TRƯỜNG ĐẠI HỌC KHOA HỌC HUẾ
KHOA CÔNG NGHỆ THÔNG TIN

GIÁO TRÌNH SQL

(*Lưu hành nội bộ*)

Biên soạn: Trần Nguyên Phong

Huế, 2004

MỤC LỤC

MỤC LỤC 2
LỜI NÓI ĐẦU 5
CHƯƠNG 1: TỔNG QUAN VỀ SQL 7
1.1 SQL là ngôn ngữ cơ sở dữ liệu quan hệ 7
1.2 Vai trò của SQL 8
1.3 Tổng quan về cơ sở dữ liệu quan hệ 9
1.3.1 Mô hình dữ liệu quan hệ 9
1.3.2 Bảng (Table) 9
1.3.3 Khoá của bảng 10
1.3.4 Mối quan hệ và khoá ngoài 11
1.4 Sơ lược về SQL 12
1.4.1 Câu lệnh SQL 12
1.4.2 Qui tắc sử dụng tên trong SQL 14
1.4.3 Kiểu dữ liệu 14
1.4.4 Giá trị NULL 16
1.5 Kết chương 16
CHƯƠNG 2: NGÔN NGỮ THAO TÁC DỮ LIỆU 18
2.1 Truy xuất dữ liệu với câu lệnh SELECT 18
2.1.1 Mệnh đề FROM 19
2.1.2 Danh sách chọn trong câu lệnh SELECT 20
2.1.3 Chỉ định điều kiện truy vấn dữ liệu 25
2.1.4 Tạo mới bảng dữ liệu từ kết quả của câu lệnh SELECT 29
2.1.5 Sắp xếp kết quả truy vấn 29
2.1.6 Phép hợp 31
2.1.7 Phép nối 33
2.1.7.1 Sử dụng phép nối 34
2.1.7.2 Các loại phép nối 36
2.1.7.4 Sử dụng phép nối trong SQL2 40
2.1.8 Thống kê dữ liệu với GROUP BY 43
2.1.9 Thống kê dữ liệu với COMPUTE 46
2.1.10 Truy vấn con (Subquery) 49
2.2 Bổ sung, cập nhật và xoá dữ liệu 53
2.2.1 Bổ sung dữ liệu 53
2.2.2 Cập nhật dữ liệu 54
2.2.3 Xoá dữ liệu 56
📖 Bài tập chương 2 58
CHƯƠNG 3: NGÔN NGỮ ĐỊNH NGHĨA DỮ LIỆU 69

3.1 Tạo bảng dữ liệu ... 69
3.1.1 Ràng buộc CHECK ... 72
3.1.2 Ràng buộc PRIMARY KEY ... 74
3.1.3 Ràng buộc UNIQUE ... 76
3.1.4 Ràng buộc FOREIGN KEY ... 76
3.2 Sửa đổi định nghĩa bảng ... 79
3.3 Xoá bảng ... 81
3.4 Khung nhìn ... 82
3.4.1 Tạo khung nhìn ... 84
3.4.2 Cập nhật, bổ sung và xoá dữ liệu thông qua khung nhìn ... 86
3.4.3 Sửa đổi khung nhìn ... 89
3.4.4 Xoá khung nhìn ... 90
📖 Bài tập chương 3 ... 90

CHƯƠNG 4: BẢO MẬT TRONG SQL ... 96

4.1 Các khái niệm ... 96
4.2 Cấp phát quyền ... 97
4.2.1 Cấp phát quyền cho người dùng trên các đối tượng cơ sở dữ liệu ... 97
4.2.2 Cấp phát quyền thực thi các câu lệnh ... 99
4.3 Thu hồi quyền ... 100
4.3.1 Thu hồi quyền trên đối tượng cơ sở dữ liệu: ... 100
4.3.2 Thu hồi quyền thực thi các câu lênh: ... 103

CHƯƠNG 5: THỦ TỤC LƯU TRỮ, HÀM VÀ TRIGGER ... 104

5.1 Thủ tục lưu trữ (stored procedure) ... 104
5.1.1 Các khái niệm ... 104
5.1.2 Tạo thủ tục lưu trữ ... 105
5.1.3 Lời gọi thủ tục lưu trữ ... 107
5.1.4 Sử dụng biến trong thủ tục ... 107
5.1.5 Giá trị trả về của tham số trong thủ tục lưu trữ ... 108
5.1.6 Tham số với giá trị mặc định ... 109
5.1.7 Sửa đổi thủ tục ... 110
5.2 Hàm do người dùng định nghĩa ... 111
5.2.1 Định nghĩa và sử dụng hàm ... 111
5.2.2 Hàm với giá trị trả về là "dữ liệu kiểu bảng" ... 112
5.3 Trigger ... 116
5.3.1 Định nghĩa trigger ... 117
5.3.2 Sử dụng mệnh đề IF UPDATE trong trigger ... 119
5.3.3 ROLLBACK TRANSACTION và trigger ... 121
5.3.4 Sử dụng trigger trong trường hợp câu lệnh INSERT, UPDATE và DELETE có tác động đến nhiều dòng dữ liệu ... 122
5.3.4.1 Sử dụng truy vấn con ... 122
5.3.4.2 Sử dụng biến con trỏ ... 125
📖 Bài tập chương 5 ... 127

CHƯƠNG 6: GIAO TÁC SQL ... 132

6.1 Giao tác và các tính chất của giao tác ... 132
6.2 Mô hình giao tác trong SQL ... 133

6.3 Giao tác lồng nhau 136
PHỤ LỤC 138
A. Cơ sở dữ liệu mẫu sử dụng trong giáo trình 138
B. Một số hàm thường sử dụng 141
B.1 Các hàm trên dữ liệu kiểu chuỗi 141
B.2 Các hàm trên dữ liệu kiểu ngày giờ 143
B.3 Hàm chuyển đổi kiểu 144
TÀI LIỆU THAM KHẢO 146

LỜI NÓI ĐẦU

Ngôn ngữ hỏi có cấu trúc (SQL), có tiền thân là SEQUEL, là một ngôn ngữ được IBM phát triển và sử dụng trong hệ cơ sở dữ liệu thử nghiệm có tên là System/R vào năm 1974, chính thức được ANSI/ISO công nhận là một chuẩn ngôn ngữ sử dụng trong cơ sở dữ liệu quan hệ vào năm 1986. Cho đến hiện nay, SQL đã được sử dụng phổ biến trong các hệ quản trị cơ sở dữ liệu thương mại và có vai trò quan trọng trong những hệ thống này.

Được sự động viên của các đồng nghiệp trong Khoa Công nghệ Thông tin (Trường Đại học Khoa học - Đại học Huế), chúng tôi mạnh dạn viết và giới thiệu G*iáo trình SQL* đến bạn đọc. Trong giáo trình này, chúng tôi không có tham vọng đề cập đến mọi khía cạnh của SQL mà chỉ mong muốn rằng đây sẽ là tài liệu tham khảo tương đối đầy đủ về các câu lệnh thường được sử dụng trong SQL. Giáo trình được chia thành sáu chương với nội dung như sau:

- Chương 1 giới thiệu tổng quan về SQL và một số khái cơ bản liên quan đến cơ sở dữ liệu quan hệ.
- Chương 2 được dành để bàn luận đến các câu lệnh thao tác dữ liệu bao gồm SELECT, INSERT, UPDATE và DELETE, trong đó tập trung nhiều vào câu lệnh SELECT.
- Chương 3 trình bày một số câu lệnh cơ bản được sử dụng trong định nghĩa các đối tượng cơ sở dữ liệu.
- Một số vấn đề liên quan đến bảo mật dữ liệu trong SQL được đề cập đến trong chương 4.
- Nội dung của chương 5 liên quan đến việc sử dụng thủ tục lưu trữ, hàm và trigger trong cơ sở dữ liệu.
- Trong chương cuối cùng, chương 6, chúng tôi giới thiệu đến bạn đọc một số vấn đề liên quan đến xử lý giao tác trong SQL

Ngoài sáu chương trên, phần phụ lục ở cuối giáo trình đề cập đến cơ sở dữ liệu mẫu được sử dụng trong hầu hết các ví dụ và một số hàm thường được sử dụng trong hệ quản trị SQL Server 2000 để bạn đọc tiện trong việc tra cứu.

So với chuẩn SQL do ANSI/ISO đề xuất, bản thân các hệ quản trị cơ sở dữ liệu quan hệ thương mại lại có thể có một số thay đổi nào đó; Điều này đôi khi dẫn đến sự khác biệt, mặc dù không đáng kể, giữa SQL chuẩn và SQL được sử dụng trong các hệ quản trị cơ sở dữ liệu cụ thể. Trong giáo trình này, chúng tôi chọn hệ quản trị cơ sở dữ

liệu SQL Server 2000 của hãng Microsoft để sử dụng cho các ví dụ minh hoạ cũng như lời giải của các bài tập.

Chúng tôi hi vọng rằng giáo trình này sẽ thực sự có ích đối với bạn đọc. Chúng tôi rất mong nhận được sự cổ vũ và những ý kiến đóng góp thẳng thắn của các bạn.

Cuối cùng, xin gởi lời cảm ơn đến các thầy cô, đồng nghiệp và các bạn sinh viên đã động viên và giúp đỡ chúng tôi hoàn thành giáo trình này.

Huế, 2003

Trần Nguyên Phong

Chương 1:

TỔNG QUAN VỀ SQL

Ngôn ngữ hỏi có cấu trúc (SQL) và các hệ quản trị cơ sở dữ liệu quan hệ là một trong những nền tảng kỹ thuật quan trọng trong công nghiệp máy tính. Cho đến nay, có thể nói rằng SQL đã được xem là ngôn ngữ chuẩn trong cơ sở dữ liệu. Các hệ quản trị cơ sở dữ liệu quan hệ thương mại hiện có như Oracle, SQL Server, Informix, DB2,... đều chọn SQL làm ngôn ngữ cho sản phẩm của mình

Vậy thực sự SQL là gì? Tại sao nó lại quan trọng trong các hệ quản trị cơ sở dữ liệu? SQL có thể làm được những gì và như thế nào? Nó được sử dụng ra sao trong các hệ quản trị cơ sở dữ liệu quan hệ? Nội dung của chương này sẽ cung cấp cho chúng ta cái nhìn tổng quan về SQL và một số vấn đề liên quan.

1.1 SQL là ngôn ngữ cơ sở dữ liệu quan hệ

SQL, viết tắt của *Structured Query Language* (ngôn ngữ hỏi có cấu trúc), là công cụ sử dụng để tổ chức, quản lý và truy xuất dữ liệu được lưu trữ trong các cơ sở dữ liệu. SQL là một hệ thống ngôn ngữ bao gồm tập các câu lệnh sử dụng để tương tác với cơ sở dữ liệu quan hệ.

Tên gọi *ngôn ngữ hỏi có cấu trúc* phần nào làm chúng ta liên tưởng đến một công cụ (ngôn ngữ) dùng để truy xuất dữ liệu trong các cơ sở dữ liệu. Thực sự mà nói, khả năng của SQL vượt xa so với một công cụ truy xuất dữ liệu, mặc dù đây là mục đích ban đầu khi SQL được xây dựng nên và truy xuất dữ liệu vẫn còn là một trong những chức năng quan trọng của nó. SQL được sử dụng để điều khiển tất cả các chức năng mà một hệ quản trị cơ sở dữ liệu cung cấp cho người dùng bao gồm:

- **Định nghĩa dữ liệu:** SQL cung cấp khả năng định nghĩa các cơ sở dữ liệu, các cấu trúc lưu trữ và tổ chức dữ liệu cũng như mối quan hệ giữa các thành phần dữ liệu.
- **Truy xuất và thao tác dữ liệu:** Với SQL, người dùng có thể dễ dàng thực hiện các thao tác truy xuất, bổ sung, cập nhật và loại bỏ dữ liệu trong các cơ sở dữ liệu.
- **Điều khiển truy cập:** SQL có thể được sử dụng để cấp phát và kiểm soát các thao tác của người sử dụng trên dữ liệu, đảm bảo sự an toàn cho cơ sở dữ liệu

- **Đảm bảo toàn vẹn dữ liệu:** SQL định nghĩa các ràng buộc toàn vẹn trong cơ sở dữ liệu nhờ đó đảm bảo tính hợp lệ và chính xác của dữ liệu trước các thao tác cập nhật cũng như các lỗi của hệ thống.

Như vậy, có thể nói rằng SQL là một ngôn ngữ hoàn thiện được sử dụng trong các hệ thống cơ sở dữ liệu và là một thành phần không thể thiếu trong các hệ quản trị cơ sở dữ liệu. Mặc dù SQL không phải là một ngôn ngữ lập trình như C, C++, Java,... song các câu lệnh mà SQL cung cấp có thể được nhúng vào trong các ngôn ngữ lập trình nhằm xây dựng các ứng dụng tương tác với cơ sở dữ liệu.

Khác với các ngôn ngữ lập trình quen thuộc như C, C++, Java,... SQL là ngôn ngữ có tính khai báo. Với SQL, người dùng chỉ cần mô tả các yêu cầu cần phải thực hiện trên cơ sở dữ liệu mà không cần phải chỉ ra cách thức thực hiện các yêu cầu như thế nào. Chính vì vậy, SQL là ngôn ngữ dễ tiếp cận và dễ sử dụng.

1.2 Vai trò của SQL

Bản thân SQL không phải là một hệ quản trị cơ sở dữ liệu, nó không thể tồn tại độc lập. SQL thực sự là một phần của hệ quản trị cơ sở dữ liệu, nó xuất hiện trong các hệ quản trị cơ sở dữ liệu với vai trò ngôn ngữ và là công cụ giao tiếp giữa người sử dụng và hệ quản trị cơ sở dữ liệu.

Trong hầu hết các hệ quản trị cơ sở dữ liệu quan hệ, SQL có những vai trò như sau:

- **SQL là ngôn ngữ hỏi có tính tương tác:** Người sử dụng có thể dễ dàng thông qua các trình tiện ích để gởi các yêu cầu dưới dạng các câu lệnh SQL đến cơ sở dữ liệu và nhận kết quả trả về từ cơ sở dữ liệu
- **SQL là ngôn ngữ lập trình cơ sở dữ liệu**: Các lập trình viên có thể nhúng các câu lệnh SQL vào trong các ngôn ngữ lập trình để xây dựng nên các chương trình ứng dụng giao tiếp với cơ sở dữ liệu
- **SQL là ngôn ngữ quản trị cơ sở dữ liệu:** Thông qua SQL, người quản trị cơ sở dữ liệu có thể quản lý được cơ sở dữ liệu, định nghĩa các cấu trúc lưu trữ dữ liệu, điều khiển truy cập cơ sở dữ liệu,...
- **SQL là ngôn ngữ cho các hệ thống khách/chủ (client/server):** Trong các hệ thống cơ sở dữ liệu khách/chủ, SQL được sử dụng như là công cụ để giao tiếp giữa các trình ứng dụng phía máy khách với máy chủ cơ sở dữ liệu.
- **SQL là ngôn ngữ truy cập dữ liệu trên Internet:** Cho đến nay, hầu hết các máy chủ Web cũng như các máy chủ trên Internet sử dụng SQL với vai trò là ngôn ngữ để tương tác với dữ liệu trong các cơ sở dữ liệu.
- **SQL là ngôn ngữ cơ sở dữ liệu phân tán:** Đối với các hệ quản trị cơ sở dữ liệu phân tán, mỗi một hệ thống sử dụng SQL để giao tiếp với các hệ thống khác trên mạng, gởi và nhận các yêu cầu truy xuất dữ liệu với nhau.

- **SQL là ngôn ngữ sử dụng cho các cổng giao tiếp cơ sở dữ liệu:** Trong một hệ thống mạng máy tính với nhiều hệ quản trị cơ sở dữ liệu khác nhau, SQL thường được sử dụng như là một chuẩn ngôn ngữ để giao tiếp giữa các hệ quản trị cơ sở dữ liệu.

1.3 Tổng quan về cơ sở dữ liệu quan hệ

1.3.1 Mô hình dữ liệu quan hệ

Mô hình dữ liệu quan hệ được Codd đề xuất năm 1970 và đến nay trở thành mô hình được sử dụng phổ biến trong các hệ quản trị cơ sở dữ liệu thương mại. Nói một cách đơn giản, một cơ sở dữ liệu quan hệ là một cơ sở dữ liệu trong đó tất cả dữ liệu được tổ chức trong các bảng có mối quan hệ với nhau. Mỗi một bảng bao gồm các dòng và các cột: mỗi một dòng được gọi là một bản ghi (bộ) và mỗi một cột là một trường (thuộc tính).

Hình 1.1 minh hoạ cho ta thấy được 3 bảng trong một cơ sở dữ liệu

Bảng KHOA

MAKHOA	TENKHOA	DIENTHOAI
DHTO1	Khoa Toán cơ - Tin học	054822407
DHTO2	Khoa Công nghệ thông tin	054826767
DHTO3	Khoa Vật lý	[illegible]
DHTO4	Khoa Hoá học	
...	...	

Bảng LOP

MALOP	TENLOP	KHOA	HEDAOTAO	NAMNHAPHOC	SISO	MAKHOA
C24101	Toán K24	24	Chính quy	2000	5	DHTO1
C24102	Tin K24	24	Chính quy	2000	8	DHTO2
C24103	Lý K24	24	Chính quy	2000	7	DHTO3
C24301	Sinh K24	24	Chính quy	2000	5	DHTO5

Bảng SINHVIEN

MASV	HODEM	TEN	NGAYSINH	GIOTINH	NOISINH	MALOP
0241010001	Ngô Thị Nhật	Anh	Nov 27 1982	0	Quảng Ninh, Quảng Bình	C24101
0241010002	Nguyễn Thị Ngọc	Anh	Mar 21 1983	0	Tân Kỳ, Nghệ An	C24101
0241010003	Ngô Việt	Bắc	May 11 1982	1	Yên Khánh, Ninh Bình	C24101
0241010004	Nguyễn Đình	Bình	Oct 6 1982	1	Huế	C24101
0241010005	Hồ Đăng	Chiến	Jan 20 1982	1	Phong Điền, TTHuế	C24101
0241020001	Nguyễn Tuấn	Anh	Jul 15 1979	1	Do Linh, Quảng Trị	C24102
0241020002	Trần Thị Kim	Anh	Nov 4 1982	0	Phong Điền, TTHuế	C24102
0241020003	Võ Đức	Ẩn	May 24 1982	1	Huế	C24102
0241020004	Nguyễn Công	Bình	Jun 6 1979	1	Thăng Bình, Quảng Nam	C24102
0241020005	Nguyễn Thanh	Bình	Apr 24 1982	1	Huế	C24102
...	...	...	...	...	...	...

Hình 1.1: *Các bảng trong một cơ sở dữ liệu*

1.3.2 Bảng (Table)

Như đã nói ở trên, trong cơ sở dữ liệu quan hệ, bảng là đối tượng được sử dụng để tổ chức và lưu trữ dữ liệu. Một cơ sở dữ liệu bao gồm nhiều bảng và mỗi bảng được xác định duy nhất bởi tên bảng. Một bảng bao gồm một tập các dòng và các cột: mỗi

một dòng trong bảng biểu diễn cho một thực thể (trong hình 1.1, mỗi một dòng trong bảng SINHVIEN tương ứng với một sinh viên); và mỗi một cột biểu diễn cho một tính chất của thực thể (chẳng hạn cột NGAYSINH trong bảng SINHVIEN biểu diễn cho ngày sinh của các sinh viên được lưu trữ trong bảng).

Như vậy, liên quan đến mỗi một bảng bao gồm các yếu tố sau:

- **Tên của bảng:** được sử dụng để xác định duy nhất mỗi bảng trong cơ sở dữ liệu.
- **Cấu trúc của bảng:** Tập các cột trong bảng. Mỗi một cột trong bảng được xác định bởi một *tên cột* và phải có một kiểu dữ liệu nào đó (chẳng hạn cột NGAYSINH trong bảng SINHVIEN ở hình 1.1 có kiểu là DATETIME). Kiểu dữ liệu của mỗi cột qui định giá trị dữ liệu có thể được chấp nhận trên cột đó.
- **Dữ liệu của bảng:** Tập các dòng (bản ghi) hiện có trong bảng.

1.3.3 Khoá của bảng

Trong một cơ sở dữ liệu được thiết kế tốt, mỗi một bảng phải có một hoặc một tập các cột mà giá trị dữ liệu của nó xác định duy nhất một dòng trong một tập các dòng của bảng. Tập một hoặc nhiều cột có tính chất này được gọi là khoá của bảng.

Việc chọn khoá của bảng có vai trò quan trọng trong việc thiết kế và cài đặt các cơ sở dữ liệu quan hệ. Các dòng dữ liệu trong một bảng phải có giá trị khác nhau trên khoá. Bảng MONHOC trong hình dưới đây có khoá là cột MAMONHOC

MAMONHOC	TENMONHOC	SODVHT
HO-001	Hoá đại cương	3
TI-001	Tin học đại cương	4
TI-002	Ngôn ngữ C	5
TI-003	Lý thuyết hệ điều hành	4
TI-004	Cấu trúc dữ liệu và giải thuật	4
TO-001	Đại số tuyến tính	4
TO-002	Giải tích 1	4
TO-003	Bài tập Đại số	2
TO-004	Bài tập Giải tích 1	2
VL-001	Vật lý đại cương	3

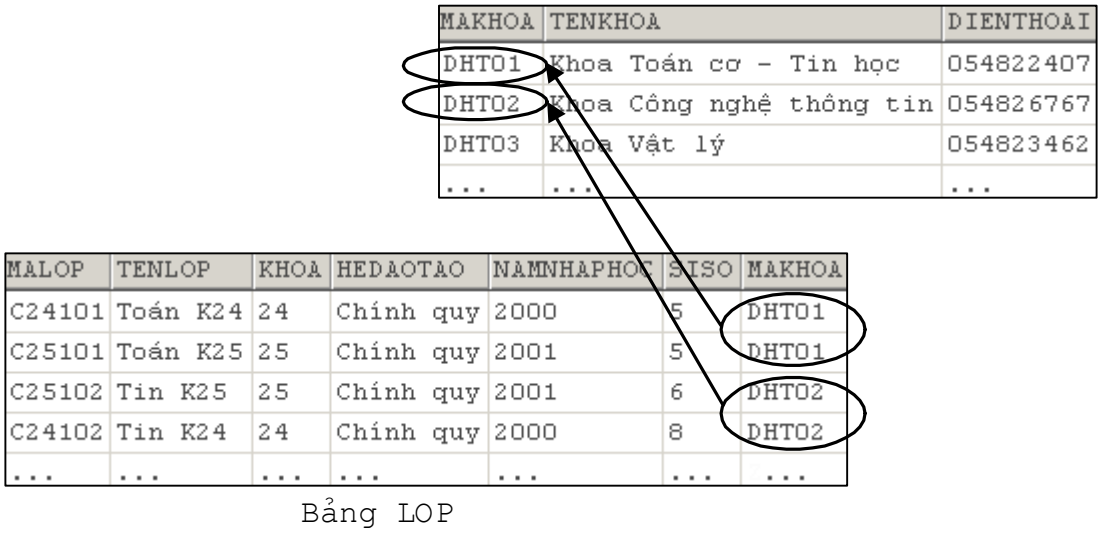
Hình 1.2: *Bảng MONHOC với khoá chính là MAMONHOC*

Một bảng có thể có nhiều tập các cột khác nhau có tính chất của khoá (tức là giá trị của nó xác định duy nhất một dòng dữ liệu trong bảng). Trong trường hợp này, khoá được chọn cho bảng được gọi là *khoá chính* (*primary key*) và những khoá còn lại được gọi là khoá phụ hay là *khoá dự tuyển* (*candidate key*/*unique key*).

1.3.4 Mối quan hệ và khoá ngoài

Các bảng trong một cơ sở dữ liệu không tồn tại độc lập mà có mối quan hệ mật thiết với nhau về mặt dữ liệu. Mối quan hệ này được thể hiện thông qua ràng buộc *giá trị dữ liệu xuất hiện ở bảng này phải có xuất hiện trước trong một bảng khác*. Mối quan hệ giữa các bảng trong cơ sở dữ liệu nhằm đảm bảo được tính đúng đắn và hợp lệ của dữ liệu trong cơ sở dữ liệu.

Trong hình 1.3, hai bảng LOP và KHOA có mối quan hệ với nhau. Mối quan hệ này đòi hỏi giá trị cột MAKHOA của một dòng (tức là một lớp) trong bảng LOP phải được xác định từ cột MAKHOA của bảng KHOA.



MAKHOA	TENKHOA	DIENTHOAI
DHTO1	Khoa Toán cơ - Tin học	054822407
DHTO2	Khoa Công nghệ thông tin	054826767
DHTO3	Khoa Vật lý	054823462
...	...	...

MALOP	TENLOP	KHOA	HEDAOTAO	NAMNHAPHOC	SISO	MAKHOA
C2401	Toán K24	24	Chính quy	2000	5	DHTO1
C2501	Toán K25	25	Chính quy	2001	5	DHTO1
C2502	Tin K25	25	Chính quy	2001	6	DHTO2
C2402	Tin K24	24	Chính quy	2000	8	DHTO2
...	...	...	...	...	...	...

Bảng LOP

***Hình 1.3**: Mối quan hệ giữa hai bảng LOP và KHOA trong cơ sở dữ liệu*

Mối quan hệ giữa các bảng trong một cơ sở dữ liệu thể hiện đúng mối quan hệ giữa các thực thể trong thế giới thực. Trong hình 1.3, mối quan hệ giữa hai bảng LOP và KHOA không cho phép một lớp nào đó tồn tại mà lại thuộc vào một khoa không có thật.

Khái niệm *khoá ngoài* (*Foreign Key*) trong cơ sở dữ liệu quan hệ được sử dụng để biểu diễn mối quan hệ giữa các bảng dữ liệu. Một hay một tập các cột trong một bảng mà giá trị của nó được xác định từ khóa chính của một bảng khác được gọi là khoá ngoài. Trong hình 1.3, cột MAKHOA của bảng LOP được gọi là khoá ngoài của bảng này, khoá ngoài này tham chiếu đến khoá chính của bảng KHOA là cột MAKHOA.

1.4 Sơ lược về SQL

1.4.1 Câu lệnh SQL

SQL chuẩn bao gồm khoảng 40 câu lệnh. Bảng 1.1 liệt kê danh sách các câu lệnh thường được sử dụng nhất trong số các câu lệnh của SQL. Trong các hệ quản trị cơ sở dữ liệu khác nhau, mặc dù các câu lệnh đều có cùng dạng và cùng mục đích sử dụng song mỗi một hệ quản trị cơ sở dữ liệu có thể có một số thay đổi nào đó. Điều này đôi khi dẫn đến cú pháp chi tiết của các câu lệnh có thể sẽ khác nhau trong các hệ quản trị cơ cơ sở dữ liệu khác nhau.

Câu lệnh	**Chức năng**
Thao tác dữ liệu	
SELECT	Truy xuất dữ liệu
INSERT	Bổ sung dữ liệu
UPDATE	Cập nhật dữ liệu
DELETE	Xoá dữ liệu
TRUNCATE	Xoá toàn bộ dữ liệu trong bảng
Định nghĩa dữ liệu	
CREATE TABLE	Tạo bảng
DROP TABLE	Xoa bảng
ALTER TABLE	Sửa đổi bảng
CREATE VIEW	Tạo khung nhìn
ALTER VIEW	Sửa đổi khung nhìn
DROP VIEW	Xoá khung nhìn
CREATE INDEX	Tạo chỉ mục
DROP INDEX	Xoá chỉ mục
CREATE SCHEMA	Tạo lược đồ cơ sở dữ liệu
DROP SCHEMA	Xoá lược đồ cơ sở dữ liệu
CREATE PROCEDURE	Tạo thủ tục lưu trữ
ALTER PROCEDURE	Sửa đổi thủ tục lưu trữ
DROP PROCEDURE	Xoá thủ tục lưu trữ

CREATE FUNCTION	Tạo hàm (do người sử dụng định nghĩa)
ALTER FUNCTION	Sửa đổi hàm
DROP FUNCTION	Xoá hàm
CREATE TRIGGER	Tạo trigger
ALTER TRIGGER	Sửa đổi trigger
DROP TRIGGER	Xoá trigger
Điều khiển truy cập	
GRANT	Cấp phát quyền cho người sử dụng
REVOKE	Thu hồi quyền từ người sử dụng
Quản lý giao tác	
COMMIT	Uỷ thác (kết thúc thành công) giao tác
ROLLBACK	Quay lui giao tác
SAVE TRANSACTION	Đánh dấu một điểm trong giao tác
Lập trình	
DECLARE	Khai báo biến hoặc định nghĩa con trỏ
OPEN	Mở một con trỏ để truy xuất kết quả truy vấn
FETCH	Đọc một dòng trong kết quả truy vấn (sử dụng con trỏ)
CLOSE	Đóng một con trỏ
EXECUTE	Thực thi một câu lệnh SQL

Bảng 1.1: *Một số câu lệnh thông dụng trong SQL*

Các câu lệnh của SQL đều được bắt đầu bởi các từ lệnh, là một từ khoá cho biết chức năng của câu lệnh (chẳng hạn SELECT, DELETE, COMMIT). Sau từ lệnh là các mệnh đề của câu lệnh. Mỗi một mệnh đề trong câu lệnh cũng được bắt đầu bởi một từ khoá (chẳng hạn FROM, WHERE,...).

Ví dụ 1.1: Câu lệnh:

```
SELECT masv,hodem,ten
FROM sinhvien
WHERE malop='C24102'
```

dùng để truy xuất dữ liệu trong bảng SINHVIEN được bắt đầu bởi từ lệnh SELECT, trong câu lệnh bao gồm hai mệnh đề: mệnh đề FROM chỉ định tên của bảng cần truy xuất dữ liệu và mệnh đề WHERE chỉ định điều kiện truy vấn dữ liệu.

1.4.2 Qui tắc sử dụng tên trong SQL

Các đối tượng trong cơ sở dữ liệu dựa trên SQL được xác định thông qua tên của đối tượng. Tên của các đối tượng là duy nhất trong mỗi cơ sở dữ liệu. Tên được sử dụng nhiều nhất trong các truy vấn SQL và được xem là nền tảng trong cơ sở dữ liệu quan hệ là tên bảng và tên cột.

Trong các cơ sở dữ liệu lớn với nhiều người sử dụng, khi ta chỉ định tên của một bảng nào đó trong câu lệnh SQL, hệ quản trị cơ sở dữ liệu hiểu đó là tên của bảng do ta sở hữu (tức là bảng do ta tạo ra). Thông thường, trong các hệ quản trị cơ sở dữ liệu này cho phép những người dùng khác nhau tạo ra những bảng trùng tên với nhau mà không gây ra xung đột về tên. Nếu trong một câu lệnh SQL ta cần chỉ đến một bảng do một người dùng khác sở hữu (hiển nhiên là phải được phép) thì tên của bảng phải được viết sau tên của người sở hữu và phân cách với tên người sở hữu bởi dấu chấm:

```
tên_người_sở_hữu.tên_bảng
```

Một số đối tượng cơ sở dữ liệu khác (như khung nhìn, thủ tục, hàm), việc sử dụng tên cũng tương tự như đối với bảng.

Ta có thể sử dụng tên cột một cách bình thường trong các câu lệnh SQL bằng cách chỉ cần chỉ định tên của cột trong bảng. Tuy nhiên, nếu trong câu lệnh có liên quan đến hai cột trở lên có cùng tên trong các bảng khác nhau thì bắt buộc phải chỉ định thêm tên bảng trước tên cột; tên bảng và tên cột được phân cách nhau bởi dấu chấm.

Ví dụ: Ví dụ dưới đây minh hoạ cho ta thấy việc sử dụng tên bảng và tên cột trong câu lệnh SQL

```
SELECT  masv,hodem,ten,sinhvien.malop,tenlop
FROM dbo.sinhvien,dbo.lop
WHERE sinhvien.malop = lop.malop
```

1.4.3 Kiểu dữ liệu

Chuẩn ANSI/ISO SQL cung cấp các kiểu dữ liệu khác nhau để sử dụng trong các cơ sở dữ liệu dựa trên SQL và trong ngôn ngữ SQL. Dựa trên cơ sở các kiểu dữ liệu do chuẩn ANSI/ISO SQL cung cấp, các hệ quản trị cơ sở dữ liệu thương mại hiện nay có thể sử dụng các dạng dữ liệu khác nhau trong sản phẩm của mình. Bảng 1.2 dưới đây liệt kê một số kiểu dữ liệu thông dụng được sử dụng trong SQL.

Tên kiểu	**Mô tả**
CHAR (*n*)	Kiểu chuỗi với độ dài cố định

NCHAR (*n*)	Kiểu chuỗi với độ dài cố định hỗ trợ UNICODE
VARCHAR (*n*)	Kiểu chuỗi với độ dài chính xác
NVARCHAR (*n*)	Kiểu chuỗi với độ dài chính xác hỗ trợ UNICODE
INTEGER	Số nguyên có giá trị từ -2^{31} đến $2^{31} - 1$
INT	Như kiểu Integer
TINYTINT	Số nguyên có giá trị từ 0 đến 255.
SMALLINT	Số nguyên có giá trị từ -2^{15} đến $2^{15} – 1$
BIGINT	Số nguyên có giá trị từ -2^{63} đến $2^{63}-1$
NUMERIC (*p,s*)	Kiểu số với độ chính xác cố định.
DECIMAL (*p,s*)	Tương tự kiểu Numeric
FLOAT	Số thực có giá trị từ -1.79E+308 đến 1.79E+308
REAL	Số thực có giá trị từ -3.40E + 38 đến 3.40E + 38
MONEY	Kiểu tiền tệ
BIT	Kiểu bit (có giá trị 0 hoặc 1)
DATETIME	Kiểu ngày giờ (chính xác đến phần trăm của giây)
SMALLDATETIME	Kiểu ngày giờ (chính xác đến phút)
TIMESTAMP	
BINARY	Dữ liệu nhị phân với độ dài cố định (tối đa 8000 bytes)
VARBINARY	Dữ liệu nhị phân với độ dài chính xác (tối đa 8000 bytes)
IMAGE	Dữ liệu nhị phân với độ dài chính xác (tối đa 2,147,483,647 bytes)
TEXT	Dữ liệu kiểu chuỗi với độ dài lớn (tối đa 2,147,483,647 ký tự)
NTEXT	Dữ liệu kiểu chuỗi với độ dài lớn và hỗ trợ UNICODE (tối đa 1,073,741,823 ký tự)

Bảng 1.2: *Một số kiểu dữ liệu thông dụng trong SQL*

Ví dụ 1.2: Câu lệnh dưới đây định nghĩa bảng với kiểu dữ liệu được qui định cho các cột trong bảng

```
CREATE TABLE  NHANVIEN
```

```
(
     MANV       NVARCHAR(10)    NOT NULL,
     HOTEN      NVARCHAR(30)    NOT NULL,
     GIOITINH   BIT,
     NGAYSINH   SMALLDATETIME,
     NOISINH    NCHAR(50),
     HSLUONG    DECIMAL(4,2),
     MADV       INT
)
```

1.4.4 Giá trị NULL

Một cơ sở dữ liệu là sự phản ánh của một hệ thống trong thế giới thực, do đó các giá trị dữ liệu tồn tại trong cơ sở dữ liệu có thể không xác định được. Một giá trị không xác định được xuất hiện trong cơ sở dữ liệu có thể do một số nguyên nhân sau:

- Giá trị đó có tồn tại nhưng không biết.
- Không xác định được giá trị đó có tồn tại hay không.
- Tại một thời điểm nào đó giá trị chưa có nhưng rồi có thể sẽ có.
- Giá trị bị lỗi do tính toán (tràn số, chia cho không,...)

Những giá trị không xác định được biểu diễn trong cơ sở dữ liệu quan hệ bởi các giá trị NULL. Đây là giá trị đặc biệt và không nên nhầm lẫn với chuỗi rỗng (đối với dữ liệu kiểu chuỗi) hay giá trị không (đối với giá trị kiểu số). Giá trị NULL đóng một vai trò quan trọng trong các cơ sở dữ liệu và hầu hết các hệ quản trị cơ sở dữ liệu quan hệ hiện nay đều hỗ trợ việc sử dụng giá trị này.

1.5 Kết chương

Như vậy, SQL (viết tắt của *Structured Query Language*) là hệ thống ngôn ngữ được sử dụng cho các hệ quản trị cơ sở dữ liệu quan hệ. Thông qua SQL có thể thực hiện được các thao tác trên cơ sở dữ liệu như định nghĩa dữ liệu, thao tác dữ liệu, điều khiển truy cập, quản lý toàn vẹn dữ liệu... SQL là một thành phần quan trọng và không thể thiếu trong hệ quản trị cơ sở dữ liệu quan hệ.

SQL ra đời nhằm sử dụng cho các cơ sở dữ liệu theo mô hình quan hệ. Trong một cơ sở dữ liệu quan hệ, dữ liệu được tổ chức và lưu trữ trong các bảng. Mỗi một bảng là một tập hợp bao gồm các dòng và các cột; mỗi một dòng là một bản ghi và mỗi một cột tương ứng với một trường, tập các tên cột cùng với kiểu dữ liệu và các tính chất khác tạo nên cấu trúc của bảng, tập các dòng trong bảng chính là dữ liệu của bảng.

Các bảng trong một cơ sở dữ liệu có mối quan hệ với nhau. Các mối quan hệ được biểu diễn thông qua khoá chính và khoá ngoài của các bảng. Khoá chính của bảng là tập một hoặc nhiều cột có giá trị duy nhất trong bảng và do đó giá trị của nó xác định duy nhất một dòng dữ liệu trong bảng. Một khoá ngoài là một tập một hoặc nhiều cột có giá trị được xác định từ khoá chính của các bảng khác.

Chương 2

NGÔN NGỮ THAO TÁC DỮ LIỆU

Đối với đa số người sử dụng, SQL được xem như là công cụ hữu hiệu để thực hiện các yêu cầu truy vấn và thao tác trên dữ liệu. Trong chương này, ta sẽ bàn luận đến nhóm các câu lệnh trong SQL được sử dụng cho mục đích này. Nhóm các câu lệnh này được gọi chung là ngôn ngữ thao tác dữ liệu (DML: Data Manipulation Language) bao gồm các câu lệnh sau:

- SELECT: Sử dụng để truy xuất dữ liệu từ môt hoặc nhiều bảng.
- INSERT: Bổ sung dữ liệu.
- UPDATE: Cập nhật dữ liệu
- DELETE: Xoá dữ liệu

Trong số các câu lệnh này, có thể nói SELECT là câu lệnh tương đối phức tạp và được sử dụng nhiều trong cơ sở dữ liệu. Với câu lệnh này, ta không chỉ thực hiện các yêu cầu truy xuất dữ liệu đơn thuần mà còn có thể thực hiện được các yêu cầu thống kê dữ liệu phức tạp. Cũng chính vì vậy, phần đầu của chương này sẽ tập trung tương đối nhiều đến câu lệnh SELECT. Các câu lệnh INSERT, UPDATE và DELETE được bàn luận đến ở cuối chương

2.1 Truy xuất dữ liệu với câu lệnh SELECT

Câu lệnh SELECT được sử dụng để truy xuất dữ liệu từ các dòng và các cột của một hay nhiều bảng, khung nhìn. 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). Ngoài ra, câu lệnh này còn cung cấp khả năng thực hiện các thao tác truy vấn và thống kê dữ liệu phức tạp khác.

Cú pháp chung của câu lệnh SELECT có dạng:

```
SELECT [ALL | DISTINCT][TOP n] danh_sách_chọn
[INTO tên_bảng_mới]
FROM  danh_sách_bảng/khung_nhìn
[WHERE     điều_kiện]
[GROUP BY  danh_sách_cột]
[HAVING điều_kiện]
```

```
[ORDER BY  cột_sắp_xếp]
[COMPUTE  danh_sách_hàm_gộp [BY danh_sách_cột]]
```

Điều cần lưu ý đầu tiên đối với câu lệnh này là các thành phần trong câu lệnh SELECT nếu được sử dụng phải tuân theo đúng thứ tự như trong cú pháp. Nếu không, câu lệnh sẽ được xem là không hợp lệ.

Câu lệnh SELECT được sử dụng để tác động lên các bảng dữ liệu và kết quả của câu lệnh cũng được hiển thị dưới dạng bảng, tức là một tập hợp các dòng và các cột (ngoại trừ trường hợp sử dụng câu lệnh SELECT với mệnh đề COMPUTE).

Ví dụ 2.1: Kết quả của câu lệnh sau đây cho biết mã lớp, tên lớp và hệ đào tạo của các lớp hiện có

```
SELECT malop,tenlop,hedaotao
FROM lop
```

MALOP	TENLOP	HEDAOTAO
C24101	Toán K24	Chính quy
C24102	Tin K24	Chính quy
C24103	Lý K24	Chính quy
C24301	Sinh K24	Chính quy
C25101	Toán K25	Chính quy
C25102	Tin K25	Chính quy
C25103	Lý K25	Chính quy
C25301	Sinh K25	Chính quy
C26101	Toán K26	Chính quy
C26102	Tin K26	Chính quy

2.1.1 Mệnh đề FROM

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

Ví dụ 2.2: Câu lệnh dưới đây hiển thị danh sách các khoa trong trường

```
SELECT * FROM khoa
```

kết quả câu lệnh như sau:

MAKHOA	TENKHOA	DIENTHOAI
DHT01	Khoa Toán cơ - Tin học	054822407
DHT02	Khoa Công nghệ thông tin	054826767
DHT03	Khoa Vật lý	054823462
DHT04	Khoa Hoá học	054823951
DHT05	Khoa Sinh học	054822934
DHT06	Khoa Địa lý - Địa chất	054823837
DHT07	Khoa Ngữ văn	054821133
DHT08	Khoa Lịch sử	054823833
DHT09	Khoa Mác - Lê Nin	054825698
DHT10	Khoa Luật	054821135

Ta có thể sử dụng các bí danh cho các bảng hay khung nhìn trong câu lệnh SELECT. Bí danh được gán trong mệnh đề FROM bằng cách chỉ định bí danh ngay sau tên bảng.

Ví dụ 2.3: câu lệnh sau gán bí danh là *a* cho bảng *khoa*

```
SELECT * FROM khoa a
```

2.1.2 Danh sách chọn trong câu lệnh SELECT

Danh sách chọn trong câu lệnh SELECT được sử dụng để chỉ định các trường, các biểu thức cần hiển thị trong các cột của kết quả truy vấn. Các trường, các biểu thức được chỉ định ngay sau từ khoá SELECT và phân cách nhau bởi dấu phẩy. Sử dụng danh sách chọn trong câu lệnh SELECT bao gồm các trường hợp sau:

a. Chọn tất cả các cột trong bảng

Khi cần hiển thị tất cả các trường trong các bảng, sử dụng ký tự * trong danh sách chọn thay vì phải liệt kê danh sách tất cả các cột. Trong trường hợp này, các cột được hiển thị trong kết quả truy vấn sẽ tuân theo thứ tự mà chúng đã được tạo ra khi bảng được định nghĩa.

Ví dụ 2.4: Câu lệnh

```
SELECT * FROM lop
```

cho kết quả bao như sau:

MALOP	TENLOP	KHOA	HEDAOTAO	NAMNHAPHOC	MAKHOA
C24101	Toán K24	24	Chính quy	2000	DHTO1
C24102	Tin K24	24	Chính quy	2000	DHTO2
C24103	Lý K24	24	Chính quy	2000	DHTO3
C24301	Sinh K24	24	Chính quy	2000	DHTO5
C25101	Toán K25	25	Chính quy	2001	DHTO1
C25102	Tin K25	25	Chính quy	2001	DHTO2
C25103	Lý K25	25	Chính quy	2001	DHTO3
C25301	Sinh K25	25	Chính quy	2001	DHTO5
C26101	Toán K26	26	Chính quy	2002	DHTO1
C26102	Tin K26	26	Chính quy	2002	DHTO2

b. Tên cột trong danh sách chọn

Trong trường hợp cần chỉ định cụ thể các cột cần hiển thị trong kết quả truy vấn, ta chỉ định danh sách các tên cột trong danh sách chọn. Thứ tự của các cột trong kết quả truy vấn tuân theo thứ tự của các trường trong danh sách chọn.

Ví dụ 2.5: Câu lệnh

```
SELECT malop,tenlop,namnhaphoc,khoa
FROM lop
```

cho biết mã lớp, tên lớp, năm nhập học và khoá của các lớp và có kết quả như sau:

MALOP	TENLOP	NAMNHAPHOC	KHOA
C24101	Toán K24	2000	24
C24102	Tin K24	2000	24
C24103	Lý K24	2000	24
C24301	Sinh K24	2000	24
C25101	Toán K25	2001	25
C25102	Tin K25	2001	25
C25103	Lý K25	2001	25
C25301	Sinh K25	2001	25
C26101	Toán K26	2002	26
C26102	Tin K26	2002	26

Lưu ý: Nếu truy vấn được thực hiện trên nhiều bảng/khung nhìn và trong các bảng/khung nhìn có các trường trùng tên thì tên của những trường này nếu xuất hiện trong danh sách chọn phải được viết dưới dạng:

```
tên_bảng.tên_trường
```

Ví dụ 2.6:

```
SELECT malop, tenlop, lop.makhoa, tenkhoa
FROM lop, khoa
WHERE lop.malop = khoa.makhoa
```

c. Thay đổi tiêu đề các cột

Trong kết quả truy vấn, tiêu đề của các cột mặc định sẽ là tên của các trường tương ứng trong bảng. Tuy nhiên, để các tiêu đề trở nên thân thiện hơn, ta có thể đổi tên các tiêu đề của các cột. Để đặt tiêu đề cho một cột nào đó, ta sử dụng cách viết:

```
tiêu_đề_cột = tên_trường
```
hoặc
```
tên_trường AS tiêu_đề_cột
```
hoặc
```
tên_trường  tiêu_đề_cột
```

Ví dụ 2.7: Câu lệnh dưới đây:

```
SELECT 'Mã lớp'= malop,tenlop 'Tên lớp',khoa AS 'Khoá'
FROM lop
```

cho biết mã lớp, tên lớp và khoá học của các lớp trong trường. Kết quả của câu lệnh như sau:

Mã lớp	Tên Lớp	Khoá
C24101	Toán K24	24
C24102	Tin K24	24
C24103	Lý K24	24
C24301	Sinh K24	24
C25101	Toán K25	25
C25102	Tin K25	25
C25103	Lý K25	25
C25301	Sinh K25	25
C26101	Toán K26	26
C26102	Tin K26	26

d. Sử dụng cấu trúc CASE trong danh sách chọn

Cấu trúc CASE được sử dụng trong danh sách chọn nhằm thay đổi kết quả của truy vấn tuỳ thuộc vào các trường hợp khác nhau. Cấu trúc này có cú pháp như sau:

```
CASE biểu_thức
   WHEN biểu_thức_kiểm_tra THEN kết_quả
   [ ... ]
   [ELSE kết_quả_của_else]
END
```

hoặc:

```
CASE
   WHEN điều_kiện THEN kết_quả
   [ ... ]
   [ELSE kết_quả_của_else]
END
```

Ví dụ 2.8: Để hiển thị mã, họ tên và giới tính (nam hoặc nữ) của các sinh viên, ta sử dụng câu lệnh

```
SELECT masv,hodem,ten,
       CASE gioitinh
          WHEN 1 THEN 'Nam'
          ELSE 'Nữ'
       END AS gioitinh
FROM sinhvien
```

hoặc:

```
SELECT masv,hodem,ten,
       CASE
          WHEN gioitinh=1 THEN 'Nam'
          ELSE 'Nữ'
       END AS gioitinh
FROM sinhvien
```

Kết quả của hai câu lệnh trên đều có dạng như sau

MASV	HODEM	TEN	GIOITINH
0241010001	Ngô Thị Nhật	Anh	Nữ
0241010002	Nguyễn Thị Ngọc	Anh	Nữ
0241010003	Ngô Việt	Bắc	Nam
0241010004	Nguyễn Đình	Bình	Nam
0241010005	Hồ Đăng	Chiến	Nam
0241020001	Nguyễn Tuấn	Anh	Nam
0241020002	Trần Thị Kim	Anh	Nữ
0241020003	Võ Đức	Ân	Nam
0241020004	Nguyễn Công	Bình	Nam
0241020005	Nguyễn Thanh	Bình	Nam
0241020006	Lê Thị Thanh	Châu	Nữ
0241020007	Bùi Đình	Chiến	Nam
0241020008	Nguyễn Công	Chính	Nam
...	...	...	...

e. Hằng và biểu thức trong danh sách chọn

Ngoài danh sách trường, trong danh sách chọn của câu lệnh SELECT còn có thể sử dụng các biểu thức. Mỗi một biểu thức trong danh sách chọn trở thành một cột trong kết quả truy vấn.

Ví dụ 2.9: câu lệnh dưới đây cho biết tên và số tiết của các môn học

```
SELECT tenmonhoc,sodvht*15 AS sotiet
FROM monhoc
```

TENMONHOC	SOTIET
Hoá đại cương	45
Tin học đại cương	60
Ngôn ngữ C	75
Lý thuyết hệ điều hành	60
Cấu trúc dữ liệu và ...	60
Đại số tuyến tính	60
Giải tích 1	60
Bài tập Đại số	30
Bài tập Giải tích 1	30
Vật lý đại cương	45

Nếu trong danh sách chọn có sự xuất hiện của giá trị hằng thì giá trị này sẽ xuất hiện trong một cột của kết quả truy vấn ở tất cả các dòng

Ví dụ 2.10: Câu lệnh

```
SELECT tenmonhoc,'Số tiết: ',sodvht*15 AS sotiet
FROM monhoc
```

cho kết quả như sau:

TENMONHOC	(No column name)	SOTIET
Hoá đại cương	Số tiết:	45
Tin học đại cương	Số tiết:	60
Ngôn ngữ C	Số tiết:	75
Lý thuyết hệ điều hành	Số tiết:	60
Cấu trúc dữ liệu và giải thuật	Số tiết:	60
Đại số tuyến tính	Số tiết:	60
Giải tích 1	Số tiết:	60
Bài tập Đại số	Số tiết:	30
Bài tập Giải tích 1	Số tiết:	30
Vật lý đại cương	Số tiết:	45

f. Loại bỏ các dòng dữ liệu trùng nhau trong kết quả truy vấn

Trong kết quả của truy vấn có thể xuất hiện các dòng dữ liệu trùng nhau. Để loại bỏ bớt các dòng này, ta chỉ định thêm từ khóa DISTINCT ngay sau từ khoá SELECT.

Ví dụ 2.11: Hai câu lệnh dưới đây

```
SELECT khoa FROM lop
```

và:

```
SELECT DISTINCT khoa FROM lop
```

có kết quả lần lượt như sau:

KHOA
24
24
24
24
25
25
25
25
26
26

KHOA
24
25
26

g. Giới hạn số lượng dòng trong kết quả truy vấn

Kết quả của truy vấn được hiển thị thường sẽ là tất cả các dòng dữ liệu truy vấn được. Trong trường hợp cần hạn chế số lượng các dòng xuất hiện trong kết quả truy vấn, ta chỉ định thêm mệnh đề TOP ngay trước danh sách chọn của câu lệnh SELECT.

Ví dụ 2.12: Câu lệnh dưới đây hiển thị họ tên và ngày sinh của 5 sinh viên đầu tiên trong danh sách

```
SELECT TOP 5 hodem,ten,ngaysinh
FROM sinhvien
```

Ngoài cách chỉ định cụ số lượng dòng cần hiển thị trong kết quả truy vấn, ta có thể chỉ định số lượng các dòng cần hiển thị theo tỷ lệ phần trăm bằng cách sử dụng thêm từ khoá PERCENT như ở ví dụ dưới đây.

Ví dụ 2.13: Câu lệnh dưới đây hiển thị họ tên và ngày sinh của 10% số lượng sinh viên hiện có trong bảng SINHVIEN

```
SELECT TOP 10 PERCENT hodem,ten,ngaysinh
FROM sinhvien
```

2.1.3 Chỉ định điều kiện truy vấn dữ liệu

Mệnh đề WHERE trong câu lệnh SELECT được sử dụng nhằm 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 điều kiện được chỉ định mới được hiển thị trong kết quả truy vấn.

Ví dụ 2.14: Câu lệnh dưới đây hiển thị danh sách các môn học có số đơn vị học trình lớn hơn 3

```
SELECT * FROM  monhoc
WHERE sodvht>3
```

Kết quả của câu lệnh này như sau:

MAMONHOC	TENMONHOC	SODVHT
TI-001	Tin học đại cương	4
TI-002	Ngôn ngữ C	5
TI-003	Lý thuyết hệ điều hành	4
TI-004	Cấu trúc dữ liệu và giải thuật	4
TO-001	Đại số tuyến tính	4
TO-002	Giải tích 1	4

Trong mệnh đề WHERE thường sử dụng:

- Các toán tử kết hợp điều kiện (AND, OR)
- Các toán tử so sánh
- Kiểm tra giới hạn của dữ liệu (BETWEEN/ NOT BETWEEN)
- Danh sách
- Kiểm tra khuôn dạng dữ liệu.
- Các giá trị NULL

a. 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 2.15: Câu lệnh:

```
SELECT masv,hodem,ten,ngaysinh
FROM sinhvien
WHERE (ten='Anh')
      AND (YEAR(GETDATE())-YEAR(ngaysinh)<=20)
```

cho biết mã, họ tên và ngày sinh của các sinh viên có tên là *Anh* và có tuổi nhỏ hơn hoặc bằng 20.

MASV	HODEM	TEN	NGAYSINH
0261010001	Lê Hoàng Phương	Anh	1984-03-04 00:00:00
0261010002	Lê Thị Vân	Anh	1984-10-14 00:00:00
0261020002	Lê Thúc Quốc	Anh	1984-12-04 00:00:00
0261020004	Nguyễn Thị Lan	Anh	1984-08-23 00:00:00
0261020005	Nguyễn Thị Lan	Anh	1984-07-25 00:00:00

b. Kiểm tra giới hạn của dữ liệu

Để kiểm tra xem giá trị dữ liệu nằm trong (ngoài) một khoảng nào đó, ta sử dụng toán tử BETWEEN (NOT BETWEEN) như sau:

Cách sử dụng	Ý nghĩa
giá_trị BETWEEN a AND b	a ≤ giá_trị ≤ b
giá_trị NOT BETWEEN a AND b	(giá_trị < a) AND (giá_trị>b)

Ví dụ 2.16: Câu lệnh dưới đây cho biết họ tên và tuổi của các sinh viên có tên là *Bình* và có tuổi nằm trong khoảng từ 20 đến 22

```
SELECT hodem,ten,year(getdate())-year(ngaysinh) AS tuoi
FROM sinhvien
WHERE ten='Bình' AND
      YEAR(GETDATE())-YEAR(ngaysinh) BETWEEN 20 AND 22
```

c. 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 2.17: Để biết danh sách các môn học có số đơn vị học trình là 2, 4 hoặc 5, thay vì sử dụng câu lệnh

```
SELECT * FROM monhoc
WHERE sodvht=2 OR sodvht=4 OR sodvht=5
```

ta có thể sử dụng câu lệnh

```
SELECT * FROM monhoc
WHERE sodvht IN (2,4,5)
```

d. Toán tử LIKE và các ký tự đại diện

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 2.18: Câu lệnh dưới đây

```
SELECT hodem,ten FROM sinhvien
WHERE hodem LIKE 'Lê%'
```

cho biết họ tên của các sinh viên có họ là *Lê* và có kết quả như sau

HODEM	TEN
Lê Thị Thanh	Châu
Lê Thị	Anh
Lê Văn Khoa	Bảo
Lê Thị	Dí
Lê Tất Uyên	Châu
Lê Hoàng Phương	Anh
Lê Thị Vân	Anh
Lê Đăng	Ánh
Lê Huy	Đan
Lê Thúc Quốc	Anh

Câu lệnh:

```
SELECT hodem,ten FROM sinhvien
WHERE hodem LIKE 'Lê%' AND ten LIKE '[AB]%'
```

Có kết quả là:

HODEM	TEN
Lê Thị	Anh
Lê Văn Khoa	Bảo
Lê Hoàng Phương	Anh
Lê Thị Vân	Anh
Lê Thúc Quốc	Anh

e. Giá trị NULL

Dữ liệu trong một cột cho phép NULL sẽ nhận giá trị NULL trong các trường hợp sau:

- Nếu không có dữ liệu được nhập cho cột 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, để kiểm tra giá trị của một cột có giá trị NULL hay không, ta sử dụng cách viết:

```
WHERE  tên_cột  IS NULL
```

hoặc:

```
WHERE tên_cột IS NOT NULL
```

2.1.4 Tạo mới bảng dữ liệu từ kết quả của câu lệnh SELECT

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 là số dòng kết quả của truy vấn

Ví dụ 2.19: Câu lệnh dưới đây truy vấn dữ liệu từ bảng SINHVIEN và tạo một bảng TUOISV bao gồm các trường HODEM, TEN và TUOI

```
SELECT hodem,ten,YEAR(GETDATE())-YEAR(ngaysinh) AS tuoi
INTO tuoisv
FROM sinhvien
```

Lưu ý: Nếu trong danh sách chọn có các biểu thức thì những biểu thức này phải được đặt tiêu đề.

2.1.5 Sắp xếp kết quả truy vấn

Mặc định, các dòng dữ liệu trong kết quả của câu truy vấn tuân theo thứ tự của chúng trong bảng dữ liệu hoặc được sắp xếp theo chỉ mục (nếu trên bảng có chỉ mục). Trong trường hợp muốn dữ liệu được sắp xếp theo chiều tăng hoặc giảm của giá trị của một hoặc nhiều trường, ta sử dụng thêm mệnh đề ORDER BY trong câu lệnh SELECT; Sau ORDER BY là danh sách các cột cần sắp xếp (tối đa là 16 cột). Dữ liệu được sắp xếp có thể theo chiều tăng (ASC) hoặc giảm (DESC), mặc định là sắp xếp theo chiều tăng.

Ví dụ 2.20: Câu lệnh dưới đây hiển thị danh sách các môn học và sắp xếp theo chiều giảm dần của số đơn vị học trình

```
SELECT * FROM monhoc
ORDER BY sodvht DESC
```

MAMONHOC	TENMONHOC	SODVHT
TI-002	Ngôn ngữ C	5
TI-003	Lý thuyết hệ điều hành	4
TI-001	Tin học đại cương	4
TI-004	Cấu trúc dữ liệu và giải thuật	4
TO-001	Đại số tuyến tính	4
TO-002	Giải tích 1	4
HO-001	Hoá đại cương	3
VL-001	Vật lý đại cương	3
TO-004	Bài tập Giải tích 1	2
TO-003	Bài tập Đại số	2

Nếu sau ORDER BY có nhiều cột thì việc sắp xếp dữ liệu sẽ được ưu tiên theo thứ tự từ trái qua phải.

Ví dụ 2.21: Câu lệnh

```
SELECT hodem,ten,gioitinh,
       YEAR(GETDATE())-YEAR(ngaysinh) AS tuoi
FROM sinhvien
WHERE ten='Bình'
ORDER BY gioitinh,tuoi
```

có kết quả là:

HODEM	TEN	GIOITINH	TUOI
Nguyễn Thị	Bình	0	23
Hoàng Văn	Bình	1	21
Châu Văn Quốc	Bình	1	21
Nguyễn Thanh	Bình	1	22
Nguyễn Đình	Bình	1	22
Nguyễn Công	Bình	1	25

Thay vì chỉ định tên cột sau ORDER BY, ta có thể chỉ định số thứ tự của cột cần được sắp xếp. Câu lệnh ở ví dụ trên có thể được viết lại như sau:

```
SELECT hodem,ten,gioitinh,
       YEAR(GETDATE())-YEAR(ngaysinh) AS tuoi
FROM sinhvien
WHERE ten='Bình'
ORDER BY 3, 4
```

2.1.6 Phép hợp

Phép hợp được sử dụng trong trường hợp ta cần gộp kết quả của hai hay nhiều truy vấn thành một tập kết quả duy nhất. SQL cung cấp toán tử UNION để thực hiện phép hợp. Cú pháp như sau

```
Câu_lệnh_1
UNION [ALL] Câu_lệnh_2
[UNION [ALL] Câu_lệnh_3]
...
[UNION [ALL] Câu_lệnh_n]
[ORDER BY  cột_sắp_xếp]
[COMPUTE  danh_sách_hàm_gộp [BY danh_sách_cột]]
```

Trong đó

Câu_lệnh_1 có dạng

```
SELECT  danh_sách_cột
[INTO tên_bảng_mới]
[FROM danh_sách_bảng|khung_nhìn]
[WHERE điều_kiện]
[GROUP BY danh_sách_cột]
[HAVING điều_kiện]
```

và *Câu_lệnh_i* (i = 2,..,n) có dạng

```
SELECT  danh_sách_cột
[FROM danh_sách_bảng|khung_nhìn]
[WHERE điều_kiện]
[GROUP BY danh_sách_cột]
[HAVING điều_kiện]
```

Ví dụ 2.22: Giả sử ta có hai bảng Table1 và Table2 lần lượt như sau:

A	B	C
a	1	10
b	2	20
c	3	30
d	4	40
a	5	50
b	6	60

D	E
a	1
b	2
d	3
e	4

câu lệnh

```
SELECT A,B FROM Table1
UNION
SELECT D,E FROM table2
```

Cho kết quả như sau:

A	B
a	1
a	5
b	2
b	6
c	3
d	3
d	4
e	4

Mặc định, nếu trong các truy vấn thành phần của phép hợp xuất hiện những dòng dữ liệu giống nhau thì trong kết quả truy vấn chỉ giữ lại một dòng. Nếu muốn giữ lại các dòng này, ta phải sử dụng thêm từ khoá ALL trong truy vấn thành phần.

Ví dụ 2.23: Câu lệnh

```
SELECT A,B FROM Table1
UNION ALL
SELECT D,E FROM table2
```

Cho kết quả như sau

A	B
a	1
b	2
c	3
d	4
a	5
b	6
a	1
b	2
d	3
e	4

Khi sử dụng toán tử UNION để thực hiện phép hợp, ta cần chú ý các nguyên tắc sau:

- Danh sách cột trong các truy vấn thành phần phải có cùng số lượng.
- 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 thành phần phải cùng kiểu dữ liệu.
- Các cột tương ứng trong bản thân từng truy vấn thành phần 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.

- Truy vấn thành phần đầu tiên có thể có INTO để tạo mới một bảng từ kết quả của chính phép hợp.
- Mệnh đề ORDER BY và COMPUTE dùng để sắp xếp kết quả truy vấn hoặc tính toán các giá trị thống kê chỉ được sử dụng ở cuối câu lệnh UNION. Chúng không được sử dụng ở trong bất kỳ truy vấn thành phần nào.
- Mệnh đề GROUP BY và HAVING chỉ có thể được sử dụng trong bản thân từng truy vấn thành phần. Chúng không được phép sử dụng để tác động lên kết quả chung của phép hợp.
- Phép toán UNION có thể được sử dụng bên trong câu lệnh INSERT.
- Phép toán UNION không được sử dụng trong câu lệnh CREATE VIEW.

2.1.7 Phép nối

Khi cần thực hiện một yêu cầu truy vấn dữ liệu từ hai hay nhiều bảng, ta phải sử dụng đến phép nối. Một câu lệnh nối kết hợp các dòng dữ liệu trong các bảng khác nhau lại theo một hoặc nhiều điều kiện nào đó và hiển thị chúng trong kết quả truy vấn.

Xét hai bảng sau đây:

Bảng KHOA

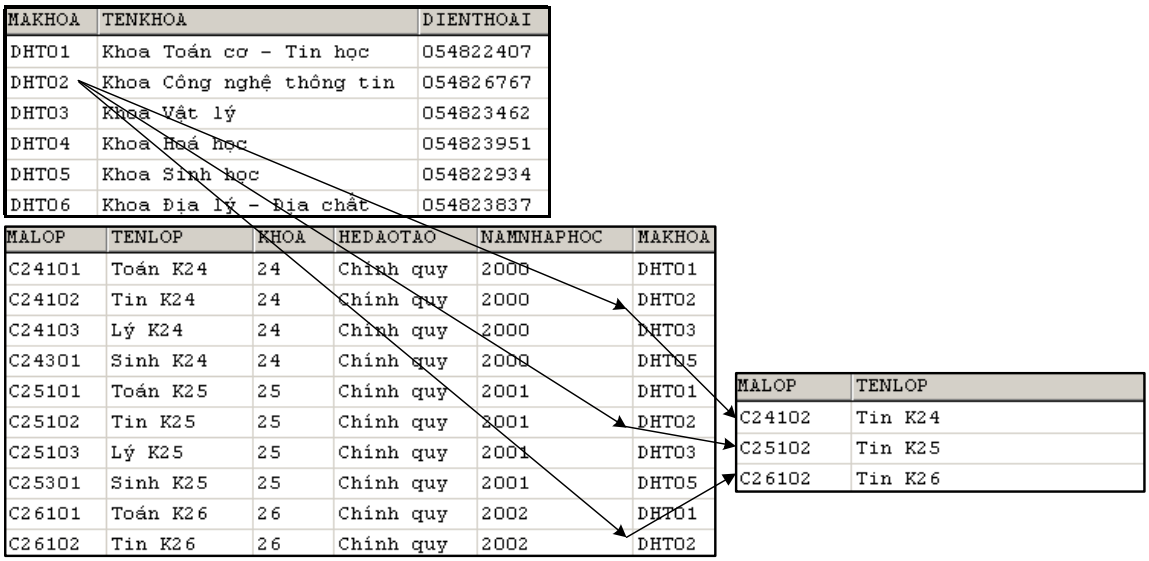
MAKHOA	TENKHOA	DIENTHOAI
DHT01	Khoa Toán cơ - Tin học	054822407
DHT02	Khoa Công nghệ thông tin	054826767
DHT03	Khoa Vật lý	054823462
DHT04	Khoa Hoá học	054823951
DHT05	Khoa Sinh học	054822934
DHT06	Khoa Địa lý - Địa chất	054823837

Bảng LOP

MALOP	TENLOP	KHOA	HEDAOTAO	NAMNHAPHOC	MAKHOA
C24101	Toán K24	24	Chính quy	2000	DHT01
C24102	Tin K24	24	Chính quy	2000	DHT02
C24103	Lý K24	24	Chính quy	2000	DHT03
C24301	Sinh K24	24	Chính quy	2000	DHT05
C25101	Toán K25	25	Chính quy	2001	DHT01
C25102	Tin K25	25	Chính quy	2001	DHT02
C25103	Lý K25	25	Chính quy	2001	DHT03
C25301	Sinh K25	25	Chính quy	2001	DHT05
C26101	Toán K26	26	Chính quy	2002	DHT01
C26102	Tin K26	26	Chính quy	2002	DHT02

Giả sử ta cần biết mã lớp và tên lớp của các lớp thuộc *Khoa Công nghệ Thông tin*, ta phải làm như sau:

- Chọn ra dòng trong bảng KHOA có tên khoa là *Khoa Công nghệ Thông tin*, từ đó xác định được mã khoa (MAKHOA) là *DHT02.*
- Tìm kiếm trong bảng LOP những dòng có giá trị trường MAKHOA là *DHT02* (tức là bằng MAKHOA tương ứng trong bảng KHOA) và đưa những dòng này vào kết quả truy vấn



MAKHOA	TENKHOA	DIENTHOAI
DHT01	Khoa Toán cơ - Tin học	054822407
DHT02	Khoa Công nghệ thông tin	054826767
DHT03	Khoa Vật lý	054823462
DHT04	Khoa Hoá học	054823951
DHT05	Khoa Sinh học	054822934
DHT06	Khoa Địa lý - Địa chất	054823837

MALOP	TENLOP	KHOA	HEDAOTAO	NAMNHAPHOC	MAKHOA
C24101	Toán K24	24	Chính quy	2000	DHT01
C24102	Tin K24	24	Chính quy	2000	DHT02
C24103	Lý K24	24	Chính quy	2000	DHT03
C24301	Sinh K24	24	Chính quy	2000	DHT05
C25101	Toán K25	25	Chính quy	2001	DHT01
C25102	Tin K25	25	Chính quy	2001	DHT02
C25103	Lý K25	25	Chính quy	2001	DHT03
C25301	Sinh K25	25	Chính quy	2001	DHT05
C26101	Toán K26	26	Chính quy	2002	DHT01
C26102	Tin K26	26	Chính quy	2002	DHT02

MALOP	TENLOP
C24102	Tin K24
C25102	Tin K25
C26102	Tin K26

Như vậy, để thực hiện được yêu cầu truy vấn dữ liệu trên, ta phải thực hiện phép nối giữa hai bảng KHOA và LOP với điều kiện nối là MAKHOA của KHOA bằng với MAKHOA của LOP. Câu lệnh sẽ được viết như sau:

```
SELECT malop,tenlop
FROM khoa,lop
WHERE khoa.makhoa = lop.makhoa AND
      tenkhoa='Khoa Công nghệ Thông tin'
```

2.1.7.1 Sử dụng phép nối

Phép nối là cơ sở để thực hiện các yêu cầu truy vấn dữ liệu liên quan đến nhiều bảng. Một câu lệnh nối thực hiện lấy các dòng dữ liệu trong các bảng tham gia truy vấn, so sánh giá trị của các dòng này trên một hoặc nhiều cột được chỉ định trong điều kiện nối và kết hợp các dòng thoả mãn điều kiện thành những dòng trong kết quả truy vấn.

Để thực hiện được một phép nối, cần phải xác định được những yếu tố sau:

- Những cột nào cần hiển thị trong kết quả truy vấn
- Những bảng nào có tham gia vào truy vấn.
- Điều kiện để thực hiện phép nối giữa các bảng dữ liệu là gì

Trong các yếu tố kể trên, việc xác định chính xác điều kiện để thực hiện phép nối giữa các bảng đóng vai trò quan trọng nhất. Trong đa số các trường hợp, điều kiện của phép nối được xác định nhờ vào mối quan hệ giữa các bảng cần phải truy xuất dữ liệu. Thông thường, đó là điều kiện bằng nhau giữa khoá chính và khoá ngoài của hai bảng có mối quan hệ với nhau. Như vậy, để có thể đưa ra một câu lệnh nối thực hiện chính xác yêu cầu truy vấn dữ liệu đòi hỏi phải hiểu được mối quan hệ cũng như ý nghĩa của chúng giữa các bảng dữ liệu.

Danh sách chọn trong phép nối

Một câu lệnh nối cũng được bắt đầu với từ khóa SELECT. Các cột được chỉ định tên sau từ khoá SELECT là các cột được hiển thị trong kết quả truy vấn. Việc sử dụng tên các cột trong danh sách chọn có thể là:

- Tên của một số cột nào đó trong các bảng có tham gia vào truy vấn. Nếu tên cột trong các bảng trùng tên nhau thì tên cột phải được viết dưới dạng

```
tên_bảng.tên_cột
```

- Dấu sao (*) được sử dụng trong danh sách chọn khi cần hiển thị tất cả các cột của các bảng tham gia truy vấn.
- Trong trường hợp cần hiển thị tất cả các cột của một bảng nào đó, ta sử dụng cách viết:

```
tên_bảng.*
```

Mệnh đề FROM trong phép nối

Sau mệnh đề FROM của câu lệnh nối là danh sách tên các bảng (hay khung nhìn) tham gia vào truy vấn. Nếu ta sử dụng dấu * trong danh sách chọn thì thứ tự của các bảng liệt kê sau FROM sẽ ảnh hưởng đến thứ tự các cột được hiển thị trong kết quả truy vấn.

Mệnh đề WHERE trong phép nối

Khi hai hay nhiều bảng được nối với nhau, ta phải chỉ định điều kiện để thực hiện phép nối ngay sau mệnh đề WHERE. Điều kiện nối được biểu diễn dưới dạng biểu thức logic so sánh giá trị dữ liệu giữa các cột của các bảng tham gia truy vấn.

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

Ví dụ 2.24: Câu lệnh dưới đây hiển thị danh sách các sinh viên với các thông tin: mã sinh viên, họ và tên, mã lớp, tên lớp và tên khoa

```
SELECT masv,hodem,ten,sinhvien.malop,tenlop,tenkhoa
FROM sinhvien,lop,khoa
WHERE sinhvien.malop = lop.malop AND
      lop.makhoa=khoa.makhoa
```

Trong câu lệnh trên, các bảng tham gia vào truy vấn bao gồm SINHVIEN, LOP và KHOA. Điều kiện để thực hiện phép nối giữa các bảng bao gồm hai điều kiện:

```
sinhvien.malop = lop.malop
```

và

```
lop.malop = khoa.malop
```

Điều kiện nối giữa các bảng trong câu lệnh trên là điều kiện bằng giữa khoá ngoài và khoá chính của các bảng có mối quan hệ với nhau. Hay nói cách khác, điều kiện của phép nối được xác định dựa vào mối quan hệ giữa các bảng trong cơ sở dữ liệu.

2.1.7.2 Các loại phép nối

Phép nối bằng và phép nối tự nhiên

Một *phép nối bằng* (equi-join) 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 2.25: Câu lệnh dưới đây thực hiện phép nối bằng giữa hai bảng LOP và KHOA

```
SELECT *
FROM lop,khoa
WHERE lop.makhoa=khoa.makhoa
```

Trong kết quả của câu lệnh trên, cột *makhoa* (mã khoa) xuất hiện hai lần trong kết quả phép nối (cột *makhoa* của bảng *khoa* và cột *makhoa* của bảng *lop*) và như vậy là không cần thiết. Ta có thể loại bỏ bớt đi những cột trùng tên trong kết quả truy vấn bằng cách chỉ định danh sách cột cần được hiển thị trong danh sách chọn của câu lệnh.

Một dạng đặc biệt của phép nối bằng được sử dụng nhiều là *phép nối tự nhiên* (natural-join). Trong phép nối tự nhiên, điều kiện nối giữa hai bảng chính là điều kiện bằng giữa khoá ngoài và khoá chính của hai bảng; Và trong danh sách chọn của câu lệnh chỉ giữ lại một cột trong hai cột tham gia vào điều kiện của phép nối

Ví dụ 2.26: Để thực hiện phép nối tự nhiên, câu lệnh trong ví dụ 2.25 được viết lại như sau

```
SELECT malop,tenlop,khoa,hedaotao,namnhaphoc,
       siso,lop.makhoa,tenkhoa,dienthoai
FROM lop,khoa
WHERE lop.makhoa=khoa.makhoa
```

hoặc viết dưới dạng ngắn gọn hơn:

```
SELECT lop.*,tenkhoa,dienthoai
FROM lop,khoa
WHERE lop.makhoa=khoa.makhoa
```

Phép nối với các điều kiện bổ sung

Trong các câu lệnh nối, ngoài điều kiện của phép nối được chỉ định trong mệnh đề WHERE còn có thể chỉ định các điều kiện tìm kiếm dữ liệu khác (điều kiện chọn). Thông thường, các điều kiện này được kết hợp với điều kiện nối thông qua toán tử AND.

Ví dụ 2.27: Câu lệnh dưới đây hiển thị họ tên và ngày sinh của các sinh viên *Khoa Công nghệ Thông tin*

```
SELECT hodem,ten,ngaysinh
FROM sinhvien,lop,khoa
WHERE tenkhoa='Khoa Công nghệ Thông tin' AND
       sinhvien.malop = lop.malop AND
       lop.makhoa = khoa.makhoa
```

Phép tự nối và các bí danh

Phép tự nối là phép nối mà trong đó điều kiện nối được chỉ định liên quan đến các cột của cùng một bảng. Trong trường hợp này, sẽ có sự xuất hiện tên của cùng một bảng nhiều lần trong mệnh đề FROM và do đó các bảng cần phải được đặt bí danh.

Ví dụ 2.28: Để biết được họ tên và ngày sinh của các sinh viên có cùng ngày sinh với sinh viên *Trần Thị Kim Anh*, ta phải thực hiện phép tự nối ngay trên chính bảng *sinhvien*. Trong câu lệnh nối, bảng *sinhvien* xuất hiện trong mệnh đề FROM với bí danh là *a* và *b*. Bảng *sinhvien* với bí danh là *a* sử dụng để chọn ra sinh viên có họ tên là *Trần Thị Kim Anh* và bảng *sinhvien* với bí danh là *b* sử dụng để xác định các sinh viên trùng ngày sinh với sinh viên *Trần Thị Kim Anh*. Câu lệnh được viết như sau:

```
SELECT b.hodem,b.ten,b.ngaysinh
FROM sinhvien a, sinhvien b
WHERE a.hodem='Trần Thị Kim' AND a.ten='Anh' AND
      a.ngaysinh=b.ngaysinh AND a.masv<>b.masv
```

Phép nối không dựa trên tiêu chuẩn bằng

Trong phép nối này, điều kiện để thực hiện phép nối giữa các bảng dữ liệu không phải là điều kiện so sành bằng giữa các cột. Loại phép nối này trong thực tế thường ít được sử dụng.

Phép nối ngoài (outer-join)

Trong các phép nối đã đề cập ở trên, chỉ những dòng có giá trị trong các cột được chỉ định thoả mãn điều kiện kết nối mới được hiển thị trong kết quả truy vấn, và được gọi là phép nối trong (inner join) Theo một nghĩa nào đó, những phép nối này loại bỏ thông tin chứa trong những dòng không thoả mãn điều kiện nối. Tuy nhiên, đôi khi ta cũng cần giữ lại những thông tin này bằng cách cho phép những dòng không thoả mãn điều kiện nối có mặt trong kết quả của phép nối. Để làm điều này, ta có thể sử dụng *phép nối ngoài*.

SQL cung cấp các loại phép nối ngoài sau đây:

- **Phép nối ngoài trái** (ký hiệu: *=): Phép nối này hiển thị trong kết quả truy vấn tất cả các dòng dữ liệu của bảng nằm bên trái trong điều kiện nối cho dù những dòng này không thoả mãn điều kiện của phép nối
- **Phép nối ngoài phải** (ký hiệu: =*): Phép nối này hiển thị trong kết quả truy vấn tất cả các dòng dữ liệu của bảng nằm bên phải trong điều kiện nối cho dù những dòng này không thoả điều kiện của phép nối.

Ví dụ 2.29: Giả sử ta có hai bảng DONVI và NHANVIEN như sau:

Bảng DONVI

MADV	TENDV
1	Doi ngoai
2	Hanh chinh
3	Ke toan
4	Kinh doanh

Bảng NHANVIEN

HOTEN	MADV
Thanh	1
Hoa	2
Nam	2
Vinh	1
Hung	5
Phuong	NULL

Câu lệnh:

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

có kết quả là:

HOTEN	MADV	MADV	TENDV
Thanh	1	1	Doi ngoai
Hoa	2	2	Hanh chinh
Nam	2	2	Hanh chinh
Vinh	1	1	Doi ngoai

Nếu thực hiện phép nối ngoài trái giữa bảng NHANVIEN và bảng DONVI:

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

kết quả của câu lệnh sẽ là:

HOTEN	MADV	MADV	TENDV
Thanh	1	1	Doi ngoai
Hoa	2	2	Hanh chinh
Nam	2	2	Hanh chinh
Vinh	1	1	Doi ngoai
Hung	5	NULL	NULL
Phuong	NULL	NULL	NULL

Và kết quả của phép nối ngoài phải:

```
select *
from nhanvien,donvi
where nhanvien.madv=*donvi.madv
```

như sau:

HOTEN	MADV	MADV	TENDV
Thanh	1	1	Doi ngoai
Vinh	1	1	Doi ngoai
Hoa	2	2	Hanh chinh
Nam	2	2	Hanh chinh
NULL	NULL	3	Ke toan
NULL	NULL	4	Kinh doanh

Phép nối và các giá trị NULL

Nếu trong các cột của các bảng tham gia vào điều kiện của phép nối có các giá trị NULL thì các giá trị NULL được xem như là không bằng nhau.

Ví dụ 2.30: Giả sử ta có hai bảng TABLE1 và TABLE2 như sau:

TABLE1

A	B
1	b1
NULL	b2
4	b3

TABLE2

C	D
NULL	d1
4	d2

Câu lệnh:

```
SELECT *
FROM table1, table2
WHERE A *= C
```

Có kết quả là:

A	B	C	D
1	b1	NULL	NULL
NULL	b2	NULL	NULL
4	b3	4	d2

2.1.7.4 Sử dụng phép nối trong SQL2

Ở phần trước đã đề cập đến phương pháp sử dụng phép nối trong và phép nối ngoài trong truy vấn SQL. Như đã trình bày, điều kiện của phép nối trong câu lệnh được chỉ định trong mệnh đề WHERE thông qua các biểu thức so sánh giữa các bảng tham gia truy vấn.

Chuẩn SQL2 (SQL-92) đưa ra một cách khác để biểu diễn cho phép nối, trong cách biểu diễn này, điều kiện của phép nối không được chỉ định trong mệnh đề WHERE mà được chỉ định ngay trong mệnh đề FROM của câu lệnh. Cách sử dụng phép nối này cho phép ta biểu diễn phép nối cũng như điều kiện nối được rõ ràng, đặc biệt là trong trường hợp phép nối được thực hiện trên ba bảng trở lên.

Phép nối trong

Điều kiện để thực hiện phép nối trong được chỉ định trong mệnh đề FROM theo cú pháp như sau:

```
tên_bảng_1 [INNER] JOIN tên_bảng_2 ON điều_kiện_nối
```

Ví dụ 2.31: Để hiển thị họ tên và ngày sinh của các sinh viên lớp *Tin K24*, thay vì sử dụng câu lệnh:

```
SELECT hodem,ten,ngaysinh
FROM sinhvien,lop
WHERE tenlop='Tin K24' AND
      sinhvien.malop=lop.malop
```

ta có thể sử dụng câu lệnh như sau:

```
SELECT hodem,ten,ngaysinh
FROM sinhvien INNER JOIN lop
     ON sinhvien.malop=lop.malop
WHERE tenlop='Tin K24'
```

Phép nối ngoài

SQL2 cung cấp các phép nối ngoài sau đây:

- Phép nối ngoài trái (LEFT OUTER JOIN)
- Phép nối ngoài phải (RIGHT OUTER JOIN)
- Phép nối ngoài đầy đủ (FULL OUTER JOIN)

Cũng tương tự như phép nối trong, điều kiện của phép nối ngoài cũng được chỉ định ngay trong mệnh đề FROM theo cú pháp:

```
tên_bảng_1 LEFT|RIGHT|FULL [OUTER] JOIN tên_bảng_2
            ON điều_kiện_nối
```

Ví dụ 2.32: Giả sử ta có hai bảng dữ liệu như sau:

Bảng DONVI

MADV	TENDV
1	Doi ngoai
2	Hanh chinh
3	Ke toan
4	Kinh doanh

Bảng NHANVIEN

HOTEN	MADV
Thanh	1
Hoa	2
Nam	2
Vinh	1
Hung	5
Phuong	NULL

Phép nối ngoài trái giữa hai bảng NHANVIEN và DONVI được biểu diễn bởi câu lệnh:

```
SELECT *
FROM nhanvien LEFT OUTER JOIN donvi
     ON nhanvien.madv=donvi.madv
```

có kết quả là:

HOTEN	MADV	MADV	TENDV
Thanh	1	1	Doi ngoai
Hoa	2	2	Hanh chinh
Nam	2	2	Hanh chinh
Vinh	1	1	Doi ngoai
Hung	5	NULL	NULL
Phuong	NULL	NULL	NULL

Câu lệnh:

```
SELECT *
FROM nhanvien RIGHT OUTER JOIN donvi
     ON nhanvien.madv=donvi.madv
```

thực hiện phép nối ngoài phải giữa hai bảng NHANVIEN và DONVI, và có kết quả là:

HOTEN	MADV	MADV	TENDV
Thanh	1	1	Doi ngoai
Vinh	1	1	Doi ngoai
Hoa	2	2	Hanh chinh
Nam	2	2	Hanh chinh
NULL	NULL	3	Ke toan
NULL	NULL	4	Kinh doanh

Nếu phép nối ngoài trái (tương ứng phải) hiển thị trong kết quả truy vấn cả những dòng dữ liệu không thoả điều kiện nối của bảng bên trái (tương ứng phải) trong phép nối thì phép nối ngoài đầy đủ hiển thị trong kết quả truy vấn cả những dòng dữ liệu không thoả điều kiện nối của cả hai bảng tham gia vào phép nối.

Ví dụ 2.33: Với hai bảng NHANVIEN và DONVI như ở trên, câu lệnh

```
SELECT *
FROM nhanvien FULL OUTER JOIN donvi
     ON nhanvien.madv=donvi.madv
```

cho kết quả là:

HOTEN	MADV	MADV	TENDV
Thanh	1	1	Doi ngoai
Hoa	2	2	Hanh chinh
Nam	2	2	Hanh chinh
Vinh	1	1	Doi ngoai
Hung	5	NULL	NULL
Phuong	NULL	NULL	NULL
NULL	NULL	4	Kinh doanh
NULL	NULL	3	Ke toan

Thực hiện phép nối trên nhiều bảng

Một đặc điểm nổi bật của SQL2 là cho phép biểu diễn phép nối trên nhiều bảng dữ liệu một cách rõ ràng. Thứ tự thực hiện phép nối giữa các bảng được xác định theo nghĩa kết quả của phép nối này được sử dụng trong một phép nối khác.

Ví dụ 2.34: Câu lệnh dưới đây hiển thị họ tên và ngày sinh của các sinh viên thuộc *Khoa Công nghệ Thông tin*

```
SELECT hodem,ten,ngaysinh
FROM (sinhvien INNER JOIN lop
                ON sinhvien.malop=lop.malop)
      INNER JOIN khoa ON lop.makhoa=khoa.makhoa
WHERE tenkhoa=N'Khoa công nghệ thông tin'
```

Trong câu lệnh trên, thứ tự thực hiện phép nối giữa các bảng được chỉ định rõ ràng: phép nối giữa hai bảng *sinhvien* và *lop* được thực hiện trước và kết quả của phép nối này lại tiếp tục được nối với bảng *khoa*.

2.1.8 Thống kê dữ liệu với GROUP BY

Ngoài khả năng thực hiện các yêu cầu truy vấn dữ liệu thông thường (chiếu, chọn, nối,…) như đã đề cập như ở các phần trước, câu lệnh SELECT còn cho phép thực hiện các thao tác truy vấn và tính toán thống kê trên dữ liệu như: c*ho biết tổng số tiết dạy của mỗi giáo viên, điểm trung bình các môn học của mỗi sinh viên*,…

Mệnh đề GROUP BY sử dụng trong câu lệnh SELECT nhằm phân hoạch các dòng dữ liệu trong bảng thành các nhóm dữ liệu, và trên mỗi nhóm dữ liệu thực hiện tính toán các giá trị thống kê như tính tổng, tính giá trị trung bình,...

Các hàm gộp được sử dụng để tính giá trị thống kê cho toàn bảng hoặc trên mỗi nhóm dữ liệu. Chúng có thể được sử dụng như là các cột trong danh sách chọn của câu lệnh SELECT hoặc xuất hiện trong mệnh đề HAVING, nhưng không được phép xuất hiện trong mệnh đề WHERE

SQL cung cấp các hàm gộp dưới đây:

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

Trong đó:

- Hàm SUM và AVG chỉ làm việc với các biểu thức số.
- Hàm SUM, AVG, COUNT, MIN và MAX bỏ qua các giá trị NULL khi tính toán.
- Hàm COUNT(*) không bỏ qua các giá trị NULL.

Mặc định, các hàm gộp thực hiện tính toán thống kê trên toàn bộ dữ liệu. Trong trường hợp cần loại bỏ bớt các giá trị trùng nhau (chỉ giữ lại một giá trị), ta chỉ định thêm từ khoá DISTINCT ở trước biểu thức là đối số của hàm.

Thống kê trên toàn bộ dữ liệu

Khi cần tính toán giá trị thống kê trên toàn bộ dữ liệu, ta sử dụng các hàm gộp trong danh sách chọn của câu lệnh SELECT. Trong trường hợp này, trong danh sách chọn không được sử dụng bất kỳ một tên cột hay biểu thức nào ngoài các hàm gộp.

Ví dụ 2.35: Để thống kê trung bình điểm lần 1 của tất cả các môn học, ta sử dụng câu lệnh như sau:

```
SELECT AVG(diemlan1)
FROM diemthi
```

còn câu lệnh dưới đây cho biết tuổi lớn nhất, tuổi nhỏ nhất và độ tuổi trung bình của tất cả các sinh viên sinh tại *Huế*:

```
SELECT MAX(YEAR(GETDATE())-YEAR(ngaysinh)),
       MIN(YEAR(GETDATE())-YEAR(ngaysinh)),
       AVG(YEAR(GETDATE())-YEAR(ngaysinh))
FROM sinhvien
WHERE noisinh='Huế'
```

Thống kê dữ liệu trên các nhóm

Trong trường hợp cần thực hiện tính toán các giá trị thống kê trên các nhóm dữ liệu, ta sử dụng mệnh đề GROUP BY để phân hoạch dữ liệu vào trong các nhóm. Các hàm gộp được sử dụng sẽ thực hiện thao tác tính toán trên mỗi nhóm và cho biết giá trị thống kê theo các nhóm dữ liệu.

Ví dụ 2.36: Câu lệnh dưới đây cho biết sĩ số (số lượng sinh viên) của mỗi lớp

```
SELECT lop.malop,tenlop,COUNT(masv) AS siso
FROM lop,sinhvien
WHERE lop.malop=sinhvien.malop
GROUP BY lop.malop,tenlop
```

và có kết quả là

MALOP	TENLOP	SISO
C24101	Toán K24	5
C24102	Tin K24	8
C24103	Lý K24	7
C24301	Sinh K24	5
C25101	Toán K25	5
C25102	Tin K25	6
C25103	Lý K25	6
C25301	Sinh K25	8
C26101	Toán K26	5
C26102	Tin K26	5

còn câu lệnh:

```
SELECT sinhvien.masv,hodem,ten,
       sum(diemlan1*sodvht)/sum(sodvht)
FROM sinhvien,diemthi,monhoc
WHERE sinhvien.masv=diemthi.masv AND
      diemthi.mamonhoc=monhoc.mamonhoc
GROUP BY sinhvien.masv,hodem,ten
```

cho biết trung bình điểm thi lần 1 các môn học của các sinh viên

Lưu ý: Trong trường hợp danh sách chọn của câu lệnh SELECT có cả các hàm gộp và những biểu thức không phải là hàm gộp thì những biểu thức này phải có mặt đầy đủ trong mệnh đề GROUP BY, nếu không câu lệnh sẽ không hợp lệ.

Ví dụ 2.37: Dưới đây là một câu lệnh sai

```
SELECT lop.malop,tenlop,COUNT(masv)
FROM lop,sinhvien
WHERE lop.malop=sinhvien.malop
GROUP BY lop.malop
```

do thiếu trường TENLOP sau mệnh đề GROUP BY.

Chỉ định điều kiện đối với hàm gộp

Mệnh đề HAVING được sử dụng nhằm chỉ định điều kiện đối với các giá trị thống kê được sản sinh từ các hàm gộp tương tự như cách thức mệnh đề WHERE thiết lập các điều kiện cho câu lệnh SELECT. Mệnh đề HAVING thường không thực sự có nghĩa nếu như không sử dụng kết hợp với mệnh đề GROUP BY. Một điểm khác biệt giữa HAVING và WHERE là trong điều kiện của WHERE không được có các hàm gộp trong khi HAVING lại cho phép sử dụng các hàm gộp trong điều kiện của mình.

Ví dụ 2.38: Để biết trung bình điểm thi lần 1 của các sinh viên có điểm trung bình lớn hơn hoặc bằng 5, ta sử dụng câu lệnh như sau:

```
SELECT sinhvien.masv,hodem,ten,
       SUM(diemlan1*sodvht)/sum(sodvht)
FROM sinhvien,diemthi,monhoc
WHERE sinhvien.masv=diemthi.masv AND
      diemthi.mamonhoc=monhoc.mamonhoc
GROUP BY sinhvien.masv,hodem,ten
HAVING sum(diemlan1*sodvht)/sum(sodvht)>=5
```

2.1.9 Thống kê dữ liệu với COMPUTE

Khi thực hiện thao tác thống kê với GROUP BY, kết quả thống kê (được sản sinh bởi hàm gộp) xuất hiện dưới một cột trong kết quả truy vấn. Thông qua dạng truy vấn này, ta biết được giá trị thống kê trên mỗi nhóm dữ liệu nhưng không biết được chi tiết dữ liệu trên mỗi nhóm

Ví dụ 2.39: Câu lệnh:

```
SELECT khoa.makhoa,tenkhoa,COUNT(malop) AS solop
FROM khoa,lop
WHERE khoa.makhoa=lop.makhoa
GROUP BY khoa.makhoa,tenkhoa
```

cho ta biết được số lượng lớp của mỗi khoa với kết quả như sau:

MAKHOA	TENKHOA	SOLOP
DHTO1	Khoa Toán cơ - Tin học	3
DHTO2	Khoa Công nghệ thông tin	3
DHTO3	Khoa Vật lý	2
DHTO5	Khoa Sinh học	2

nhưng cụ thể mỗi khoa bao gồm những lớp nào thì chúng ta không thể biết được trong kết quả truy vấn trên.

Mệnh đề COMPUTE sử dụng kết hợp với các hàm gộp (dòng) và ORDER BY trong câu lệnh SELECT cũng cho chúng ta các kết quả thống kê (của hàm gộp) trên các nhóm dữ liệu. Điểm khác biệt giữa COMPUTE và GROUP BY là kết quả thống kê xuất hiện dưới dạng một dòng trong kết quả truy vấn và còn cho chúng ta cả chi tiết về dữ liệu trong mỗi nhóm. Như vậy, câu lệnh SELECT với COMPUTE cho chúng ta cả chi tiết dữ liệu và giá trị thống kê trên mỗi nhóm.

Mệnh đề COMPUTE …BY có cú pháp như sau:

```
COMPUTE hàm_gộp(tên_cột) [,…, hàm_gộp (tên_cột)]
BY danh_sách_cột
```

Trong đó:

- Các hàm gộp có thể sử dụng bao gồm SUM, AVG, MIN, MAX và COUNT.
- *danh_sách_cột*: là danh sách cột sử dụng để phân nhóm dữ liệu

Ví dụ 2.40: Câu lệnh dưới đây cho biết danh sách các lớp của mỗi khoa và tổng số các lớp của mỗi khoa:

```
SELECT khoa.makhoa,tenkhoa,malop,tenlop FROM khoa,lop
WHERE khoa.makhoa=lop.makhoa
ORDER BY khoa.makhoa
COMPUTE COUNT(malop) BY khoa.makhoa
```

kết quả của câu lệnh như sau:

MAKHOA	TENKHOA	MALOP	TENLOP
DHT01	Khoa Toán cơ - Tin học	C24101	Toán K24
DHT01	Khoa Toán cơ - Tin học	C25101	Toán K25
DHT01	Khoa Toán cơ - Tin học	C26101	Toán K26
		CNT	
		3	

MAKHOA	TENKHOA	MALOP	TENLOP
DHT02	Khoa Công nghệ thông tin	C26102	Tin K26
DHT02	Khoa Công nghệ thông tin	C25102	Tin K25
DHT02	Khoa Công nghệ thông tin	C24102	Tin K24
		CNT	
		3	

MAKHOA	TENKHOA	MALOP	TENLOP
DHT03	Khoa Vật lý	C24103	Lý K24
DHT03	Khoa Vật lý	C25103	Lý K25
		CNT	
		2	

MAKHOA	TENKHOA	MALOP	TENLOP
DHT05	Khoa Sinh học	C25301	Sinh K25
DHT05	Khoa Sinh học	C24301	Sinh K24
		CNT	
		2	

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
- Hàm COUNT(*) không được sử dụng trong COMPUTE.
- Sau COMPUTE có thể sử dụng nhiều hàm gộp, khi đó các hàm phải phân cách nhau bởi dấu phẩy.
- Các cột sử dụng trong các hàm gộp xuất hiện trong mệnh đề COMPUTE phải có mặt 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 với hàm gộp F trên cột X theo một trong các cách dưới đây là hợp lệ:

```
COMPUTE F(X) BY a, b, c
COMPUTE F(X) BY a, b
COMPUTE F(X) BY a
```

Và các cách sử dụng dưới đây là sai:

```
COMPUTE F(X) BY b, c
COMPUTE F(X) BY a, c
COMPUTE F(X) BY c
```

- Phải sử dụng một tên cột hoặc một biểu thức trong mệnh đề ORDER BY, việc sắp xếp không được thực hiện dựa trên tiêu đề cột.

Trong trường hợp sử dụng COMPUTE mà không có BY thì có thể không cần sử dụng ORDER BY, khi đó phạm vi tính toán của hàm gộp là trên toàn bộ dữ liệu.

Ví dụ 2.41: Câu lệnh dưới đây hiển thị danh sách các lớp và tổng số lớp hiện có:

```
SELECT malop,tenlop,hedaotao
FROM lop
ORDER BY makhoa
COMPUTE COUNT(malop)
```

kết quả của câu lệnh như sau:

```
MALOP      TENLOP         HEDAOTAO
C24101     Toán K24       Chính quy
C25101     Toán K25       Chính quy
C26101     Toán K26       Chính quy
C26102     Tin K26        Chính quy
C25102     Tin K25        Chính quy
C24102     Tin K24        Chính quy
C24103     Lý K24         Chính quy
C25103     Lý K25         Chính quy
C25301     Sinh K25       Chính quy
C24301     Sinh K24       Chính quy
CNT
10
```

Có thể thực hiện việc tính toán hàm gộp dòng trên các nhóm lồng nhau bằng cách sử dụng nhiều mệnh đề COMPUTE … BY trong cùng một câu lệnh SELECT

Ví dụ 2.42: Câu lệnh:

```
SELECT khoa.makhoa,tenkhoa,malop,tenlop FROM khoa,lop
WHERE khoa.makhoa=lop.makhoa
ORDER BY khoa.makhoa
COMPUTE COUNT(malop) BY khoa.makhoa
COMPUTE COUNT(malop)
```

Cho biết danh sách các lớp của mỗi khoa, tổng số lớp theo mỗi khoa và tổng số lớp hiện có với kết quả như sau:

MAKHOA	TENKHOA	MALOP	TENLOP
DHT01	Khoa Toán cơ - Tin học	C24101	Toán K24
DHT01	Khoa Toán cơ - Tin học	C25101	Toán K25
DHT01	Khoa Toán cơ - Tin học	C26101	Toán K26
		CNT	
		3	

MAKHOA	TENKHOA	MALOP	TENLOP
DHT02	Khoa Công nghệ thông tin	C26102	Tin K26
DHT02	Khoa Công nghệ thông tin	C25102	Tin K25
DHT02	Khoa Công nghệ thông tin	C24102	Tin K24
		CNT	
		3	

MAKHOA	TENKHOA	MALOP	TENLOP
DHT03	Khoa Vật lý	C24103	Lý K24
DHT03	Khoa Vật lý	C25103	Lý K25
		CNT	
		2	

MAKHOA	TENKHOA	MALOP	TENLOP
DHT05	Khoa Sinh học	C25301	Sinh K25
DHT05	Khoa Sinh học	C24301	Sinh K24
		CNT	
		2	
		CNT	
		10	

2.1.10 Truy vấn con (Subquery)

Truy vấn con là một câu lệnh SELECT được lồng vào bên trong một câu lệnh SELECT, INSERT, UPDATE, DELETE hoặc bên trong một truy vấn con khác. Loại

truy vấn này được sử dụng để biểu diễn cho những truy vấn trong đó điều kiện truy vấn dữ liệu cần phải sử dụng đến kết quả của một truy vấn khác.

Cú pháp của truy vấn con như sau:

```
(SELECT [ALL | DISTINCT] danh_sách_chọn
FROM  danh_sách_bảng
[WHERE     điều_kiện]
[GROUP BY  danh_sách_cột]
[HAVING điều_kiện])
```

Khi sử dụng truy vấn con cần lưu ý một số quy tắc sau:

- Một truy vấn con phải được viết trong cặp dấu ngoặc. Trong hầu hết các trường hợp, một truy vấn con thường phải có kết quả là một cột (tức là chỉ có duy nhất một cột trong danh sách chọn).
- Mệnh đề COMPUTE và ORDER BY không được phép sử dụng trong truy vấn con.
- Các tên cột xuất hiện trong truy vấn con có thể là các cột của các bảng trong truy vấn ngoài.
- Một truy vấn con thường được sử dụng làm điều kiện trong mệnh đề WHERE hoặc HAVING của một truy vấn khác.
- Nếu truy vấn con trả về đúng một giá trị, nó có thể sử dụng như là một thành phần bên trong một biểu thức (chẳng hạn xuất hiện trong một phép so sánh bằng)

Phép so sánh đối với với kết quả truy vấn con

Kết quả của truy vấn con có thể được sử dụng đề thực hiện phép so sánh số học với một biểu thức của truy vấn cha. Trong trường hợp này, truy vấn con được sử dụng dưới dạng:

```
WHERE biểu_thức phép_toán_số_học [ANY|ALL]
      (truy_vấn_con)
```

Trong đó phép toán số học có thể sử dụng bao gồm: =, <>, >, <, >=, <=; Và truy vấn con phải có kết quả bao gồm đúng một cột.

Ví dụ 2.43: Câu lệnh dưới đây cho biết danh sách các môn học có số đơn vị học trình lớn hơn hoặc bằng số đơn vị học trình của môn học có mã là *TI-001*

```
SELECT *
FROM monhoc
WHERE sodvht>=(SELECT sodvht
               FROM monhoc
```

```
                WHERE mamonhoc='TI-001')
```

Nếu truy vấn con trả về nhiều hơn một giá trị, việc sử dụng phép so sánh như trên sẽ không hợp lệ. Trong trường hợp này, sau phép toán so sánh phải sử dụng thêm lượng từ ALL hoặc ANY. Lượng từ ALL được sử dụng khi cần so sánh giá trị của biểu thức với tất cả các giá trị trả về trong kết quả của truy vấn con; ngược lai, phép so sánh với lượng từ ANY có kết quả đúng khi chỉ cần một giá trị bất kỳ nào đó trong kết quả của truy vấn con thoả mãn điều kiện.

Ví dụ 2.44: Câu lệnh dưới đây cho biết họ tên của những sinh viên lớp *Tin K25* sinh trước tất cả các sinh viên của lớp *Toán K25*

```
SELECT hodem,ten
FROM sinhvien JOIN lop ON sinhvien.malop=lop.malop
WHERE tenlop='Tin K25' AND
     ngaysinh<ALL(SELECT ngaysinh
                  FROM sinhvien JOIN lop
                  ON sinhvien.malop=lop.malop
                  WHERE lop.tenlop='Toán K25')
```

và câu lệnh:

```
SELECT hodem,ten
FROM sinhvien JOIN lop on sinhvien.malop=lop.malop
WHERE tenlop='Tin K25' AND
      year(ngaysinh)= ANY(SELECT year(ngaysinh)
                          FROM sinhvien JOIN lop
                          ON sinhvien.malop=lop.malop
                          WHERE lop.tenlop='Toán K25')
```

cho biết họ tên của những sinh viên lớp *Tin K25* có năm sinh trùng với năm sinh của bất kỳ một sinh viên nào đó của lớp *Toán K25*.

Sử dụng truy vấn con với toán tử IN

Khi cần thực hiện phép kiểm tra giá trị của một biểu thức có xuất hiện (không xuất hiện) trong tập các giá trị của truy vấn con hay không, ta có thể sử dụng toán tử IN (NOT IN) như sau:

```
WHERE biểu_thức [NOT] IN (truy_vấn_con)
```

Ví dụ 2.45: Để hiển thị họ tên của những sinh viên lớp Tin K25 có năm sinh bằng với năm sinh của một sinh viên nào đó của lớp Toán K25, thay vì sử dụng câu lệnh như ở ví dụ trên, ta có thể sử dụng câu lệnh như sau:

```
SELECT hodem,ten
```

```
FROM sinhvien JOIN lop on sinhvien.malop=lop.malop
WHERE tenlop='Tin K25' AND
      year(ngaysinh)IN(SELECT year(ngaysinh)
                       FROM sinhvien JOIN lop
                       ON sinhvien.malop=lop.malop
                       WHERE lop.tenlop='Toán K25')
```

Sử dụng lượng từ EXISTS với truy vấn con

Lượng từ EXISTS được sử dụng kết hợp với truy vấn con dưới dạng:

```
WHERE [NOT] EXISTS (truy_vấn_con)
```

để kiểm tra xem một truy vấn con có trả về dòng kết quả nào hay không. Lượng từ EXISTS (tương ứng NOT EXISTS) trả về giá trị True (tương ứng False) nếu kết quả của truy vấn con có ít nhất một dòng (tương ứng không có dòng nào). Điều khác biệt của việc sử dụng EXISTS với hai cách đã nêu ở trên là trong danh sách chọn của truy vấn con có thể có nhiều hơn hai cột.

Ví dụ 2.46: Câu lệnh dưới đây cho biết họ tên của những sinh viên hiện chưa có điểm thi của bất kỳ một môn học nào

```
SELECT hodem,ten
FROM sinhvien
WHERE NOT EXISTS(SELECT masv FROM diemthi
                 WHERE diemthi.masv=sinhvien.masv)
```

Sử dụng truy vấn con với mệnh đề HAVING

Một truy vấn con có thể được sử dụng trong mệnh đề HAVING của một truy vấn khác. Trong trường hợp này, kết quả của truy vấn con được sử dụng để tạo nên điều kiện đối với các hàm gộp.

Ví dụ 2.47: Câu lệnh dưới đây cho biết mã, tên và trung bình điểm lần 1 của các môn học có trung bình lớn hơn trung bình điểm lần 1 của tất cả các môn học

```
SELECT diemthi.mamonhoc,tenmonhoc,AVG(diemlan1)
FROM diemthi,monhoc
WHERE diemthi.mamonhoc=monhoc.mamonhoc
GROUP BY diemthi.mamonhoc,tenmonhoc
HAVING AVG(diemlan1)>
       (SELECT AVG(diemlan1) FROM diemthi)
```

2.2 Bổ sung, cập nhật và xoá dữ liệu

Các câu lệnh thao tác dữ liệu trong SQL không những chỉ sử dụng để truy vấn dữ liệu mà còn để thay đổi và cập nhật dữ liệu trong cơ sở dữ liệu. So với câu lệnh SELECT, việc sử dụng các câu lệnh để bổ sung, cập nhật hay xoá dữ liệu đơn giản hơn nhiều. Trong phần còn lại của chương này sẽ đề cập đến 3 câu lệnh:

- Lệnh INSERT
- Lệnh UPDATE
- Lệnh DELETE

2.2.1 Bổ sung dữ liệu

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

- Bổ sung từng dòng dữ liệu với mỗi câu lệnh INSERT. Đây là các sử dụng thường gặp nhất trong giao tác SQL.
- Bổ sung nhiều dòng dữ liệu bằng cách truy xuất dữ liệu từ các bảng dữ liệu khác.

Bổ sung từng dòng dữ liệu với lệnh INSERT

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

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

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

Ví dụ 2.48: Câu lệnh dưới đây bổ sung thêm một dòng dữ liệu vào bảng KHOA

```
INSERT INTO khoa
VALUES('DHT10','Khoa Luật','054821135')
```

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

phép chấp nhận giá trị NULL). Nếu một cột không có giá trị mặc định và không chấp nhận giá trị NULL mà không được nhập dữ liệu, câu lệnh sẽ bị lỗi.

Ví dụ 2.49: Câu lệnh dưới đây bổ sung một bản ghi mới cho bảng SINHVIEN

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

câu lệnh trên còn có thể được viết như sau:

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

Bổ sung nhiều dòng dữ liệu từ bảng khác

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

Cú pháp câu lệnh INSERT có dạng như sau:

```
INSERT  INTO tên_bảng[(danh_sách_cột)] câu_lệnh_SELECT
```

Ví dụ 2.50: Giả sử ta có bảng LUUSINHVIEN bao gồm các trường HODEM, TEN, NGAYSINH. Câu lệnh dưới đây bổ sung vào bảng LUUSINHVIEN các dòng dữ liệu có được từ câu truy vấn SELECT:

```
INSERT INTO luusinhvien
SELECT  hodem,ten,ngaysinh
FROM sinhvien
WHERE noisinh like '%Huế%'
```

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

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

2.2.2 Cập nhật dữ liệu

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

```
UPDATE tên_bảng
```

```
SET  tên_cột = biểu_thức
     [, ..., tên_cột_k = biểu_thức_k]
[FROM  danh_sách_bảng]
[WHERE điều_kiện]
```

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

Ví dụ 2.51: Câu lệnh dưới đây cập nhật lại số đơn vị học trình của các môn học có số đơn vị học trình nhỏ hơn 2

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

Sử dụng cấu trúc CASE trong câu lệnh UPDATE

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

Ví dụ 2.52: Giả sử ta có bảng NHATKYPHONG sau đây

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

Sau khi thực hiện câu lệnh:

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

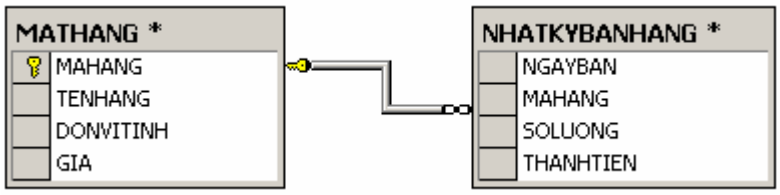
Dữ liệu trong bảng sẽ là:

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

Điều kiện cập nhật dữ liệu liên quan đến nhiều bảng

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

Ví dụ 2.53: Giả sử ta có hai bảng MATHANG và NHATKYBANHANG như sau:



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

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

Câu lệnh UPDATE với truy vấn con

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

Ví dụ 2.54: Câu lệnh ở trên có thể được viết như sau:

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

2.2.3 Xoá dữ liệu

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

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

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

Ví dụ 2.55: Câu lệnh dưới đây xoá khỏi bảng SINHVIEN những sinh viên sinh tại Huế

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

Xoá dữ liệu khi điều kiện liên quan đến nhiều bảng

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

Ví dụ 2.56: Câu lệnh dưới đây xoá ra khỏi bảng SINHVIEN những sinh viên lớp *Tin K24*

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

Sử dụng truy vấn con trong câu lệnh DELETE

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

Ví dụ 2.57: Câu lệnh dưới đây xoá khỏi bảng LOP những lớp không có sinh viên nào học

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

Xoá toàn bộ dữ liệu trong bảng

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

```
TRUNCATE TABLE tên_bảng
```

Ví dụ 2.58: Câu lệnh sau xoá toàn bộ dữ liệu trong bảng *diemthi*:

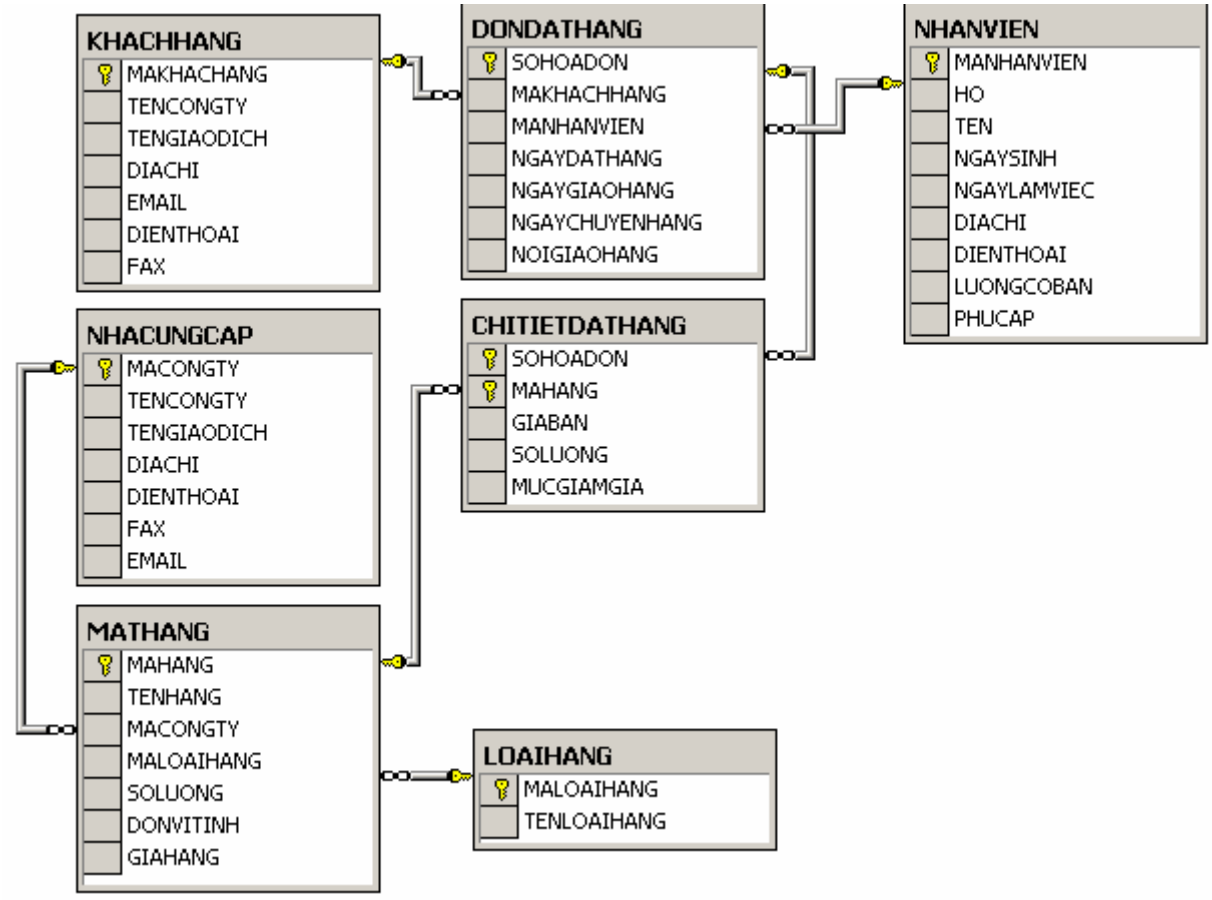
```
DELETE FROM diemthi
```

có tác dụng tương tự với câu lệnh

```
TRUNCATE TABLE diemthi
```

🕮 Bài tập chương 2

Cơ sở dữ liệu dưới đây được sử dụng để quản lý công tác giao hàng trong một công ty kinh doanh. Các bảng trong cơ sở dữ liệu này được biểu diễn trong sơ đồ dưới đây:



Trong đó:

- Bảng NHACUNGCAP lưu trữ dữ liệu về các đối tác cung cấp hàng cho công ty.
- Bảng MATHANG lưu trữ dữ liệu về các mặt hàng hiện có trong công ty.
- Bảng LOAIHANG phân loại các mặt hàng hiện có.
- Bảng NHANVIEN có dữ liệu là thông tin về các nhân viên làm việc trong công ty.
- Bảng KHACHHANG được sử dụng để lưu giữ thông tin về các khách hàng của công ty.

- Khách hàng đặt hàng cho công ty thông qua các đơn đặt hàng. Thông tin chung về các đơn đặt hàng được lưu trữ trong bảng DONDATHANG (Mỗi một đơn đặt hàng phải do một nhân viên của công ty lập và do đó bảng này có quan hệ với bảng NHANVIEN)
- Thông tin chi tiết của các đơn đặt hàng (đặt mua mặt hàng gì, số lượng, giá cả,...) được lưu trữ trong bảng CHITIETDATHANG. Bảng này có quan hệ với hai bảng DONDATHANG và MATHANG.

Sử dụng câu lệnh SELECT để viết các yêu cầu truy vấn dữ liệu sau đây:

2. 1 Cho biết danh sách các đối tác cung cấp hàng cho công ty.

2. 2 Mã hàng, tên hàng và số lượng của các mặt hàng hiện có trong công ty.

2. 3 Họ tên và địa chỉ và năm bắt đầu làm việc của các nhân viên trong công ty.

2. 4 Địa chỉ và điện thoại của nhà cung cấp có tên giao dịch *VINAMILK* là gì?

2. 5 Cho biết mã và tên của các mặt hàng có giá lớn hơn 100000 và số lượng hiện có ít hơn 50.

2. 6 Cho biết mỗi mặt hàng trong công ty do ai cung cấp.

2. 7 Công ty *Việt Tiến* đã cung cấp những mặt hàng nào?

2. 8 Loại hàng *thực phẩm* do những công ty nào cung cấp và địa chỉ của các công ty đó là gì?

2. 9 Những khách hàng nào (tên giao dịch) đã đặt mua mặt hàng *Sữa hộp XYZ* của công ty?

2. 10 Đơn đặt hàng số 1 do ai đặt và do nhân viên nào lập, thời gian và địa điểm giao hàng là ở đâu?

2. 11 Hãy cho biết số tiền lương mà công ty phải trả cho mỗi nhân viên là bao nhiêu (lương = lương cơ bản + phụ cấp).

2. 12 Trong đơn đặt hàng số 3 đặt mua những mặt hàng nào và số tiền mà khách hàng phải trả cho mỗi mặt hàng là bao nhiêu (số tiền phải trả được tính theo công thức SOLUONG×GIABAN – SOLUONG×GIABAN×MUCGIAMGIA/100)

2. 13 Hãy cho biết có những khách hàng nào lại chính là đối tác cung cấp hàng của công ty (tức là có cùng tên giao dịch).

2. 14 Trong công ty có những nhân viên nào có cùng ngày sinh?

2. 15 Những đơn đặt hàng nào yêu cầu giao hàng ngay tại công ty đặt hàng và những đơn đó là của công ty nào?

2. 16 Cho biết tên công ty, tên giao dịch, địa chỉ và điện thoại của các khách hàng và các nhà cung cấp hàng cho công ty.

2. 17 Những mặt hàng nào chưa từng được khách hàng đặt mua?

2. 18 Những nhân viên nào của công ty chưa từng lập bất kỳ một hoá đơn đặt hàng nào?

2. 19 Những nhân viên nào của công ty có lương cơ bản cao nhất?

2. 20 Tổng số tiền mà khách hàng phải trả cho mỗi đơn đặt hàng là bao nhiêu?

2. 21 Trong năm 2003, những mặt hàng nào chỉ được đặt mua đúng một lần.

2. 22 Hãy cho biết mỗi một khách hàng đã phải bỏ ra bao nhiêu tiền để đặt mua hàng của công ty?

2. 23 Mỗi một nhân viên của công ty đã lập bao nhiêu đơn đặt hàng (nếu nhân viên chưa hề lập một hoá đơn nào thì cho kết quả là 0)

2. 24 Cho biết tổng số tiền hàng mà cửa hàng thu được trong mỗi tháng của năm 2003 (thời được gian tính theo ngày đặt hàng).

2. 25 Hãy cho biết tổng số tiền lời mà công ty thu được từ mỗi mặt hàng trong năm 2003.

2. 26 Hãy cho biết tổng số lượng hàng của mỗi mặt hàng mà công ty đã có (tổng số lượng hàng hiện có và đã bán).

2. 27 Nhân viên nào của công ty bán được số lượng hàng nhiều nhất và số lượng hàng bán được của những nhân viên này là bao nhiêu?

2. 28 Đơn đặt hàng nào có số lượng hàng được đặt mua ít nhất?

2. 29 Số tiền nhiều nhất mà mỗi khách hàng đã từng bỏ ra để đặt hàng trong các đơn đặt hàng là bao nhiêu?

2. 30 Mỗi một đơn đặt hàng đặt mua những mặt hàng nào và tổng số tiền mà mỗi đơn đặt hàng phải trả là bao nhiêu?

2. 31 Hãy cho biết mỗi một loại hàng bao gồm những mặt hàng nào, tổng số lượng hàng của mỗi loại và tổng số lượng của tất cả các mặt hàng hiện có trong công ty là bao nhiêu?

2. 32 Thống kê xem trong năm 2003, mỗi một mặt hàng trong mỗi tháng và trong cả năm bán được với số lượng bao nhiêu

Yêu cầu: Kết quả được hiển thị dưới dạng bảng, hai cột cột đầu là mã hàng và tên hàng, các cột còn lại tương ứng với các tháng từ 1 đến 12 và cả năm. Như vậy mỗi dòng trong kết quả cho biết số lượng hàng bán được mỗi tháng và trong cả năm của mỗi mặt hàng.

Sử dụng câu lệnh UPDATE để thực hiện các yêu cầu sau:

2. 33 Cập nhật lại giá trị trường NGAYCHUYENHANG của những bản ghi có NGAYCHUYENHANG chưa xác định (NULL) trong bảng DONDATHANG bằng với giá trị của trường NGAYDATHANG.

2. 34 Tăng số lượng hàng của những mặt hàng do công ty VINAMILK cung cấp lên gấp đôi.

2. 35 Cập nhật giá trị của trường NOIGIAOHANG trong bảng DONDATHANG bằng địa chỉ của khách hàng đối với những đơn đặt hàng chưa xác định được nơi giao hàng (giá trị trường NOIGIAOHANG bằng NULL).

2. 36 Cập nhật lại dữ liệu trong bảng KHACHHANG sao cho nếu tên công ty và tên giao dịch của khách hàng trùng với tên công ty và tên giao dịch của một nhà cung cấp nào đó thì địa chỉ, điện thoại, fax và e-mail phải giống nhau.

2. 37 Tăng lương lên gấp rưỡi cho những nhân viên bán được số lượng hàng nhiều hơn 100 trong năm 2003.

2. 38 Tăng phụ cấp lên bằng 50% lương cho những nhân viên bán được hàng nhiều nhất.

2. 39 Giảm 25% lương của những nhân viên trong năm 2003 không lập được bất kỳ đơn đặt hàng nào.

2. 40 Giả sử trong bảng DONDATHANG có thêm trường SOTIEN cho biết số tiền mà khách hàng phải trả trong mỗi đơn đặt hàng. Hãy tính giá trị cho trường này.

Thực hiện các yêu cầu dưới đây bằng câu lệnh DELETE.

2. 41 Xoá khỏi bảng NHANVIEN những nhân viên đã làm việc trong công ty quá 40 năm.

2. 42 Xoá những đơn đặt hàng trước năm 2000 ra khỏi cơ sở dữ liệu.

2. 43 Xoá khỏi bảng LOAIHANG những loại hàng hiện không có mặt hàng.

2. 44 Xoá khỏi bảng KHACHHANG những khách hàng hiện không có bất kỳ đơn đặt hàng nào cho công ty.

2. 45 Xoá khỏi bảng MATHANG những mặt hàng có số lượng bằng 0 và không được đặt mua trong bất kỳ đơn đặt hàng nào.

✍ Lời giải:

Các phép nối được sử dụng trong các truy vấn dưới đây sử dụng cú pháp của SQL2.

```
2.1  SELECT macongty,tencongty,tengiaodich
     FROM nhacungcap
2.2  SELECT mahang,tenhang,soluong
     FROM mathang
2.3  SELECT ho,ten,year(ngaylamviec) AS namlamviec
     FROM nhanvien
```

```
2.4  SELECT diachi,dienthoai
     FROM nhacungcap
     WHERE tengiaodich='VINAMILK'
2.5  SELECT mahang,tenhang
     FROM mathang
     WHERE giahang>100000 AND soluong<50
2.6  SELECT mahang,tenhang,
            nhacungcap.macongty,tencongty,tengiaodich
     FROM mathang INNER JOIN nhacungcap
          ON mathang.macongty=nhacungcap.macongty
2.7  SELECT mahang,tenhang
     FROM mathang INNER JOIN nhacungcap
          ON mathang.macongty=nhacungcap.macongty
     WHERE tencongty='Việt Tiến'
2.8  SELECT DISTINCT nhacungcap.macongty,tencongty,diachi
     FROM (loaihang INNER JOIN mathang
          ON loaihang.maloaihang=mathang.maloaihang)
          INNER JOIN nhacungcap
          ON mathang.macongty=nhacungcap.macongty
     WHERE tenloaihang='Thực phẩm'
2.9  SELECT DISTINCT tengiaodich
     FROM ((mathang INNER JOIN chitietdathang
          ON mathang.mahang=chitietdathang.mahang)
          INNER JOIN dondathang
          ON chitietdathang.sohoadon=dondathang.sohoadon)
          INNER JOIN khachhang
          ON dondathang.makhachhang=khachhang.makhachhang
     WHERE tenhang='Sữa hộp'
2.10 SELECT dondathang.manhanvien,ho,ten,
            ngaygiaohang,noigiaohang
     FROM nhanvien INNER JOIN dondathang
          ON nhanvien.manhanvien=dondathang.manhanvien
     WHERE sohoadon=1
2.11 SELECT manhanvien,ho,ten,
     luongcoban + CASE
                    WHEN phucap IS NULL THEN 0
                    ELSE phucap
                  END AS luong
     FROM nhanvien
2.12 SELECT a.mahang,tenhang,
```

```
            a.soluong*giaban*(1-mucgiamgia/100) AS sotien
     FROM chitietdathang AS a INNER JOIN mathang AS b
          ON a.mahang=b.mahang
2.13 SELECT makhachhang,khachhang.tencongty,
            khachhang.tengiaodich
     FROM khachhang INNER JOIN nhacungcap
          ON khachhang.tengiaodich=nhacungcap.tengiaodich
2.14 SELECT a.ho,a.ten,b.ho,b.ten,b.ngaysinh
     FROM nhanvien a INNER JOIN nhanvien b
          ON a.ngaysinh=b.ngaysinh AND
             a.manhanvien<>b.manhanvien
2.15 SELECT sohoadon,tencongty,tengiaodich,
            ngaydathang,noigiaohang
     FROM dondathang INNER JOIN khachhang
          ON dondathang.noigiaohang=khachhang.diachi
2.16 SELECT tencongty,tengiaodich,diachi,dienthoai
     FROM khachhang
     UNION ALL
     SELECT tencongty,tengiaodich,diachi,dienthoai
     FROM nhacungcap
2.17 SELECT mahang,tenhang
     FROM mathang
     WHERE NOT EXISTS (SELECT mahang FROM chitietdathang
                       WHERE mahang=mathang.mahang)
2.18 SELECT manhanvien,ho,ten
     FROM nhanvien
     WHERE NOT EXISTS (SELECT manhanvien FROM dondathang
                       WHERE manhanvien=nhanvien.manhanvien)
2.19 SELECT manhanvien,ho,ten,luongcoban
     FROM nhanvien
     WHERE luongcoban=(SELECT MAX(luongcoban) FROM nhanvien)
2.20 SELECT dondathang.sohoadon,dondathang.makhachhang,
            tencongty,tengiaodich,
            SUM(soluong*giaban-soluong*giaban*mucgiamgia/100)
     FROM (khachhang INNER JOIN dondathang
          ON khachhang.makhachhang=dondathang.makhachhang)
          INNER JOIN chitietdathang
          ON dondathang.sohoadon=chitietdathang.sohoadon
          GROUP BY dondathang.makhachhang,tencongty,
                   tengiaodich,dondathang.sohoadon
```

```
2.21 SELECT mathang.mahang,tenhang
     FROM (mathang INNER JOIN chitietdathang
           ON mathang.mahang=chitietdathang.mahang)
           iNNER JOIN dondathang
           ON chitietdathang.sohoadon=dondathang.sohoadon
     WHERE YEAR(ngaydathang)=2003
     GROUP BY mathang.mahang,tenhang
     HAVING COUNT(chitietdathang.mahang)=1
2.22 SELECT khachhang.makhachhang,tencongty,tengiaodich,
           SUM(soluong*giaban-soluong*giaban*mucgiamgia/100)
     FROM (khachhang INNER JOIN dondathang
           ON khachhang.makhachhang = dondathang.makhachhang)
           INNER JOIN chitietdathang
           ON dondathang.sohoadon=chitietdathang.sohoadon
     GROUP BY khachhang.makhachhang,tencongty,tengiaodich
2.23 SELECT nhanvien.manhanvien,ho,ten,COUNT(sohoadon)
     FROM nhanvien LEFT OUTER JOIN dondathang
          ON nhanvien.manhanvien=dondathang.manhanvien
     GROUP BY nhanvien.manhanvien,ho,ten
2.24 SELECT MONTH(ngaydathang) AS thang,
            SUM(soluong*giaban-soluong*giaban*mucgiamgia/100)
     FROM dondathang INNER JOIN chitietdathang
          ON dondathang.sohoadon=chitietdathang.sohoadon
     WHERE year(ngaydathang)=2003
     GROUP BY month(ngaydathang)
2.25 SELECT c.mahang,tenhang,
       SUM(b.soluong*giaban-b.soluong*giaban*mucgiamgia/100)-
       SUM(b.soluong*giahang)
     FROM (dondathang AS a INNER JOIN chitietdathang AS b
          ON a.sohoadon=b.sohoadon)
          INNER JOIN mathang AS c
          ON b.mahang=c.mahang
     WHERE YEAR(ngaydathang)=2003
     GROUP BY c.mahang,tenhang
2.26 SELECT mathang.mahang,tenhang,
            mathang.soluong +
            CASE
               WHEN SUM(chitietdathang.soluong) IS NULL THEN 0
               ELSE SUM(chitietdathang.soluong)
            END AS tongsoluong
```

```
     FROM mathang LEFT OUTER JOIN chitietdathang
     ON mathang.mahang=chitietdathang.mahang
     GROUP BY mathang.mahang,tenhang,mathang.soluong
2.27 SELECT nhanvien.manhanvien,ho,ten,sum(soluong)
     FROM (nhanvien INNER JOIN dondathang
           ON nhanvien.manhanvien=dondathang.manhanvien)
           INNER JOIN chitietdathang
           ON dondathang.sohoadon=chitietdathang.sohoadon
     GROUP BY nhanvien.manhanvien,ho,ten
     HAVING sum(soluong)>=ALL(SELECT sum(soluong)
           FROM (nhanvien INNER JOIN dondathang
                ON nhanvien.manhanvien=dondathang.manhanvien)
                INNER JOIN chitietdathang ON
                dondathang.sohoadon=chitietdathang.sohoadon
           GROUP BY nhanvien.manhanvien,ho,ten)
2.28 SELECT dondathang.sohoadon,SUM(soluong)
     FROM dondathang INNER JOIN chitietdathang
          ON dondathang.sohoadon=chitietdathang.sohoadon
     GROUP BY dondathang.sohoadon
     HAVING sum(soluong)<=ALL(SELECT sum(soluong)
           FROM dondathang INNER JOIN chitietdathang
               ON dondathang.sohoadon=chitietdathang.sohoadon
          GROUP BY dondathang.sohoadon)
2.29 SELECT TOP 1
          SUM(soluong*giaban-soluong*giaban*mucgiamgia/100)
     FROM dondathang INNER JOIN chitietdathang
          ON dondathang.sohoadon=chitietdathang.sohoadon
     ORDER BY 1 DESC
2.30 SELECT a.sohoadon,b.mahang,tenhang,
          b.soluong*giaban-b.soluong*giaban*mucgiamgia/100
     FROM (dondathang AS a INNER JOIN chitietdathang AS b
          ON a.sohoadon = b.sohoadon)
          INNER JOIN mathang AS c ON b.mahang = c.mahang
     ORDER BY a.sohoadon
     COMPUTE SUM(b.soluong*giaban-
             b.soluong*giaban*mucgiamgia/100) BY a.sohoadon
2.31 SELECT loaihang.maloaihang,tenloaihang,
            mahang,tenhang,soluong
     FROM loaihang INNER JOIN mathang
          ON loaihang.maloaihang=mathang.maloaihang
```

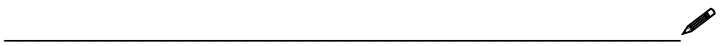
```
     ORDER BY loaihang.maloaihang
     COMPUTE SUM(soluong) BY loaihang.maloaihang
     COMPUTE SUM(soluong)
2.32 SELECT b.mahang,tenhang,
           SUM(CASE MONTH(ngaydathang) WHEN 1 THEN b.soluong
               ELSE 0 END) AS Thang1,
           SUM(CASE MONTH(ngaydathang) WHEN 2 THEN b.soluong
               ELSE 0 END) AS Thang2,
           SUM(CASE MONTH(ngaydathang) WHEN 3 THEN b.soluong
               ELSE 0 END) AS Thang3,
           SUM(CASE MONTH(ngaydathang) WHEN 4 THEN b.soluong
               ELSE 0 END) AS Thang4,
           SUM(CASE MONTH(ngaydathang) WHEN 5 THEN b.soluong
               ELSE 0 END) AS Thang5,
           SUM(CASE MONTH(ngaydathang) WHEN 6 THEN b.soluong
               ELSE 0 END) AS Thang6,
           SUM(CASE MONTH(ngaydathang) WHEN 7 THEN b.soluong
               ELSE 0 END) AS Thang7,
           SUM(CASE MONTH(ngaydathang) WHEN 8 THEN b.soluong
               ELSE 0 END) AS Thang8,
           SUM(CASE MONTH(ngaydathang) WHEN 9 THEN b.soluong
               ELSE 0 END) AS Thang9,
           SUM(CASE MONTH(ngaydathang) WHEN 10 THEN b.soluong
               ELSE 0 END) AS Thang10,
           SUM(CASE MONTH(ngaydathang) WHEN 11 THEN b.soluong
               ELSE 0 END) AS Thang11,
           SUM(CASE MONTH(ngaydathang) WHEN 12 THEN b.soluong
               ELSE 0 END) AS Thang12,
           SUM(b.soluong) AS CaNam
     FROM (dondathang AS a INNER JOIN chitietdathang AS b
           ON a.sohoadon=b.sohoadon)
           INNER JOIN mathang AS c ON b.mahang=c.mahang
     WHERE YEAR(ngaydathang)=1996
     GROUP BY b.mahang,tenhang

2.33 UPDATE dondathang
     SET ngaychuyenhang = ngaydathang
     WHERE ngaychuyenhang IS NULL
2.34 UPDATE mathang
     SET soluong=soluong*2
```

```
     FROM nhacungcap
     WHERE nhacungcap.macongty=mathang.macongty AND
           tencongty='VINAMILK'
2.35 UPDATE dondathang
     SET noigiaohang=diachi
     FROM khachhang
     WHERE dondathang.makhachhang=khachhang.makhachang AND
           noigiaohang IS NULL
2.36 UPDATE khachhang
     SET  khachhang.diachi = nhacungcap.diachi,
          khachhang.dienthoai = nhacungcap.dienthoai,
          khachhang.fax = nhacungcap.fax,
          khachhang.email = nhacungcap.email
     FROM nhacungcap
     WHERE khachhang.tencongty = nhacungcap.tencongty AND
           khachhang.tengiaodich = nhacungcap.tengiaodich
2.37 UPDATE nhanvien
     SET luongcoban=luongcoban*1.5
     WHERE manhanvien =
          (SELECT manhanvien
          FROM dondathang INNER JOIN chitietdathang
          ON dondathang.sohoadon=chitietdathang.sohoadon
          WHERE manhanvien=nhanvien.manhanvien
          GROUP BY manhanvien
          HAVING SUM(soluong)>100)
2.38 UPDATE nhanvien
     SET phucap=luongcoban/2
     WHERE manhanvien IN
          (SELECT manhanvien
          FROM dondathang INNER JOIN chitietdathang
          ON dondathang.sohoadon=chitietdathang.sohoadon
          GROUP BY manhanvien
          HAVING SUM(soluong)>=ALL
               (SELECT SUM(soluong)
               FROM dondathang INNER JOIN chitietdathang
               ON dondathang.sohoadon=chitietdathang.sohoadon
               GROUP BY manhanvien))
2.39 UPDATE nhanvien
     SET luongcoban=luongcoban*0.85
     WHERE NOT EXISTS (SELECT manhanvien
```

```
                        FROM dondathang
                        WHERE manhanvien=nhanvien.manhanvien)
2.40 UPDATE dondathang
     SET sotien =
       (SELECT SUM(soluong*giaban+soluong*giaban*mucgiamgia)
        FROM chitietdathang
        WHERE sohoadon=dondathang.sohoadon
        GROUP BY sohoadon)

2.41 DELETE FROM nhanvien
     WHERE DATEDIFF(YY,ngaylamviec,GETDATE())>40
2.42 DELETE FROM dondathang
     WHERE ngaydathang<'1/1/2000'
2.43 DELETE FROM loaihang
     WHERE NOT EXISTS (SELECT mahang
                       FROM mathang
                       WHERE maloaihang=loaihang.maloaihang)
2.44 DELETE FROM khachhang
     WHERE NOT EXISTS (SELECT sohoadon FROM dondathang
                       WHERE makhachhang=khachhang.makhachhang)
2.45 DELETE FROM mathang
     WHERE soluong=0 AND
          NOT EXISTS (SELECT sohoadon
          FROM chitietdathang
          WHERE mahang=mathang.mahang)
```



Chương 3

NGÔN NGỮ ĐỊNH NGHĨA DỮ LIỆU

Các câu lệnh SQL đã đề cập đến trong chương 3 được sử dụng nhằm thực hiện các thao tác bổ sung, cập nhật, loại bỏ và xem dữ liệu. Nhóm các câu lệnh này được gọi là ngôn ngữ thao tác dữ liệu (DML). Trong chương này, chúng ta sẽ tìm hiểu nhóm các câu lệnh được sử dụng để định nghĩa và quản lý các đối tượng CSDL như bảng, khung nhìn, chỉ mục,... và được gọi là ngôn ngữ định nghĩa dữ liệu (DLL).

Về cơ bản, ngôn ngữ định nghĩa dữ liệu bao gồm các lệnh:

- CREATE: định nghĩa và tạo mới đối tượng CSDL.
- ALTER: thay đổi định nghĩa của đối tượng CSDL.
- DROP: Xoá đối tượng CSDL đã có.

3.1 Tạo bảng dữ liệu

Như đã nói đến ở chương 1, bảng dữ liệu là cấu trúc có vai trò quan trọng nhất trong cơ sở dữ liệu quan hệ. Toàn bộ dữ liệu của cơ sở dữ liệu được tổ chức trong các bảng, những bảng này có thể là những bảng hệ thống được tạo ra khi tạo lập cơ sở dữ liệu, và cũng có thể là những bảng do người sử dụng định nghĩa.

MAKHOA	TENKHOA	DIENTHOAI	TRUONGKHOA
DHT01	Khoa Toán cơ - Tin học	054822407	Trần Lộc Hùng
DHT02	Khoa Công nghệ thông tin	054826767	Nguyễn Mậu Hân
DHT03	Khoa Vật lý	054823462	Trương Văn Chương
DHT04	Khoa Hoá học	054823951	Nguyễn Văn Hợp
DHT05	Khoa Sinh học	054822934	Đỗ Quý Hai
DHT06	Khoa Địa lý - Địa chất	054823837	NULL
DHT07	Khoa Ngữ văn	054821133	Hoàng Tất Thắng
DHT08	Khoa Lịch sử	054823833	Nguyễn Văn Mạnh
DHT09	Khoa Mác - Lê Nin	054825698	Đoàn Đức Hiếu
DHT10	Khoa Luật	054821135	Đoàn Đức Lương

***Hình 3.1** Bảng trong cơ sở dữ liệu quan hệ*

Trong các bảng, dữ liệu được tổ chức dưới dạng các dòng và cột. Mỗi một dòng là một bản ghi duy nhất trong bảng và mỗi một cột là một trường. Các bảng trong cơ sở

dữ liệu được sử dụng để biểu diễn thông tin, lưu giữ dữ liệu về các đối tượng trong thế giới thực và/hoặc mối quan hệ giữa các đối tượng. Bảng trong hình 3.1 bao gồm 10 bản ghi và 4 trường là MAKHOA, TENKHOA, DIENTHOAI và TRUONGKHOA.

Câu lệnh CREATE TABLE được sử dụng để định nghĩa một bảng dữ liệu mới trong cơ sở dữ liệu. Khi định nghĩa một bảng dữ liệu mới, ta cần phải xác định được các yêu cầu sau đây:

- Bảng mới được tạo ra sử dụng với mục đích gì và có vai trò như thế nào trong cơ sở dữ liệu.
- Cấu trúc của bảng bao gồm những trường (cột) nào, mỗi một trường có ý nghĩa như thế nào trong việc biểu diễn dữ liệu, kiểu dữ liệu của mỗi trường là gì và trường đó có cho phép nhận giá trị NULL hay không.
- Những trường nào sẽ tham gia vào khóa chính của bảng. Bảng có quan hệ với những bảng khác hay không và nếu có thì quan hệ như thế nào.
- Trên các trường của bảng có tồn tại những ràng buộc về khuôn dạng, điều kiện hợp lệ của dữ liệu hay không; nếu có thì sử dụng ở đâu và như thế nào.

Câu lệnh CREATE TABLE có cú pháp như sau

```
CREATE TABLE tên_bảng
(
      tên_cột   thuộc_tính_cột  các_ràng_buộc
     [,...
     ,tên_cột_n thuộc_tính_cột_n  các_ràng_buộc_cột_n]
     [,các_ràng_buộc_trên_bảng]
)
```

Trong đó:

`tên_bảng` Tên của bảng cần tạo. Tên phải tuân theo qui tắc định danh và không được vượt quá 128 ký tự.

`tên_cột` Là tên của cột (trường) cần định nghĩa, tên cột phải tuân theo qui tắc định danh và không được trùng nhau trong mỗi một bảng. Mỗi một bảng phải có ít nhất một cột. Nếu bảng có nhiều cột thì định nghĩa của các cột (tên cột, thuộc tính và các ràng buộc) phải phân cách nhau bởi dấu phẩy.

`thuộc_tính_cột` Mỗi một cột trong một bảng ngoài tên cột còn có các thuộc tính bao gồm:

- Kiểu dữ liệu của cột. Đây là thuộc tính bắt buộc phải có đối với mỗi cột.
- Giá trị mặc định của cột: là giá trị được tự động gán cho cột nếu như người sử dụng không nhập dữ liệu

cho cột một cách tường minh. Mỗi một cột chỉ có thể có nhiều nhất một giá trị mặc định.

- Cột có tính chất IDENTITY hay không? tức là giá trị của cột có được tự động tăng mỗi khi có bản ghi mới được bổ sung hay không. Tính chất này chỉ có thể sử dụng đối với các trường kiểu số.
- Cột có chấp nhận giá trị NULL hay không

Ví dụ 3.1: Khai báo dưới đây định nghĩa cột STT có kiểu dữ liệu là *int* và cột có tính chất IDENTITY:

```
stt INT IDENTITY
```

hay định nghĩa cột *NGAY* có kiểu *datetime* và không cho phép chấp nhận giá trị *NULL*:

```
ngay DATETIME NOT NULL
```

và định nghĩa cột *SOLUONG* kiểu *int* và có giá trị mặc định là *0*:

```
soluong INT DEFAULT (0)
```

`các_ràng_buộc`

Các ràng buộc được sử dụng trên mỗi cột hoặc trên bảng nhằm các mục đích sau:

- Quy định khuôn dạng hay giá trị dữ liệu được cho phép trên cột (chẳng hạn qui định tuổi của một học sinh phải lớn hơn 6 và nhỏ hơn 20, số điện thoại phải là một chuỗi bao gồm 6 chữ số,...). Những ràng buộc kiểu này được gọi là ràng buộc CHECK
- Đảm bảo tính toàn vẹn dữ liệu trong một bảng và toàn vẹn tham chiếu giữa các bảng trong cơ sở dữ liệu. Những loại ràng buộc này nhằm đảm bảo tính đùng của dữ liệu như: số chứng minh nhân dân của mỗi một người phải duy nhất, nếu sinh viên học một lớp nào đó thì lớp đó phải tồn tại,... Liên quan đến những loại ràng buộc này bao gồm các ràng buộc PRIMARY KEY (khoá chính), UNIQUE (khóa dự tuyển) và FOREIGN KEY (khoá ngoài)

Các loại ràng buộc này sẽ được trình bày chi tiết hơn ở phần sau.

Ví dụ 3.2: Câu lệnh dưới đây định nghĩa bảng NHANVIEN với các trường MANV (mã nhân viên), HOTEN (họ và tên), NGAYSINH (ngày sinh của nhân viên), DIENTHOAI (điện thoại) và HSLUONG (hệ số lương)

```
CREATE TABLE nhanvien
(
manv      NVARCHAR(10)   NOT NULL,
hoten     NVARCHAR(50)   NOT NULL,
ngaysinh  DATETIME       NULL,
dienthoai NVARCHAR(10)   NULL,
hsluong   DECIMAL(3,2)   DEFAULT (1.92)
)
```

Trong câu lệnh trên, trường MANV và HOTEN của bảng NHANVIEN không được NULL (tức là bắt buộc phải có dữ liệu), trường NGAYSINH và DIENTHOAI sẽ nhận giá trị NULL nếu ta không nhập dữ liệu cho chúng còn trường HSLUONG sẽ nhận giá trị mặc định là 1.92 nếu không được nhập dữ liệu.

Nếu ta thực hiện các câu lệnh dưới đây sau khi thực hiện câu lệnh trên để bổ sung dữ liệu cho bảng NHANVIEN

```
INSERT INTO nhanvien
     VALUES('NV01','Le Van A','2/4/75','886963',2.14)
INSERT INTO nhanvien(manv,hoten)
     VALUES('NV02','Mai Thi B')
INSERT INTO nhanvien(manv,hoten,dienthoai)
     VALUES('NV03','Tran Thi C','849290')
```

Ta sẽ có được dữ liệu trong bảng NHANVIEN như sau:

MANV	HOTEN	NGAYSINH	DIENTHOAI	HSLUONG
NV01	Le Van A	1975-02-04 00:00:00.000	886963	2.14
NV02	Mai Thi B	NULL	NULL	1.92
NV03	Tran Thi C	NULL	849290	1.92

3.1.1 Ràng buộc CHECK

Ràng buộc CHECK được sử dụng nhằm chỉ định điều kiện hợp lệ đối với dữ liệu. Mỗi khi có sự thay đổi dữ liệu trên bảng (INSERT, UPDATE), những ràng buộc này sẽ được sử dụng nhằm kiểm tra xem dữ liệu mới có hợp lệ hay không.

Ràng buộc CHECK được khai báo theo cú pháp như sau:

```
[CONSTRAINT tên_ràng_buộc]
CHECK (điều_kiện)
```

Trong đó, điều_kiện là một biểu thức logic tác động lên cột nhằm qui định giá trị hoặc khuôn dạng dữ liệu được cho phép. Trên mỗi một bảng cũng như trên mỗi một cột có thể có nhiều ràng buộc CHECK.

Ví dụ 3.3: Câu lệnh dưới đây tạo bảng DIEMTOTNGHIEP trong đó qui định giá trị của cột DIEMVAN và DIEMTOAN phải lớn hơn hoặc bằng 0 và nhỏ hơn hoặc bằng 10.

```
CREATE TABLE diemtotnghiep
(
hoten     NVARCHAR(30) NOT NULL,
ngaysinh  DATETIME,
diemvan   DECIMAL(4,2)
          CONSTRAINT chk_diemvan
          CHECK(diemvan>=0 AND diemvan<=10),
diemtoan  DECIMAL(4,2)
          CONSTRAINT chk_diemtoan
          CHECK(diemtoan>=0 AND diemtoan<=10),
)
```

Như vậy, với định nghĩa như trên của bảng DIEMTOTNGHIEP, các câu lệnh dưới đây là hợp lệ:

```
INSERT INTO diemtotnghiep(hoten,diemvan,diemtoan)
       VALUES('Le Thanh Hoang',9.5,2.5)
INSERT INTO diemtotnghiep(hoten,diemvan)
       VALUES('Hoang Thi Mai',2.5)
```

còn câu lệnh dưới đây là không hợp lệ:

```
INSERT INTO diemtotnghiep(hoten,diemvan,diemtoan)
       VALUES('Tran Van Hanh',6,10.5)
```

do cột DIEMTOAN nhận giá trị 10.5 không thoả mãn điều kiện của ràng buộc

Trong ví dụ trên, các ràng buôc được chỉ định ở phần khai báo của mỗi cột. Thay vì chỉ định ràng buộc trên mỗi cột, ta có thể chỉ định các ràng buộc ở mức bảng bằng cách khai báo các ràng buộc sau khi đã khai báo xong các cột trong bảng.

Ví dụ 3.4: Câu lệnh

```
CREATE TABLE lop
(
malop       NVARCHAR(10)      NOT NULL ,
tenlop      NVARCHAR(30)      NOT NULL ,
khoa        SMALLINT          NULL ,
hedaotao    NVARCHAR(25)      NULL
```

```
                CONSTRAINT chk_lop_hedaotao
                CHECK (hedaotao IN ('chính quy','tại chức')),
    namnhaphoc  INT                NULL
                CONSTRAINT chk_lop_namnhaphoc
                CHECK (namnhaphoc<=YEAR(GETDATE())),
    makhoa      NVARCHAR(5)
    )
```

có thể được viết lại như sau:

```
    CREATE TABLE lop
    (
    malop       NVARCHAR(10)      NOT NULL ,
    tenlop      NVARCHAR(30)      NOT NULL ,
    khoa        SMALLINT          NULL ,
    hedaotao    NVARCHAR(25)      NULL,
    namnhaphoc  INT               NULL ,
    makhoa      NVARCHAR(5),
    CONSTRAINT chk_lop
    CHECK (namnhaphoc<=YEAR(GETDATE()) AND
           hedaotao IN ('chính quy','tại chức'))
    )
```

3.1.2 Ràng buộc PRIMARY KEY

Ràng buộc PRIMARY KEY được sử dụng để định nghĩa khoá chính của bảng. Khoá chính của một bảng là một hoặc một tập nhiều cột mà giá trị của chúng là duy nhất trong bảng. 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, ta sử dụng cú pháp như sau:

```
[CONSTRAINT tên_ràng_buộc]
PRIMARY KEY [(danh_sách_cột)]
```

Nếu khoá chính của bảng chỉ bao gồm đúng một cột và ràng buộc PRIMARY KEY được chỉ định ở mức cột, ta không cần thiết phải chỉ định danh sách cột sau từ khoá PRIMARY KEY. Tuy nhiên, nếu việc khai báo khoá chính được tiến hành ở mức bảng (sử dụng khi số lượng các cột tham gia vào khoá là từ hai trở lên) thì bắt buộc phải chỉ định danh sách cột ngay sau từ khóa PRIMARY KEY và tên các cột được phân cách nhau bởi dấu phẩy.

Ví dụ 3.5: Câu lệnh dưới đây định nghĩa bảng SINHVIEN với khoá chính là MASV

```
CREATE TABLE sinhvien
(
masv      NVARCHAR(10)
          CONSTRAINT pk_sinhvien_masv PRIMARY KEY,
hodem     NVARCHAR(25)   NOT NULL ,
ten       NVARCHAR(10)   NOT NULL ,
ngaysinh  DATETIME,
gioitinh  BIT,
noisinh   NVARCHAR(255),
malop     NVARCHAR(10)
)
```

Với bảng vừa được tạo bởi câu lệnh ở trên, nếu ta thực hiện câu lệnh:

```
INSERT INTO sinhvien(masv,hodem,ten,gioitinh,malop)
VALUES('0261010001','Lê Hoàng Phương','Anh',0,'C26101')
```

một bản ghi mới sẽ được bổ sung vào bảng này. Nhưng nếu ta thực hiện tiếp câu lệnh:

```
INSERT INTO sinhvien(masv,hodem,ten,gioitinh,malop)
VALUES('0261010001','Lê Huy','Đan',1,'C26101')
```

thì câu lệnh này sẽ bị lỗi do trùng giá trị khoá với bản ghi đã có.

Ví dụ 3.6: Câu lệnh dưới đây tạo bảng DIEMTHI với khoá chính là tập bao gồm hai cột MAMONHOC và MASV

```
CREATE TABLE diemthi
(
mamonhoc  NVARCHAR(10)   NOT NULL ,
masv      NVARCHAR(10)   NOT NULL ,
diemlan1  NUMERIC(4, 2),
diemlan2  NUMERIC(4, 2),
CONSTRAINT pk_diemthi PRIMARY KEY(mamonhoc,masv)
)
```

Lưu ý:

- Mỗi một bảng chỉ có thể có nhiều nhất một ràng buộc PRIMARY KEY.
- Một khoá chính có thể bao gồm nhiều cột nhưng không vượt quá 16 cột.

3.1.3 Ràng buộc UNIQUE

Trên một bảng chỉ có thể có nhiều nhất một khóa chính nhưng có thể có nhiều cột hoặc tập các cột có tính chất như khoá chính, tức là giá trị của chúng là duy nhất trong bảng. Tập một hoặc nhiều cột có giá trị duy nhất và không được chọn làm khoá chính được gọi là khoá phụ (khoá dự tuyển) của bảng. Như vậy, một bảng chỉ có nhiều nhất một khoá chính nhưng có thể có nhiều khoá phụ.

Ràng buộc UNIQUE được sử dụng trong câu lệnh CREATE TABLE để định nghĩa khoá phụ cho bảng và được khai báo theo cú pháp sau đây:

```
[CONSTRAINT tên_ràng_buộc]
 UNIQUE [(danh_sách_cột)]
```

Ví dụ 3.7: Giả sử ta cần định nghĩa bảng LOP với khoá chính là cột MALOP nhưng đồng thời lại không cho phép các lớp khác nhau được trùng tên lớp với nhau, ta sử dụng câu lệnh như sau:

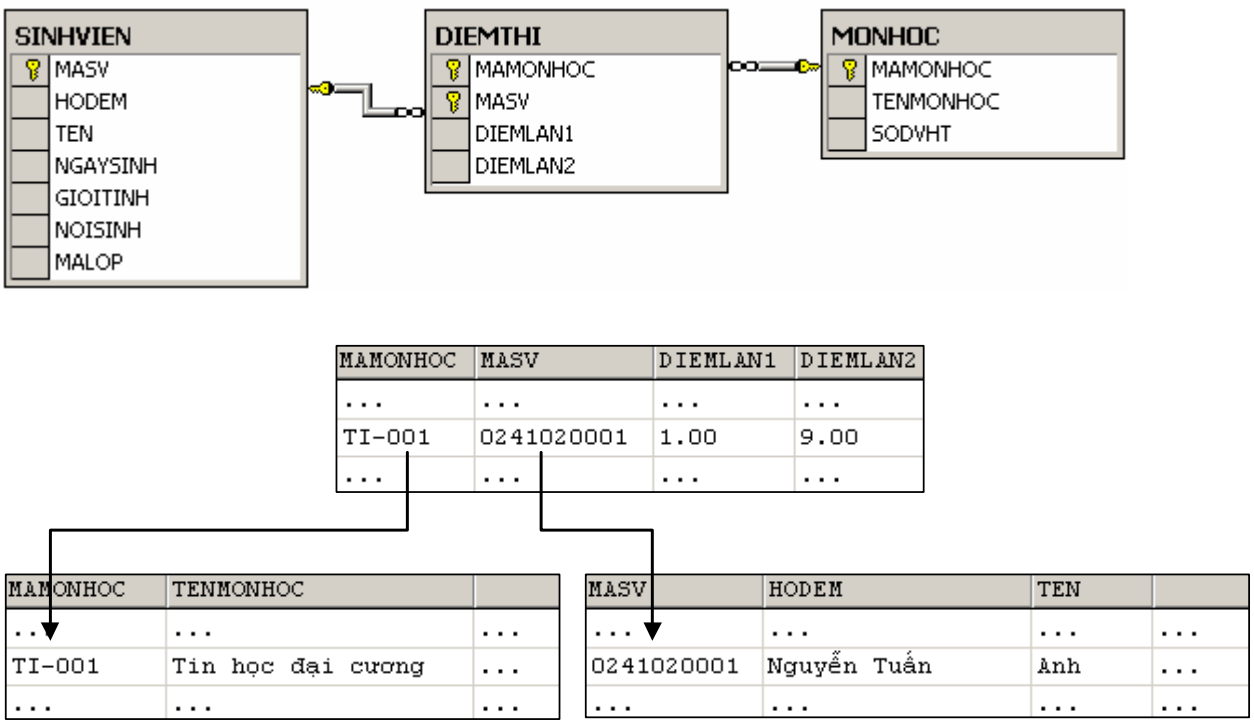
```
CREATE TABLE lop
(
malop       NVARCHAR(10)      NOT NULL,
tenlop      NVARCHAR(30)      NOT NULL,
khoa        SMALLINT          NULL,
hedaotao    NVARCHAR(25)      NULL,
namnhaphoc  INT               NULL,
makhoa      NVARCHAR(5),
CONSTRAINT pk_lop PRIMARY KEY (malop),
CONSTRAINT unique_lop_tenlop UNIQUE(tenlop)
)
```

3.1.4 Ràng buộc FOREIGN KEY

Các bảng trong một cơ sở dữ liệu có mối quan hệ với nhau. Những mối quan hệ này biểu diễn cho sự quan hệ giữa các đối tượng trong thế giới thực. Về mặt dữ liệu, những mối quan hệ được đảm bảo thông qua việc đòi hỏi sự có mặt của một giá trị dữ liệu trong bảng này phải phụ thuộc vào sự tồn tại của giá trị dữ liệu đó ở trong một bảng khác.

Ràng buộc FOREIGN KEY được sử dụng trong định nghĩa bảng dữ liệu nhằm tạo nên mối quan hệ giữa các bảng trong một cơ sở dữ liệu. Một hay một tập các cột trong một bảng được gọi là khoá ngoại, tức là có ràng buộc FOREIGN KEY, nếu giá trị của nó được xác định từ khoá chính (PRIMARY KEY) hoặc khoá phụ (UNIQUE) của một bảng dữ liệu khác.

Hình dưới đây cho ta thấy được mối quan hệ giữa 3 bảng DIEMTHI, SINHVIEN và MONHOC. Trong bảng DIEMTHI, MASV là khoá ngoài tham chiếu đến cột MASV của bảng SINHVIEN và MAMONHOC là khoá ngoài tham chiếu đến cột MAMONHOC của bảng MONHOC.



Hình 3.2 *Mối quan hệ giữa các bảng*

Với mối quan hệ được tạo ra như hình trên, hệ quản trị cơ sở dữ liệu sẽ kiểm tra tính hợp lệ của mỗi bản ghi trong bảng DIEMTHI mỗi khi được bổ sung hay cập nhật. Một bản ghi bất kỳ trong bảng DIEMTHI chỉ hợp lệ (đảm bảo ràng buộc FOREIGN KEY) nếu giá trị của cột MASV phải tồn tại trong một bản ghi nào đó của bảng SINHVIEN và giá trị của cột MAMONHOC phải tồn tại trong một bản ghi nào đó của bảng MONHOC.

Ràng buộc FOREIGN KEY được định nghĩa theo cú pháp dưới đây:

```
[CONSTRAINT tên_ràng_buộc]
FOREIGN KEY [(danh_sách_cột)]
REFERENCES tên_bảng_tham_chiếu(danh_sách_cột_tham_chiếu)
[ON DELETE CASCADE | NO ACTION | SET NULL | SET DEFAULT]
[ON UPDATE CASCADE | NO ACTION | SET NULL | SET DEFAULT]
```

Việc định nghĩa một ràng buộc FOREIGN KEY bao gồm các yếu tố sau:

- Tên cột hoặc danh sách cột của bảng được định nghĩa tham gia vào khoá ngoài.
- Tên của bảng được tham chiếu bởi khoá ngoài và danh sách các cột được tham chiếu đến trong bảng tham chiếu.
- Cách thức xử lý đối với các bản ghi trong bảng được định nghĩa trong trường hợp các bản ghi được tham chiếu trong bảng tham chiếu bị xoá (ON DELETE) hay cập nhật (ON UPDATE). SQL chuẩn đưa ra 4 cách xử lý:
    - CASCADE: Tự động xoá (cập nhật) nếu bản ghi được tham chiếu bị xoá (cập nhật).
    - NO ACTION: (Mặc định) Nếu bản ghi trong bảng tham chiếu đang được tham chiếu bởi một bản ghi bất kỳ trong bảng được định nghĩa thì bàn ghi đó không được phép xoá hoặc cập nhật (đối với cột được tham chiếu).
    - SET NULL: Cập nhật lại khoá ngoài của bản ghi thành giá trị NULL (nếu cột cho phép nhận giá trị NULL).
    - SET DEFAULT: Cập nhật lại khoá ngoài của bản ghi nhận giá trị mặc định (nếu cột có qui định giá trị mặc định).

Ví dụ 3.8: Câu lệnh dưới đây định nghĩa bảng DIEMTHI với hai khoá ngoài trên cột MASV và cột MAMONHOC (giả sử hai bảng SINHVIEN và MONHOC đã được định nghĩa)

```
CREATE TABLE diemthi
(
mamonhoc  NVARCHAR(10)   NOT NULL ,
masv      NVARCHAR(10)   NOT NULL ,
diemlan1  NUMERIC(4, 2),
diemlan2  NUMERIC(4, 2),
CONSTRAINT pk_diemthi PRIMARY KEY(mamonhoc,masv),
CONSTRAINT fk_diemthi_mamonhoc
          FOREIGN KEY(mamonhoc)
          REFERENCES monhoc(mamonhoc)
          ON DELETE CASCADE
          ON UPDATE CASCADE,
CONSTRAINT fk_diemthi_masv
          FOREIGN KEY(masv)
          REFERENCES sinhvien(masv)
          ON DELETE CASCADE
          ON UPDATE CASCADE
```

```
)
```

Lưu ý:

- Cột được tham chiếu trong bảng tham chiếu phải là khoá chính (hoặc là khoá phụ).
- Cột được tham chiếu phải có cùng kiểu dữ liệu và độ dài với cột tương ứng trong khóa ngoài.
- Bảng tham chiếu phải được định nghĩa trước. Do đó, nếu các bảng có mối quan hệ vòng, ta có thể không thể định nghĩa ràng buộc FOREIGN KEY ngay trong câu lệnh CREATE TABLE mà phải định nghĩa thông qua lệnh ALTER TABLE.

3.2 Sửa đổi định nghĩa bảng

Một bảng sau khi đã được định nghĩa bằng câu lệnh CREATE TABLE có thể được sửa đổi thông qua câu lệnh ALTER TABLE. Câu lệnh này cho phép chúng ta thực hiện được các thao tác sau:

- Bổ sung một cột vào bảng.
- Xoá một cột khỏi bảng.
- Thay đổi định nghĩa của một cột trong bảng.
- Xoá bỏ hoặc bổ sung các ràng buộc cho bảng

Cú pháp của câu lệnh ALTER TABLE như sau:

```
ALTER TABLE tên_bảng
  ADD định_nghĩa_cột |
  ALTER COLUMN tên_cột  kiểu_dữ_liệu [NULL | NOT NULL] |
  DROP COLUMN tên_cột |
  ADD CONSTRAINT tên_ràng_buộc  định_nghĩa_ràng_buộc |
  DROP CONSTRAINT tên_ràng_buộc
```

Ví dụ 3.9: Các ví dụ dưới đây minh hoạ cho ta cách sử dụng câu lệnh ALTER TABLE trong các trường hợp.

Giả sử ta có hai bảng DONVI và NHANVIEN với định nghĩa như sau:

```
CREATE TABLE donvi
(
madv      INT            NOT NULL  PRIMARY KEY,
tendv     NVARCHAR(30)   NOT NULL
```

```
)

CREATE TABLE nhanvien
(
manv      NVARCHAR(10)   NOT NULL,
hoten     NVARCHAR(30)   NOT NULL,
ngaysinh  DATETIME,
diachi    CHAR(30)       NOT NULL
)
```

Bổ sung vào bảng NHANVIEN cột DIENTHOAI với ràng buộc CHECK nhằm qui định điện thoại của nhân viên là một chuỗi 6 chữ số:

```
ALTER TABLE nhanvien
ADD
  dienthoai NVARCHAR(6)
  CONSTRAINT chk_nhanvien_dienthoai
  CHECK (dienthoai LIKE '[0-9][0-9][0-9][0-9][0-9][0-9]')
```

Bổ sung thêm cột MADV vào bảng NHANVIEN:

```
ALTER TABLE nhanvien
ADD
     madv INT NULL
```

Định nghĩa lại kiểu dữ liệu của cột DIACHI trong bảng NHANVIEN và cho phép cột này chấp nhận giá trị NULL:

```
ALTER TABLE nhanvien
ALTER COLUMN diachi NVARCHAR(100) NULL
```

Xoá cột ngày sinh khỏi bảng NHANVIEN:

```
ALTER TABLE nhanvien
DROP COLUMN ngaysinh
```

Định nghĩa khoá chính (ràng buộc PRIMARY KEY) cho bảng NHANVIEN là cột MANV:

```
ALTER TABLE nhanvien
ADD
     CONSTRAINT pk_nhanvien PRIMARY KEY(manv)
```

Định nghĩa khoá ngoài cho bảng NHANVIEN trên cột MADV tham chiếu đến cột MADV của bảng DONVI:

```
ALTER TABLE nhanvien
ADD
     CONSTRAINT fk_nhanvien_madv
     FOREIGN KEY(madv) REFERENCES donvi(madv)
```

```
        ON DELETE CASCADE
        ON UPDATE CASCADE
```

Xoá bỏ ràng buộc kiểm tra số điện thoại của nhân viên

```
ALTER TABLE nhanvien
DROP CONSTRAINT CHK_NHANVIEN_DIENTHOAI
```

Lưu ý:

- Nếu bổ sung thêm một cột vào bảng và trong bảng đã có ít nhất một bản ghi thì cột mới cần bổ sung phải cho phép chấp nhận giá trị NULL hoặc phải có giá trị mặc định.
- Muốn xoá một cột đang được ràng buộc bởi một ràng buộc hoặc đang được tham chiếu bởi một khoá ngoài, ta phải xoá ràng buộc hoặc khoá ngoài trước sao cho trên cột không còn bất kỳ một ràng buộc và không còn được tham chiếu bởi bất kỳ khoá ngoài nào.
- Nếu bổ sung thêm ràng buộc cho một bảng đã có dữ liệu và ràng buộc cần bổ sung không được thoả mãn bởi các bản ghi đã có trong bảng thì câu lệnh ALTER TABLE không thực hiện được.

3.3 Xoá bảng

Khi một bảng không còn cần thiết , ta có thể xoá nó ra khỏi cơ sở dữ liệu bằng câu lệnh DROP TABLE. Câu lệnh này cũng đồng thời xoá tất cả những ràng buộc, chỉ mục, trigger liên quan đến bảng đó.

Câu lệnh có cú pháp như sau:

```
DROP TABLE tên_bảng
```

Trong các hệ quản trị cơ sở dữ liệu, khi đã xoá một bảng bằng lệnh DROP TABLE, ta không thể khôi phục lại bảng cũng như dữ liệu của nó. Do đó, cần phải cẩn thận khi sử dụng câu lệnh này.

Câu lệnh DROP TABLE không thể thực hiện được nếu bảng cần xoá đang được tham chiếu bởi một ràng buộc FOREIGN KEY. Trong trường hợp này, ràng buộc FOREIGN KEY đang tham chiếu hoặc bảng đang tham chiếu đến bảng cần xoá phải được xoá trước.

Khi một bảng bị xoá, tất cả các ràng buộc, chỉ mục và trigger liên quan đến bảng cũng đồng thời bị xóa theo. Do đó, nếu ta tạo lại bảng thì cũng phải tạo lại các đối tượng này.

Ví dụ 3.10: Giả sử cột MADV trong bảng DONVI đang được tham chiếu bởi khoá ngoài *fk_nhanvien_madv* trong bảng NHANVIEN. Để xoá bảng DONVI ra khỏi cơ sở dữ liệu, ta thực hiện hai câu lệnh sau:

Xoá bỏ ràng buộc *fk_nhanvien_madv* khỏi bảng NHANVIEN:

```
ALTER TABLE nhanvien
DROP CONSTRAINT fk_nhanvien_madv
```

Xoá bảng DONVI:

```
DROP TABLE donvi
```

3.4 Khung nhìn

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

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

Hình 3.3 dưới đây minh hoạ cho ta thấy khung nhìn có tên DSSV được định nghĩa thông qua câu lệnh SELECT truy vấn dữ liệu trên hai bảng SINHVIEN và LOP:

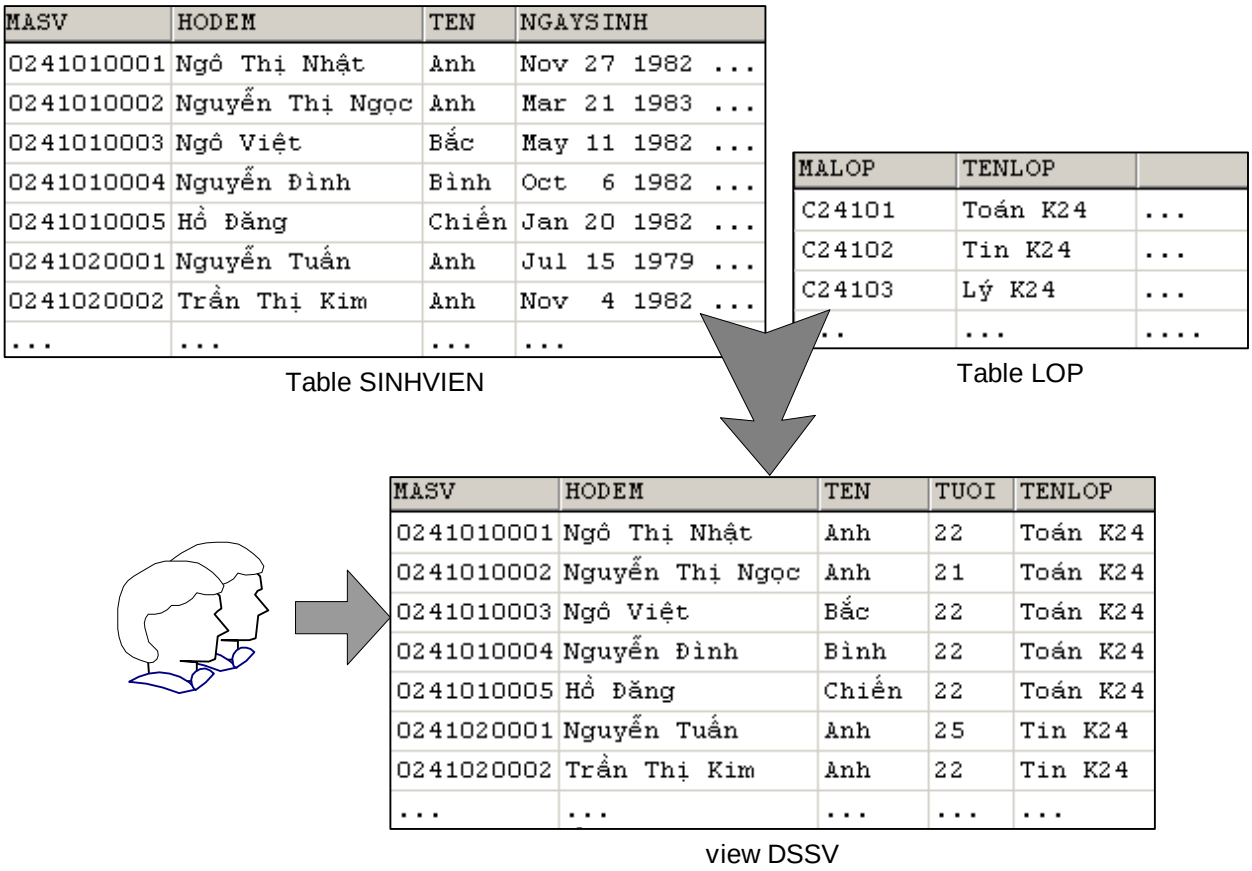
```
SELECT masv,hodem,ten,
       DATEDIFF(YY,ngaysinh,GETDATE()) AS tuoi,tenlop
FROM sinhvien,lop
WHERE sinhvien.malop=lop.malop
```

Khi khung nhìn DSSV đã được định nghĩa, ta có thể sử dụng câu lệnh SELECT để truy vấn dữ liệu từ khung nhìn như đối với các bảng. Khi trong câu truy vấn xuất hiện khung nhìn, hệ quản trị cơ sở dữ liệu sẽ dựa vào đinh nghĩa của khung nhìn để chuyển yêu cầu truy vấn dữ liệu liên quan đến khung nhìn thành yêu cầu tương tự trên các bảng cơ sở và việc truy vấn dữ liệu được thực hiện bởi yêu cầu tương đương trên các bảng.

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

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

- **Đơn giản hoá các thao tác truy vấn dữ liệu:** Một khung nhìn đóng vai trò như là một đối tượng tập hợp dữ liệu từ nhiều bảng khác nhau vào trong một "bảng". Nhờ vào đó, người sử dụng có thể thực hiện các yêu cầu truy vấn dữ liệu một cách đơn giản từ khung nhìn thay vì phải đưa ra những câu truy vấn phức tạp.
- **Tập trung và đơn giản hoà dữ liệu:** Thông qua khung nhìn ta có thể cung cấp cho người sử dụng những cấu trúc đơn giản, dễ hiểu hơn về dữ liệu trong cơ sở dữ liệu đồng thời giúp cho người sử dụng tập trung hơn trên những phần dữ liệu cần thiết.
- **Độc lập dữ liệu:** Một khung nhìn có thể cho phép người sử dụng có được cái nhìn về dữ liệu độc lập với cấu trúc của các bảng trong cơ sở dữ liệu cho dù các bảng cơ sở có bị thay đổi phần nào về cấu trúc.



***Hình 3.3** Khung nhìn DSSV với dữ liệu được lấy từ bảng SINHVIEN và LOP*

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

- Do hệ quản trị cơ sở dữ liệu thực hiện việc chuyển đổi các truy vấn trên khung nhìn thành những truy vấn trên các bảng cơ sở nên nếu một khung

nhìn được định nghĩa bởi một truy vấn phức tạp thì sẽ dẫn đến chi phí về mặt thời gian khi thực hiện truy vấn liên quan đến khung nhìn sẽ lớn.

- Mặc dù thông qua khung nhìn có thể thực hiện được thao tác bổ sung và cập nhật dữ liệu cho bảng cơ sở nhưng chỉ hạn chế đối với những khung nhìn đơn giản. Đối với những khung nhìn phức tạp thì thường không thực hiện được; hay nói cách khác là dữ liệu trong khung nhìn là chỉ đọc.

3.4.1 Tạo khung nhìn

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

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

Ví dụ 3.11: Câu lệnh dưới đây tạo khung nhìn có tên DSSV từ câu lệnh SELECT truy vấn dữ liệu từ hai bảng SINHVIEN và LOP

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

và nếu thực hiện câu lệnh:

```
SELECT * FROM dssv
```

ta có được kết quả như sau:

MASV	HODEM	TEN	TUOI	TENLOP
0241010001	Ngô Thị Nhật	Anh	22	Toán K24
0241010002	Nguyễn Thị Ngọc	Anh	21	Toán K24
0241010003	Ngô Việt	Bắc	22	Toán K24
0241010004	Nguyễn Đình	Bình	22	Toán K24
0241010005	Hồ Đăng	Chiến	22	Toán K24
0241020001	Nguyễn Tuấn	Anh	25	Tin K24
0241020002	Trần Thị Kim	Anh	22	Tin K24
0241020003	Võ Đức	Ân	22	Tin K24
0241020004	Nguyễn Công	Bình	25	Tin K24
0241020005	Nguyễn Thanh	Bình	22	Tin K24
...	...	...	...	...

Nếu trong câu lệnh CREATE VIEW, ta không chỉ định danh sách các tên cột cho khung nhìn, tên các cột trong khung nhìn sẽ chính là tiêu đề các cột trong kết quả của câu lệnh SELECT. Trong trường hợp tên các cột của khung nhìn được chỉ định, chúng phải có cùng số lượng với số lượng cột trong kết quả của câu truy vấn.

Ví dụ 3.12: Câu lệnh dưới đây tạo khung nhìn từ câu truy vấn tương tự như ví dụ trên nhưng có đặt tên cho các cột trong khung nhìn:

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

và câu lệnh:

```
SELECT * FROM dssv
```

trong trường hợp này có kết quả như sau:

MA	HO	TEN	TUOI	LOP
0241010001	Ngô Thị Nhật	Anh	22	Toán K24
0241010002	Nguyễn Thị Ngọc	Anh	21	Toán K24
0241010003	Ngô Việt	Bắc	22	Toán K24
0241010004	Nguyễn Đình	Bình	22	Toán K24
0241010005	Hồ Đăng	Chiến	22	Toán K24
0241020001	Nguyễn Tuấn	Anh	25	Tin K24
0241020002	Trần Thị Kim	Anh	22	Tin K24
0241020003	Võ Đức	Ân	22	Tin K24
0241020004	Nguyễn Công	Bình	25	Tin K24
0241020005	Nguyễn Thanh	Bình	22	Tin K24
...	...	...	...	...

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

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

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

Ví dụ 3.13: Câu lệnh dưới đây là câu lệnh sai do cột thứ 4 không xác định được tên cột

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

3.4.2 Cập nhật, bổ sung và xoá dữ liệu thông qua khung nhìn

Đối với một số khung nhìn, ta có thể tiến hành thực hiện các thao tác cập nhập, bổ sung và xoá dữ liệu. Thực chất, những thao tác này sẽ được chuyển thành những thao tác tương tự trên các bảng cơ sở và có tác động đến những bảng cơ sở.

Về mặt lý thuyết, để có thể thực hiện thao tác bổ sung, cập nhật và xoá, một khung nhìn trước tiên phải thoả mãn các điều kiện sau đây:

- Trong câu lệnh SELECT định nghĩa khung nhìn không được sử dụng từ khoá DISTINCT, TOP, GROUP BY và UNION.
- Các thành phần xuất hiện trong danh sách chọn của câu lệnh SELECT phải là các cột trong các bảng cơ sở. Trong danh sách chọn không được chứa các biểu thức tính toán, các hàm gộp.

Ngoài những điều kiện trên, các thao tác thay đổi đến dữ liệu thông qua khung nhìn còn phải đảm bảo thoả mãn các ràng buộc trên các bảng cơ sở, tức là vẫn đảm bảo tính toàn vẹn dữ liệu. Ví dụ dưới đây sẽ minh hoạ cho ta thấy việc thực hiện các thao tác bổ sung, cập nhật và xoá dữ liệu thông qua khung nhìn.

Ví dụ 3.14: Xét định nghĩa hai bảng DONVI và NHANVIEN như sau:

```
CREATE TABLE donvi
(
madv        INT             PRIMARY KEY,
tendv       NVARCHAR(30)    NOT NULL,
dienthoai   NVARCHAR(10)    NULL,
)

CREATE TABLE  nhanvien
(
```

```
manv        NVARCHAR(10)   PRIMARY KEY,
hoten       NVARCHAR(30)   NOT NULL,
ngaysinh    DATETIME       NULL,
diachi      NVARCHAR(50)   NULL,
madv        INT            FOREIGN KEY
                           REFERENCES donvi(madv)
                           ON DELETE CASCADE
                           ON UPDATE CASCADE
)
```

Giả sử trong hai bảng này đã có dữ liệu như sau:

MADV	TENDV	DIENTHOAI
1	P. Kinh doanh	822321
2	P. Tiep thi	822012

Bảng DONVI

MANV	HOTEN	NGAYSINH	DIACHI	MADV
NV01	Tran Van A	1975-02-03 00:00:00	77 Tran Phu	1
NV02	Mai Thi B	1977-05-04 00:00:00	34 Nguyen Hue	2
NV03	Nguyen Van C	NULL	NULL	2

Bảng NHANVIEN

Câu lệnh dưới đây định nghĩa khung nhìn NV1 cung cấp các thông tin về mã nhân viên, họ tên và mã đơn vị nhân viên làm việc:

```
CREATE VIEW nv1
AS
     SELECT manv,hoten,madv FROM nhanvien
```

Nếu ta thực hiện câu lệnh

```
INSERT INTO nv1 VALUES('NV04','Le Thi D',1)
```

Một bản ghi mới sẽ được bổ sung vào bảng NHANVIEN và dữ liệu trong bảng này sẽ là:

MANV	HOTEN	NGAYSINH	DIACHI	MADV
NV01	Tran Van A	1975-02-03 00:00:00	77 Tran Phu	1
NV02	Mai Thi B	1977-05-04 00:00:00	34 Nguyen Hue	2
NV03	Nguyen Van C	NULL	NULL	2
NV04	Le Thi D	NULL	NULL	1

Thông qua khung nhìn này, ta cũng có thể thực hiện thao tác cập nhật và xoá dữ liệu. Chẳng hạn, nếu ta thực hiện câu lệnh:

```
DELETE FROM nv1 WHERE manv='NV04'
```

Thì bản ghi tương ứng với nhân viên có mã NV04 sẽ bị xoá khỏi bảng NHANVIEN

Nếu trong danh sách chọn của câu lệnh SELECT có sự xuất hiện của biểu thức tính toán đơn giản, thao tác bổ sung dữ liệu thông qua khung nhìn không thể thực hiện được. Tuy nhiên, trong trường hợp này thao tác cập nhật và xoá dữ liệu vẫn có thể có khả năng thực hiện được (hiển nhiên không thể cập nhật dữ liệu đối với một cột có được từ một biểu thức tính toán).

Ví dụ 3.15: Xét khung nhìn NV2 được định nghĩa như sau:

```
CREATE VIEW nv2
AS
     SELECT manv,hoten,YEAR(ngaysinh) AS namsinh,madv
     FROM nhanvien
```

Đối với khung nhìn NV2, ta không thể thực hiện thao tác bổ sung dữ liệu nhưng có thể cập nhật hoặc xoá dữ liệu trên bảng thông qua khung nhìn này. Câu lệnh dưới đây là không thể thực hiện được trên khung nhìn NV2

```
INSERT INTO nv2(manv,hoten,madv)
VALUES('NV05','Le Van E',1)
```

Nhưng câu lệnh:

```
UPDATE nv2 SET hoten='Le Thi X' WHERE manv='NV04'
```

hoặc câu lệnh

```
DELETE FROM nv2 WHERE manv='NV04'
```

lại có thể thực hiện được và có tác động đối với dữ liệu trong bảng NHANVIEN.

Trong trường hợp khung nhìn được tạo ra từ một phép nối (trong hoặc ngoài) trên nhiều bảng, ta có thể thực hiện được thao tác bổ sung hoặc cập nhật dữ liệu nếu thao tác này chỉ có tác động đến đúng một bảng cơ sở (câu lệnh DELETE không thể thực hiện được trong trường hợp này).

Ví dụ 3.16: Với khung nhìn được định nghĩa như sau:

```
CREATE VIEW nv3
AS
     SELECT manv,hoten,ngaysinh,
            diachi,nhanvien.madv AS noilamviec,
            donvi.madv,tendv,dienthoai
```

```
FROM nhanvien FULL OUTER JOIN donvi
     ON nhanvien.madv=donvi.madv
```

Câu lệnh:

```
INSERT INTO nv3(manv,hoten,noilamviec)
VALUES('NV05','Le Van E',1)
```

sẽ bổ sung thêm vào bảng NHANVIEN một bản ghi mới. Hoặc câu lệnh:

```
INSERT INTO nv3(madv,tendv) VALUES(3,'P. Ke toan')
```

bổ sung thêm vào bảng DONVI một bản ghi do cả hai câu lệnh này chỉ có tác động đến đúng một bảng cơ sở.

Câu lệnh dưới đây không thể thực hiện được do có tác động một lúc đến hai bảng cơ sở.

```
INSERT INTO nv3(manv,hoten,noilamviec,madv,tendv)
VALUES('NV05','Le Van E',1,3,'P. Ke toan')
```

3.4.3 Sửa đổi khung nhìn

Câu lệnh ALTER VIEW được sử dụng để định nghĩa lại khung nhìn hiện có nhưng không làm thay đổi các quyền đã được cấp phát cho người sử dụng trước đó. Câu lệnh này sử dụng tương tự như câu lệnh CREATE VIEW và có cú pháp như sau:

```
ALTER VIEW tên_khung_nhìn [(danh_sách_tên_cột)]
AS
     Câu_lệnh_SELECT
```

Ví dụ 3.17: Ta định nghĩa khung nhìn như sau:

```
CREATE VIEW viewlop
AS
     SELECT malop,tenlop,tenkhoa
     FROM lop INNER JOIN khoa ON lop.makhoa=khoa.makhoa
     WHERE tenkhoa='Khoa Vật lý'
```

và có thể định nghĩa lại khung nhìn trên bằng câu lệnh:

```
ALTER VIEW view_lop
AS
     SELECT malop,tenlop,hedaotao
     FROM lop INNER JOIN khoa ON lop.makhoa=khoa.makhoa
     WHERE tenkhoa='Khoa Công nghệ thông tin'
```

3.4.4 Xoá khung nhìn

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

```
DROP VIEW tên_khung_nhìn
```

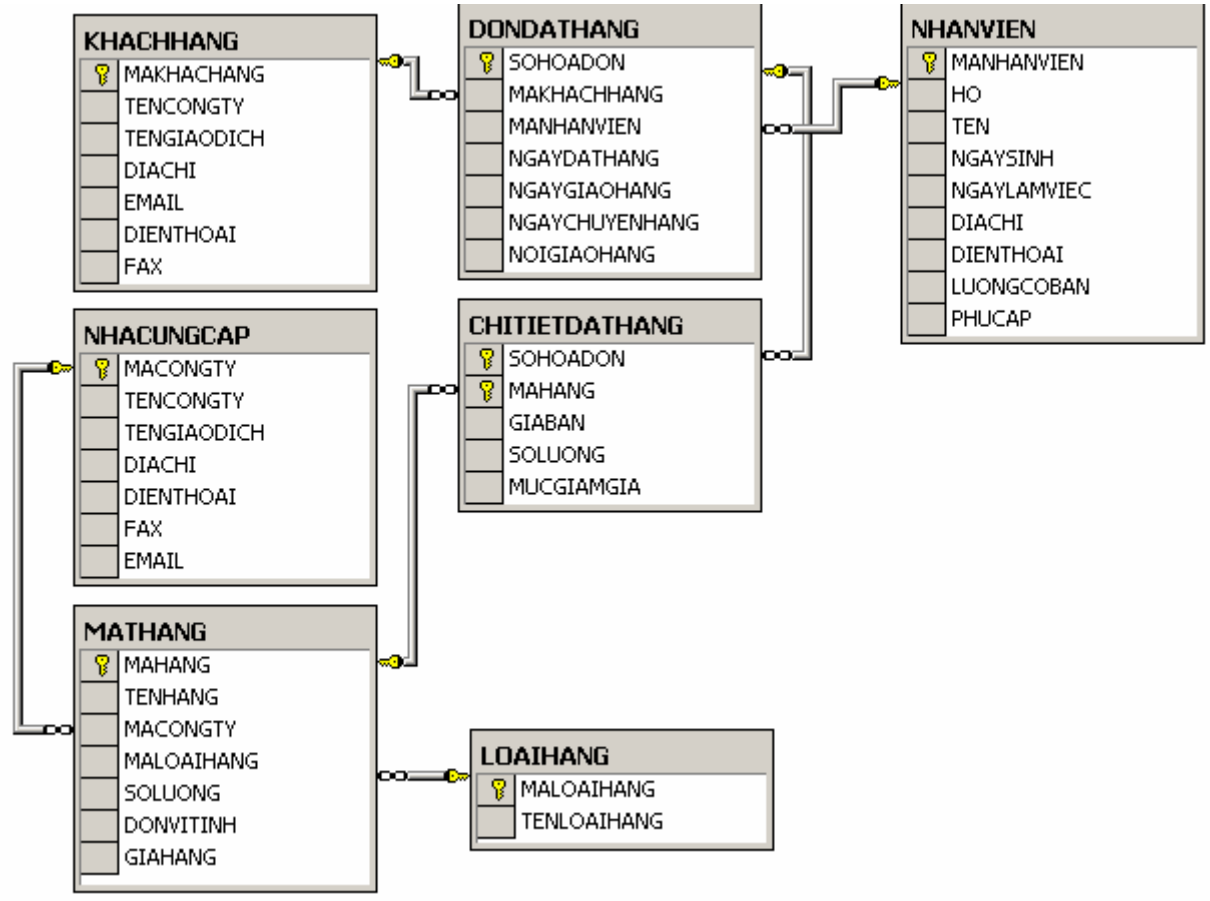
Nếu một khung nhìn bị xoá, toàn bộ những quyền đã cấp phát cho người sử dụng trên khung nhìn cũng đồng thời bị xoá. Do đó, nếu ta tạo lại khung nhìn thì phải tiến hành cấp phát lại quyền cho người sử dụng.

Ví dụ 3.18: Câu lệnh dưới đây xoá khung nhìn VIEW_LOP ra khỏi cơ sở dữ liệu

```
DROP VIEW view_lop
```

🕮 Bài tập chương 3

3.1 Sử dụng câu lệnh CREATE TABLE để tạo các bảng trong cơ sở dữ liệu như sơ đồ dưới đây (bạn tự lựa chọn kiểu dữ liệu cho phù hợp)



3.2 Bổ sung ràng buộc thiết lập giá trị mặc định bằng 1 cho cột SOLUONG và bằng 0 cho cột MUCGIAMGIA trong bảng CHITIETDATHANG

3.3 Bổ sung cho bảng DONDATHANG ràng buộc kiểm tra ngày giao hàng và ngày chuyển hàng phải sau hoặc bằng với ngày đặt hàng.

3.4 Bổ sung ràng buộc cho bảng NHANVIEN để đảm bảo rằng một nhân viên chỉ có thể làm việc trong công ty khi đủ 18 tuổi và không quá 60 tuổi.

3.5 Với các bảng đã tạo được, câu lệnh:

```
DROP TABLE nhacungcap
```

có thể thực hiện được không? Tại sao?

3.6 Cho khung nhìn được định nghĩa như sau:

```
CREATE VIEW view_donhang
AS
      SELECT dondathang.sohoadon,makhachhang,manhanvien,
             ngaydathang,ngaygiaohang,ngaychuyenhang,
             noigiaohang,mahang,
             giaban,soluong,mucgiamgia
      FROM dondathang INNER JOIN chitietdathang
      ON dondathang.sohoadon = chitietdathang.sohoadon
```

a. Có thể thông qua khung nhìn này để bổ sung dữ liệu cho bảng DONDATHANG được không?

b. Có thể thông qua khung nhìn này để bổ sung dữ liệu cho bảng CHITIETDATHANG được không?

3.7 Với khung nhìn được định nghĩa như sau:

```
CREATE VIEW view_donhang
AS
      SELECT dondathang.sohoadon,makhachhang,manhanvien,
             ngaydathang,ngaygiaohang,ngaychuyenhang,
             noigiaohang,mahang,
             giaban*soluong as thanhtien,
             mucgiamgia
      FROM dondathang INNER JOIN chitietdathang
      ON dondathang.sohoadon = chitietdathang.sohoadon
```

a. Có thể thông qua khung nhìn này để xoá hay cập nhật dữ liệu trong bảng DONDATHANG được không?

b. Có thể thông qua khung nhìn này để cập nhật dữ liệu trong bảng CHITIETDATHANG được không?

Lời giải

3.1 Tạo các bảng dữ liệu:

```
CREATE TABLE nhacungcap
(
     macongty       NVARCHAR(10)   NOT NULL
                    CONSTRAINT pk_nhacungcap
                    PRIMARY KEY(macongty),
     tencongty      NVARCHAR(40)   NOT NULL,
     tengiaodich    NVARCHAR(30)   NULL,
     diachi         NVARCHAR(60)   NULL,
     dienthoai      NVARCHAR(20)   NULL,
     fax            NVARCHAR(20)   NULL,
     email          NVARCHAR(50)   NULL
)

CREATE TABLE loaihang
(
     maloaihang     INT            NOT NULL
                    CONSTRAINT pk_loaihang
                    PRIMARY KEY(maloaihang),
     tenloaihang    NVARCHAR(15)   NOT NULL
)

CREATE TABLE mathang
(
     mahang         NVARCHAR(10)   NOT NULL
                    CONSTRAINT pk_mathang
                    PRIMARY KEY(mahang),
     tenhang        NVARCHAR(50)   NOT NULL,
     macongty       NVARCHAR(10)   NULL ,
     maloaihang     INT            NULL ,
     soluong        INT            NULL,
     donvitinh      NVARCHAR(20)   NULL ,
     giahang        MONEY          NULL
)

CREATE TABLE nhanvien
(
     manhanvien     NVARCHAR(10)   NOT NULL
```

```
                    CONSTRAINT pk_nhanvien
                    PRIMARY KEY(manhanvien),
     ho             NVARCHAR(20)   NOT NULL ,
     ten            NVARCHAR(10)   NOT NULL ,
     ngaysinh       DATETIME       NULL ,
     ngaylamviec    DATETIME       NULL ,
     diachi         NVARCHAR(50)   NULL ,
     dienthoai      NVARCHAR(15)   NULL ,
     luongcoban     MONEY          NULL ,
     phucap         MONEY          NULL
)

CREATE TABLE khachhang
(
     makhachhang    NVARCHAR(10)   NOT NULL
                    CONSTRAINT pk_khachhang
                    PRIMARY KEY(makhachhang),
     tencongty      NVARCHAR(50)   NOT NULL ,
     tengiaodich    NVARCHAR(30)   NOT NULL ,
     diachi         NVARCHAR(50)   NULL ,
     email          NVARCHAR(30)   NULL ,
     dienthoai      NVARCHAR(15)   NULL ,
     fax            NVARCHAR(15)   NULL
)

CREATE TABLE dondathang
(
     sohoadon       INT            NOT NULL
                    CONSTRAINT pk_dondathang
                    PRIMARY KEY(sohoadon),
     makhachhang    NVARCHAR(10)   NULL ,
     manhanvien     NVARCHAR(10)   NULL ,
     ngaydathang    SMALLDATETIME  NULL ,
     ngaygiaohang   SMALLDATETIME  NULL ,
     ngaychuyenhang SMALLDATETIME  NULL ,
     noigiaohang    NVARCHAR(50)   NULL
)

CREATE TABLE chitietdathang
(
```

```
    sohoadon        INT             NOT NULL ,
    mahang          NVARCHAR(10)    NOT NULL ,
    giaban          MONEY           NOT NULL ,
    soluong         SMALLINT        NOT NULL ,
    mucgiamgia      REAL            NOT NULL,
    CONSTRAINT pk_chitietdathang
    PRIMARY KEY(sohoadon,mahang)
)
```

Thiết lập mối quan hệ giữa các bảng

```
ALTER TABLE mathang
ADD
    CONSTRAINT fk_mathang_loaihang
    FOREIGN KEY (maloaihang)
    REFERENCES loaihang(maloaihang)
    ON DELETE CASCADE  ON UPDATE CASCADE ,
    CONSTRAINT fk_mathang_nhacungcap
    FOREIGN KEY (macongty)
    REFERENCES nhacungcap(macongty)
    ON DELETE CASCADE  ON UPDATE CASCADE

ALTER TABLE dondathang
ADD
    CONSTRAINT fk_dondathang_khachhang
    FOREIGN KEY (makhachhang)
    REFERENCES khachhang(makhachhang)
    ON DELETE CASCADE  ON UPDATE CASCADE ,
    CONSTRAINT fk_dondathang_nhanvien
    FOREIGN KEY (manhanvien)
    REFERENCES nhanvien(manhanvien)
    ON DELETE CASCADE  ON UPDATE CASCADE

ALTER TABLE chitietdathang
ADD
    CONSTRAINT fk_chitiet_dondathang
    FOREIGN KEY (sohoadon)
    REFERENCES dondathang(sohoadon)
    ON DELETE CASCADE  ON UPDATE CASCADE ,
    CONSTRAINT fk_chitiet_mathang
```

```
        FOREIGN KEY (mahang)
        REFERENCES mathang(mahang)
        ON DELETE CASCADE  ON UPDATE CASCADE
```

3.2
```
ALTER TABLE chitietdathang
ADD
        CONSTRAINT df_chitietdathang_soluong
        DEFAULT(1) FOR soluong,
        CONSTRAINT df_chitietdathang_mucgiamgia
        DEFAULT(0) FOR Mucgiamgia
```

3.3
```
ALTER TABLE dondathang
ADD
        CONSTRAINT chk_dondathang_ngay
        CHECK (ngaygiaohang>=ngaydathang AND
               ngaychuyenhang>=ngaydathang)
```

3.4
```
ALTER TABLE nhanvien
ADD
        CONSTRAINT chk_nhanvien_ngaylamviec
        CHECK (datediff(yy,ngaysinh,ngaylamviec)
               BETWEEN 18 AND 60)
```

3.5 Câu lệnh không thực hiện được do bảng cần xoá đang được tham chiếu bởi bảng MATHANG

3.6 a. Không. b. Không

3.7 a.Có thể cập nhật nhưng không thể xoá b. Có thể được

Chương 4

BẢO MẬT TRONG SQL

4.1 Các khái niệm

Bảo mật là một trong những yếu tố đóng vai trò quan trọng đối với sự sống còn của cơ sở dữ liệu. Hầu hết các hệ quản trị cơ sở dữ liệu thương mại hiện nay đều cung cấp khả năng bảo mật cơ sở dữ liệu với những chức năng như:

- Cấp phát quyền truy cập cơ sở dữ liệu cho người dùng và các nhóm người dùng, phát hiện và ngăn chặn những thao tác trái phép của người sử dụng trên cơ sở dữ liệu.
- Cấp phát quyền sử dụng các câu lệnh, các đối tượng cơ sở dữ liệu đối với người dùng.
- Thu hồi (huỷ bỏ) quyền của người dùng.

Bảo mật dữ liệu trong SQL được thực hiện dựa trên ba khái niệm chính sau đây:

- **Người dùng cơ sở dữ liệu (Database user):** Là đối tượng sử dụng cơ sở dữ liệu, thực thi các thao tác trên cơ sở dữ liệu như tạo bảng, truy xuất dữ liệu,... Mỗi một người dùng trong cơ sở dữ liệu được xác định thông qua tên người dùng (User ID). Một tập nhiều người dùng có thể được tổ chức trong một nhóm và được gọi là nhóm người dùng (User Group). Chính sách bảo mật cơ sở dữ liệu có thể được áp dụng cho mỗi người dùng hoặc cho các nhóm người dùng.
- **Các đối tượng cơ sở dữ liệu (Database objects):** Tập hợp các đối tượng, các cấu trúc lưu trữ được sử dụng trong cơ sở dữ liệu như bảng, khung nhìn, thủ tục, hàm được gọi là các đối tượng cơ sở dữ liệu. Đây là những đối tượng cần được bảo vệ trong chính sách bảo mật của cơ sở dữ liệu.
- **Đặc quyền (Privileges):** Là tập những thao tác được cấp phát cho người dùng trên các đối tượng cơ sở dữ liệu. Chẳng hạn một người dùng có thể truy xuất dữ liệu trên một bảng bằng câu lệnh SELECT nhưng có thể không thể thực hiện các câu lệnh INSERT, UPDATE hay DELETE trên bảng đó.

SQL cung cấp hai câu lệnh cho phép chúng ta thiết lập các chính sách bảo mật trong cơ sở dữ liệu:

- Lệnh GRANT: Sử dụng để cấp phát quyền cho người sử dụng trên các đối tượng cơ sở dữ liệu hoặc quyền sử dụng các câu lệnh SQL trong cơ sở dữ liệu.
- Lệnh REVOKE: Được sử dụng để thu hồi quyền đối với người sử dụng.

4.2 Cấp phát quyền

Câu lệnh GRANT được sử dụng để cấp phát quyền cho người dùng hay nhóm người dùng trên các đối tượng cơ sở dữ liệu. Câu lệnh này thường được sử dụng trong các trường hợp sau:

- Người sở hữu đối tượng cơ sở dữ liệu muốn cho phép người dùng khác quyền sử dụng những đối tượng mà anh ta đang sở hữu.
- Người sở hữu cơ sở dữ liệu cấp phát quyền thực thi các câu lệnh (như CREATE TABLE, CREATE VIEW,...) cho những người dùng khác.

4.2.1 Cấp phát quyền cho người dùng trên các đối tượng cơ sở dữ liệu

Chỉ có người sở hữu cơ sở dữ liệu hoặc người sở hữu đối tượng cơ sở dữ liệu mới có thể cấp phát quyền cho người dùng trên các đối tượng cơ sở dữ liệu. Câu lệnh GRANT trong trường hợp này có cú pháp như sau:

```
GRANT ALL [PRIVILEGES]| các_quyền_cấp_phát
   [(danh_sách_cột)] ON tên_bảng | tên_khung_nhìn
   |ON tên_bảng | tên_khung_nhìn [(danh_sách_cột)]
   |ON tên_thủ_tục
   |ON tên_hàm
TO danh_sách_người_dùng | nhóm_người_dùng
[WITH GRANT OPTION ]
```

Trong đó:

`ALL [PRIVILEGES]` Cấp phát tất cả các quyền cho người dùng trên đối tượng cơ sở dữ liệu được chỉ định. Các quyền có thể cấp phát cho người dùng bao gồm:

- Đối với bảng, khung nhìn, và hàm trả về dữ liệu kiểu bảng: SELECT, INSERT, DELETE, UPDATE và REFERENCES.
- Đối với cột trong bảng, khung nhìn: SELECT và UPDATE.
- Đối với thủ tục lưu trữ và hàm vô hướng:

	EXECUTE. Trong các quyền được đề cập đến ở trên, quyền REFERENCES được sử dụng nhằm cho phép tạo khóa ngoài tham chiếu đến bảng cấp phát.
các_quyền_cấp_phát	Danh sách các quyền cần cấp phát cho người dùng trên đối tượng cơ sở dữ liệu được chỉ định. Các quyền được phân cách nhau bởi dấu phẩy
tên_bảng|tên_khung_nhìn	Tên của bảng hoặc khung nhìn cần cấp phát quyền.
danh_sách_cột	Danh sách các cột của bảng hoặc khung nhìn cần cấp phát quyền.
tên_thủ_tục	Tên của thủ tục được cấp phát cho người dùng.
tên_hàm	Tên hàm (do người dùng định nghĩa) được cấp phát quyền.
danh_sách_người_dùng	Danh sách tên người dùng nhận quyền được cấp phát. Tên của các người dùng được phân cách nhau bởi dấu phẩy.
WITH GRANT OPTION	Cho phép người dùng chuyển tiếp quyền cho người dùng khác.

Các ví dụ dưới đây sẽ minh hoạ cho ta cách sử dụng câu lệnh GRANT để cấp phát quyền cho người dùng trên các đối tượng cơ sở dữ liệu.

Ví dụ 4.1: Cấp phát cho người dùng có tên *thuchanh* quyền thực thi các câu lệnh SELECT, INSERT và UPDATE trên bảng LOP

```
GRANT SELECT,INSERT,UPDATE
ON lop
TO thuchanh
```

Cho phép người dùng *thuchanh* quyền xem họ tên và ngày sinh của các sinh viên (cột HODEM,TEN và NGAYSINH của bảng SINHVIEN)

```
GRANT SELECT
(hodem,ten,ngaysinh) ON sinhvien
TO thuchanh
```

hoặc:

```
GRANT SELECT
ON sinhvien(hodem,ten,ngaysinh)
TO thuchanh
```

Với quyền được cấp phát như trên, người dùng *thuchanh* có thể thực hiện câu lệnh sau trên bảng SINHVIEN

```
SELECT hoden,ten,ngaysinh
FROM sinhvien
```

Nhưng câu lệnh dưới đây lại không thể thực hiện được

```
SELECT * FROM sinhvien
```

Trong trường hợp cần cấp phát tất cả các quyền có thể thực hiện được trên đối tượng cơ sở dữ liệu cho người dùng, thay vì liệt kê các câu lệnh, ta chỉ cần sử dụng từ khoá ALL PRIVILEGES (từ khóa PRIVILEGES có thể không cần chỉ định). Câu lệnh dưới đây cấp phát cho người dùng *thuchanh* các quyền SELECT, INSERT, UPDATE, DELETE VÀ REFERENCES trên bảng DIEMTHI

```
GRANT ALL
ON DIEMTHI
TO thuchanh
```

Khi ta cấp phát quyền nào đó cho một người dùng trên một đối tượng cơ sở dữ liệu, người dùng đó có thể thực thi câu lệnh được cho phép trên đối tượng đã cấp phát. Tuy nhiên, người dùng đó không có quyền cấp phát những quyền mà mình được phép cho những người sử dụng khác.Trong một số trường hợp, khi ta cấp phát quyền cho một người dùng nào đó, ta có thể cho phép người đó chuyển tiếp quyền cho người dùng khác bằng cách chỉ định tuỳ chọn WITH GRANT OPTION trong câu lệnh GRANT.

Ví dụ 4.2: Cho phép người dùng *thuchanh* quyền xem dữ liệu trên bảng SINHVIEN đồng thời có thể chuyển tiếp quyền này cho người dùng khác

```
GRANT SELECT
ON sinhvien
TO thuchanh
WITH GRANT OPTION
```

4.2.2 Cấp phát quyền thực thi các câu lệnh

Ngoài chức năng cấp phát quyền cho người sử dụng trên các đối tượng cơ sở dữ liệu, câu lệnh GRANT còn có thể sử dụng để cấp phát cho người sử dụng một số quyền trên hệ quản trị cơ sở dữ liệu hoặc cơ sở dữ liệu. Những quyền có thể cấp phát trong trường hợp này bao gồm:

- Tạo cơ sở dữ liệu: CREATE DATEBASE.
- Tạo bảng: CREATE RULE
- Tạo khung nhìn: CREATE VIEW

- Tạo thủ tục lưu trữ: CREATE PROCEDURE
- Tạo hàm: CREATE FUNCTION
- Sao lưu cơ sở dữ liệu: BACKUP DATABASE

Câu lệnh GRANT sử dụng trong trường hợp này có cú pháp như sau:

```
GRANT  ALL | danh_sách_câu_lênh
TO danh_sách_người_dùng
```

Ví dụ 4.3: Để cấp phát quyền tạo bảng và khung nhìn cho người dùng có tên là *thuchanh*, ta sử dụng câu lệnh như sau:

```
GRANT CREATE TABLE,CREATE VIEW
TO thuchanh
```

Với câu lệnh GRANT, ta có thể cho phép người sử dụng tạo các đối tượng cơ sở dữ liệu trong cơ sở dữ liệu. Đối tượng cơ sở dữ liệu do người dùng nào tạo ra sẽ do người đó sở hữu và do đó người này có quyền cho người dùng khác sử dụng đối tượng và cũng có thể xóa bỏ (DROP) đối tượng do mình tạo ra.

Khác với trường hợp sử dụng câu lệnh GRANT để cấp phát quyền trên đối tượng cơ sở dữ liệu, câu lệnh GRANT trong trường hợp này không thể sử dụng tuỳ chọn WITH GRANT OPTION, tức là người dùng không thể chuyển tiếp được các quyền thực thi các câu lệnh đã được cấp phát.

4.3 Thu hồi quyền

Câu lệnh REVOKE được sử dụng để thu hồi quyền đã được cấp phát cho người dùng. Tương ứng với câu lệnh GRANT, câu lệnh REVOKE được sử dụng trong hai trường hợp:

- Thu hồi quyền đã cấp phát cho người dùng trên các đối tượng cơ sở dữ liệu.
- Thu hồi quyền thực thi các câu lệnh trên cơ sở dữ liệu đã cấp phát cho người dùng.

4.3.1 Thu hồi quyền trên đối tượng cơ sở dữ liệu:

Cú pháp câu lệnh REVOKE sử dụng để thu hồi quyền đã cấp phát trên đối tượng cơ sở dữ liệu có cú pháp như sau:

```
REVOKE [GRANT OPTION FOR]
       ALL [PRIVILEGES]| các_quyền_cần_thu_hồi
[(danh_sách_cột)] ON tên_bảng | tên_khung_nhìn
|ON tên_bảng | tên_khung_nhìn [(danh_sách_cột)]
```

```
|ON tên_thủ_tục
|ON tên_hàm
FROM danh_sách_người_dùng
[CASCADE]
```

Câu lệnh REVOKE có thể sử dụng để thu hồi một số quyền đã cấp phát cho người dùng hoặc là thu hồi tất cả các quyền (ALL PRIVILEGES).

Ví dụ 4.4: Thu hồi quyền thực thi lệnh INSERT trên bảng LOP đối với người dùng *thuchanh*.

```
REVOKE INSERT
ON lop
FROM thuchanh
```

Giả sử người dùng *thuchanh* đã được cấp phát quyền xem dữ liệu trên các cột HODEM, TEN và NGAYSINH của bảng SINHVIEN, câu lệnh dưới đây sẽ thu hồi quyền đã cấp phát trên cột NGAYSINH (chỉ cho phép xem dữ liệu trên cột HODEM và TEN)

```
REVOKE SELECT
ON sinhvien(ngaysinh)
FROM thuchanh
```

Khi ta sử dụng câu lệnh REVOKE để thu hồi quyền trên một đối tượng cơ sở dữ liêu từ một người dùng náo đó, chỉ những quyền mà ta đã cấp phát trước đó mới được thu hồi, những quyền mà người dùng này được cho phép bởi những người dùng khác vẫn còn có hiệu lực. Nói cách khác, nếu hai người dùng khác nhau cấp phát cùng các quyền trên cùng một đối tượng cơ sở dữ liệu cho một người dùng khác, sau đó người thu nhất thu hồi lại quyền đã cấp phát thì những quyền mà người dùng thứ hai cấp phát vẫn có hiệu lực.

Ví dụ 4.5: Giả sử trong cơ sở dữ liệu ta có 3 người dùng là *A*, *B* và *C*. A và B đều có quyền sử dụng và cấp phát quyền trên bảng R. A thực hiện lệnh sau để cấp phát quyền xem dữ liệu trên bảng R cho C:

```
GRANT SELECT
ON R TO C
```

và B cấp phát quyền xem và bổ sung dữ liệu trên bảng R cho C bằng câu lệnh:

```
GRANT SELECT, INSERT
ON R TO C
```

Như vậy, C có quyền xem và bổ sung dữ liệu trên bảng R. Bây giờ, nếu B thực hiện lệnh:

```
REVOKE SELECT, INSERT
ON R FROM C
```

Người dùng C sẽ không còn quyền bổ sung dữ liệu trên bảng R nhưng vẫn có thể xem được dữ liệu của bảng này (quyền này do A cấp cho C và vẫn còn hiệu lực).

Nếu ta đã cấp phát quyền cho người dùng nào đó bằng câu lệnh GRANT với tuỳ chọn WITH GRANT OPTION thì khi thu hồi quyền bằng câu lệnh REVOKE phải chỉ định tuỳ chọn CASCADE. Trong trường hợp này, các quyền được chuyển tiếp cho những người dùng khác cũng đồng thời được thu hồi.

Ví dụ 4.6: Ta cấp phát cho người dùng A trên bảng R với câu lệnh GRANT như sau:

```
GRANT SELECT
ON R TO A
WITH GRANT OPTION
```

sau đó người dùng A lại cấp phát cho người dùng B quyền xem dữ liệu trên R với câu lệnh:

```
GRANT SELECT
ON R TO B
```

Nếu muốn thu hồi quyền đã cấp phát cho người dùng A, ta sử dụng câu lệnh REVOKE như sau:

```
REVOKE SELECT
ON NHANVIEN
FROM A CASCADE
```

Câu lệnh trên sẽ đồng thời thu hồi quyền mà A đã cấp cho B và như vậy cả A và B đều không thể xem được dữ liệu trên bảng R.

Trong trường hợp cần thu hồi các quyền đã được chuyển tiếp và khả năng chuyển tiếp các quyền đối với những người đã được cấp phát quyền với tuỳ chọn WITH GRANT OPTION, trong câu lệnh REVOKE ta chỉ định mệnh đề GRANT OPTION FOR.

Ví dụ 4.7: Trong ví dụ trên, nếu ta thay câu lệnh:

```
REVOKE SELECT
ON NHANVIEN
FROM A CASCADE
```

bởi câu lệnh:

```
REVOKE GRANT OPTION FOR SELECT
ON NHANVIEN
FROM A CASCADE
```

Thì B sẽ không còn quyền xem dữ liệu trên bảng R đồng thời A không thể chuyển tiếp quyền mà ta đã cấp phát cho những người dùng khác (tuy nhiên A vẫn còn quyền xem dữ liệu trên bảng R).

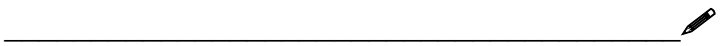
4.3.2 Thu hồi quyền thực thi các câu lênh:

Việc thu hồi quyền thực thi các câu lệnh trên cơ sở dữ liệu (CREATE DATABASE, CREATE TABLE, CREATE VIEW,...) được thực hiện đơn giản với câu lệnh REVOKE có cú pháp:

```
REVOKE ALL | các_câu_lệnh_cần_thu_hồi
FROM danh_sách_người_dùng
```

Ví dụ 4.8: Để không cho phép người dùng *thuchanh* thực hiện lệnh CREATE TABLE trên cơ sở dữ liệu, ta sử dụng câu lệnh:

```
REVOKE CREATE TABLE
FROM thuchanh
```



Chương 5

THỦ TỤC LƯU TRỮ, HÀM VÀ TRIGGER

5.1 Thủ tục lưu trữ (stored procedure)

5.1.1 Các khái niệm

Như đã đề cập ở các chương trước, SQL được thiết kế và cài đặt như là một ngôn ngữ để thực hiện các thao tác trên cơ sở dữ liệu như tạo lập các cấu trúc trong cơ sở dữ liệu, bổ sung, cập nhật, xoá và truy vấn dữ liệu trong cơ sở dữ liệu. Các câu lệnh SQL được người sử dụng viết và yêu cầu hệ quản trị cơ sở dữ liệu thực hiện theo chế độ tương tác.

Các câu lệnh SQL có thể được nhúng vào trong các ngôn ngữ lập trình, thông qua đó chuỗi các thao tác trên cơ sở dữ liệu được xác định và thực thi nhờ vào các câu lệnh, các cấu trúc điều khiển của bản thân ngôn ngữ lập trình được sử dụng.

Với thủ tục lưu trữ, một phần nào đó khả năng của ngôn ngữ lập trình được đưa vào trong ngôn ngữ SQL. Một thủ tục là một đối tượng trong cơ sở dữ liệu bao gồm một tập nhiều câu lệnh SQL được nhóm lại với nhau thành một nhóm với những khả năng sau:

- Các cấu trúc điều khiển (IF, WHILE, FOR) có thể được sử dụng trong thủ tục.
- Bên trong thủ tục lưu trữ có thể sử dụng các biến như trong ngôn ngữ lập trình nhằm lưu giữ các giá trị tính toán được, các giá trị được truy xuất được từ cơ sở dữ liệu.
- Một tập các câu lệnh SQL được kết hợp lại với nhau thành một khối lệnh bên trong một thủ tục. Một thủ tục có thể nhận các tham số truyền vào cũng như có thể trả về các giá trị thông qua các tham số (như trong các ngôn ngữ lập trình). Khi một thủ tục lưu trữ đã được định nghĩa, nó có thể được gọi thông qua tên thủ tục, nhận các tham số truyền vào, thực thi các câu lệnh SQL bên trong thủ tục và có thể trả về các giá trị sau khi thực hiện xong.

Sử dụng các thủ tục lưu trữ trong cơ sở dữ liệu sẽ giúp tăng hiệu năng của cơ sở dữ liệu, mang lại các lợi ích sau:

- Đơn giản hoá các thao tác trên cơ sở dữ liệu nhờ vào khả năng module hoá các thao tác này.

- Thủ tục lưu trữ được phân tích, tối ưu khi tạo ra nên việc thực thi chúng nhanh hơn nhiều so với việc phải thực hiện một tập rời rạc các câu lệnh SQL tương đương theo 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 và trên các đối tượng cơ sở dữ liệu, 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.

5.1.2 Tạo thủ tục lưu trữ

Thủ tục lưu trữ được tạo bởi câu lệnh CREATE PROCEDURE với cú pháp như sau:

```
CREATE PROCEDURE tên_thủ_tục [(danh_sách_tham_số)]
[WITH RECOMPILE|ENCRYPTION|RECOMPILE,ENCRYPTION]
AS
     Các_câu_lệnh_của_thủ_tục
```

Trong đó:

`tên_thủ_tục` — Tên của thủ tục cần tạo. Tên phải tuân theo qui tắc định danh và không được vượt quá 128 ký tự.

`danh_sách_tham_số` — Các tham số của thủ tục được khai báo ngay sau tên thủ tục và nếu thủ tục có nhiều tham số thì các khai báo phân cách nhau bởi dấu phẩy. Khai báo của mỗi một tham số tối thiểu phải bao gồm hai phần:

- tên tham số được bắt đầu bởi dấu @.
- kiểu dữ liệu của tham số

Ví dụ:

```
@mamonhoc  nvarchar(10)
```

`RECOMPILE` — Thông thường, thủ tục sẽ được phân tích, tối ưu và dịch sẵn ở lần gọi đầu tiên. Nếu tuỳ chọn WITH RECOMPILE được chỉ định, thủ tục sẽ được dịch lại mỗi khi được gọi.

ENCRYPTION	Thủ tục sẽ được mã hoá nếu tuỳ chọn WITH ENCRYPTION được chỉ định. Nếu thủ tục đã được mã hoá, ta không thể xem được nội dung của thủ tục.
các_câu_lệnh_của_thủ_tục	Tập hợp các câu lệnh sử dụng trong nội dung thủ tục. Các câu lệnh này có thể đặt trong cặp từ khoá BEGIN...END hoặc có thể không.

Ví dụ 5.1: Giả sử ta cần thực hiện một chuỗi các thao tác như sau trên cơ sở dữ liệu

1. Bổ sung thêm môn học *cơ sở dữ liệu* có mã *TI-005* và số đơn vị học trình là *5* vào bảng MONHOC
2. Lên danh sách nhập điểm thi môn *cơ sở dữ liệu* cho các sinh viên học lớp có mã *C24102* (tức là bổ sung thêm vào bảng DIEMTHI các bản ghi với cột MAMONHOC nhận giá trị *TI-005,* cột MASV nhận giá trị lần lượt là mã các sinh viên học lớp có mã *C24105* và các cột điểm là NULL).

Nếu thực hiện yêu cầu trên thông qua các câu lệnh SQL như thông thường, ta phải thực thi hai câu lệnh như sau:

```
INSERT INTO MONHOC
VALUES('TI-005','Cơ sở dữ liệu',5)

INSERT INTO DIEMTHI(MAMONHOC,MASV)
       SELECT 'TI-005',MASV
       FROM SINHVIEN
       WHERE MALOP='C24102'
```

Thay vì phải sử dụng hai câu lệnh như trên, ta có thể định nghĩa môt thủ tục lưu trữ với các tham số vào là *@mamonhoc*, *@tenmonhoc*, *@sodvht* và *@malop* như sau:

```
CREATE PROC sp_LenDanhSachDiem(
                          @mamonhoc      NVARCHAR(10),
                          @tenmonhoc     NVARCHAR(50),
                          @sodvht        SMALLINT,
                          @malop         NVARCHAR(10))
AS
     BEGIN
          INSERT INTO monhoc
          VALUES(@mamonhoc,@tenmonhoc,@sodvht)

          INSERT INTO diemthi(mamonhoc,masv)
```

```
                SELECT @mamonhoc,masv
                FROM sinhvien
                WHERE malop=@malop
        END
```

Khi thủ tục trên đã được tạo ra, ta có thể thực hiện được hai yêu cầu đặt ra ở trên một cách đơn giản thông qua lời gọi thủ tục:

```
sp_LenDanhSachDiem 'TI-005','Cơ sở dữ liệu',5,'C24102'
```

5.1.3 Lời gọi thủ tục lưu trữ

Như đã thấy ở ví dụ ở trên, khi một thủ tục lưu trữ đã được tạo ra, ta có thể yêu cầu hệ quản trị cơ sở dữ liệu thực thi thủ tục bằng lời gọi thủ tục có dạng:

```
tên_thủ_tục  [danh_sách_các_đối_số]
```

Số lượng các đối số cũng như thứ tự của chúng phải phù hợp với số lượng và thứ tự của các tham số khi định nghĩa thủ tục.

Trong trường hợp lời gọi thủ tục được thực hiện bên trong một thủ tục khác, bên trong một trigger hay kết hợp với các câu lệnh SQL khác, ta sử dụng cú pháp như sau:

```
EXECUTE tên_thủ_tục  [danh_sách_các_đối_số]
```

Thứ tự của các đối số được truyền cho thủ tục có thể không cần phải tuân theo thứ tự của các tham số như khi định nghĩa thủ tục nếu tất cả các đối số được viết dưới dạng:

```
@tên_tham_số  = giá_trị
```

Ví dụ 5.2: Lời gọi thủ tục ở ví dụ trên có thể viết như sau:

```
sp_LenDanhSachDiem  @malop='C24102',
                    @tenmonhoc='Cơ sở dữ liệu',
                    @mamonhoc='TI-005',
                    @sodvht=5
```

5.1.4 Sử dụng biến trong thủ tục

Ngoài những tham số được truyền cho thủ tục, bên trong thủ tục còn có thể sử dụng các biến nhằm lưu giữ các giá trị tính toán được hoặc truy xuất được từ cơ sở dữ liệu. Các biến trong thủ tục được khai báo bằng từ khoá DECLARE theo cú pháp như sau:

```
DECLARE  @tên_biến  kiểu_dữ_liệu
```

Tên biến phải bắt đầu bởi ký tự @ và tuân theo qui tắc về định danh. Ví dụ dưới đây minh hoạ việc sử dụng biến trong thủ tục

Ví dụ 5.3: Trong định nghĩa của thủ tục dưới đây sử dung các biến chứa các giá trị truy xuất được từ cơ sở dữ liệu.

```
CREATE PROCEDURE sp_Vidu(
          @malop1 NVARCHAR(10),
          @malop2 NVARCHAR(10))
AS
     DECLARE @tenlop1 NVARCHAR(30)
     DECLARE @namnhaphoc1 INT
     DECLARE @tenlop2 NVARCHAR(30)
     DECLARE @namnhaphoc2 INT

     SELECT @tenlop1=tenlop,
            @namnhaphoc1=namnhaphoc
     FROM lop WHERE malop=@malop1

     SELECT @tenlop2=tenlop,
            @namnhaphoc2=namnhaphoc
     FROM lop WHERE malop=@malop2

     PRINT @tenlop1+' nhap hoc nam '+str(@namnhaphoc1)
     print @tenlop2+' nhap hoc nam '+str(@namnhaphoc2)

     IF @namnhaphoc1=@namnhaphoc2
          PRINT 'Hai lớp nhập học cùng năm'
     ELSE
          PRINT 'Hai lớp nhập học khác năm'
```

5.1.5 Giá trị trả về của tham số trong thủ tục lưu trữ

Trong các ví dụ trước, nếu đối số truyền cho thủ tục khi có lời gọi đến thủ tục là biến, những thay đổi giá trị của biền trong thủ tục sẽ không được giữ lại khi kết thúc quá trình thực hiện thủ tục.

Ví dụ 5.4: Xét câu lệnh sau đây

```
CREATE PROCEDURE sp_Conghaiso(@a  INT,@b INT, @c INT)
AS
     SELECT @c=@a+@b
```

Nếu sau khi đã tạo thủ tục với câu lệnh trên, ta thực thi một tập các câu lệnh như sau:

```
DECLARE @tong INT
```

```
SELECT @tong=0
EXECUTE sp_Conghaiso 100,200,@tong
SELECT @tong
```

Câu lệnh "`SELECT @tong`" cuối cùng trong loạt các câu lệnh trên sẽ cho kết quả là: 0

Trong trường hợp cần phải giữ lại giá trị của đối số sau khi kết thúc thủ tục, ta phải khai báo tham số của thủ tục theo cú pháp như sau:

```
@tên_tham_số  kiểu_dữ_liệu  OUTPUT
```

hoặc:

```
@tên_tham_số  kiểu_dữ_liệu  OUT
```

và trong lời gọi thủ tục, sau đối số được truyền cho thủ tục, ta cũng phải chỉ định thêm từ khoá OUTPUT (hoặc OUT)

Ví dụ 5.5: Ta định nghĩa lại thủ tục ở ví dụ 5.4 như sau:

```
CREATE PROCEDURE sp_Conghaiso(
                                @a   INT,
                                @b   INT,
                                @c   INT OUTPUT)
AS
     SELECT @c=@a+@b
```

và thực hiện lời gọi thủ tục trong một tập các câu lệnh như sau:

```
DECLARE @tong INT
SELECT @tong=0
EXECUTE sp_Conghaiso 100,200,@tong OUTPUT
SELECT @tong
```

thì câu lệnh "`SELECT @tong`" sẽ cho kết quả là: 300

5.1.6 Tham số với giá trị mặc định

Các tham số được khai báo trong thủ tục có thể nhận các giá trị mặc định. Giá trị mặc định sẽ được gán cho tham số trong trường hợp không truyền đối số cho tham số khi có lời gọi đến thủ tục.

Tham số với giá trị mặc định được khai báo theo cú pháp như sau:

```
@tên_tham_số  kiểu_dữ_liệu  =  giá_trị_mặc_định
```

Ví dụ 5.6: Trong câu lệnh dưới đây:

```
CREATE PROC sp_TestDefault(
```

```
                    @tenlop NVARCHAR(30)=NULL,
                    @noisinh NVARCHAR(100)='Huế')
AS
     BEGIN
          IF @tenlop IS NULL
               SELECT hodem,ten
               FROM sinhvien INNER JOIN lop
                    ON sinhvien.malop=lop.malop
               WHERE noisinh=@noisinh
          ELSE
               SELECT hodem,ten
               FROM sinhvien INNER JOIN lop
                    ON sinhvien.malop=lop.malop
               WHERE noisinh=@noisinh AND
                     tenlop=@tenlop
     END
```

thủ tục *sp_TestDefault* được định nghĩa với tham số *@tenlop* có giá trị mặc định là *NULL* và tham số *@noisinh* có giá trị mặc định là *Huế*. Với thủ tục được định nghĩa như trên, ta có thể thực hiện các lời gọi với các mục đích khác nhau như sau:

- Cho biết họ tên của các sinh viên sinh tại *Huế*:
  ```
  sp_testdefault
  ```
- Cho biết họ tên của các sinh viên lớp *Tin K24* sinh tại *Huế*:
  ```
  sp_testdefault @tenlop='Tin K24'
  ```
- Cho biết họ tên của các sinh viên sinh tại *Nghệ An*:
  ```
  sp_testDefault @noisinh=N'Nghệ An'
  ```
- Cho biết họ tên của các sinh viên lớp *Tin K26* sinh tại *Đà Nẵng*:
  ```
  sp_testdefault @tenlop='Tin K26',@noisinh='Đà Nẵng'
  ```

5.1.7 Sửa đổi thủ tục

Khi một thủ tục đã được tạo ra, ta có thể tiến hành định nghĩa lại thủ tục đó bằng câu lệnh ALTER PROCEDURE có cú pháp như sau:

```
ALTER PROCEDURE tên_thủ_tục [(danh_sách_tham_số)]
[WITH RECOMPILE|ENCRYPTION|RECOMPILE,ENCRYPTION]
AS
     Các_câu_lệnh_Của_thủ_tục
```

Câu lệnh này sử dụng tương tự như câu lệnh CREATE PROCEDURE. Việc sửa đổi lại một thủ tục đã có không làm thay đổi đến các quyền đã cấp phát trên thủ tục cũng như không tác động đến các thủ tục khác hay trigger phụ thuộc vào thủ tục này.

5.1.8 Xoá thủ tục

Để xoá một thủ tục đã có, ta sử dụng câu lệnh DROP PROCEDURE với cú pháp như sau:

```
DROP PROCEDURE tên_thủ_tục
```

Khi xoá một thủ tục, tất cả các quyền đã cấp cho người sử dụng trên thủ tục đó cũng đồng thời bị xoá bỏ. Do đó, nếu tạo lại thủ tục, ta phải tiến hành cấp phát lại các quyền trên thủ tục đó.

5.2 Hàm do người dùng định nghĩa

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

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

5.2.1 Định nghĩa và sử dụng hàm

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

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

Ví dụ 5.7: Câu lệnh dưới đây định nghĩa hàm tính ngày trong tuần (thứ trong tuần) của một giá trị kiểu ngày

```
CREATE FUNCTION thu(@ngay DATETIME)
RETURNS NVARCHAR(10)
AS
     BEGIN
```

```
        DECLARE @st NVARCHAR(10)
        SELECT @st=CASE DATEPART(DW,@ngay)
                        WHEN 1 THEN 'Chu nhật'
                        WHEN 2 THEN 'Thứ hai'
                        WHEN 3 THEN 'Thứ ba'
                        WHEN 4 THEN 'Thứ tư'
                        WHEN 5 THEN 'Thứ năm'
                        WHEN 6 THEN 'Thứ sáu'
                        ELSE 'Thứ bảy'
                    END
        RETURN (@st)  /* Trị trả về của hàm */
END
```

Một hàm khi đã được định nghĩa có thể được sử dụng như các hàm do hệ quản trị cơ sở dữ liệu cung cấp (thông thường trước tên hàm ta phải chỉ định thêm tên của người sở hữu hàm)

Ví dụ 5.8: Câu lệnh SELECT dưới đây sử dụng hàm đã được định nghĩa ở ví dụ trước:

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

có kết quả là:

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

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

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

Ví dụ 5.9: Xét khung nhìn được định nghĩa như sau:

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

với khung nhìn trên, thông qua câu lệnh:

```
SELECT * FROM sinhvien_K25
```

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

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

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

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

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

- Kiểu trả về của hàm phải được chỉ định bởi mệnh đề RETURNS TABLE.
- Trong phần thân của hàm chỉ có duy nhất một câu lệnh RETURN xác định giá trị trả về của hàm thông qua duy nhất một câu lệnh SELECT. Ngoài ra, không sử dụng bất kỳ câu lệnh nào khác trong phần thân của hàm.

Ví dụ 5.10: Ta định nghĩa hàm *func_XemSV* như sau:

```
CREATE FUNCTION func_XemSV(@khoa SMALLINT)
RETURNS TABLE
AS
```

```
RETURN(SELECT masv,hodem,ten,ngaysinh
       FROM sinhvien INNER JOIN lop
            ON sinhvien.malop=lop.malop
       WHERE khoa=@khoa)
```

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

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

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

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

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

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

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

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

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

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

Ví dụ 5.11: Ta định nghĩa hàm *func_TongSV* như sau:

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

Với hàm được định nghĩa như trên, câu lệnh:

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

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

Hàm trên cân master

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

Còn câu lệnh:

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

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

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

5.3 Trigger

Trong chương 4, ta đã biết các ràng buộc được sử dụng để đảm bảo tính toàn vẹn dữ liệu trong cơ sở dữ liệu. Một đối tượng khác cũng thường được sử dụng trong các cơ sở dữ liệu cũng với mục đích này là các trigger. Cũng tương tự như thủ tục lưu trữ, một trigger là một đối tượng chứa một tập các câu lệnh SQL và tập các câu lệnh này sẽ được thực thi khi trigger được gọi. Điểm khác biệt giữa thủ tục lưu trữ và trigger là: các thủ tục lưu trữ được thực thi khi người sử dụng có lời gọi đến chúng còn các trigger lại được "gọi" tự động khi xảy ra những giao tác làm thay đổi dữ liệu trong các bảng.

Mỗi một trigger được tạo ra và gắn liền với một bảng nào đó trong cơ sở dữ liệu. Khi dữ liệu trong bảng bị thay đổi (tức là khi bảng chịu tác động của các câu lệnh INSERT, UPDATE hay DELETE) thì trigger sẽ được tự động kích hoạt. 'ràng buộc'

Sử dụng trigger một cách hợp lý trong cơ sở dữ liệu sẽ có tác động rất lớn trong việc tăng hiệu năng của cơ sở dữ liệu. Các trigger thực sự hữu dụng với những khả năng sau:

- Một trigger có thể nhận biết, ngăn chặn và huỷ bỏ được những thao tác làm thay đổi trái phép dữ liệu trong cơ sở dữ liệu.
- Các thao tác trên dữ liệu (xoá, cập nhật và bổ sung) có thể được trigger phát hiện ra và tự động thực hiện một loạt các thao tác khác trên cơ sở dữ liệu nhằm đảm bảo tính hợp lệ của dữ liệu.

- Thông qua trigger, ta có thể tạo và kiểm tra được những mối quan hệ phức tạp hơn giữa các bảng trong cơ sở dữ liệu mà bản thân các ràng buộc không thể thực hiện được.

5.3.1 Định nghĩa trigger

Một trigger là một đối tượng gắn liền với một bảng và được tự động kích hoạt khi xảy ra những giao tác làm thay đổi dữ liệu trong bảng. Định nghĩa một trigger bao gồm các yếu tố sau:

- Trigger sẽ được áp dụng đối với bảng nào?
- Trigger được kích hoạt khi câu lệnh nào được thực thi trên bảng: INSERT, UPDATE, DELETE?
- Trigger sẽ làm gì khi được kích hoạt?

Câu lệnh CREATE TRIGGER được sử dụng để đinh nghĩa trigger và có cú pháp như sau:

```
CREATE TRIGGER tên_trigger
ON tên_bảng
FOR {[INSERT][,][UPDATE][,][DELETE]}
AS
     [IF UPDATE(tên_cột)
       [AND UPDATE(tên_cột)|OR UPDATE(tên_cột)]
     ...]
     các_câu_lệnh_của_trigger
```

Ví dụ 5.12: Ta định nghĩa các bảng như sau:

Bảng MATHANG lưu trữ dữ liệu về các mặt hàng:

```
CREATE TABLE mathang
(
mahang    NVARCHAR(5)    PRIMARY KEY,   /*mã hàng*/
tenhang   NVARCHAR(50)   NOT NULL,      /*tên hàng*/
soluong   INT,      /*số lượng hàng hiện có*/
)
```

Bảng NHATKYBANHANG lưu trữ thông tin về các lần bán hàng

```
CREATE TABLE   nhatkybanhang
(
stt       INT  IDENTITY  PRIMARY KEY,
ngay      DATETIME,      /*ngày bán hàng*/
```

```
nguoimua  NVARCHAR(30),  /*tên người mua hàng*/
mahang    NVARCHAR(5)    /*mã mặt hàng được bán*/
          FOREIGN KEY REFERENCES mathang(mahang),
soluong   INT,           /*giá bán hàng*/
giaban    MONEY          /*số lượng hàng được bán*/
)
```

Câu lệnh dưới đây định nghĩa trigger *trg_nhatkybanhang_insert*. Trigger này có chức năng tự động giảm số lượng hàng hiện có khi một mặt hàng nào đó được bán (tức là khi câu lệnh INSERT được thực thi trên bảng NHATKYBANHANG).

```
CREATE TRIGGER trg_nhatkybanhang_insert
ON nhatkybanhang
FOR INSERT
AS
  UPDATE mathang
  SET mathang.soluong=mathang.soluong-inserted.soluong
  FROM mathang INNER JOIN inserted
              ON mathang.mahang=inserted.mahang
```

Với trigger vừa tạo ở trên, nếu dữ liệu trong bảng MATHANG là:

thì sau khi ta thực hiện câu lênh:

```
INSERT INTO nhatkybanhang
          (ngay,nguoimua,mahang,soluong,giaban)
VALUES('5/5/2004','Tran Ngoc Thanh','H1',10,5200)
```

dữ liệu trong bảng MATHANG sẽ như sau:

MAHANG	TENHANG	SOLUONG
H1	Xà phòng	20
H2	Kem đánh răng	45

Trong câu lệnh CREATE TRIGGER ở ví dụ trên, sau mệnh đề ON là tên của bảng mà trigger cần tạo sẽ tác động đến. Mệnh đề tiếp theo chỉ định câu lệnh sẽ kích hoạt trigger (FOR INSERT). Ngoài INSERT, ta còn có thể chỉ định UPDATE hoặc DELETE cho mệnh đề này, hoặc có thể kết hợp chúng lại với nhau. Phần thân của

MAHANG	TENHANG	SOLUONG
H1	Xà phòng	30
H2	Kem đánh răng	45

trigger nằm sau từ khoá AS bao gồm các câu lệnh mà trigger sẽ thực thi khi được kích hoạt.

Chuẩn SQL định nghĩa hai bảng logic INSERTED và DELETED để sử dụng trong các trigger. Cấu trúc của hai bảng này tương tự như cấu trúc của bảng mà trigger tác động. Dữ liệu trong hai bảng này tuỳ thuộc vào câu lệnh tác động lên bảng làm kích hoạt trigger; cụ thể trong các trường hợp sau:

- Khi câu lệnh DELETE được thực thi trên bảng, các dòng dữ liệu bị xoá sẽ được sao chép vào trong bảng DELETED. Bảng INSERTED trong trường hợp này không có dữ liệu.
- Dữ liệu trong bảng INSERTED sẽ là dòng dữ liệu được bổ sung vào bảng gây nên sự kích hoạt đối với trigger bằng câu lệnh INSERT. Bảng DELETED trong trường hợp này không có dữ liệu.
- Khi câu lệnh UPDATE được thực thi trên bảng, các dòng dữ liệu cũ chịu sự tác động của câu lệnh sẽ được sao chép vào bảng DELETED, còn trong bảng INSERTED sẽ là các dòng sau khi đã được cập nhật.

5.3.2 Sử dụng mệnh đề IF UPDATE trong trigger

Thay vì chỉ định một trigger được kích hoạt trên một bảng, ta có thể chỉ định trigger được kích hoạt và thực hiện những thao tác cụ thể khi việc thay đổi dữ liệu chỉ liên quan đến một số cột nhất định nào đó của cột. Trong trường hợp này, ta sử dụng mệnh đề IF UPDATE trong trigger. IF UPDATE không sử dụng được đối với câu lệnh DELETE.

Ví dụ 5.13: Xét lại ví dụ với hai bảng MATHANG và NHATKYBANHANG, trigger dưới đây được kích hoạt khi ta tiến hành cập nhật cột SOLUONG cho một bản ghi của bảng NHATKYBANHANG (lưu ý là chỉ cập nhật đúng một bản ghi)

```
CREATE TRIGGER trg_nhatkybanhang_update_soluong
ON nhatkybanhang
FOR UPDATE
AS
IF UPDATE(soluong)
  UPDATE mathang
  SET mathang.soluong = mathang.soluong –
       (inserted.soluong-deleted.soluong)
  FROM (deleted INNER JOIN inserted ON
        deleted.stt = inserted.stt) INNER JOIN mathang
        ON mathang.mahang = deleted.mahang
```

Với trigger ở ví dụ trên, câu lệnh:

```
UPDATE nhatkybanhang
SET soluong=soluong+20
WHERE stt=1
```

sẽ kích hoạt trigger ứng với mệnh đề IF UPDATE (soluong) và câu lệnh UPDATE trong trigger sẽ được thực thi. Tuy nhiên câu lệnh:

```
UPDATE nhatkybanhang
SET nguoimua='Mai Hữu Toàn'
WHERE stt=3
```

lại không kích hoạt trigger này.

Mệnh đề IF UPDATE có thể xuất hiện nhiều lần trong phần thân của trigger. Khi đó, mệnh đề IF UPDATE nào đúng thì phần câu lệnh của mệnh đề đó sẽ được thực thi khi trigger được kích hoạt.

Ví dụ 5.14: Giả sử ta định nghĩa bảng R như sau:

```
CREATE TABLE R
(
     A    INT,
     B    INT,
     C    INT
)
```

và trigger *trg_R_update* cho bảng R:

```
CREATE TRIGGER trg_R_test
ON R
FOR UPDATE
AS
     IF UPDATE(A)
          Print 'A updated'
     IF UPDATE(C)
          Print 'C updated'
```

Câu lệnh:

```
UPDATE R SET A=100 WHERE A=1
```

sẽ kích hoạt trigger và cho kết quả là:

```
A updated
```

và câu lệnh:

```
UPDATE R SET C=100 WHERE C=2
```

cũng kích hoạt trigger và cho kết quả là:

```
C updated
```

còn câu lệnh:

```
UPDATE R SET B=100 WHERE B=3
```

hiển nhiên sẽ không kích hoạt trigger

5.3.3 ROLLBACK TRANSACTION và trigger

Một trigger có khả năng nhận biết được sự thay đổi về mặt dữ liệu trên bảng dữ liệu, từ đó có thể phát hiện và huỷ bỏ những thao tác không đảm bảo tính toàn vẹn dữ liệu. Trong một trigger, để huỷ bỏ tác dụng của câu lệnh làm kích hoạt trigger, ta sử dụng câu lệnh[(1)]:

```
ROLLBACK TRANSACTION
```

Ví dụ 5.15: Nếu trên bảng MATHANG, ta tạo một trigger như sau:

```
CREATE TRIGGER trg_mathang_delete
ON mathang
FOR DELETE
AS
     ROLLBACK TRANSACTION
```

Thì câu lệnh DELETE sẽ không thể có tác dụng đối với bảng MATHANG. Hay nói cách khác, ta không thể xoá được dữ liệu trong bảng.

Ví dụ 5.16: Trigger dưới đây được kích hoạt khi câu lệnh INSERT được sử dụng để bổ sung một bản ghi mới cho bảng NHATKYBANHANG. Trong trigger này kiểm tra điều kiện hợp lệ của dữ liệu là số lượng hàng bán ra phải nhỏ hơn hoặc bằng số lượng hàng hiện có. Nếu điều kiện này không thoả mãn thì huỷ bỏ thao tác bổ sung dữ liệu.

```
CREATE TRIGGER trg_nhatkybanhang_insert
ON NHATKYBANHANG
FOR INSERT
AS
   DECLARE @sl_co int  /* Số lượng hàng hiện có */
   DECLARE @sl_ban int /* Số lượng hàng được bán */
   DECLARE @mahang nvarchar(5) /* Mã hàng được bán */

   SELECT @mahang=mahang,@sl_ban=soluong
   FROM inserted

   SELECT @sl_co = soluong
   FROM mathang where mahang=@mahang

   /*Nếu số lượng hàng hiện có nhỏ hơn số lượng bán
     thì huỷ bỏ thao tác bổ sung dữ liệu */
```

(1) Cách sử dụng và ý nghĩa của câu lệnh ROLLBACK TRANSACTION được bàn luận chi tiết ở chương 6.

```
IF @sl_co<@sl_ban
        ROLLBACK TRANSACTION
/* Nếu dữ liệu hợp lệ
  thì giảm số lượng hàng hiện có */
ELSE
          UPDATE mathang
          SET soluong=soluong-@sl_ban
          WHERE mahang=@mahang
```

5.3.4 Sử dụng trigger trong trường hợp câu lệnh INSERT, UPDATE và DELETE có tác động đến nhiều dòng dữ liệu

Trong các ví dụ trước, các trigger chỉ thực sự hoạt động đúng mục đích khi các câu lệnh kích hoạt trigger chỉ có tác dụng đối với đúng một dòng dữ liêu. Ta có thể nhận thấy là câu lệnh UPDATE và DELETE thường có tác dụng trên nhiều dòng, câu lệnh INSERT mặc dù ít rơi vào trường hợp này nhưng không phải là không gặp; đó là khi ta sử dụng câu lệnh có dạng INSERT INTO ... SELECT ... Vậy làm thế nào để trigger hoạt động đúng trong trường hợp những câu lệnh có tác động lên nhiều dòng dữ liệu?

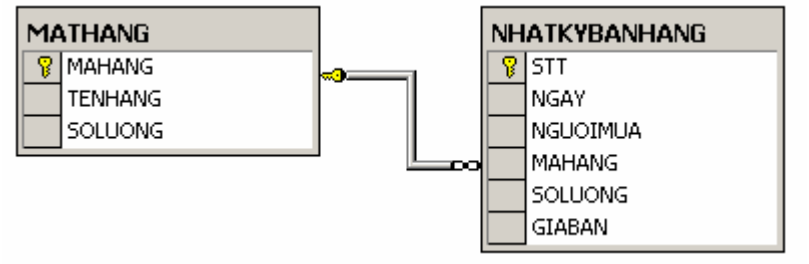
Có hai giải pháp có thể sử dụng đối với vấn đề này:

- Sử dụng truy vấn con.
- Sử dụng biến con trỏ.

5.3.4.1 Sử dụng truy vấn con

Ta hình dung vấn đề này và cách khắc phục qua ví dụ dưới đây:

Ví dụ 5.17: Ta xét lại trường hợp của hai bảng MATHANG và NHATKYBANHANG như sơ đồ dưới đây:



MAHANG	TENHANG	SOLUONG
H1	Xà phòng	30
H2	Kem đánh răng	45

STT	NGAY	NGUOIMUA	MAHANG	SOLUONG	GIABAN
1	1-1-2004	Ha	H1	10	10000.0000
2	2-2-2004	Phong	H2	20	5000.0000
3	3-3-2004	Thuy	H2	30	6000.0000

Trigger dưới đây cập nhật lại số lượng hàng của bảng MATHANG khi câu lệnh UPDATE được sử dụng để cập nhật cột SOLUONG của bảng NHATKYBANHANG.

```
CREATE TRIGGER trg_nhatkybanhang_update_soluong
ON nhatkybanhang
FOR UPDATE
AS
IF UPDATE(soluong)
  UPDATE mathang
  SET mathang.soluong = mathang.soluong –
       (inserted.soluong-deleted.soluong)
  FROM (deleted INNER JOIN inserted ON
        deleted.stt = inserted.stt) INNER JOIN mathang
        ON mathang.mahang = deleted.mahang
```

Với trigger được định nghĩa như trên, nếu thực hiện câu lệnh:

```
UPDATE nhatkybanhang
SET soluong = soluong + 10
WHERE stt = 1
```

thì dữ liệu trong hai bảng MATHANG và NHATKYBANHANG sẽ là:

MAHANG	TENHANG	SOLUONG
H1	Xà phòng	20
H2	Kem đánh răng	45

Bảng MATHANG

STT	NGAY	NGUOIMUA	MAHANG	SOLUONG	GIABAN
1	1-1-2004	Ha	H1	20	10000.0000
2	2-2-2004	Phong	H2	20	5000.0000
3	3-3-2004	Thuy	H2	30	6000.0000
4	4-4-2004	Dung	H1	40	9000.0000

Bảng NHATKYBANHANG

Tức là số lượng của mặt hàng có mã *H1* đã được giảm đi *10*. Nhưng nếu thực hiện tiếp câu lệnh:

```
UPDATE nhatkybanhang
SET soluong=soluong + 5
WHERE mahang='H2'
```

dữ liệu trong hai bảng sau khi câu lệnh thực hiện xong sẽ như sau:

MAHANG	TENHANG	SOLUONG
H1	Xà phòng	20
H2	Kem đánh răng	40

Bảng MATHANG

STT	NGAY	NGUOIMUA	MAHANG	SOLUONG	GIABAN
1	1-1-2004	Ha	H1	20	10000.0000
2	2-2-2004	Phong	H2	25	5000.0000
3	3-3-2004	Thuy	H2	35	6000.0000

Bảng NHATKYBANHANG

Ta có thể nhận thấy số lượng của mặt hàng có mã *H2* còn lại *40* (giảm đi 5) trong khi đúng ra phải là *35* (tức là phải giảm 10). Như vậy, trigger ở trên không hoạt động đúng trong trường hợp này.

Để khắc phục lỗi gặp phải như trên, ta định nghĩa lại trigger như sau:

```
CREATE TRIGGER trg_nhatkybanhang_update_soluong
ON nhatkybanhang
FOR UPDATE
AS
IF UPDATE(soluong)
  UPDATE mathang
  SET mathang.soluong = mathang.soluong -
        (SELECT SUM(inserted.soluong-deleted.soluong)
         FROM inserted INNER JOIN deleted
                       ON inserted.stt=deleted.stt
         WHERE inserted.mahang = mathang.mahang)
  WHERE mathang.mahang IN (SELECT mahang
                           FROM inserted)
```

hoặc:

```
CREATE TRIGGER trg_nhatkybanhang_update_soluong
ON nhatkybanhang
FOR UPDATE
AS
IF UPDATE(soluong)
/* Nếu số lượng dòng được cập nhật bằng 1 */
 IF @@ROWCOUNT = 1
 BEGIN
   UPDATE mathang
   SET mathang.soluong = mathang.soluong -
        (inserted.soluong-deleted.soluong)
   FROM (deleted INNER JOIN inserted ON
        deleted.stt = inserted.stt) INNER JOIN mathang
        ON mathang.mahang = deleted.mahang
```

```
 END
ELSE
 BEGIN
   UPDATE mathang
   SET mathang.soluong = mathang.soluong -
       (SELECT SUM(inserted.soluong-deleted.soluong)
        FROM inserted INNER JOIN deleted
                      ON inserted.stt=deleted.stt
        WHERE inserted.mahang = mathang.mahang)
   WHERE mathang.mahang IN (SELECT mahang
                            FROM inserted)
END
```

5.3.4.2 Sử dụng biến con trỏ

Một cách khác để khắc phục lỗi xảy ra như trong ví dụ 5.17 là sử dụng con trỏ để duyệt qua các dòng dữ liệu và kiểm tra trên từng dòng. Tuy nhiên, sử dụng biến con trỏ trong trigger là giải pháp nên chọn trong trường hợp thực sự cần thiết.

Một biến con trỏ được sử dụng để duyệt qua các dòng dữ liệu trong kết quả của một truy vấn và được khai báo theo cú pháp như sau:

```
DECLARE tên_con_trỏ  CURSOR
FOR câu_lệnh_SELECT
```

Trong đó câu lệnh SELECT phải có kết quả dưới dạng bảng. Tức là trong câu lệnh không sử dụng mệnh đề COMPUTE và INTO.

Để mở một biến con trỏ ta sử dụng câu lệnh:

```
OPEN tên_con_trỏ
```

Để sử dụng biến con trỏ duyệt qua các dòng dữ liệu của truy vấn, ta sử dụng câu lệnh FETCH. Giá trị của biến trạng thái @@FETCH_STATUS bằng không nếu chưa duyệt hết các dòng trong kết quả truy vấn.

Câu lệnh FETCH có cú pháp như sau:

```
FETCH [[NEXT|PRIOR|FIST|LAST] FROM] tên_con_trỏ
[INTO danh_sách_biến ]
```

Trong đó các biến trong danh sách biến được sử dụng để chứa các giá trị của các trường ứng với dòng dữ liệu mà con trỏ trỏ đến. Số lượng các biến phải bằng với số lượng các cột của kết quả truy vấn trong câu lệnh DECLARE CURSOR.

Ví dụ 5.18: Tập các câu lệnh trong ví dụ dưới đây minh hoạ cách sử dụng biến con trỏ để duyệt qua các dòng trong kết quả của câu lệnh SELECT

```
DECLARE contro CURSOR
        FOR SELECT mahang,tenhang,soluong FROM mathang
OPEN contro
DECLARE @mahang  NVARCHAR(10)
DECLARE @tenhang NVARCHAR(10)
DECLARE @soluong INT
/*Bắt đầu duyệt qua các dòng trong kết quả truy vấn*/
FETCH NEXT FROM contro
      INTO @mahang,@tenhang,@soluong
WHILE @@FETCH_STATUS=0
     BEGIN
          PRINT 'Ma hang:'+@mahang
          PRINT 'Ten hang:'+@tenhang
          PRINT 'So luong:'+STR(@soluong)
          FETCH NEXT FROM contro
                INTO @mahang,@tenhang,@soluong
     END
/*Đóng con trỏ và giải phóng vùng nhớ*/
CLOSE contro
DEALLOCATE contro
```

Ví dụ 5.19: Trigger dưới đây là một cách giải quyết khác của trường hợp được đề cập ở ví dụ 5.17

```
CREATE TRIGGER trg_nhatkybanhang_update_soluong
ON nhatkybanhang
FOR UPDATE
AS
IF UPDATE(soluong)
BEGIN
     DECLARE @mahang NVARCHAR(10)
     DECLARE @soluong INT

     DECLARE contro CURSOR FOR
       SELECT inserted.mahang,
              inserted.soluong-deleted.soluong AS soluong
       FROM inserted INNER JOIN deleted
            ON inserted.stt=deleted.stt

     OPEN contro
```

```
            FETCH NEXT FROM contro INTO @mahang,@soluong
            WHILE @@FETCH_STATUS=0
              BEGIN
                 UPDATE mathang SET soluong=soluong-@soluong
                 WHERE mahang=@mahang
                 FETCH NEXT FROM contro INTO @mahang,@soluong
              END
            CLOSE contro
            DEALLOCATE contro
      END
END
```

🕮 Bài tập chương 5

Dựa trên cơ sở dữ liệu ở bài tập chương 2, thực hiện các yêu cầu sau:

5.1 Tạo thủ tục lưu trữ để thông qua thủ tục này có thể bổ sung thêm một bản ghi mới cho bảng MATHANG (thủ tục phải thực hiện kiểm tra tính hợp lệ của dữ liệu cần bổ sung: không trùng khoá chính và đảm bảo toàn vẹn tham chiếu)

5.2 Tạo thủ tục lưu trữ có chức năng thống kê tổng số lượng hàng bán được của một mặt hàng có mã bất kỳ (mã mặt hàng cần thống kê là tham số của thủ tục).

5.3 Viết hàm trả về một bảng trong đó cho biết tổng số lượng hàng bán được của mỗi mặt hàng. Sử dụng hàm này để thống kê xem tổng số lượng hàng (hiện có và đã bán) của mỗi mặt hàng là bao nhiêu.

5.4 Viết trigger cho bảng CHITIETDATHANG theo yêu cầu sau:

- Khi một bản ghi mới được bổ sung vào bảng này thì giảm số lượng hàng hiện có nếu số lượng hàng hiện có lớn hơn hoặc bằng số lượng hàng được bán ra. Ngược lại thì huỷ bỏ thao tác bổ sung.
- Khi cập nhật lại số lượng hàng được bán, kiểm tra số lượng hàng được cập nhật lại có phù hợp hay không (số lượng hàng bán ra không được vượt quá số lượng hàng hiện có và không được nhỏ hơn 1). Nếu dữ liệu hợp lệ thì giảm (hoặc tăng) số lượng hàng hiện có trong công ty, ngược lại thì huỷ bỏ thao tác cập nhật.

5.5 Viết trigger cho bảng CHITIETDATHANG để sao cho chỉ chấp nhận giá hàng bán ra phải nhỏ hơn hoặc bằng giá gốc (giá của mặt hàng trong bảng MATHANG)

5.6 Để quản lý các bản tin trong một Website, người ta sử dụng hai bảng sau:

Bảng LOAIBANTIN (loại bản tin)

```
CREATE TABLE loaibantin
(
     maphanloai     INT             NOT NULL
                    PRIMARY KEY,
     tenphanloai    NVARCHAR(100)   NOT NULL ,
     bantinmoinhat  INT             DEFAULT(0)
)
```

Bảng BANTIN (bản tin)

```
CREATE TABLE bantin
 (
     maso           INT             NOT NULL
                    PRIMARY KEY,
     ngayduatin     DATETIME        NULL ,
     tieude         NVARCHAR(200)   NULL ,
     noidung        NTEXT           NULL ,
     maphanloai     INT             NULL
                    FOREIGN KEY
                    REFERENCES loaibantin(maphanloai)
)
```

Trong bảng LOAIBANTIN, giá trị cột BANTINMOINHAT cho biết mã số của bản tin thuộc loại tương ứng mới nhất (được bổ sung sau cùng).

Hãy viết các trigger cho bảng BANTIN sao cho:

- Khi một bản tin mới được bổ sung, cập nhật lại cột BANTINMOINHAT của dòng tương ứng với loại bản tin vừa bổ sung.
- Khi một bản tin bị xoá, cập nhật lại giá trị của cột BANTINMOINHAT trong bảng LOAIBANTIN của dòng ứng với loại bản tin vừa xóa là mã số của bản tin trước đó (dựa vào ngày đưa tin). Nếu không còn bản tin nào cùng loại thì giá trị của cột này bằng 0.
- Khi cập nhật lại mã số của một bản tin và nếu đó là bản tin mới nhất thì cập nhật lại giá trị cột BANTINMOINHAT là mã số mới.

✍ Lời giải:

```
5.1  CREATE PROCEDURE sp_insert_mathang(
               @mahang        NVARCHAR(10),
```

```
                @tenhang        NVARCHAR(50),
                @macongty       NVARCHAR(10) = NULL,
                @maloaihang     INT = NULL,
                @soluong        INT = 0,
                @donvitinh      NVARCHAR(20) = NULL,
                @giahang  money = 0)
    AS
         IF NOT EXISTS(SELECT mahang FROM mathang
                        WHERE mahang=@mahang)
           IF (@macongty IS NULL OR EXISTS(SELECT macongty
                        FROM nhacungcap
                        WHERE macongty=@macongty))
                        AND
                        (@maloaihang IS NULL OR
                        EXISTS(SELECT maloaihang FROM loaihang
                        WHERE maloaihang=@maloaihang))
                     INSERT INTO mathang
                     VALUES(@mahang,@tenhang,
                            @macongty,@maloaihang,
                            @soluong,@donvitinh,@giahang)
5.2 CREATE PROCEDURE sp_thongkebanhang(@mahang NVARCHAR(10))
    AS
         SELECT mathang.mahang,tenhang,
                SUM(chitietdathang.soluong) AS tongsoluong
         FROM mathang LEFT OUTER JOIN chitietdathang
              ON mathang.mahang=chitietdathang.mahang
         WHERE mathang.mahang=@mahang
         GROUP BY mathang.mahang,tenhang
```

5.3 *Định nghĩa hàm*:

```
    CREATE FUNCTION func_banhang()
    RETURNS TABLE
    AS
    RETURN (SELECT mathang.mahang,tenhang,
              CASE
               WHEN sum(chitietdathang.soluong) IS NULL THEN 0
               ELSE sum(chitietdathang.soluong)
              END AS tongsl
             FROM mathang LEFT OUTER JOIN chitietdathang
                  ON mathang.mahang = chitietdathang.mahang
```

```
            GROUP BY mathang.mahang,tenhang)
```

Sử dụng hàm đã định nghĩa:

```
     SELECT a.mahang,a.tenhang,soluong+tongsl
     FROM mathang AS a INNER JOIN dbo.func_banhang() AS b
          ON a.mahang=b.mahang
5.4  CREATE TRIGGER trg_chitietdathang_insert
     ON chitietdathang
     FOR INSERT
     AS
     BEGIN
          DECLARE @mahang NVARCHAR(100)
          DECLARE @soluongban INT
          DECLARE @soluongcon INT
          SELECT @mahang=mahang,@soluongban=soluong
          FROM inserted
          SELECT @soluongcon=soluong FROM mathang
          WHERE mahang=@mahang
          IF @soluongcon>=@soluongban
               UPDATE mathang SET soluong=soluong-@soluongban
               WHERE mahang=@mahang
          ELSE
               ROLLBACK TRANSACTION
     END

     CREATE TRIGGER trg_chitietdathang_update_soluong
     ON chitietdathang
     FOR UPDATE
     AS
     IF UPDATE(soluong)
     BEGIN
       IF EXISTS(SELECT sohoadon FROM inserted WHERE soluong<0)
               ROLLBACK TRANSACTION
       ELSE
         BEGIN
           UPDATE mathang
           SET soluong=soluong-
               (SELECT SUM(inserted.soluong-deleted.soluong)
                FROM inserted INNER JOIN deleted
                    ON inserted.sohoadon=deleted.sohoadon AND
```

```
                        inserted.mahang=deleted.mahang
                WHERE inserted.mahang=mathang.mahang
                GROUP BY inserted.mahang)
          WHERE mahang IN (SELECT DISTINCT mahang
                           FROM inserted)
          IF EXISTS(SELECT mahang FROM mathang
                    WHERE soluong<0)
              ROLLBACK TRANSACTION
        END
    END
5.5 CREATE TRIGGER trg_chitietdathang_giaban
    ON chitietdathang
    FOR INSERT,UPDATE
    AS
    IF UPDATE(giaban)
        IF EXISTS(SELECT inserted.mahang
                  FROM mathang INNER JOIN inserted
                       ON mathang.mahang=inserted.mahang
                  WHERE mathang.giahang>inserted.giaban)
            ROLLBACK TRANSACTION
```

Chương 6

GIAO TÁC SQL

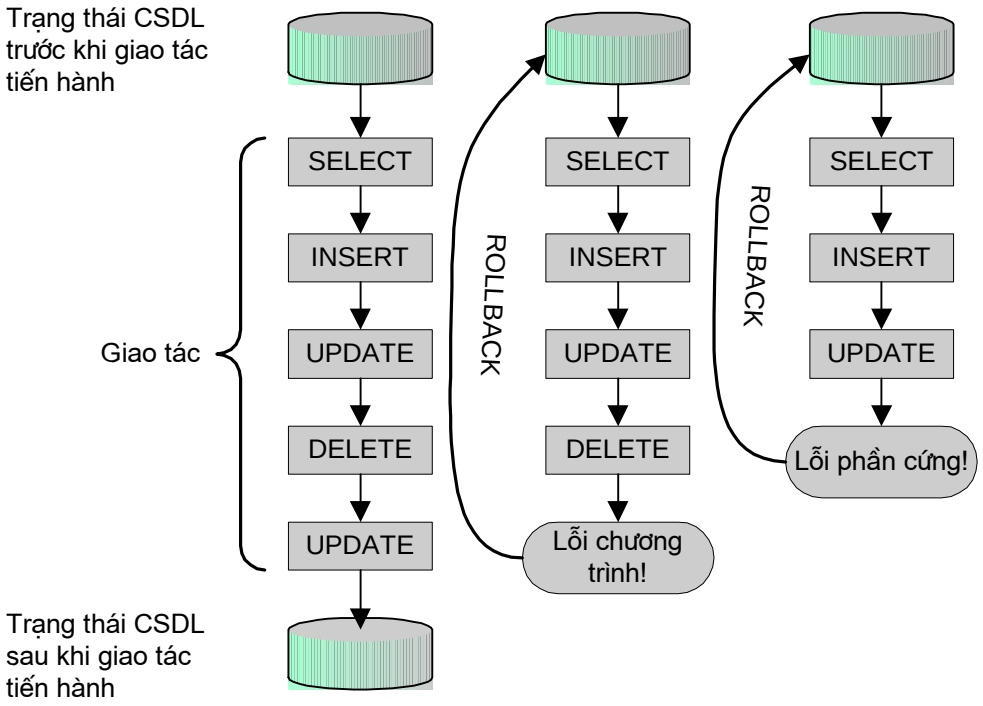
6.1 Giao tác và các tính chất của giao tác

Một giao tác (transaction) là một chuỗi một hoặc nhiều câu lệnh SQL được kết hợp lại với nhau thành một khối công việc. Các câu lệnh SQL xuất hiện trong giao tác thường có mối quan hệ tương đối mật thiết với nhau và thực hiện các thao tác độc lập. Việc kết hợp các câu lệnh lại với nhau trong một giao tác nhằm đảm bảo tính toàn vẹn dữ liệu và khả năng phục hồi dữ liệu. Trong một giao tác, các câu lệnh có thể độc lập với nhau nhưng tất cả các câu lệnh trong một giao tác đòi hỏi hoặc phải thực thi trọn vẹn hoặc không một câu lệnh nào được thực thi.

Các cơ sở dữ liệu sử dụng *nhật ký giao tác* (*transaction log*) để ghi lại các thay đổi mà giao tác tạo ra trên cơ sở dữ liệu và thông qua đó có thể phục hồi dữ liệu trong trường hợp gặp lỗi hay hệ thống có sự cố.

Một giao tác đòi hỏi phải có được bốn tính chất sau đây:

- **Tính nguyên tử (Atomicity):** Mọi thay đổi về mặt dữ liệu hoặc phải được thực hiện trọn vẹn khi giao tác thực hiện thành công hoặc không có bất kỳ sự thay đổi nào về dữ liệu xảy ra nếu giao tác không thực hiện được trọn vẹn. Nói cách khác, tác dụng của các câu lệnh trong một giao tác phải như là một câu lệnh đơn.
- **Tính nhất quán (Consistency):** Tính nhất quan đòi hỏi sau khi giao tác kết thúc, cho dù là thành công hay bị lỗi, tất cả dữ liệu phải ở trạng thái nhất quán (tức là sự toàn vẹn dữ liệu phải luôn được bảo toàn).
- **Tính độc lập (Isolation):** Tính độc lập của giao tác có nghĩa là tác dụng của mỗi một giao tác phải giống như khi chỉ mình nó được thực hiện trên chính hệ thống đó. Nói cách khác, một giao tác khi được thực thi đồng thời với những giao tác khác trên cùng hệ thống không chịu bất kỳ sự ảnh hưởng nào của các giao tác đó.
- **Tính bền vững (Durability):** Sau khi một giao tác đã thực hiện thành công, mọi tác dụng mà nó đã tạo ra phải tồn tại bền vững trong cơ sở dữ liệu, cho dù là hệ thống có bị lỗi đi chăng nữa.



Hình 6.1: *Giao tác SQL*

6.2 Mô hình giao tác trong SQL

Giao tác SQL được định nghĩa dựa trên các câu lệnh xử lý giao tác sau đây:

- BEGIN TRANSACTION: Bắt đầu một giao tác
- SAVE TRANSACTION: Đánh dấu một vị trí trong giao tác (gọi là điểm đánh dấu).
- ROLLBACK TRANSACTION: Quay lui trở lại đầu giao tác hoặc một điểm đánh dấu trước đó trong giao tác.
- COMMIT TRANSACTION: Đánh dấu điểm kết thúc một giao tác. Khi câu lệnh này thực thi cũng có nghĩa là giao tác đã thực hiện thành công.
- ROLLBACK [WORK]: Quay lui trở lại đầu giao tác.
- COMMIT [WORK]: Đánh dấu kết thúc giao tác.

Một giao tác trong SQL được bắt đầu bởi câu lệnh BEGIN TRANSACTION. Câu lệnh này đánh dấu điểm bắt đầu của một giao tác và có cú pháp như sau:

```
BEGIN TRANSACTION  [tên_giao_tác]
```

Một giao tác sẽ kết thúc trong các trường hợp sau:

- Câu lệnh COMMIT TRANSACTION (hoặc COMMIT WORK) được thực thi. Câu lệnh này báo hiệu sự kết thúc thành công của một giao tác. Sau câu lệnh này, một giao tác mới sẽ được bắt đầu.
- Khi câu lệnh ROLLBACK TRANSACTION (hoặc ROLLBACK WORK) được thực thi để huỷ bỏ một giao tác và đưa cơ sở dữ liệu về trạng thái như trước khi giao tác bắt đầu. Một giao tác mới sẽ bắt đầu sau khi câu lệnh ROLLBACK được thực thi.
- Một giao tác cũng sẽ kết thúc nếu trong quá trình thực hiện gặp lỗi (chẳng hạn hệ thống gặp lỗi, kết nối mạng bị "đứt",...). Trong trường hợp này, hệ thống sẽ tự động phục hồi lại trạng thái cơ sở dữ liệu như trước khi giao tác bắt đầu (tương tự như khi câu lệnh ROLLBACK được thực thi để huỷ bỏ một giao tác). Tuy nhiên, trong trường hợp này sẽ không có giao tác mới được bắt đầu.

Ví dụ 6.1: Giao tác dưới đây kết thúc do lệnh ROLLBACK TRANSACTION và mọi thay đổi về mặt dữ liệu mà giao tác đã thực hiện (UPDATE) đều không có tác dụng.

```
BEGIN TRANSACTION giaotac1
UPDATE monhoc SET sodvht=4 WHERE sodvht=3
UPDATE diemthi SET diemlan2=0 WHERE diemlan2 IS NULL
ROLLBACK TRANSACTION giaotac1
```

còn giao tác dưới đây kết thúc bởi lệnh COMMIT và thực hiện thành công việc cập nhật dữ liệu trên các bảng MONHOC và DIEMTHI.

```
BEGIN TRANSACTION giaotac2
UPDATE monhoc SET sodvht=4 WHERE sodvht=3
UPDATE diemthi SET diemlan2=0 WHERE diemlan2 IS NULL
COMMIT TRANSACTION giaotac2
```

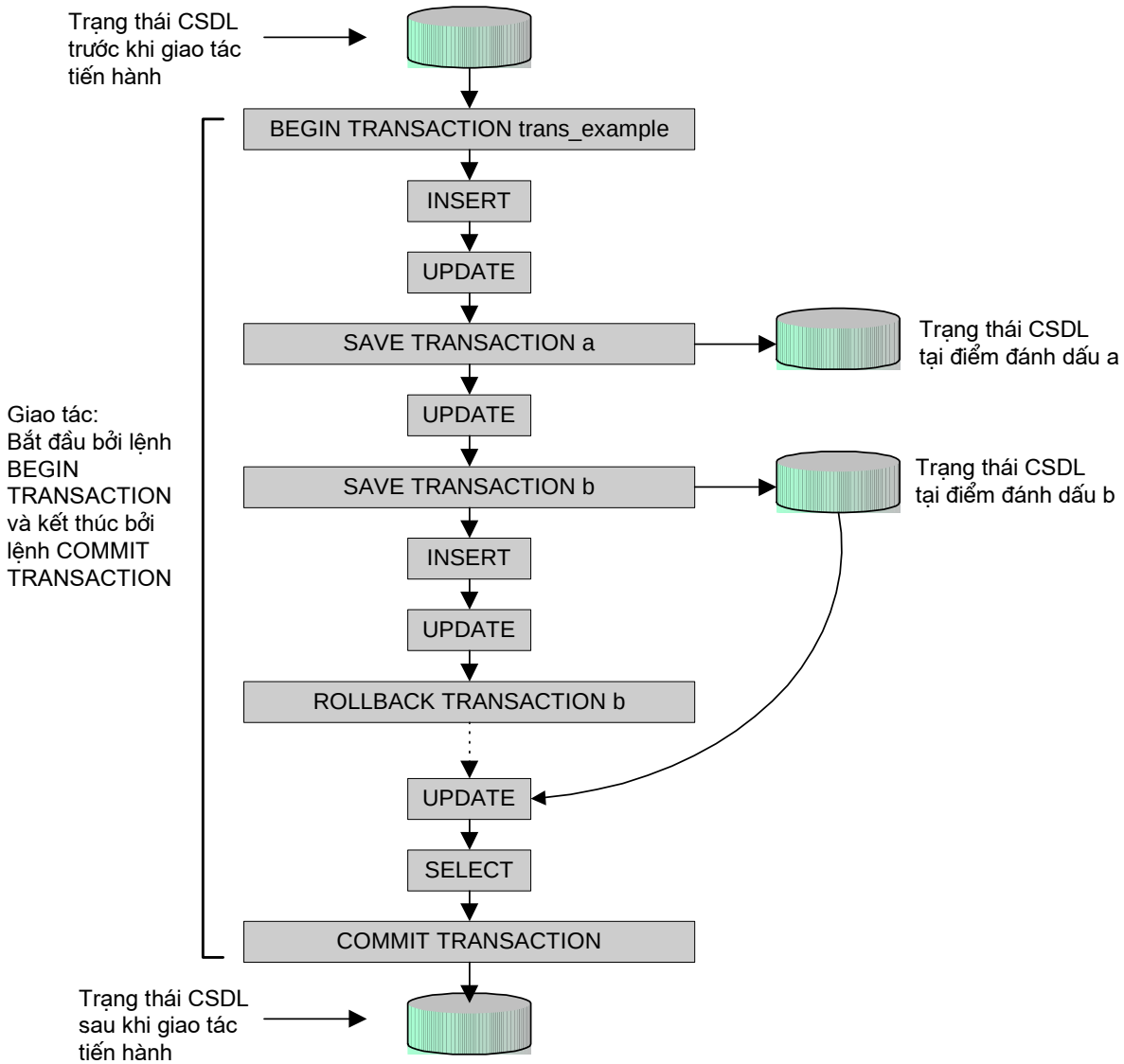
Câu lệnh:

```
SAVE TRANSACTION tên_điểm_đánh_dấu
```

được sử dụng để đánh dấu một vị trí trong giao tác. Khi câu lệnh này được thực thi, trạng thái của cơ sở dữ liệu tại thời điểm đó sẽ được ghi lại trong nhật ký giao tác. Trong quá trình thực thi giao tác có thể quay trở lại một điểm đánh dấu bằng cách sử dụng câu lệnh:

```
ROLLBACK TRANSACTION tên_điểm_đánh_dấu
```

Trong trường hợp này, những thay đổi về mặt dữ liệu mà giao tác đã thực hiện từ điểm đánh dấu đến trước khi câu lệnh ROLLBACK được triệu gọi sẽ bị huỷ bỏ. Giao tác sẽ được tiếp tục với trạng thái cơ sở dữ liệu có được tại điểm đánh dấu . Hình 6.2 mô tả cho ta thấy hoạt động của một giao tác có sử dụng các điểm đánh dấu:



Hình 6.2: *Hoạt động của một giao tác*

Sau khi câu lệnh ROLLBACK TRANSACTION được sử dụng để quay lui lại một điểm đánh dấu trong giao tác, giao tác vẫn được tiếp tục với các câu lệnh sau đó. Nhưng nếu câu lệnh này được sử dụng để quay lui lại đầu giao tác (tức là huỷ bỏ giao tác), giao tác sẽ kết thúc và do đó câu lệnh COMMIT TRANSACTION trong trường hợp này sẽ gặp lỗi.

Ví dụ 6.2: Câu lệnh COMMIT TRANSACTION trong giao tác dưới đây kết thúc thành công một giao tác

```
BEGIN TRANSACTION giaotac3
UPDATE diemthi SET diemlan2=0 WHERE diemlan2 IS NULL
```

```
SAVE TRANSACTION a
UPDATE monhoc SET sodvht=4 WHERE sodvht=3
ROLLBACK TRANSACTION a
UPDATE monhoc SET sodvht=2 WHERE sodvht=3
COMMIT TRANSACTION giaotac3
```

và trong ví dụ dưới đây, câu lệnh COMMIT TRANSACTION gặp lỗi:

```
BEGIN TRANSACTION giaotac4
UPDATE diemthi SET diemlan2=0 WHERE diemlan2 IS NULL
SAVE TRANSACTION a
UPDATE monhoc SET sodvht=4 WHERE sodvht=3
ROLLBACK TRANSACTION giaotac4
UPDATE monhoc SET sodvht=2 WHERE sodvht=3
COMMIT TRANSACTION giaotac4
```

6.3 Giao tác lồng nhau

Các giao tác trong SQL có thể được lồng vào nhau theo từng cấp. Điều này thường gặp đối với các giao tác trong các thủ tục lưu trữ được gọi hoặc từ một tiến trình trong một giao tác khác.

Ví dụ dưới đây minh hoạ cho ta trường hợp các giao tác lồng nhau.

Ví dụ 6.3: Ta định nghĩa bảng T như sau:

```
CREATE TABLE T
(
     A    INT  PRIMARY KEY,
     B    INT
)
```

và thủ tục **sp_TransEx**:

```
CREATE PROC sp_TranEx(@a INT,@b INT)
AS
BEGIN
     BEGIN TRANSACTION T1
     IF NOT EXISTS (SELECT * FROM T WHERE A=@A )
          INSERT INTO T VALUES(@A,@B)
     IF NOT EXISTS (SELECT * FROM T WHERE A=@A+1)
          INSERT INTO T VALUES(@A+1,@B+1)
     COMMIT TRANSACTION T1
END
```

Lời gọi đến thủ tuch sp_TransEx được thực hiện trong một giao tác khác như sau:

```
BEGIN TRANSACTION T3
```

```
        EXECUTE sp_tranex 10,20
    ROLLBACK TRANSACTION T3
```

Trong giao tác trên, câu lệnh `ROLLBACK TRANSACTION T3` huỷ bỏ giao tác và do đó tác dụng của lời gọi thủ tục trong giao tác không còn tác dụng, tức là không có dòng dữ liệu nào mới được bổ sung vào bảng T (cho dù giao tác T1 trong thủ tục *sp_tranex* đã thực hiện thành công với lệnh `COMMIT TRANSACTION T1`).

Ta xét tiếp một trường hợp của một giao tác khác trong đó có lời gọi đến thủ tục *sp_tranex* như sau:

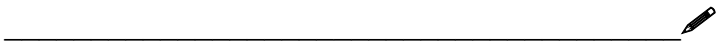
```
BEGIN TRANSACTION
EXECUTE sp_tranex 20,40
SAVE TRANSACTION a
EXECUTE sp_tranex 30,60
ROLLBACK TRANSACTION a
EXECUTE sp_tranex 40,80
COMMIT TRANSACTION
```

sau khi giao tác trên thực hiện xong, dữ liệu trong bảng T sẽ là:

A	B
20	40
21	41
40	80
41	81

Như vậy, tác dụng của lời gọi thủ tục `sp_tranex 30,60` trong giao tác đã bị huỷ bỏ bởi câu lệnh `ROLLBACK TRANSACTION` trong giao tác.

Như đã thấy trong ví dụ trên, khi các giao tác SQL được lồng vào nhau, giao tác ngoài cùng nhất là giao tác có vai trò quyết định. Nếu giao tác ngoài cùng nhất được uỷ thác (commit) thì các giao tác được lồng bên trong cũng đồng thời uỷ thác; Và nếu giao tác ngoài cùng nhất thực hiện lệnh ROLLBACK thì những giao tác lồng bên trong cũng chịu tác động của câu lệnh này (cho dù những giao tác lồng bên trong đã thực hiện lệnh COMMIT TRANSACTION).



PHỤ LỤC

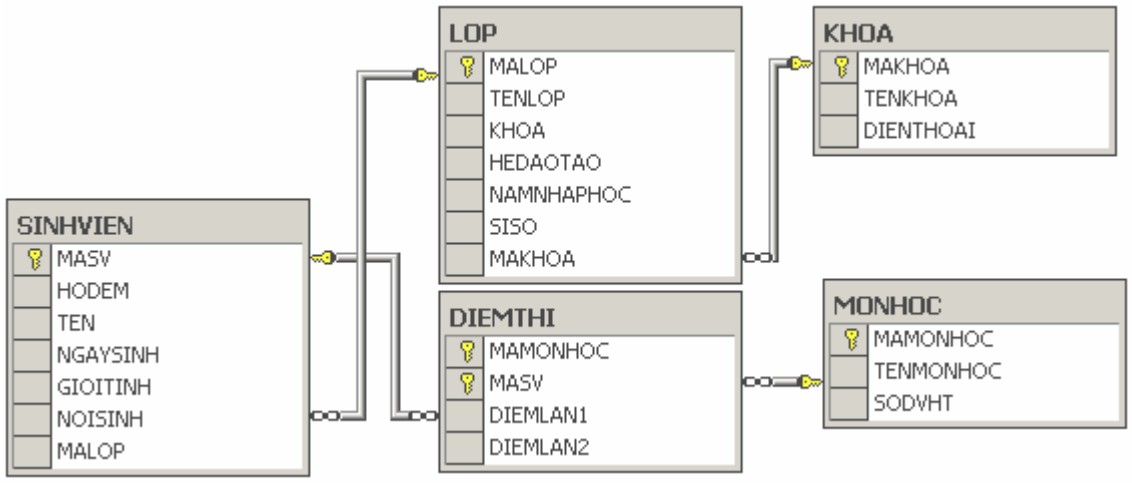
A. Cơ sở dữ liệu mẫu sử dụng trong giáo trình

Trong toàn bộ nội dung giáo trình, hầu hết các ví dụ được dựa trên cơ sở dữ liệu mẫu được mô tả dưới đây. Cơ sở dữ liệu này được cài đặt trong hệ quản trị cơ sở dữ liệu SQL Server 2000 và được sử dụng để quản lý sinh viên và điểm thi của sinh viên trong một trường đại học. Để tiện cho việc tra cứu và kiểm chứng đối với các ví dụ, trong phần đầu của phụ lục chúng tôi giới thiệu sơ qua về cơ sở dữ liệu này.

Cơ sở dữ liệu bao gồm các bảng sau đây:

- Bảng KHOA lưu trữ dữ liệu về các khoa hiện có ở trong trường
- Bảng LOP bao gồm dữ liệu về các lớp trong trường
- Bảng SINHVIEN được sử dụng để lưu trữ dữ liệu về các sinh viên trong trường.
- Bảng MONHOC bao gồm các môn học (học phần) được giảng dạy trong trường
- Bảng DIEMTHI với dữ liệu cho biết điểm thi kết thúc môn học của các sinh viên

Mối quan hệ giữa các bảng được thể hiện qua sơ đồ dưới đây



Các bảng trong cơ sở dữ liệu, mối quan hệ giữa chúng và một số ràng buộc được cài đặt như sau:

```
CREATE TABLE khoa
(
makhoa          NVARCHAR(5)     NOT NULL
                CONSTRAINT pk_khoa PRIMARY KEY,
tenkhoa         NVARCHAR(50)    NOT NULL ,
dienthoai       NVARCHAR(15)    NULL
)

CREATE TABLE lop
(
malop           NVARCHAR(10)    NOT NULL
                CONSTRAINT pk_lop  PRIMARY KEY,
tenlop          NVARCHAR(30)    NULL ,
khoa            SMALLINT        NULL ,
hedaotao        NVARCHAR(25)    NULL ,
namnhaphoc      INT             NULL ,
siso            INT             NULL ,
makhoa          NVARCHAR(5)     NULL
)

CREATE TABLE sinhvien
(
masv            NVARCHAR(10)    NOT NULL
                CONSTRAINT pk_sinhvien PRIMARY KEY,
hodem           NVARCHAR(25)    NOT NULL ,
ten             NVARCHAR(10)    NOT NULL ,
ngaysinh        SMALLDATETIME   NULL ,
gioitinh        BIT             NULL ,
noisinh         NVARCHAR(100)   NULL ,
malop           NVARCHAR(10)    NULL
)

CREATE TABLE  monhoc
(
mamonhoc        NVARCHAR(10)    NOT NULL
                CONSTRAINT pk_monhoc PRIMARY KEY,
tenmonhoc       NVARCHAR(50)    NOT NULL ,
```

```
sodvht          SMALLINT        NOT NULL
)

CREATE TABLE diemthi
(
mamonhoc        NVARCHAR(10)    NOT NULL ,
masv            NVARCHAR(10)    NOT NULL ,
diemlan1        NUMERIC(5, 2)   NULL ,
diemlan2        NUMERIC(5, 2)   NULL,
CONSTRAINT pk_diemthi PRIMARY KEY(mamonhoc,masv)
)

ALTER TABLE  lop
ADD
     CONSTRAINT fk_lop_khoa
     FOREIGN KEY(makhoa)
     REFERENCES khoa(makhoa)
     ON DELETE CASCADE
     ON UPDATE CASCADE

ALTER TABLE sinhvien
ADD
     CONSTRAINT fk_sinhvien_lop
     FOREIGN KEY (malop)
     REFERENCES lop(malop)
     ON DELETE CASCADE
     ON UPDATE CASCADE

ALTER TABLE diemthi
ADD
     CONSTRAINT fk_diemthi_monhoc
     FOREIGN KEY (mamonhoc)
     REFERENCES monhoc(mamonhoc)
     ON DELETE CASCADE
     ON UPDATE CASCADE,

     CONSTRAINT fk_diemthi_sinhvien
     FOREIGN KEY (masv)
     REFERENCES sinhvien(masv)
     ON DELETE CASCADE
```

```
        ON UPDATE CASCADE

ALTER TABLE monhoc
ADD
        CONSTRAINT chk_monhoc_sodht
        CHECK(sodvht>0 and sodvht<=5)

ALTER TABLE diemthi
ADD
        CONSTRAINT chk_diemthi_diemlan1
        CHECK (diemlan1>=0 and diemlan1<=10),
        CONSTRAINT chk_diemthi_diemlan2
        CHECK (diemlan2>=0 and diemlan2<=10)
```

B. Một số hàm thường sử dụng

Mặc dù trong SQL chuẩn không cung cấp cụ thể các nhưng trong các hệ quản trị cơ sở dữ liệu luôn cung cấp cho người sử dụng các hàm cài sẵn (hay còn gọi là các hàm của hệ thống). Trong phần này, chúng tôi cung cấp một số hàm thường được sử dụng trong SQL Server để tiện cho việc tra cứu và sử dụng trong thực hành

B.1 Các hàm trên dữ liệu kiểu chuỗi

Hàm ASCII

```
ASCII(string)
```

Hàm trả về mã ASCII của ký tự đầu tiên bên trái của chuỗi đối số

Hàm CHAR

```
CHAR(ascii_code)
```

Hàm trả về ký tự có mã ASCII tương ứng với đối số

Hàm CHARINDEX

```
CHARINDEX(string1,string2[,start])
```

Hàm trả về vị trí đầu tiên tính từ vị trí *start* tại đó chuỗi *string1* xuất hiện trong chuỗi *string2*.

Hàm LEFT

```
LEFT(string,number)
```

Hàm trích ra *number* ký tự từ chuỗi *string* tính từ phía bên trái

Hàm LEN

```
LEN(string)
```

Hàm trả về độ dài của chuỗi *string*.

Hàm LOWER

```
LOWER(string)
```

Hàm có chức năng chuyển chuỗi *string* thành chữ thường, kết quả được trả về cho hàm

Hàm LTRIM

```
LTRIM(string)
```

Cắt bỏ các khoảng trắng thừa bên trái chuỗi *string*

Hàm NCHAR

```
NCHAR(code_number)
```

Hàm trả về ký tự UNICODE có mã được chỉ định

Hàm REPLACE

```
REPLACE(string1,string2,string3)
```

Hàm trả về một chuỗi có được bằng cách thay thế các chuỗi *string2* trong chuỗi *string1* bởi chuỗi *string3*.

Hàm REVERSE

```
REVERSE(string)
```

Hàm trả về chuỗi đảo ngược của chuỗi *string*.

Hàm RIGHT

```
RIGHT(string, number)
```

Hàm trích ra *number* ký tự từ chuỗi *string* tính từ phía bên phải.

Hàm RTRIM

```
RTRIM(string)
```

Cắt bỏ các khoảng trắng thừa bên phải của chuỗi s*tring*.

Hàm SPACE

```
SPACE(number)
```

Hàm trả về một chuỗi với *number* khoảng trắng.

Hàm STR

```
STR(number [,length [,decimal]])
```

Chuyển giá trị kiểu số *number* thành chuỗi

Hàm SUBSTRING

```
SUBSTRING(string, m, n)
```

Trích ra từ *n* ký tự từ chuỗi *string* bắt đầu từ ký tự thứ *m*.

Hàm UNICODE

```
UNICODE(UnicodeString)
```

Hàm trả về mã UNICODE của ký tự đầu tiên bên trái của chuỗi *UnicodeString*.

Hàm UPPER

```
UPPER(string)
```

Chuyển chuỗi *string* thành chữ hoa

B.2 Các hàm trên dữ liệu kiểu ngày giờ

Hàm DATEADD

```
DATEADD(datepart, number, date)
```

Hàm trả về một giá trị kiểu DateTime bằng cách cộng thêm một khoảng giá trị là *number* vào ngày *date* được chỉ định. Trong đó, d*atepart* là 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

Hàm DATEDIFF

```
DATEDIFF(datepart, startdate, enddate)
```

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

Hàm DATEPART

```
DATEPART(datepart, date)
```

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.

Hàm GETDATE

```
GETDATE()
```

Hàm trả về ngày hiện tại

Hàm DAY, MONTH, YEAR

```
DAY(date), MONTH(date), YEAR(date)
```

Hàm trả về giá trị ngày (tháng hoặc năm) của giá trị kiểu ngày được chỉ định.

B.3 Hàm chuyển đổi kiểu

Hàm CAST

```
CAST (biểu_thức AS kiểu_dữ_liệu)
```

Chuyển đổi giá trị của biểu thức sang kiểu được chỉ định

Hàm CONVERT

```
CONVERT(kiểu_dữ_liệu, biểu_thức [,kiểu_chuyển_đổi])
```

Hàm có chức năng chuyển đổi giá trị của biểu thức sang kiểu dữ liệu được chỉ định. Tham số *kiểu_chuyển_đổi* là một giá trị số thường được sử dụng khi chuyển đổi giá trị kiểu ngày sang kiểu chuỗi nhằm qui định khuôn dạng dữ liệu được hiển thị và được qui định như sau:

Năm 2 chữ số	Năm 4 chữ số	Khuôn dạng dữ liệu
	0 hoặc 100	mon dd yyyy hh:mi AM (PM)
1	101	mm/dd/yy
2	102	yy.mm.dd
3	103	dd/mm/yy
4	104	dd.mm.yy
5	105	dd-mm-yy
6	106	dd mon yy
7	107	Mon dd, yy
8	108	hh:mm:ss
	9 hoặc 109	mon dd yyyy hh:mi:ss:mmmAM (PM)
10	110	mm-dd-yy
11	111	yy/mm/dd
12	112	yymmdd
	13 hoặc 113	dd mon yyyy hh:mm:ss:mmm(24h)
14	114	hh:mi:ss:mmm(24h)

```
20 hoặc 120  yyyy-mm-dd hh:mi:ss(24h)
21 hoặc 121  yyyy-mm-dd hh:mi:ss.mmm(24h)
126          yyyy-mm-dd Thh:mm:ss:mmm(no spaces)
130          dd mon yyyy hh:mi:ss:mmmAM
131          dd/mm/yy hh:mi:ss:mmmAM
```

Ví dụ: Câu lênh:

```
SELECT hodem,ten,
       CONVERT(NVARCHAR(20),ngaysinh,101) AS ngaysinh
FROM sinhvien
```

cho kết quả là:

HODEM	TEN	NGAYSINH
Ngô Thị Nhật	Anh	27.11.1982
Nguyễn Thị Ngọc	Anh	21.03.1983
Ngô Việt	Bắc	11.05.1982
Nguyễn Đình	Bình	06.10.1982
Hồ Đăng	Chiến	20.01.1982
...	...	...

TÀI LIỆU THAM KHẢO

1. James R, Groff, Paul N.Weinberg, SQL: The Complete Reference, McGraw-Hill/Osborne, 2002.
2. Diana Lorentz, SQL Reference, Oracle Corporation, 2001.
3. Marcilina S. Garcia, Jamie Reding, Edward Whalen, Steve Adrien DeLuca, SQL Server 2000 Administrator's Companion, Microsoft Press, 2000.
4. C. J. Date, Hugh Darwen, A Guide to the SQL Standard, Addison-Wesley Publishing, 1992.