

**ỦY BAN NHÂN DÂN THÀNH PHỐ HÀ NỘI  
SỞ BƯU CHÍNH VIỄN THÔNG HÀ NỘI**

**GIÁO TRÌNH  
NGÔN NGỮ SQL  
(Mã số giáo trình: 3CD3)**



HÀ NỘI. 2005

HÀ NỘI, 12-2004

## LỜI MỞ ĐẦU

Ngôn ngữ SQL (Structured Query Language) được sử dụng trong hầu hết các hệ quản trị cơ sở dữ liệu để truy vấn và sửa đổi cơ sở dữ liệu. Ngôn ngữ SQL hỗ trợ các truy vấn dựa trên các phép toán đại số quan hệ, đồng thời cũng chứa các lệnh sửa đổi cơ sở dữ liệu và mô tả lược đồ cơ sở dữ liệu. Như vậy, SQL vừa là một ngôn ngữ thao tác dữ liệu, vừa là một ngôn ngữ định nghĩa dữ liệu. Ngoài ra SQL cũng tiêu chuẩn hoá nhiều lệnh cơ sở dữ liệu khác.

Có nhiều phiên bản khác nhau của SQL. Trước tiên, có ba bản chuẩn. Đó là ANSI (American National Standards Institute) SQL. Sau đó đến năm 1992, bản chuẩn SQL-92 ra đời gọi là SQL2. Gần đây nhất, chuẩn SQL-99 (trước đó gọi là SQL3) mở rộng SQL2 với các đặc trưng quan hệ - đối tượng và một số khả năng mới khác. Ngoài ra còn có nhiều phiên bản của SQL được các nhà bán các hệ quản trị cơ sở dữ liệu sản xuất. Các phiên bản này có tất cả các khả năng của chuẩn ANSI nguyên gốc và chúng cũng phù hợp với các mở rộng của SQL cũng như các tính chất của chuẩn SQL-99. Trong giáo trình này chúng tôi trình bày dựa trên chuẩn SQL-99. Giáo trình gồm ba chương:

Chương 1: SQL cơ bản, trình bày các truy vấn cơ bản trên các bảng cơ sở dữ liệu, các kiểu dữ liệu cơ bản trong SQL và cách tạo cơ sở dữ liệu đơn giản trong SQL

Chương 2: Các ràng buộc và các trigger. Chương này trình bày các loại ràng buộc: ràng buộc miền, ràng buộc khóa, ràng buộc toàn vẹn thực thể, ràng buộc toàn vẹn tham chiếu, các ràng buộc khác và cách thể hiện chúng trong SQL.

Chương 3: Lập trình với SQL, trình bày các phương pháp lập trình trong SQL: lập trình nhúng, SQL động, các hàm và các thủ tục PSM, sử dụng giao diện gọi. Ngoài ra, chương này còn đề cập đến vấn đề an toàn trên cơ sở dữ liệu SQL.

Cuối mỗi chương có tổng kết các vấn đề trình bày trong chương và một số bài tập. Để hiểu được giáo trình này bạn đọc cần phải có các kiến thức về cơ sở dữ liệu quan hệ.

Do hạn chế về thời gian và kinh nghiệm, chắc chắn giáo trình vẫn còn nhiều thiếu sót. Mong các bạn đọc góp ý, phê bình. Chúng tôi xin cảm ơn trước và hứa sẽ tiếp thu để hoàn thiện giáo trình hơn.

- Tên môn học: Ngôn ngữ SQL.

- Mã số môn học: 3CD3
- Thời gian: 45 tiết (lí thuyết + thực hành)
- Mục tiêu: Hướng dẫn học viên sử dụng thành thạo ngôn ngữ truy vấn SQL.
- Những kiến thức cần được trang bị trước: Cơ sở dữ liệu quan hệ.
- Nội dung môn học:

**Chương I: CƠ BẢN VỀ SQL.**

**Chương II: CÁC Ràng BUỘC VÀ TRIGGER.**

**Chương III: LẬP TRÌNH**

- Đối tượng học: Các lập trình viên.
- Biên soạn: Bộ môn Các hệ thống thông tin, Khoa Công nghệ thông tin, Trường ĐH Công Nghệ, ĐHQG Hà Nội.

LỜI MỞ ĐẦU.....	2
MỤC LỤC .....	<b>Error! Bookmark not defined.</b>
CHƯƠNG I: SQL CƠ BẢN .....	8
1.1 CÁC TRUY VẤN ĐƠN GIẢN TRONG SQL.....	8
1.1.1 Phép chiếu trong SQL.....	9
1.1.2 Phép chọn trong SQL.....	11
1.1.3 So sánh các xâu.....	13
1.1.4 Ngày tháng và thời gian.....	14
1.1.5 Các giá trị NULL và các so sánh bao hàm NULL.....	15
1.1.6 Giá trị lôgic UNKNOWN.....	16
1.1.7 Sắp thứ tự dữ liệu ra.....	17
1.1.8 Các hàm thông dụng trong SQL.....	18
1.2 CÁC TRUY VẤN BAO GỒM NHIỀU HƠN MỘT QUAN HỆ.....	20
1.2.1 Tích và nối trong SQL.....	20
1.2.2 Làm rõ nghĩa các thuộc tính.....	21
1.2.3 Các biến bộ.....	22
1.2.4 Phép hợp, phép giao, phép trừ của các truy vấn.....	23
1.3 CÁC TRUY VẤN CON.....	25
1.3.1 Các truy vấn con tạo ra các giá trị vô hướng.....	25
1.3.2 Các điều kiện có bao hàm các quan hệ.....	27
1.3.3 Các điều kiện có bao hàm các bộ.....	27
1.3.4 Các truy vấn con tương quan với nhau.....	28
1.3.5 Các truy vấn con trong mệnh đề FROM.....	30
1.3.6 Các biểu thức nối của SQL.....	31
1.3.7 Nối tự nhiên (Natural Join).....	32
1.3.8 Nối ngoài.....	33
1.4 CÁC PHÉP TOÁN QUAN HỆ ĐẦY ĐỦ.....	34
1.4.1 Loại bỏ trùng lặp.....	34
1.4.2 Trùng lặp trong phép hợp, phép giao và phép trừ.....	34
1.4.3 Nhóm và sự kết hợp trong SQL.....	36
1.4.4 Các phép toán nhóm.....	36
1.4.5 Nhóm.....	37
1.4.6 Các mệnh đề HAVING.....	40
1.5 SỬA ĐỔI CƠ SỞ DỮ LIỆU.....	41
1.5.1 Chèn.....	41
1.5.2 Xóa.....	43
1.5.3 Cập nhật.....	44
1.6 ĐỊNH NGHĨA MỘT LƯỢC ĐỒ QUAN HỆ TRONG SQL.....	45

1.6.1	Các kiểu dữ liệu .....	45
1.6.2	Các khai báo bảng đơn giản .....	46
1.6.4	Các giá trị ngầm định .....	48
1.6.5	Các chỉ số .....	48
1.6.6	Nhập môn về việc lựa chọn các chỉ số .....	49
1.7	KHUNG NHÌN (VIEW) .....	50
1.7.1	Khai báo các khung nhìn .....	50
1.7.2	Truy vấn các khung nhìn .....	51
1.7.3	Đặt tên lại các thuộc tính .....	53
1.7.4	Sửa đổi các khung nhìn .....	53
1.7.5	Giải thích các truy vấn có chứa các khung nhìn .....	56
1.8	TỔNG KẾT CHƯƠNG I .....	59
	MỘT SỐ BÀI TẬP .....	61
	CHƯƠNG II: CÁC RÀNG BUỘC VÀ CÁC TRIGGER .....	65
2.1	KHOÁ VÀ KHOÁ NGOÀI .....	66
2.1.1	Mô tả khoá chính .....	66
2.1.2	Các khoá được mô tả với UNIQUE .....	67
2.1.3	Làm có hiệu lực các ràng buộc khoá .....	68
2.1.4	Mô tả các ràng buộc khoá ngoài .....	69
2.1.5	Duy trì toàn vẹn tham chiếu .....	71
2.1.6	Làm chậm việc kiểm tra ràng buộc .....	73
2.2	CÁC RÀNG BUỘC TRÊN CÁC THUỘC TÍNH VÀ CÁC BỘ .....	76
2.2.1	Các ràng buộc Not-Null .....	76
2.2.2	Các ràng buộc kiểm tra (CHECK) dựa trên thuộc tính .....	77
2.2.3	Các ràng buộc kiểm tra (CHECK) dựa trên bộ giá trị .....	79
2.3	SỬA ĐỔI CÁC RÀNG BUỘC .....	80
2.3.1	Đặt tên cho các ràng buộc .....	80
2.3.2	Sửa đổi các ràng buộc trên các bảng .....	80
2.4	CÁC RÀNG BUỘC MỨC LỰC ĐỒ VÀ CÁC TRIGGER .....	81
2.4.1	Các khẳng định (assertion) .....	82
	So sánh các ràng buộc .....	85
2.4.2	Trigger .....	85
	Trigger trong SQL .....	86
	Các trigger Thay vì (Instead-Of) .....	91
	Chúng ta nhìn thấy từ khóa INSTEAD OF ở dòng 2) chứng minh rằng một phép chèn vào NVHÀNỘI sẽ không bao giờ xảy ra. ....	92
2.5	TỔNG KẾT CHƯƠNG II .....	92
	MỘT SỐ BÀI TẬP .....	93
	CHƯƠNG III: LẬP TRÌNH .....	96
3.1	SQL TRONG MÔI TRƯỜNG LẬP TRÌNH .....	96

3.1.1 Vấn đề trở ngại không phù hợp .....	97
3.1.2 Giao diện ngôn ngữ chủ /SQL .....	98
3.1.3 Phần khai báo (DECLARE) .....	99
3.1.4 Sử dụng các biến dùng chung .....	100
3.1.5 Các câu lệnh Select đơn hàng .....	102
3.1.6 Con trỏ .....	103
3.1.7 Cập nhật bằng con trỏ .....	107
3.1.8 Bảo vệ khỏi sự cập nhật đồng thời .....	108
3.1.9 Con trỏ cuộn (Scrolling Cursor) .....	110
3.1.10 SQL động .....	111
3.2 CÁC THỦ TỤC ĐƯỢC LƯU GIỮ (stored procedure) .....	113
3.2.1 Tạo các hàm và các thủ tục PSM .....	113
3.2.2 Một vài dạng câu lệnh đơn giản trong PSM .....	115
3.2.3 Các câu lệnh rẽ nhánh .....	117
3.2.4 Các truy vấn trong PSM .....	119
3.2.5 Vòng lặp trong PSM .....	120
3.2.6 Vòng lặp for .....	123
3.2.7 Những câu lệnh lặp khác .....	124
3.2.8 Những loại trừ trong PSM .....	125
3.2.9 Sử dụng các hàm và các thủ tục PSM .....	127
3.3 MÔI TRƯỜNG SQL .....	128
3.3.1 Môi trường .....	128
3.3.2 Lược đồ .....	129
3.3.3 Các danh mục (Catalog) .....	131
<i>Thêm về các phần tử lược đồ</i> .....	131
3.3.4 Kết nối .....	132
3.3.5 Phiên (Session) .....	133
3.3.6 Modules .....	134
3.4 SỬ DỤNG GIAO DIỆN MỨC GỌI (call-level interface) .....	135
3.4.1 Nhập môn SQL/CLI .....	135
3.4.2 Xử lý các lệnh .....	138
3.4.3 Lấy dữ liệu ra từ kết quả truy vấn .....	140
3.5 GIAO TÁC TRONG SQL .....	142
3.5.1 Xếp hàng theo thứ tự .....	142
3.5.2 Atomicity .....	145
3.5.3 Giao tác (Transaction) .....	147
3.5.4 Read-Only Transaction .....	148
3.5.5 Dirty Read .....	150
3.5.6 Các mức cô lập khác .....	153
3.6 AN TOÀN VÀ CẤP QUYỀN TRONG SQL .....	154

3.6.1 Các quyền .....	155
3.6.2 Tạo các quyền .....	157
3.6.3 Tiến trình kiểm tra đặc quyền.....	158
3.6.4 Cấp các quyền.....	159
3.6.5 Biểu đồ grant.....	161
3.6.6 Hủy bỏ các quyền.....	162
3.7 TỔNG KẾT CHƯƠNG III.....	167

## CHƯƠNG I: SQL CƠ BẢN

Giống như các ngôn ngữ bậc cao khác, ngôn ngữ SQL được xây dựng dựa trên các chữ cái, các chữ số, các ký tự (dấu phép toán, dấu ngăn, dấu cách và các ký tự đặc biệt) và một tập các từ khóa. Một lệnh của SQL có thể được viết trên một dòng hoặc nhiều dòng, kết thúc bằng dấu chấm phẩy “;”.

Ngôn ngữ SQL được chia thành ba nhóm:

- Ngôn ngữ định nghĩa dữ liệu dùng để mô tả cấu trúc của cơ sở dữ liệu (các bảng, các khung nhìn, các thuộc tính, các chỉ mục, ...)
- Ngôn ngữ thao tác dữ liệu cho phép thực hiện các thao tác trên cơ sở dữ liệu như cập nhật cơ sở dữ liệu và truy vấn lấy ra các thông tin từ cơ sở dữ liệu.
- Ngôn ngữ kiểm soát dữ liệu bao gồm các lệnh dùng để quản lý các giao tác, các quyền truy cập dữ liệu, kết nối với server..

Ngôn ngữ SQL có thể sử dụng theo hai kiểu: kiểu trực tiếp và lập trình. SQL trực tiếp cho phép thực hiện một truy vấn và nhận được kết quả ngay tức khắc. SQL lập trình cho phép sử dụng SQL trong một chương trình viết bằng ngôn ngữ lập trình bậc cao khác (C, Pascal,..), hoặc viết các chương trình con.

Trong chương này chúng ta sẽ làm quen với các lệnh cơ bản của SQL. Các lệnh này được minh họa dựa trên một cơ sở dữ liệu “CÔNGTY” cho ở phần PHỤ LỤC của giáo trình.

### 1.1 CÁC TRUY VẤN ĐƠN GIẢN TRONG SQL.

Giả sử chúng ta muốn đưa ra các nhân viên của đơn vị có MãSốĐV = 5, chúng ta viết trong SQL như sau

```
SELECT *  
FROM NHÂNVIÊN  
WHERE MãSốĐV = 5 ;
```

Truy vấn này trình bày dạng đặc trưng *select-from-where* của hầu hết các truy vấn SQL.

- Mệnh đề FROM cho quan hệ hoặc các quan hệ mà truy vấn tham chiếu đến. Trong ví dụ trên, quan hệ đó là NHÂNVIÊN.



- Mệnh đề WHERE là một điều kiện, giống như điều kiện chọn trong đại số quan hệ. Các bộ phải thoả mãn điều kiện chọn để phù hợp với truy vấn. Điều kiện ở đây là thuộc tính MãSốĐV của bộ phải có giá trị 5. Tất cả các bộ đáp ứng điều kiện đó sẽ thoả mãn điều kiện chọn.
- Mệnh đề SELECT nói các thuộc tính nào của các bộ đáp ứng điều kiện sẽ được đưa ra như một phần của câu trả lời. Dấu \* trong ví dụ này chỉ ra rằng tất cả các thuộc tính của bộ sẽ được đưa ra. Kết quả của truy vấn là một quan hệ chứa tất cả các bộ do tiến trình này sản xuất ra.

Một cách giải thích truy vấn này là xem xét từng bộ giá trị của quan hệ được kê ra trong mệnh đề FROM. Điều kiện trong mệnh đề WHERE được áp dụng cho bộ. Chính xác hơn, các thuộc tính được kê ra trong mệnh đề WHERE được thay thế bằng các giá trị của thuộc tính đó ở trong bộ. Sau đó, điều kiện được tính, và nếu đúng thì các thành phần xuất hiện trong mệnh đề SELECT được sản xuất ra như là một bộ của câu trả lời.

### 1.1.1 Phép chiếu trong SQL

a) Chúng ta có thể chiếu một quan hệ do một truy vấn SQL sản xuất ra lên trên một số thuộc tính của nó. Để làm điều đó, ở vị trí của dấu \* trong mệnh đề SELECT ta liệt kê ra một số thuộc tính của quan hệ được chỉ ra trong mệnh đề FROM. Kết quả sẽ được chiếu lên các thuộc tính được liệt kê.

Ví dụ 1: Đưa ra Họđệm và Tên của các nhân viên ở đơn vị có mã số bằng 5. Chúng ta có thể viết:

```
SELECT Họđệm, Tên
FROM NHÂNVIÊN
WHERE MãSốĐV =5;
```

Kết quả là một bảng có hai cột, có tên là Họđệm và Tên. Các bộ của bảng này là các cặp, mỗi cặp gồm Họđệm và Tên của nhân viên, đó là các nhân viên của đơn vị có mã số bằng 5. Bảng kết quả có dạng như sau:

<u>Họđệm</u>	<u>Tên</u>
Lê	Vân

Trần Đức Nam  
Nguyễn Sơn  
Vũ Hương Giang

b) Đôi khi chúng ta muốn tạo ra một quan hệ với đầu cột khác với các thuộc tính của quan hệ được kể ra trong mệnh đề FROM. Chúng ta có thể viết sau tên của thuộc tính một từ khoá AS và một bí danh (alias), bí danh đó sẽ trở thành đầu cột của quan hệ kết quả. Từ khoá AS là tùy chọn, nghĩa là có thể viết bí danh đi ngay sau tên thuộc tính mà không cần phải có từ khoá AS.

Ví dụ 2: Ta có thể sửa đổi ví dụ 1 ở trên để đưa ra một quan hệ có các thuộc tính HỌNHÂNVIÊN và TÊN NHÂNVIÊN thay cho vị trí của HỌĐỆM và TÊN như sau:

```
SELECT HỌĐỆM AS HỌNHÂNVIÊN, TÊN AS TÊN NHÂNVIÊN  
FROM NHÂNVIÊN  
WHERE MÃSỐĐV = 5 ;
```

Bảng kết quả có dạng như sau:

<u>HỌNHÂNVIÊN</u>	<u>TÊN NHÂNVIÊN</u>
Lê	Vân
Trần Đức	Nam
Nguyễn	Sơn
Vũ Hương	Giang

c) Một tùy chọn khác trong mệnh đề SELECT sử dụng một biểu thức ở vị trí của một thuộc tính.

Ví dụ 3: Chúng ta muốn đưa ra HỌĐỆM, TÊN và lương sau khi đã được tăng 10% của các nhân viên ở đơn vị có mã số bằng 5. Ta viết:

```
SELECT HỌĐỆM, TÊN, LƯƠNG*1.1 AS LƯƠNG MỚI  
FROM NHÂNVIÊN  
WHERE MÃSỐĐV =5;
```

<u>Kết quả</u>	<u>HỌĐỆM</u>	<u>TÊN</u>	<u>LƯƠNG MỚI</u>
	Lê	Vân	3300

Trần Đức	Nam	4400
Nguyễn	Son	4180
Vũ Hương	Giang	2750

d) Chúng ta cũng có thể cho phép một hằng như là một biểu thức trong mệnh đề SELECT.

Ví dụ 4: Ta muốn đưa thêm từ ‘ngàn đồng’ vào sau giá trị của lương, ta viết:

```
SELECT Họđệm, Tên, Lương*1.1 AS Lươngmới, ‘ngàn đồng’ AS Đonvítính
FROM NHÂNVIÊN
WHERE MẫsốĐV =5;
```

Kết quả	<u>Họđệm</u>	<u>Tên</u>	<u>Lươngmới</u>	<u>Đonvítính</u>
	Lê	Vân	3300	ngàn đồng
	Trần Đức	Nam	4400	ngàn đồng
	Nguyễn	Son	4180	ngàn đồng
	Vũ Hương	Giang	2750	ngàn đồng

Chúng ta đã sắp xếp một cột có tên là Đonvítính và mỗi bộ trong câu trả lời sẽ có hằng ‘ngàn đồng’ ở cột thứ tư.

### 1.1.2 Phép chọn trong SQL

Phép toán chọn của đại số quan hệ và nhiều thứ nữa sẵn có trong mệnh đề WHERE của SQL. Các biểu thức đi sau WHERE bao gồm các biểu thức điều kiện giống như các biểu thức điều kiện trong các ngôn ngữ lập trình. Chúng ta có thể xây dựng các điều kiện bằng cách so sánh các giá trị sử dụng sáu phép toán so sánh =, <>, <, >, <=, >=. Các giá trị có thể được so sánh bao gồm các hằng và các thuộc tính của các quan hệ được kể ra sau FROM. Chúng ta cũng có thể áp dụng các phép toán số học thông thường như +, -, \*, / đối với các giá trị số trước khi chúng ta so sánh chúng và áp dụng phép nối || đối với các xâu. Một ví dụ về phép so sánh là

MẫsốĐV = 5

Ở trong các ví dụ ở trên. Thuộc tính MẫsốĐV được kiểm tra xem có bằng hằng 5 hay không. Hằng này là một giá trị số. Các hằng số, như các số nguyên và số thực được sử dụng và được ghi như cách thông thường trong

các ngôn ngữ lập trình. Ngoài các hằng số còn có các hằng xâu. Các xâu trong SQL được ghi bằng cách đặt chúng và trong cặp dấu nháy đơn, ví dụ, ‘Hà nội’.

Kết quả của một phép so sánh là một giá trị lô gic TRUE hoặc FALSE. Các giá trị lô gic có thể được kết hợp bằng các phép toán logic AND, OR, NOT với các ý nghĩa của chúng.

Ví dụ 5: Truy vấn sau đây hỏi về Họđệm, Tên và Giớitính của các nhân viên ở đơn vị có mã số bằng 5 và Giớitính = ‘Nam’

```
SELECT Họđệm, Tên, Giớitính
FROM NHÂNVIÊN
WHERE (MãSốĐV =5) AND (Giớitính = ‘Nam’);
```

Kết quả	<u>Họđệm</u>	<u>Tên</u>	<u>Giớitính</u>
	Lê	Vân	Nam
	Trần Đức	Nam	Nam
	Nguyễn	Sơn	Nam

Trong điều kiện này, chúng ta có AND của hai giá trị logic. Các giá trị đó là các phép so sánh bình thường. Tiếp theo, ta xét ví dụ sau:

```
SELECT Họđệm, Tên
FROM NHÂNVIÊN
WHERE (MãSốĐV =5) AND (Giớitính = ‘Nữ’ OR Lương <= 3000);
```

<u>Họđệm</u>	<u>Tên</u>
Lê	Vân
Vũ Hương	Giang

Truy vấn này đòi hỏi các nhân viên hoặc là nữ hoặc có lương nhỏ hơn hoặc bằng 3000. Chú ý rằng các phép so sánh có thể nhóm lại bằng việc sử dụng các dấu ngoặc đơn. Các dấu ngoặc là cần thiết bởi vì thứ tự ưu tiên của các phép toán lô gic trong SQL là giống như trong các ngôn ngữ lập trình, AND có thứ tự cao hơn OR, NOT có thứ tự cao hơn cả AND và OR.

### 1.1.3 So sánh các chuỗi

Hai chuỗi là bằng nhau nếu chúng là cùng một dãy ký tự. SQL cho phép các mô tả các kiểu chuỗi khác nhau, ví dụ, các mảng ký tự có độ dài cố định và các danh sách ký tự có độ dài thay đổi.

Khi chúng ta so sánh các chuỗi bằng một trong các phép toán “nhỏ hơn” như là  $<$  hoặc  $>=$ , chúng ta đang hỏi xem có phải chuỗi này đi trước chuỗi kia trong thứ tự từ điển. Như vậy, nếu  $a_1a_2\dots a_n$  và  $b_1b_2\dots b_m$  là hai chuỗi, thì chuỗi thứ nhất là “nhỏ hơn” chuỗi thứ hai nếu hoặc  $a_1 < b_1$ , hoặc nếu  $a_1 = b_1$  và  $a_2 < b_2$ , hoặc  $a_1 = b_1$ ,  $a_2 = b_2$  và  $a_3 < b_3$  ... Ta cũng nói rằng  $a_1a_2\dots a_n < b_1b_2\dots b_m$  nếu  $n < m$  và  $a_1a_2\dots a_n = b_1b_2\dots b_n$ , nghĩa là chuỗi thứ nhất là một tiền tố đúng của chuỗi thứ hai. Ví dụ ‘na’  $<$  ‘nam’.

SQL cũng cung cấp khả năng để so sánh các chuỗi trên cơ sở một mẫu đối chiếu đơn giản. Một dạng lựa chọn của biểu thức logic là

$s$  LIKE  $p$

trong đó  $s$  là một chuỗi và  $p$  là một mẫu đối chiếu. Một mẫu đối chiếu là một chuỗi có sử dụng hai ký tự đặc biệt % và \_. Các ký tự thông thường trong  $p$  chỉ đối sánh được với chính chúng ở trong  $s$ , nhưng % có thể đối sánh với một dãy có 0 hoặc nhiều hơn các ký tự trong  $s$ , và \_ đối sánh với bất kỳ ký tự nào trong  $s$ . Giá trị của biểu thức này là đúng khi và chỉ khi chuỗi  $s$  hợp với mẫu  $p$ . Một cách tương tự,  $s$  NOT LIKE  $p$  là đúng khi và chỉ khi chuỗi  $s$  không hợp với mẫu  $p$ .

Ví dụ 6:

```
SELECT Tên  
FROM NHÂNVIÊN  
WHERE Tên LIKE ‘N_ _’;
```

Truy vấn này đòi hỏi thuộc tính Tên có giá trị gồm 3 ký tự, ký tự đầu tiên là N và sau đó là một dãy nào đó gồm hai ký tự. Kết quả của truy vấn này là một tập các tên nhân viên thích hợp, chẳng hạn như Nam, Núi, ..

Ví dụ 7: Chúng ta hãy tìm tên của các nhân viên có chứa chữ a. Ta có truy vấn sau:

```
SELECT Tên
```

FROM NHÂNVIÊN

WHERE Tên LIKE '%a%';

Kết quả của truy vấn này là một tập các tên nhân viên thoả mãn điều kiện chọn, chẳng hạn như Nam, Thanh, Hoa.

#### 1.1.4 Ngày tháng và thời gian

Các thể hiện của SQL nói chung hỗ trợ ngày tháng và thời gian như những kiểu dữ liệu đặc biệt. Các giá trị này thường trình bày được trong nhiều dạng khác nhau như 14/5/1948 hoặc 14-05-48. Ở đây chúng ta sẽ chỉ mô tả cách ghi chuẩn của SQL.

Một hằng ngày tháng được biểu diễn bằng từ khoá DATE sau đó là một chuỗi có dạng đặc biệt để bên trong cặp dấu nháy đơn. Ví dụ: DATE'1948-05-14'. Bốn ký tự đầu là các chữ số biểu diễn năm, sau đó là dấu -, hai ký tự tiếp theo là các chữ số biểu diễn tháng, tiếp theo là dấu - và cuối cùng là hai ký tự số biểu diễn ngày.

Một hằng thời gian được biểu diễn tương tự bằng từ khoá TIME và một chuỗi được đặt trong cặp dấu nháy đơn. Chuỗi này có hai chữ số cho giờ trên đồng hồ quân sự (24 giờ), sau đó là dấu hai chấm, hai chữ số cho phút, một dấu hai chấm nữa và hai chữ số cho giây. Nếu phần lẻ của giây là cần thiết, chúng ta có thể tiếp tục với một dấu chấm và một số các chữ số có nghĩa. Ví dụ, TIME'15:00:02.5' biểu diễn thời gian 15 giờ không phút hai giây 5 phần mười. Thời gian còn có thể được biểu diễn theo nhiều cách khác nữa.

Để kết hợp ngày tháng và thời gian chúng ta sử dụng một giá trị kiểu TIMESTAMP. Các giá trị này gồm một từ khoá TIMESTAMP, một giá trị ngày tháng, một khoảng cách và một giá trị thời gian. Ví dụ, TIMESTAMP'1948-05-14 12:00:00' biểu diễn 12 giờ trưa ngày 14 tháng 5 năm 1948.

Chúng ta có thể so sánh ngày tháng và thời gian bằng cách sử dụng các phép toán so sánh giống như đối với các số hoặc các chuỗi. Như vậy, dấu < trên ngày tháng chứng tỏ ngày tháng thứ nhất sớm hơn ngày tháng thứ hai, còn dấu < trên thời gian chứng tỏ thời gian thứ nhất sớm hơn thời gian thứ hai.

### 1.1.5 Các giá trị NULL và các so sánh bao hàm NULL.

SQL cho phép các thuộc tính có giá trị đặc biệt NULL, được gọi là giá trị null. Một thuộc tính có giá trị null khi không biết giá trị của nó hoặc khi giá trị là không áp dụng được hoặc giá trị bị giấu.

Trong mệnh đề WHERE, chúng ta có thể được chuẩn bị cho khả năng một thành phần của một bộ nào đó là null. Có hai quy tắc quan trọng cần phải nhớ khi chúng ta làm phép toán trên các giá trị null.

1. Khi chúng ta làm phép toán một giá trị null và một giá trị nào đó, bao gồm cả giá trị null khác, bằng việc sử dụng một phép toán số học như là \* hoặc +, kết quả phép toán là null.

2. Khi chúng ta so sánh một giá trị null và một giá trị khác, bao hàm cả giá trị null khác, bằng cách sử dụng các phép so sánh như là = hoặc >, kết quả phép so sánh là UNKNOWN. Giá trị UNKNOWN là một giá trị lô gic khác, giống như TRUE và FALSE.

Chúng ta phải nhớ rằng mặc dù NULL là một giá trị có thể xuất hiện trong các bộ nhưng nó không phải là một hằng. Như vậy, trong khi các quy tắc ở trên áp dụng khi chúng ta cố gắng làm phép toán trên một biểu thức mà giá trị của nó là NULL, chúng ta không thể dùng NULL một cách rõ như là một toán hạng.

Ví dụ 8: Giả sử x có giá trị null. Khi đó giá trị của x+3 cũng là null. Tuy nhiên null+3 không phải là một biểu thức SQL hợp lệ. Tương tự, giá trị của x = 3 là UNKNOWN bởi vì chúng ta không thể nói rằng giá trị của x (một giá trị NULL) là bằng 3. Tuy nhiên, phép so sánh NULL = 3 không phải là phép so sánh SQL đúng.

Cách đúng đắn để hỏi xem x có giá trị null hay không là dùng biểu thức x IS NULL. Biểu thức này có giá trị TRUE nếu x có giá trị NULL và nó có giá trị FALSE trong trường hợp ngược lại. Một cách tương tự, x IS NOT NULL có giá trị TRUE trừ khi giá trị của x là NULL. Trong một số phiên bản của SQL, trước khi thực hiện các phép toán với các giá trị null, người ta sử dụng các hàm chuyển đổi giá trị null thành ra giá trị 0 (nếu toán hạng

tương ứng có kiểu số) hoặc thành một chuỗi rỗng ‘ ’ nếu toán hạng tương ứng là kiểu ký tự.

### 1.1.6 Giá trị logic UNKNOWN

Ở trên, chúng ta giả thiết rằng kết quả của một phép so sánh hoặc là TRUE hoặc là FALSE, và các giá trị logic này được kết hợp một cách rõ ràng bằng việc sử dụng các phép toán AND, OR, NOT. Khi xuất hiện giá trị NULL, các phép so sánh có thể cho một giá trị logic thứ ba UNKNOWN. Bây giờ chúng ta phải biết các phép toán logic đối xử như thế nào trên các tổ hợp của ba giá trị logic này. Chúng ta có thể nghĩ đến TRUE như là 1, FALSE như là 0, UNKNOWN như là 1/2. Khi đó:

1. AND của hai giá trị logic là min của các giá trị đó. Như vậy,  $x \text{ AND } y$  là FALSE nếu  $x$  hoặc  $y$  là FALSE, là UNKNOWN nếu  $x$  và  $y$  không là FALSE nhưng ít nhất có một giá trị là UNKNOWN và là TRUE khi cả  $x$  và  $y$  là TRUE.

2. OR của hai giá trị logic là max của các giá trị đó. Như vậy  $x \text{ OR } y$  là TRUE nếu  $x$  hoặc  $y$  là TRUE, là UNKNOWN nếu  $x$  và  $y$  không là TRUE nhưng có ít nhất là một giá trị UNKNOWN và có giá trị FALSE nếu cả  $x$  và  $y$  đều FALSE.

3. Phủ định của giá trị logic  $v$  là  $1-v$ . Như vậy, NOT  $x$  có giá trị TRUE khi  $x$  là FALSE, có giá trị FALSE khi  $x$  là TRUE và có giá trị UNKNOWN khi  $x$  là UNKNOWN. Bảng dưới đây tổng kết các phép toán logic trên các giá trị logic:

$x$	$y$	$x \text{ AND } y$	$x \text{ OR } y$	NOT $x$
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE



FALSE FALSE FALSE FALSE TRUE

Hình 1: Bảng chân trị cho logic ba giá trị

Các điều kiện SQL như xuất hiện trong các mệnh đề WHERE của các lệnh *select-from-where*, áp dụng cho mỗi bộ trong một quan hệ nào đấy, và với mỗi bộ, một trong ba giá trị TRUE, FALSE, hoặc UNKNOWN được sinh ra. Tuy nhiên, chỉ có các bộ mà đối với nó điều kiện có giá trị TRUE sẽ trở thành một phần của câu trả lời; các bộ có UNKNOWN hoặc FALSE như là giá trị sẽ bị loại ra khỏi câu trả lời.

Ví dụ 9: Giả sử chúng ta có truy vấn trên đây trên quan hệ NHÂNVIÊN\_DỰÁN

```
SELECT *  
FROM NHÂNVIÊN_DỰÁN  
WHERE Sôgiờ <=12 OR Sôgiờ > 12 ;
```

Một cách trực quan, chúng ta mong đợi nhận được một bản sao của quan hệ NHÂNVIÊN\_DỰÁN, bởi vì mỗi nhân viên có thể làm việc cho một dự án ít hơn hoặc bằng 12 giờ hoặc nhiều hơn 12 giờ. Tuy nhiên, giả sử rằng có các bộ của NHÂNVIÊN\_DỰÁN có giá trị NULL trong thành phần Sôgiờ. Khi đó cả hai phép so sánh Sôgiờ <= 12 và Sôgiờ >12 được tính là UNKNOWN. OR của hai UNKNOWN là UNKNOWN. Như vậy, với mọi bộ có một NULL trong thành phần Sôgiờ, mệnh đề WHERE được tính là UNKNOWN. Một bộ như vậy không được trả lại như một phần của câu trả lời cho truy vấn. Kết quả được đưa ra theo ý nghĩa đúng đắn của truy vấn là “tìm tất cả các bộ NHÂNVIÊN\_DỰÁN có Sôgiờ không NULL”.

### 1.1.7 Sắp thứ tự dữ liệu ra

Chúng ta có thể yêu cầu rằng các bộ được một truy vấn tạo ra sẽ được biểu diễn trong một thứ tự sắp xếp. Thứ tự có thể dựa trên giá trị của một thuộc tính nào đó, kết hợp với giá trị của thuộc tính thứ hai, .... Để nhận được dữ liệu ra theo một thứ tự sắp xếp, chúng ta thêm vào lệnh *select-from-where* một mệnh đề

```
ORDER BY < danh sách các thuộc tính >
```

Thứ tự được ngầm định là tăng dần nhưng chúng ta có thể nhận dữ liệu ra theo thứ tự giảm dần bằng cách thêm vào từ khoá DESC. Tương tự, chúng ta có thể chỉ ra thứ tự tăng dần bằng cách thêm vào từ khoá ASC (tùy chọn).

Ví dụ 10: Để nhận được các Họđệm, Tên theo thứ tự tăng dần của Tên của tất cả các nhân viên trong đơn vị có mã số bằng 5, ta có truy vấn sau:

```
SELECT Họđệm, Tên
FROM NHÂNVIÊN
WHERE MẫsốĐV = 5
ORDER BY Tên ;
```

### 1.1.8 Các hàm thông dụng trong SQL

Trong SQL có một số các hàm xây dựng sẵn. Sau đây là một số hàm thông dụng.

#### 1) Các hàm nhóm:

Hàm AVG trả về giá trị trung bình của cột. Ví dụ:

```
SELECT AVG(Lương)
FROM NHÂNVIÊN;
```

Hàm MIN trả về giá trị nhỏ nhất của cột. Ví dụ:

```
SELECT MIN(Lương)
FROM NHÂNVIÊN ;
```

Hàm MAX trả về giá trị lớn nhất của cột. Ví dụ:

```
SELECT MAX(Lương)
FROM NHÂNVIÊN ;
```

Hàm SUM trả về tổng các giá trị của cột. Ví dụ:

```
SELECT SUM(Lương)
FROM NHÂNVIÊN ;
```

Hàm COUNT trả về số lượng các bản ghi. Ví dụ:

```
SELECT COUNT(*)
FROM NHÂNVIÊN ;
```

Việc sử dụng các hàm này trong các phép toán nhóm sẽ nói đến trong các phần sau.

## 2) Các hàm xử lý các chuỗi ký tự

Hàm ASCII, trả về giá trị mã ASCII của ký tự bên trái chuỗi. Ví dụ:

Print ASCII('TÔI'); trả về kết quả 84 (mã ASCII của T).

Hàm CHAR, chuyển đổi mã ASCII sang ký tự. Ví dụ:

Print CHAR(35); trả về kết quả ký tự #

Hàm UPPER, chuyển đổi chuỗi sang kiểu chữ hoa. Ví dụ:

Print UPPER('Nam'); trả về kết quả NAM

Hàm LOWER, chuyển đổi chuỗi sang kiểu chữ thường. Ví dụ:

Print LOWER('NAM'); trả về kết quả nam

Hàm LEN, trả về độ dài của chuỗi. Ví dụ:

Print LEN('NAM'); trả về kết quả 3.

Hàm LTRIM, loại bỏ các khoảng trống bên trái của chuỗi. Ví dụ:

Print LTRIM(' NAM'); trả về kết quả 'NAM'.

Hàm RTRIM, loại bỏ các khoảng trống bên phải của chuỗi. Ví dụ:

Print RTRIM('NAM '); trả về kết quả 'NAM'.

Hàm LEFT(chuỗi,n) trả về n ký tự bên trái của chuỗi. Ví dụ

Print LEFT('NAM', 2); trả về kết quả 'NA'.

Hàm RIGHT(chuỗi,n) trả về n ký tự bên phải của chuỗi. Ví dụ

Print RIGHT('NAM', 1); trả về kết quả 'AM'.

Hàm CHARINDEX (chuỗi1, chuỗi2) trả về vị trí bắt đầu của chuỗi 1 trong chuỗi 2. Ví dụ:

CHARINDEX('Tâm', 'Hữu Tâm') trả về kết quả 4.

## 3) Các hàm thời gian

Hàm GETDATE() trả về ngày tháng năm của hệ thống.

Ví dụ SELECT GETDATE() trả về kết quả: 2004-10-17 14:25:36.234

Hàm DATEPART() trả về một phần của một chuỗi dạng ngày tháng đầy đủ

DATEPART(d,GETDATE()), trả về ngày

DATEPART(m,GETDATE()), trả về tháng

DATEPART(yy,GETDATE()), trả về năm ....Các tham số d,m,yy là định dạng ngày, tháng, năm, ...

Hàm DATEDIFF (định dạng, Ngàytrước, Ngàysau) hiệu số giữa Ngày sau và

Ngàytrước

Hàm DAY trả về ngày, Hàm MONTH trả về tháng, Hàm YEAR trả về năm

#### 4) Các hàm toán học

Hàm SQUARE trả về bình phương của một biểu thức.

Hàm SQRT trả về căn bậc hai của một biểu thức

Hàm ROUND trả về số làm tròn của một biểu thức

#### 5) Các hàm chuyển đổi

Hàm CAST trả về giá trị có kiểu dữ liệu theo định nghĩa. Ví dụ

PRINT CAST (GETDATE() AS VARCHAR) trả về Oct 18 2004.

Hàm CONVERT chuyển đổi giá trị từ kiểu này sang kiểu khác.

## 1.2 CÁC TRUY VẤN BAO GỒM NHIỀU HƠN MỘT QUAN HỆ

Sức mạnh của đại số quan hệ là khả năng tổ hợp hai hoặc nhiều quan hệ thông qua các phép nối, tích, hợp, giao và trừ. Trong SQL có tất cả các phép toán đó.

### 1.2.1 Tích và nối trong SQL

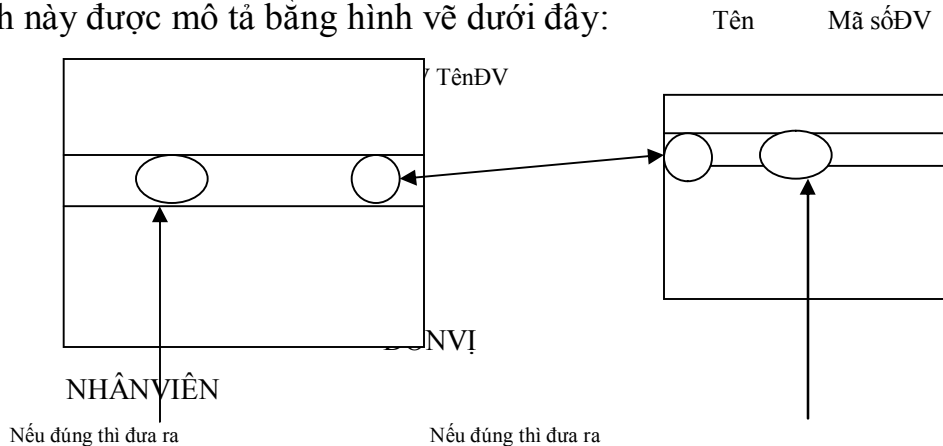
SQL có một cách đơn giản để ghép cặp các quan hệ vào một truy vấn: liệt kê từng quan hệ trong mệnh đề FROM. Sau đó, các mệnh đề SELECT và WHERE có thể tham chiếu đến các thuộc tính của bất kỳ quan hệ nào bên trong mệnh đề FROM.

Ví dụ 11: Giả sử chúng ta muốn biết tên của nhân viên và tên đơn vị của người đó. Để trả lời cho câu hỏi này, chúng ta cần hai quan hệ NHÂNVIÊN và ĐƠNVI. Ta có truy vấn sau:

```
SELECT Tên, TênĐV
FROM NHÂNVIÊN, ĐƠNVI
```

WHERE NHÂNVIÊN.MãSốĐV = ĐƠNVI.MãSốĐV ;

Truy vấn này đòi hỏi chúng ta xem xét tất cả các cặp bộ giá trị, một từ ĐƠNVI và bộ kia từ NHÂNVIÊN. Điều kiện trên các cặp này được nói rõ trong mệnh đề WHERE: Thành phần MãSốĐV trong các bộ này phải có giá trị như nhau. Khi nào chúng ta tìm thấy một cặp bộ thoả mãn điều kiện, chúng ta đưa ra các thuộc tính Tên của bộ từ quan hệ NHÂNVIÊN và thuộc tính TênĐV của bộ từ quan hệ ĐƠNVI như một phần của câu trả lời. Quá trình này được mô tả bằng hình vẽ dưới đây:



Hình 2: minh họa truy vấn của ví dụ 11.

### 1.2.2 Làm rõ nghĩa các thuộc tính

Đôi khi chúng ta đòi hỏi một truy vấn bao gồm nhiều quan hệ và trong những quan hệ này có hai hoặc nhiều thuộc tính có cùng tên. Nếu như vậy, chúng ta cần có cách để chỉ rõ thuộc tính nào trong số các thuộc tính đó là được sử dụng. SQL giải quyết vấn đề này bằng cách cho phép ta đặt tên quan hệ và một dấu chấm ở đằng trước thuộc tính. Như vậy, R.A tham chiếu đến thuộc tính A của quan hệ R.

Trong ví dụ 11, hai quan hệ NHÂNVIÊN và ĐƠNVI có các thuộc tính MãSốĐV trùng tên. Để phân biệt, trong mệnh đề WHERE ta viết

NHÂNVIÊN.MãSốĐV = ĐƠNVI.MãSốĐV

Một quan hệ, theo sau là một dấu chấm được cho phép ngay cả trong trường hợp khi không có sự không rõ nghĩa. Ví dụ, chúng ta hoàn toàn thoải mái khi viết truy vấn có dạng như sau:

```
SELECT NHÂNVIÊN.Tên, ĐƠNVI.TênĐV  
FROM NHÂNVIÊN, ĐƠNVI  
WHERE NHÂNVIÊN.MãSốĐV = ĐƠNVI.MãSốĐV ;
```

Kết quả của truy vấn 11 là: Tên      TênĐV

Vân	Nghiêncứu
Nam	Nghiêncứu
Thanh	Hànhchính
Bằng	Hànhchính
Sơn	Nghiêncứu
Giang	Nghiêncứu
Hoa	Hànhchính
Giáp	Lãnhđạo

### 1.2.3 Các biến bộ

Việc làm rõ nghĩa các thuộc tính bằng cách thêm tên quan hệ vào đầu hoạt động khi một truy vấn bao hàm tổ hợp nhiều quan hệ khác nhau. Tuy nhiên, đôi khi chúng ta cần đòi hỏi một truy vấn bao hàm hai hoặc nhiều bộ từ cùng một quan hệ. Chúng ta có thể liệt kê một quan hệ R bao nhiêu lần như ta muốn trong một mệnh đề FROM, nhưng chúng ta cần có cách tham chiếu đến từng lần có mặt của R. SQL cho phép chúng ta định nghĩa đối với từng lần có mặt của R trong mệnh đề FROM một “bí danh” mà chúng ta sẽ tham chiếu đến nó như là một biến bộ. Mỗi lần sử dụng của R trong mệnh đề FROM được theo sau bằng một từ khoá AS và tên của biến bộ. Từ khoá AS là không bắt buộc, có thể có hoặc không. Trong phạm vi tài liệu này, chúng ta sẽ bỏ qua từ khoá AS.

Trong các mệnh đề SELECT và WHERE, chúng ta có thể làm rõ nghĩa các thuộc tính của R bằng cách thêm vào trước chúng một biến bộ thích hợp và một dấu chấm. Như vậy, một biến bộ phục vụ như là một tên khác của quan hệ R và có thể được sử dụng trong vị trí của nó khi chúng ta muốn.

Ví dụ 13: Giả sử chúng ta muốn đưa ra tên của nhân viên và tên của người giám sát nhân viên đó. Như vậy, các tên này đều được lấy từ quan hệ NHÂNVIÊN. Sử dụng các biến bộ như các bí danh cho hai sử dụng của NHÂNVIÊN, chúng ta có thể viết truy vấn như sau:

```
SELECT NV.Tên, NV1.Tên
FROM NHÂNVIÊN NV, NHÂNVIÊN NV1
WHERE NV.MãSốNGS = NV1.Mã sốNV;
```

Chúng ta nhìn thấy trong mệnh đề FROM mô tả của hai biến bộ, NV và NV1; mỗi biến bộ là một bí danh cho quan hệ NHÂNVIÊN. Các biến bộ được sử dụng trong mệnh đề SELECT để tham chiếu đến các thành phần Tên của hai bộ. Các bí danh này cũng được sử dụng trong mệnh đề WHERE để nói rằng hai bộ từ NHÂNVIÊN được biểu diễn bởi NV và NV1 có giá trị như nhau trong các thành phần MãSốNGS và MãSốNV của chúng

Kết quả của truy vấn 13	<u>NV.Tên</u>	<u>NV1.Tên</u>
	Vân	Nam
	Nam	Giáp
	Thanh	Bằng
	Bằng	Giáp
	Son	Nam
	Giang	Nam
	Hoa	Bằng

#### 1.2.4 Phép hợp, phép giao, phép trừ của các truy vấn

Đôi khi chúng ta muốn tổ hợp các quan hệ bằng cách sử dụng các phép toán tập hợp của đại số quan hệ: hợp, giao, trừ. SQL cung cấp các phép toán tương ứng áp dụng cho các kết quả của các truy vấn với điều kiện là các truy vấn đó tạo ra các quan hệ có cùng danh sách các thuộc tính và các kiểu thuộc tính. Các từ khoá được sử dụng là UNION, INTERSECT và EXCEPT cho hợp, giao và trừ tương ứng. Các từ như UNION được sử dụng giữa hai truy vấn, và các truy vấn này phải được đặt trong cặp dấu ngoặc đơn.

Ví dụ 14: Giả sử chúng ta muốn đưa ra MãSốNV của các nhân viên làm việc cho dự án có MãSốDA = 1 và các nhân viên làm việc cho dự án có Mã sốDA = 2. Sử dụng quan hệ NHÂNVIÊN\_DỰÁN, ta viết truy vấn như sau:

```
(SELECT MãSốNV
FROM NHÂNVIÊN_DỰÁN
```

```
WHERE MăsốDA = 1)
UNION
(SELECT MăsốNV
FROM NHÂNVIÊN_DỰÁN
WHERE MăsốDA = 2)
```

Lệnh SELECT đầu đưa ra các MăsốNV của các nhân viên làm việc cho dự án có MăsốDA = 1, lệnh SELECT sau đưa ra các MăsốNV của các nhân viên làm việc cho dự án có MăsốDA = 2, hai tập hợp này hợp với nhau tạo thành câu trả lời cho truy vấn.

Kết quả	<u>MăsốNV</u>
	NV001
	NV002
	NV016
	NV018 (các bộ trùng lặp bị loại bỏ)

Ví dụ 15: Theo cách tương tự, chúng ta muốn đưa ra MăsốNV của các nhân viên vừa làm việc cho dự án có MăsốDA = 1 vừa làm việc cho dự án có MăsốDA = 2. Sử dụng quan hệ NHÂNVIÊN\_DỰÁN, ta viết truy vấn như sau:

```
(SELECT MăsốNV
FROM NHÂNVIÊN_DỰÁN WHERE MăsốDA = 1)
INTERSECT
(SELECT MăsốNV
FROM NHÂNVIÊN_DỰÁN WHERE MăsốDA = 2)
```

Kết quả	<u>MăsốNV</u>
	NV001
	NV018

Ví dụ 16: Giả sử chúng ta muốn đưa ra MăsốNV của các nhân viên làm việc cho dự án có MăsốDA = 1 nhưng không làm việc cho dự án có MăsốDA = 2. Sử dụng quan hệ NHÂNVIÊN\_DỰÁN, ta viết truy vấn như sau:



```
(SELECT MăsốNV  
FROM NHÂNVIÊN_DỰÁN  
WHERE MăsốDA = 1)  
EXCEPT  
(SELECT MăsốNV  
FROM NHÂNVIÊN_DỰÁN  
WHERE MăsốDA = 2)
```

Kết quả MăsốNV  
(không có)

### 1.3 CÁC TRUY VẤN CON

Trong SQL, một truy vấn có thể được sử dụng trong nhiều cách khác nhau để giúp việc tính giá trị của truy vấn khác. Một truy vấn là một phần của truy vấn khác được gọi là một truy vấn con. Các truy vấn con lại có thể có các truy vấn con và như vậy có thể đi xuống rất nhiều mức. Chúng ta đã có cơ hội nhìn thấy việc sử dụng truy vấn con. Trong các ví dụ ở phần trên, chúng ta đã xây dựng các truy vấn hợp, giao, trừ bằng cách nối hai truy vấn con để tạo nên truy vấn đầy đủ. Có rất nhiều cách để sử dụng các truy vấn con:

1. Các truy vấn con có thể trả lại một hằng đơn và có thể so sánh hằng đó với giá trị khác trong mệnh đề WHERE.
2. Các truy vấn con có thể trả lại các quan hệ và có thể sử dụng các quan hệ này bằng nhiều cách trong mệnh đề WHERE.
3. Các truy vấn con có thể có các quan hệ của chúng xuất hiện trong mệnh đề FROM giống như các quan hệ được lưu giữ có thể.

#### 1.3.1 Các truy vấn con tạo ra các giá trị vô hướng

Một giá trị nguyên tử có thể xuất hiện như một thành phần của một bộ được xem là một vô hướng. Một biểu thức *select-from-where* có thể tạo ra một quan hệ có số các thuộc tính tùy ý và có số bộ giá trị tùy ý trong quan hệ. Tuy nhiên, thông thường chúng ta chỉ quan tâm đến các giá trị của một thuộc tính đơn. Hơn nữa, đôi lúc chúng ta có thể suy ra từ thông tin về khoá

hoặc từ các thông tin khác. Chẳng hạn, chúng ta có thể so sánh kết quả của một truy vấn con như vậy với một hằng hoặc một thuộc tính.

Ví dụ 17: Chúng ta muốn đưa ra Họđệm và Tên của các nhân viên trong đơn vị có tên là ‘Nghiêncứ’. Chúng ta cần truy vấn hai quan hệ: NHÂNVIÊN và ĐƠNVI. Bởi vì chỉ có quan hệ đầu có thông tin về Họđệm và Tên và chỉ có quan hệ thứ hai có các tên của đơn vị. Thông tin được liên kết bằng ‘MãSốĐV’.

Có nhiều cách khác nhau để nhìn vào truy vấn này. Chúng ta chỉ cần quan hệ ĐƠNVI để nhận được số MãSốĐV cho đơn vị có tên là ‘Nghiên cứu’. Mỗi khi chúng ta có nó, chúng ta có thể truy vấn quan hệ NHÂNVIÊN để tìm ra họ đệm và tên của các nhân viên trong đơn vị đó. Vấn đề đầu tiên là nhận được MãSốĐV. Chúng ta có thể viết nó như một truy vấn con và kết quả của nó sẽ là một giá trị đơn. Có thể sử dụng giá trị này trong truy vấn “chính” để đạt được kết quả mong muốn:

- 1) SELECT Họđệm, Tên
- 2) FROM NHÂNVIÊN
- 3) WHERE MãSốĐV =
- 4) (SELECT MãSốĐV
- 5) FROM ĐƠNVI
- 6) WHERE TênĐV = ‘Nghiêncứ’ ;

Các dòng từ 4) đến 6) là truy vấn con. Chỉ nhìn vào truy vấn con này chúng ta sẽ thấy rằng kết quả sẽ là một quan hệ có một thuộc tính là MãSốĐV và chúng ta hy vọng sẽ tìm thấy chỉ một bộ trong quan hệ này, giả sử đó là 5.

Khi đã thực hiện truy vấn con này, chúng ta có thể thực hiện các dòng từ 1) đến 3) của truy vấn trên như là giá trị 5 đã thay thế truy vấn con. Như vậy, truy vấn “chính” sẽ được thực hiện như là

```
SELECT Họđệm, Tên
FROM NHÂNVIÊN
WHERE MãSốĐV = 5 ;
```

Kết quả của truy vấn là Họđệm và Tên của các nhân viên của đơn vị có tên là ‘Nghiên cứu’.

### 1.3.2 Các điều kiện có bao hàm các quan hệ

Có một số các phép toán của SQL mà ta có thể áp dụng cho một quan hệ R và tạo ra một kết quả logic. Thông thường, quan hệ R sẽ là kết quả của một truy vấn con *select-from-where*. Một số các phép toán (EXISTS, IN, ALL, ANY) sẽ được giải thích ở đây dưới dạng đơn giản nhất của nó, trong đó có chứa một giá trị vô hướng s. Trong hoàn cảnh này, R phải là một quan hệ có một cột

1. EXISTS R là một điều kiện có giá trị true khi và chỉ khi R không rỗng.
2. s IN R có giá trị TRUE khi và chỉ khi s bằng một trong các giá trị của R. Tương tự, s NOT IN R có giá trị TRUE khi và chỉ khi s không bằng giá trị nào trong R. Ở đây, chúng ta giả thiết R là quan hệ có một cột.
3. s > ALL R là đúng khi và chỉ khi s lớn hơn mọi giá trị trong quan hệ một cột R. Tương tự, có thể thay dấu > bằng một trong các phép so sánh khác.
4. s > ANY R là đúng khi và chỉ khi s lớn hơn ít nhất là một giá trị trong quan hệ một cột R

Các phép toán EXISTS, IN, ALL và ANY có thể được phủ định bằng cách đặt NOT ở trước biểu thức. Ví dụ, NOT EXISTS R đúng khi và chỉ khi R là rỗng. NOT s > ALL R là đúng khi và chỉ khi s không phải là giá trị max trong R và NOT s > ANY R là đúng khi và chỉ khi s là giá trị min trong R.

### 1.3.3 Các điều kiện có bao hàm các bộ

Một bộ trong SQL được biểu diễn bằng một danh sách các giá trị vô hướng được đặt trong dấu ngoặc. Ví dụ: (5, 'Nghiên cứu', 'NV002', 2000-09-15) là một bộ của quan hệ ĐONVI.

Nếu một bộ t có cùng số thành phần như một quan hệ R thì có thể so sánh t và R. Ví dụ: t IN R hoặc t <> ANY R (Chú ý rằng khi so sánh một bộ với các thành phần của một quan hệ R, chúng ta phải so sánh các thành phần theo thứ tự của các thuộc tính của R).

Ví dụ 18: Giả sử ta muốn đưa ra Họđệm và tên của các nhân viên có lương cao nhất ở trong từng đơn vị. Ta viết truy vấn sau:

1) SELECT Họđệm, Tên

- 2) FROM NHÂNVIÊN
- 3) WHERE (Lương, Mã sốĐV) IN
- 4) (SELECT (MAX(Lương), Mã sốĐV
- 5) FROM NHÂNVIÊN
- 6) GROUP BY Mã sốĐV);

Truy vấn này bao gồm một truy vấn chính và một truy vấn con. Truy vấn con khảo sát các bộ của quan hệ NHÂNVIÊN và đưa ra một quan hệ gồm các bộ với giá trị của các thuộc tính MAX(Lương) và Mã sốĐV. Truy vấn chính, từ dòng 1) đến dòng 3) xem xét các bộ của quan hệ NHÂNVIÊN để tìm các bộ có thành phần Lương và Mã sốĐV là một trong các bộ do truy vấn con đưa ra. Với mỗi bộ như vậy, Họđệm và Tên sẽ được đưa ra và cho chúng ta tập hợp những nhân viên có lương cao nhất trong từng đơn vị như chúng ta mong muốn.

Kết quả:      Họđệm    Tên  
                  Trần Đức Nam  
                  Phạm    Bằng  
                  Hoàng    Giáp

#### 1.3.4 Các truy vấn con tương quan với nhau

Các truy vấn con đơn giản nhất có thể được thực hiện một lần cho tất cả, và kết quả được sử dụng trong truy vấn mức cao hơn. Việc sử dụng các truy vấn con lồng nhau phức tạp hơn đòi hỏi truy vấn con được thực hiện nhiều lần, mỗi lần đối với một phép gán giá trị cho một mục nào đó trong truy vấn con. Giá trị gán xuất phát từ một biến bộ ở bên ngoài truy vấn con. Một truy vấn con kiểu như vậy được gọi là một truy vấn con tương quan. Chúng ta bắt đầu bằng một ví dụ.

Ví dụ 19: Chúng ta muốn đưa ra Họđệm và Tên của các nhân viên có lương lớn hơn lương trung bình của đơn vị của họ. Chúng ta xem xét lần lượt các bộ của quan hệ NHÂNVIÊN, với mỗi bộ như vậy, chúng ta đòi hỏi trong một truy vấn con liệu giá trị của Lương có lớn hơn trung bình lương của đơn vị có mã số giống như giá trị của thành phần Mã sốĐV của bộ hay không. Toàn bộ truy vấn được viết như sau:

- 1) SELECT Họđệm, Tên
- 2) FROM NHÂNVIÊN NV
- 3) WHERE (Lương >
- 4) (SELECT (AVG(Lương)
- 5) FROM NHÂNVIÊN
- 6) WHERE Mẫ sốĐV = NV.Mẫ sốĐV;

Giống như với các truy vấn lồng nhau khác, chúng ta hãy bắt đầu từ truy vấn con trong cùng, các dòng từ 4) đến 6). Nếu NV.Mẫ sốĐV ở dòng 6 được thay bằng một số nguyên như là 5, chúng ta có thể hiểu nó hoàn toàn dễ dàng như là một truy vấn hỏi trung bình lương của đơn vị có mã số là 5. Truy vấn con hiện tại có khác một tí. Vấn đề là ở chỗ chúng ta không biết Mẫ sốĐV có giá trị như thế nào. Tuy nhiên, như chúng ta sắp xếp trên các bộ của truy vấn ngoài từ dòng 1) đến dòng 3), mỗi bộ cung cấp một giá trị của Mã sốĐV. Sau đó chúng ta thực hiện truy vấn từ dòng 4) đến dòng 6) với giá trị đó của Mã sốĐV để quyết định chân trị của mệnh đề WHERE trải trên các dòng từ 3) đến 6). Điều kiện của dòng 3) là đúng nếu có một đơn vị có trung bình lương nhỏ hơn lương của bộ đang xét.

Kết quả	<u>Họđệm</u>	<u>Tên</u>
	Trần Đức	Nam
	Nguyễn	Son
	Phạm	Bằng
	Hoàng	Giáp

Khi viết một truy vấn tương quan, điều quan trọng là chúng ta phải nhận thức được quy tắc phạm vi đối với các tên. Nói chung, một thuộc tính trong một truy vấn con thuộc về một trong các biến bộ của mệnh đề FROM của truy vấn con đó nếu một quan hệ nào đó của biến bộ có thuộc tính đó trong lược đồ của nó. Nếu không, chúng ta tìm ở truy vấn trực tiếp ngay bên ngoài ....

Tuy nhiên, chúng ta có thể sắp xếp cho một thuộc tính thuộc về một biến bộ khác nếu chúng ta viết trước nó một biến bộ và một dấu chấm. Điều đó là vì chúng ta đã đưa ra bí danh NV cho quan hệ NHÂNVIÊN của truy vấn ngoài và vì chúng ta tham chiếu đến NV.Mẫ sốĐV trong dòng 6). Chú ý rằng nếu

hai quan hệ trong các mệnh đề FROM của các dòng 2) và 5) là khác nhau, chúng ta không cần đến bí danh. Dĩ nhiên, trong truy vấn con, chúng ta có thể tham chiếu trực tiếp đến các thuộc tính của quan hệ được chỉ ra trong dòng 2).

### 1.3.5 Các truy vấn con trong mệnh đề FROM

Một cách dùng khác đối với các truy vấn con là như các quan hệ trong một mệnh đề FROM. Trong danh sách FROM, thay cho một quan hệ được lưu giữ, chúng ta có thể sử dụng một truy vấn con để trong dấu ngoặc. Bởi vì chúng ta không có tên cho kết quả của một truy vấn con, chúng ta phải cho nó một bí danh biến bộ. Sau đó chúng ta tham chiếu đến các bộ trong kết quả của một truy vấn con như chúng ta sẽ tham chiếu đến các bộ trong một quan hệ bất kỳ xuất hiện trong danh sách FROM.

Ví dụ 20: Chúng ta hãy xem lại vấn đề của ví dụ 18, ở đó chúng ta đã viết một truy vấn tìm Họđệm và Tên của các nhân viên có lương cao nhất trong các đơn vị. Giả sử rằng chúng ta đã có một quan hệ chứa Max(lương) và MãSốĐV. Khi đó việc tìm các Họđệm và Tên sẽ đơn giản hơn bằng cách tìm trong quan hệ NHÂNVIÊN. Truy vấn như vậy có dạng

- 1) SELECT Họđệm, Tên
- 2) FROM NHÂNVIÊN, (SELECT MAX(Lương), MãSốĐV)
- 3) FROM NHÂNVIÊN
- 4) GROUP BY MãSốĐV) NV1
- 5) WHERE (Lương, MãSốĐV) = NV1.(MAX(Lương),MãSốĐV);

Các dòng từ 2) đến 4) là mệnh đề FROM của truy vấn ngoài. Trong mệnh đề đó, ngoài quan hệ NHÂNVIÊN nó còn có một truy vấn con. Truy vấn này đưa ra quan hệ có hai thuộc tính là MAX(Lương) và MãSốĐV. Tập hợp đó được gán cho bí danh là NV1 ở dòng 4. Ở dòng 5), các quan hệ NHÂNVIÊN và truy vấn con có bí danh NV1 được nối theo hai điều kiện là Lương và MãSốĐV phải như nhau. Kết quả là đưa ra được tập Họđệm và Tên giống như trong ví dụ 18.

### 1.3.6 Các biểu thức nối của SQL

Chúng ta có thể xây dựng các quan hệ bằng các phép nối khác nhau áp dụng trên hai quan hệ. Các cách này gồm tích, nối tự nhiên, nối teta, và nối ngoài. Kết quả có thể là một truy vấn. Vì các biểu thức này tạo ra các quan hệ nên chúng có thể được sử dụng như các truy vấn con trong mệnh đề FROM của một biểu thức select-from-where.

Dạng đơn giản nhất của một biểu thức nối là nối chéo (cross join). Thuật ngữ này đồng nghĩa với tích Đềcac hoặc tích. Ví dụ, nếu chúng ta muốn có tích Đềcac của hai quan hệ NHÂNVIÊN và ĐƠNVI. Chúng ta có thể nói

NHÂNVIÊN CROSS JOIN ĐƠNVI ;

và kết quả sẽ là một quan hệ có 13 cột, chứa tất cả các thuộc tính của NHÂNVIÊN và ĐƠNVI. Mỗi một cặp gồm một bộ của NHÂNVIÊN một bộ của ĐƠNVI sẽ là một bộ của quan hệ kết quả.

Các thuộc tính trong quan hệ tích có thể được gọi là R.A, trong đó R là một trong hai quan hệ nối và A là một trong các thuộc tính của nó. Nếu chỉ có một trong các quan hệ có thuộc tính là A thì có thể bỏ R và dấu chấm đi. Trong hoàn cảnh hiện tại, bởi vì quan hệ NHÂNVIÊN và quan hệ ĐƠNVI có một thuộc tính chung là MãSốĐV, nên ở trong quan hệ tích cần phải phân biệt chúng NHÂNVIÊN.MãSốĐV và ĐƠNVI.MãSốĐV, các tên khác của các thuộc tính trong hai quan hệ là khác nhau nên không cần có tên quan hệ và dấu chấm ở trước. Tuy nhiên, phép toán tích là một phép toán ít khi được sử dụng. Phép nối teta là thuận tiện hơn. Phép nối này gồm từ khoá JOIN được đặt giữa hai tên quan hệ R và S, sau chúng là từ khoá ON và một điều kiện. Ý nghĩa của JOIN ...ON ... là phép tính tích  $R \times S$ , sau đó là một phép chọn theo điều kiện đi sau ON.

Ví dụ 21: Giả sử chúng ta muốn nối hai quan hệ NHÂNVIÊN và ĐƠNVI với điều kiện là các bộ được nối là các bộ tham chiếu đến cùng một mã số đơn vị. Như vậy, các mã số đơn vị từ cả hai quan hệ phải như nhau. Chúng ta có thể đòi hỏi truy vấn này là:

NHÂNVIÊN JOIN ĐƠNVI ON NHÂNVIÊN.MãSốĐV = ĐƠNVI.MãSốĐV;

Kết quả lại là một quan hệ với 13 cột cùng với các tên thuộc tính như trên. Tuy nhiên, bây giờ một bộ từ NHÂNVIÊN và một bộ từ ĐƠNVI kết hợp với nhau để tạo thành một bộ kết quả chỉ khi hai bộ có mã số đơn vị như nhau. Trong kết quả, một cột là thừa bởi vì mỗi một bộ của kết quả sẽ cùng giá trị trong cả hai thành phần MãSốĐV

Nếu chúng ta lo lắng với sự kiện là phép nối ở trên có một thành phần thừa, chúng ta có thể sử dụng biểu thức đầy đủ như là một truy vấn con trong mệnh đề FROM và sử dụng mệnh đề SELECT để loại bỏ các thuộc tính không mong muốn. Như vậy, chúng ta có thể viết:

```
SELECT <danh sách các thuộc tính trong hai quan hệ nhưng thuộc tính MãSốĐV chỉ xuất hiện một lần>
```

```
FROM NHÂNVIÊN JOIN ĐƠNVI ON NHÂNVIÊN.MãSốĐV = ĐƠNVI.MãSốĐV;
```

để nhận được một quan hệ có 12 cột, đó là các bộ của quan hệ NHÂNVIÊN được mở rộng thêm các bộ của ĐƠNVI

### 1.3.7 Nối tự nhiên (Natural Join)

Phép nối tự nhiên khác với phép nối teta ở chỗ:

1. Điều kiện nối là tất cả các cặp thuộc tính từ hai quan hệ có một tên chung được so sánh bằng và không có điều kiện nào khác.
2. Một thuộc tính trong mỗi cặp thuộc tính được so sánh bằng được chiếu ra ngoài. (nghĩa là trong quan hệ kết quả không có hai cột giống nhau).

Phép nối tự nhiên của SQL ứng xử một cách chính xác theo cách đó. Các từ khoá NATURAL JOIN xuất hiện giữa các quan hệ để biểu thị phép nối.

Ví dụ 22: Giả sử chúng ta muốn làm phép nối tự nhiên của hai quan hệ ĐƠNVI và NHÂNVIÊN. Kết quả sẽ là một quan hệ có lược đồ chứa thuộc tính MãSốĐV cộng với tất cả các thuộc tính xuất hiện trong cả hai quan hệ.  
Biểu thức

```
NHÂNVIÊN NATURAL JOIN ĐƠNVI
```

Mô tả súc tích quan hệ mong muốn.



### 1.3.8 Nói ngoài

Nói ngoài là một cách để làm tăng kết quả của một phép nối bằng các bộ treo, độn thêm vào các giá trị null. Trong SQL, chúng ta có thể chỉ rõ một nói ngoài; NULL được sử dụng như là giá trị null.

Ví dụ 23: Giả sử chúng ta đưa ra Họđệm và Tên của các nhân viên cũng như Họđệm và Tên của những người giám sát họ. Trên thực tế, không phải nhân viên nào cũng có người giám sát trực tiếp, vì vậy đối với những người không có người giám sát trực tiếp hoặc thông tin về người giám sát của họ là không xác định (null). Nếu muốn hiển thị cả những bộ như vậy, ta sử dụng nói ngoài

SQL xem nói ngoài chuẩn độn thêm vào các bộ treo từ hai phía của các đối số của chúng là nói ngoài đầy đủ (full outerjoin). Cú pháp như sau:

```
NHÂNVIÊN FULL OUTER JOIN NHÂNVIÊN ON MăsốNV = MăsốNGS;
```

Kết quả của phép toán này là một quan hệ, trong đó có những bộ được độn vào các giá trị NULL do không có giá trị nối tương ứng. (Chú ý, trong phép nối bình thường không có những bộ như vậy). Tất cả các loại nói ngoài được nói đến trong các phép toán đại số quan hệ đều có sẵn trong SQL. Nếu chúng ta muốn một left- hoặc right-outerjoin, ta thêm vào từ LEFT hoặc RIGHT thích hợp vào vị trí của từ FULL. Ví dụ:

```
NHÂNVIÊN LEFT OUTER JOIN NHÂNVIÊN ON MăsốNV = MăsốNGS;  
NHÂNVIÊN RIGHT OUTER JOIN NHÂNVIÊN ON MăsốNV = MăsốNGS;
```

Tiếp theo, giả sử ta muốn một nói ngoài tự nhiên thay vì một nói ngoài teta. Khi đó chúng ta sẽ sử dụng từ khóa NATURAL đặt vào trước từ JOIN và bỏ ON đi.

Ví dụ 24: Chúng ta hãy xem lại ví dụ 22, ở đó chúng ta muốn nối hai quan hệ NHÂNVIÊN và ĐƠNVI với điều kiện là các thuộc tính MăsốĐV của hai quan hệ là bằng nhau. Nếu chúng ta sửa đổi ví dụ này như sau

```
NHÂNVIÊN NATURAL FULL OUTER JOIN ĐƠNVI
```

thì chúng ta sẽ nhận được không chỉ là các bộ được tạo nên từ các bộ tham gia nối mà còn có thêm các bộ được độn vào các giá trị NULL

Từ khoá FULL có thể được thay thế bằng LEFT hoặc RIGHT trong phép nối ngoài ở trên.

## 1.4 CÁC PHÉP TOÁN QUAN HỆ ĐẦY ĐỦ

Trước tiên chúng ta để ý rằng SQL sử dụng các quan hệ như là các túi (bag) chứ không phải như tập hợp. Điều đó có nghĩa là một bộ có thể xuất hiện nhiều lần trong một quan hệ.

### 1.4.1 Loại bỏ trùng lặp

Như đã nói đến ở trên, khái niệm quan hệ của SQL khác với khái niệm quan hệ trừu tượng được trình bày trong mô hình quan hệ. Một quan hệ là một tập hợp, không thể có nhiều hơn một bản sao của một bộ cho trước. Khi một truy vấn SQL tạo một quan hệ mới, hệ thống SQL không loại bỏ các trùng lặp. Như vậy, SQL trả lời cho một truy vấn có thể liệt kê nhiều lần cùng một bộ.

Nhớ lại rằng một định nghĩa cho một truy vấn select-from-where của SQL là như sau: Chúng ta bắt đầu với tích Đềcax của các quan hệ được tham chiếu đến trong mệnh đề FROM. Mỗi bộ của tích được kiểm tra bằng điều kiện trong mệnh đề WHERE và những bộ nào qua được kiểm tra sẽ được đưa cho dữ liệu ra cho phép chiếu phù hợp với mệnh đề SELECT. Phép chiếu này có thể sinh ra cùng một bộ cho kết quả từ nhiều bộ khác nhau của tích, và nếu như vậy, mỗi bản sao của kết quả sẽ được in ra. Hơn nữa, không có gì sai đối với một quan hệ SQL có trùng lặp.

Nếu chúng ta không muốn có sự trùng lặp trong kết quả, ta có thể tiếp theo sau từ khoá SELECT bằng từ khoá DISTINCT. Từ đó nói với SQL chỉ tạo ra một bản sao cho một bộ giá trị. Chẳng hạn

```
SELECT DISTINCT Lương  
FROM NHÂNVIÊN ;
```

### 1.4.2 Trùng lặp trong phép hợp, phép giao và phép trừ

Không giống như lệnh SELECT giữ gìn sự trùng lặp như mặc định và chỉ loại bỏ chúng khi đưa vào từ khoá DISTINCT, các phép toán hợp, giao và trừ thường loại bỏ sự trùng lặp. Như vậy, các túi được đổi thành tập hợp và

bản tập hợp của phép toán được áp dụng. Để ngăn ngừa việc loại bỏ trùng lặp, chúng ta phải viết sau các phép toán UNION, INTERSECT, EXCEPT từ khoá ALL. Nếu chúng ta làm như vậy thì chúng ta nhận được cú pháp túi thông qua các phép toán này.

Ví dụ 26: Xét biểu thức hợp của ví dụ 14 nhưng có thêm vào từ khoá ALL:

```
(SELECT MăsốNV
FROM NHÂNVIÊN_DỰÁN
WHERE MăsốDA = 1)
UNION ALL
(SELECT MăsốNV
FROM NHÂNVIÊN_DỰÁN
WHERE MăsốDA = 2)
```

Kết quả	<u>MăsốNV</u>
	NV001
	NV001
	NV002
	NV018
	NV018

Ta thấy bây giờ trong kết quả xuất hiện các bộ trùng nhau. Nếu một nhân viên làm việc cho cả dự án 1 và dự án 2 thì mã số của nhân viên đó xuất hiện trong kết quả hai lần.

Cũng như đối với UNION, các phép toán INTERSECT ALL và EXCEPT ALL là giao và trừ của các túi (bag). Như vậy, nếu R và S là các quan hệ thì kết quả của biểu thức

```
R INTERSECT ALL S
```

là một quan hệ trong đó số lần xuất hiện của một bộ t là số nhỏ nhất của số lần xuất hiện của bộ đó trong R và số lần xuất hiện của bộ đó trong S.

Kết quả của biểu thức

```
R EXCEPT ALL S
```

là một quan hệ trong đó số lần xuất hiện bộ t bằng số lần xuất hiện của nó trong R trừ đi số lần xuất hiện của nó trong S với điều kiện hiệu số này là dương.

### 1.4.3 Nhóm và sự kết hợp trong SQL

Phép toán nhóm và kết hợp trong đại số quan hệ cho phép ta phân chia các bộ của một quan hệ thành các nhóm dựa trên các giá trị của một hoặc nhiều thuộc tính trong các bộ. Sau đó chúng ta có thể kết hợp một số các cột khác của quan hệ bằng cách áp dụng phép toán kết hợp đối với các cột đó. Nếu có các nhóm thì phép kết hợp sẽ được thực hiện riêng rẽ cho từng nhóm. SQL cung cấp mọi khả năng của phép toán trên thông qua việc sử dụng các phép toán nhóm trong mệnh đề SELECT và một mệnh đề GROUP BY đặc biệt.

### 1.4.4 Các phép toán nhóm

SQL sử dụng 5 phép toán nhóm SUM, AVG, MIN, MAX, và COUNT. Các phép toán này được sử dụng bằng cách áp dụng chúng cho các biểu thức có giá trị vô hướng, thường là một tên cột, ở trong mệnh đề SELECT. Có một ngoại lệ là biểu thức COUNT(\*), biểu thức này đếm tất cả các bộ trong một quan hệ được thiết lập từ mệnh đề FROM và mệnh đề WHERE của truy vấn.

Hơn nữa, chúng ta có tùy chọn loại trừ trùng lặp ra khỏi cột trước khi áp dụng phép toán nhóm bằng việc sử dụng từ khoá DISTINCT. Như vậy, một biểu thức như là COUNT(DISTINCT x) đếm số các giá trị khác nhau trong cột x. Chúng ta có thể sử dụng các phép toán khác ở vị trí của COUNT ở đây nhưng biểu thức như SUM(DISTINCT x) thường không có ý nghĩa mấy, bởi vì nó yêu cầu ta tính tổng các giá trị khác nhau trong cột x.

Ví dụ 27: Truy vấn sau đây tìm giá trị lương trung bình của tất cả các nhân viên:

```
SELECT AVG(Luong)
FROM NHÂNVIÊN ;
```

Chú ý rằng ở đây không có mệnh đề WHERE. Truy vấn này xem xét cột Lương của quan hệ NHÂNVIÊN, tính tổng các giá trị tìm được ở đây, một giá trị cho mỗi bộ (cho dù nếu bộ là trùng lặp của một vài bộ khác), và chia tổng số cho số các bộ. Nếu không có các bộ trùng lặp thì truy vấn này cho

lương trung bình như chúng ta mong đợi. Nếu có các bộ trùng lặp, thì một giá trị lương trùng lặp n lần sẽ được tính n lần trong trung bình.

Ví dụ 28:

Truy vấn sau đây:

```
SELECT COUNT(*)  
FROM NHÂNVIÊN ;
```

đếm số các bộ trong quan hệ NHÂNVIÊN.

Truy vấn tương tự:

```
SELECT COUNT(Lương)  
FROM NHÂNVIÊN ;
```

đếm số giá trị trong cột Lương của quan hệ. Bởi vì các giá trị trùng lặp không bị loại bỏ khi chúng ta chiếu lên cột Lương trong SQL, tổng đếm này sẽ giống như tổng đếm do truy vấn với COUNT(\*) sinh ra.

Nếu chúng ta muốn chắc chắn rằng ta không đếm các giá trị trùng lặp quá một lần, chúng ta có thể sử dụng từ khoá DISTINCT trước thuộc tính nhóm, như:

```
SELECT COUNT(DISTINCT Lương)  
FROM NHÂNVIÊN ;
```

Bây giờ mỗi lương sẽ được đếm một lần, không cần quan tâm đến việc nó xuất hiện trong bao nhiêu bộ.

#### **1.4.5 Nhóm**

Để nhóm các bộ, chúng ta sử dụng mệnh đề GROUP BY, đi sau mệnh đề WHERE. Theo sau từ khoá GROUP BY là một danh sách các thuộc tính nhóm. Trong hoàn cảnh đơn giản nhất, chỉ có một tham chiếu quan hệ trong mệnh đề FROM, và quan hệ này có các bộ của nó được nhóm theo theo các giá trị của chúng trong các thuộc tính nhóm. Dù phép toán nhóm nào được sử dụng trong mệnh đề SELECT cũng chỉ được áp dụng bên trong các nhóm.

Ví dụ 29: Vấn đề tìm trong quan hệ NHÂNVIÊN tổng lương theo từng đơn vị:

```
SELECT MẫsốĐV, SUM(Lương)
FROM NHÂNVIÊN
GROUP BY MẫsốĐV ;
```

Chúng ta có thể tưởng tượng là các bộ của quan hệ NHÂNVIÊN được sắp xếp lại và được nhóm sao cho tất các các bộ đối với đơn vị 1 là cùng với nhau, tất cả các bộ của đơn vị 4 là cùng với nhau, .... Các tổng của các thành phần Lương của các bộ trong từng nhóm được tính toán, MẫsốĐV được đưa ra cùng với tổng đó.

Quan sát ví dụ 29 ta thấy mệnh đề SELECT có hai loại số hạng:

1. Các kết hợp, ở đó một phép toán nhóm được áp dụng cho một thuộc tính hoặc một biểu thức bao gồm các thuộc tính. Như đã đề cập đến, các số hạng này được tính giá trị trên cơ sở từng nhóm. Trong ví dụ này, SUM(Lương) là một kết hợp.
2. Các thuộc tính, chẳng hạn như MẫsốĐV trong ví dụ này, xuất hiện trong mệnh đề GROUP BY. Trong một mệnh đề SELECT có các phép toán nhóm, chỉ những thuộc tính nào được đề cập đến trong mệnh đề GROUP BY mới có thể xuất hiện như các thuộc tính không nhóm trong mệnh đề SELECT.

Khi các truy vấn có chứa GROUP BY nói chung có cả các thuộc tính nhóm và sự kết hợp trong mệnh đề SELECT, về mặt kỹ thuật không cần thiết có mặt cả hai. Ví dụ, chúng ta có thể viết:

```
SELECT MẫsốĐV
FROM NHÂNVIÊN
GROUP BY MẫsốĐV;
```

Truy vấn này sẽ nhóm các bộ của NHÂNVIÊN theo mã số đơn vị của nó và sau đó in ra mã số đơn vị cho mỗi nhóm, không cần quan tâm đến có bao nhiêu bộ có cùng mã số đơn vị. Như vậy, truy vấn ở trên có cùng kết quả như

```
SELECT DISTINCT MãSốĐV  
FROM NHÂNVIÊN ;
```

Có thể sử dụng mệnh đề GROUP BY trong một truy vấn với nhiều quan hệ. Các truy vấn như vậy được thể hiện bằng dãy các bước sau đây:

1. Tính quan hệ R được biểu diễn bằng các mệnh đề FROM và WHERE. Như vậy, quan hệ R là tích Đềcac của các quan hệ được chỉ ra trong mệnh đề FROM và áp dụng phép chọn của mệnh đề WHERE đối với nó.
2. Nhóm các bộ của R theo các thuộc tính trong mệnh đề GROUP BY.
3. Kết quả là các thuộc tính và các kết hợp của mệnh đề SELECT được tạo ra cứ như là truy vấn trên một quan hệ được lưu trữ R.

Ví dụ 30: Giả sử chúng ta muốn đưa ra tên đơn vị và số lượng các nhân viên trong từng đơn vị. Chúng ta cần lấy thông tin từ hai quan hệ: NHÂNVIÊN và ĐƠNVI. Chúng ta bắt đầu bằng cách nối teta chúng bằng cách so sánh bằng các mã số đơn vị từ hai quan hệ. Bước này cho chúng ta một quan hệ mà trong đó mỗi bộ ĐƠNVI được cặp với các bộ NHÂNVIÊN có mã số đơn vị giống với mã số đơn vị của nó. Bây giờ, chúng ta có thể nhóm các bộ được chọn của quan hệ này theo tên của đơn vị. Cuối cùng, chúng ta đếm số các nhân viên trong từng nhóm. Truy vấn được viết như sau:

```
SELECT TênĐV, COUNT(*)  
FROM NHÂNVIÊN NV, ĐƠNVI ĐV  
WHERE NV.MãSốĐV = ĐV.MãSốĐV  
GROUP BY TênĐV ;
```

Kết quả	<u>TênĐV</u>	<u>COUNT(*)</u>
	Nghiên cứu	4
	Hành chính	3
	Lãnh đạo	1

### 1.4.6 Các mệnh đề HAVING

Giả sử rằng chúng ta không muốn tính đến tất cả các tên đơn vị trong bảng của chúng ta ở ví dụ 30 ở trên. Chúng ta có thể hạn chế các bộ trước khi nhóm theo cách có thể làm rộng các nhóm không mong muốn. Ví dụ, nếu chúng ta chỉ muốn số các nhân viên của một đơn vị phải lớn hơn hoặc bằng 3. Khi đó, chúng ta tiếp theo mệnh đề GROUP BY một mệnh đề HAVING. Mệnh đề HAVING bao gồm từ khoá HAVING theo sau là một điều kiện về nhóm.

Ví dụ 31: Giả sử chúng ta muốn in ra Tên đơn vị và số nhân viên trong từng đơn vị đối với những đơn vị nào có nhiều hơn hoặc bằng 3 nhân viên. Chúng ta có thể thêm vào ví dụ 30 mệnh đề

HAVING COUNT(\*) >= 3 ;

Truy vấn kết quả được cho như dưới đây:

Kết quả	<u>TênĐV</u>	<u>COUNT(*)</u>
	Nghiên cứu	4
	Hành chính	3

Chúng ta phải nhớ một số quy tắc về các mệnh đề HAVING:

. Một phép nhóm trong mệnh đề HAVING chỉ áp dụng đối với các bộ của nhóm đã được kiểm tra.

. Bất kỳ thuộc tính nào của các quan hệ trong mệnh đề FROM đều có thể được nhóm trong mệnh đề HAVING, nhưng chỉ có các thuộc tính có thể xuất hiện trong danh sách GROUP BY không được nhóm trong mệnh đề HAVING (cùng quy tắc như với mệnh đề SELECT).

Một số điều cần nhớ:

\* Thứ tự của các mệnh đề trong các truy vấn SQL:

Cho đến bây giờ chúng ta đã gặp tất cả sáu mệnh đề trong một truy vấn “select-from-where”: SELECT, FROM, WHERE, GROUP BY, HAVING và ORDER BY. Chỉ có hai mệnh đề đầu là bắt buộc, nhưng ta không thể sử



dụng mệnh đề HAVING mà không có mệnh đề GROUP BY. Bất kỳ mệnh đề phụ thêm nào cũng phải xuất hiện theo thứ tự ở trên.

\* Nhóm, Tập hợp và Null:

Khi các bộ có giá trị Null, cần nhớ một số quy tắc sau:

- Giá trị Null được lờ đi trong tập hợp. Nó không góp phần vào **sum**, **average**, hoặc **count** hoặc không là **min** hoặc **max** trong cột của chúng. Ví dụ, COUNT(\*) luôn luôn là một phép đếm của số các bộ trong một quan hệ, nhưng COUNT(A) là số các bộ với giá trị của thuộc tính A không Null.
- Mặt khác, NULL được xử lý như là một giá trị thông thường trong một thuộc tính nhóm. Ví dụ, SELECT a, AVG(b) FROM R sẽ tạo ra một bộ với NULL cho giá trị của a và giá trị trung bình của b đối với các bộ với a =NULL, nếu có ít nhất một bộ trong R với thành phần a là NULL.

## 1.5 SỬA ĐỔI CƠ SỞ DỮ LIỆU

Ngoài dạng truy vấn SQL chuẩn select-from-where, có một số các dạng lệnh khác không trả lại một kết quả nhưng làm thay đổi trạng thái của quan hệ. Trong phần này chúng ta sẽ hướng đến các dạng lệnh cho phép ta

- Chèn các bộ vào một quan hệ
- Xoá một số bộ ra khỏi quan hệ
- Cập nhật các giá trị của một số thành phần của một số bộ đã tồn tại.

Các phép toán như vậy gọi là các phép toán sửa đổi.

### 1.5.1 Chèn

Dạng cơ bản của lệnh chèn bao gồm:

1. Từ khoá INSERT TO
2. Tên của một quan hệ R
3. Một danh sách các thuộc tính của quan hệ R đặt trong dấu ngoặc
4. Từ khoá VALUES

5. Một biểu thức bộ, nghĩa là một danh sách các giá trị cụ thể được đặt trong dấu ngoặc, một giá trị cho mỗi thuộc tính ở trong danh sách ở điểm 3.

Như vậy, dạng chèn cơ bản là

```
INSERT INTO R(A1,A2,...,An) VALUES (v1,v2,...,vn)
```

Một bộ được tạo ra bằng cách sử dụng giá trị  $v_i$  cho thuộc tính  $A_i$ , với  $i = 1, 2, \dots, n$ . Nếu danh sách của các thuộc tính không bao hàm tất cả các thuộc tính của quan hệ  $R$  thì bộ được tạo ra có các giá trị ngầm định cho tất cả các thuộc tính bị thiếu. Giá trị ngầm định hay dùng là NULL, nhưng cũng có thể có các tùy chọn khác.

Ví dụ 32:

- 1) INSERT INTO ĐƠN VỊ (Mã số ĐV, Tên ĐV, Mã số NQL, Ngày bắt đầu)
- 2) VALUES (8, 'Kế hoạch', 'NV018', '1977-16-24');

Kết quả của việc thực hiện lệnh này là một bộ với bốn thành phần ở dòng 2) sẽ được chèn vào quan hệ ĐƠN VỊ. Vì tất cả các thuộc tính của ĐƠN VỊ đã được kể đến ở dòng 1) nên không cần phải thêm vào các thành phần ngầm định. Các giá trị trên dòng 2) phù hợp với các thuộc tính trên dòng 1) theo thứ tự cho trước, như vậy, 'Kế hoạch' trở thành giá trị của thuộc tính Tên ĐV.

Nếu chúng ta cung cấp tất cả các giá trị cho tất cả các thuộc tính của quan hệ thì chúng ta có thể bỏ qua danh sách các thuộc tính đi theo sau tên quan hệ. Ví dụ, chúng ta có thể viết lại lệnh trên dưới dạng

```
INSERT INTO ĐƠN VỊ  
VALUES (8, 'Kế hoạch', 'NV018', '1977-16-24');
```

Tuy nhiên, nếu chúng ta lấy tùy chọn này thì ta phải chắc chắn rằng thứ tự của các giá trị là giống như thứ tự chuẩn của các thuộc tính đối với quan hệ. Nếu ta không chắc chắn về thứ tự chuẩn đối với các thuộc tính thì tốt nhất là liệt kê chúng trong mệnh đề INSERT để ta chọn cho các giá trị của nó trong mệnh đề VALUES.

Lệnh INSERT đơn giản được mô tả ở trên chỉ đặt một bộ vào một quan hệ. Thay vì sử dụng các giá trị rõ cho một bộ, chúng ta có thể tính toán một tập hợp các bộ được chèn vào bằng cách sử dụng một truy vấn con. Truy vấn con này thay thế từ khoá VALUES và biểu thức bộ trong dạng lệnh INSERT được mô tả ở trên.

Ví dụ 33: Giả sử chúng ta có một quan hệ ĐƠNVI1 chứa các bộ giá trị cùng kiểu với các bộ trong quan hệ ĐƠNVI. Nếu muốn chèn tất cả các bộ của ĐƠNVI1 vào quan hệ ĐƠNVI ta viết lệnh chèn như sau:

- 1) INSERT INTO ĐƠNVI
- 2)       SELECT \*
- 3)       FROM ĐƠNVI1 ;

### 1.5.2 Xóa

Một lệnh xóa bao gồm

1. Các từ khoá DELETE FROM
2. Tên của một quan hệ, R
3. Từ khoá WHERE và
4. một điều kiện.

Như vậy, dạng của phép xóa là

```
DELETE FROM R WHERE < điều kiện > ;
```

Hậu quả của việc thực hiện lệnh này là mỗi bộ thoả mãn điều kiện sẽ bị xóa khỏi quan hệ.

Ví dụ 34: Chúng ta có thể xóa khỏi quan hệ ĐƠNVI bộ giá trị (8, 'Kếhoạch', 'NV018', '1977-16-24') bằng lệnh SQL sau đây:

```
DELETE FROM ĐƠNVI  
WHERE TênĐV = 'Kếhoạch';
```

Chú ý rằng không giống như lệnh chèn ở ví dụ 32, chúng ta không thể chỉ ra một cách đơn giản một bộ sẽ bị xoá. Đúng hơn, chúng ta phải mô tả bộ một cách chính xác bằng mệnh đề WHERE.

Ví dụ 35 Lệnh sau đây

```
DELETE FROM NHÂNVIÊN
```

```
WHERE Lương < 3000 ;
```

sẽ xoá khỏi quan hệ NHÂNVIÊN tất cả các bộ giá trị có thành phần Lương nhỏ hơn 3000.

### 1.5.3 Cập nhật

Lệnh update trong SQL làm thay đổi một số thành phần của các bộ đã tồn tại trong cơ sở dữ liệu. Dạng tổng quát của lệnh update là:

1. Từ khoá UPDATE
2. Một tên quan hệ, R
3. Từ khoá SET
4. Một danh sách các công thức, mỗi công thức đặt một thuộc tính của quan hệ R bằng một giá trị của một biểu thức hoặc một hằng.
5. Từ khoá WHERE và
6. Một điều kiện.

Như vậy dạng lệnh update là

```
UPDATE R SET < gán giá trị mới > WHERE < điều kiện > ;
```

Mỗi một < gán giá trị mới > là một thuộc tính, một dấu bằng và một công thức. Nếu có nhiều hơn một phép gán thì chúng được phân cách nhau bằng dấu chấm phẩy (;).

Hậu quả của lệnh này là tìm tất cả các bộ giá trị trong R thoả mãn điều kiện. Mỗi bộ này sau đó sẽ được thay đổi bằng cách tính giá trị các công thức và gán cho các thành phần của bộ với các thuộc tính tương ứng của R.

Ví dụ 36: Hãy sửa đổi TênSV của các bộ trong quan hệ ĐƠNVI có tên đơn vị là ‘Hànhchính’ thành tên mới là ‘Kếhoạch’. Ta viết lệnh UPDATE như sau:

- 1) UPDATE ĐƠNVI
- 2) SET TênĐV = ‘Kếhoạch’
- 3) WHERE TênĐV = ‘Hànhchính’ ;

Dòng 3) kiểm tra rằng có phải tên đơn vị là ‘Hànhchính’ hay không. Nếu đúng, dòng 2) sẽ thay thế tên này bằng ‘Kếhoạch’.

## 1.6 ĐỊNH NGHĨA MỘT LƯỢC ĐỒ QUAN HỆ TRONG SQL

Trong phần này chúng ta sẽ thảo luận về định nghĩa dữ liệu, một phần của ngôn ngữ SQL cho phép mô tả các cấu trúc thông tin trong cơ sở dữ liệu. Ngược lại, các khía cạnh SQL thảo luận trước kia - các truy vấn và cập nhật-thường được gọi là thao tác dữ liệu.

Chủ đề của phần này là mô tả các lược đồ của các quan hệ được lưu giữ. Chúng ta sẽ thấy mô tả một quan hệ (bảng) mới, một khung nhìn như thế nào.

### 1.6.1 Các kiểu dữ liệu

Trước tiên chúng ta đưa ra các kiểu dữ liệu nguyên tử được hệ thống SQL hỗ trợ. Mọi thuộc tính phải có một kiểu dữ liệu.

1. Các xâu ký tự có độ dài thay đổi hoặc cố định. Kiểu CHAR(n) ký hiệu một xâu gồm n ký tự. Như vậy, nếu một thuộc tính có kiểu CHAR(n) thì trong một bộ bất kỳ, thành phần cho thuộc tính này là một xâu gồm n ký tự. VARCHAR(n) ký hiệu một xâu gồm nhiều nhất là n ký tự. Các thành phần cho các thuộc tính thuộc kiểu này sẽ là một xâu có từ 0 đến n ký tự. SQL cho phép các ép buộc hợp lý giữa các giá trị của các kiểu xâu ký tự. Thường thường, các xâu có độ dài cố định được thêm vào các dấu khoảng trống nếu giá trị của nó nhỏ hơn độ dài cho phép. Khi so sánh một xâu với một xâu khác, các dấu trống thêm vào sẽ được bỏ qua.

2. Các chuỗi bit có độ dài cố định hoặc thay đổi. Các chuỗi này tương tự như các chuỗi ký tự có độ dài cố định hoặc thay đổi, nhưng giá trị của chúng là các chuỗi bit. BIT(n) ký hiệu các chuỗi bit có độ dài n, trong khi đó BIT VARYING(n) ký hiệu các chuỗi bit có độ dài nhỏ hơn hoặc bằng n.
3. Kiểu BOOLEAN ký hiệu các thuộc tính có giá trị lô gic. Các giá trị có thể có của thuộc tính thuộc loại này là TRUE, FALSE và UNKNOWN.
4. Kiểu INT hoặc INTEGER ký hiệu các giá trị nguyên. Kiểu SHORTINT cũng ký hiệu các giá trị nguyên nhưng có số các chữ số ít hơn.
5. Các số dấu phẩy động có thể biểu diễn bằng nhiều cách. Chúng ta sử dụng kiểu REAL hoặc FLOAT (hai kiểu này cùng nghĩa) đối với các số dấu phẩy động. Độ chính xác cao hơn có thể nhận được với kiểu DOUBLE PRECISION. SQL cũng có các kiểu với các số thực dấu phẩy cố định. Đó là kiểu DECIMAL(n,d) cho phép các giá trị chứa n chữ số thập phân, với vị trí của dấu chấm thập phân được giả thiết là vị trí d kể từ bên phải sang. Kiểu NUMERIC hầu như đồng nghĩa với DECIMAL.
6. Ngày và giờ cũng có thể được biểu diễn nhờ các kiểu DATE và TIME. Các giá trị của chúng là các chuỗi ký tự dạng đặc biệt. Thực vậy, chúng ta có thể biến đổi ngày và giờ thành kiểu chuỗi và ngược lại.

### 1.6.2 Các khai báo bảng đơn giản

Dạng đơn giản nhất của khai báo một lược đồ quan hệ bao gồm các từ khoá CREATE TABLE sau đó là tên của quan hệ và một danh sách các thuộc tính cùng với kiểu của chúng được đặt trong dấu ngoặc.

Ví dụ 1.37 Lược đồ quan hệ ĐƠN VỊ biểu thị trong SQL như sau:

- 1) CREATE TABLE ĐƠN VỊ (
- 2) Mã số ĐV INT,
- 3) Tên ĐV VARCHAR(15),
- 4) Mã số NQL CHAR(9),
- 5) Ngày bắt đầu DATE );

Thuộc tính đầu tiên, MãSốĐV là một số nguyên. Thuộc tính thứ hai là xâu ký tự có độ dài nhỏ hơn hoặc bằng 15. Thuộc tính thứ ba là một xâu có độ dài cố định gồm 9 ký. Như vậy nếu có một mã số người quản lý không có đủ 9 ký tự thì nó sẽ được hệ thống đưa thêm vào một số khoảng trống, còn một mã số có quá 9 ký tự thì sẽ bị chặt bớt đi. Cuối cùng, thuộc tính Ngàybắtđầu kiểu DATE. Trong SQL chuẩn không có kiểu này, chúng ta thay nó bằng CHAR(10).

### 1.6.3 Sửa đổi các lược đồ quan hệ

- Chúng ta có thể loại bỏ một quan hệ R bằng lệnh SQL:

```
DROP TABLE R;
```

Quan hệ R sẽ không còn là một phần của lược đồ cơ sở dữ liệu và chúng ta không còn có thể truy cập đến các bộ giá trị của nó nữa.

- Thông thường chúng ta hay sửa đổi lược đồ của một quan hệ đã tồn tại hơn là xoá bỏ một quan hệ là một phần của cơ sở dữ liệu tồn tại lâu dài. Những sự sửa đổi này được thực hiện bằng một lệnh bắt đầu với từ khoá ALTER TABLE và tên của quan hệ. Sau đó chúng ta có nhiều tùy chọn, quan trọng nhất là

1. ADD sau đó là một tên cột và kiểu của nó.
2. DROP sau đó là một tên cột.

Ví dụ 38: Chúng ta có thể thêm vào quan hệ ĐƠNVI Sốđiệnthoại và bỏ đi thuộc tính Ngàybắtđầu bằng các lệnh sau:

```
DROP TABLE ĐƠNVI ADD Sốđiệnthoại CHAR(10);
```

```
DROP TABLE ĐƠNVI DROP Ngàybắtđầu ;
```

Kết quả là quan hệ ĐƠNVI được thêm vào một thuộc tính Sốđiệnthoại, đó là một xâu ký tự có độ dài cố định gồm 10 ký tự. Trong quan hệ hiện tại, các bộ giá trị đều có các thành phần đối với Sốđiệnthoại nhưng chúng ta biết rằng không có số điện thoại nào được đặt vào đó. Như vậy, giá trị của mỗi thành phần sẽ là NULL. Sau đây chúng ta sẽ thấy có khả năng chọn một giá trị “ngầm định” thay cho NULL đối với các giá trị không biết.

#### 1.6.4 Các giá trị ngầm định

Khi chúng ta tạo ra hoặc sửa đổi các bộ giá trị, đôi lúc chúng ta không có các giá trị cho tất cả các thành phần. Như ví dụ ở trên, khi ta thêm một cột vào một quan hệ, các bộ giá trị đang tồn tại sẽ không có giá trị cho thuộc tính đó và NULL được sử dụng để thay thế cho giá trị “thực”. Tuy nhiên SQL còn cung cấp khả năng chọn giá trị ngầm định, một giá trị xuất hiện trong cột mỗi khi không các giá trị khác được biết.

Nói chung, bất kỳ chỗ nào chúng ta khai báo một thuộc tính và kiểu dữ liệu của nó chúng ta có thể thêm vào từ khoá DEFAULT và một giá trị thích hợp. Giá trị đó hoặc là NULL hoặc là một hằng. Một số các giá trị khác cũng được hệ thống cung cấp như là thời gian hiện tại, hoặc một tùy chọn.

Ví dụ 39 Xét ví dụ 37 ở trên. Chúng ta có thể sử dụng ký tự ? làm ngầm định cho Mẫ số NQL, sử dụng ‘0000-00-00’ làm ngầm định cho Ngày bắt đầu, ta viết như sau:

- 4) Mẫ số NQL CHAR(9) DEFAULT ‘?’,
- 5) Ngày bắt đầu DATE DEFAULT DATE ‘0000-00-00’

#### 1.6.5 Các chỉ số

Một chỉ số trên một thuộc tính A của một quan hệ là một cấu trúc dữ liệu làm có hiệu quả việc tìm các bộ giá trị có giá trị cố định đối với thuộc tính A. Các chỉ số thường giúp đỡ với các truy vấn trong đó thuộc tính A của chúng ta được so sánh với một hằng, ví dụ  $A = 3$  hoặc  $A \leq 3$ .

Khi các quan hệ là rất lớn, việc quét tất cả các bộ của quan hệ để tìm các bộ thoả mãn một điều kiện cho trước trở nên rất tốn kém. Ví dụ, xét truy vấn sau đây:

```
SELECT *  
FROM NHÂNVIÊN  
WHERE Tên = ‘Thanh’ AND Ngàysinh = ‘1965-08-23’ ;
```

Giả sử có 10000 bộ NHÂNVIÊN, trong đó chỉ có 100 có tên là Thanh và có 10 bộ có tên là Thanh và ngày sinh là ‘1965-08-23’.



Một cách vụng về để thực hiện truy vấn này là nhận tất cả 10000 bộ và kiểm tra điều kiện của mệnh đề WHERE trên từng bộ. Một cách có hiệu quả hơn là chỉ nhận 100 bộ có tên là 'Thanh' và kiểm tra từng bộ xem nó có phải là sinh vào '1965-08-23' hay không. Nó sẽ còn hiệu quả hơn nếu chỉ nhận 10 bộ thoả mãn cả hai điều kiện của mệnh đề WHERE. Điều đó sẽ làm được nhờ kỹ thuật chỉ số (index).

Giả sử chúng ta muốn có một chỉ số trên thuộc tính Tên đối với quan hệ NHÂNVIÊN, ta viết

```
CREATE INDEX TênIndex ON NHÂNVIÊN(Tên);
```

Kết quả là một chỉ số có tên là TênIndex sẽ được tạo ra trên thuộc tính Tên của quan hệ NHÂNVIÊN. Từ nay về sau, những truy vấn SQL có chỉ ra một Tên có thể được bộ xử lý truy vấn SQL thực hiện theo cách là chỉ những bộ nào của NHÂNVIÊN với Tên được chỉ rõ là được xem xét. Như vậy, thời gian cần để trả lời cho truy vấn sẽ giảm xuống rất nhiều.

Thông thường, một hệ quản trị cơ sở dữ liệu cho phép ta xây dựng một chỉ số đơn trên nhiều thuộc tính. Kiểu chỉ số này lấy các giá trị của nhiều thuộc tính và tìm được các bộ với các giá trị đã cho đối với các thuộc tính này. Ví dụ, chúng ta có thể khai báo một chỉ số trên các thuộc tính của quan hệ NHÂNVIÊN\_DỰÁN như sau:

```
CREATE INDEX DA_NVIndex ON NHÂNVIÊN (Mã sốDA,  
Mã sốNV);
```

Nếu chúng ta muốn xoá bỏ chỉ số, ta sử dụng lệnh sau:

```
DROP INDEX <Tên chỉ số>
```

Ví dụ: DROP INDEX TênIndex ;

### 1.6.6 Nhập môn về việc lựa chọn các chỉ số

Việc lựa chọn các chỉ số đòi hỏi người thiết kế cơ sở dữ liệu phải cân bằng nhiều yếu tố và trên thực tế việc lựa chọn này là một trong các sự kiện chính có ảnh hưởng đến việc một thiết kế cơ sở dữ liệu có chấp nhận được hay không. Hai sự kiện quan trọng cần xét là:

- Sự tồn tại của một chỉ số trên một thuộc tính làm tăng nhanh tốc độ của các truy vấn trong đó có chỉ ra một giá trị đối với thuộc tính, và trong một số trường hợp có thể tăng tốc độ các phép nối liên quan đến thuộc tính đó.

- Mặt khác, mỗi một chỉ số được xây dựng cho một thuộc tính của một quan hệ nào đấy làm cho các phép chèn, xoá và cập nhật đối với quan hệ đó phức tạp và tốn thời gian hơn.

Việc lựa chọn chỉ số là một trong những phần khó nhất của việc thiết kế cơ sở dữ liệu vì nó đòi hỏi phải đánh giá sự trộn lẫn đặc thù của các truy vấn và các phép toán khác sẽ có trên cơ sở dữ liệu. Nếu một quan hệ được truy vấn thường xuyên hơn là cập nhật thì các chỉ số trên các thuộc tính thường được chỉ ra trong truy vấn là có ý nghĩa. Các chỉ số có lợi với các thuộc tính có khả năng được so sánh với các hằng trong mệnh đề WHERE của các truy vấn. Các chỉ số cũng có lợi đối với các thuộc tính thường xuất hiện trong các điều kiện nối.

## 1.7 KHUNG NHÌN (VIEW)

Các quan hệ được định nghĩa với lệnh CREATE TABLE tồn tại thực sự trong cơ sở dữ liệu. Như vậy, hệ thống SQL lưu trữ các bảng trong một tổ chức vật lý nào đó. Chúng là thường trực và tồn tại lâu dài, chỉ bị thay đổi khi thực hiện lệnh INSERT hoặc các lệnh cập nhật.

Có một lớp các quan hệ khác của SQL, gọi là các khung nhìn, không tồn tại một cách vật lý. Đúng hơn là chúng được định nghĩa bằng một biểu thức giống như một truy vấn. Các khung nhìn có thể được truy vấn như là chúng tồn tại một cách vật lý, và trong một số trường hợp, chúng ta có thể sửa đổi các khung nhìn.

### 1.7.1 Khai báo các khung nhìn

Dạng đơn giản nhất của một định nghĩa khung nhìn là

1. Các từ khoá CREATE VIEW,
2. Tên của khung nhìn,
3. Từ khoá AS và

4. Một truy vấn Q. Truy vấn này là định nghĩa của khung nhìn. Mỗi khi chúng ta truy vấn khung nhìn, SQL ứng xử như là Q đã được thực hiện tại thời điểm đó và truy vấn được áp dụng đối với quan hệ do Q sinh ra.

Như vậy, một khai báo khung nhìn đơn giản có dạng:

```
CREATE VIEW < tên khung nhìn> AS < định nghĩa khung nhìn>;
```

Ví dụ 40: Giả sử chúng ta muốn có một khung nhìn là một phần của quan hệ NHÂNVIÊN, chứa MăsốNV, Họđệm, Tên, Lương và MăsốĐV của các nhân viên có địa chỉ là ‘Hà nội’. Chúng ta có thể định nghĩa khung nhìn này bằng:

- 1) CREATE VIEW NVHÀNỘI AS
- 2) SELECT MăsốNV, Họđệm, Tên, Lương, MăsốĐV
- 3) FROM NHÂNVIÊN
- 4) WHERE Địa chỉ = ‘Hà nội’ ;

Theo định nghĩa này, tên của khung nhìn là NVHÀNỘI, các thuộc tính của khung nhìn là MăsốNV, Họđệm, Tên, Lương, Địa chỉ, MăsốĐV. Định nghĩa của khung nhìn là từ dòng 2 đến dòng 4).

### 1.7.2 Truy vấn các khung nhìn

Quan hệ NVHÀNỘI không chứa các bộ theo nghĩa thông thường. Đúng hơn là nếu chúng ta truy vấn NVHÀNỘI, các bộ thích hợp sẽ nhận được từ bảng cơ sở NHÂNVIÊN, vì vậy truy vấn có thể được trả lời. Kết quả là chúng ta có thể hỏi NVHÀNỘI hai lần cùng một truy vấn và nhận được các trả lời khác nhau. Lý do là ở chỗ, mặc dù chúng ta không thay đổi định nghĩa của khung nhìn NVHÀNỘI nhưng bảng cơ sở NHÂNVIÊN có thể bị thay đổi trong thời gian giữa hai lần truy vấn.

Ví dụ 41 Chúng ta có thể truy vấn khung nhìn NVHÀNỘI như thể nó là một bảng được lưu giữ, chẳng hạn:

```
SELECT Tên  
FROM NVHÀNỘI  
WHERE MăsốĐV = 4 ;
```

Định nghĩa của khung nhìn NVHÀNỘI được sử dụng để biến đổi truy vấn ở trên thành truy vấn mới chỉ nhắm đến bảng cơ sở NHÂNVIÊN. Truy vấn trên tương đương với truy vấn

```
SELECT Tên
FROM NHÂNVIÊN
WHERE Địa chỉ = 'Hà nội' AND Mã số ĐV = 4 ;
```

Ví dụ 42 Có thể viết các truy vấn chứa cả bảng lẫn khung nhìn, chẳng hạn:

```
SELECT Tên ĐV, Tên
FROM NVHÀNỘI, ĐƠN VỊ
WHERE NVHÀNỘI.Mã số ĐV = ĐƠN VỊ.Mã số ĐV
```

Truy vấn này đòi hỏi tên của đơn vị và tên của các nhân viên có địa chỉ tại Hà nội.

Ví dụ 43 Chúng ta hãy xét một truy vấn phức tạp hơn được sử dụng để định nghĩa một khung nhìn.

```
CREATE VIEW NVĐV AS
SELECT Tên ĐV, Tên
FROM NHÂNVIÊN, ĐƠN VỊ
WHERE NHÂNVIÊN.Mã số ĐV = ĐƠN VỊ.Mã số ĐV;
```

Chúng ta có thể truy vấn khung nhìn này như thể nó là một quan hệ được lưu trữ, ví dụ

```
SELECT Tên
FROM NVĐV
WHERE Tên = 'Thanh';
```

Truy vấn ở trên tương đương với truy vấn:

```
SELECT Tên
FROM NHÂNVIÊN, ĐƠN VỊ
WHERE (NHÂNVIÊN.Mã số ĐV = ĐƠN VỊ.Mã số ĐV)
```

AND (Tên = 'Thanh');

### 1.7.3 Đặt tên lại các thuộc tính

Đôi khi chúng ta thích đặt tên mới cho các thuộc tính của khung nhìn. Để làm điều đó, chúng ta chỉ ra các thuộc tính của khung nhìn bằng cách liệt kê chúng ở trong cặp dấu ngoặc và đặt ở sau tên của khung nhìn trong lệnh CREATE VIEW. Ví dụ, chúng ta có thể viết lại định nghĩa khung nhìn ở ví dụ 1.43 như sau:

```
CREATE VIEW NVĐV(Tênđơnvị, Tênnhânviên) AS
SELECT TênĐV, Tên
FROM NHÂNVIÊN, ĐƠNVỊ
WHERE NHÂNVIÊN.MãSốĐV = ĐƠNVỊ.MãSốĐV;
```

Hai khung nhìn là hoàn toàn như nhau nhưng các thuộc tính được đặt tên lại, Tênđơnvị và Tênnhânviên thay cho TênĐV và Tên.

### 1.7.4 Sửa đổi các khung nhìn

Trong nhiều trường hợp chúng ta không thể thực hiện một lệnh chèn, xóa hoặc cập nhật đối với một khung nhìn bởi vì khung nhìn không giống như một bảng cơ sở. Tuy nhiên, đối với các khung nhìn tương đối đơn giản, gọi là các khung nhìn cập nhật được, có khả năng chuyển đổi cập nhật của khung nhìn thành một cập nhật tương đương trên một bảng cơ sở và phép cập nhật có thể được thực hiện đối với bảng cơ sở. Điều kiện để khung nhìn cập nhật được là các khung nhìn được định nghĩa bằng phép chọn một số thuộc tính từ một quan hệ R (quan hệ này cũng có thể là một khung nhìn cập nhật được). Hai điểm kỹ thuật quan trọng:

- Mệnh đề WHERE không được bao hàm R trong một truy vấn con
- Danh sách trong mệnh đề SELECT phải chứa đủ các thuộc tính sao cho với mỗi bộ được chèn vào khung nhìn, chúng ta có thể điền các thuộc tính khác vào với các giá trị null hoặc ngầm định thích hợp và có một bộ của quan hệ cơ sở sẽ được tạo nên từ bộ được chèn vào của khung nhìn.

Ví dụ 44 Giả sử chúng ta cố gắng chèn vào khung nhìn NVHÀNỘI một bộ:

INSERT INTO NVHÀNỘI

VALUES ('NV065', 'Nguyễn Đình', 'Thi', 4500, 4);

Khung nhìn NVHÀNỘI hầu như thoả mãn các điều kiện cập nhật được của SQL bởi vì khung nhìn chỉ yêu cầu một số thành phần của các bộ của bảng cơ sở NHÂNVIÊN. Chỉ có một vấn đề là vì thuộc tính Địa chỉ của bảng NHÂNVIÊN không phải là một thuộc tính của khung nhìn, bộ giá trị mà chúng ta chèn vào NHÂNVIÊN sẽ có giá trị NULL chứ không phải là 'Hà nội' như là giá trị của nó cho Địa chỉ. Bộ giá trị này không thoả mãn điều kiện là địa chỉ của nhân viên là Hà nội.

Như vậy, để làm cho khung nhìn NVHÀNỘI cập nhật được, chúng ta sẽ thêm thuộc tính Địa chỉ cho mệnh đề SELECT của nó, mặc dù rõ ràng là địa chỉ nhân viên là Hà nội. Định nghĩa lại của khung nhìn NVHÀNỘI là:

- 1) CREATE VIEW NVHÀNỘI AS
- 2) SELECT Mẫ sốNV, Họđệm, Tên, Lương, Địa chỉ, Mẫ sốĐV
- 3) FROM NHÂNVIÊN
- 4) WHERE Địa chỉ = 'Hà nội' ;

Sau đó, chúng ta viết lệnh chèn vào khung nhìn cập nhật được NVHÀNỘI như sau:

INSERT INTO NVHÀNỘI

VALUES ('NV065', 'Nguyễn Đình', 'Thi', 4500, 4);

Thực hiện lệnh chèn, chúng ta tạo ra một bộ của NHÂNVIÊN sinh từ một bộ của khung nhìn được chèn vào khi định nghĩa khung nhìn được áp dụng cho NHÂNVIÊN. Các thuộc tính khác không xuất hiện trong khung nhìn chắc chắn tồn tại trong bộ NHÂNVIÊN được chèn vào. Tuy nhiên chúng ta không thể suy ra giá trị của chúng. Trong kết quả bộ mới của NHÂNVIÊN phải có trong các thành phần đối với mỗi thuộc tính này các giá trị mặc định thích hợp hoặc NULL hoặc một ngầm định nào đó đã được khai báo cho thuộc tính.

Chúng ta cũng có thể loại bỏ ra khỏi một khung nhìn cập nhật được. Lệnh xoá, cũng như lệnh chèn, được thực hiện thông qua một quan hệ nền R và gây ra việc loại bỏ một bộ của R gây ra bộ được xoá của khung nhìn.

Ví dụ 45: Giả sử chúng ta muốn xoá khỏi khung nhìn cập nhật được NVHÀNỘI tất cả các bộ có tên chứa từ ‘an’. Ta có thể viết lệnh xoá như sau:

```
DELETE FROM NVHÀNỘI  
WHERE Tên LIKE “%an%” ;
```

Lệnh xoá này được chuyển thành một lệnh xoá tương đương trên bảng cơ sở NHÂNVIÊN; chỉ khác ở chỗ là điều kiện định nghĩa khung nhìn NVHÀNỘI được thêm vào các điều kiện của mệnh đề WHERE. Kết quả là lệnh xoá như sau:

```
DELETE FROM NHÂNVIÊN  
WHERE Tên LIKE “%an%” AND Địa chỉ = ‘Hà nội’;
```

Tương tự, một lệnh cập nhật trên một khung nhìn cập nhật được được thực hiện thông qua quan hệ nền. Như vậy lệnh cập nhật khung nhìn có tác dụng cập nhật tất cả các bộ của quan hệ nền sinh ra các bộ được cập nhật trong khung nhìn.

Ví dụ 46: Lệnh cập nhật khung nhìn

```
UPDATE NVHÀNỘI  
SET Lương = 4500  
WHERE Mã số NV = ‘NV002’;
```

được chuyển thành lệnh cập nhật bảng cơ sở:

```
UPDATE NHÂNVIÊN  
SET Lương = 4500  
WHERE Mã số NV = ‘NV002’ AND Địa chỉ = ‘Hà nội’;
```

Loại cập nhật một khung nhìn cuối cùng là loại bỏ nó. Lệnh cập nhật này có thể thực hiện dù khung nhìn có cập nhật được hay không. Lệnh DROP thông thường là:

DROP VIEW NVHÀNỘI ;

Chú ý rằng lệnh này xoá định nghĩa của khung nhìn vì vậy chúng ta không thể tiếp tục truy vấn hoặc cập nhật đối với khung nhìn này nữa. Tuy nhiên, việc xoá bỏ một khung nhìn không làm ảnh hưởng đến một bộ nào của quan hệ nền NHÂNVIÊN. Ngược lại

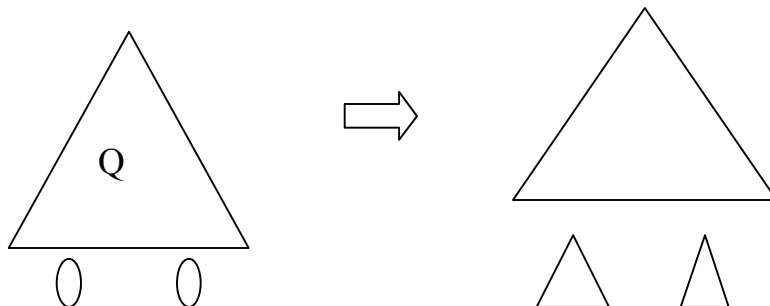
DROP TABLE Movie

sẽ không chỉ xoá bỏ bảng NHÂNVIÊN, nó cũng sẽ làm cho khung nhìn NVHÀNỘI trở nên vô dụng bởi vì một truy vấn sử dụng nó sẽ tham chiếu đến một quan hệ NHÂNVIÊN không tồn tại.

### 1.7.5 Giải thích các truy vấn có chứa các khung nhìn

Để giải thích các truy vấn khung nhìn, chúng ta lần theo cách một truy vấn có chứa khung nhìn được xử lý như thế nào.

Tư tưởng cơ bản được minh hoạ ở hình vẽ dưới đây (hình 3). Một truy vấn Q được biểu thị bằng cây biểu thức trong đại số quan hệ. Cây biểu thức này sử dụng các quan hệ là các khung nhìn làm lá. Trong hình vẽ cây có hai lá, đó là các khung nhìn V và W. Để giải thích Q theo thuật ngữ của các bảng cơ sở, chúng ta tìm các định nghĩa của V và W. Các định nghĩa này cũng được biểu thị bằng các cây biểu thức trong đại số quan hệ. Trong hình 3 (ở bên phải) chúng ta thay các lá V và W bằng các định nghĩa của các khung nhìn đó. Cây kết quả là một truy vấn trên các bảng cơ sở tương đương với truy vấn gốc trên các khung nhìn.





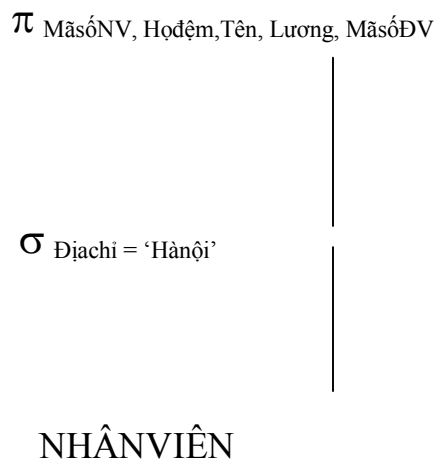
## V W

Hình 3: Thay thế các định nghĩa khung nhìn cho các tham chiếu khung nhìn

Ví dụ 47: Ta xét khung nhìn và truy vấn được định nghĩa như sau:

- 1) CREATE VIEW NVHÀNỘI AS
- 2) SELECT MăsốNV, Họđệm, Tên, Lương, MăsốĐV
- 3) FROM NHÂNVIÊN
- 4) WHERE Địachỉ = 'Hà nội' ;

Một cây biểu thức cho truy vấn định nghĩa khung nhìn này được chỉ ra ở hình 4.



Hình 4: Cây biểu thức cho khung nhìn NVHÀNỘI

Truy vấn ở ví dụ 41 có dạng

```
SELECT Tên
FROM NVHÀNỘI
WHERE MăsốĐV = 4 ;
```

Cây biểu thức cho truy vấn này được chỉ ra ở hình 5

$\pi$  Tên

$\sigma_{\text{Mã sốĐV} = 4}$

NVHÀNỘI

Hình 5: Cây biểu thức cho truy vấn ở ví dụ 41

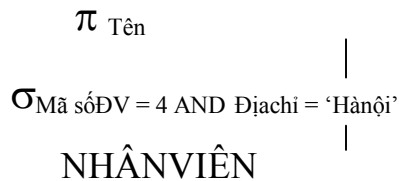
Chú ý rằng lá của cây này biểu diễn khung nhìn NVHÀNỘI. Từ đó, chúng ta giải thích truy vấn bằng cách thay thế cây truy vấn của NVHÀNỘI vào vị trí của NVHÀNỘI trong cây biểu thức của truy vấn. Kết quả, chúng ta có cây biểu thức như sau:

$\pi$  Tên  
 $\sigma_{\text{Mã sốĐV} = 4}$   
 $\pi$  Mã sốNV, Họđệm, Tên, Lương, Mã sốĐV  
 $\sigma_{\text{Địa chỉ} = \text{'Hànội'}}$   
NHÂNVIÊN

Cây truy vấn này là một giải thích chấp nhận được của truy vấn. Tuy nhiên nó được diễn đạt bằng cách phức tạp không cần thiết. Hệ thống SQL sẽ áp dụng các biến đổi đối với cây này để làm cho nó giống như biểu thức cây đối với truy vấn

```
SELECT Tên
FROM NHÂNVIÊN
WHERE Địa chỉ = 'Hànội' AND Mã sốĐV = 4 ;
```

Chẳng hạn, chúng ta có thể đưa phép chiếu  $\pi_{Mã sốNV, Họđệm,Tên, Lương, Mã sốĐV}$  lên trên phép chọn  $\sigma_{Mã sốĐV = 4}$  vì việc thay đổi đó không làm ảnh hưởng đến kết quả của biểu thức. Sau đó, chúng ta có hai phép chiếu liên tiếp, đầu tiên chiếu trên  $Mã sốNV, Họđệm,Tên, Lương, Mã sốĐV$ , sau đó chiếu trên  $Tên$ . Rõ ràng lần chiếu thứ nhất là thừa, chúng ta có thể loại bỏ nó. Như vậy chúng ta có thể thay thế hai phép chiếu bằng bằng một phép chiếu trên  $Tên$ . Hai phép chọn cũng có thể được kết hợp lại. Nói chung, có thể thay thế hai phép chọn liên tiếp bằng một phép chọn với phép toán AND của các điều kiện của chúng. Cây biểu thức kết quả là:



Đó là cây nhận được từ truy vấn:

```
SELECT Tên
FROM NH\grave{A}NVI\grave{E}N
WHERE Địachỉ = 'Hà nộ';
```

## 1.8 TỔNG KẾT CHƯƠNG I

- 1- SQL: Ngôn ngữ SQL là ngôn ngữ truy vấn chính cho các hệ cơ sở dữ liệu quan hệ. Chuẩn hiện tại được gọi là SQL-99 hoặc SQL3
- 2- Các truy vấn Select-From-Where: Dạng truy vấn SQL phổ biến nhất có dạng select-from-where. Nó cho phép ta lấy tích của nhiều quan hệ (mệnh đề FROM), áp dụng một điều kiện đối với các bộ của kết quả (mệnh đề WHERE) và sinh ra các thành phần mong muốn (mệnh đề SELECT).
- 3- Truy vấn con: Các truy vấn Select-From-Where cũng có thể được sử dụng như các truy vấn con bên trong một mệnh đề WHERE hoặc mệnh đề FROM của một truy vấn khác. Các phép toán EXIST, IN, ALL, và

ANY có thể được sử dụng để diễn đạt các điều kiện có giá trị Logic về các quan hệ là kết quả của một truy vấn con trong mệnh đề WHERE.

- 4- Các phép toán tập hợp trên các quan hệ: Chúng ta có thể lấy hợp, giao, hoặc trừ của các quan hệ bằng cách nối các quan hệ hoặc nối các truy vấn xác định các quan hệ, với các từ khóa UNION, INTERSECT và EXCEPT tương ứng.
- 5- Các biểu thức nối: SQL có các phép toán như NATURAL JOIN có thể áp dụng cho các quan hệ như các truy vấn hoặc để xác định các quan hệ trong một mệnh đề FROM.
- 6- Các giá trị Null: SQL cung cấp một giá trị NULL đặc biệt, nó xuất hiện trong các thành phần của các bộ không có giá trị cụ thể cho chúng. Các phép toán số học và logic không dùng được với NULL. Việc so sánh một giá trị bất kỳ với NULL, thậm chí giá trị so sánh là NULL, sẽ cho giá trị UNKNOWN. Giá trị UNKNOWN đối xử trong các biểu thức có giá trị logic như là nửa đường giữa TRUE và FALSE.
- 7- Nối ngoài: SQL cung cấp một phép toán OUTER JOIN. Phép toán này nối các quan hệ và tạo ra kết quả có chứa các bộ từ một hoặc cả hai quan hệ tham gia phép nối. Các bộ được thêm các giá trị NULL ở trong kết quả.
- 8- Mô hình túi (bag) của các quan hệ: SQL xem các quan hệ như các túi các bộ chứ không phải tập hợp các bộ. Chúng ta có thể ép buộc việc loại bỏ các bộ trùng lặp bằng từ khóa DISTINCT, trong khi đó từ khóa ALL cho phép kết quả là một túi trong hoàn cảnh các túi không phải là ngầm định.
- 9- Phép nhóm: Các giá trị xuất hiện trong một cột của quan hệ có thể được tính tổng (kết hợp lại) bằng cách sử dụng một trong các từ khóa SUM, AVG, MIN, MAX hoặc COUNT. Các bộ có thể được phân nhóm trước để kết hợp với các từ khóa GROUP BY. Một số nhóm có thể bị loại bỏ với một mệnh đề HAVING.
- 10- Các lệnh cập nhật: SQL cho phép chúng ta làm thay đổi các bộ trong một quan hệ. Chúng ta có thể INSERT (chèn các bộ mới), DELETE

(xóa các bộ), UPDATE(thay đổi một số bộ đã tồn tại) bằng cách viết các lệnh SQL sử dụng các từ khóa đó.

- 11- Định nghĩa dữ liệu: SQL có các lệnh mô tả các phần tử của một lược đồ cơ sở dữ liệu. Lệnh CREATE TABLE cho phép chúng ta mô tả lược đồ cho các quan hệ được lưu giữ (gọi là các bảng), chỉ ra các thuộc tính và kiểu của chúng và các giá trị ngầm định.
- 12- Sửa đổi các lược đồ: Chúng ta có thể thay đổi diện mạo của một lược đồ cơ sở dữ liệu bằng một lệnh ALTER. Các thay đổi này bao gồm việc thêm và loại bỏ các thuộc tính các lược đồ quan hệ và thay đổi các giá trị ngầm định liên kết với một thuộc tính hoặc một miền. Chúng ta cũng có thể sử dụng lệnh DROP để loại bỏ hoàn toàn các quan hệ hoặc các thành phần lược đồ khác.
- 13- Các chỉ số: Trong khi không phải là một phần của chuẩn SQL, các hệ thống SQL thương mại cho phép mô tả các chỉ số trên các thuộc tính; các chỉ số này là tăng tốc độ một số truy vấn và cập nhật chứa đặc tả của một giá trị đối với thuộc tính được chỉ số hóa.
- 14- Khung nhìn: Một khung nhìn là một định nghĩa về việc một quan hệ (khung nhìn) được xây dựng từ các bảng được lưu giữ trong cơ sở dữ liệu như thế nào. Các khung nhìn có thể được truy vấn giống như chúng là các bảng được lưu giữ, và một hệ thống SQL sửa đổi các truy vấn về một khung nhìn sao cho truy vấn được thay thế thành truy vấn trên bảng cơ sở đã được sử dụng để định nghĩa khung nhìn.

## **MỘT SỐ BÀI TẬP**

Giả sử chúng ta có cơ sở dữ liệu sau(xem ở PHỤ LỤC 2):

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

I. Viết các truy vấn sau:

- 1) Tìm số model, tốc độ, và kích cỡ đĩa cứng của tất cả các máy PC có giá thấp hơn \$1200.
- 2) Làm như câu 1) nhưng thay tên cột speed bằng megahertz và cột hd bằng gigabyte
- 3) Tìm các nhà sản xuất các máy in
- 4) Tìm số model, kích cỡ bộ nhớ và kích cỡ màn hình của các máy xách tay (Laptop) có giá trên \$2000
- 5) Đưa ra tất cả các bộ trong quan hệ Printer với các máy in màu. Chú ý rằng color là thuộc tính có giá trị logic
- 6) Đưa ra số model, tốc độ và kích cỡ của đĩa cứng đối với các máy PC có đĩa DVD 12X hoặc 16X và giá thấp hơn \$2000. Bán có thể xem thuộc tính rd như là thuộc tính có giá trị là chuỗi ký tự.

II Viết các truy vấn

- 1) Đưa ra nhà sản xuất và tốc độ của các Laptop với đĩa cứng ít nhất là 30 gigabyte.
- 2) Đưa ra số model và giá của tất cả các sản phẩm (với mọi kiểu) do nhà sản xuất B làm ra
- 3) Đưa ra những nhà sản xuất bán các Laptop nhưng không bán các máy PC
- 4) Tìm các kích cỡ đĩa cứng có trong hai hoặc nhiều hơn PC
- 5) Đưa ra các cặp model PC có cùng tốc độ và bộ nhớ (ram). Một cặp chỉ được liệt kê một lần nghĩa là liệt kê (i,j) nhưng không liệt kê (j,i).
- 6) Đưa ra các nhà sản xuất sản xuất ít nhất là hai loại máy tính khác nhau (PC hoặc Laptop) với tốc độ ít nhất là 1000.

III. Trong phần bài tập này, bạn phải sử dụng ít nhất một truy vấn con trong câu trả lời và viết mỗi truy vấn trong hai cách khác nhau.

- 1) Đưa ra những người sản xuất các PC có tốc độ ít nhất là 1200.

- 2) Đưa ra các máy in có giá cao nhất.
- 3) Đưa ra các máy xách tay có tốc độ thấp hơn tốc độ của các PC
- 4) Đưa ra số model của các mục (PC, laptop, printer) với giá cao nhất.
- 5) Đưa ra nhà sản xuất của máy in màu có giá thấp nhất.
- 6) Đưa ra các nhà sản xuất của các PC có bộ xử lý nhanh nhất trong số các PC có dung lượng RAM bé nhất.

#### IV. Sử dụng hàm nhóm

- 1) Tìm tốc độ trung bình của các PC.
- 2) Tìm tốc độ trung bình của các Laptop có giá trên \$2000
- 3) Tìm giá trung bình của các PC do nhà sản xuất “A” làm ra
- 4) Tìm giá trung bình của các PC và laptop do nhà sản xuất “D” làm ra
- 5) Đưa ra giá trung bình của các PC đối với mỗi tốc độ khác nhau.
- 6) Đối với mỗi nhà sản xuất, hãy đưa ra kích cỡ trung bình của màn hình của các laptop.
- 7) Đưa ra các nhà sản xuất làm ra ít nhất là 3 model máy PC khác nhau.
- 8) Đưa ra kích cỡ trung bình của các đĩa cứng của các PC đối với các nhà sản xuất làm ra các máy in.

#### IV. Sửa đổi cơ sở dữ liệu.

- 1) Sử dụng hai lệnh INSERT để lưu trữ vào cơ sở dữ liệu sự kiện model PC 1100 được nhà sản xuất C làm ra, có tốc độ 1800, RAM 256, đĩa cứng 80, 20x DVD và bán với giá \$2499.
- 2) Chèn vào các sự kiện kiểm tra rằng với mỗi PC có một laptop với cùng nhà sản xuất, tốc độ, RAM và đĩa cứng, một màn hình 15 inch, một số model lớn hơn 1100, và một giá trên \$500.
- 3) Loại bỏ tất cả PC với đĩa cứng nhỏ hơn 20 gigabyte.
- 4) Loại bỏ tất cả các laptop của các nhà sản xuất không sản xuất máy in.

- 5) Nhà sản xuất A mua nhà sản xuất B. Thay đổi tất cả các sản phẩm do B sản xuất thành ra do A sản xuất.
- 6) Với mỗi PC, hãy làm tăng gấp đôi dung lượng của RAM và thêm đĩa vào cứng 20 gigabyte.
- 7) Với mỗi laptop do nhà sản xuất B làm ra hãy thêm vào kích cỡ màn hình 1 inch và giảm giá \$100.

VI. Mô tả cơ sở dữ liệu. Viết các khai báo sau:

- 1) Một lược đồ thích hợp cho quan hệ Product
- 2) Một lược đồ thích hợp cho quan hệ PC
- 3) Một lược đồ thích hợp cho quan hệ Laptop
- 4) Một lược đồ thích hợp cho quan hệ Printer.
- 5) Một tùy chọn cho lược đồ Printer từ 4) để loại bỏ thuộc tính Color
- 6) Một tùy chọn cho lược đồ Laptop ở 3) để thêm và thuộc tính cd. Giả sử giá trị mặc định của thuộc tính này là none nếu laptop không có CD reader



## CHƯƠNG II: CÁC RÀNG BUỘC VÀ CÁC TRIGGER

Trong chương này chúng ta sẽ nhắm vào các khía cạnh của SQL cho phép chúng ta tạo ra các phần tử “tích cực”. Một phần tử tích cực là một biểu thức hoặc một lệnh được viết một lần, được lưu trữ vào cơ sở dữ liệu và chờ đợi phần tử thực hiện vào những lúc thích hợp. Thời gian hành động có thể là khi một sự kiện nào đó xảy ra, chẳng hạn khi chèn một bộ giá trị vào một quan hệ cụ thể hoặc có thể là khi cơ sở dữ liệu thay đổi sao cho một điều kiện có giá trị logic trở thành đúng.

Một trong những vấn đề nghiêm trọng đối mặt với những người viết chương trình ứng dụng để cập nhật cơ sở dữ liệu là thông tin mới có thể sai theo nhiều cách, chẳng hạn do các lỗi chính tả hoặc chép lại khi nhập dữ liệu bằng tay. Một cách đơn giản nhất để đảm bảo rằng việc sửa đổi cơ sở dữ liệu không cho phép các bộ giá trị không thích hợp trong các quan hệ là viết các chương trình ứng dụng sao cho mỗi lệnh chèn, xoá và cập nhật liên kết với các kiểm tra cần thiết để đảm bảo tính đúng đắn. Các đòi hỏi về tính đúng đắn luôn luôn phức tạp và chúng thường lặp lại; các chương trình ứng dụng phải thực hiện các kiểm tra như nhau sau mỗi lần sửa đổi.

Thật may mắn, SQL cung cấp nhiều kỹ thuật để biểu thị các ràng buộc toàn vẹn như là một phần của lược đồ cơ sở dữ liệu. Trong chương này chúng ta sẽ nghiên cứu các phương pháp cơ bản. Trước tiên chúng ta nói đến các ràng buộc khoá, trong đó một thuộc tính hay một tập thuộc tính được khai báo như là một khoá đối với quan hệ. Tiếp theo, chúng ta xem xét một dạng của toàn vẹn tham chiếu được gọi là “các ràng buộc khoá ngoài”, chúng đòi hỏi rằng một giá trị trong một (hoặc các) thuộc tính của một quan hệ cũng phải xuất hiện như là một giá trị của một (hoặc các) thuộc tính của một quan hệ khác. Sau đó, chúng ta xem xét các ràng buộc trên các thuộc tính, các bộ và các quan hệ. Chúng ta sẽ nhắm vào các ràng buộc giữa các quan hệ được gọi là các khẳng định (assertion). Cuối cùng, chúng ta thảo luận về “trigger”, đó là một dạng của phần tử tích cực được gọi vào hoạt động trên các sự kiện cụ thể nào đó, chẳng hạn như chèn vào một quan hệ cụ thể.

## 2.1 KHÓA VÀ KHÓA NGOÀI

Ràng buộc quan trọng nhất trong một cơ sở dữ liệu là khai báo một (hoặc một tập) thuộc tính nào đó tạo nên một khoá cho một quan hệ. Nếu một tập các thuộc tính S là một khoá cho một quan hệ R thì hai bộ bất kỳ của R phải khác nhau tại ít nhất là một thuộc tính trong tập S. Như vậy, nếu R có một khoá được khai báo thì trong R không được có các bộ trùng lặp.

Giống như các ràng buộc khác, một ràng buộc khoá được mô tả bên trong lệnh CREATE TABLE của SQL. Có hai cách tương đương để mô tả các khoá: sử dụng từ khoá PRIMARY KEY hoặc từ khoá UNIQUE. Tuy nhiên, một bảng chỉ có thể có một khoá chính nhưng có nhiều mô tả “unique”.

SQL cũng sử dụng thuật ngữ khoá kết hợp với ràng buộc toàn vẹn tham chiếu. Các ràng buộc này, được gọi là các “ràng buộc khoá ngoài”, khẳng định rằng một giá trị xuất hiện trong thành phần khoá ngoài của một quan hệ cũng phải xuất hiện trong thành phần khoá chính của một quan hệ khác.

### 2.1.1 Mô tả khoá chính

Một quan hệ chỉ có thể có một khoá chính. Có hai cách mô tả khoá chính trong lệnh CREATE TABLE.

1- Chúng ta có thể mô tả một thuộc tính là khoá chính khi thuộc tính này được liệt kê trong lược đồ quan hệ. Theo cách này, chúng ta thêm vào từ khoá PRIMARY KEY sau thuộc tính và kiểu của nó.

2- Chúng ta có thể thêm vào danh sách các mục được mô tả trong lược đồ một mô tả phụ thêm nói rằng một (hoặc một tập) thuộc tính cụ thể tạo nên khoá chính. Theo cách này, chúng ta tạo ra một phần tử mới trong danh sách các thuộc tính bao gồm từ khoá PRIMARY KEY và một danh sách các thuộc tính tạo nên khoá đặt trong các dấu ngoặc.

Chú ý rằng, nếu khoá chứa nhiều hơn một thuộc tính, chúng ta cần sử dụng cách 2.

Tác động của việc mô tả một tập thuộc tính S là khoá chính đối với quan hệ R là:

1. Hai bộ giá trị trong R không thể bằng nhau trên mọi thuộc tính trong tập S. Mọi ý đồ chèn hoặc cập nhật một bộ giá trị mà vi phạm quy tắc này sẽ gây ra việc hệ quản trị cơ sở dữ liệu loại bỏ hành động gây ra sự vi phạm.
2. Các thuộc tính trong S không được phép có giá trị NULL.

Ví dụ 2.1 Xét lược đồ quan hệ ĐƠNVI. Khoá chính của quan hệ này là Mã sốĐV. Vì vậy chúng ta có thể thêm sự kiện này vào dòng mô tả Mã sốĐV:

- 1) **CREATE TABLE ĐƠNVI**
- 2) (TênĐV VARCHAR(15),
- 3) Mã sốĐV INT PRIMARY KEY,
- 4) Mã sốNQL CHAR(9)
- 5) Ngàybắtđầu DATE);

Theo cách 2, chúng ta có thể sử dụng một định nghĩa khoá chính tách riêng. Sau dòng 5 của ví dụ 2.1, ta thêm vào mô tả của khoá chính và chúng ta không cần phải mô tả nó ở dòng 2. Ta có mô tả lược đồ như sau:

- 1) **CREATE TABLE ĐƠNVI**
- 2) (TênĐV VARCHAR(15),
- 3) Mã sốĐV INT,
- 4) Mã sốNQL CHAR(9),
- 5) Ngàybắtđầu DATE,
- 6) PRIMARY KEY(Mã sốĐV);

Cả hai cách mô tả như trên đều chấp nhận được bởi vì khoá chính chỉ gồm một thuộc tính. Tuy nhiên, trong hoàn cảnh khoá chính có nhiều hơn một thuộc tính chúng ta phải dùng cách mô tả thứ hai. Ví dụ, nếu ta mô tả lược đồ quan hệ NHÂNVIÊN\_DỰÁN có khoá chính là cặp thuộc tính Mã sốNV, Mã sốDA, sau danh sách các thuộc tính ta sẽ thêm vào dòng sau:

PRIMARY KEY(Mã sốNV, Mã sốDA)

### 2.1.2 Các khoá được mô tả với UNIQUE

Một cách khác để mô tả khoá là sử dụng từ khoá UNIQUE. Từ khoá này có thể xuất hiện ở đúng chỗ mà PRIMARY KEY có thể xuất hiện: hoặc đi sau một thuộc tính và kiểu của nó, hoặc như là một mục riêng ở bên trong lệnh

CREATE TABLE. Ý nghĩa của mô tả UNIQUE gần giống như ý nghĩa của mô tả PRIMARY KEY. Có hai điểm khác, đó là:

1- Chúng ta có thể có nhiều mô tả UNIQUE cho một bảng nhưng chỉ có một khoá chính.

2- Trong khi PRIMARY KEY ngăn cấm các giá trị NULL trong các thuộc tính của khoá thì UNIQUE lại cho phép. Hơn nữa, quy tắc hai bộ giá trị không thể bằng nhau trong mọi thuộc tính của tập thuộc tính được mô tả bằng UNIQUE có thể bị vi phạm nếu một hoặc nhiều thành phần cho phép có giá trị NULL. Trên thực tế nó còn cho phép hai bộ có NULL trong mọi thuộc tính tương ứng của khoá UNIQUE.

Ví dụ 2.2: Chúng ta có thể viết lại mô tả trong ví dụ 2.1 bằng cách sử dụng UNIQUE như sau:

- 1) **CREATE TABLE ĐƠN VỊ**
- 2) (TênĐV VARCHAR(15),
- 3) MãSốĐV INT UNIQUE,
- 4) MãSốNQL CHAR(9)
- 5) Ngàybắtđầu DATE);

Tương tự, chúng ta có thể thay dòng 6) trong cách mô tả thứ hai bằng

- 6) UNIQUE(Mã sốĐV);

### 2.1.3 Làm có hiệu lực các ràng buộc khoá

Trong phần nói về chỉ số, chúng ta đã biết rằng mặc dù chúng không phải là một phần của chuẩn SQL, mỗi phiên bản SQL có một cách tạo ra các chỉ số như là một phần của định nghĩa lược đồ cơ sở dữ liệu. Thông thường, chỉ số được xây dựng trên khoá chính để hỗ trợ kiểu truy vấn có chỉ ra một giá trị đối với khoá chính. Chúng ta cũng có thể xây dựng các chỉ số trên các thuộc tính khác được mô tả là UNIQUE. Sau khi tạo ra chỉ số, nếu một mệnh đề WHERE của truy vấn chứa một điều kiện được tạo nên từ một khoá và một giá trị cụ thể, bộ giá trị phù hợp sẽ được tìm thấy rất nhanh chóng mà không phải tìm kiếm qua tất cả các bộ giá trị của quan hệ. Nhiều phiên bản SQL cung cấp một lệnh tạo chỉ số bằng cách sử dụng từ khoá UNIQUE mô tả một

thuộc tính là khoá tại cùng thời điểm nó tạo ra một chỉ số trên thuộc tính đó. Ví dụ, lệnh

```
CREATE UNIQUE INDEX MăsốNVIndex ON NHÂNVIÊN(MăsốNV);
```

tạo ra chỉ số đồng thời mô tả ràng buộc về tính duy nhất trên thuộc tính Mã sốNV của quan hệ NHÂNVIÊN.

Bây giờ chúng ta hãy xem xét hệ thống SQL sẽ làm có hiệu lực một ràng buộc khoá như thế nào. Về nguyên tắc, ràng buộc phải được kiểm tra mỗi một lần chúng ta làm thay đổi cơ sở dữ liệu. Tuy nhiên, mỗi ràng buộc khoá đối với quan hệ R có thể bị vi phạm chỉ khi R bị thay đổi. Trên thực tế, một lệnh xoá các bộ giá trị ra khỏi quan hệ R không thể gây ra một vi phạm, chỉ có phép chèn và phép cập nhật là có thể. Như vậy, thông thường hệ thống SQL kiểm tra ràng buộc khoá chỉ khi xuất hiện một phép chèn hoặc một phép cập nhật đối với quan hệ.

Nếu có chỉ số trên thuộc tính (hoặc các thuộc tính) được mô tả là khoá thì hệ thống SQL làm hiệu lực ràng buộc khoá một cách rất hiệu quả. Thật vậy, nếu có sẵn chỉ số thì khi ta chèn một bộ giá trị vào quan hệ hoặc cập nhật một thuộc tính khoá trong một bộ nào đấy, hệ thống dùng chỉ số để kiểm tra rằng có hay không có sẵn một bộ với cùng giá trị trong thuộc tính (hoặc các thuộc tính) được mô tả là khoá. Nếu có, hệ thống phải ngăn ngừa sự thay đổi xảy ra.

Nếu không có chỉ số trên các thuộc tính khoá thì vẫn còn khả năng làm có hiệu lực một ràng buộc khoá. Việc sắp xếp quan hệ theo giá trị khoá sẽ giúp việc tìm kiếm của hệ thống. Tuy nhiên, nếu không có sự hỗ trợ tìm kiếm thì hệ thống phải xem xét toàn bộ quan hệ để tìm kiếm một bộ giá trị với giá trị khoá cho trước. Quá trình đó là cực kỳ tốn thời gian và có thể làm cho việc sửa đổi các quan hệ lớn của cơ sở dữ liệu trở nên không thể.

#### **2.1.4 Mô tả các ràng buộc khoá ngoài**

Một loại ràng buộc quan trọng thứ hai trên lược đồ cơ sở dữ liệu là giá trị đối với một số thuộc tính nào đó phải có nghĩa.

Trong SQL, chúng ta có thể khai báo một (hoặc các) thuộc tính của một quan hệ là khoá ngoài tham chiếu đến một (hoặc các) thuộc tính của một quan hệ thứ hai (có thể cùng một quan hệ). Khai báo đó có nghĩa là:

1. Các thuộc tính được tham chiếu của quan hệ thứ hai phải được tuyên bố là UNIQUE hoặc PRIMARY KEY đối với quan hệ của chúng. Ngược lại, chúng ta không thể khai báo khoá ngoài.

2. Các giá trị của khoá ngoài xuất hiện trong quan hệ thứ nhất cũng phải xuất hiện trong các thuộc tính được tham chiếu của một bộ nào đó. Chính xác hơn, giả sử có một khoá ngoài F tham chiếu một tập thuộc tính G của một quan hệ nào đó. Giả sử một bộ t của quan hệ thứ nhất có các giá trị xác định trong tất cả các thuộc tính của F; gọi danh sách các giá trị của t trong các thuộc tính đó là t[F]. Khi đó, trong quan hệ được tham chiếu phải có một bộ s nào đó phù hợp với t[F] trên các thuộc tính G, nghĩa là  $s[G] = t[F]$ .

Cũng như đối với khoá chính, chúng ta có hai cách khai báo một khoá ngoài.

a) Nếu khoá ngoài chỉ gồm một thuộc tính, chúng ta có thể tiếp sau tên và kiểu của nó bằng một khai báo rằng nó tham chiếu thuộc tính nào đó của một bảng nào đó (thuộc tính này phải là khoá chính hoặc duy nhất). Khai báo có dạng:

REFERENCE < bảng>(<thuộc tính>)

b) Cách thứ hai, chúng ta có thể thêm vào danh sách các thuộc tính trong một lệnh CREATE TABLE một hoặc nhiều khai báo phát biểu rằng một tập thuộc tính là một khoá ngoài. Sau đó chúng đưa ra tên bảng và các thuộc tính được khoá ngoài tham chiếu đến của bảng đó. Các thuộc tính này phải là các thuộc tính khoá. Khai báo có dạng như sau:

FOREIGN KEY (<các thuộc tính>) REFERENCE < Tênbảng> (các thuộc tính

được tham chiếu đến)

Ví dụ 2.3 Giả sử chúng ta muốn mô tả quan hệ

DỰÁN (Mã sốDA, TênDA, Địa điểmDA, Mã sốĐV)

có khoá chính là Mã số DA và khoá ngoài Mã số DV tham chiếu thuộc tính Mã số DV của quan hệ ĐƠN VỊ. Chúng ta có thể có hai cách mô tả như sau:

cách 1: **CREATE TABLE DỰ ÁN**

(TênDA	VARCHAR(15),	
Mã số DA	INT PRIMARY KEY,	
Địa điểm DA	VARCHAR(15),	
Mã số DV	INT REFERENCES ĐƠN VỊ(Mã số DV));	

cách 2: **CREATE TABLE DỰ ÁN**

(TênDA	VARCHAR(15),	
Mã số DA	INT PRIMARY KEY,	
Địa điểm DA	VARCHAR(15),	
Mã số DV	INT,	
FOREIGN	KEY(Mã số DV)	REFERENCES
ĐƠN VỊ(Mã số DV));		

Chú ý rằng thuộc tính được tham chiếu Mã số DV trong quan hệ ĐƠN VỊ phải là khoá của quan hệ đó. Ý nghĩa của việc mô tả khoá ngoài là ở chỗ mỗi khi một giá trị xuất hiện trong thành phần Mã số DV của một bộ của quan hệ DỰ ÁN thì giá trị đó cũng phải xuất hiện trong thành phần Mã số DV của một bộ giá trị nào đó của quan hệ ĐƠN VỊ. Có một ngoại trừ là thành phần Mã số DV của một bộ cụ thể của quan hệ DỰ ÁN có thể có giá trị NULL, khi đó sẽ không có đòi hỏi rằng NULL xuất hiện như giá trị của thành phần Mã số DV của ĐƠN VỊ (bởi vì Mã số DV là khoá chính).

### 2.1.5 Duy trì toàn vẹn tham chiếu

Chúng ta đã nhìn thấy làm thế nào để mô tả một khoá ngoài và chúng ta cũng biết rằng mô tả đó kéo theo việc một tập giá trị (khác NULL) đối với các thuộc tính khoá ngoài cũng phải xuất hiện trong các thuộc tính của quan hệ được tham chiếu. Ràng buộc này sẽ được duy trì thế nào khi có xảy ra việc cập nhật cơ sở dữ liệu? Có ba cách sau đây:

a) Chính sách ngầm định: Loại bỏ các vi phạm

SQL có một chính sách ngầm định là mọi cập nhật vi phạm ràng buộc toàn vẹn tham chiếu sẽ bị hệ thống loại ra. Ví dụ, xét ví dụ 2.3, trong đó đòi hỏi rằng một giá trị của MãSốĐV trong quan hệ DỰÁN cũng là một giá trị của MãSốĐV trong ĐƠNVI. Các hành động sau đây sẽ bị hệ thống loại bỏ:

- Chúng ta cố gắng chèn vào quan hệ DỰÁN một bộ giá trị mới mà giá trị của thuộc tính MãSốĐV của nó là khác NULL và không là giá trị của thuộc tính MãSốĐV của bất kỳ một bộ giá trị nào của quan hệ ĐƠNVI. Phép chèn bị hệ thống loại bỏ và bộ giá trị đó sẽ không bao giờ được chèn vào quan hệ

- Chúng ta cố gắng cập nhật một bộ của quan hệ DỰÁN để thay đổi thành phần MãSốĐV thành một giá trị khác NULL mà không là thành phần của bất kỳ bộ giá trị nào của quan hệ ĐƠNVI. Phép cập nhật bị loại bỏ và bộ không được thay đổi.

- Chúng ta cố gắng loại bỏ một bộ giá trị của quan hệ ĐƠNVI mà giá trị của thuộc tính MãSốĐV của nó xuất hiện như một giá trị của thuộc tính MãSốĐV của một hay nhiều bộ giá trị của quan hệ DỰÁN. Phép xoá bị loại bỏ và bộ vẫn còn trong ĐƠNVI.

- Chúng ta cố gắng cập nhật một bộ của quan hệ ĐƠNVI theo cách làm thay đổi giá trị của thuộc tính MãSốĐV và giá trị cũ của MãSốĐV là giá trị của thuộc tính MãSốĐV trong một bộ của quan hệ DỰÁN. Hệ thống loại bỏ sự thay đổi và ĐƠNVI vẫn như cũ.

## b) Chính sách Cascade

Có một cách khác để điều khiển việc xoá và cập nhật đối với một quan hệ được tham chiếu gọi là chính sách cascade. Một cách trực quan, các thay đổi đối với các thuộc tính được tham chiếu được bắt chước ở khoá ngoài.

Dưới chính sách cascade, khi chúng ta loại bỏ một bộ giá trị của quan hệ được tham chiếu, để duy trì toàn vẹn tham chiếu, hệ thống sẽ bỏ các bộ giá trị tương ứng của quan hệ tham chiếu. Các cập nhật cũng được thực hiện một cách tương tự. Nếu chúng ta thay đổi giá trị của khoá chính trong một bộ của quan hệ được tham chiếu, thì các giá trị của các thuộc tính tương ứng (khoá ngoài) trong các bộ giá trị của quan hệ tham chiếu cũng được thay đổi theo.



### c) Chính sách Set-Null

Một cách tiếp cận khác để giải quyết vấn đề là thay đổi giá trị của khoá ngoài của các bộ bị loại bỏ hoặc bị cập nhật thành NULL. Cách này gọi là chính sách Set-Null.

Các tùy chọn có thể được chọn đối với các phép cập nhật và xoá một cách độc lập và chúng được liệt kê ra cùng với khai báo của khoá ngoài. Chúng ta khai báo chúng bằng ON DELETE hoặc ON UPDATE theo sau là SET NULL hoặc CASCADE.

Ví dụ 2.4 Chúng ta hãy sửa đổi khai báo của quan hệ

DỰÁN(TênDA, MãSốDA, ĐịađiểmDA, MãSốĐV)

trong ví dụ 2.3 để chỉ rõ điều khiển xoá và cập nhật trong quan hệ ĐƠNVI.

Ta có khai báo như sau

- 1) **CREATE TABLE** DỰÁN
- 2) (TênDA VARCHAR(15),
- 3) MãSốDA INT PRIMARY KEY,
- 4) ĐịađiểmDA VARCHAR(15),
- 5) MãSốĐV INT REFERENCES ĐƠNVI(MãSốDA),
- 6) ON DELETE SET NULL
- 7) ON UPDATE CASCADE );

Khai báo trên có nghĩa là mỗi khi ta xoá một bộ giá trị của quan hệ ĐƠNVI ta sẽ làm cho giá trị của thuộc tính MãSốĐV của quan hệ DỰÁN tham chiếu đến nó trở thành NULL. Mỗi khi ta cập nhật giá trị MãSốĐV của một bộ giá trị của quan hệ ĐƠNVI, ta sẽ làm thay đổi của MãSốĐV trong bộ giá trị của quan hệ DỰÁN tham chiếu đến nó. Giá trị đã được cập nhật của MãSốĐV trong quan hệ ĐƠNVI là giá trị mới của MãSốĐV trong DỰÁN.

#### 2.1.6 Làm chậm việc kiểm tra ràng buộc.

Xét ví dụ 2.3, trong đó MãSốĐV trong DỰÁN là khoá ngoài tham chiếu đến MãSốĐV của ĐƠNVI. Nếu ta thực hiện chèn một bộ giá trị vào quan hệ DỰÁN

```
INSERT INTO DỰÁN
```

```
VALUES ('DA08', 25, 'Hà nội', 6);
```

thì chúng ta sẽ gặp rắc rối bởi vì không có bộ giá trị nào của quan hệ ĐƠNVI có giá trị 6 cho thuộc tính MãSốĐV. Như vậy, ta đã vi phạm ràng buộc khoá ngoài.

Một sửa đổi có thể được là trước tiên hãy chèn vào quan hệ DỰÁN một bộ giá trị không có thuộc tính MãSốĐV:

```
INSERT INTO DỰÁN (TênDA, MãSốDA, ĐịađiểmDA)
```

```
VALUES ('DA08', 25, 'Hà nội');
```

Thay đổi này tránh vi phạm ràng buộc bởi vì bộ này được chèn vào với giá trị null cho MãSốĐV và NULL trong một khoá ngoài không đòi hỏi phải kiểm tra sự tồn tại của bất kỳ giá trị nào ở trong cột được tham chiếu đến. Sau đó chúng ta chèn vào quan hệ ĐƠNVI một bộ giá trị mới có giá trị cho thuộc tính MãSốĐV là 6, cuối cùng ta cập nhật quan hệ DỰÁN Studio bằng lệnh:

```
UPDATE DỰÁN
```

```
SET MãSốĐV = 6
```

```
WHERE TênDA = 'DA08';
```

Nếu ta không sửa đổi quan hệ ĐƠNVI trước thì lệnh update này cũng sẽ vi phạm ràng buộc khoá ngoài. Tuy nhiên, trong một số trường hợp ta không thể sắp xếp một cách khôn ngoan các bước sửa đổi cơ sở dữ liệu như vậy.

Trong SQL để giải quyết khó khăn trên ta cần hai điểm:

1. Trước tiên ta cần có khả năng nhóm nhiều lệnh SQL (hai phép chèn, một vào DỰÁN và một vào ĐƠNVI) vào một đơn vị gọi là một giao tác (transaction).
2. Sau đó chúng ta cần một cách nói với hệ thống SQL không kiểm tra các ràng buộc cho đến sau khi toàn bộ giao tác được hoàn thành.

Để thực hiện điểm 2 chúng ta cần biết một số chi tiết:

a) Mọi ràng buộc (khóa, khóa ngoài hoặc ràng buộc khác) có thể được khai báo DEFERABLE hoặc NOT DEFERABLE. Khai báo NOT DEFERABLE là ngầm định và có nghĩa là mỗi một lần xuất hiện một sửa đổi cơ sở dữ liệu, ràng buộc được kiểm tra ngay sau đó nếu phép sửa đổi đòi hỏi rằng nó được kiểm tra. Tuy nhiên nếu ta khai báo một ràng buộc là DEFERABLE, thì chúng ta có lựa chọn bảo nó chờ đợi cho đến khi một giao tác hoàn thành trước khi kiểm tra ràng buộc.

b) Nếu một ràng buộc là chậm được (deferable), thì chúng ta cũng có thể khai báo nó là INITIALLY DEFERED hoặc INITIALLY IMMEDIATE. Trong trường hợp thứ nhất việc kiểm tra sẽ được trì hoãn đến hết của giao tác hiện tại, trừ khi chúng ta bảo hệ thống dừng việc trì hoãn ràng buộc này. Nếu mô tả INITIALLY IMMEDIATE, việc kiểm tra sẽ được làm trước mọi sửa đổi, nhưng bởi vì ràng buộc là trì hoãn được, chúng ta có quyền lựa chọn quyết định kiểm tra chậm về sau.

Ví dụ 2.5 Hình dưới đây biểu thị mô tả của DỰÁN được cải tiến để cho phép việc kiểm tra ràng buộc khóa ngoài được trì hoãn cho đến sau một giao tác. Ta cũng mô tả Mã sốĐV là UNIQUE với mục đích là nó có thể được các ràng buộc khóa ngoài của các quan hệ khác tham chiếu đến.

```
CREATE TABLE DỰÁN  
  (TênDA          VARCHAR(15),  
   Mã sốDA        INT PRIMARY KEY,  
   Địa điểmDA     VARCHAR(15),  
   Mã sốĐV        INT UNIQUE  
   REFERENCES ĐƠN VỊ(Mã sốDA)),  
  DEFERABLE INITIALLY DEFERRED);
```

Có hai điểm cần chú ý:

- Có thể đặt tên cho các ràng buộc
- Nếu một ràng buộc có một tên, ví dụ MyConstraint, thì chúng ta có thể đổi một ràng buộc chậm từ immediate thành deferred bằng lệnh SQL

```
SET CONSTRAINT MyConstraint DEFERRED ;
```

và chúng ta có thể đảo ngược xử lý bằng cách đổi từ DEFERRED ở trên thành IMMEDIATE.

## 2.2 CÁC RÀNG BUỘC TRÊN CÁC THUỘC TÍNH VÀ CÁC BỘ

Chúng ta đã xem xét các ràng buộc khoá, chúng buộc các thuộc tính nào đấy phải có các giá trị khác biệt trên tất cả các bộ giá trị của một quan hệ. Chúng ta cũng đã xem xét các ràng buộc khoá ngoài, chúng bắt tuân theo ràng buộc tham chiếu giữa các thuộc tính của hai quan hệ. Bây giờ chúng ta sẽ xem xét một loại ràng buộc quan trọng thứ ba: chúng là hạn chế các giá trị có thể xuất hiện trong các thành phần đối với một số thuộc tính. Các ràng buộc này có thể được diễn đạt bằng một trong hai cách sau:

1. Một ràng buộc trên thuộc tính trong định nghĩa của lược đồ quan hệ của nó, hoặc
2. Một ràng buộc trên một bộ giá trị. Ràng buộc này là một phần của lược đồ quan hệ, không liên quan đến bất kỳ thuộc tính nào của nó.

### 2.2.1 Các ràng buộc Not-Null

Một ràng buộc đơn giản gắn kết với một thuộc tính là NOT NULL. Hiệu quả của nó là cấm các bộ mà trong đó giá trị của thuộc tính này là NULL. Ràng buộc được mô tả bằng các từ khoá NOT NULL đi sau mô tả của thuộc tính trong lệnh CREATE TABLE

Ví dụ 2.6: Giả sử quan hệ DỰÁN đòi hỏi thuộc tính Mã sốĐV là xác định. Khi đó ta có mô tả của thuộc tính trong dòng 5) của ví dụ 2.4 được thay đổi như sau:

5) Mã sốĐV INT REFERENCES ĐƠNVI(Mã sốĐV) NOT NULL

Thay đổi này có nhiều hệ quả, chẳng hạn:

- Chúng ta không thể thêm vào quan hệ Studio một bộ giá trị bằng cách chỉ ra ba thuộc tính TênDA, Mã sốDA, Địa điểmDA bởi vì bộ thêm vào sẽ có giá trị NULL cho thành phần Mã sốĐV.

. Chúng ta không thể sử dụng chính sách set-null trong hoàn cảnh giống như dòng 6) của ví dụ 2.4, bởi vì nó yêu cầu hệ thống gán cả giá trị NULL cho Mã số ĐV để bảo toàn ràng buộc khoá ngoài.

### 2.2.2 Các ràng buộc kiểm tra (CHECK) dựa trên thuộc tính

Các ràng buộc phức tạp hơn có thể gắn với một mô tả thuộc tính bằng từ khoá CHECK sau đó là một điều kiện phải thoả mãn đối với mọi giá trị của thuộc tính này. Điều kiện được đặt trong dấu ngoặc. Trên thực tế, một ràng buộc CHECK dựa trên thuộc tính giống như một hạn chế đơn giản trên các giá trị, như là việc liệt kê các giá trị hợp thức hoặc là một bất phương trình số học. Tuy nhiên, về nguyên tắc, điều kiện có thể là bất kỳ cái gì có thể đi sau WHERE trong một truy vấn SQL. Điều kiện này có thể tham chiếu đến thuộc tính bị ràng buộc bằng cách sử dụng tên của thuộc tính đó trong biểu thức của nó. Tuy nhiên, nếu điều kiện tham chiếu đến các quan hệ khác hoặc các thuộc tính khác của quan hệ thì quan hệ phải được đưa vào trong mệnh đề FROM của một truy vấn con (ngay cả nếu quan hệ được tham chiếu đến là một quan hệ chứa thuộc tính bị kiểm tra).

Một ràng buộc CHECK dựa trên thuộc tính được kiểm tra khi một bộ nào đấy nhận một giá trị mới cho thuộc tính này. Giá trị mới có thể được đưa vào bằng một sửa đổi đối với bộ hoặc có thể là một phần của bộ được thêm vào. Nếu giá trị mới vi phạm ràng buộc thì phép sửa đổi bị loại bỏ. Chúng ta sẽ thấy trong ví dụ 2.7, ràng buộc CHECK dựa trên thuộc tính sẽ không được kiểm tra nếu một sửa đổi cơ sở dữ liệu không làm thay đổi giá trị của thuộc tính gắn với ràng buộc. Hạn chế đó có thể dẫn đến ràng buộc trở nên bị vi phạm. Trước tiên ta hãy xét một ví dụ đơn giản về kiểm tra dựa trên thuộc tính.

Ví dụ 2.7 Giả sử chúng ta muốn đòi hỏi rằng các Mã số ĐV chỉ gồm có một chữ số. Ta có thể sửa đổi dòng 5) của ví dụ 2.4, một mô tả của lược đồ đối với quan hệ DỰ ÁN như sau:

- 4) Mã số DA INT REFERENCES ĐƠN VỊ (Mã số DA)  
CHECK (Mã số DA < 10)

Việc đề cập đến các thuộc tính khác của quan hệ hoặc các quan hệ khác trong điều kiện là được phép. Để làm điều đó, trong điều kiện phải có truy vấn con. Như chúng ta đã nói, điều kiện có thể là bất kỳ cái gì có thể đi sau WHERE trong một lệnh select-from-where của SQL. Tuy nhiên, chúng ta cần nhận thấy rằng việc kiểm tra ràng buộc chỉ được kết hợp với thuộc tính cần kiểm tra chứ không phải với mỗi quan hệ hoặc thuộc tính kể ra trong ràng buộc. Kết quả là một điều kiện phức tạp có thể trở thành sai nếu một phần tử nào đó khác với thuộc tính cần kiểm tra thay đổi...

Ví dụ 2.8 Chúng ta có thể giả thiết rằng ta có thể thay thế một ràng buộc toàn vẹn tham chiếu bằng một ràng buộc CHECK dựa trên thuộc tính đòi hỏi sự tồn tại của giá trị được tham chiếu. Sau đây là một sự cố gắng sai lầm nhằm thay thế đòi hỏi rằng giá trị của MãSốDA trong một bộ (TênDA, MãSốDA, ĐịađiểmDA, MãSốĐV) của quan hệ DỰÁN phải xuất hiện trong thành phần MãSốĐV của một bộ nào đấy của quan hệ ĐƠNVI. Giả sử dòng 5) của ví dụ 2.4 được thay thế bằng:

```
4) MãSốĐV INT CHECK (MãSốĐV IN(SELECT MãSốĐV FROM ĐƠNVI))
```

Lệnh này là một ràng buộc CHECK dựa trên thuộc tính hợp lệ nhưng ta hãy xem xét hiệu quả của nó.

- Nếu ta cố gắng chèn một bộ mới vào DỰÁN và bộ đó có giá trị của MãSốĐV không là mã số của một đơn vị nào cả, khi đó phép chèn bị loại bỏ.
- Nếu ta cố gắng cập nhật thành phần MãSốĐV của một bộ DỰÁN, và giá trị mới không phải là một giá trị của thuộc tính MãSốĐV trong một bộ nào cả của ĐƠNVI, phép cập nhật bị loại bỏ.
- Tuy nhiên, nếu chúng ta thay đổi quan hệ ĐƠNVI bằng cách loại bỏ một bộ giá trị, thay đổi đó là không nhìn thấy đối với ràng buộc CHECK ở trên. Như vậy, phép loại bỏ được chấp nhận mặc dù ràng buộc CHECK dựa vào thuộc tính trên MãSốĐV bây giờ bị vi phạm.

### 2.2.3 Các ràng buộc kiểm tra (CHECK) dựa trên bộ giá trị.

Để khai báo một ràng buộc trên các bộ của một bảng R, khi chúng ta định nghĩa bảng đó bằng lệnh CREATE TABLE, chúng ta có thể thêm vào danh sách các khai báo thuộc tính, khoá và khoá ngoài từ khoá CHECK theo sau là một điều kiện đặt trong dấu ngoặc. Điều kiện này có thể là bất cứ cái gì có thể xuất hiện trong mệnh đề WHERE. Nó được cài đặt như là một điều kiện về một bộ trong bảng R, và các thuộc tính của R có thể được tham chiếu đến bằng tên trong biểu thức này. Tuy nhiên, cũng như đối với các ràng buộc CHECK dựa trên thuộc tính, điều kiện cũng có thể đề cập đến các quan hệ khác hoặc các bộ giá trị khác của cùng quan hệ R ở trong các truy vấn con.

Điều kiện của một ràng buộc CHECK dựa trên bộ được kiểm tra mỗi khi một bộ được chèn vào R và mỗi khi một bộ của R được cập nhật và được đánh giá đối với bộ được cập nhật hoặc bộ mới được chèn vào. Nếu điều kiện là sai đối với bộ đó thì ràng buộc bị vi phạm và phép chèn hoặc phép cập nhật sinh ra sự vi phạm đó sẽ bị loại bỏ. Tuy nhiên, nếu điều kiện đề cập đến một quan hệ nào đó (thậm chí đến chính R) trong một truy vấn con, và một thay đổi đối với quan hệ này gây ra việc điều kiện trở thành sai đối với một bộ nào đó của R thì kiểm tra không ngăn cản thay đổi đó. Như vậy, giống như một CHECK dựa trên thuộc tính, một CHECK dựa trên bộ là không thấy được đối với các quan hệ khác.

Mặc dù các kiểm tra dựa trên bộ có thể gồm các điều kiện rất phức tạp nhưng nên tránh trường hợp đó bởi vì nó có thể bị vi phạm dưới các điều kiện nào đấy. Nếu các kiểm tra dựa trên bộ chỉ chứa các thuộc tính của bộ cần kiểm tra thì nó luôn luôn đúng.

Ví dụ 2.9 Chúng ta viết lệnh CREATE TABLE đối với quan hệ GIÁOVIÊN, trong đó ta thêm vào một ràng buộc: nếu một giáo viên là nam giới thì tên của anh ta không được bắt đầu bằng “Ms.”:

- 1) CREATE TABLE GIÁOVIÊN (
- 2) Tên CHAR(30) PRIMARY KEY,
- 3) Địa chỉ VARCHAR(255),
- 4) Giớitính CHAR(1),

5) Ngày sinh DATE,

6) CHECK (Giới tính = 'F' OR Tên NOT LIKE 'Ms.%');

Dòng 6) ở trên mô tả ràng buộc. Điều kiện của ràng buộc này là đúng đối với các giáo viên nữ hoặc các giáo viên khác có tên không bắt đầu bằng 'Ms.'. Các bộ làm cho điều kiện trở thành sai là các bộ có giới tính là nam và tên bắt đầu bằng 'Ms.'.

## 2.3 SỬA ĐỔI CÁC RÀNG BUỘC

Có thể thêm, sửa đổi, hoặc loại bỏ các ràng buộc tại mọi thời điểm. Cách diễn đạt các sửa đổi đó phụ thuộc vào việc ràng buộc liên quan đến một thuộc tính, một bộ giá trị, một bảng hoặc một lược đồ cơ sở dữ liệu.

### 2.3.1 Đặt tên cho các ràng buộc

Để sửa đổi hoặc xoá một ràng buộc đang tồn tại điều cần thiết là ràng buộc phải có một tên. Để làm điều đó, chúng ta đặt trước ràng buộc từ khoá CONSTRAINT và một tên cho ràng buộc này.

Ví dụ 2.10:

Tên CHAR(30) CONSTRAINT Tên là khóa PRIMARY KEY,

Giới tính CHAR(1) CONSTRAINT Giá trị GT

CHECK (gender IN ('F', 'M')),

CONSTRAINT Tên đúng

CHECK (Giới tính = 'F' OR Tên NOT LIKE 'Ms.%'),

### 2.3.2 Sửa đổi các ràng buộc trên các bảng

Trong phần trước chúng ta đã nói rằng ta có thể chuyển việc kiểm tra một ràng buộc từ tức khắc (immediate) sang chậm (deferred) với lệnh SET CONSTRAINT. Các thay đổi khác đối với các ràng buộc được thực hiện với lệnh ALTER TABLE. Chúng ta cũng đã sử dụng lệnh này trong phần trước để thêm hoặc loại bỏ các thuộc tính.

Các lệnh đó cũng có thể được sử dụng để sửa đổi các ràng buộc. ALTER TABLE được sử dụng đối với cả hai kiểm tra: kiểm tra dựa trên thuộc tính và kiểm tra dựa trên bộ. Chúng ta có thể dừng một ràng buộc với từ khoá



DROP và tên của ràng buộc bị dừng. Chúng ta cũng có thể thêm một ràng buộc với từ khoá ADD, tiếp theo là ràng buộc được thêm vào. Tuy nhiên, cần để ý rằng chúng ta chỉ có thể thêm một ràng buộc vào một bảng khi ràng buộc đó thoả mãn đối với trạng thái hiện tại của bảng.

Ví dụ 2.11: Ba lệnh sau đây làm dừng các ràng buộc đối với quan hệ GIÁOVIÊN:

```
ALTER TABLE GIÁOVIÊN DROP CONSTRAINT Tênlàkhóa ;
```

```
ALTER TABLE GIÁOVIÊN DROP CONSTRAINT GiátrịGT ;
```

```
ALTER TABLE GIÁOVIÊN DROP CONSTRAINT Tênđúng ;
```

Bây giờ, giả sử ta muốn khôi phục lại các ràng buộc đó, ta viết các lệnh sau:

```
ALTER TABLE GIÁOVIÊN ADD CONSTRAINT Tênlàkhóa  
PRIMARY KEY (Tên);
```

```
ALTER TABLE GIÁOVIÊN ADD CONSTRAINT GiátrịGT  
CHECK (Giới tính IN ('F', 'M'));
```

```
ALTER TABLE GIÁOVIÊN ADD CONSTRAINT RightTitle  
CHECK (Giới tính = 'F' OR Tên NOT LIKE 'Ms.%' ;
```

Các ràng buộc này trở thành các ràng buộc dựa trên bộ chứ không còn là các ràng buộc dựa trên thuộc tính.

## 2.4 CÁC RÀNG BUỘC MỨC LƯỢC ĐỒ VÀ CÁC TRIGGER

Các dạng phần tử tích cực mạnh mẽ nhất trong SQL không kết hợp với các bộ hoặc các thành phần của các bộ cụ thể. Các phần tử này được gọi là các khẳng định (assertion) và các trigger, chúng là một thành phần của lược đồ cơ sở dữ liệu, ngang hàng với các quan hệ và các khung nhìn.

. Một khẳng định là một biểu thức SQL có giá trị lô gic, nó phải đúng tại mọi thời điểm.

. Một trigger là một loạt hành động kết hợp với các sự kiện nào đó, chẳng hạn như chèn vào một quan hệ cụ thể, và chúng được thực hiện khi các sự kiện này xảy ra.

### 2.4.1 Các khẳng định (assertion)

Chuẩn SQL đề nghị một dạng khẳng định đơn giản cho phép chúng ta bắt buộc một điều kiện nào đó. Giống như các phần tử lược đồ khác, chúng ta mô tả một khẳng định bằng lệnh CREATE. Dạng của một khẳng định là:

1. Các từ khóa CREATE ASSERTION,
2. Tên của khẳng định
3. Từ khóa CHECK, và
4. Một điều kiện được đặt trong dấu ngoặc.

Như vậy, dạng của lệnh này là

```
CREATE ASSERTION < Tên > CHECK < điều kiện >
```

Điều kiện trong assertion phải đúng khi assertion được tạo ra và phải luôn luôn vẫn còn đúng; bất kỳ một sửa đổi cơ sở dữ liệu nào làm cho nó trở thành sai sẽ bị loại bỏ. Nhớ lại rằng các kiểu ràng buộc CHECK khác mà chúng ta đã xét có thể bị vi phạm dưới một số điều kiện.

Có một sự khác nhau giữa cách viết các ràng buộc CHECK dựa trên bộ và cách viết assertion. Các kiểm tra dựa trên bộ có thể tham chiếu đến các thuộc tính của quan hệ có kiểm tra này xuất hiện trong mô tả của nó. Ví dụ, trong dòng 6) của ví dụ 2.8 chúng ta sử dụng các thuộc tính Giới tính và Tên mà không nói chúng đi đến từ đâu. Chúng tham chiếu đến các thành phần của một bộ được chèn vào hoặc được cập nhật trong bảng GIÁOVIÊN bởi vì bảng đó là một trong các bảng được mô tả trong lệnh CREATE TABLE.

Điều kiện của một assertion không có đặc quyền như vậy. Các thuộc tính bất kỳ được tham chiếu đến trong điều kiện phải được giới thiệu trong assertion, thường là bằng việc chỉ ra quan hệ của chúng trong một biểu thức select-from-where. Bởi vì điều kiện phải có một giá trị logic, việc nhóm các kết quả của điều kiện theo một cách nào đó để tạo ra một lựa chọn true/false đơn là việc bình thường. Ví dụ, chúng ta có thể viết điều kiện như là một

biểu thức sản xuất ra một quan hệ mà NOT EXISTS được áp dụng cho quan hệ đó; điều đó có nghĩa là, ràng buộc là quan hệ này luôn luôn rỗng. Một cách khác, chúng ta có thể áp dụng một phép toán nhóm như là SUM cho một cột của một quan hệ và so sánh nó với một hằng. Ví dụ, chúng ta có thể đòi hỏi rằng tổng luôn luôn bé hơn một giá trị có giới hạn nào đó.

Ví dụ 2.12: Giả sử chúng ta muốn yêu cầu rằng muốn trở thành người quản lý của một đơn vị thì phải có lương ít nhất là 4000. Ta sẽ khai báo một assertion với với mục đích rằng tập hợp các đơn vị có người quản lý với lương nhỏ hơn 4000 là rỗng. Assertion này cần có hai quan hệ NHÂNVIÊN và ĐƠNVI.

```
CREATE ASSERTION Quanly CHECK
(NOT EXISTS
(SELECT *
FROM NHÂNVIÊN NV, ĐƠNVI ĐV
WHERE NV.MãSốNV = ĐV.Mã sốNQL
AND Lương <4000);
```

Nhân tiện, cần chú ý rằng mặc dù ràng buộc này có hai quan hệ, chúng ta có thể viết nó như các ràng buộc CHECK dựa trên bộ trên hai quan hệ hơn là như một assertion đơn. Ví dụ, chúng ta có thể viết như sau

```
CREATE TABLE ĐƠNVI
(TênĐV VARCHAR(15) NOT NULL,
MãSốĐV INT PRIMARY KEY,
Ngàybắtđầu DATE,
MãSốNQL CHAR(9) REFERENCES
NHÂNVIÊN(MãSốNV),
CHECK (MãSốNQL NOT IN
(SELECT MãSốNV
FROM NHÂNVIÊN
WHERE Lương < 4000)));
```

Tuy nhiên, chú ý rằng ràng buộc này chỉ sẽ được kiểm tra khi có một thay đổi xảy ra đối với quan hệ của nó, quan hệ ĐƠN VỊ. Có thể nó không chặn được tình trạng có những người quản lý được ghi vào quan hệ NHÂN VIÊN nhưng lại có lương < 4000. Để nhận được hiệu quả đầy đủ của assertion, chúng ta phải thêm một ràng buộc khác vào khai báo của bảng NHÂN VIÊN, đòi hỏi rằng lương của một nhân viên phải >=4000 nếu anh ta là người quản lý một đơn vị.

Ví dụ 2.13 Assertion sau đây nói rằng tổng lương của mỗi đơn vị không được vượt quá 100000. Nó bao hàm quan hệ NHÂN VIÊN.

```
CREATE ASSERTION Tổnglương
CHECK (100000 >= ALL
      (SELECT SUM(Lương)
       FROM NHÂNVIÊN
       GROUP BY MãSốĐV));
```

Vì ràng buộc này chỉ bao hàm có một quan hệ NHÂN VIÊN, nó có thể được trình bày như một ràng buộc CHECK dựa trên biến bộ trong lược đồ đối với NHÂN VIÊN hơn là như một assertion. Vì vậy, chúng ta có thể thêm vào định nghĩa của bảng NHÂN VIÊN ràng buộc CHECK dựa trên biến bộ như sau:

```
CHECK (100000 >= ALL
      (SELECT SUM(Lương) FROM NHÂNVIÊN GROUP BY MãSốĐV));
```

Đề ý rằng, về nguyên tắc, điều kiện này áp dụng cho mỗi bộ của bảng NHÂN VIÊN. Tuy nhiên, nó không kể ra các thuộc tính của bộ một cách rõ ràng và mọi công việc được thực hiện trong truy vấn con. Cũng đề ý thêm rằng nếu được cài đặt như một ràng buộc dựa trên bộ thì việc kiểm tra không thể thực hiện trên phép xóa một bộ ra khỏi quan hệ NHÂN VIÊN. Trong ví dụ này, sự khác nhau đó không gây ra tai hại bởi vì nếu ràng buộc được thỏa mãn trước khi xóa thì sau khi xóa nó cũng được thỏa mãn. Tuy nhiên, nếu ràng buộc không phải là cận trên mà là cận dưới của tổng lương theo từng

đơn vị thì chúng ta có thể tìm thấy rằng buộc bị vi phạm ngay cả khi chúng ta viết nó như là một kiểm tra dựa trên bộ mà không phải là một assertion.

Cuối cùng, ta có thể bỏ một assertion. Lệnh làm việc đó cũng tuân theo mẫu đối với mọi phần tử của lược đồ cơ sở dữ liệu.

DROP ASSERTION <Tên của assertion>.

## So sánh các ràng buộc

Bảng sau đây liệt kê các khác nhau cơ bản giữa các kiểm tra dựa trên thuộc tính, các kiểm tra dựa trên bộ và assertion

<i>Kiểu ràng buộc</i>	<i>Được mô tả ở đâu</i>	<i>Khi nào được kích hoạt</i>	<i>Được đảm bảo đúng ?</i>
CHECK dựa trên thuộc tính	Cùng với thuộc tính	Trên phép chèn vào quan hệ hoặc cập nhật thuộc tính	Không đảm bảo nếu có truy vấn con
CHECK dựa trên bộ	Phần tử của lược đồ quan hệ	Trên phép chèn vào quan hệ hoặc cập nhật bộ	Không đảm bảo nếu có truy vấn con
Assertion	Phần tử của lược đồ cơ sở dữ liệu	Trên một thay đổi đối với quan hệ được kể ra	Đảm bảo

### 2.4.2 Trigger

Các trigger (hay còn gọi là các quy tắc ECA – event-condition-action rules) khác với các loại ràng buộc thảo luận ở trên ở ba điểm:

1. Các trigger chỉ được đánh thức khi xảy ra một sự kiện do người lập trình cơ sở dữ liệu chỉ ra. Các loại sự kiện cho phép thường là chèn, xóa

hoặc cập nhật đối với một quan hệ cụ thể. Một loại sự kiện khác được cho phép trong nhiều hệ thống SQL là một kết thúc giao tác.

2. Thay cho việc ngăn ngừa tức khắc sự kiện đã đánh thức nó, trigger kiểm tra một điều kiện. Nếu điều kiện không thỏa mãn thì chẳng có cái gì liên quan tới trigger xảy ra trong trả lời cho sự kiện này.
3. Nếu điều kiện của trigger được thỏa mãn thì hệ quản trị cơ sở dữ liệu sẽ thực hiện hành động liên kết với trigger. Hành động đó có thể là ngăn ngừa sự kiện xảy ra hoặc loại bỏ (undo) sự kiện (ví dụ như xóa bộ được chèn vào). Nói tóm lại, hành động có thể là một hệ quả nào đó của các phép toán cơ sở dữ liệu, ngay cả các phép toán không kết nối theo một cách nào đó với sự kiện nổ ra

### **Trigger trong SQL**

Một câu lệnh trigger của SQL cho người sử dụng một số các tùy chọn trong các phần sự kiện, điều kiện và hành động. Sau đây là một số đặc trưng chính:

1. Hành động có thể được thực hiện hoặc trước hoặc sau khi sự kiện nổ ra
2. Hành động có thể tham chiếu đến cả các giá trị cũ và giá trị mới của các bộ được chèn, xóa hoặc cập nhật trong sự kiện làm bùng nổ hành động.
3. Các sự kiện cập nhật có thể được giới hạn đến một thuộc tính hoặc một tập thuộc tính cụ thể.
4. Một điều kiện có thể được chỉ ra bằng mệnh đề WHEN; hành động chỉ được thực hiện khi quy tắc được bùng nổ và điều kiện thỏa mãn khi xảy ra sự kiện bùng nổ
5. Người lập trình có tùy chọn chỉ ra rằng hành động được thực hiện hoặc:
  - a) Một lần đối với mỗi bộ được cập nhật
  - b) Một lần đối với tất cả các bộ được thay đổi trong một phép toán cơ sở dữ liệu.

Trước khi đưa ra chi tiết về cú pháp của trigger, chúng ta hãy xét một ví dụ minh họa các điểm quan trọng về cú pháp cũng như về ngữ nghĩa. Trong ví dụ này, trigger thực hiện một lần đối với mỗi bộ được cập nhật

Ví dụ 2.14: Viết một trigger của SQL áp dụng cho quan hệ NHÂNVIÊN. Nó được kích hoạt do các cập nhật đối với thuộc tính Lương. Hiệu quả của trigger này là chặn ý định làm giảm lương của một nhân viên. Mô tả của trigger như sau:

- 1) CREATE TRIGGER LươngTrigger
- 2) AFTER UPDATE OF Lương ON NHÂNVIÊN
- 3) REFERENCING
- 4) OLD ROW AS OldTuple
- 5) NEW ROW AS NewTuple
- 6) FOR EACH ROW
- 7) WHEN (OldTuple.Lương > Newtuple.Lương)
- 8) UPDATE NHÂNVIÊN
- 9) SET Lương = Oldtuple.Lương
- 10) WHERE MãSốNV = Newtuple.MãSốNV;

Dòng 1) đưa ra mô tả với các từ khóa CREATE TRIGGER và tên của trigger. Dòng 2) cung cấp sự kiện bùng nổ, đó là cập nhật thuộc tính Lương của quan hệ NHÂNVIÊN. Dòng 3) đến dòng 5) thiết lập một cách để các phần hành động và điều kiện nói về bộ cũ (bộ trước khi cập nhật) và bộ mới (bộ sau khi cập nhật). Các bộ này sẽ được tham chiếu đến như là Oldtuple và Newtuple theo mô tả ở dòng 4) và dòng 5) tương ứng. Trong điều kiện và hành động, các tên này có thể được sử dụng như là chúng là các biến bộ được mô tả trong mệnh đề FROM của một truy vấn SQL thông thường. Dòng 6), câu FOR EACH ROW, biểu thị đòi hỏi rằng trigger này được thực hiện một lần đối với mỗi một bộ được cập nhật. Nếu không có câu này hoặc nó được thay bằng ngầm định FOR EACH STATEMENT, thì trigger sẽ xuất hiện một lần đối với một lệnh SQL, không quan tâm đến sự kiện bùng nổ đã thay đổi các bộ bao nhiêu lần. Chúng ta không thể khai báo bí danh cho các hàng mới và các hàng cũ nhưng chúng ta có thể sử dụng OLD TABLE và NEW TABLE sẽ đưa ra sau đây. Dòng 7) là phần điều kiện của trigger. Nó nói rằng chúng ta chỉ thực hiện hành động khi lương mới thấp hơn lương cũ,

nghĩa là lương của một nhân viên bị cắt giảm. Các dòng 8) đến 10) tạo nên phần hành động. Hành động này là một lệnh cập nhật của SQL thông thường, nó có tác động khôi phục lương của nhân viên thành lương trước khi cập nhật. Để ý rằng về nguyên tắc, mỗi một bộ của NHÂNVIÊN được xem xét để cập nhật, nhưng mệnh đề WHERE của dòng 10) đảm bảo rằng chỉ có bộ được cập nhật là sẽ bị ảnh hưởng.

Sau đây chúng ta sẽ đưa ra các tùy chọn do trigger cung cấp và biểu diễn các tùy chọn đó như thế nào.

- Dòng 2) của ví dụ trên nói rằng hành động của quy tắc được thực hiện sau sự kiện xảy ra như được chỉ ra bằng từ khóa AFTER. Chúng ta có thể thay thế AFTER bằng BEFORE, trong trường hợp này điều kiện WHEN được kiểm tra trước khi sự kiện xảy ra, nghĩa là trước khi sự cập nhật làm đánh thức trigger được thực hiện đối với cơ sở dữ liệu. Nếu điều kiện là đúng thì hành động của trigger được thực hiện. Sau đó, sự kiện đánh thức trigger được thực hiện mà không cần quan tâm đến điều kiện có đúng hay không.

- Ngoài UPDATE, các sự kiện bùng nổ khác là INSERT, DELETE. Mệnh đề OF Lương trong dòng 2) là tùy chọn đối với sự kiện UPDATE và nếu có mặt thì nó xác định sự kiện chỉ là cập nhật của các thuộc tính được liệt kê sau từ khóa OF. Một mệnh đề OF không được cho phép đối với các sự kiện INSERT hoặc DELETE; các sự kiện này chỉ có nghĩa đối với các bộ toàn vẹn.

- Mệnh đề WHEN là tùy chọn. Nếu không có nó thì hành động được thực hiện mỗi khi trigger được đánh thức.

- Trong khi chúng ta chỉ ra một câu lệnh SQL đơn như là một hành động, Hành động có thể gồm nhiều các câu lệnh như vậy. Các câu lệnh cách nhau bằng dấu hai chấm và được đặt trong cặp BEGIN ... END.

- Khi một sự kiện bùng nổ là một cập nhật thì sẽ có các bộ cũ và các bộ mới, đó là các bộ trước và sau cập nhật tương ứng. Chúng ta cho các bộ này các tên OLD ROW AS VÀ NEW ROW AS như đã thấy trong các dòng 4) và 5). Nếu sự kiện bùng nổ là một phép chèn thì chúng ta có thể sử dụng



mệnh đề NEW ROW AS để đặt tên cho bộ được chèn vào và không cho phép có OLD ROW AS. Ngược lại, trong phép xóa OLD ROW AS được sử dụng để đặt tên cho bộ bị xóa và không cho phép có NEW ROW AS.

- Nếu chúng ta bỏ qua FOR EACH ROW ở dòng 6) thì trigger mức dòng như ở ví dụ trên sẽ trở thành trigger mức lệnh. Một trigger mức lệnh được thực hiện một lần khi một lệnh của một kiểu thích hợp được thực hiện, không quan tâm đến bao nhiêu hàng – không, một, nhiều - có ảnh hưởng. Ví dụ, nếu chúng ta cập nhật toàn bộ bảng với một lệnh cập nhật SQL, một trigger cập nhật mức lệnh sẽ chỉ thực hiện một lần, trong khi đó một trigger mức bộ sẽ thực hiện một lần đối với mỗi bộ được cập nhật. Trong một trigger mức lệnh, chúng ta không thể tham chiếu đến các bộ cũ và các bộ mới một cách trực tiếp giống như chúng ta đã nói trong các dòng 4) và 5). Tuy nhiên, một trigger bất kỳ - mức dòng hoặc mức bộ - có thể tham chiếu đến quan hệ của các bộ cũ (các bộ bị xóa hoặc phiên bản cũ của các bộ được cập nhật) và quan hệ của các bộ mới (các bộ được chèn vào hoặc phiên bản mới của các bộ được cập nhật) bằng việc sử dụng các mô tả giống như OLD TABLE AS OldStuff hoặc NEW TABLE AS NewStuff.

Ví dụ 2.15: Giả sử chúng ta muốn ngăn ngừa trung bình lương của các nhân viên xuống dưới 3000. Ràng buộc này có thể bị vi phạm bằng một phép chèn, một phép xóa hoặc bằng một phép cập nhật đối với Lương trong bảng NHÂNVIÊN. Điểm tế nhị là ở chỗ, trong một lệnh INSERT hoặc UPDATE chúng ta có thể chèn vào hoặc sửa đổi nhiều bộ của NHÂNVIÊN và trong quá trình cập nhật, trung bình lương có thể tạm thấp xuống dưới 3000 và sau đó lại vượt lên trên 3000 tại thời điểm sự cập nhật hoàn thành. Chúng ta chỉ muốn loại bỏ toàn bộ tập hợp các cập nhật nếu trung bình lương xuống dưới 3000 ở cuối mỗi lệnh.

Việc viết một trigger cho mỗi sự kiện chèn, xóa, cập nhật của quan hệ NHÂNVIÊN là cần thiết. Sau đây là trigger cho sự kiện cập nhật. Các trigger cho chèn và xóa cũng tương tự.

- 1) CREATE TRIGGER TB LươngTrigger
- 2) AFTER UPDATE OF Lương ON NHÂNVIÊN

- 3) REFERENCING
- 4) OLD TABLE AS OldStuff
- 5) NEW TABLE AS NewStuff
- 6) FOR EACH STATEMENT
- 7) WHEN (3000 > (SELECT AVG(Luong) FROM NHÂNVIÊN))
- 8) BEGIN
- 9) DELETE FROM NHÂNVIÊN
- 10) WHERE (Họđệm,Tên, Mã sốNV, Ngàysinh, Địa chỉ, Gióitính,Luong, Mã sốNGS, Mã sốĐV) IN NewStuff ;
- 11) INSERT INTO NHÂNVIÊN
- 12) (SELECT \* FROM OldStuff)
- 13) END;

Các dòng từ 3) đến 5) khai báo rằng NewStuff và OldStuff là tên của các quan hệ chứa các bộ mới và các bộ cũ dính dáng đến phép toán quan hệ làm đánh thức trigger của chúng ta. Chú ý rằng một phép toán quan hệ có thể sửa đổi nhiều bộ của một quan hệ và nếu một lệnh như vậy được thực hiện thì sẽ có nhiều bộ trong Newstuff và OldStuff.

Nếu một phép toán là một UPDATE thì NewStuff và OldStuff tương ứng là phiên bản mới và cũ của các bộ được cập nhật. Nếu trigger tương tự được viết cho phép xóa thì các bộ bị xóa sẽ ở trong OldStuff và sẽ không có mô tả tên quan hệ NewStuff cho NEW TABLE như trong trigger này. Cũng như vậy, trong trigger tương tự cho phép chèn, các bộ mới sẽ ở trong NewStuff và không có khai báo cho OldStuff.

Dòng 6) nói với ta rằng trigger này được thực hiện một lần cho một lệnh, không cần quan tâm đến bao nhiêu bộ đã được cập nhật Dòng 7) là một điều kiện. Điều kiện này được thỏa mãn nếu trung bình lương sau khi cập nhật là nhỏ hơn 3000.

Hành động từ dòng 8) đến dòng 13) gồm hai lệnh làm khôi phục quan hệ NHÂNVIÊN cũ nếu điều kiện của mệnh đề WHEN được thỏa mãn; nghĩa là

trung bình lương thấp quá. Các dòng 9) và 10) loại bỏ tất cả các bộ mới (các bộ đã được cập nhật) còn các dòng 11) và 12) khôi phục các bộ như chúng đã có trước khi cập nhật.

### **Các trigger Thay vì (Instead-Of)**

Có một đặc trưng tiện lợi của trigger chưa có trong chuẩn SQL-99 nhưng đã có mặt trong các cuộc thảo luận về chuẩn và được một số hệ thống thương mại hỗ trợ. Mở rộng này cho phép thay thế BEFORE hoặc AFTER bằng INSTEAD OF, ý nghĩa của nó là khi một sự kiện đánh thức một trigger, hành động của trigger sẽ được thực hiện thay cho bản thân sự kiện.

Khả năng này là bé nhỏ khi một trigger được dùng trên một bảng được lưu giữ, nhưng nó rất mạnh khi được sử dụng trên một khung nhìn. Lý do là ở chỗ chúng ta không thể cập nhật một khung nhìn một cách thật sự. Một trigger instead of ngăn chặn âm mưu cập nhật một khung nhìn và ở vị trí của nó thực hiện bất kỳ hành động nào mà người thiết kế cơ sở dữ liệu cho là thích hợp. Sau đây là một ví dụ mẫu:

Ví dụ 2.16 Ta hãy xem lại định nghĩa khung nhìn ở ví dụ 40 chương 1:

- 1) CREATE VIEW NVHÀNỘI AS
- 2) SELECT MăsốNV, Họđệm, Tên, Lương, MăsốĐV
- 3) FROM NHÂNVIÊN
- 4) WHERE Địachỉ = 'Hà nội' ;

Như chúng ta đã thảo luận, khung nhìn này là cập nhật được nhưng nó có một kẽ hở không được tôn trọng là khi ta chèn một bộ vào NVHÀNỘI, hệ thống không thể suy diễn được giá trị của thuộc tính Địachỉ chính là 'Hà nội', vì vậy thuộc tính là NULL trong bộ nhân viên được chèn vào.

Sẽ có một kết quả tốt hơn nếu chúng ta tạo ra một trigger instead of trên khung nhìn này như sau

- 1) CREATE TRIGGER NVInsert
- 2) INSTEAD OF INSERT ON NVHÀNỘI
- 3) REFERENCING NEW ROW AS NewRow

4) FOR EACH ROW

5) INSERT INTO NHÂNVIÊN (Mã sốNV, Họ tên, Tên, Lương, Mã sốĐV  
, Địa chỉ)

6) VALUES ('NV065', 'Nguyễn Đình', 'Thị', 4500, 4, 'Hà nội');

Chúng ta nhìn thấy từ khóa INSTEAD OF ở dòng 2) chứng minh rằng một phép chèn vào NVHÀNỘI sẽ không bao giờ xảy ra.

Hơn nữa, các dòng 5) và 6) là hành động thay thế cho phép chèn bị ngăn chặn. Có một phép chèn vào NHÂNVIÊN, nó chỉ ra các thuộc tính, địa chỉ. Các thuộc tính Mã sốNV, Họ tên, Tên, Lương, Mã sốĐV là các thuộc tính từ bộ mà ta cố gắng chèn vào khung nhìn; Chúng ta tham chiếu đến các giá trị này bằng biến bộ NewRow được mô tả ở dòng 3). Biến bộ này biểu thị bộ mà chúng ta muốn chèn. Giá trị của thuộc tính Địa chỉ là hằng 'Hà nội'. Giá trị này không phải là thành phần của bộ được chèn. Đúng hơn là chúng ta giả thiết đó là địa chỉ đúng đối với bộ nhân viên được chèn vào bởi vì phép chèn đi qua khung nhìn NVHÀNỘI.

## 2.5 TỔNG KẾT CHƯƠNG II

- 1- Các ràng buộc khóa: Chúng ta có thể khai báo một thuộc tính hoặc một tập thuộc tính là một khóa với một mô tả UNIQUE hoặc PRIMARY KEY trong lược đồ quan hệ.
- 2- Các ràng buộc toàn vẹn tham chiếu: Chúng ta có thể khai báo rằng một giá trị xuất hiện trong một thuộc tính hoặc một tập thuộc tính nào đó cũng phải xuất hiện trong các thuộc tính tương ứng của một bộ nào đó của một quan hệ khác với mô tả REFERENCE hoặc FOREIGN KEY trong lược đồ quan hệ
- 3- Các ràng buộc kiểm tra dựa trên thuộc tính: Chúng ta có thể đặt một ràng buộc trên giá trị của một thuộc tính bằng cách thêm từ khóa CHECK và điều kiện sẽ được kiểm tra vào sau mô tả của thuộc tính này trong lược đồ quan hệ của nó.

- 4- Các ràng buộc kiểm tra dựa trên bộ: Chúng ta có thể đặt lên các bộ của một quan hệ bằng cách thêm từ khóa CHECK và điều kiện sẽ được kiểm tra vào mô tả của chính quan hệ đó.
- 5- Sửa đổi các ràng buộc: Có thể thêm và xóa một kiểm tra dựa trên bộ bằng lệnh ALTER đối với bảng thích hợp.
- 6- Assertion: Chúng ta có thể khai báo một assertion như một thành phần của lược đồ cơ sở dữ liệu với từ khóa CHECK và một điều kiện sẽ được kiểm tra. Điều kiện này có thể dính dáng đến một hoặc nhiều quan hệ của lược đồ cơ sở dữ liệu, và có thể dính dáng đến một quan hệ đầy đủ, chẳng hạn với phép nhóm cũng như là các điều kiện về các bộ riêng rẽ.
- 7- Việc gọi các kiểm tra: Các assertion được kiểm tra mỗi khi có một thay đổi đối với một trong các quan hệ dính dáng đến. Các kiểm tra dựa trên bộ và dựa trên thuộc tính chỉ được kiểm tra khi phép chèn hoặc phép cập nhật làm thay đổi thuộc tính hoặc quan hệ mà các kiểm tra áp dụng với chúng. Như vậy, các ràng buộc này có thể bị vi phạm nếu chúng có các truy vấn con
- 8- Trigger: Chuẩn SQL có các trigger chỉ ra các sự kiện nào đây (Chèn, xóa hoặc cập nhật một quan hệ cụ thể). Mỗi lần được đánh thức, một điều kiện có thể được kiểm tra và nếu đúng thì một dãy hành động ghi rõ sẽ được thực hiện. Các hành động là các lệnh SQL như là các truy vấn và cập nhật cơ sở dữ liệu.

## MỘT SỐ BÀI TẬP

I. Giả sử chúng ta có cơ sở dữ liệu sau:

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

Hãy viết mô tả đầy đủ của lược đồ cơ sở dữ liệu trên.

II. Viết các ràng buộc trên đây trên các thuộc tính của các quan hệ trên:

- 1) Tốc độ của laptop ít nhất là 800.
- 2) Đĩa CD chỉ có thể là 32x hoặc 40x hoặc đĩa DVD là 12x hoặc 16x.
- 3) Các máy in chỉ có kiểu là laser, ink-jet hoặc bubble.
- 4) Chỉ có các kiểu sản phẩm là PC, laptop và printer.

III. Viết các điều sau đây như là các trigger hoặc assertion. Trong mỗi trường hợp loại bỏ hoặc không cho phép các cập nhật nếu chúng không thỏa mãn các ràng buộc.

- 1) Khi cập nhật giá của một PC, hãy kiểm tra rằng không có PC nào có cùng tốc độ nhưng có giá rẻ hơn.
- 2) Không có nhà sản xuất PC có thể sản xuất laptop.
- 3) Một nhà sản xuất PC phải sản xuất một laptop với ít nhất một tốc độ bộ xử lý lớn
- 4) Khi chèn vào một printer mới hãy kiểm tra rằng số model tồn tại trong quan hệ Product.
- 5) Khi thực hiện một cập nhật bất kỳ trong quan hệ Laptop, hãy kiểm tra rằng giá trung bình của các laptop đối với mỗi nhà sản xuất ít nhất là \$2000.
- 6) Khi sửa đổi RAM hoặc đĩa cứng của một PC bất kỳ, hãy kiểm tra rằng PC được cập nhật có đĩa cứng nhiều hơn RAM 100 lần.
- 7) Nếu một laptop có bộ nhớ chính rộng hơn một bộ nhớ một PC thì laptop cũng phải có giá cao hơn PC.
- 8) Khi chèn vào một PC, laptop, hoặc printer mới hãy đảm bảo rằng số model không có xuất hiện trước đó trong các quan hệ PC, Laptop, Printer
- 9) Nếu quan hệ Product đề cập đến một model và kiểu của nó thì model này phải xuất hiện trong quan hệ thích hợp với kiểu đó.



## CHƯƠNG III: LẬP TRÌNH

Trong các chương trước, chúng ta đã chỉ ra rằng SQL là một ngôn ngữ truy vấn tương tác mạnh. Kiểu truy cập đến thông tin này rất tiện lợi bởi vì những người sử dụng luôn luôn có yêu cầu truy vấn cơ sở dữ liệu theo một logic không được thiết lập trước. Về nguyên tắc, SQL tương tác rất tiện lợi cho nhiệm vụ đó bởi vì nó đủ để cho người sử dụng xây dựng truy vấn đáp ứng các yêu cầu của anh ta. Trên thực tế, điều đó không thích hợp lắm với những người dùng cuối bởi vì cú pháp của ngôn ngữ phức tạp. Việc viết các điều kiện nối thích hợp và các truy vấn lồng nhau đặt ra rất nhiều vấn đề. Vì vậy, hiển nhiên là người dùng cuối không thích thú việc tự mình xây dựng các truy vấn SQL. Với những người dùng cuối cần có một công cụ cho phép họ tạo ra các truy vấn và soạn thảo các báo cáo dễ dàng hơn, không cần có nhiều hiểu biết về cú pháp SQL và cấu trúc của các bảng. Công cụ đó là chương trình. Có nhiều phương pháp để viết một chương trình ứng dụng dựa trên SQL. Đó là nhúng các câu lệnh SQL vào trong các chương trình được viết trong một ngôn ngữ lập trình thông thường, chẳng hạn như C, hoặc kết hợp SQL với lập trình chung, hoặc xây dựng các thư viện SQL mẫu, v.v. Trong chương này chúng ta sẽ xét đến các phương pháp lập trình dựa trên SQL và những vấn đề liên quan.

### 3.1 SQL TRONG MÔI TRƯỜNG LẬP TRÌNH

Tới thời điểm này, chúng ta đã sử dụng giao diện SQL chung trong các ví dụ. Đó là, chúng ta đã giả sử rằng có một bộ thông dịch SQL, nó chấp nhận và thực hiện các loại truy vấn và các câu lệnh chúng ta đã học. Mặc dù được hầu hết các hệ quản trị cơ sở dữ liệu cung cấp như một tùy chọn, kiểu thao tác này hiếm khi được sử dụng. Trong thực tế, hầu hết các câu lệnh SQL là một phần của một mẫu phần mềm lớn. Một khung cảnh thực tiễn hơn là có một chương trình được viết trong một ngôn ngữ chủ thông thường nào đó chẳng hạn như C, nhưng một số bước trong chương trình này là những câu lệnh SQL. Trong phần này chúng ta sẽ mô tả một cách để làm cho SQL thao tác trong một chương trình thông thường.

Hình 3.1 là một phác họa của một hệ thống lập trình điển hình có chứa những câu lệnh SQL. Ở đó, chúng ta nhìn thấy người lập trình đang viết các chương trình



trong một ngôn ngữ chủ nhưng với một vài câu lệnh SQL “nhúng” đặc biệt không là một phần của ngôn ngữ chủ. Toàn bộ chương trình được gửi tới một bộ tiền xử lý, nó biến đổi những câu lệnh SQL nhúng thành những thứ có ý nghĩa trong ngôn ngữ chủ. Sự biểu diễn của SQL có thể đơn giản như là một lời gọi tới một hàm, hàm này nhận câu lệnh SQL như một tham số xâu ký tự và thực hiện câu lệnh SQL đó.

Chương trình ngôn ngữ chủ được tiền xử lý sau đó được biên dịch theo cách thông thường. Những nhà cung cấp hệ quản trị cơ sở dữ liệu thường chuẩn bị đầy đủ một thư viện cung cấp những định nghĩa hàm cần thiết. Do vậy, những hàm thể hiện SQL có thể được xử lý, và toàn bộ chương trình ứng xử như một đơn vị. Chúng ta cũng đưa vào trong Hình 3.1 khả năng người lập trình viết mã lệnh trực tiếp trong ngôn ngữ chủ, sử dụng những lời gọi hàm đó khi cần. Cách tiếp cận này, thường được tham khảo đến như *một giao diện mức gọi* (call-level interface – CLI), sẽ được thảo luận trong phần 3.4.

### 3.1.1 Vấn đề trở ngại không phù hợp

Vấn đề cơ bản của việc kết nối những câu lệnh SQL với những câu lệnh của một ngôn ngữ lập trình thông thường là trở ngại không phù hợp, trên thực tế là mô hình dữ liệu của SQL khác các mô hình của những ngôn ngữ khác rất nhiều. Như chúng ta biết, SQL sử dụng mô hình dữ liệu quan hệ ở lõi của nó. Tuy nhiên, C và những ngôn ngữ lập trình phổ biến khác sử dụng một mô hình dữ liệu với những số nguyên, số thực, các số, các ký tự, con trỏ, cấu trúc bản ghi, mảng v.v. Tập hợp không có mặt trực tiếp trong C hoặc những ngôn ngữ khác đó, trong khi SQL không sử dụng biến trỏ, vòng lặp và rẽ nhánh, hoặc nhiều cấu trúc ngôn ngữ lập trình phổ biến khác. Kết quả là việc trao đổi dữ liệu truyền dữ liệu giữa SQL và những ngôn ngữ khác là không đơn giản, và phải đưa ra một kỹ thuật cho phép xây dựng những chương trình sử dụng cả SQL và ngôn ngữ khác.

Đầu tiên người ta có thể giả thiết rằng tốt hơn hết là sử dụng một ngôn ngữ đơn; hoặc làm tất cả những tính toán trong SQL hoặc quên SQL đi và làm tất cả những tính toán trong một ngôn ngữ thông thường. Tuy nhiên, chúng ta có thể nhanh chóng bỏ qua ý tưởng bỏ qua SQL khi có những phép toán cơ sở dữ liệu được đưa vào. Các hệ thống SQL hỗ trợ rất nhiều cho người lập trình trong việc viết những phép toán cơ sở dữ liệu có thể được

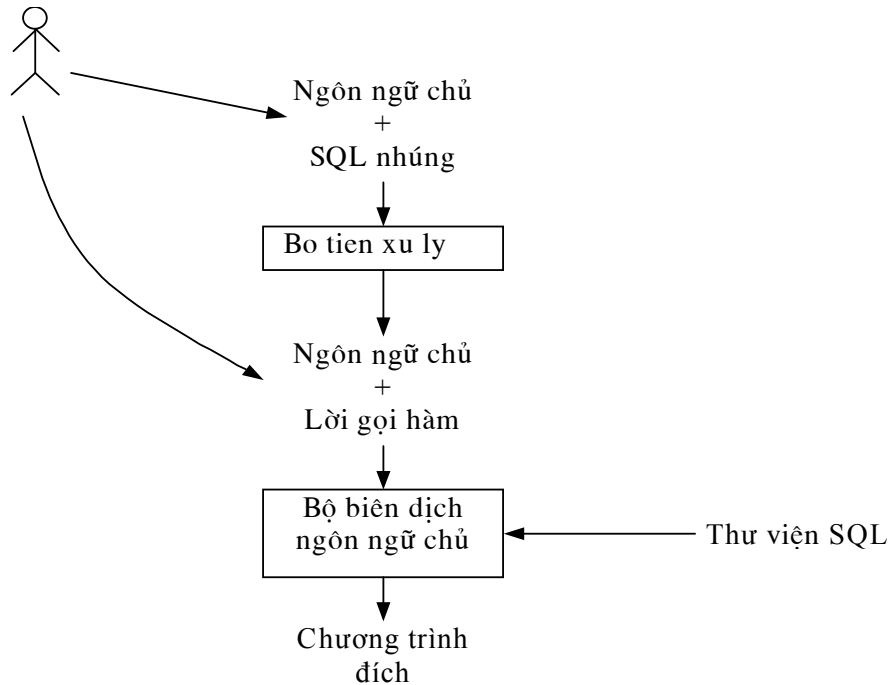
thực hiện một cách hiệu quả tuy rằng chúng có thể được biểu diễn ở mức rất cao. SQL giải phóng người lập trình khỏi sự cần thiết để hiểu dữ liệu được tổ chức như thế nào trong bộ nhớ hoặc khai thác cấu trúc lưu trữ đó như thế nào để thao tác một cách có hiệu quả trên cơ sở dữ liệu.

Mặc khác, có nhiều việc quan trọng mà SQL không thể làm nổi. Ví dụ, người ta không thể viết một truy vấn SQL để tính toán giai thừa của một số  $n$  [ $n! = n * (n-1) * \dots * 2 * 1$ ], dù đó là một bài tập đơn giản trong C hoặc trong những ngôn ngữ tương tự. Một ví dụ khác, SQL không thể định dạng đầu ra của nó một cách trực tiếp thành một dạng thích hợp như là một đồ họa. Do vậy, lập trình cơ sở dữ liệu thực sự đòi hỏi cả SQL và một ngôn ngữ thông thường; ngôn ngữ này thường được tham chiếu đến như là ngôn ngữ chủ (host language).

### 3.1.2 Giao diện ngôn ngữ chủ /SQL

Việc truyền thông tin giữa cơ sở dữ liệu (chỉ có thể được truy cập bởi những câu lệnh SQL) và chương trình ngôn ngữ chủ thông qua những biến của ngôn ngữ chủ, gọi là các biến dùng chung. Những biến này có thể được các câu lệnh SQL đọc hoặc ghi. Tất cả những *biến dùng chung* (*shared variable*) như vậy có một dấu hai chấm đi trước khi chúng được tham chiếu đến trong một câu lệnh SQL, nhưng chúng xuất hiện không có dấu hai chấm trong những câu lệnh ngôn ngữ chủ.

Khi chúng ta muốn sử dụng một câu lệnh SQL trong một chương trình ngôn ngữ chủ, chúng ta phải đặt các từ khóa EXEC SQL ở trước nó. Một hệ thống điển hình sẽ tiên xử lý những câu lệnh đó và thay thế chúng bởi những lời gọi hàm phù hợp trong ngôn ngữ chủ, tạo ra việc sử dụng một thư viện các hàm SQL có liên quan.



Hình 3.1: Việc xử lý các chương trình với các câu lệnh SQL nhúng

Một biến đặc biệt, được gọi là `SQLSTATE` trong SQL chuẩn, phục vụ kết nối chương trình ngôn ngữ chủ với hệ thống thực hiện SQL. Kiểu `SQLSTATE` là một mảng năm ký tự. Mỗi khi một hàm của thư viện SQL được gọi, một mã được đặt vào trong biến `SQLSTATE` chỉ ra những vấn đề bất kỳ được tìm thấy trong suốt quá trình gọi đó. SQL chuẩn định rõ một lượng lớn những mã năm ký tự và ý nghĩa của chúng. Ví dụ, '00000' (năm số không) chỉ ra rằng tình trạng không lỗi xảy ra, và '02000' chỉ ra rằng không tìm thấy bộ giá trị được yêu cầu. Chúng ta sẽ thấy rằng mã '02000' rất quan trọng, vì nó cho phép chúng ta tạo một vòng lặp trong chương trình ngôn ngữ chủ kiểm tra lần lượt các bộ của một quan hệ và phá bỏ vòng lặp sau khi bộ cuối cùng đã được kiểm tra. Chương trình ngôn ngữ chủ có thể đọc được giá trị của `SQLSTATE` và đưa một quyết định dựa trên cơ sở giá trị tìm được đó.

### 3.1.3 Phân khai báo (DECLARE)

Để khai báo những biến dùng chung, chúng ta đặt những khai báo của chúng giữa hai câu lệnh SQL nhúng:

```
EXEC SQL BEGIN DECLARE SECTION;
```

.....  
EXEC SQL END DECLARE SECTION;

Những gì xuất hiện giữa chúng được gọi là phần khai báo. Dạng của những khai báo biến trong phần khai báo là tất cả mọi thứ mà ngôn ngữ chủ yêu cầu. Hơn nữa, nó chỉ có nghĩa đối với khai báo các biến có kiểu mà cả ngôn ngữ chủ và SQL có thể giao tiếp, chẳng hạn như số nguyên, số thực, và xâu ký tự hoặc mảng.

Ví dụ 3.1: Những câu lệnh sau đây có thể xuất hiện trong một hàm C cập nhật quan hệ ĐƠN VỊ:

```
EXEC SQL BEGIN DECLARE SECTION;  
INT Mẫ số INT, CHAR Tên[15];  
char SQLSTATE[6];  
EXEC SQL END DECLARE SECTION;
```

Các dòng lệnh đầu và cuối là quy định để bắt đầu và kết thúc phần khai báo. Ở giữa là một dòng lệnh khai báo hai biến Mẫ số và Tên. Chúng là một số nguyên và một mảng ký tự và, như chúng ta sẽ thấy, chúng có thể được sử dụng để lưu giữ một mã số và tên của một đơn vị được đưa vào trong một bộ và được chèn vào trong quan hệ ĐƠN VỊ. Dòng lệnh thứ ba khai báo SQLSTATE là một mảng sáu ký tự.

#### **3.1.4 Sử dụng các biến dùng chung.**

Một biến dùng chung có thể được sử dụng trong các câu lệnh SQL ở những vị trí chúng ta muốn hoặc được phép đặt một hằng. Nhắc lại rằng những biến dùng chung có một dấu hai chấm đặt trước nó khi sử dụng. Đây là một ví dụ trong đó chúng ta sử dụng các biến của Ví dụ 3.1 như là những thành phần của một bộ được chèn vào trong quan hệ ĐƠN VỊ.

Ví dụ 3.2: Trong Hình 3.2 là một minh họa của một hàm C getDonvi nhắc người dùng cho mẫ số và tên của một đơn vị, đọc trả lời và chèn bộ thích hợp vào ĐƠN VỊ. Dòng (1) đến dòng (4) là những khai báo chúng ta đã được học trong Ví dụ 3.1. Chúng ta bỏ qua các lệnh của C yêu cầu đưa các giá trị vào hai biến Mẫ số và Tên. Sau đó, trong các dòng (5) và (6) là một câu lệnh

SQL nhúng, đó là một câu lệnh INSERT thông thường. Lệnh này có từ khóa EXEC SQL đi trước để chỉ ra rằng nó quả thực là một câu lệnh SQL nhúng chứ không phải là một câu lệnh C sai ngữ pháp. Bộ tiền xử lý được đề nghị trong Hình 3.1 sẽ tìm kiếm EXEC SQL để phát hiện những câu lệnh phải được tiền xử lý.

Những giá trị được các dòng (5) và (6) chèn vào không phải là các hằng rõ ràng. Những giá trị xuất hiện trong dòng (6) là những biến dùng chung mà giá trị hiện thời của chúng trở thành những thành phần của bộ được chèn.

Cùng với câu lệnh INSERT còn có nhiều kiểu câu lệnh SQL khác có thể được nhúng vào trong một ngôn ngữ chủ, chúng sử dụng những biến dùng chung như một giao diện. Mỗi câu lệnh nhúng SQL có các từ khóa EXEC SQL đi trước trong chương trình ngôn ngữ chủ và có thể tham chiếu đến các biến dùng chung trong vị trí của các hằng. Câu lệnh SQL bất kỳ không trả về một kết quả (tức là, không phải là một truy vấn) có thể được nhúng. Những ví dụ về những câu lệnh SQL có thể nhúng được bao gồm những câu lệnh xóa và cập nhật và những câu lệnh để tạo, sửa, hoặc xóa những thành phần lược đồ chẳng hạn như các bảng và các khung nhìn.

```
Void getDONVI() {  
1) EXEC SQL BEGIN DECLARE SECTION;  
2)     INT Mãsố INT, CHAR Tên[15];  
3)     char SQLSTATE[6];  
4) EXEC SQL END DECLARE SECTION;  
   /* Các câu lệnh C yêu cầu nhập dữ liệu cho Mãsố và Tên */  
5) EXEC SQL INSERT INTO ĐƠNVI(MãsốĐV, TênĐV)  
6)     VALUES(:Mã số,:Tên) }  
}
```

Hình 3.2: Minh họa ví dụ 3.2

Tuy nhiên, những truy vấn select-from-where là không nhúng được một cách trực tiếp vào trong một ngôn ngữ chủ, bởi vì “trở ngại không phù hợp”. Các truy vấn đưa ra kết quả là những tập hợp các bộ, trong khi không có một

ngôn ngữ chủ nào hỗ trợ kiểu dữ liệu tập hợp một cách trực tiếp. Do vậy, SQL nhúng phải sử dụng một trong hai kỹ thuật để kết nối kết quả của các truy vấn với một chương trình ngôn ngữ chủ.

1. Một truy vấn đưa ra một bộ đơn có thể có bộ đó được lưu trữ trong các biến dùng chung, một biến cho mỗi thành phần của bộ. Để làm như vậy, chúng ta sử dụng một dạng sửa đổi của câu lệnh select-from-where được gọi là một select đơn-hàng (single-row select).

2. Các truy vấn tạo ra nhiều hơn một bộ có thể được thực hiện nếu chúng ta khai báo một con trỏ cho truy vấn. Con trỏ duyệt qua tất cả các bộ trong quan hệ trả lời, và mỗi bộ có thể lần lượt được đặt vào trong những biến dùng chung và được xử lý bởi chương trình ngôn ngữ chủ.

Chúng ta sẽ xem xét lần lượt từng kỹ thuật một.

### 3.1.5 Các câu lệnh Select đơn hàng

Khuôn mẫu của một select đơn hàng cũng giống như một câu lệnh select-from-where bình thường, ngoại trừ rằng theo sau mệnh đề SELECT là từ khóa INTO và một danh sách các biến dùng chung. Những biến dùng chung có các dấu hai chấm đi trước, như là trường hợp cho tất cả những biến dùng chung trong một câu lệnh SQL. Nếu kết quả của truy vấn là một bộ đơn, các thành phần của bộ này sẽ trở thành các giá trị của những biến dùng chung. Nếu kết quả hoặc là không có bộ hoặc có nhiều hơn một bộ, thì sẽ không có phép gán nào cho các biến dùng chung được thực hiện, và một mã lỗi thích hợp được ghi vào biến SQLSTATE.

Ví dụ 3.3: Chúng ta sẽ viết một hàm C để đọc tên của một đơn vị và in ra lương của người quản lý đơn vị. Một phác họa của hàm này được thể hiện trong Hình 3.3. Nó bắt đầu với một phần khai báo, dòng (1) đến dòng (5), cho những biến chúng ta sẽ cần đến. Tiếp theo là các câu lệnh C yêu cầu nhập vào Tênđơnvị (các lệnh này không được viết rõ).

Dòng (6) đến (9) là câu lệnh select đơn hàng. Nó hoàn toàn tương tự các truy vấn chúng ta đã được biết. Hai điều khác biệt đó là giá trị của biến Tênđơnvị được sử dụng thay cho một hằng xâu trong điều kiện của dòng (9),

và có một mệnh đề INTO ở dòng (7) cho chúng ta biết nơi đặt kết quả của truy vấn. Trong trường hợp này, chúng ta mong đợi một bộ đơn, và các tuple chỉ có một thành phần cho thuộc tính Lương. Giá trị của một thành phần này của một bộ được lưu trữ trong biến chia dùng chung LuongNQL.

```
void printLuong() {  
  
1) EXEC SQL BEGIN DECLARE SECTION;  
2)     char Tendonvi[15];  
3)     int LuongNQL;  
4)     char SQLSTATE[6];  
5) EXEC SQL END DECLARE SECTION;  
  
/* Các lệnh C yêu cầu nhập giá trị cho Tendonvi */  
  
6) EXEC SQL SELECT Lương  
7)     INTO:LuongNQL  
8) FROM NHÂNVIÊN, ĐƠNVI  
9) WHERE NHÂNVIÊN.Mã sốNV = ĐƠNVI.Mã sốNQL  
     AND TênĐV =:Tendonvi;  
  
/* Các lệnh C kiểm tra xem có phải SQLSTATE chứa toàn số 0 hay  
không. Nếu đúng thì in ra giá trị của LuongNQL */  
}
```

Hình 3.3: Một select đơn hàng được nhúng vào một hàm C

### 3.1.6 Con trỏ

Cách linh hoạt nhất để kết nối các truy vấn SQL tới một ngôn ngữ chủ là dùng một con trỏ chạy qua các bộ của một quan hệ. Quan hệ này có thể là một bảng được lưu trữ, hoặc nó có thể là một bảng do một truy vấn sinh ra. Để tạo và sử dụng một con trỏ, chúng ta cần những câu lệnh sau đây:

1. Một khai báo con trỏ. Hình thức đơn giản nhất của một khai báo con trỏ bao gồm:

- (a) Một mở đầu EXEC SQL, giống như tất cả những câu lệnh SQL nhúng.
- (b) Từ khóa DECLARE.
- (c) Tên của con trỏ.
- (d) Các từ khóa CURSOR FOR.
- (e) Một biểu thức như một tên quan hệ hoặc một biểu thức select-from-where mà giá trị của nó là một quan hệ. Con trỏ được khai báo duyệt qua các bộ của quan hệ này; nghĩa là, con trỏ lần lượt tham chiếu tới mỗi bộ của quan hệ này, như là chúng ta “lấy ra” bằng việc sử dụng con trỏ.

Tổng quát, dạng của một khai báo con trỏ là:

EXEC SQL DECLARE <con trỏ> CURSOR FOR <truy vấn>

2. Một câu lệnh EXEC SQL OPEN tiếp theo sau là tên con trỏ. Câu lệnh này khởi tạo con trỏ đến một vị trí nơi nó đã sẵn sàng để lấy ra bộ đầu tiên của quan hệ mà con trỏ sẽ đi qua trên đó.

3. Một hoặc nhiều sử dụng của câu lệnh FETCH. Mục đích của một lệnh FETCH là để nhận bộ tiếp theo của quan hệ mà con trỏ đi qua trên đó. Nếu các bộ đã được vét cạn thì không có bộ nào được trả về, và giá trị của SQLSTATE được thiết lập thành ‘02000’, một mã có nghĩa là “không tìm thấy bộ”. Câu lệnh FETCH bao gồm những thành phần sau đây:

- (a) Các từ khóa EXEC SQL FETCH FROM.
- (b) Tên của con trỏ.
- (c) Từ khóa INTO.
- (d) Một danh sách các biến dùng chung, được phân cách bởi các dấu phẩy. Nếu có một bộ để lấy ra, thì những thành phần của bộ này được đặt trong những biến đó, theo thứ tự.

Như vậy, dạng của một câu lệnh FETCH là:

EXEC SQL FETCH FROM <con trỏ> INTO <danh sách các biến>



4. Câu lệnh EXEC SQL CLOSE sau đó là tên của con trỏ. Câu lệnh này đóng con trỏ, bây giờ nó không còn đi qua trên các bộ của quan hệ nữa. Tuy nhiên nó có thể được khởi tạo lại bởi câu lệnh OPEN khác, trong trường hợp đó, một lần nữa nó lại duyệt qua các bộ của quan hệ này.

Ví dụ 3.4: Giả sử chúng ta mong muốn đếm xem có bao nhiêu nhân viên có lương gồm 1 chữ số, 2 chữ số, 3 chữ số, v.v ... Chúng ta sẽ thiết kế một truy vấn lấy ra trường Lương của tất cả các bộ NHÂNVIÊN đưa vào một biến dùng chung gọi là LuongNV. Một con trỏ được gọi là execCursor sẽ duyệt qua tất cả những bộ có một thành phần đó. Mỗi lần một bộ được lấy ra, chúng ta tính số các chữ số trong biến nguyên LuongNV và gia tăng phần tử tương ứng của một mảng Đếm

Hàm C Hạnglương bắt đầu trong dòng (1) của Hình 3.4. Dòng (2) khai báo một số biến chỉ được hàm C sử dụng, SQL nhưng không sử dụng. Mảng Đếm lưu giữ số lượng các nhân viên ; biến Chũsố đếm số lượng các chữ số trong một giá trị Lương, và i là một chỉ số duyệt qua các phần tử của mảng Đếm.

```
1) void Hạnglương () {
2)   i, Chuso, Dem[15];
3)   EXEC SQL BEGIN DECLARE SECTION;
4)   int LuongNV;
5)   char SQLSTATE[6];
6)   EXEC SQL END DECLARE SECTION;
7)   EXEC SQL DECLARE execCursor CURSOR FOR
8)     SELECT Lương FROM NHÂNVIÊN
9)   EXEC SQL OPEN execCursor;
10)  for (i=0; i<15; i++) Dem[i] = 0;
11)  while (1) {
12)    EXEC SQL FETCH FROM execCursor INTO:LuongNV;
13)    if (NO_MORE_TUPLES) break;
14)    Chuso = 1;
15)    while((LuongNV /= 10) > 0) Chuso++;
16)    if (Chuso <= 14) Dem[Chuso]++;      }
17)  EXEC SQL CLOSE execCursor ;
18)  for (i=0; i<15; i++)
19)    printf("chuso = %d: so nhan vien = %d\n",
```

```
i, Dem[i]);          }
```

Hình 3.4 Đếm số nhân viên có lương gồm 1 chữ số, 2 chữ số, ....

Dòng (3) đến (6) là một phần khai báo SQL ở đó biến dùng chung LuongNV và SQLSTATE thường dùng được khai báo. Dòng (7) và (8) khai báo execCursor là một con trỏ duyệt qua những giá trị được truy vấn trên dòng (8) cung cấp. Truy vấn này đơn giản là yêu cầu những thành phần Lương của tất cả các bộ trong NHÂNVIÊN. Con trỏ này sau đó được mở ở dòng (9). Dòng (10) hoàn thiện việc khởi tạo bằng việc gán 0 cho các phần tử của mảng Dem.

Công việc chính được làm bởi vòng lặp ở dòng (11) đến dòng (16). Ở dòng (12) một bộ được lấy ra đặt vào trong biến dùng chung LuongNV. Khi những bộ do truy vấn của dòng (8) tuple cung cấp chỉ có một thành phần, chúng ta chỉ cần một biến dùng chung mặc dù trong trường hợp tổng quát sẽ có nhiều biến như là số thành phần của những bộ tìm được. Dòng (13) kiểm tra việc lấy ra đã thành công chưa. Ở đây, chúng ta sử dụng một macro NO\_MORE\_TUPLES. Chúng ta có thể giả thiết macro này được định nghĩa bởi:

```
#define NO_MORE_TUPLE !(strcmp(SQLSTATE, "02000"))
```

Nhắc lại rằng "02000" là mã SQLSTATE có nghĩa là không có bộ nào được tìm thấy. Do vậy, dòng (13) kiểm tra xem có phải là tất cả các bộ được truy vấn trả về đã được tìm thấy trước đó và không có bộ "tiếp theo" để lấy. Nếu đúng, chúng ta thoát khỏi vòng lặp và đi tới dòng (17).

Nếu một bộ đã được lấy ra, thì ở dòng (14) chúng ta khởi tạo số lượng các chữ số trong Lương là 1. Dòng (15) là một vòng lặp lặp lại việc chia LuongNV cho 10 và tăng Chuso lên 1. Khi LuongNV đạt tới 0 sau khi chia cho 10, Chuso lưu giữ số lượng các chữ số trong giá trị của LuongNV được lấy ra. Cuối cùng, dòng (16) tăng phần tử tương ứng của mảng Dem lên 1. Chúng ta giả sử rằng số lượng các chữ số không nhiều hơn 14. Tuy nhiên, có thể có một giá trị Lương với 15 chữ số hoặc nhiều hơn, dòng (16) sẽ không tăng bất kỳ phần tử nào của mảng Dem, vì không có hạng thích hợp;

nghĩa là, những Lương rất lớn bị vớt đi và không ảnh hưởng đến những thông tin thống kê.

Dòng (17) bắt đầu kết thúc hàm. Con trỏ được đóng, và các dòng (18) và (19) in các giá trị trong mảng Dem.

### 3.1.7 Cập nhật bằng con trỏ

Khi một con trỏ duyệt qua các bộ của một bảng cơ sở (nghĩa là, một quan hệ được lưu trữ trong cơ sở dữ liệu chứ không phải là một khung nhìn hoặc một quan hệ do một truy vấn tạo ra), thì người ta không chỉ có thể đọc và xử lý giá trị của mỗi bộ, mà còn có thể cập nhật hoặc xóa các bộ. Cú pháp của những câu lệnh UPDATE và DELETE đó giống như chúng ta đã bắt gặp chương 1, ngoại trừ mệnh đề WHERE. Mệnh đề đó có thể chỉ là WHERE CURRENT OF sau đó là tên của con trỏ. Dĩ nhiên chương trình ngôn ngữ chủ có thể đọc bộ để áp dụng một điều kiện nào đó mà nó muốn trước khi quyết định có xóa hoặc cập nhật nó hay không.

Ví dụ 3.5: Trong Hình 3.5 chúng ta thấy một hàm C, hàm này xem xét mỗi bộ của NHÂNVIÊN và quyết định hoặc là xóa bộ nếu lương nhỏ hơn 1000 hoặc là nhân đôi giá trị Lương. Trong các dòng (3) và (4) chúng ta khai báo các biến tương ứng với chín thuộc tính của NHÂNVIÊN, cùng với biến SQLSTATE. Sau đó, ở dòng (6), execCursor được khai báo để tự nó duyệt qua quan hệ được lưu trữ NHÂNVIÊN. Ghi nhớ rằng, trong khi chúng ta có thể cố gắng sửa đổi các bộ thông qua một con trỏ duyệt qua một quan hệ tạm thời là kết quả của một truy vấn nào đó, thì chúng ta chỉ có thể có một ảnh hưởng bền vững trên cơ sở dữ liệu nếu con trỏ duyệt qua một quan hệ được lưu trữ chẳng hạn như NHÂNVIÊN. Dòng (8) đến (14) là vòng lặp, ở đó con trỏ execCursor lần lượt tham chiếu đến từng bộ của NHÂNVIÊN. Dòng (9) lấy ra bộ hiện hành đặt vào trong chín biến được sử dụng cho mục đích này; lưu ý rằng chỉ có Luong1 được sử dụng thực sự. Dòng (10) kiểm tra xem chúng ta đã lấy hết các bộ của NHÂNVIÊN hay chưa. Macro NO\_MORE\_TUPLES được sử dụng lại cho điều kiện biến SQLSTATE có mã “không còn tuple” “02000”.

Trong kiểm tra điều kiện ở dòng (11) chúng ta hỏi lương có dưới 1000 hay không. Nếu đúng, bộ bị xóa do câu lệnh DELETE của dòng (12). Ghi

nhớ rằng mệnh đề WHERE tham chiếu con trỏ, vì thế bộ hiện thời của NHÂNVIÊN, bộ mà chúng ta vừa lấy được, bị xóa khỏi NHÂNVIÊN. Nếu lương ít nhất là 1000 thì dòng (14), Lương trong bộ đó được thay thế bởi giá trị gấp đôi.

```
1) void ThaydoiLuong() {
2) EXEC SQL BEGIN DECLARE SECTION;
3) Char Họđệm1[20], Tên1[15], Mẫ sốNV1[9], date ngaysinh1,
   Char Địachỉ1[30], char Giớitính1, Int Luong1,
4) char Mẫ sốNGS1[9], Int Mẫ sốĐV1, SQLSTATE[6];
5) EXEC SQL END DECLARE SECTION;
6) EXEC SQL DECLARE execCursor CURSOR FOR NHÂNVIÊN;
7) EXEC SQL OPEN execCursor;
8) While (1) {
9) EXEC SQL FETCH FROM execCursor INTO:Họđệm1,:Tên1,
   Mẫ sốNV1,:ngaysinh1,:Địachỉ1,:Giớitính1,:Luong1,:
   Mẫ sốNGS1,:Mẫ sốĐV1 ;
10)     if(NO_MORE_TUPLES) break;
11)     if (Luong1 < 1000)
12)         EXEC SQL DELETE FROM MovieExec
           WHERE CURRENT OF execCursor;
13)     else
14)         EXEC SQL UPDATE MovieExec SET Lương = 2 * Lương
           WHERE CURRENT OF execCursor     }
15) EXEC SQL CLOSE execCursor;          }
```

Hình 3.5: Sửa đổi lương của nhân viên

### 3.1.8 Bảo vệ khỏi sự cập nhật đồng thời

Giả sử rằng khi chúng ta kiểm tra các lương của các nhân viên và sử dụng hàm HangLuong của Hình 3.4, một vài tiến trình khác đang sửa đổi quan hệ NHÂNVIÊN. Chúng ta sẽ nói nhiều hơn về nhiều tiến trình truy cập đồng thời đến một cơ sở dữ liệu khi chúng ta nói về giao tác trong phần 3.6. Tuy

nhiên, lúc này, đơn giản là chúng ta chấp nhận khả năng có những tiến trình khác có thể sửa đổi một quan hệ như chúng ta sử dụng nó.

Chúng ta phải làm gì về khả năng này? Hiển nhiên là chúng ta có thể không muốn cho phép những thay đổi cạnh tranh làm ảnh hưởng đến những bộ chúng ta thấy thông qua con trỏ này. Ta muốn là những thống kê được lấy trên quan hệ như nó tồn tại ở một thời điểm nào đó. Chúng ta không thể điều khiển một cách chính xác những sửa đổi nào đối với quan hệ NHÂNVIÊN xảy ra trước việc tập hợp các thống kê của chúng ta, nhưng chúng ta có thể mong chờ rằng những câu lệnh sửa đổi hoặc xảy ra hoàn toàn trước hoặc hoàn toàn sau khi hàm Hangluong chạy, bất chấp có bao nhiêu nhân viên bị ảnh hưởng bởi một câu lệnh sửa đổi. Để đạt được sự đảm bảo này, chúng ta có thể khai báo con trỏ ‘không nhạy cảm’ (insensitive) cho những thay đổi cạnh tranh.

Ví dụ 3.6: Chúng ta có thể sửa đổi dòng (7) và (8) của Hình.4 thành:

- 7) EXEC SQL DECLARE execCursor INSENSITIVE CURSOR FOR
- 8) SELECT Lương FROM NHÂNVIÊN;

Nếu execCursor được khai báo như vậy, thì đó hệ thống SQL sẽ đảm bảo rằng những thay đổi đối với quan hệ NHÂNVIÊN thực hiện giữa một mở và đóng execCursor sẽ không ảnh hưởng đến tập những bộ được lấy ra.

Một con trỏ insensitive có thể đắt, theo nghĩa là hệ thống SQL có thể sử dụng nhiều thời gian cho việc quản lý truy cập dữ liệu để đảm bảo rằng con trỏ là insensitive. Một lần nữa, một thảo luận về việc quản lý những thao tác đồng thời trên cơ sở dữ liệu được hoãn tới phần 3.6. Tuy nhiên, một cách đơn giản để hỗ trợ một con trỏ insensitive là cho hệ thống SQL trì hoãn lại bất kỳ tiến trình nào có thể truy cập các quan hệ mà các truy vấn của con trỏ insensitive của chúng ta đang sử dụng.

Có những con trỏ nào đó đang duyệt qua một quan hệ R mà chúng ta có thể nói về chúng một cách chắc chắn rằng chúng sẽ không thay đổi R. Một con trỏ như vậy có thể chạy đồng thời với một con trỏ insensitive của R, không có sự mạo hiểm làm thay đổi quan hệ R mà con trỏ insensitive nhìn

thấy. Nếu chúng ta khai báo một con trỏ FOR READ ONLY, thì hệ cơ sở dữ liệu có thể chắc chắn rằng quan hệ nền sẽ không bị sửa đổi bởi việc truy nhập tới quan hệ thông qua con trỏ này.

Ví dụ 3.7: Chúng ta có thể gắn vào sau dòng (8) của Hình 3.4 một dòng FOR READ ONLY;

Nếu vậy, thì bất kỳ nỗ lực thực hiện một sửa đổi thông qua con trỏ execCursor sẽ gây ra một lỗi.

### 3.1.9 Con trỏ cuộn (Scrolling Cursor)

Các con trỏ cho chúng ta một sự lựa chọn về việc chúng ta di chuyển qua các bộ của quan hệ như thế nào. Sự lựa chọn mặc định và phổ biến nhất là bắt đầu từ đầu và lấy ra các bộ theo thứ tự, cho đến khi kết thúc. Tuy nhiên, có những thứ tự khác mà theo đó các bộ có thể được lấy ra và các bộ có thể được quét vài lần trước khi con trỏ bị đóng. Để lợi dụng những thứ tự chọn này, chúng ta cần làm hai việc sau:

1. Khi khai báo một con trỏ, đặt từ khóa SCROLL đằng trước từ khóa CURSOR. Thay đổi này nói với hệ thống SQL rằng con trỏ có thể được sử dụng ở một kiểu khác với việc di chuyển về phía trước theo trật tự của các bộ.

2. Trong một câu lệnh FETCH, theo sau từ khóa FETCH là một trong nhiều tùy chọn cho biết tìm bộ mong muốn ở đâu. Những tùy chọn đó là:

- (a) NEXT hoặc PRIOR để nhận bản ghi tiếp theo hoặc phía trước theo thứ tự. Nhắc lại rằng bản ghi này có liên quan với vị trí hiện thời của con trỏ. NEXT là mặc định nếu không chỉ ra tùy chọn nào, và nó là một lựa chọn thường xuyên.
- (b) FIRST hoặc LAST để nhận bản ghi đầu tiên hoặc bản ghi cuối cùng theo thứ tự.
- (c) RELATIVE theo sau là một số nguyên dương hoặc âm, nó chỉ ra bao nhiêu bản ghi được di chuyển tới (nếu số nguyên là dương) hoặc lui (nếu âm) theo thứ tự. Cho ví dụ, RELATIVE 1 đồng nghĩa với NEXT, và RELATIVE -1 đồng nghĩa với PRIOR.

- (d) ABSOLUTE theo sau là một số nguyên dương hoặc âm, nó chỉ ra vị trí của bản ghi mong muốn đếm từ phía trước (nếu dương) hoặc phía sau (nếu âm). Cho ví dụ, ABSOLUTE 1 đồng nghĩa với FIRST và ABSOLUTE -1 đồng nghĩa với LAST.

Ví dụ 3.8: Chúng ta hãy viết lại hàm trong Hình 3.5 bắt đầu ở bản ghi cuối cùng và di chuyển lùi lại thông qua một danh sách các bản ghi. Đầu tiên, chúng ta cần khai báo con trỏ execCursor là cuộn được, điều này chúng ta làm bằng cách thêm từ khóa SCOLL vào dòng (6), như sau:

```
6) EXEC SQL DECLARE execCursor SCROLL CURSOR FOR
NHÂNVIÊN;
```

Hơn nữa, chúng ta cần khởi tạo việc lấy về những bản ghi với một câu lệnh FETCH LAST, và trong vòng lặp chúng ta sử dụng FETCH PRIOR. Vòng lặp từ dòng (8) đến dòng (14) trong Hình 3.5 được viết lại như trong Hình 3.6. Người đọc sẽ không cho rằng có ưu điểm nào khi đọc những bản ghi theo thứ tự ngược với thứ tự mà chúng được lưu trữ trong NHÂNVIÊN.

```
EXEC SQL FETCH LAST FROM execCursor INTO:Họđệm1,:Tên1,
:
: MãSốNV1,:ngaysinh1,:Địa chỉ1,:Giới tính1,:Luong1,:
MãsôNGS1,:MãsôĐV1 ;
while (1) {
/* giống như các dòng từ (10) đến (14) */
EXEC SQL FETCH PRIOR FROM execCursor INTO:Luong1;
}
```

Hình 8.6: Đọc LÙI các bộ NHÂNVIÊN

### 3.1.10 SQL động

Mô hình SQL nhúng trong một ngôn ngữ chủ của chúng ta là chỉ ra những truy vấn và câu lệnh SQL ở bên trong một chương trình ngôn ngữ chủ. Một kiểu xen kẽ của SQL nhúng có những câu lệnh tự chúng được tính toán bởi ngôn ngữ chủ. Những câu lệnh như vậy không được biết ở thời gian biên dịch, và do vậy không thể được điều khiển bởi một bộ tiền xử lý SQL hoặc một bộ biên dịch ngôn ngữ chủ.

Một ví dụ của một tình trạng như vậy là một chương trình nhắc người sử dụng nhập một truy vấn SQL, đọc truy vấn, và sau đó xử lý truy vấn đó.

Giao diện chung của những truy vấn tức thời (ad-hoc)SQL mà chúng ta đã thừa nhận trong Chương 1 là một ví dụ của một chương trình như vậy; tất cả mọi hệ thống SQL thương mại cung cấp kiểu giao diện SQL chung này. Nếu các truy vấn được đọc và thực hiện ở thời gian chạy, thì không có gì được thực hiện ở thời gian biên dịch. Truy vấn phải được phân tích cú pháp và hệ thống SQL sẽ tìm ra một cách phù hợp để thực hiện truy vấn, ngay lập tức sau khi truy vấn được đọc.

Chương trình ngôn ngữ chủ phải chỉ thị hệ thống SQL lấy những xâu ký tự vừa được đọc, chuyển nó thành câu lệnh SQL thực hiện được, và cuối cùng thực hiện câu lệnh này. Có hai câu lệnh SQL động thực hiện hai bước này.

1. EXEC SQL PREPARE, sau đó là biến V của SQL, từ khóa FROM, và một biến hoặc biểu thức ngôn ngữ chủ kiểu xâu ký tự. Câu lệnh này sinh ra một xâu được xem như một câu lệnh SQL. Có lẽ, câu lệnh SQL được phân tích cú pháp và hệ thống SQL tìm ra một cách tốt để thực hiện nó, nhưng câu lệnh không được thực thi. Đúng hơn là, kế hoạch để thực hiện câu lệnh SQL trở thành giá trị của V.
2. EXEC SQL EXECUTE được theo sau bởi một biến SQL chẳng hạn như V trong (1). Câu lệnh này làm cho câu lệnh SQL được biểu thị bởi V được thực thi.

Hai bước có thể được kết hợp thành một, với câu lệnh:

EXEC SQL EXECUTE IMMEDIATE sau đó là một biến dùng chung có giá trị kiểu xâu hoặc một biểu thức có giá trị kiểu xâu. Nhược điểm của việc kết hợp hai thành phần này được nhìn nhận nếu chúng ta chuẩn bị một lần một câu lệnh và thực thi nó nhiều lần. Với EXECUTE IMMEDIATE giá của việc chuẩn bị câu lệnh phải chịu mỗi lần câu lệnh được thực thi, thay vì chỉ chịu một lần, khi chúng ta chuẩn bị nó.

Ví dụ 3.9: Trong Hình 3.7 là một phác họa của một chương trình C đọc văn bản từ đầu vào chuẩn vào trong một biến query, chuẩn bị nó, và thực thi nó. Biến SQL SQLquery lưu giữ truy vấn đã được chuẩn bị. Vì truy vấn chỉ được thực thi một lần duy nhất, dòng:

```
EXEC SQL EXECUTE IMMEDIATE:query;
```



Có thể thay thế dòng (6) và dòng (7) của Hình 3.7.

- 1) void readQuery() {
- 2) EXEC SQL BEGIN DECLARE SECTION;
- 3) char \*query;
- 4) EXEC SQL END DECLARE SECTION;
- 5) /\* nhắc người dùng nhập truy vấn vào, phân phối chỗ (tức là sử dụng một malloc) và tạo biến dùng chung:query chỉ đến ký tự đầu tiên của truy vấn \*/
- 6) EXEC SQL PREPARE SQLquery FROM:query;
- 7) EXEC SQL EXECUTE SQLquery; }

Hình 3.7: Chuẩn bị và thực hiện một truy vấn SQL động.

## 3.2 CÁC THỦ TỤC ĐƯỢC LƯU GIỮ (stored procedure)

Trong phần này, chúng ta làm quen với một chuẩn SQL gần đây được gọi là Persistent, Stored Modules (SQL/PSM, hoặc đúng hơn là PSM, hoặc PSM-96). Mỗi hệ quản trị cơ sở dữ liệu cung cấp cho người sử dụng một cách lưu trữ cùng với một lược đồ cơ sở dữ liệu một số hàm hoặc thủ tục có thể được sử dụng trong các truy vấn SQL hoặc các câu lệnh SQL khác. Những mẫu chương trình được viết bằng một ngôn ngữ đơn giản, đa năng, và cho phép chúng ta thực hiện những tính toán không thể được biểu diễn trong ngôn ngữ truy vấn SQL ở ngay bên trong cơ sở dữ liệu. Trong tài liệu này chúng ta sẽ mô tả chuẩn SQL/PSM. Chuẩn này cho phép ta viết được các mẫu chương trình theo ý tưởng trên và giúp ta hiểu ngôn ngữ được liên kết với hệ thống cụ thể bất kỳ.

Trong PSM, chúng ta định nghĩa các môđun (modules), đó là các tập hợp của các định nghĩa hàm và thủ tục, các khai báo quan hệ tạm thời, và nhiều khai báo tùy chọn khác. Chúng ta thảo luận sâu hơn về các môđun trong Phần 3.3.7; ở đây chúng ta sẽ chỉ thảo luận các hàm và các thủ tục của PSM.

### 3.2.1 Tạo các hàm và các thủ tục PSM

Các yếu tố chính của một khai báo thủ tục là

CREATE PROCEDURE <tên> (<danh sách tham số>)

các khai báo cục bộ

thân thủ tục;

Dạng này đã được quen biết từ một số ngôn ngữ lập trình; nó bao gồm một tên thủ tục, một danh sách các tham số đặt trong dấu ngoặc đơn, một số khai báo biến cục bộ tùy chọn, và phần thân gồm các lệnh thực hiện được định nghĩa thủ tục. Một hàm được định nghĩa hầu như giống như vậy, ngoại trừ từ khóa FUNCTION được sử dụng và phải chỉ rõ một kiểu giá trị trả lại. Do vậy, các yếu tố của một định nghĩa hàm là:

CREATE FUNCTION <tên> (<danh sách tham số>) RETURN <kiểu>

các khai báo cục bộ

thân hàm;

Các tham số của một thủ tục là bộ ba của chế độ - tên - kiểu, rất giống danh sách tham số của các phương pháp ODL. Như vậy, một tên tham số không chỉ có kiểu khai báo của nó đi sau như thường dùng trong ngôn ngữ lập trình, mà nó còn có một “chế độ” đặt trước, đó là IN, OUT, hoặc INOUT. Ba từ khóa này chỉ ra rằng tham số chỉ là input, chỉ là output hoặc cả input và output một cách tương ứng. IN là mặc định, và có thể được bỏ qua.

Mặt khác, các tham số hàm có thể chỉ là ở chế độ IN. Như vậy, PSM ngăn cấm hiệu ứng phụ bên trong các hàm vì vậy cách duy nhất để lấy thông tin từ một hàm là thông qua giá trị trả về của nó. Chúng ta sẽ không chỉ ra chế độ IN cho các tham số hàm, mặc dù chúng ta sẽ làm điều đó trong các định nghĩa thủ tục.

Ví dụ 3.10: Trong khi chúng ta chưa được học những câu lệnh khác nhau có thể xuất hiện trong các thân hàm và thủ tục, ta xét một loại lệnh quen biết: đó là một câu lệnh SQL. Hạn chế trên những câu lệnh này giống như đối với SQL nhưng: chỉ những câu lệnh select đơn hàng và những truy cập dựa vào con trỏ mới được cho phép như là các truy vấn. Trong Hình 3.8 là một thủ tục PSM, thủ tục này lấy hai địa chỉ - một địa chỉ cũ và một địa chỉ mới - và đổi thuộc tính Địa chỉ của tất cả nhân viên từ địa chỉ cũ sang địa chỉ mới.

- 1) CREATE PROCEDURE Thaydoi(  
2) IN Diachicu VARCHAR[30],  
3) IN Diachimoi VARCHAR[30])  
4) UPDATE NHÂNVIÊN  
5) SET Địa chỉ = Diachimoi  
6) WHERE Địa chỉ = Diachicu

Hình 3.8: Một thủ tục thay đổi địa chỉ

Dòng (1) giới thiệu thủ tục và tên của nó: Thaydoi. Dòng (2) và (3) chứa hai tham số, cả hai đều là những tham số đầu vào, kiểu của chúng là những xâu ký tự độ dài biến đổi có độ dài 30. Ghi nhớ rằng kiểu này phù hợp với kiểu chúng ta khai báo cho thuộc tính Địa chỉ của NHÂNVIÊN trong chương 1. Dòng (4) đến (6) là những câu lệnh UPDATE thông thường. Tuy nhiên, ghi nhớ rằng các tên của tham số có thể được sử dụng như nếu nó là các hằng. Không giống các biến ngôn ngữ chủ, các biến này đòi hỏi có một dấu hai chấm đặt trước chúng khi được sử dụng trong SQL (xem phần 3.1.2), các tham số và các biến địa phương khác của các thủ tục và hàm PSM đòi hỏi không có dấu hai chấm.

### 3.2.2 Một vài dạng câu lệnh đơn giản trong PSM

1. **Câu lệnh gọi:** Dạng của một lời gọi thủ tục là:

CALL <tên thủ tục> (<danh sách tham số>);

Tức là, từ khóa CALL theo sau là tên của thủ tục và danh sách các tham số được đặt trong dấu ngoặc đơn, giống như trong hầu hết các ngôn ngữ. Tuy nhiên, lời gọi này có thể được làm từ nhiều vị trí:

i. Từ một chương trình ngôn ngữ chủ. Ví dụ, nó có thể xuất hiện như sau:

EXEC SQL CALL Foo(:x, 3);

ii. Như một câu lệnh của hàm hoặc thủ tục PSM khác

iii. Như một câu lệnh SQL được đưa ra cho giao diện SQL chung. Ví dụ, chúng ta có thể đưa ra một câu lệnh chẳng hạn như

```
CALL Foo(1, 3);
```

ở một giao diện như vậy, và có thủ tục được lưu giữ Foo được thực hiện với hai tham số của nó được thiết lập là 1 và 3 tương ứng.

Ghi nhớ rằng không được phép gọi một hàm. Ta yêu cầu một hàm trong PSM như chúng ta làm trong C: sử dụng tên hàm và các tham số phù hợp như là một phần của một biểu thức.

2. **Câu lệnh trả về:** Dạng của nó là

```
RETURN <biểu thức>;
```

Câu lệnh này chỉ có thể xuất hiện trong một hàm. Nó tính giá trị biểu thức và đặt giá trị trả về của hàm bằng với kết quả đó. Tuy nhiên, khác với các ngôn ngữ lập trình thông thường, câu lệnh trả về của PSM không kết thúc hàm. Đúng hơn là điều khiển tiếp tục với câu lệnh theo sau, và có thể giá trị trả về sẽ được thay đổi trước khi hàm hoàn thành.

3. **Khai báo các biến địa phương:** Dạng câu lệnh

```
DECLARE <tên> <kiểu>;
```

khai báo một biến với một tên cho trước có kiểu cho trước. Biến này là địa phương, và giá trị của nó không được hệ quản trị cơ sở dữ liệu duy trì sau một quá trình chạy của hàm hay thủ tục. Các khai báo phải đặt trước các câu lệnh thực hiện được ở trong thân hàm hoặc thủ tục.

4. **Các câu lệnh gán:**

Dạng của một lệnh gán là:

```
SET <biến> = <biểu thức>;
```

Trừ từ khóa mở đầu SET, lệnh gán trong PSM khá giống với lệnh gán trong các ngôn ngữ khác. Biểu thức bên phải của dấu bằng được tính giá trị, và giá trị của nó trở thành giá trị của biến ở bên trái. NULL là một biểu thức được phép. Thậm chí biểu thức có thể là một truy vấn khi nó trả về một giá trị đơn.

### 5. Nhóm câu lệnh:

Chúng ta có thể tạo lập một danh sách những câu lệnh kết thúc bởi dấu chấm phẩy và được bao quanh bởi từ khóa BEGIN và END. Cấu trúc này được xem như một câu lệnh đơn và có thể xuất hiện ở bất cứ chỗ nào mà một câu lệnh đơn có thể xuất hiện. Đặc biệt, vì một thân thủ tục hoặc hàm được mong muốn là một câu lệnh đơn, chúng ta có thể đặt một dãy bất kỳ của các câu lệnh trong thân bằng cách bao quanh chúng bởi BEGIN...END.

### 6. Nhãn câu lệnh:

Chúng ta sẽ thấy trong phần 3.2.5 một lý do tại sao các câu lệnh nào đó cần một nhãn. Chúng ta gắn nhãn một câu lệnh bằng việc đặt trước nó một tên (nhãn) và một dấu hai chấm.

### 3.2.3 Các câu lệnh rẽ nhánh.

Với kiểu câu lệnh phức tạp đầu tiên của PSM, chúng ta xét câu lệnh IF. Dạng lệnh này chỉ có chút ít khác lạ. Nó khác với C hoặc các ngôn ngữ tương tự ở chỗ:

1. Câu lệnh kết thúc với các từ khóa END IF.
2. Câu lệnh *if* lồng trong mệnh đề *else* được mở đầu bằng từ khóa đơn ELSEIF.

Vì vậy, dạng chung của một câu lệnh if giống như đề xuất trong hình 3.9. Điều kiện là một biểu thức có giá trị logic bất kỳ, như có thể xuất hiện trong mệnh đề WHERE của câu lệnh SQL. Mỗi danh sách câu lệnh bao gồm những câu lệnh kết thúc bởi dấu chấm phẩy, nhưng không cần được bao quanh bởi BEGIN...END. ELSE cuối cùng và những câu lệnh của nó là tùy chọn. Ví dụ IF...THEN...END IF một mình hoặc cùng với ELSEIF đều có thể chấp nhận.

```
IF <điều kiện> THEN  
<danh sách câu lệnh>  
ELSEIF <điều kiện > THEN  
<danh sách câu lệnh>  
ELSEIF
```

...

```
ELSE <danh sách câu lệnh>  
END IF;
```

Hình 3.9: Dạng của một câu lệnh if

Ví dụ 3.11: Giả sử chúng ta có một lược đồ quan hệ

PHIM (Tênphim, Nămsảnxuất, Độdài, Màu, Tênxưởngphim, Tênđạodiễn)  
Trong đó thuộc tính Màu có kiểu Logic, có giá trị TRUE nếu là phim màu, FALSE nếu là phim đen trắng.

Chúng ta hãy viết một hàm để lấy năm y và một xưởng phim s, và trả về một giá trị boolean là TRUE khi và chỉ khi xưởng phim s sản xuất ít nhất một bộ phim đen trắng trong năm y hoặc không sản xuất bất cứ bộ phim nào ở tất cả mọi thời điểm trong năm đó. Chương trình được trình bày ở hình 3.10

- 1) CREATE FUNCTION Ví dụ(y INT, s CHAR[15]) RETURNS  
BOOLEAN
- 2) IF NOT EXISTS(  
3) SELECT \* FROM PHIM WHERE Nămsảnxuất = y AND  
Tênxưởngphim = s)  
4) THEN RETURN TRUE;  
5) ELSEIF 1<=  
6) (SELECT COUNT(\*) FROM PHIM WHERE Nămsảnxuất = y  
AND Tênxưởngphim = s AND NOT Màu)  
7) THEN RETURN TRUE;  
8) ELSE RETURN FALSE;  
9) END IF;

Hình 3.10: Ví dụ về câu lệnh IF

Dòng (1) giới thiệu hàm bao gồm các tham số của nó. Chúng ta không cần xác định một chế độ cho các tham số, vì nó chỉ có thể là IN cho một hàm. Dòng (2) và (3) kiểm tra trường hợp khi không có bất cứ bộ phim nào của xưởng phim s trong năm y, trong trường hợp đó chúng ta thiết lập giá trị trả về TRUE ở dòng (4). Ghi nhớ rằng dòng (4) không là nguyên nhân để hàm

trả về. Về kỹ thuật, một dòng điều khiển tuân theo các câu lệnh if sinh ra điều khiển nhảy từ dòng (4) tới dòng (9), ở đó hàm hoàn thành và trả về.

Nếu xướng phim s có làm phim trong năm y, thì dòng (5) và (6) kiểm tra xem có phải ít nhất một trong số chúng không phải phim màu. Nếu như vậy, giá trị trả về lại được đặt là TRUE, lần này ở dòng (7). Trong trường hợp ngược lại, xướng phim s có làm các bộ phim nhưng chỉ toàn là phim màu, do vậy chúng ta thiết lập giá trị trả về là FALSE ở dòng (8).

### 3.2.4 Các truy vấn trong PSM

Có nhiều cách sử dụng các truy vấn select-from-where trong PSM.

Các truy vấn con có thể được sử dụng trong các điều kiện, hoặc tổng quát, bất cứ vị trí nào một truy vấn con được phép trong SQL. Để minh họa, chúng ta xem hai ví dụ về những truy vấn con trong dòng (3) và (6) của hình 3.10.

1. Các truy vấn trả về một giá trị đơn có thể được sử dụng ở bên phải của các câu lệnh gán.

2. Một câu lệnh select đơn hàng là một câu lệnh hợp lệ trong PSM. Nhắc lại là câu lệnh này có một mệnh đề INTO chỉ định những biến vào nơi đặt những thành phần của bản ghi được trả về đơn. Những biến này có thể là những biến địa phương hoặc các tham số của một thủ tục PSM. Dạng tổng quát được thảo luận trong ngữ cảnh của SQL nhúng trong Phần 3.1.5.

3. Chúng ta có thể khai báo và sử dụng một con trỏ, về cơ bản nó được mô tả trong Phần 8.1.6 của SQL nhúng. Việc khai báo con trỏ, tất các câu lệnh OPEN, FETCH, và CLOSE như được mô tả ở đó, với các ngoại lệ sau:

- (a) Không có EXEC SQL xuất hiện trong các câu lệnh, và
- (b) Các biến (là cục bộ) không sử dụng tiền tố hai chấm

```
CREATE PROCEDURE SomeProc(IN Tendonvi CHAR[15])
DECLARE LuongNQL INTEGER;
SELECT Luong
INTO LuongNQL
FROM NHÂNVIÊN, ĐƠNVI
WHERE NHÂNVIÊN.Mã số NV = ĐƠNVI.Mã số NQL
```

```
AND TênĐV = Tendonvi;    ...  
Hình 3.11: Một select đơn hàng trong PSM
```

Ví dụ 3.12: Trong Hình 3.11 là một select đơn hàng của Hình 3.3, làm lại bằng PSM và đặt trong ngữ cảnh một định nghĩa thủ tục giả thuyết. Ghi nhớ rằng, bởi vì select đơn hàng trả về một bản ghi có một thành phần, chúng ta cũng có thể nhận ảnh hưởng từ một câu lệnh chỉ định giống như:

```
SET LuongNQL = (SELECT Luong  
FROM NHÂNVIÊN,ĐƠNVI  
WHERE NHÂNVIÊN.Mã sốNV = ĐƠNVI.Mã sốNQL  
AND TênĐV = Tendonvi;
```

Chúng ta sẽ hoãn những ví dụ của con trỏ cho đến khi chúng ta học các câu lệnh lặp PSM trong phần tiếp theo.

### 3.2.5 Vòng lặp trong PSM

Cấu trúc vòng lặp cơ bản trong PSM là:

```
LOOP  
  < danh sách tham số >  
END LOOP;
```

Người ta thường gắn nhãn cho câu lệnh LOOP, vì vậy nó có thể phá vỡ vòng lặp, bằng việc sử dụng câu lệnh:

```
LEAVE < nhãn vòng lặp >;
```

Trong trường hợp phổ biến loop tham gia lấy về những bản ghi bằng một con trỏ, chúng ta thường mong muốn rời bỏ vòng lặp khi không còn bản ghi. Sẽ hữu ích nếu khai báo một tên trạng thái của giá trị SQLSTATE, nó chỉ ra rằng không tìm thấy bản ghi nào (tức là khi SQLSTATE = '02000'); chúng ta làm vậy với:

```
DECLARE Not_Found CONDITION FOR SQLSTATE '02000';
```

Tổng quát hơn, chúng ta có thể khai báo một trạng thái với bất kỳ một tên mong muốn tương ứng với giá trị SQLSTATE bằng

```
DECLARE < tên > CONDITION FOR SQLSTATE < giá trị >;
```



Bây giờ chúng ta sẵn sàng dẫn ra một ví dụ trói buộc những thao tác con trỏ và vòng lặp trong PSM.

Ví dụ 3.13: Hình 3.12 thể hiện một thủ tục PSM nó nhận một tên xưởng quay  $s$  như một tham số đầu vào và đưa ra tham số đầu ra mean và variance giá trị trung bình và sự biến thiên về độ dài của tất cả những bộ phim của xưởng  $s$ . Dòng (1) đến dòng (4) khai báo thủ tục và các tham số của nó.

Dòng (5) đến dòng (8) là các khai báo cục bộ. Chúng ta định nghĩa `Not_Found` là tên của trạng thái có nghĩa là một FETCH thất bại để trả về một bản ghi ở dòng (5). Sau đó, ở dòng (6), con trỏ `MovieCursor` được định nghĩa để trả về tập hợp những độ dài của những bộ phim của xưởng quay  $s$ . Dòng (7) và (8) khai báo hai biến cục bộ mà chúng ta sẽ cần. Số nguyên `newLength` lưu giữ kết quả của một FETCH, trong khi `movieCount` đếm số lượng những bộ phim của xưởng quay  $s$ . Chúng ta cần `movieCount` bởi vì, cuối cùng, chúng ta có thể chuyển một tổng các độ dài vào trong một giá trị trung bình của các độ dài và một tổng của các bình phương các độ dài vào trong một sự biến thiên.

Phần còn lại của các dòng là thân của thủ tục. Chúng ta sẽ sử dụng mean và variance là các biến tạm thời, cũng như khi để “trả về” những kết quả cuối cùng. Trong vòng lặp chính, mean thực sự nắm giữ tổng của các độ dài, và variance thực sự nắm giữ tổng các bình phương các độ dài. Do vậy, các dòng (9) đến (11) khởi tạo các biến này và tổng số các bộ phim là 0. Dòng (12) mở con trỏ, và dòng (13) đến dòng (19) thiết lập vòng lặp được gán nhãn `movieLoop`.

Dòng (14) biểu diễn một fetch, và ở dòng (15) chúng ta kiểm tra xem những bản ghi khác có được tìm thấy không. Nếu không, chúng ta rời khỏi vòng lặp. Dòng (15) đến (18) tích lũy các giá trị; chúng ta thêm 1 vào `movieCount`, thêm độ dài tới mean (nhắc lại, nó tính toán tổng các độ dài), và chúng ta thêm bình phương của độ dài tới variance.

Khi tất cả những bộ phim của xưởng quay  $s$  được tìm thấy, chúng ta rời khỏi vòng lặp, và điều khiển chuyển đến dòng (20). Ở dòng này, chúng ta trả chuyển mean thành giá trị chính xác của nó bằng cách chia tổng của các độ dài cho tổng số các bộ phim. Ở dòng (21), chúng ta tạo variance thực sự lưu

giữ sự biến thiên bằng cách chia tổng bình phương của độ dài cho tổng số các bộ phim và trừ cho bình phương của trung bình. Xem Hình 8.2.4 để thảo luận tại sao tính toán này chính xác. Dòng (22) đóng con trỏ, và chúng ta hoàn thành.

```
1) CREATE PROCEDURE MeanVar(  
2)     IN s CHAR[[15]],  
3)     OUT mean REAL,  
4)     OUT variance REAL )  
5) DECLARE Not_Found CONDITION FOR SQLSTATE '02000';  
6) DECLARE MovieCursor CURSOR FOR  
       SELECT Độdài FROM PHIM WHERE Tênxưởngphim = s;  
7) DECLARE newLength INTEGER;  
8) DECLARE movieCount INTEGER;  
   BEGIN  
9)   SET mean = 0.0;  
10)  SET variance = 0.0;  
11)  SET movieCount = 0;  
12)  OPEN MovieCursor;  
13)  MovieLoop: LOOP  
14)      FETCH MovieCursor INTO newLength;  
15)      IF Not_Found THEN LEAVE MovieLoop END IF;  
16)      SET movieCount = movieCount + 1;  
17)      SET mean = mean + newLength;  
18)      SET variance = variance + newLength * newLength;  
19)  END LOOP;  
20)  SET mean = mean/movieCount;  
21)  SET variance = variance/movieCount - mean * mean;  
22)  CLOSE MovieCursor;  
   END;
```

Hình 3.12: Tính giá trị trung bình và biến thiên của độ dài các cuốn phim do một xưởng phim sản xuất

### 3.2.6 Vòng lặp for

Trong PSM cũng có một cấu trúc for, nhưng nó cho một mục đích quan trọng duy nhất: để lặp lại thông qua một con trỏ. Dạng của câu lệnh là:

```
FOR <tên vòng lặp> AS <tên con trỏ> CURSOR FOR  
    <truy vấn>  
DO  
    <danh sách câu lệnh>  
END FOR;
```

Câu lệnh này không chỉ khai báo một con trỏ mà nó còn vận hành cho chúng ta một số lượng “các chi tiết khó chịu”: việc mở và đóng con trỏ, việc lấy về và kiểm tra xem có phải không có bản ghi nào được lấy không. Tuy nhiên, vì chúng ta không tự lấy các bản ghi, chúng ta không thể chỉ ra các biến được đặt cho những thành phần nào của bản ghi. Do vậy, các tên được sử dụng cho các thuộc tính trong kết quả của truy vấn cũng được PSM coi như các biến cục bộ có cùng kiểu.

- 1) CREATE PROCEDURE MeanVar(  
2) IN s CHAR[[15]],  
3) OUT mean REAL,  
4) OUT variance REAL )  
5) DECLARE movieCount INTEGER;  
 BEGIN  
6) SET mean = 0.0;  
7) SET variance = 0.0;  
8) SET movieCount = 0;  
9) FOR movieLoop AS MovieCursor CURSOR FOR  
 SELECT Độdài FROM PHIM WHERE studioName = s;  
10) DO  
11) SET movieCount = movieCount + 1;  
12) SET mean = mean + length ;  
13) SET variance = variance + length \* length;  
14) END FOR;  
15) SET mean = mean/movieCount;

```
16) SET variance = variance/movieCount – mean * mean;  
END;
```

Hình 3.13: Tính giá trị trung bình và biến thiên của độ dài các cuốn phim sử dụng vòng lặp for

### 3.2.7 Những câu lệnh lặp khác

PSM cũng cho phép các vòng lặp while và repeat, chúng có ý nghĩa như trong C. Như vậy chúng ta có thể viết một vòng lặp dưới dạng:

```
WHILE <Điều kiện> DO  
    <Danh sách các lệnh>  
END WHILE;
```

Hoặc một vòng lặp dưới dạng

```
REPEAT  
    <danh sách các lệnh>  
UNTIL < Điều kiện >  
END REPEAT;
```

Tiện đây, nếu chúng ta gán nhãn cho các vòng lặp này, hoặc vòng lặp được một lệnh lặp hoặc một lệnh for tạo ra, thì chúng ta có thể đặt nhãn sau END LOOP hoặc sau kết thúc khác cũng được. Ưu điểm của việc làm như vậy là ở chỗ nó làm rõ kết thúc của mỗi vòng lặp ở đâu và nó cho phép bộ thông dịch PSM bắt được một số lỗi ngữ pháp bao hàm việc bỏ qua một END.

Ví dụ 3.14: Chúng ta hãy làm lại thủ tục của Hình 3.12 bằng việc sử dụng một vòng lặp for. Mã lệnh được thể hiện trong Hình 3.13. Nhiều thứ không thay đổi. Phần khai báo của thủ tục từ dòng (1) đến dòng (4) của Hình 3.13 thì tương tự, cũng như việc khai báo biến cục bộ movieCount ở dòng (5).

Tuy nhiên, chúng ta không cần khai báo một con trỏ trong phần khai báo của thủ tục nữa, và chúng ta không cần định nghĩa điều kiện Not\_Found. Dòng (6) đến dòng (8) khởi tạo các biến, như trước đây. Sau đó, ở dòng (9) chúng ta xem vòng lặp for, nó cũng định nghĩa con trỏ MovieCursor. Các dòng (11) đến (13) là thân của vòng lặp. Ghi nhớ rằng trong các dòng (12)

và (13), chúng ta tham khảo độ dài lấy về bởi con trỏ bằng thuộc tính tên là length, thay vì bằng biến cục bộ newLength, biến này không tồn tại trong phiên bản này của thủ tục. Các dòng (15) và (16) tính toán các giá trị chính xác cho các biến đầu ra, đúng như trong phiên bản trước đây của thủ tục này.

### 3.2.8 Những loại trừ trong PSM

Một hệ thống PSM chỉ ra các điều kiện lỗi bằng cách thiết lập một chuỗi khác không các con số trong một xâu năm ký tự SQLSTATE. Chúng ta đã xem một ví dụ về những mã này: '02000' của “không bản ghi nào được tìm thấy”. Ví dụ khác, '21000' chỉ ra rằng một select đơn hàng đã trả về nhiều hơn một hàng.

PSM cho phép chúng ta khai báo một phần mã lệnh, được gọi là bộ điều khiển loại trừ, nó được gọi mỗi khi một lỗi trong danh sách những mã lỗi này xuất hiện trong SQLSTATE trong khi thực thi một câu lệnh hoặc một danh sách các câu lệnh. Mỗi bộ điều khiển loại trừ được liên kết với một khối mã lệnh, được mô tả bởi BEGIN...END. Bộ điều khiển xuất hiện trong khối này, và nó chỉ áp dụng cho những câu lệnh trong khối.

Các thành phần của bộ điều khiển là:

Một danh sách các điều kiện loại trừ gọi đến bộ điều khiển khi được kích hoạt.

Mã lệnh được thi hành khi một trong những biểu thức liên kết được kích hoạt.

Một chỉ thị vị trí đến sau khi bộ điều khiển đã kết thúc công việc của nó.

Cách thức khai báo của một bộ điều khiển là:

```
DECLARE <vị trí đến> HANDLER FOR <danh sách điều kiện>  
      <câu lệnh>
```

Các lựa chọn của “vị trí đến” là:

CONTINUE, nó có nghĩa là sau khi thực hiện câu lệnh trong phần khai báo điều khiển, chúng ta thực hiện câu lệnh sau khi sau câu lệnh phát sinh loại trừ.

EXIT, có nghĩa rằng sau sự thực thi câu lệnh của bộ điều khiển, điều khiển rời khỏi khối BEGIN...END ở chỗ bộ điều khiển được khai báo. Câu lệnh sau khối này sẽ được thực hiện tiếp.

UNDO, nó giống như là EXIT, ngoại trừ rằng bất cứ thay đổi nào tới cơ sở dữ liệu hoặc các biến cục bộ được làm bởi các câu lệnh của khối cho đến bây giờ bị “tháo bỏ”. Tức là, những ảnh hưởng của nó bị xóa bỏ, và cũng như là những câu lệnh này chưa hề được thực hiện.

“Danh sách điều kiện” là một danh sách những điều kiện được phân cách bởi dấu phẩy, chúng hoặc là những điều kiện được khai báo, giống như Not\_Found trong dòng (5) của Hình 8.12, hoặc những biểu thức của dạng SQLSTATE và một xâu năm ký tự.

Ví dụ 3.15: Chúng ta hãy viết một hàm PSM, hàm này lấy một tiêu đề bộ phim làm tham số và trả về năm của bộ phim. Nếu không có bộ phim nào có tiêu đề đó hoặc có nhiều hơn một bộ phim có tiêu đề đó, thì NULL phải được trả về. Mã lệnh được thể hiện trong Hình 3.14.

```
CREATE FUNCTION GetYear (t VARCHAR[255]) RETURN INTEGER
DECLARE Not_Found CONDITION FOR SQLSTATE '02000' ;
DECLARE Too_Many CONDITION FOR SQLSTATE '21000' ;
BEGIN
DECLARE EXIT HANDLER FOR Not_Found, Too_Many
RETURN NULL ;
RETURN (SELECT Năm FROM PHIM WHERE Tênphim = t);
END;
```

Hình 3.14 Điều khiển ngoại trừ trong đó một SELECT đơn hàng trả về những cái khác với một bộ

Dòng (2) và (3) khai báo các điều kiện tiêu biểu; chúng ta không phải tạo những định nghĩa này, và cũng có thể sử dụng các trạng thái SQL với việc chúng đặt trong dòng (4). Dòng (4), (5) và (6) là một khối, ở đó đầu tiên chúng ta khai báo một bộ điều khiển cho hai điều kiện ở đó hoặc là không có

bản ghi nào được trả về, hoặc có nhiều hơn một bản ghi được trả về. Hành động của bộ điều khiển, trên dòng (5), đơn giản là thiết lập giá trị trả về là NULL.

Dòng (6) là câu lệnh làm công việc của hàm GetYear. Nó là một câu lệnh SELECT mong muốn trả về chính xác một số nguyên, vì đó là hàm giá trị hàm GetYear trả về. Nếu chính xác có một bộ phim có tiêu đề t (tham số đầu vào của hàm), thì giá trị này sẽ được trả về. Tuy nhiên, nếu một loại trừ phát sinh ở dòng (6), hoặc là bởi vì không có bộ phim nào có tiêu đề t hoặc vài bộ phim có tiêu đề đó, thì bộ điều khiển được gọi, một NULL thay thế, bởi trở thành giá trị trả về. Hơn nữa, vì bộ điều khiển là một bộ điều khiển EXIT, điều khiển chuyển tiếp tới con trỏ sau END. Vì con trỏ đó là cuối của hàm, GetYear trả về ở thời điểm đó, với giá trị trả về NULL.

### 3.2.9 Sử dụng các hàm và các thủ tục PSM

Như chúng ta đề cập trong Phần 3.2.2, chúng ta có thể gọi một hàm hoặc thủ tục PSM từ một chương trình với SQL nhúng, từ chính mã lệnh PSM, hoặc từ các câu lệnh SQL nguyên thủy xuất phát từ giao diện chung. Việc sử dụng những thủ tục và hàm này thì giống như trong các ngôn ngữ lập trình hơn cả, với các thủ tục được gọi bởi CALL, và các hàm xuất hiện như thành phần của một biểu thức. Chúng ta sẽ đưa ra một ví dụ tại sao một hàm có thể bị hủy bỏ từ giao diện chung.

Ví dụ 3.16: Giả sử rằng lược đồ của chúng ta bao gồm một mô đun với hàm GetYear của Hình 3.14. Tưởng tượng rằng chúng ta đang ở giao diện chung, và chúng ta muốn nhập vào sự thật là Denzel Washington là một ngôi sao của Remember the Titans. Tuy nhiên, chúng ta quên năm mà bộ phim đó được làm. Bởi vì chỉ có duy nhất một bộ phim có tên như vậy, và nó ở trong quan hệ Movie, chúng ta không phải tìm kiếm trong một truy vấn mở đầu. Hơn nữa, chúng ta có thể đưa tới giao diện SQL chung lệnh chèn sau đây:

```
INSERT INTO StarsIn(Tênphim, Năm, Têndiễnviên)
VALUES('Remember the Titans', GetYear('Remember the Titans'),
'Dezel Washington');
```

Vì GetYear trả về giá trị NULL nếu không có một bộ phim duy nhất có tên Remember the Titans, có thể là việc chèn này sẽ cho giá trị NULL trong thành phần ở giữa.

### 3.3 MÔI TRƯỜNG SQL

Trong phần này chúng ta sẽ trình bày một khung nhìn rộng rãi nhất có thể về một DBMS và các cơ sở dữ liệu và các chương trình mà nó hỗ trợ. Chúng ta sẽ xem các cơ sở dữ liệu được định nghĩa và tổ chức vào trong các cluster, các danh mục, và các lược đồ như thế nào. Chúng ta cũng sẽ xem các chương trình được liên kết với dữ liệu chúng cần thao tác như thế nào. Nhiều chi tiết phụ thuộc vào sự cài đặt cụ thể, vì vậy chúng ta sẽ tập trung vào những ý tưởng chung có ở trong chuẩn SQL.

#### 3.3.1 Môi trường

Một môi trường SQL là một khung cảnh mà bên dưới nó dữ liệu có thể tồn tại và các thao tác SQL trên dữ liệu có thể được thực hiện. Trong thực tiễn, chúng ta nên nghĩ về một môi trường SQL như là một hệ quản trị cơ sở dữ liệu đang chạy ở một cài đặt nào đó. Ví dụ, công ty ABC mua một bản quyền của DBMS Megatron 2002 để chạy trên một tập hợp các máy của ABC. Hệ thống đang chạy trên những máy này tạo thành một môi trường SQL.

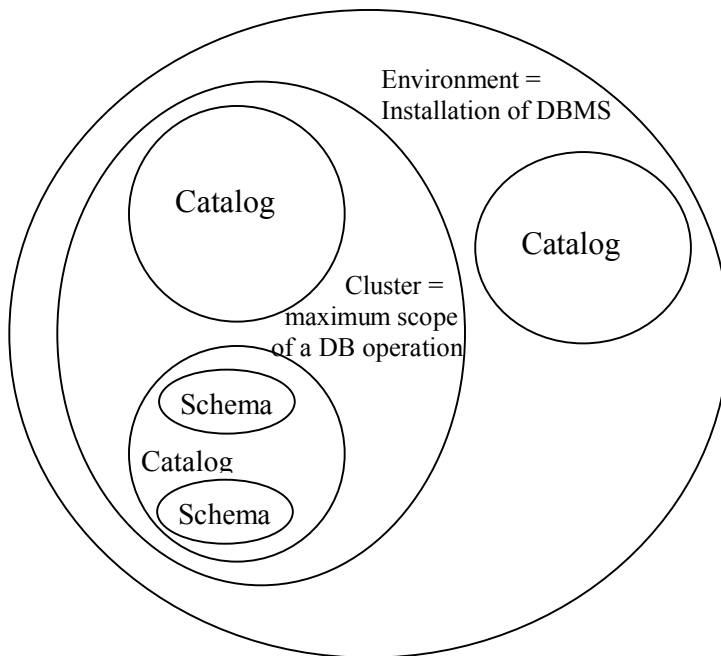
Tất cả các yếu tố của cơ sở dữ liệu mà chúng ta đã thảo luận – các bảng, các khung nhìn, các trigger, các thủ tục được lưu giữ, v.v - được định nghĩa trong một môi trường SQL. Những yếu tố này được tổ chức thành một phân cấp của các cấu trúc, mỗi cấu trúc đóng một vai trò riêng biệt trong tổ chức. Những cấu trúc được chuẩn SQL định nghĩa được chỉ ra trong Hình 3.15.

Một cách ngắn gọn, tổ chức bao gồm những cấu trúc sau đây:

1. *Các lược đồ (schemas)*. Đó là tập hợp các bảng, các khung nhìn, các khẳng định nhận, các trigger, các mô đun PSM, và một vài kiểu thông tin khác mà chúng ta không thảo luận trong quyển sách này (nhưng xem hộp “Thêm về các yếu tố lược đồ nữa” trong Phần 3.3.2). Các lược đồ là những đơn vị tổ chức cơ bản, gần với cái chúng ta có thể



- nghĩ như một “cơ sở dữ liệu”, nhưng thực sự có phần kém hơn một cơ sở dữ liệu như chúng ta sẽ thấy trong điểm (3) dưới đây.
2. *Các danh mục (catalog)*. Đó là tập hợp các lược đồ. Mỗi danh mục có một hoặc nhiều lược đồ; các tên của lược đồ trong một danh mục phải là duy nhất, và mỗi danh mục bao chứa một lược đồ đặc biệt gọi là lược đồ thông tin, nó bao chứa thông tin về tất cả các lược đồ trong danh mục.
  3. *Các cluster*. Đó là tập hợp các danh mục. Mỗi người sử dụng có một cluster liên kết: đó là tất cả các danh mục mà người sử dụng có thể truy cập tới (xem Phần 8.7 để được lý giải việc truy cập tới các danh mục và các yếu tố khác được điều khiển như thế nào). Một cluster là phạm vi lớn nhất mà thông qua nó một truy vấn có thể được đưa ra; như vậy, theo một nghĩa nào đó, một cluster là “cơ sở dữ liệu” như một người dùng cụ thể nhìn thấy.



Hình 3.15: Sự tổ chức của các yếu tố cơ sở dữ liệu trong môi trường

### 3.3.2 Lược đồ

Dạng đơn giản của khai báo lược đồ bao gồm:

## Từ khóa CREATE SCHEMA

### Tên của lược đồ

Một danh sách các khai báo của các yếu tố lược đồ chẳng hạn như các bảng cơ sở, các khung nhìn, và các khẳng định.

Chẳng hạn, một lược đồ có thể được khai báo như sau:

```
CREATE <tên lược đồ> <các khai báo thành phần>
```

Các khai báo thành phần là những mẫu được thảo luận trong ở những nơi khác nhau, chẳng hạn như trong chương 1, chương 2 và đầu của chương này.

Ví dụ: Chúng ta có thể khai báo một lược đồ bao gồm năm quan hệ về các bộ phim chúng ta đang sử dụng trong ví dụ đang chạy của chúng ta, cộng với một số các yếu tố khác chúng ta đã giới thiệu, chẳng hạn như những khung nhìn. Hình 3.16 phác họa dạng một khai báo như vậy

```
CREATE SCHEMA CÔNGTY
```

```
CREATE TABLE NHÂNVIÊN ....
```

Các lệnh tạo bảng đối với các bảng khác

```
CREATE VIEW NVHANOI .....
```

Các lệnh tạo khung nhìn khác

```
CREATE ASSERTION Quanly.....
```

H×nh 3.16: Khai báo lược đồ

Việc khai báo lược đồ tất cả một lần là không cần thiết. Người ta có thể sửa đổi hoặc thêm vào một lược đồ bằng việc sử dụng các câu lệnh CREATE, DROP, hoặc ALTER một cách thích hợp, ví dụ CREATE TABLE theo sau bởi việc khai báo một bảng mới cho một lược đồ. Một vấn đề là hệ thống SQL cần biết bảng mới thuộc về lược đồ nào. Nếu chúng ta sửa đổi hoặc xóa một bảng hoặc một phần tử lược đồ khác, chúng ta cũng cần làm rõ tên của các phần tử, vì hai hoặc nhiều lược đồ có thể có những phần tử khác nhau nhưng cùng tên.

Chúng ta thay đổi lược đồ “hiện tại” với một câu lệnh SET SCHEMA. Ví dụ

SET SCHEMA CONGTY;

Làm cho lược đồ được mô tả trong Hình 3.16 trở thành lược đồ hiện tại. Sau đó, các mô tả của các phân tử lược đồ được thêm vào lược đồ đó, và các câu lệnh DROP hoặc ALTER bất kỳ tham khảo đến các phân tử đã có trong lược đồ đó.

### 3.3.3 Các danh mục (Catalog)

Cũng như là các yếu tố lược đồ giống như các bảng được tạo trong một lược đồ, các lược đồ được tạo và sửa đổi trong một danh mục. Về nguyên tắc, chúng ta có thể hy vọng quá trình tạo ra và phổ biến các danh mục cũng tương tự với quá trình tạo ra và phổ biến các lược đồ. Thật đáng tiếc, SQL không định nghĩa một cách thức chuẩn để làm việc này, chẳng hạn như một câu lệnh

```
CREATE CATALOG <tên danh mục>
```

Theo sau bởi một danh sách các lược đồ thuộc về danh mục đó và các khai báo của các lược đồ này.

Tuy nhiên, SQL quy định một câu lệnh

```
SET CATALOG <tên danh mục>
```

Câu lệnh này cho phép chúng ta thiết lập một danh mục “hiện thời”, như vậy các lược đồ mới sẽ đi vào trong danh mục đó và các sửa đổi lược đồ sẽ tham khảo tới các lược đồ trong danh mục đó sẽ có sự nhập nhằng về tên.

Thêm về các phân tử lược đồ

Một số các phân tử lược đồ chúng ta chưa kể ra nhưng đôi khi chúng rất hữu ích, đó là:

**Domains:** Đó là các tập hợp giá trị hoặc các kiểu dữ liệu đơn giản. Ngày nay chúng ít được sử dụng bởi vì các hệ quản trị cơ sở dữ liệu quan hệ - đối tượng cung cấp các cơ cấu tạo kiểu mạnh hơn.

**Tập các ký tự:** Đó là các tập ký tự và các phương pháp để mã hóa chúng. ASCII là tập ký tự nổi tiếng nhất, nhưng một cài đặt SQL có thể hỗ trợ nhiều tập ký tự khác, chẳng hạn như tập hợp các ngôn ngữ nước ngoài khác nhau.

Sự đối chiếu (Collations): Cho phép đối chiếu (so sánh) các xâu ký tự. Một collation chỉ rõ những ký tự nào là “nhỏ hơn” các ký tự khác

Các lệnh Grant: Chúng liên quan đến những người truy cập đến các phần tử của lược đồ. Chúng ta sẽ thảo luận về việc cấp đặc quyền trong phần 3.7

### 3.3.4 Client và Server trong môi trường SQL

Một môi trường SQL có nhiều hơn một tập hợp các danh mục và lược đồ. Nó bao chứa các các phần tử có mục đích là hỗ trợ các thao tác trên cơ sở dữ liệu hoặc các cơ sở dữ liệu được các danh mục và lược đồ đó miêu tả. Trong một môi trường SQL có hai kiểu tiến trình đặc biệt: các SQL client và các SQL server. Một server hỗ trợ các thao tác trên các phần tử cơ sở dữ liệu, và một client cho phép một người dùng kết nối đến một server và thao tác trên cơ sở dữ liệu. Có thể tưởng tượng rằng server chạy trên một máy chủ lớn, máy chủ này lưu trữ các cơ sở dữ liệu và client chạy trên một máy chủ khác, có thể là một máy trạm cá nhân cách xa so với server. Tuy nhiên, cũng có thể cả client và server chạy trên cùng một máy chủ.

### 3.3.5 Kết nối

Nếu chúng ta có một chương trình nào đó có chứa SQL tại một máy mà ở đó có tồn tại SQL client thì chúng ta có thể mở một kết nối giữa Client và Server bằng cách thực hiện lệnh SQL:

```
CONNECT TO < Tên Server> AS < Tên kết nối>  
AUTHORIZATION < Tên và mật khẩu >
```

Tên của Server phụ thuộc vào sự cài đặt. Từ DEFAULT có thể thay thế cho một tên và sẽ kết nối người dùng với một SQL Server nào đó mà sự cài đặt xem như một Server mặc định. Tên và mật khẩu của người dùng là một phương pháp để cho Server có thể xác định được người dùng. Người ta có thể sử dụng một xâu khác đi sau AUTHORIZATION.

Tên kết nối có thể được sử dụng để sau này tham chiếu đến kết nối. Nguyên nhân của việc chúng ta phải tham chiếu đến kết nối là SQL cho phép người sử dụng mở nhiều kết nối nhưng tại một thời điểm chỉ có một kết nối hoạt động. Để chuyển qua chuyển lại giữa các kết nối, chúng ta có thể sử dụng lệnh

SET CONNECTION conn1;

Lệnh này kích hoạt kết nối có tên là conn1, kết nối hiện thời trở thành “ngủ”. Một kết nối “ngủ” được kích hoạt lại bằng lệnh SET CONNECTION có ghi rõ tên nó.

Chúng ta cũng sử dụng tên kết nối khi muốn đình chỉ kết nối. Ví dụ, chúng ta có thể đình chỉ kết nối conn1 bằng lệnh:

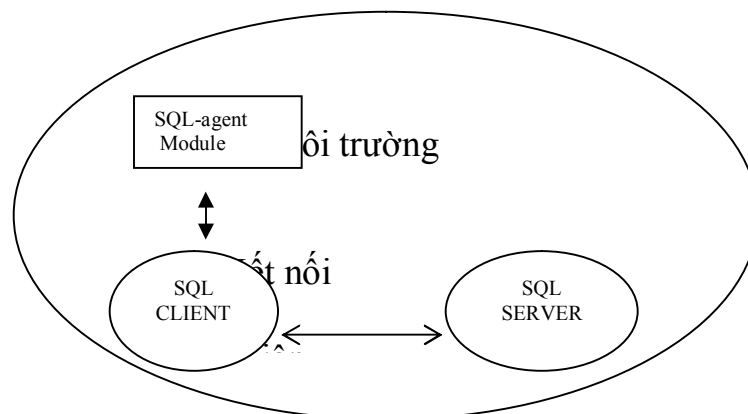
DISCONNECT conn1;

Bây giờ conn1 kết thúc. Nó không phải ngủ và không thể kích hoạt lại được.

Tuy nhiên, nếu chúng ta sẽ không bao giờ cần tham chiếu đến một kết nối đã được tạo ra thì AS và tên kết nối có thể bị bỏ khỏi lệnh CONNECT TO. Việc chuyển các lệnh kết nối liên tiếp nhau cũng được cho phép. Nếu chúng ta thực hiện các lệnh SQL một cách đơn giản tại một trạm có SQL Client thì một kết nối mặc định sẽ được thiết lập thay cho chúng ta.

### 3.3.6 Phiên (Session)

Các thao tác SQL được thực hiện khi một kết nối hoạt động được gọi là một phiên. Phiên tồn tại chung cùng với kết nối đã tạo ra nó. Ví dụ, khi một kết nối được làm ngủ, phiên của nó cũng trở thành ngủ và việc hoạt động trở lại của kết nối bằng lệnh SET CONNECTION cũng làm cho phiên hoạt động. Như vậy, chúng ta đã chỉ ra phiên và kết nối như hai mặt của kết nối giữa Client và Server ở trong hình 3.17



Hình 3.17: Các giao tiếp SQL client-server

Mỗi phiên có một danh mục hiện tại và một lược đồ hiện tại bên trong danh mục đó. Chúng có thể được thiết lập bằng các lệnh SET SCHEMA và SET CATALOG tương ứng. Với mỗi phiên có một người sử dụng được phép.

### 3.3.7 Modules

Một module là một thuật ngữ SQL dành cho một chương trình ứng dụng. Chuẩn SQL đề xuất ba loại module nhưng chỉ nhấn mạnh rằng mỗi cài đặt SQL cung cấp cho người dùng ít nhất là một trong các loại đó.

1. Giao diện SQL chung (generic SQL interface): Người sử dụng gõ vào các lệnh SQL, các lệnh này sẽ được một SQL server thực hiện. Trong chế độ này, một lệnh hoặc một truy vấn chính là một module. Đa số các ví dụ được cho trong tài liệu này là một module thuộc loại này mặc dù trong thực tế nó rất ít được sử dụng.

2. SQL nhúng: Kiểu này đã được trình bày ở phần 3.1, trong đó các lệnh SQL xuất hiện bên trong các chương trình ngôn ngữ chủ và được giới thiệu bằng EXEC SQL. Trước tiên, một bộ tiền xử lý sẽ biến đổi các lệnh SQL nhúng thành các lời gọi hàm hoặc thủ tục đến hệ thống SQL thích hợp. Chương trình ngôn ngữ chủ đã được biên dịch, bao gồm cả các lời gọi này, là một module.

3. Các module thực thụ (True Modules): Dạng tổng quát nhất của module do SQL cung cấp là dạng mà trong đó có một tập các hàm hoặc các thủ tục được lưu giữ, một số trong chúng là mã ngôn ngữ chủ và một số là các câu lệnh SQL. Chúng trao đổi với nhau bằng cách chuyển tham số thông qua các biến dùng chung. Các module PSM là các ví dụ về kiểu module này.

Một thực hiện của một module được gọi là một SQL-agent. Trong hình 3.17 chúng ta đã chỉ ra cả một module và một SQL-agent, như là một đơn vị yêu cầu một SQL Client thiết lập một kết nối. Tuy nhiên, chúng ta cần nhớ rằng sự khác nhau giữa một module và một SQL-agent tương tự như sự khác

nhau giữa một chương trình và một tiến trình. Chương trình là mã lệnh còn tiến trình là sự thực hiện các mã lệnh đó.

### **3.4 SỬ DỤNG GIAO DIỆN MỨC GỌI (call-level interface)**

Trong phần này chúng ta trở lại việc kết hợp các thao tác SQL và các chương trình ngôn ngữ chủ. Chúng ta đã nhìn thấy SQL nhúng trong phần 3.1, chúng ta cũng đã nói đến các thủ tục được lưu giữ trong lược đồ ở phần 3.2. Trong phần này chúng ta thảo luận một cách tiếp cận thứ ba, đó là việc sử dụng giao diện mức gọi. Khi sử dụng một giao diện mức gọi (call-level interface – CLI), chúng ta viết một chương trình bằng một ngôn ngữ chủ thông thường và chúng ta sử dụng một thư viện các hàm, các hàm này cho phép chúng ta kết nối đến và truy cập một cơ sở dữ liệu bằng cách chuyển các lệnh SQL cho cơ sở dữ liệu đó.

Sự khác nhau giữa cách tiếp cận này và lập trình SQL nhúng, theo một nghĩa nào đó, là hình thức. Nếu chúng ta đã quan sát bộ tiền xử lý làm gì với các lệnh SQL nhúng, chúng ta có thể tìm thấy rằng chúng đã được thay thế bằng các lời gọi đến các hàm thư viện giống như các hàm trong chuẩn SQL/CLI. Tuy nhiên, khi các hàm CLI chuyển SQL một cách trực tiếp đến máy chủ cơ sở dữ liệu, có một ích lợi về một mức độ lập hệ thống nào đấy. Như vậy, về nguyên tắc, chúng ta có thể chạy cùng một chương trình ngôn ngữ chủ tại nhiều trạm có sử dụng các hệ quản trị cơ sở dữ liệu khác nhau. Chừng nào các hệ quản trị cơ sở dữ liệu này chấp nhận SQL chuẩn thì cùng một chương trình có thể chạy tại mọi trạm mà không cần các bộ tiền xử lý được thiết kế đặc biệt.

Một trong các giao diện mức gọi là SQL/CLI chuẩn.

#### **3.4.1 Nhập môn SQL/CLI**

Một chương trình được viết trong ngôn ngữ C và sử dụng SQL/CLI sẽ chứa file đầu sqlcli.h, từ đó nó nhận được một số lớn các hàm, các định nghĩa kiểu, các cấu trúc và các hằng ký hiệu. Sau đó chương trình có thể tạo ra và làm việc với 4 loại bản ghi:

1. Môi trường: Một bản ghi kiểu này được tạo ra bằng một chương trình ứng dụng (client) trong việc chuẩn bị cho một hoặc nhiều kết nối đến server cơ sở dữ liệu.
2. Kết nối: Một trong các bản ghi này được tạo ra để kết nối chương trình ứng dụng với cơ sở dữ liệu. Mỗi kết nối tồn tại bên trong một môi trường nào đấy.
3. Câu lệnh: Một chương trình ứng dụng có thể tạo ra một hoặc nhiều bản ghi câu lệnh. Mỗi bản ghi giữ thông tin về một lệnh SQL đơn giản bao gồm một cursor mặc nhiên nếu câu lệnh là một truy vấn. Ở các thời điểm khác nhau, cùng một lệnh CLI có thể biểu diễn các lệnh SQL khác nhau. Mỗi một câu lệnh CLI tồn tại bên trong một kết nối nào đấy.
4. Mô tả: Các bản ghi này giữ thông tin về các bộ hoặc về các tham số. Chương trình ứng dụng hoặc máy chủ cơ sở dữ liệu sẽ thiết lập các thành phần của các bản ghi mô tả để chỉ ra tên và kiểu của các thuộc tính và/hoặc các giá trị của chúng. Mỗi câu lệnh có nhiều các bản ghi được tạo ra một cách không tường minh như vậy, và người dùng có thể tạo ra nhiều hơn nếu cần. Trong trình bày của chúng ta về CLI, các bản ghi mô tả nối chung là sẽ không nhìn thấy.

Mỗi một bản ghi này được biểu diễn trong chương trình ứng dụng bằng một handle, đó là một con trỏ đến bản ghi. File sqlcli.h cung cấp các kiểu đối với các handle môi trường, kết nối, câu lệnh, mô tả tương ứng là: SQLHENV, SQLHDBC, SQLHSTMT, SQLHDESC, mặc dù chúng ta có thể nghĩ về chúng như các biến trỏ hoặc các số nguyên. Chúng ta sẽ sử dụng các kiểu này và một số kiểu được định nghĩa khác với các giải thích rõ ràng, như là SQLCHAR và SQLINTEGER. Các kiểu này được cung cấp trong sqlcli.h.

Chúng ta sẽ không đi đến chi tiết về việc các mô tả được thiết lập và được sử dụng như thế nào. Tuy nhiên ta sẽ bàn về ba kiểu bản ghi khác được tạo ra bằng cách sử dụng một hàm

`SQLAllocHandle(hType, hIn, hOut)`

Trong đó, ba tham số là:



1. hType là kiểu của handle mong muốn. Sử dụng SQL\_HANDLE\_ENV cho một môi trường mới, SQL\_HANDLE\_DBC cho một kết nối mới, hoặc SQL\_HANDLE\_STMT cho một câu lệnh mới.
2. hIn là một handle của phần tử mức cao trong đó phần tử vừa mới tạo ra sống. Tham số này là SQL\_NULL\_HANDLE nếu chúng ta muốn một môi trường; tên của môi trường là một hằng được định nghĩa nói với SQLAllocHandle rằng không có giá trị thích hợp ở đây. Nếu chúng ta muốn một handle kết nối thì hIn là handle của môi trường mà trong đó có tồn tại kết nối. Nếu chúng ta muốn một handle câu lệnh, thì hIn là handle của kết nối mà trong đó câu lệnh sẽ tồn tại.
3. hOut là địa chỉ của handle do SQLAllocHandle tạo ra.

SQLAllocHandle cũng trả lại một giá trị thuộc kiểu SQLRETURN (một số nguyên). Giá trị này là 0 nếu không có lỗi nào xảy ra, và là một giá trị khác 0 nếu có xuất hiện lỗi.

Ví dụ 3.18 Chúng ta hãy xem hàm HangLuong của hình 3.4 sẽ bắt đầu trong CLI như thế nào. (Hàm HangLuong đã được chúng ta sử dụng như một ví dụ về SQL nhúng). Nhớ lại rằng hàm này khảo sát tất cả các bộ của NHÂNVIÊN và xếp loại lương của chúng. Các bước đầu tiên được chỉ ra ở hình 3.1.8:

- 1) #include sqlcli.h
- 2) SQLHENV MyEnv;
- 3) SQLHDBC MyCon;
- 4) SQLHSTMT execStat
- 5) SQLRETURN errorCode1, errorCode2, errorCode3;
- 6) ErrorCode1=SQLAllocHandle(SQL\_HANDLE\_ENV, SQL\_NULL\_HANDLE, &myEnv);
- 7) If(!errorCode1)
- 8) ErrorCode2 = SQLAllocHandle(SQL\_HANDLE\_DBC, myEnv, &myCon);
- 9) If(!errorCode2)

10)

```
ErrorCode3=SQLAllocHandle(SQL_HANDLE_STMT,myCon,&execStat);
```

Hình 3.18: Mô tả và tạo ra một môi trường, một kết nối và một lệnh.

Các dòng từ 2) đến 4) mô tả handle cho một môi trường, kết nối và câu lệnh tương ứng. Tên của chúng là myEnv, myCon và execStat tương ứng. Chúng ta dự kiến rằng execStat sẽ biểu diễn lệnh

```
SELECT Luong FROM NHANVIEN ;
```

giống như con trỏ execCursor đã làm trong hình 3.4 nhưng vẫn chưa có lệnh SQL nào được liên kết với execStat. Dòng 5) khai báo ba biến mà các lời gọi hàm có thể đặt trả lời của chúng vào đó và chỉ ra một lỗi. Một giá trị 0 chỉ ra không có lỗi nào xảy ra trong lời gọi và chúng ta đang mong đợi trường hợp đó.

Dòng 6) gọi SQLAllocHandle, yêu cầu một handle môi trường (đối số thứ nhất), cung cấp một handle null trong đối số thứ hai (bởi vì khi chúng ta đang yêu cầu một handle môi trường thì chúng ta không cần gì cả) và cung cấp một địa chỉ của myEnv như là đối số thứ ba; handle được tạo ra sẽ được đặt ở đây. Nếu dòng 6) là thành công, các dòng 7) và 8) sử dụng handle môi trường để nhận một handle kết nối trong myCon. Giả thiết rằng lời gọi đó cũng thành công, các dòng 9) và 10) nhận một handle lệnh cho execStat.

### 3.4.2 Xử lý các lệnh

Ở cuối của hình 3.18, một bản ghi lệnh có handle là execStat đã được tạo ra. Tuy nhiên bản ghi này vẫn chưa được nối với lệnh SQL. Tiến trình kết nối và thực hiện các lệnh SQL với các handle tương tự như SQL động được mô tả trong phần 3.1.10. Ở đây, chúng ta kết hợp văn bản của một lệnh SQL với một “ biến SQL” bằng cách sử dụng PREPARE và sau đó thực hiện nó bằng cách sử dụng EXECUTE.

Tình trạng trong CLI hoàn toàn tương tự nếu chúng ta nghĩ về “ biến SQL” như là một handle lệnh. Có một hàm

SQLPrepare (sh, st, sl), nó lấy:

1. Một handle lệnh sh
2. Một biến trỏ đến một lệnh SQL st, và
3. Một độ dài sl cho xâu ký tự do st chỉ đến. Nếu chúng ta không biết độ dài, một hằng được định nghĩa SQL\_NTS yêu cầu SQLPrepare tự tính nó từ xâu. Có thể là xâu là một “xâu kết thúc bằng null” và SQLPrepare kiểm tra nó cho đến khi gặp dấu hết xâu ‘\0’ là đủ.

Hiệu quả của hàm này là chuẩn bị để cho lệnh được handle sh trỏ đến bây giờ biểu thị lệnh SQL cụ thể st.

Một hàm khác

SQLExecute(sh)

làm cho lệnh do sh trỏ đến được thực hiện. Đối với nhiều dạng lệnh SQL, chẳng hạn như các lệnh chèn hoặc các lệnh xóa, hiệu quả của việc thực hiện lệnh này trên cơ sở dữ liệu là hiển nhiên. Khi lệnh SQL do sh tham chiếu đến là một truy vấn, hiệu quả của lệnh kém rõ ràng hơn. Như chúng ta sẽ thấy trong phần 8.4.3, có một con trỏ ẩn cho lệnh này, con trỏ đó là một phần của chính bản ghi lệnh. Về nguyên tắc lệnh sẽ được thực hiện, vì vậy chúng ta có thể tưởng tượng rằng tất cả các bộ của trả lời đang nằm ở một nơi nào đó sẵn sàng để được truy cập. Chúng ta có thể lấy ra các bộ, mỗi lần một bộ bằng cách sử dụng con trỏ ẩn giống như chúng ta làm việc với các con trỏ thật trong các phần 3.1 và 3.2.

Ví dụ 3.19: Chúng ta hãy tiếp tục với hàm HangLuong ở ví dụ 3.18. Hai lời gọi hàm sau đây sẽ liên kết truy vấn SELECT Luong FROM NHANVIEN với lệnh do handle execStat trỏ đến:

- 11) SQLPrepare(execStat, “SELECT Luong FROM NHANVIEN “, SQL\_NTS);
- 12) SQLExecute(execStat) ;

Hai lệnh này có thể xuất hiện ngay sau dòng 10) của hình 3.18. Nhớ lại rằng SQL\_NTS yêu cầu SQLPrepare xác định độ dài của xâu kết thúc bằng null do đối số thứ hai của nó tham chiếu đến.

Giống như với SQL động, các bước chuẩn bị và thực hiện có thể được tổ hợp thành một nếu chúng ta sử dụng hàm SQLExecDirect. Một ví dụ về tổ hợp các dòng 110 và 12) như sau:

```
SQLExecDirect(execStat, "SELECT Luong FROM NHANVIEN ",  
SQL_NTS);
```

### 3.4.3 Lấy dữ liệu ra từ kết quả truy vấn

Hàm tương ứng với lệnh FETCH trong SQL nhúng hoặc PSM là

```
SQLFetch(sh)
```

trong đó sh là một handle lệnh. Chúng ta giả thiết lệnh do sh trở đến đã được thực hiện, hoặc việc lấy ra sẽ gây ra một lỗi. SQLFetch, giống như tất cả các hàm CLI, trả lại một giá trị kiểu SQLRETURN chỉ ra hoặc thành công hoặc bị lỗi. Chúng ta có thể nhận thấy rằng giá trị trở về do hằng ký hiệu SQL\_NO\_DATA biểu thị, chỉ ra rằng không bộ nào còn lại trong kết quả truy vấn. Giống như trong các ví dụ trước đây của chúng ta về việc lấy ra, giá trị này sẽ được sử dụng để đi ra khỏi vòng lặp mà trong đó chúng ta lấy ra liên tiếp các bộ từ kết quả truy vấn.

Tuy nhiên, nếu chúng ta đặt sau SQLExecute của ví dụ 8.19 một hoặc nhiều dòng gọi SQLFetch thì bộ xuất hiện ở đâu? Câu trả lời là các thành phần của nó đi vào một trong các bản ghi mô tả liên kết với lệnh mà handle của nó xuất hiện trong lời gọi SQLFetch. Chúng ta có thể rút ra cùng một thành phần tại mỗi lần thử bằng cách liên kết thành phần với một biến ngôn ngữ chủ trước khi chúng ta bắt đầu lấy ra. Hàm làm nhiệm vụ đó là

```
SQLBindCol (sh, colNo, colType, pVar, varSize, varInfo)
```

Ý nghĩa của sáu đối số này là:

1. sh là handle của của lệnh liên quan
2. colNo là số của thành phần (bên trong bộ) mà chúng ta nhận giá trị của nó

3. colType là mã của kiểu của biến mà giá trị của thành phần được đặt vào đây. Ví dụ về các mã do sqlcli.h cung cấp là SQL\_CHAR đối với các mảng hoặc các chuỗi ký tự, SQL\_INTEGER đối với các số nguyên.
4. pVar là một biến trỏ chỉ đến biến mà giá trị được đặt vào đó.
5. varSize là độ dài tính bằng byte của giá trị của biến được pVar chỉ đến.
6. varInfor là biến trỏ chỉ đến một số nguyên có thể được SQLBindCol sử dụng để cung cấp thông tin phụ về giá trị được sản xuất ra.

Ví dụ 3.20. Chúng ta làm lại toàn bộ hàm HangLuong từ hình 3.4 bằng cách sử dụng các dòng gọi CLI thay cho SQL nhúng. Chúng ta bắt đầu từ Hình 3.18 nhưng để ngắn gọn, chúng ta bỏ qua các kiểm tra lỗi trừ kiểm tra xem có phải SQLFetch chỉ ra rằng không còn bộ nào xuất hiện hay không. Chương trình được chỉ ra ở hình 3.19:

- 1) #include sqlcli.h
- 2) void Hạngluong () {
- 3) int i, Chuso, Dem[15];
- 4) SQLHENV MyEnv;
- 5) SQLHDBC MyCon;
- 6) SQLHSTMT execStat
- 7) SQLINTEGER luong, luongInfo;
- 8) SQLAllocHandle(SQL\_HANDLE\_ENV,SQL\_NULL\_HANDLE,  
&myEnv);
- 9) SQLAllocHandle(SQL\_HANDLE\_DCB,myEnv, &myCon);
- 10) SQLAllocHandle(SQL\_HANDLE\_STMT,myCon,&execStat);
- 11) SQLPrepare(execStat, “SELECT Luong FROM NHANVIEN “,  
SQL\_NTS);
- 12) SQLExecute(execStat) ;
- 13) SQLBindCol(execStat, 1, SQL\_INTEGER, &luong, size(luong),  
&luongInfo);
- 14) While(SQLFetch (execStat != SQL\_NO\_DATA {

```
15) Chuso = 1;
16) while((LuongNV /= 10) > 0) Chuso++;
    8)  if(Chuso <= 14) Dem[Chuso]++;          }
    9)  for (i=0; i<15; i++)
10)   printf("chuso = %d: so nhan vien = %d\n",
           i, Dem[i]);                          }
```

Hình 3.19 Đếm số nhân viên có lương gồm 1 chữ số, 2 chữ số, ....

Dòng 3) khai báo các biến cục bộ giống như trong SQL nhưng và các dòng 4) và 7) khai báo các biến cục bộ thêm bằng cách sử dụng các kiểu được sqlcli.h cung cấp.

### 3.5 GIAO TÁC TRONG SQL

Cho đến lúc này, mô hình các thao tác trên cơ sở dữ liệu của chúng ta là một người sử dụng đang truy vấn và sửa đổi cơ sở dữ liệu. Như vậy, các thao tác trên cơ sở dữ liệu được thực hiện một lần một thao tác, và trạng thái cơ sở dữ liệu để lại sau mỗi thao tác là trạng thái mà thao tác tiếp theo sẽ hành động trên đó. Hơn nữa, chúng ta hình dung rằng các thao tác được thực hiện trong trạng thái nguyên vẹn của nó ("một cách nguyên tử"). Như vậy, chúng ta đã giả thiết rằng phần cứng và phần mềm không thể hỏng ở giữa thao tác, để lại cơ sở dữ liệu trong tình trạng không thể giải thích được như là kết quả của thao tác thực hiện trên nó.

Cuộc sống thực thường là phức tạp hơn nhiều. Trước tiên chúng ta xem xét cái có thể xảy ra làm cho cơ sở dữ liệu ở trạng thái không phản ánh các thao tác thực hiện trên nó và sau đó chúng ta sẽ xem xét các công cụ SQL cung cấp cho các người sử dụng để đảm bảo rằng những vấn đề đó không xảy ra.

#### 3.5.1 Xếp hàng theo thứ tự

Trong các ứng dụng như là ngân hàng hoặc đăng ký vé máy bay, mỗi giây có hàng trăm thao tác có thể thực hiện trên cơ sở dữ liệu. Các thao tác này bắt đầu tại một trong hàng trăm hoặc hàng nghìn trạm như là máy trên bàn

của một đại lý du lịch, nhân viên hàng không, .... Việc chúng ta có thể có hai thao tác thực hiện trên cùng một tài khoản hoặc một chuyến bay và chồng chéo thời gian là điều hoàn toàn có thể xảy ra. Nếu như vậy, chúng có thể tương tác theo các cách lạ lùng. Đây là một ví dụ về cái có thể dẫn đến sai lầm nếu như hệ quản trị cơ sở dữ liệu không bị ràng buộc đầy đủ đối với thứ tự mà theo thứ tự đó nó thao tác trên cơ sở dữ liệu. Chúng ta nhấn mạnh rằng các hệ cơ sở dữ liệu không xử sự một cách bình thường theo cách này. Và rằng người ta có thể đi ra theo cách của người ta để làm cho các loại sai lầm này xảy ra khi sử dụng một hệ quản trị thương mại.

- 1) EXEC SQL BEGIN DECLARE SECTION;
- 2) int flight ; /\* số chuyến bay \*/
- 3) char date[10] ; /\*ngày bay theo khuôn dạng SQL \*/
- 4) char seat[3] ; /\* hai chữ số và một chữ cái biểu thị một chỗ ngồi \*/
- 5) int occ ; /\* biến logic để xem chỗ có bị bận hay chưa \*/
- 6) EXEC SQL END DECLARE SECTION ;
- 7) void chooseSeat () {
- 8) /\* Mã chương trình C để nhắc người sử dụng nhập vào chuyến bay, ngày và chỗ ngồi và lưu giữ chúng vào ba biến có tên ở trên \*/
- 9) EXEC SQL UPDATE occupied INTO:occ
- 10) FROM Flights
- 11) WHERE fltNum =:flight AND fltDate =: date AND fltSeat =:seat;
- 12) if(!occ) {
- 13) EXEC SQL UPDATE Flights
- 14) SET occupied = TRUE
- 15) WHERE fltNum =: flight AND fltDate =:date AND fltSeat = seat ;

- ```
16)      /* Mã chương trình C để ghi lại việc phân chỗ và thông báo
          cho người sử dụng việc phân chỗ */ }
17)      else /* mã chương trình C để thông báo cho người sử dụng
          việc chỗ đã bị chiếm và yêu cầu chọn chỗ khác */
          }
```

Hình 3.22: Chọn một chỗ

Ví dụ 3.26: Giả sử rằng chúng ta viết một hàm chooseSeat() trong C với SQL nhúng để đọc một quan hệ về các chuyến bay và các chỗ sẵn có, tìm một chỗ sẵn có cụ thể và làm cho nó trở thành bị chiếm. Quan hệ mà chúng ta thao tác trên đó được gọi là Flights và nó có các thuộc tính fltNum, fltDate, fltSeat, và occ với ý nghĩa rõ ràng.

Từ dòng (9) đến dòng (11) là một select đơn hàng và làm cho biến chia sẻ occ trở thành true hoặc false (1 hoặc 0) phụ thuộc vào việc chỗ được chỉ ra đã bị chiếm hay chưa. Dòng (12) kiểm tra xem chỗ đã bị chiếm hay chưa, và nếu chưa, bộ giá trị đối với chỗ này sẽ được cập nhật để làm cho nó trở thành bị chiếm. Việc cập nhật được thực hiện từ dòng (13) đến dòng (15) và tại dòng (16) chỗ được phân cho người sử dụng yêu cầu nó. Trên thực tế, chúng ta có thể lưu trữ thông tin phân chỗ vào một quan hệ khác. Cuối cùng, ở dòng (17), nếu chỗ đã bị chiếm thì người sử dụng cũng được thông báo như vậy.

Bây giờ hãy nhớ rằng hàm chooseSeat () có thể được hai hay nhiều khách hàng thực hiện một cách đồng thời. Giả sử rằng hai đại lý đang cố gắng mua cùng một chỗ đối với cùng chuyến bay và cùng ngày tại cùng một thời điểm như gợi ý trong hình 3.23. Cả hai đi đến dòng (9) cùng một lúc và cả hai bản sao biến cục bộ occ nhận giá trị 0; như vậy, chỗ ngồi hiện đang còn trống. Tại dòng (12), mỗi thực hiện của chooseSeat() quyết định sửa đổi occ thành TRUE và như vậy là làm cho chỗ thành bị chiếm. Các cập nhật này thực hiện đồng thời và mỗi thực hiện nói với khách hàng ở dòng (16) rằng chỗ thuộc về họ.

Như chúng ta thấy từ ví dụ 3.26, có thể thừa nhận rằng hai thao tác có thể được thực hiện một cách đúng đắn nhưng kết quả cuối cùng là không đúng



đến: cả hai khách hàng tin rằng đã đăng ký được chỗ theo yêu cầu. Vấn đề có thể được giải quyết bằng nhiều cơ cấu SQL phục vụ cho xếp hàng có thứ tự sự thực hiện của hai thực hiện hàm. Chúng ta nói về các hàm thao tác trên cùng một cơ sở dữ liệu là có thứ tự nếu một hàm thực hiện hoàn thành trước khi một hàm khác bắt đầu. Chúng ta nói thực hiện là làm có thứ tự được nếu chúng xử sự như là chúng đã chạy một cách có thứ tự dù rằng việc thực hiện của chúng là có thời gian gối lên nhau.

Rõ ràng là nếu hai lời gọi hàm choSeat() chạy có thứ tự thì sai sót mà chúng ta đã nhìn thấy không thể xảy ra. Một lời gọi của khách hàng xuất hiện đầu tiên. Khách hàng này nhìn thấy một chỗ trống và mua chỗ đó. Sau đó lời gọi của khách hàng thứ hai bắt đầu và nhìn thấy chỗ đó đã bị chiếm. Nó có thể quan trọng đối với những khách hàng đăng ký chỗ nhưng đối với cơ sở dữ liệu, điều quan trọng là một chỗ chỉ được bán một lần.

### 3.5.2 Atomicity

Bên cạnh cách xử sự không có thứ tự có thể xảy ra nếu có hai hoặc nhiều phép toán cơ sở dữ liệu thực hiện cùng một thời điểm, còn có thể có khả năng một phép toán đơn đặt cơ sở dữ liệu vào một trạng thái không chấp nhận được nếu có một sự sụp đổ phần cứng hoặc phần mềm khi phép toán đang thực hiện. Đây là một ví dụ khác gợi ý cái gì có thể xảy ra. Cũng như trong ví dụ 8.26, chúng ta phải nhớ rằng các hệ cơ sở dữ liệu thực sự không cho phép kiểu sai sót này xảy ra trong các chương trình ứng dụng được thiết kế đúng đắn.

Ví dụ 3.27: Chúng ta xét một loại cơ sở dữ liệu khác: các bản ghi tài khoản của ngân hàng. Chúng ta có thể trình bày tình huống bằng một quan hệ Accounts với các thuộc tính accNo và balance. Cặp trong quan hệ này là mã số tài khoản và số dư trong tài khoản đó.

Chúng ta muốn viết một hàm transfer() đưa vào hai tài khoản và một số tiền, kiểm tra xem tài khoản thứ nhất có ít nhất là số tiền đó hay không và nếu có thì chuyển tiền từ tài khoản thứ nhất sang tài khoản thứ hai. Hình 8.24 là phác thảo của hàm transfer():

1) EXEC SQL BEGIN DECLARE SECTION;

```
2) int acct, acct2 ; /* hai tài khoản */
3) int balance1 ; /* số tiền trong tài khoản thứ nhất */
4) int amount ; /* số tiền cần chuyển */
5) EXEC SQL END DECLARE SECTION;
6) void transfer() {
7) /* Mã C nhắc người sử dụng nhập vào các tài khoản 1 và 2 và một số
   tiền cần chuyển, vào các biến acct1, acct2, và amount. */
8) EXEC SQL SELECT balance INTO:balance1
9) FROM Accounts
10) WHERE acctNo =:acct1 ;
11) if(balance1 >= amount) {
12) EXEC SQL UPDATE Accounts
13) SET balance = balance +:amount
14) WHERE acctNo =:acct2 ;
15) EXEC SQL UPDATE Accounts
16) SET balance = balance -:amount
17) WHERE acctNo =:acct1 ;
   }
18) else /* Mã C để in thông báo rằng không có đủ tiền để chuyển */
```

Hình 3.24 Chuyển tiền từ tài khoản này sang tài khoản khác.

Công việc của hình 3.24 là kế tiếp. Các dòng (8) đến dòng (10) lấy ra số dư của tài khoản thứ nhất. Ở dòng (11), kiểm tra xem số dư đó có đủ để rút số tiền mong muốn ra hay không. Nếu đủ, thì các dòng từ (12) đến (14) thêm số tiền đó vào tài khoản thứ hai và các dòng từ (15) đến (17) trừ số tiền đó ra khỏi tài khoản thứ nhất. Nếu số tiền trong tài khoản thứ nhất là không đủ thì không có sự chuyển tiền xảy ra và một lời cảnh báo sẽ được in ra ở dòng (18).

Bây giờ chúng ta xem cái gì xảy ra nếu có một hư hỏng sau dòng (14). Có thể đó là một sự hư hỏng của máy tính hoặc mạng nối cơ sở dữ liệu với bộ xử lý đang thực hiện việc chuyển tiền hồng. Khi đó cơ sở dữ liệu được đặt ở trạng thái tiền đã được chuyển vào tài khoản thứ hai nhưng chưa được trừ đi khỏi tài khoản thứ nhất. Kết quả là ngân hàng mất số tiền đã được chuyển đi.

Vấn đề được minh họa bởi ví dụ 3.27 là một tổ hợp các phép toán cơ sở dữ liệu, như là hai phép update ở hình 3.24, cần được thực hiện một cách nguyên tử, nghĩa là cả hai đều được thực hiện hoặc không phép toán nào được thực hiện. Ví dụ, một lời giải đơn giản làm tất cả các thay đổi trong một nơi cục bộ và chỉ sau khi tất cả công việc được thực hiện chúng ta sẽ ghi các thay đổi vào cơ sở dữ liệu, và sau đó tất cả các thay đổi trở thành một phần của cơ sở dữ liệu và nhìn thấy được đối với các phép toán khác.

### 3.5.3 Giao tác (Transaction)

Cách giải quyết các vấn đề về xếp hàng thứ tự và nguyên tử hóa đặt ra trong các phần 3.6.1 và 3.6.2 là nhóm các phép toán vào các giao tác. Một giao tác là một tập hợp của một hoặc nhiều phép toán trên cơ sở dữ liệu sao cho chúng được thực hiện một cách nguyên tử; nghĩa là hoặc tất cả các phép toán được thực hiện hoặc không phép toán nào được thực hiện. Hơn nữa, SQL đòi hỏi rằng, như ngầm định, các giao tác được thực hiện theo cách xếp hàng thứ tự. Một hệ quản trị cơ sở dữ liệu có thể cho phép người sử dụng chỉ ra một ràng buộc ít nghiêm ngặt hơn trên việc chèn của các phép toán từ hai hoặc nhiều giao tác. Chúng ta sẽ thảo luận những sửa đổi đối với điều kiện xếp hàng thứ tự này trong các phần sau. Khi sử dụng giao diện SQL chung, mỗi một lệnh chính là một giao tác. Tuy nhiên, Khi viết chương trình với SQL nhúng hoặc chương trình sử dụng SQL/CLI hoặc JDBC, chúng ta thường muốn kiểm tra các giao tác một cách rõ ràng. Các giao tác bắt đầu một cách tự động khi một lệnh SQL bắt đầu truy vấn hoặc thao tác cơ sở dữ liệu hoặc lược đồ. Nếu muốn chúng ta có thể sử dụng lệnh SQL `START TRANSACTION`.

Trong giao diện chung, trừ phi được bắt đầu bằng lệnh `START TRANSACTION` giao tác kết thúc cùng với lệnh. Trong các trường hợp khác, có hai cách để kết thúc một giao tác:

1. Lệnh SQL COMMIT khiến giao tác kết thúc một cách thành công. Bất cứ cái gì làm thay đổi đối với cơ sở dữ liệu được gây ra do một lệnh hoặc các lệnh SQL từ khi giao tác hiện tại bắt đầu sẽ được đặt thường trực vào cơ sở dữ liệu (tức là chúng được giữ lại). Trước khi lệnh COMMIT được thực hiện, các thay đổi là không dứt khoát và có thể hoặc không thể nhìn thấy được đối với các giao tác khác.
2. Lệnh SQL ROLLBACK khiến giao tác kết thúc không thành công (hoặc bỏ dở). Mọi thay đổi trong việc trả lời cho các lệnh SQL của giao tác là không thực hiện (tức là chúng được rolled back), vì vậy chúng không còn xuất hiện trong cơ sở dữ liệu.

Có một ngoại lệ đối với hai điểm trên. Nếu chúng ta cố gắng lưu giữ một giao tác nhưng có các ràng buộc chậm và các ràng buộc đó bây giờ bị vi phạm thì giao tác không được ghi lại dù chúng ta nói với nó bằng lệnh COMMIT. Hơn nữa, giao tác sẽ bị bỏ dở và một chỉ dẫn trong SQLSTATE nói với ứng dụng rằng giao tác bị bỏ dở do nguyên nhân này.

Ví dụ 3.28: Giả sử chúng ta muốn một thực hiện của hàm transfer() của hình 8.24 là một giao tác đơn. Giao tác bắt đầu ở dòng (8) khi chúng ta đọc số dư của tài khoản thứ nhất. Nếu kiểm tra ở dòng (11) là đúng và chúng ta thực hiện việc chuyển tiền thì chúng ta có thể muốn ghi lại các thay đổi đã được làm. Như vậy, chúng ta đặt ở cuối khối if từ dòng (12) đến dòng (17) thêm lệnh SQL

```
EXEC SQL COMMIT ;
```

Nếu kiểm tra ở dòng (11) là sai – nghĩa là không có đủ tiền để thực hiện việc chuyển – thì chúng ta có thể bỏ dở việc chuyển. Chúng ta có thể làm như vậy bằng cách đặt EXEC SQL ROLLBACK ở cuối khối else ở dòng (18). Hiện tại, bởi vì trong nhánh này không có lệnh sửa đổi cơ sở dữ liệu nào được thực hiện, chúng ta lưu giữ (commit) hoặc bỏ dở (abort) cũng không vấn đề gì bởi vì không có thay đổi nào được lưu giữ cả.

### 3.5.4 Read-Only Transaction

Mỗi ví dụ 3.26 và 3.27 kéo theo một giao tác đọc và sau đó có thể ghi một vài dữ liệu vào cơ sở dữ liệu. Loại giao tác này nghiêng về các vấn đề xếp

hàng thứ tự. Như chúng ta đã nhìn thấy trong ví dụ 3.26 cái gì có thể xảy ra nếu hai thực hiện của hàm cố gắng mua cùng một chỗ tại cùng một thời điểm, và chúng ta cũng đã nhìn thấy trong ví dụ 3.27 cái gì có thể xảy ra nếu có một sự hỏng hóc trong thời gian thực hiện hàm. Tuy nhiên, khi một giao tác chỉ đọc dữ liệu và không ghi dữ liệu chúng ta sẽ tự do hơn trong việc cho một giao tác thực hiện song song với các giao tác khác.

Ví dụ 3.29: Giả sử chúng ta đã viết một hàm đọc các dữ liệu để xác định xem một chỗ nào đó có sẵn sàng hay không, hàm này sẽ xử sự giống như dòng (1) đến dòng (11) của hình 8.22. Chúng ta có thể thực hiện nhiều lần gọi hàm này cùng một lúc mà không sợ làm hại đến cơ sở dữ liệu. Điều tệ hại nhất có thể xảy ra là khi chúng ta đang đọc về sự sẵn sàng của một chỗ nào đó thì chỗ đó có thể đã bị mua hoặc được giải phóng bởi sự thực hiện của một hàm khác nào đó. Như vậy, chúng ta có thể nhận được câu trả lời “sẵn sàng” hoặc “bị chiếm” trong khoản thời gian rất ngắn khi ta thực hiện truy vấn nhưng câu trả lời sẽ có nghĩa trong một lúc.

Nếu chúng ta báo với hệ thống thực hiện SQL rằng giao tác hiện tại của chúng ta là read-only nghĩa là chúng sẽ chẳng bao giờ làm thay đổi cơ sở dữ liệu thì hệ thống SQL hoàn toàn có thể có khả năng nhận ưu điểm của kiến thức đó. Nói chung, việc nhiều giao tác read-only truy cập đến cùng một dữ liệu để chạy song song là có thể được, trong khi đó chúng sẽ không cho phép chạy song song với một giao tác ghi cùng một dữ liệu.

Chúng ta báo cho hệ thống SQL rằng giao tác tiếp theo sau là read-only bằng lệnh

```
SET TRANSACTION READ ONLY ;
```

Lệnh này phải được thực hiện trước khi giao tác bắt đầu. Ví dụ, nếu chúng ta có một hàm bao gồm các dòng từ (1) đến dòng (11) của hình 3.22, chúng ta có thể khai báo nó là read only bằng cách đặt

```
EXEC SQL SET TRANSACTION READ ONLY ;
```

ngay trước dòng (9), nơi bắt đầu giao tác. Nếu khai báo read-only sau dòng (9) thì sẽ quá muộn.

Chúng ta cũng có thể thông báo cho SQL rằng giao tác sắp tới có thể ghi dữ liệu bằng lệnh

```
SET TRANSACTION READ WRITE ;
```

Tuy nhiên, tùy chọn này là ngầm định và làm điều đó là không cần thiết.

### 3.5.5 Dirty Read

Các dữ liệu bẩn (dirty data) là một thuật ngữ chung chỉ các dữ liệu được ghi bằng một giao tác nhưng còn chưa được lưu giữ lại (committed). Một dirty read dùng để đọc các dữ liệu bẩn. Điều nguy hiểm của việc đọc các dữ liệu bẩn là ở chỗ một giao tác ghi nó có thể bị bỏ dở. Nếu vậy thì các dữ liệu bẩn sẽ bị đẩy ra khỏi cơ sở dữ liệu và mọi người được phép xử sự như là các dữ liệu đó chưa bao giờ tồn tại. Nếu một giao tác khác nào đó đã đọc các dữ liệu bẩn thì giao tác đó có thể lưu giữ hoặc thực hiện một hành động nào đó phản ánh sự hiểu biết của nó về dữ liệu bẩn.

Đôi lúc dirty read có ý nghĩa, đôi lúc nó không có ý nghĩa. Lúc khác nó có ý nghĩa rất nhỏ đủ để tạo ý nghĩa về nguy cơ của một dirty read phụ động và như vậy làm ngăn cản:

1. Công việc tốn thời gian của hệ quản trị cơ sở dữ liệu cần để ngăn ngừa dirty read và
2. Mất tính song song gây ra từ sự chờ đợi cho đến khi không có thể có một dirty read.

Sau đây là một số ví dụ về những cái có thể xảy ra khi cho phép có các dirty read.

Ví dụ 3.30: Chúng ta hãy xem xét việc chuyển tài khoản của ví dụ 8.27. Tuy nhiên, giả sử rằng các vụ chuyển được thực hiện bằng một chương trình P thực hiện dãy các bước sau đây:

1. Thêm tiền vào tài khoản 2
2. Kiểm tra nếu tài khoản 1 có đủ tiền
  - a) Nếu không có đủ tiền, lấy tiền ra khỏi tài khoản 2 và kết thúc
  - b) Nếu có đủ tiền, trừ số tiền từ tài khoản 1 và kết thúc.

Nếu chương trình P được thực hiện một cách có thứ tự thì việc chúng ta thêm tiền tạm thời vào tài khoản 2 sẽ không có ý nghĩa gì. Không ai sẽ nhìn thấy số tiền đó và nó sẽ bị loại bỏ nếu việc chuyển tiền là không thực hiện được.

Tuy nhiên, giả sử rằng có các dirty read. Hãy tưởng tượng có 3 tài khoản A1, A2, A3 với 100\$, 200\$ và 300\$ tương ứng. Giả sử rằng giao tác T1 thực hiện chương trình P để chuyển 150\$ từ A1 đến A2. Cùng một thời gian, giao tác T2 chạy chương trình P để chuyển 250\$ từ A2 đến A3. Có khả năng có các dãy sự kiện sau:

1. T2 thực hiện bước 1 và thêm 250\$ vào A3 và bây giờ A3 có 550\$
2. T1 thực hiện bước 1 và thêm 150\$ vào A2 và bây giờ A2 có 350\$
3. T2 thực hiện kiểm tra của bước 2 và tìm ra rằng A2 có đủ tiền (350\$) để cho phép chuyển 250\$ từ A2 sang A3.
4. T1 thực hiện kiểm tra của bước 2 và tìm ra rằng T1 không có đủ tiền (100\$) để cho phép chuyển 150\$ từ A1 sang A2.
5. T2 thực hiện bước 2b. Nó trừ đi 250\$ khỏi A2 và bây giờ A2 có 100\$ và kết thúc.
6. T1 thực hiện bước 2a. Nó trừ 150\$ khỏi A2, bây giờ A2 có -50\$ và kết thúc.

Tổng số tiền không thay đổi; trong ba tài khoản vẫn còn 600\$. Nhưng bởi vì T2 đọc dữ liệu bản ở bước 3 trong 6 bước trên, chúng ta không bảo vệ được việc một tài khoản trở nên âm, đó là mục đích của việc kiểm tra tài khoản thứ nhất để xem tài khoản này có số tiền thích hợp hay không.

Ví dụ 3.31: Chúng ta hãy xét một thay đổi trên hàm seat-choosing của ví dụ 8.26. Trong cách tiếp cận mới:

1. Chúng ta tìm thấy một chỗ sẵn sàng và đăng ký nó bằng cách làm cho occ thành TRUE đối với chỗ đó.
2. Chúng ta hỏi khách hàng có đồng ý với chỗ. Nếu khách hàng đồng ý, ta giữ (commit). Nếu không, ta giải phóng chỗ bằng cách làm cho occ thành FALSE và lặp lại bước 1 để lấy chỗ khác.

Nếu hai giao tác đang thực hiện thuật toán tại cùng một thời điểm, một giao tác có thể đăng ký chỗ S và sau đó lại giải phóng nó do khách hàng. Nếu giao tác thứ hai thực hiện bước 1 tại thời điểm khi chỗ S được đánh dấu bị chiếm, khách hàng đối với giao tác này sẽ không được cho lựa chọn để lấy chỗ S.

Cũng như trong ví dụ 8.30, vấn đề là ở chỗ có dirty read. Giao tác thứ hai nhìn thấy bộ giá trị (với S được đánh dấu là bị chiếm) đã được giao tác thứ nhất ghi và sau đó được giao tác thứ nhất sửa đổi.

Sự kiện một read là dirty là quan trọng như thế nào? Trong ví dụ 30 nó rất quan trọng; nó gây ra một tài khoản trở thành âm mặc dù bộ phận an toàn chống lại điều đó. Trong ví dụ 8.31, vấn đề không đến nỗi quan trọng lắm. Hiển nhiên, khách hàng thứ hai có thể không chọn được chỗ ưng ý hoặc ngay cả được trả lời là không còn chỗ nào. Tuy nhiên, trong trường hợp sau, khi chạy lại giao tác một lần nữa thì hầu như sẽ nhận được sự sẵn sàng của chỗ S. Việc cài đặt hàm seat-choosing này theo cách cho phép các dirty read để làm nhanh thời gian xử lý trung bình đối với các yêu cầu mua vé là một điều có ý nghĩa.

SQL cho phép chúng ta chỉ ra rằng các dirty read là chấp nhận được với một giao tác cho trước. Chúng ta sử dụng lệnh SET TRANSACTION. Dạng thích hợp cho một giao tác giống như mô tả trong ví dụ 8.31 là:

- 1) SET TRANSACTION READ WRITE
- 2) ISOLATION LEVEL READ UNCOMMITTED ;

Lệnh trên làm hai việc:

1. Dòng (1) khai báo rằng giao tác có thể ghi dữ liệu
2. Dòng (2) khai báo rằng giao tác có thể chạy với “ isolation level” read-uncommitted. Điều đó có nghĩa là giao tác được cho phép đọc các dữ liệu bản.

Chú ý rằng, nếu giao tác không phải là read-only (tức là có thể sửa đổi cơ sở dữ liệu) và chúng ta chỉ ra mức cô lập (isolation level) READ UNCOMMITTED thì chúng ta cũng phải chỉ ra READ WRITE. Nhắc lại từ



phần 8.6.4 rằng giả thiết ngầm định là các giao tác là read-write. Tuy nhiên SQL có một ngoại lệ đối với trường hợp có cho phép các dirty-read. Trong trường hợp đó, giả thiết ngầm định là giao tác là read-only, bởi vì các giao tác read-write với dirty read gây ra các nguy hiểm đáng kể như chúng ta đã thấy. Nếu chúng ta muốn một giao tác read-write chạy với read-uncommitted như là mức cô lập thì chúng ta cần chỉ ra READ WRITE một cách rõ ràng như ở trên.

### 3.5.6 Các mức cô lập khác

SQL cung cấp một tổng gồm bốn mức cô lập. Hai mức đã được xem xét: xếp hàng thứ tự và read-uncommitted (cho phép các dirty read). Hai mức còn lại là read-committed và repeatable-read. Chúng có thể được chỉ ra đối với một giao tác cho trước bằng

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
```

hoặc

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

Tương ứng, đối với mỗi mức, ngầm định là các giao tác là read write, vì vậy chúng ta có thể thêm vào READ ONLY cho chúng nếu muốn. Nhân tiện, chúng ta cũng có thể có tùy chọn chỉ ra

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Tuy nhiên, đó là điều mà SQL ngầm định và không cần phải chỉ rõ.

Mức cô lập read-committed, như tên của nó chỉ ra, ngăn cấm việc đọc các dữ liệu bản (uncommitted). Tuy nhiên, nó cho phép một giao tác đưa ra cùng một truy vấn nhiều lần và nhận các trả lời khác nhau miễn là các câu trả lời phản ánh các dữ liệu đã được các giao tác ghi và đã được lưu giữ.

Ví dụ 3.32: Chúng ta hãy xem lại hàm seat-choosing của ví dụ 8.31. Nhưng giả thiết rằng chúng ta khai báo nó chạy với mức cô lập read-committed. Như vậy, khi chúng ta tìm một chỗ ở bước 1, nó sẽ không nhìn thấy chỗ như là đã được mua nếu một vài giao tác nào đó đang đăng ký chúng nhưng chưa lưu giữ. Tuy nhiên, nếu khách hàng không chấp thuận chỗ và một thực hiện của hàm truy vấn các chỗ sẵn sàng nhiều lần, nó có thể

nhìn thấy một tập hợp khác các chỗ sẵn sàng, như là các giao tác khác mua chỗ thành công hoặc bỏ chỗ song song với giao tác của chúng ta.

Bây giờ chúng ta xem mức cô lập repeatable read. Thuật ngữ đôi khi dùng sai bởi vì cùng một truy vấn được đưa ra nhiều hơn một lần không hoàn toàn được đảm bảo là nhận được cùng một câu trả lời. Dưới cô lập repeatable-read, nếu một bộ được lấy ra lần đầu tiên thì chúng ta có thể tin chắc là bộ đó sẽ được lấy ra lại nếu truy vấn được lặp lại. Tuy nhiên, có khả năng là một thực hiện thứ hai hoặc một thực hiện xảy ra sau của cùng một truy vấn sẽ lấy ra các bộ “ma”. Các bộ đó là các bộ là kết quả của các phép chèn vào cơ sở dữ liệu khi giao tác của chúng ta đang được thực hiện.

Ví dụ 3.33: Chúng ta hãy tiếp tục với vấn đề seat-choosing của các ví dụ 3.31 và 3.32. Nếu chúng ta thực hiện hàm này dưới mức cô lập repeatable-read thì một chỗ sẵn sàng trên truy vấn đầu tiên ở bước 1 sẽ còn sẵn sàng ở các truy vấn xảy ra sau.

Tuy nhiên, giả sử rằng có một số bộ mới được chèn vào quan hệ Flights. Ví dụ, công ty hàng không có thể di chuyển đột ngột chuyến bay sang một máy bay lớn hơn, tạo ra một số bộ mới mà trước đó chưa có. Khi đó, dưới mức cô lập repeatable-read, một truy vấn xảy ra sau đối với các chỗ sẵn sàng cũng có thể lấy ra các chỗ mới.

### **3.6 AN TOÀN VÀ CẤP QUYỀN TRONG SQL**

SQL đòi hỏi sự tồn tại của các authorization ID, đó chính là các tên người sử dụng. SQL cũng có một authorization ID riêng, PUBLIC, bao gồm mọi người sử dụng. Các authorization ID có thể được đảm bảo các đặc quyền, giống như chúng ở trong môi trường hệ thống file do hệ điều hành quản lý. Ví dụ, một hệ thống UNIX nói chung kiểm soát ba loại đặc quyền: read, write và execute. Danh sách các đặc quyền đó là có ý nghĩa bởi vì các đối tượng được bảo vệ của hệ thống UNIX là các file và ba phép toán này đặc trưng tốt cho những gì người ta làm việc với các file. Tuy nhiên, các cơ sở dữ liệu phức tạp hơn các hệ thống file nhiều và các loại đặc quyền được sử dụng trong SQL cũng phức tạp hơn tương ứng.

Trong phần này, chúng ta sẽ xem các đặc quyền nào SQL cho phép trên các phần tử cơ sở dữ liệu. Sau đó chúng ta sẽ xem làm thế nào các người sử dụng có thể nhận được các đặc quyền. Cuối cùng, chúng ta sẽ xem các đặc quyền có thể bị tước đi như thế nào.

### 3.6.1 Các quyền

SQL định nghĩa 9 loại quyền: SELECT, INSERT, DELETE, UPDATE, REFERENCES, USAGE, TRIGGER, EXECUTE, và UNDER. Bốn quyền đầu áp dụng đối với quan hệ, có thể đó là một bảng cơ sở hoặc một view. Như tên của nó đã chỉ ra, các quyền này cho người có quyền quyền được truy vấn (select from) quan hệ, chèn vào quan hệ, xóa khỏi quan hệ và sửa đổi các bộ giá trị của quan hệ.

Một modul chứa một lệnh SQL không thể được thực hiện mà không có đặc quyền thích hợp đối với lệnh đó; tức là lệnh select-from-where đòi hỏi đặc quyền SELECT trên mỗi bảng mà nó truy cập đến. Chúng ta sẽ xem một modul nhận các quyền truy cập đó như thế nào một cách ngắn gọn. SELECT, INSERT, và UPDATE cũng có thể có một danh sách các thuộc tính liên kết, ví dụ, SELECT(Tên, Địa chỉ). Nếu có danh sách như vậy thì chỉ có các thuộc tính này mới có thể được nhìn thấy trong phép lựa chọn, được chỉ ra trong phép chèn hoặc được thay đổi trong phép sửa đổi. Chú ý rằng, khi được cấp, các đặc quyền này sẽ liên kết với một quan hệ cụ thể.

Quyền REFERENCES trên một quan hệ là quyền tham chiếu đến một quan hệ trong một ràng buộc tham chiếu. Các ràng buộc này có thể có một trong các dạng đã được chỉ ra trong chương 2, như là các khẳng định, các kiểm tra dựa trên bộ hoặc thuộc tính, hoặc các ràng buộc toàn vẹn tham chiếu. Quyền REFERENCES cũng có thể có một danh sách các thuộc tính kèm theo, trong trường hợp này, chỉ các thuộc tính đó mới có thể được tham chiếu trong các ràng buộc. Một ràng buộc không thể được kiểm tra trừ phi người chủ của lược đồ trong đó xuất hiện ràng buộc có đặc quyền REFERENCES trên tất cả các dữ liệu có trong ràng buộc.

USAGE là một quyền áp dụng cho nhiều loại phần tử lược đồ khác với các quan hệ và các khẳng định. (xem 3.3.2); Nó là quyền sử dụng phần tử ở trong các mô tả của người chủ. Đặc quyền TRIGGER trên một quan hệ là

quyền được định nghĩa trigger trên quan hệ đó. EXECUTE là quyền thực hiện một mẫu chương trình, chẳng hạn như một thủ tục PSM hoặc một hàm. Cuối cùng, UNDER là quyền tạo ra các kiểu con của một kiểu cho trước.

Ví dụ 3.34: Chúng ta hãy xem xét các đặc quyền nào là cần thiết để thực hiện lệnh chèn ở hình 8.25

- 1) INSERT INTO Studio(name)
- 2) SELECT DISTINCT studioName
- 3) FROM Movie
- 4) WHERE studioName NOT IN
- 5) (SELECT name
- 6) FROM Studio);

Hình 3.25 Thêm vào các studio mới.

Trước tiên, nó là một phép chèn vào quan hệ Studio, vì vậy chúng ta đòi hỏi một quyền INSERT trên Studio. Tuy nhiên, bởi vì phép chèn chỉ chỉ ra một thành phần đối với thuộc tính name, việc có quyền INSERT hoặc có đặc quyền INSERT(name) trên quan hệ Studio là chấp nhận được. Quyền INSERT(name) cho phép chúng ta chèn các bộ chỉ chỉ ra thành phần name và từ chối các thành phần khác để lấy các giá trị ngầm định của chúng hoặc NULL.

Tuy nhiên, lệnh chèn ở hình 3.25 kéo theo hai truy vấn con, bắt đầu tại các dòng (2) và (5). Để thực hiện hai phép chọn này chúng ta đòi hỏi các quyền cần thiết cho các truy vấn con. Như vậy, chúng ta cần quyền SELECT trên cả hai quan hệ có trong các mệnh đề FROM: Studio và Movie. Chú ý rằng việc chúng ta vừa có quyền INSERT trên Studio không có nghĩa là chúng ta có quyền SELECT trên Studio và ngược lại. Bởi vì chỉ có các thuộc tính cụ thể của Movie và Studio được lựa chọn nên chỉ cần có đặc quyền SELECT(studioName) trên Movie và quyền SELECT(name) trên Studio hoặc các đặc quyền chứa các thuộc tính này trong danh sách các thuộc tính là đủ.

### 3.6.2 Tạo các quyền

Chúng ta vừa nhìn thấy các đặc quyền SQL là gì và đã quan sát rằng chúng được yêu cầu để thực hiện các phép toán SQL. Bây giờ chúng ta cần biết làm thế nào để nhận được các đặc quyền cần thiết để thực hiện một phép toán. Có hai khía cạnh để nhận các đặc quyền: chúng được tạo ra từ đầu như thế nào và chúng được chuyển từ người dùng này sang người dùng khác như thế nào. Ở đây chúng ta sẽ thảo luận về việc cài đặt các đặc quyền (việc chuyển các đặc quyền sẽ được thảo luận sau).

Trước tiên, các phần tử SQL chẳng hạn như các lược đồ hoặc các module có một chủ (owner). Chủ của cái gì thì có tất cả các đặc quyền đối với cái đó. Có ba điểm mà ở đó chủ quyền được xác lập trong SQL.

1. Khi một lược đồ được tạo ra, nó và tất cả các bảng, các phần tử lược đồ khác trong nó được giả thiết là do người sử dụng tạo ra nó làm chủ. Vì vậy người sử dụng này có tất cả các quyền có thể trên các phần tử của lược đồ.
2. Khi một kết nối được bắt đầu bằng một lệnh CONNECT, có một cơ hội để chỉ ra người sử dụng với một mệnh đề AUTHOIATION. Ví dụ, lệnh kết nối

```
CONNECT TO Starfleet-sql-server AS conn1
```

```
AUTHOIATION nam;
```

sẽ tạo ra một kết nối có tên là conn1 đến một server SQL có tên là Starfleet-sql-server nhân danh người sử dụng Nam.

3. Khi một module được tạo ra, có một tùy chọn để cho nó một chủ bằng cách sử dụng mệnh đề AUTHOIATION. Ví dụ, mệnh đề

```
AUTHOIATION thanh ;
```

trong lệnh tạo module sẽ làm cho người sử dụng Thanh trở thành chủ của module. Việc không chỉ ra chủ cho một module là chấp nhận được, trong trường hợp đó module được thực hiện một cách công cộng nhưng các đặc quyền cần thiết để thực hiện các phép toán trong module phải đến từ một

nguồn khác nào đó, chẳng hạn như người sử dụng liên kết với kết nối và phiên mà trong đó module được thực hiện.

### 3.6.3 Tiến trình kiểm tra đặc quyền

Như chúng ta nhìn thấy ở trên, mỗi module, lược đồ và phiên có một người sử dụng liên kết. Trong thuật ngữ SQL, có một authorizationID liên kết đối với mỗi đối tượng. Phép toán nào của SQL cũng có hai phần:

1. Các phần tử của cơ sở dữ liệu mà phép toán được thực hiện trên chúng và
2. Agen gây nên phép toán.

Các đặc quyền sẵn sàng cho agent rút ra từ một authorizationID cụ thể gọi là authorizationID hiện tại. ID này hoặc là

- a) authorizationID của module nếu module mà agent này đang thực hiện có một authorizationID, hoặc
- b) authorizationID của phiên nếu module mà agent này đang thực hiện không có một authorizationID

Chúng ta chỉ có thể thực hiện phép toán SQL nếu authorizationID hiện tại có tất cả các quyền cần thiết để thực hiện phép toán trên các phần tử cơ sở dữ liệu được bao hàm.

Ví dụ 3.35: Để xem cơ cấu kiểm tra các quyền, ta xem lại ví dụ 3.34. Chúng ta có thể giả thiết rằng các bảng được tham chiếu – Movie và Studio – là một phần của lược đồ gọi là MovieSchema được Hoa tạo ra và làm chủ. Ở điểm này, người sử dụng Hoa có tất cả các quyền trên các bảng và tất cả các phần tử của lược đồ MovieSchems này. Anh ta có thể chọn và phân một vài quyền cho người khác. Có nhiều cách để phép chèn trong ví dụ 8.34 có thể được thực hiện.

1. Phép chèn có thể được thực hiện như là một phần của module do Hoa tạo ra và chứa mệnh đề AUTHOIATION Hoa. AuthorizationID của module nếu có luôn luôn trở thành authorizationID hiện tại. Khi đó, module và lệnh chèn SQL của nó có cùng các quyền như người sử dụng Hoa có, bao gồm tất cả các quyền trên các bảng Movie và Studio.

2. Phép chèn có thể là một phần của một module không có chủ. Người sử dụng Hoa mở một kết nối với một mệnh đề AUTHOIATION hoa trong lệnh CONNECT. Bây giờ Hoa authorizationID hiện tại, vì vậy lệnh chèn có tất cả các quyền cần thiết.
3. Người sử dụng cấp tất cả các đặc quyền trên các bảng Movie và Studio cho người sử dụng Huy hoặc cho một người sử dụng đặc biệt PUBLIC (thay thế cho tất cả các người sử dụng). Lệnh chèn ở trong một module với mệnh đề

AUTHOIATION huy

Bởi vì bây giờ authorizationID hiện tại là Huy, và người sử dụng này có các đặc quyền cần thiết, phép chèn một lần nữa được chấp nhận.

4. Giống như trong (3), người sử dụng Hoa đã cho người sử dụng Huy các quyền cần thiết. Lệnh chèn ở trong một module không có chủ; nó được thực hiện một phiên mà authorizationID của nó được thiết lập bằng mệnh đề AUTHOIATION sisko. AuthorizationID hiện tại là Huy và ID này có các quyền cần thiết.

### 3.6.4 Cấp các quyền

Chúng ta đã thấy trong ví dụ 8.35, điều quan trọng đối với một người sử dụng (tức là một authoriationID) là có các quyền cần thiết. Nhưng ở trên, chúng ta vừa thấy cách duy nhất để có các quyền trên một phần tử cơ sở dữ liệu là phải là người tạo ra và là chủ của phần tử đó. SQL cung cấp lệnh GRANT cho phép một người sử dụng cấp một quyền cho người khác. Người sử dụng thứ nhất vẫn tiếp tục có đặc quyền đã được cho đi ; như vậy, GRANT có thể được hiểu như là “copy quyền”.

Có một điểm khác nhau quan trọng giữa cấp các đặc quyền và sao chép. Mỗi một quyền có một grant option liên kết. Như vậy, một người sử dụng có thể có một quyền như là SELECT trên bảng Movie “with grant option”, trong khi người sử dụng thứ hai có cùng đặc quyền nhưng không có grant option. Khi đó, người sử dụng thứ nhất có thể cấp quyền SELECT trên Movie cho một người sử dụng thứ ba và việc cấp này có thể với hoặc không

với grant option. Tuy nhiên, người sử dụng thứ hai không thể cấp quyền SELECT trên Movie cho bất cứ ai.

Lệnh Grant bao gồm các phần tử sau:

1. Từ khóa GRANT
2. Một danh sách gồm một hoặc nhiều quyền, chẳng hạn SELECT hoặc INSERT(name). Một cách tùy chọn, các từ khóa ALL PRIVILEGES có thể xuất hiện ở đây (chỉ tất cả các quyền đối với cơ sở dữ liệu đang xét)
3. Từ khóa ON
4. Một phần tử cơ sở dữ liệu. Phần tử này thường là một quan hệ, hoặc một bảng cơ sở hoặc một khung nhìn. Nó cũng có thể là một miền hoặc là các phần tử khác.
5. Từ khóa TO
6. Một danh sách gồm một hoặc nhiều người sử dụng (các authorization ID).
7. Các từ khóa WITH GRANT OPTION (tùy chọn).

Như vậy, dạng của lệnh GRANT là

GRANT < danh sách quyền> ON < phần tử của CSDL> TO < danh sách người sử dụng>[< WITH GRANT OPTION>].

Để thực hiện lệnh grant này một cách hợp pháp, người thực hiện nó phải có quyền được cấp và các quyền này phải được giữ với grant option. Tuy nhiên, người cấp quyền có thể có quyền tổng quát hơn (with grant option) quyền được đem cấp. Ví dụ, quyền INSERT(name) trên bảng Studio có thể được đem cấp trong khi người cấp giữ quyền tổng quát hơn INSERT vào bảng Studio với grant option.

Ví dụ 3.36: Người sử dụng Hoa là chủ của lược đồ MovieSchema chứa các bảng

Movie(title, year, length, inColor, studioName, producerC#)

Studio(name, address, pres#)



cấp các quyền INSERT và SELECT trên bảng Studio và quyền SELECT trên Movie cho các người sử dụng Nam và Thanh. Hơn nữa, anh ta kèm theo grant option đối với các quyền này. Các lệnh Grant là:

```
GRANT SELECT, INSERT ON Studio TO nam, thanh  
    WITH GRANT OPTION;  
GRANT SELECT ON Movie TO nam, thanh  
    WITH GRANT OPTION;
```

Bây giờ, Thanh cấp cho người sử dụng Huy các quyền đó nhưng không có grant option. Thanh thực hiện các lệnh:

```
GRANT SELECT, INSERT ON Studio TO huy;  
GRANT SELECT ON Movie TO huy ;
```

Cũng như vậy, Nam cấp cho Huy các quyền tối thiểu đối với phép chèn INSERT(name) trên Studio và SELECT trên Movie. Các lệnh là:

```
GRANT SELECT, INSERT(name) ON Studio TO huy ;  
GRANT SELECT ON Movie TO huy ;
```

Chú ý rằng huy đã nhận quyền SELECT trên Movie và Studio từ hai người sử dụng khác nhau. Anh ta cũng nhận quyền Insert(name) trên Studio hai lần: một cách trực tiếp từ Nam và thông qua quyền tổng quát hơn INSERT từ Thanh.

### 3.6.5 Biểu đồ grant

Vì một dãy grant có thể tạo nên một mạng các cấp quyền và các đặc quyền phức tạp nên người ta biểu diễn các cấp quyền bằng một đồ thị gọi là grant diagram (biểu đồ grant) cho tiện. Một hệ thống SQL duy trì một biểu diễn của biểu đồ này để giữ dấu vết của các quyền và nguyên gốc của nó.

Các đỉnh trong biểu đồ grant tương ứng với một người sử dụng và một quyền. Chú ý rằng một quyền với hoặc không có grant option phải được biểu diễn bằng hai đỉnh khác nhau. Nếu người sử dụng U cấp quyền P cho người sử dụng V và việc cấp đó có thể dựa trên sự kiện là U giữ quyền Q (Q có thể

là P với tùy chọn grant, hoặc có thể là một quyền tổng quát hơn P) thì chúng ta vẽ một cạnh từ đỉnh đối với U/Q đến đỉnh đối với V/P.

Ví dụ 3.37: Hình 3.26 biểu diễn biểu đồ grant của dãy các lệnh cấp quyền ở ví dụ 3.36. Chúng ta sử dụng quy ước rằng dấu \* đi sau tổ hợp người sử dụng - quyền chỉ ra rằng quyền bao gồm grant option. Dấu \*\* đi sau tổ hợp người sử dụng-quyền chỉ ra rằng quyền lấy ra từ quyền sở hữu phần tử cơ sở dữ liệu đang xét và không phải do một sự cấp quyền ở đâu cả. Sự phân biệt này sẽ có tầm quan trọng khi chúng ta thảo luận về việc hủy bỏ các quyền trong phần 8.7.6. Một quyền với hai dấu sao chứa grant option một cách tự động.

### 3.6.6 Hủy bỏ các quyền

Một quyền được cấp có thể bị hủy bỏ bất cứ lúc nào. Trên thực tế, việc hủy bỏ các quyền có thể được yêu cầu theo kiểu dây chuyền (cascade) theo nghĩa là việc hủy bỏ một quyền với grant option đã được chuyển cho các người sử dụng khác có thể yêu cầu các quyền đó cũng bị hủy bỏ. Dạng đơn giản của lệnh hủy là:

1. Từ khóa REVOKE
2. Một danh sách gồm một hoặc nhiều quyền
3. Từ khóa ON
4. Một phần tử cơ sở dữ liệu
5. Từ khóa FROM
6. Một danh sách gồm một hoặc nhiều người sử dụng (các authorization ID).

Như vậy, dạng của một lệnh hủy quyền là như sau:

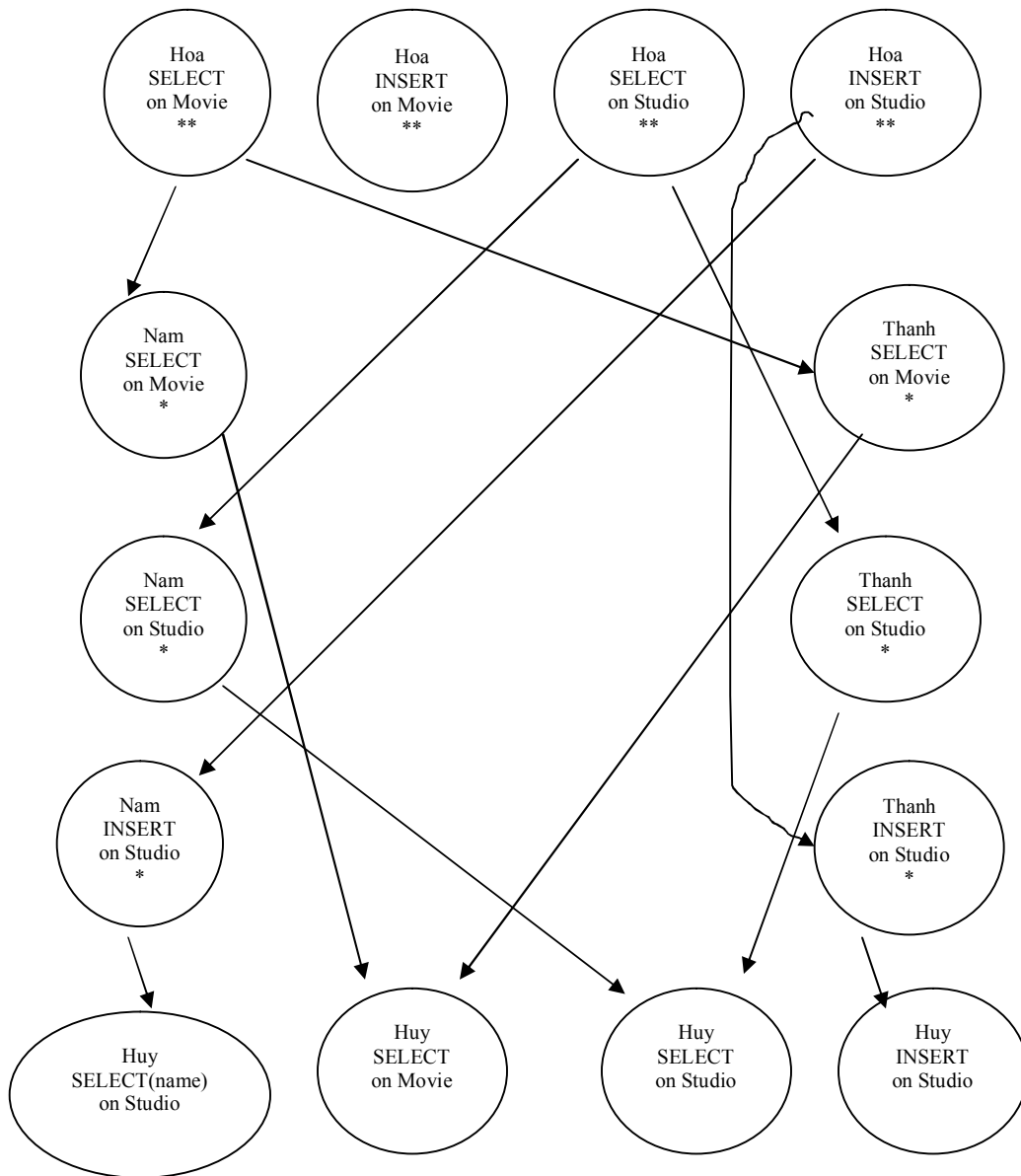
REVOKE < danh sách quyền> ON < phần tử cơ sở dữ liệu> FROM < danh sách người sử dụng>

Tuy nhiên, Một trong các mục sau đây cũng phải có trong lệnh:

1. Một lệnh có thể kết thúc với từ CASCADE. Nếu như vậy thì khi các quyền được chỉ ra bị hủy bỏ, chúng ta cũng hủy bỏ mọi quyền được các

quyền bị hủy bỏ cấp. Chính xác hơn, nếu người sử dụng U đã hủy bỏ quyền P của người sử dụng V dựa trên quyền Q thuộc vào U thì chúng ta loại bỏ cạnh từ U/Q đến V/P trong biểu đồ cấp quyền. Bây giờ đỉnh nào không truy cập được từ một đỉnh chủ cũng bị loại bỏ.

2. Một lệnh có thể kết thúc bằng RESTRICT, điều đó có nghĩa là lệnh hủy không thể được thực hiện nếu luật lan truyền được mô tả trong mục trước có thể dẫn đến việc loại bỏ các quyền do các quyền bị hủy đã chuyển cho quyền khác.



### Hình 3.26 Biểu đồ cấp quyền

Việc thay thế REVOKE bằng REVOKE GRANT OPTION FOR là được phép, trong trường hợp này các quyền cốt lõi vẫn còn nhưng tùy chọn cấp chúng cho người khác bị loại bỏ. Chúng ta có thể sửa đổi một đỉnh, định hướng lại các cạnh, hoặc tạo ra một đỉnh mới để phản ánh các thay đổi đối với các người sử dụng bị ảnh hưởng. Dạng này của REVOKE cũng có thể được thực hiện cùng với CASCADE hoặc RESTRICT.

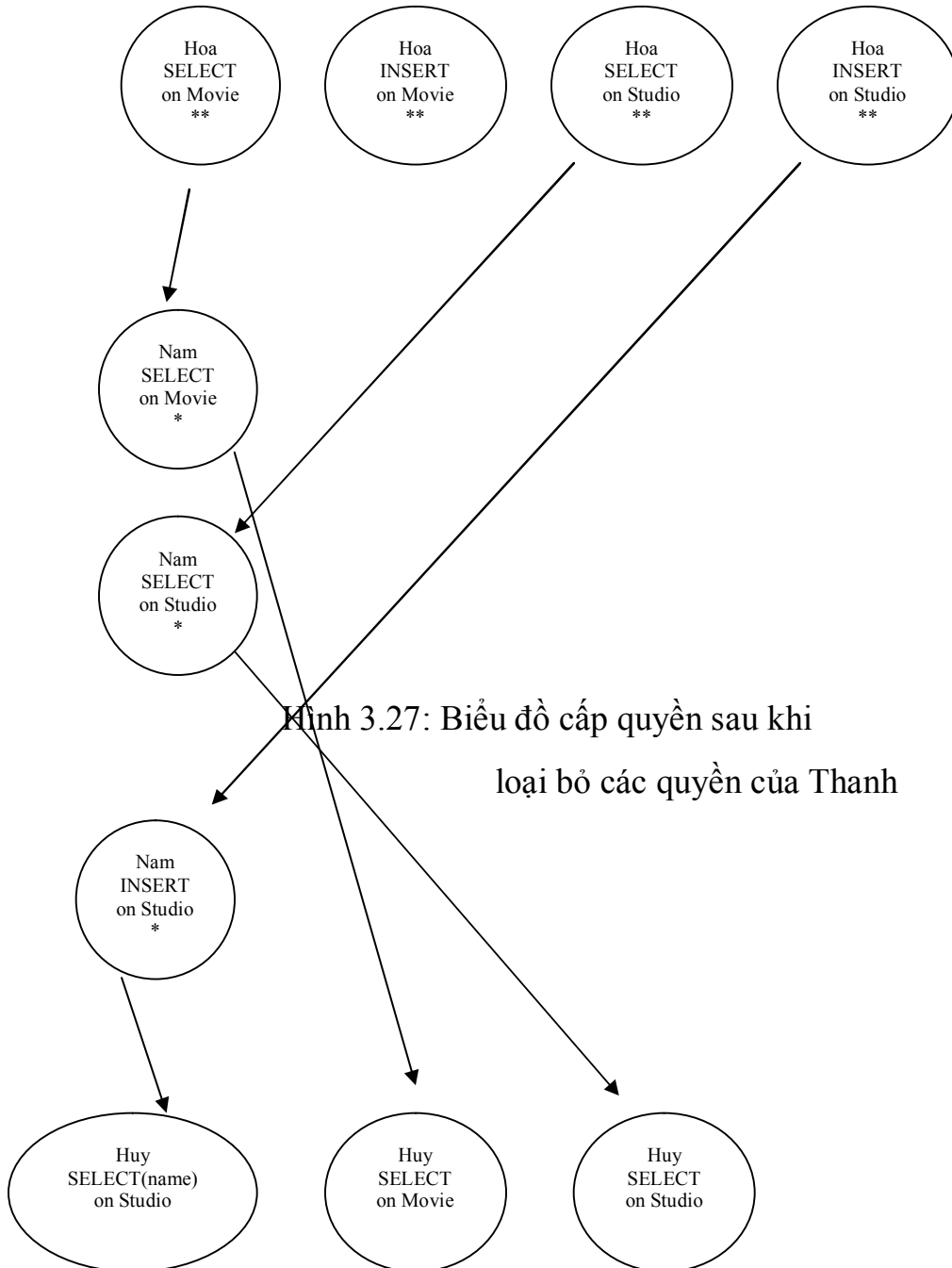
Ví dụ 3.38: Tiếp tục với ví dụ 3.36, giả sử rằng Hoa hủy bỏ các quyền mà anh ta đã cấp cho Thanh với các lệnh:

```
REVOKE SELECT, INSERT ON Studio FROM thanh CASCADE ;
```

```
REVOKE SELECT ON Movie FROM thanh CASCADE ;
```

Chúng ta loại bỏ các cạnh của hình 3.26 từ các quyền này của Hoa đến các quyền tương ứng của Thanh. Bởi vì CASCADE được quy định, chúng ta cũng phải nhìn xem có những quyền nào không thể đi đến được từ một quyền có hai dấu sao. Khảo sát hình 8.26 chúng ta thấy rằng các quyền của Thanh không còn đi đến được từ một đỉnh có hai dấu sao. Cũng như vậy, quyền INSERT vào Studio của Huy cũng không đi đến được. Vậy chúng ta không chỉ bỏ các quyền của Thanh ra khỏi biểu đồ cấp quyền mà còn bỏ quyền INSERT của Huy.

Chú ý rằng chúng ta không loại bỏ các quyền SELECT trên Movie và Studio của Huy hoặc quyền INSERT(name) trên Studio của Huy vì những quyền này có thể đi đến được từ các quyền sở hữu của Hoa thông qua các quyền của Nam. Biểu đồ cấp quyền kết quả được chỉ ra ở hình 3.27.

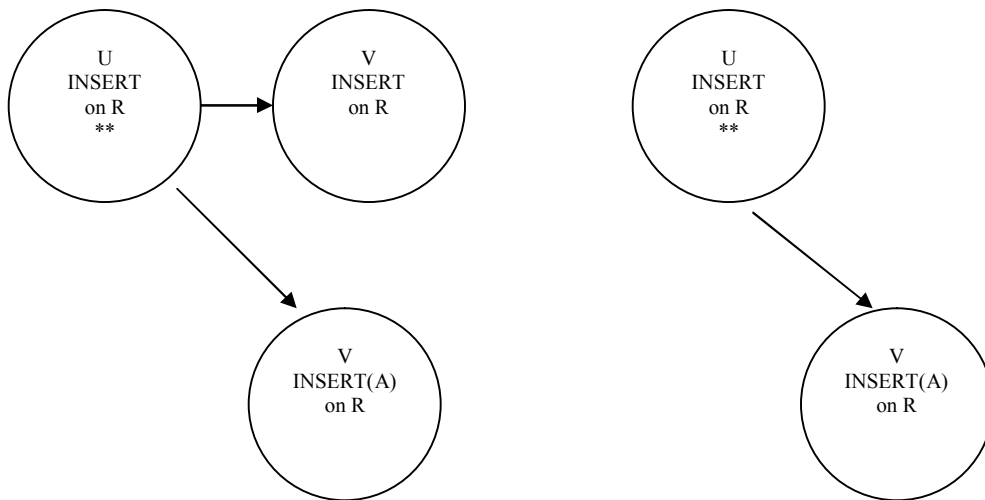


Hình 3.27: Biểu đồ cấp quyền sau khi  
loại bỏ các quyền của Thanh

Ví dụ 3.39: Có một vài điều tinh tế mà chúng ta sẽ minh họa bằng các ví dụ trừu tượng. Đầu tiên, khi chúng ta hủy bỏ một quyền tổng quát p, chúng ta không hủy bỏ một quyền là trường hợp riêng của p. Ví dụ, xét dãy các bước sau đây, người sử dụng U nào đó là chủ của quan hệ R cấp quyền INSERT trên quan hệ R cho V, và cũng cấp quyền INSERT(A) trên cùng một quan hệ

1. U GRANT INSERT ON R TO V
2. U GRANT INSERT(A) ON R TO V
3. U REVOKE INSERT ON R FROM V RESTRICT

Khi U hủy bỏ INSERT ra khỏi V, quyền INSERT(A) vẫn còn. Các biểu đồ cấp quyền sau bước (2) và bước (3) được cho ở hình 3.28.



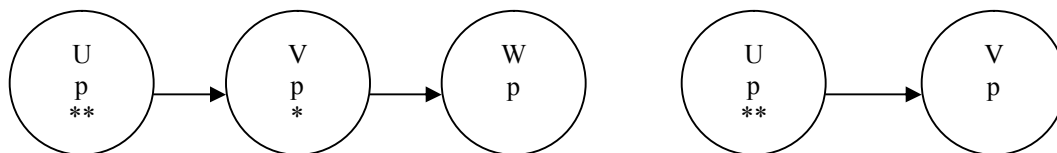
Hình 3.28 Hủy bỏ một quyền tổng quát để lại một quyền riêng

Chú ý rằng sau bước (2) có hai đỉnh rời nhau cho hai quyền khác nhau của V. Cũng chú ý rằng tùy chọn RESTRICT trong bước (3) không ngăn ngừa sự hủy bỏ bởi vì V không cấp cấp tùy chọn cho các người sử dụng khác. Trên thực tế, V không thể cấp quyền bởi vì V nhận được quyền đó không có tùy chọn cấp quyền.

Ví dụ 3.40 Bây giờ chúng ta hãy xét một ví dụ tương tự trong đó U cấp cho V quyền p với tùy chọn cấp quyền và sau đó chỉ hủy tùy chọn cấp quyền. Trong trường hợp này, chúng ta phải thay đổi đỉnh của V để phản ánh việc mất tùy chọn cấp quyền và các cấp quyền p do V thực hiện phải được hủy bỏ bằng cách loại các cạnh đi ra từ đỉnh V/p. Dãy các bước như sau:

1. U GRANT p TO V WITH GRANT OPTION
2. V GRANT p TO W
3. U REVOKE OPTION FOR p FROM V CASCADE

Trong bước (1), U cấp quyền p cho V với tùy chọn cấp quyền. Trong bước (2), V sử dụng tùy chọn cấp quyền để cấp p cho W. Biểu đồ được đưa ra ở hình 3.29(a). Sau đó trong bước (3), U hủy bỏ tùy chọn cấp quyền đối với quyền p từ V nhưng không hủy bỏ quyền p. Như vậy, dấu sao sẽ bị loại bỏ khỏi đỉnh đối với V và p. Tuy nhiên, một đỉnh không có dấu \* không thể có một cạnh đi ra vì một đỉnh như vậy không thể là nguồn của một việc cấp quyền. Vì thế chúng ta cũng phải loại bỏ cạnh đi ra từ đỉnh V/p đi đến đỉnh W/p. Bây giờ đỉnh W/p không có một đường đi đến nó từ một đỉnh có hai dấu sao biểu thị nguồn gốc của quyền p. Kết quả là đỉnh W/p bị loại bỏ khỏi biểu đồ. Tuy nhiên đỉnh V/p vẫn còn nhưng bị bỏ mất dấu sao biểu thị tùy chọn cấp quyền. Biểu đồ cấp quyền kết quả được cho ở hình 3.29(b).



Sau bước (2)

Sau bước (3)

Hình 3.29 Hủy bỏ một tùy chọn cấp quyền để lại quyền cơ bản

### 3.7 TỔNG KẾT CHƯƠNG III

• SQL nhúng: Thay vì sử dụng giao diện truy vấn chung để biểu thị các truy vấn và các sửa đổi, việc viết chương trình nhúng các truy vấn SQL vào trong một ngôn ngữ chủ thích hợp thường là hiệu quả hơn. Một bộ tiền dịch sẽ chuyển đổi các lệnh SQL nhúng thành ra các lời gọi hàm thích hợp của ngôn ngữ chủ.

- Trở ngại không phù hợp: Mô hình dữ liệu SQL hoàn toàn khác với các mô hình dữ liệu của các ngôn ngữ chủ thông thường. Vì vậy, thông tin trao đổi giữa SQL và ngôn ngữ chủ thông qua các biến dùng chung có thể biểu diễn các thành phần của các bộ trong phần SQL của chương trình.

- Con trỏ: Một con trỏ là một biến của SQL chỉ một trong các bộ giá trị của quan hệ. Việc kết nối giữa ngôn ngữ chủ và SQL được làm dễ dàng nhờ có con trỏ đi cùng với mỗi bộ của quan hệ trong khi các thành phần của bộ hiện tại được lấy đưa vào các biến dùng chung và được xử lý bằng cách sử dụng ngôn ngữ chủ.

- SQL động: Thay vì nhúng các lệnh cụ thể của SQL vào một chương trình ngôn ngữ chủ, chương trình chủ có thể tạo ra các chuỗi ký tự được hệ thống SQL dịch như là các lệnh SQL và thực hiện.

- Các module được lưu giữ thường trực: Chúng ta có thể tạo ra một tập hợp các thủ tục và các hàm như là một phần của lược đồ cơ sở dữ liệu. Chúng được viết trong một ngôn ngữ đặc biệt có tất cả các cấu trúc kiểm tra quen biết cũng như các lệnh SQL. Nó có thể được gọi từ SQL nhúng hoặc thông qua một giao diện truy vấn chung.

- Môi trường cơ sở dữ liệu: Một cài đặt sử dụng một hệ quản trị cơ sở dữ liệu SQL tạo ra một môi trường SQL. Bên trong môi trường, các phần tử cơ sở dữ liệu như là các quan hệ được nhóm thành các lược đồ cơ sở dữ liệu, catalog, các cluster. Một catalog là một tập hợp các lược đồ và một cluster là một tập hợp lớn nhất các phần tử mà một người sử dụng có thể nhìn thấy.

- Các hệ thống Client/Server: Một SQL Client kết nối với một SQL Server tạo nên một kết nối (kết nối giữa hai tiến trình) và một phiên (dãy các thao tác). Các chương trình được thực hiện trong một phiên đi đến từ một module và sự thực hiện của một module được gọi là một agent SQL.

- Giao diện mức gọi: Đó là một thư viện chuẩn các hàm gọi là SQL/CLI hoặc ODBC, nó có thể được kết nối vào chương trình C bất kỳ. Các hàm đó cho các khả năng tương tự như SQL nhúng nhưng không cần có một bộ tiền xử lý.



- Kiểm tra cạnh tranh: SQL cung cấp hai cơ cấu nhằm ngăn ngừa các thao tác cạnh tranh khỏi sự gây phiền phức lẫn nhau: các giao tác và sự hạn chế trên các con trỏ. Sự hạn chế trên các con trỏ bao gồm khả năng khai báo một con trỏ là “không nhạy cảm”(insensitive), trong trường hợp đó con trỏ không nhìn thấy các thay đổi đối với quan hệ của nó.

- Các giao tác: SQL cho phép người lập trình nhóm các lệnh SQL vào các giao tác, chúng có thể được ghi (committed) hoặc khôi phục lại (rolled back hoặc aborted). Các giao tác có thể được rolled back bằng chương trình ứng dụng để xóa bỏ các thay đổi hoặc bằng hệ thống để đảm bảo tính nguyên tử và sự cô lập.

- Các mức cô lập: SQL cho phép các giao tác chạy với bốn mức cô lập, từ ít chặt chẽ đến chặt chẽ nhất: “xếp hàng thứ tự (serializable)”(giao tác phải chạy trước hoặc sau một giao tác khác đã hoàn thành), “repeatable read” (mỗi bộ được đọc trong trả lời cho một truy vấn sẽ xuất hiện lại nếu truy vấn đó được lặp lại), “read-commited” (chỉ có các bộ được ghi bằng các giao tác và đã được ghi mới có thể được giao tác này nhìn thấy) và “read uncommitted” (không có ràng buộc nào trên những cái mà giao tác có thể nhìn thấy).

- Con trỏ và các giao tác chỉ đọc: Một con trỏ hoặc một giao tác có thể được khai báo là chỉ đọc (read-only). Khai báo này là một đảm bảo rằng con trỏ và giao tác sẽ không làm thay đổi cơ sở dữ liệu, do đó thông báo cho hệ thống SQL rằng nó sẽ không làm ảnh hưởng các giao tác hoặc các con trỏ khác theo cách có thể vi phạm sự không nhạy cảm, việc xếp hàng có thứ tự, hoặc các yêu cầu khác.

- Quyền: Với mục đích an toàn, các hệ thống SQL cho phép nhiều loại quyền khác nhau có thể nhận được trên các phần tử cơ sở dữ liệu. Các quyền này bao gồm quyền select (đọc), insert, delete hoặc update các quan hệ, quyền tham chiếu các quan hệ (tham chiếu đến chúng trong một ràng buộc) và quyền được tạo ra các trigger.

- Biểu đồ cấp quyền: Các quyền có thể người sở hữu cấp cho các người sử dụng khác hoặc cho một người sử dụng tổng quát PUBLIC. Nếu được

cấp với tùy chọn cấp quyền thì các quyền có thể được chuyển cho những người khác. Các quyền cũng có thể bị hủy bỏ. Biểu đồ cấp quyền là một cách hữu ích để nhớ lại về lịch sử cấp quyền và hủy bỏ quyền, để giữ dấu vết ai có quyền gì và họ nhận được các quyền đó từ người nào.

## MỘT SỐ BÀI TẬP

I. Với cơ sở dữ liệu như trên, hãy viết các chương trình nhúng (ngôn ngữ chủ là C++) sau:

- 1) Yêu cầu người dùng về giá và tìm PC có giá gần nhất với giá mong muốn. In ra nhà sản xuất, model number và tốc độ của PC.
- 2) Hỏi người dùng về các giá trị tối thiểu của tốc độ, RAM, kích cỡ đĩa cứng mà họ sẽ chấp nhận. Tìm tất cả các laptop thỏa mãn các đòi hỏi đó. In ra các đặc trưng của nó (tất cả các thuộc tính của Laptop) và nhà sản xuất của nó.
- 3) Hỏi người sử dụng về nhà sản xuất. In ra các đặc trưng của tất cả các sản phẩm do nhà sản xuất làm ra. Điều đó có nghĩa là số model, kiểu sản phẩm, và tất cả các thuộc tính của những quan hệ nào thích hợp đối với kiểu đó.
- 4) Hỏi người dùng về “túi tiền” (tổng giá của một PC và printer), và một tốc độ tối thiểu của PC. Tìm hệ thống rẻ nhất (PC cộng với printer)
- 5) Hỏi người dùng về nhà sản xuất, số model, tốc độ, RAM, kích thước đĩa cứng, hoặc loại đĩa CD và giá của một PC mới. Hãy kiểm tra rằng không có PC nào có số model như vậy. Đưa ra lời cảnh báo nếu đúng thế, ngược lại chèn thông tin vào các bảng Product và PC.
- 6) Hạ giá tất cả các PC “cũ” \$100. Hãy đảm bảo rằng bất kỳ một PC “mới” nào được chèn vào trong thời gian chương trình của bạn chạy sẽ không bị hạ giá.

II. Viết các chương trình con

Dựa trên cơ sở dữ liệu ở trên, hãy viết các hàm và các thủ tục sau:

- 1) Lấy giá làm đối số và trả lại số model của PC có giá gần nhất.

- 2) Lấy nhà sản xuất, model, và giá làm đối số và trả lại giá của của kiểu sản phẩm nào có model đã cho.
- 3) Lấy thông tin về model, tốc độ, ram, đĩa cd và giá làm các đối số và chèn thông tin này vào quan hệ PC. Tuy nhiên nếu đã có một PC với model này (nghĩa là vi phạm toàn vẹn thực thể, SQLSTATE = '23000') thì thêm 1 vào số model cho đến khi tìm được số model chưa tồn tại trong bảng.
- 4) Cho trước một giá, hãy đưa ra số của các PC, số của các Laptop và số của các Printer đang được bán trên giá đó.

## PHỤ LỤC 1: CƠ SỞ DỮ LIỆU “CÔNG TY “

### NHÂNVIÊN

| Mã số NV | Họ tên   | Tên   | Ngày sinh  | Địa chỉ  | Giới tính | Lương | Mã số NGS | Mã số DV |
|----------|----------|-------|------------|----------|-----------|-------|-----------|----------|
| NV001    | Lê       | Vân   | 1979-12-02 | Hà nội   | Nam       | 3000  | NV002     | 5        |
| NV002    | Trần Đức | Nam   | 1966-02-14 | Hà nội   | Nam       | 4000  | NV061     | 5        |
| NV010    | Hoàng    | Thanh | 1979-05-08 | Nghệ an  | Nữ        | 2500  | NV014     | 4        |
| NV014    | Phạm     | Bằng  | 1952-06-26 | Bắc ninh | Nam       | 4300  | NV061     | 4        |
| NV016    | Nguyễn   | Son   | 1973-08-14 | Hà nam   | Nam       | 3800  | NV002     | 5        |
| NV018    | Vũ Hương | Giang | 1983-03-26 | Nam định | Nữ        | 2500  | NV002     | 5        |
| NV025    | Trần Lê  | Hoa   | 1980-03-15 | Phú thọ  | Nữ        | 2500  | NV014     | 4        |
| NV061    | Hoàng    | Giáp  | 1947-02-05 | Hà tĩnh  | Nam       | 5500  | Null      | 1        |

### ĐƠN VỊ

| Mã số DV | Tên DV     | Mã số NQL | Ngày bắt đầu |
|----------|------------|-----------|--------------|
| 5        | Nghiên cứu | NV002     | 2000-09-15   |
| 4        | Hành chính | NV014     | 1997-06-24   |
| 1        | Lãnh đạo   | NV061     | 1992-01-25   |

## DỰ ÁN

| TênDA | Mã sốDA | ĐịađiểmDA | Mã sốĐV |
|-------|---------|-----------|---------|
| DA01  | 1       | Hà nội    | 5       |
| DA02  | 2       | Nam đĩnh  | 5       |
| DA03  | 3       | Bắc Ninh  | 5       |
| DA04  | 10      | Hà nội    | 4       |
| DA05  | 20      | Hà nội    | 1       |
| DA06  | 30      | Hà nội    | 4       |

## NHÂNVIÊN DỰ ÁN

| Mã sốNV | Mã sốDA | Sốgiờ |
|---------|---------|-------|
| NV001   | 1       | 32    |
| NV001   | 2       | 7     |
| NV016   | 3       | 40    |
| NV018   | 1       | 20    |
| NV018   | 2       | 20    |
| NV002   | 2       | 10    |
| NV002   | 3       | 10    |
| NV002   | 10      | 10    |
| NV002   | 20      | 10    |
| NV010   | 30      | 30    |
| NV010   | 10      | 10    |
| NV025   | 10      | 35    |
| NV025   | 30      | 5     |
| NV014   | 30      | 20    |
| NV014   | 20      | 15    |
| NVO61   | 20      | null  |

Trong đó, các thuộc tính có ý nghĩa như sau:

### 1. Trong bảng NHÂNVIÊN

- Họđệm, Tên: chỉ họ đệm và tên của nhân viên
- Mã sốNV: Mã số của nhân viên

- Ngàysinh, Địa chỉ, Lương, Giới tính: các thuộc tính của nhân viên
- Mã sốNGS: Mã số của người giám sát nhân viên (một nhân viên có thể có một người giám sát). Thuộc tính này có cùng kiểu với Mã sốNV
- Mã sốĐV: mã số của đơn vị mà nhân viên làm việc cho

## 2. Trong bảng ĐƠN VỊ

- TênĐV: tên của đơn vị
- Mã sốĐV: mã số của đơn vị
- Mã sốNQL: mã số của người quản lý đơn vị (đơn vị trưởng). Người quản lý cũng là một nhân viên. Thuộc tính này có cùng kiểu với Mã sốNV.
- Ngàybắtđầu: chỉ ngày người quản lý bắt đầu quản lý đơn vị.

## 3. Trong bảng DỰ ÁN

- Mã sốDA: mã số của dự án
- TênDA: tên của dự án
- Địa điểmDA: địa điểm của dự án
- Mã sốĐV: Mã số của đơn vị quản lý dự án

### \* Trong bảng NHÂN VIÊN\_DỰ ÁN

- Mã sốDA: mã số của dự án
- Mã sốNV: mã số của nhân viên
- Số giờ: số giờ nhân viên làm việc cho dự án.



## PHỤ LỤC 2: CƠ SỞ DỮ LIỆU ‘MÁY TÍNH’

Lược đồ cơ sở dữ liệu gồm 4 quan hệ:

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price);

Printer(model, color, type, price)

Quan hệ Product cho nhà sản xuất, số model, và kiểu(PC,laptop,hoặc printer) của các sản phẩm khác nhau. Để tiện lợi chúng ta giả thiết rằng số model là duy nhất trên tất cả các nhà sản xuất và kiểu sản phẩm. (giả thiết như vậy là không thực tiễn và một cơ sở dữ liệu thực phải chứa một mã của nhà sản xuất như là một phần của số model). Quan hệ PC có các thuộc tính model, speed (tốc độ của bộ xử lý, tính bằng megahertz), dung lượng RAM (tính bằng megabyte), dung lượng đĩa cứng (tính bằng gigabyte), tốc độ và kiểu của đĩa có thể di chuyển được (CD hoặc DVD) và giá. Quan hệ Laptop cũng tương tự, chỉ có thuộc tính mới là Screen, cho kích thước màn hình (tính bằng inch). Quan hệ Printer có các thuộc tính model, color (máy in màu hay đen trắng, có giá trị lôgic), type (kiểu xử lý: laze, in phun hay bọt) và price (giá tính bằng dolar)

PC

| model | speed | ram | hd | rd     | price |
|-------|-------|-----|----|--------|-------|
| 1001  | 700   | 64  | 10 | 48xCD  | 799   |
| 1002  | 1500  | 128 | 60 | 12xDVD | 2499  |
| 1003  | 866   | 128 | 20 | 8xDVD  | 1999  |
| 1004  | 866   | 64  | 10 | 12xDVD | 999   |
| 1005  | 1000  | 128 | 20 | 12xDVD | 1499  |
| 1006  | 1300  | 256 | 40 | 16xDVD | 2119  |
| 1007  | 1400  | 128 | 80 | 12xDVD | 2299  |
| 1008  | 700   | 64  | 30 | 24xCD  | 999   |
| 1009  | 1200  | 128 | 80 | 16xDVD | 1699  |
| 1010  | 750   | 64  | 30 | 40xCD  | 699   |
| 1011  | 1100  | 128 | 60 | 16xDVD | 1299  |
| 1012  | 350   | 64  | 7  | 48xCD  | 799   |
| 1013  | 733   | 256 | 60 | 12xDVD | 2499  |

### PRODUCT

| maker | model | type    |
|-------|-------|---------|
| A     | 1001  | pc      |
| A     | 1002  | pc      |
| A     | 1003  | pc      |
| A     | 2004  | laptop  |
| A     | 2005  | laptop  |
| A     | 2006  | laptop  |
| B     | 1004  | pc      |
| B     | 1005  | pc      |
| B     | 1006  | pc      |
| B     | 2001  | laptop  |
| B     | 2002  | laptop  |
| B     | 2003  | laptop  |
| C     | 1007  | pc      |
| C     | 1008  | pc      |
| C     | 2008  | laptop  |
| C     | 2009  | laptop  |
| C     | 3002  | printer |
| C     | 3003  | printer |
| C     | 3006  | printer |
| D     | 1009  | pc      |
| D     | 1010  | pc      |
| D     | 1011  | pc      |
| D     | 2007  | laptop  |
| E     | 1012  | pc      |
| E     | 1013  | pc      |
| E     | 2010  | laptop  |
| F     | 3001  | printer |
| F     | 3004  | printer |
| G     | 3005  | printer |
| H     | 3007  | printer |

### LAPTOP

| model | speed | ram | hd | screen | price |
|-------|-------|-----|----|--------|-------|
| 2001  | 700   | 64  | 5  | 12.1   | 1448  |
| 2002  | 800   | 96  | 10 | 15.1   | 2584  |
| 2003  | 850   | 64  | 10 | 15.1   | 2738  |
| 2004  | 550   | 32  | 6  | 12.1   | 999   |
| 2005  | 600   | 64  | 5  | 12.1   | 2399  |
| 2006  | 800   | 96  | 20 | 15.7   | 2999  |
| 2007  | 850   | 128 | 20 | 15.0   | 3099  |
| 2008  | 650   | 64  | 10 | 12.1   | 1249  |
| 2009  | 750   | 256 | 20 | 15.1   | 2599  |
| 2010  | 366   | 64  | 10 | 12.1   | 1499  |

### PRINTER

| model | color | type    | price |  |  |
|-------|-------|---------|-------|--|--|
| 3001  | true  | ink-jet | 231   |  |  |
| 3002  | true  | ink-jet | 267   |  |  |
| 3003  | false | lazer   | 390   |  |  |
| 3004  | true  | ink-jet | 439   |  |  |
| 3005  | true  | bubble  | 200   |  |  |
| 3006  | true  | lazer   | 1999  |  |  |
| 3007  | false | lazer   | 350   |  |  |



## TÀI LIỆU THAM KHẢO

- 1) C.J.Date, Hug Darwen, **A guide to the SQL standard**, *Addison-Wesley Publishing company, 1993*.
- 2) Hector Garcia-Molina, Jeffrey D.Ulman, Jennifer Widom, **Database Systems: The Complete Book** (Chapters: 6,7,8), *Prentice Hall, 2002* .
- 3) Peter Gultzan, Trudy Pelzer, **Optimzing SQL**, *R&D Publication, Inc,1994*.
- 4) Christian Maree. Guy Ledant, **SQL2. Initiation Programmation**, *Armand Colin, Paris 1994*.