi đến thủ tục khác, thủ tục được gọi có thể truy cập đến mọi đối tượng ngoại trừ các bảng tạm thời được tạo bởi thủ tục đầu tiên.
- Nếu ta tạo một bảng tạm thời riêng (private temporary table) bên trong một thủ tục, bảng tạm thời chỉ tồn tại cho những mục đích của thủ tục đó; nó sẽ mất đi khi thoát ra khỏi thủ tục.
- Số tham số tối đa của một thủ tục là 255.
- Số biến cục bộ và toàn cục trong một thủ tục chỉ bị giới hạn bởi khả năng bộ nhớ.
- Các thủ tục tạm thời cục bộ (private) và toàn cục (public), tương tự như các bảng tạm thời, có thể được tạo với dấu # và ## đứng trước tên thủ tục. # biểu diễn thủ tục tạm thời cục bộ còn ## biểu diễn thủ tục tạm thời toàn cục.

I.4 Xác định tên bên trong các thủ tục

Bên trong một thủ tục, tên các đối tượng được sử dụng với câu lệnh ALTER TABLE, CREATE TABLE, DROP TABLE, TRUNCATE TABLE, CREATE INDEX, DROP INDEX, UPDATE STATISTICS và DBCC phải được xác định với tên của người sở hữu đối tượng (object owner's name) nếu như những người dùng (user) khác sử dụng thủ tục. Ví dụ, người dùng Mary, là sở hữu chủ của bảng *marytab*, phải chỉ định tên của bảng của mình khi nó được sử dụng với một trong những câu lệnh này nếu cô ta muốn những user khác có thể thực hiện thủ tục mà trong đó bảng được sử dụng.

Qui tắc này là cần thiết vì tên đối tượng được phân tích khi các thủ tục được chạy. Nếu *marytab* không được chỉ định và user John tìm cách thực hiện thủ tục, SQL

sẽ tìm bảng *marytab* do John sở hữu. Ví dụ dưới đây là một cách dùng đúng, nó chỉ ra cho SQL Server tìm bảng *marytab* do Mary sở hữu:

```
CREATE PROC  p1
AS
    CREATE  INDEX  marytab_ind
    ON mary.marytab(col1)
```

I.5 Đổi tên các thủ tục:

Sử dụng thủ tục:

```
sp_rename old_name, new_name
```

Ta chỉ có thể đổi tên những thủ tục mà ta sở hữu. Người sở hữu CSDL có thể thay đổi tên của bất kỳ thủ tục nào của người sử dụng. Thủ tục được đổi tên phải nằm trong CSDL hiện thời.

Ta phải xoá và tạo lại một thủ tục nếu ta thay đổi tên của một đối tượng được tham chiếu bởi thủ tục đó.

Để có được báo cáo về những đối tượng được tham chiếu bởi một thủ tục, ta sử dụng thủ tục hệ thống: **sp_depends.**

Để xem nội dung của định nghĩa một thủ tục, ta sử dụng thủ tục hệ thống: **sp_helptext**.

I.6. Xoá thủ tục:

Để xoá một thủ tục, ta sử dụng câu lệnh:

```
DROP PROCEDURE  proc_name
```

II. Sử dụng các Trigger

Một trigger là một dạng đặc biệt của thủ tục lưu trữ và nó được thực hiện tự động khi người dùng áp dụng câu lệnh sửa đổi dữ liệu lên một bảng được chỉ định. Các trigger thường được sử dụng cho việc ép buộc các qui tắc làm việc và toàn vẹn dữ liệu. Tính toàn vẹn tham chiếu có thể được định nghĩa bằng cách sử dụng ràng buộc FOREIGN KEY với câu lệnh CREATE TABLE. Nếu các ràng buộc tồn tại trong bảng có sự tác động của trigger, nó được kiểm tra trước việc thực hiện trigger. Nếu các ràng buộc bị vi phạm, trigger sẽ không thực thi.

Các trigger được sử dụng trong những cách sau:

- Các trigger có thể thay đổi đồng loạt (cascade change) các bảng có liên hệ trong một CSDL.
- Các trigger có thể không cho phép hoặc roll back những thay đổi vi phạm tính toàn vẹn tham chiếu, hủy bỏ giao tác sửa đổi dữ liệu.
- Các trigger có thể áp đặt các giới hạn phức tạp hơn những giới hạn được định nghĩa bằng ràng buộc CHECK. Khác với ràng buộc CHECK, các trigger có thể tham chiếu đến các cột trong các bảng khác.
- Các trigger còn có thể tìm sự khác biệt giữa các trạng thái của một bảng trước và sau khi sữa đổi dữ liệu và lấy ra những tác động dựa trên sự khác biệt đó.

II.1 Tạo các trigger

Một trigger là một đối tượng CSDL. Ta tạo một trigger bằng việc chỉ định bảng hiện hành và câu lệnh sửa đổi dữ liệu kích hoạt trigger. Sau đó ta xác định các công việc mà trigger làm.

Một bảng có thể có tối đa 3 loại trigger: một trigger cập nhật (update trigger), một trigger chèn (insert trigger) và một trigger xóa (delete trigger). Tuy nhiên, mỗi trigger có thể thực hiện nhiều hàm và gọi đến 16 thủ tục. Mỗi trigger chỉ có thể áp dụng cho một bảng. Tuy nhiên, một trigger đơn có thể áp dụng cho cả 3 công việc (UPDATE, INSERT và DELETE).

Ta không thể tạo một trigger trên một khung nhìn hay một bảng tạm thời mặc dù các trigger có thể tham chiếu các khung nhìn hay các bảng tạm thời.

Câu lệnh TRUNCATE TABLE mặc dù giống câu lệnh DELETE khi không có mệnh đề WHERE nhưng nó không thể kích hoạt một trigger.

Để tạo mới một trigger, ta sử dụng câu lệnh có cú pháp như sau:

```
CREATE TRIGGER trigger_name
ON table_name
FOR { INSERT, UPDATE, DELETE}
AS   sql_statements
```

Hoặc sử dụng mệnh đề IF UPDATE:

```
CREATE TRIGGER   trigger_name
ON table_name
FOR { INSERT, UPDATE}
AS
  IF UPDATE (column_name)
  [{ AND|OR} UPDATE (column_name)...] sql_statements
```

Ví dụ 4.7: Nếu chúng ta muốn sau khi ta cập nhật dữ liệu cho bảng *nhanvien*, SQL Server sẽ hiển thị nội dung của bảng để xem thì ta tạo một trigger như sau:

```
CREATE TRIGGER  tgr_check
ON nhanvien
FOR INSERT, UPDATE
AS
    print '*** Ket qua sau khi cap nhat  ***'
    SELECT * FROM nhanvien
```

II.2 Các giá trị null ngầm định và hiển (implicit and explicit null values)

Mệnh đề IF UPDATE(*tên_cột*) là đúng cho một câu lệnh INSERT khi mà cột được gán một giá trị trong danh sách chọn hay trong mệnh đề VALUES. Một NULL hiển (explicit) hay một mặc định gán một giá trị cho một cột và vì thế kích hoạt trigger. Với một NULL ngầm định, nếu giá trị không được xác định bởi câu hỏi hoặc bởi mặc định được gán, trigger trên cột đó không được kích hoạt.

Ví dụ 4.8:

```
CREATE TABLE vidu(col1 int NULL,col2 int NOT NULL)
GO
CREATE TRIGGER tgr_vidu
ON  vidu
```

```
FOR  INSERT
AS
  IF UPDATE(col1) AND UPDATE(col2)
           Print 'Firing'
GO
CREATE DEFAULT col2_default
     AS  99
GO
```

/ IF UPDATE là đúng cho cả hai cột, trigger được kích hoạt */*

```
INSERT vidu(col1,col2) VALUES(1, 2)
```

/ IF UPDATE là đúng cho cả hai cột, trigger được kích hoạt */*

```
INSERT vidu VALUES(1, 2)
```

/ NULL hiển: IF UPDATE là đúng cho cả hai cột, trigger được kích hoạt */*

```
INSERT vidu VALUES(null, 2)
```

/ Không có mặc định trên cột col1, IF UPDATE không đúng cho cả hai cột, trigger không được kích hoạt */*

```
INSERT vidu(col2) VALUES(2)
```

/ Không có mặc định trên cột col2, IF UPDATE không đúng cho cả hai cột, trigger không được kích hoạt */*

```
INSERT vidu(col1) VALUES(2)
```

Kết quả tương tự được sản sinh với việc sử dụng chỉ mệnh đề

```
IF UPDATE(col1)
```

Để tạo một trigger không cho phép việc chèn các giá trị null ngầm định, ta sử dụng:

```
IF UPDATE(col2) OR UPDATE(col2)
```

Câu lệnh SQL trong trigger có thể sau đó kiểm tra xem col1 là NULL hay không.

II.3 Việc đổi tên và các trigger

Nếu một bảng được tham chiếu bởi một trigger bị đổi tên, ta phải xoá trigger đó đi và tạo lại nó để phù hợp việc tham chiếu của nó đến bảng.

Thủ tục **sp_depends** có chức năng liệt kê tất cả các trigger tham chiếu đến đối tượng (chẳng hạn bảng hay khung nhìn) hoặc tất cả các bảng hay khung nhìn mà trigger tác động. Ví dụ sau đây liệt kê các đối tượng được tham chiếu bởi trigger *tgr_check*:

```
sp_depends  tgr_check
```

II.4 Hiển thị thông tin về các trigger

Do các trigger là các đối tượng CSDL nên chúng được liệt kê trong bảng hệ thống *sysobjects*. Cột *type* trong *sysobjects* xác định các trigger với chữ viết tắt TR. Sơ đồ thực thi các trigger được lưu trữ trong bảng *sysprocedures*.

Truy vấn dưới đây tìm các trigger trong một CSDL:

```
SELECT * FROM sysobjects WHERE type=' TR'
```

Để hiển thị thông tin về một trigger ta thực hiện thủ tục:

```
sp_help   trigger_name
```

Câu lệnh CREATE TRIGGER cho mỗi trigger được lưu trữ trong bảng hệ thống *syscomments*. Ta có thể hiển thị lời định nghĩa trigger bằng cách sử dụng thủ tục **sp_helptext**.

Ví dụ 4.9: thực hiện *sp_helptext tgr_check* ta được kết quả như sau:

```
text
--------------
create trigger tgr_check
on nhanvien
for insert,update
as
   print '***** Ket qua sau khi cap nhat *****'
   select * from nhanvien
```

II.5 Xoá trigger

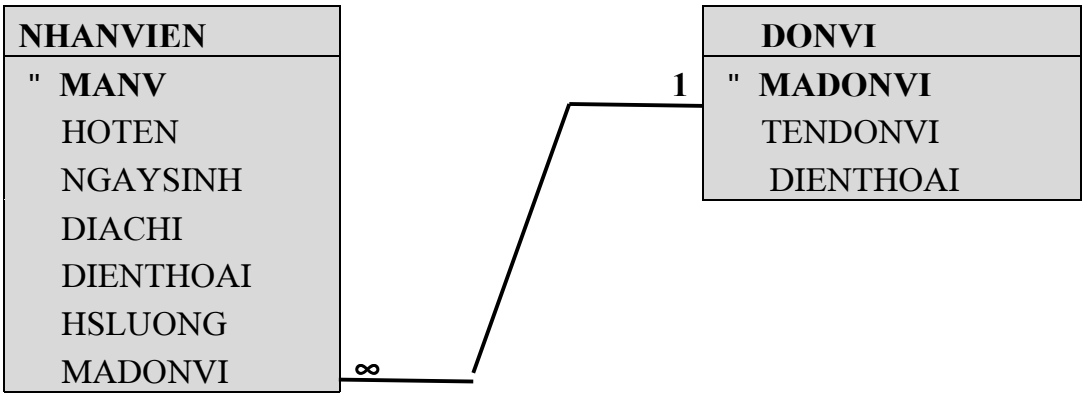
Ta có thể xoá một trigger bằng cách xoá nó hoặc xoá bảng trigger. Khi một bảng được xóa, những trigger nào có liên quan với nó cũng đồng thời bị xóa. DROP TRIGGER mặc định cho phép đối với người sử dụng bảng trigger và không thể chuyển cho người khác.

Ta có thể xóa một trigger bằng cách sử dụng câu lệnh DROP TRIGGER

Chương 5: PHỤ LỤC

Trong chương này, chúng tôi giới thiệu cấu trúc và dữ liệu của các bảng được sử dụng trong các ví dụ trong chương 2. Sau đó giới thiệu một số hàm thường sử dụng trong SQL Server để giúp các bạn tham khảo trong quá trình thực hành.

1. Cấu trúc và dữ liệu của bảng NHANVIEN và DONVI



Bảng NHANVIEN

MANV	HOTEN	NGAYSINH	DIACHI	DIENTHOAI	HSLUONG	MADONVI
NV01001	Nguyễn Thị Hoa	05/05/1976	56 Lê Duẩn	521304	2.11	01
NV02001	Lê Hoài Nam	03/05/1976	32 Trần Phú	823145	1.86	02
NV02002	Hoàng Nam Phong	05/08/1971	66 Hoàng Diệu	521247	3.21	02
NV03001	Trần Nguyên Phong	20/12/1976	7 Hà Nội	849290	1.92	03
NV03002	Nguyễn Hữu Tình	18/08/1976	7 Hà Nội	849290	1.92	03
NV05001	Nguyễn Trung Kiên	14/05/1972	77 Nguyễn Huệ	823236	1.86	05

Bảng DONVI

MADONVI	TENDONVI	DIENTHOAI
01	Phòng Kế toán	821451
02	Phòng Tổ chức	831414
03	Phòng điều hành	823351
04	Phòng đối ngoại	841457
05	Phong Tài vụ	821451

2. Một số hàm thường sử dụng trong SQL Server

2.1 Các hàm trên dữ liệu kiểu ngày và giờ

a. Hàm DATEADD

Cú pháp: DATEADD(*datepart*, *number*, *date*)

Chức năng: Hàm trả về một giá trị kiểu DateTime bằng cách cộng thêm một khoảng giá trị là *number* vào ngày *date* được chỉ định.

Datepart: tham số chỉ định thành phần sẽ được cộng đối với giá trị *date* bao gồm:

Datepart	**Viết tắt**
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
hour	hh
minute	mi, n
second	ss, s
millisecond	ms

b. Hàm DATEDIFF

Cú pháp: DATEDIFF(*datepart*, *startdate*, *enddate*)

Chức năng: Hàm trả về khoảng thời gian giữa hai giá trị kiểu này được chỉ định tuỳ thuộc vào tham số *datepart*

Ví dụ: hàm *Datediff(year,'5/20/1976','8/9/2001')* cho kết quả là 25

c. Hàm DATEPART

Cú pháp: DATEPART(*datepart, date*)

Chức năng: Hàm trả về một số nguyên được trích ra từ thành phần (được chỉ định bởi tham số *partdate*) trong giá trị kiểu ngày được chỉ định.

Ví dụ: Hàm DatePart(year,'5/20/1976') cho kết quả là 1976

d. Hàm GETDATE

Cú pháp: GETDATE()

Chức năng: Hàm trả về ngày hiện tại

e. Hàm DAY, MONTH, YEAR

Cú pháp: DAY(*date*), MONTH(*date*), YEAR(*date*)

Chức năng: Hàm trả về giá trị ngày (tương ứng tháng, năm) của giá trị kiểu ngày được chỉ định.

2.2 Các hàm về chuỗi

a. Hàm LEFT

Cú pháp: LEFT(*string*, *n*)

Chức năng:Hàm trích ra từ chuỗi *string* n ký tự tính từ bên trái

b. Hàm RIGHT

Cú pháp: LEFT(*string*, *n*)

Chức năng: Hàm trích ra từ chuỗi *string* n ký tự tính từ bên phải

c. Hàm SUBSTRING

Cú pháp: SUBSTRING(*string*, *m*, *n*)

Chức năng: Hàm trích ra từ chuỗi s*tring* n ký tự tính từ ký tự thứ m

d. Hàm LTRIM, RTRIM

Cú pháp: LTRIM(*string*), RTRIM(*string*)

Chức năng: Hàm cắt bỏ các khoảng trắng thừa bên trái/ bên phải chuỗi *string*.

e. Hàm LEN

Cú pháp: LEN(*string*)

Chức năng: Hàm trả về độ dài của chuỗi *string*.

MỤC LỤC