

NGUYỄN TRUNG TRỰC

CƠ SỞ DỮ LIỆU PHÂN BỐ



NHÀ XUẤT BẢN
ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Nguyễn Trung Trực

CƠ SỞ DỮ LIỆU PHÂN BỐ

(Tái bản lần thứ hai)

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA
TP HỒ CHÍ MINH - 2013

MỤC LỤC

LỜI NÓI ĐẦU	5
Chương 1 TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU PHÂN BỐ	7
1.1 Định nghĩa cơ sở dữ liệu phân bố	7
1.2 Các đặc điểm của CSDL phân bố so với CSDL tập trung	12
1.3 Tại sao phải sử dụng CSDL phân bố?	21
1.4 Các hệ quản trị CSDL phân bố	23
Chương 2 TỔNG QUAN VỀ CSDL VÀ MẠNG MÁY TÍNH	29
2.1 Tổng quan về CSDL	29
2.2 Tổng quan về mạng máy tính	63
Chương 3 CÁC MỨC TRONG SUỐT PHÂN BỐ	72
3.1 Kiến trúc tham khảo dùng cho các CSDL phân bố	73
3.2 Các loại phân mảnh dữ liệu	78
3.3 Tính trong suốt phân mảnh dùng cho các ứng dụng chỉ đọc	87
3.4 Tính trong suốt phân mảnh dùng cho các ứng dụng cập nhật	97
3.5 Các tác vụ cơ bản truy xuất CSDL phân bố	104
3.6 Các ràng buộc toàn vẹn trong CSDL phân bố	109
Chương 4 THIẾT KẾ CƠ SỞ DỮ LIỆU PHÂN BỐ	112
4.1 Các bước thiết kế CSDL phân bố	113
4.2 Thiết kế phân mảnh CSDL	123
4.3 Thiết kế định vị CSDL	161
Chương 5 BIẾN ĐỔI CÁC TRUY VẤN TOÀN CỤC THÀNH CÁC TRUY VẤN MẢNH	171
5.1 Các phép biến đổi tương đương dùng cho các truy vấn	172
5.2 Biến đổi các truy vấn toàn cục thành các truy vấn mảnh	180

5.3 Gom nhóm phân bố và định trị hàm kết hợp	195
5.4 Các truy vấn có tham số	201
<i>Chương 6 TỐI ƯU HÓA CÁC CHIẾN LƯỢC TRUY XUẤT</i>	206
6.1 Một cơ cấu cho tối ưu hóa truy vấn	207
6.2 Các truy vấn kết	224
6.3 Các truy vấn tổng quát	248
Thuật ngữ Việt - Anh	252
Từ viết tắt	259
<i>TÀI LIỆU THAM KHẢO</i>	260

LỜI NÓI ĐẦU

Hiện nay, công nghệ truyền thông và mạng Internet ngày càng được phát triển mạnh mẽ và cũng có rất nhiều hệ thống ứng dụng chạy trên các mạng cục bộ và mạng diện rộng. Cơ sở dữ liệu phân bố đã được sử dụng trong các hệ thống ứng dụng này, dữ liệu được lưu trữ tại nhiều nơi khác nhau của mạng máy tính và các chương trình ứng dụng truy xuất dữ liệu tại nhiều nơi. Công nghệ hệ thống cơ sở dữ liệu phân bố là một trong những phát triển của lãnh vực cơ sở dữ liệu và các hệ quản trị cơ sở dữ liệu phân bố đã được thương mại hóa như Oracle, Microsoft SQL Server....

CƠ SỞ DỮ LIỆU PHÂN BỐ được biên soạn theo đề cương chương trình môn học **Thiết kế cơ sở dữ liệu phân bố** - Khoa Công nghệ thông tin, Trường Đại học Bách khoa - Đại học Quốc gia TP HCM, nhằm mục đích giới thiệu về lý thuyết, các giải thuật và các phương pháp được sử dụng trong các hệ quản trị cơ sở dữ liệu phân bố, các nguyên lý thiết kế cơ sở dữ liệu phân bố và các chiến lược truy xuất cơ sở dữ liệu phân bố. Quyển sách này bao gồm sáu chương.

Chương 1, giới thiệu tổng quan về cơ sở dữ liệu phân bố: định nghĩa, các đặc điểm của cơ sở dữ liệu phân bố, các hệ quản trị cơ sở dữ liệu phân bố.

Chương 2, giới thiệu sơ lược về cơ sở dữ liệu quan hệ và mạng máy tính. Đây là một số kiến thức cần thiết cho các chương kế tiếp.

Chương 3, trình bày các mức trong suốt của cơ sở dữ liệu phân bố và các chương trình ứng dụng truy xuất và cập nhật dữ liệu như thế nào tại các mức trong suốt này.

Chương 4, trình bày các bài toán thiết kế cơ sở dữ liệu phân bố, các nguyên lý và các giải thuật dùng để thiết kế một cơ sở dữ liệu phân bố.

Chương 5, trình bày cách thức biến đổi việc truy xuất dữ liệu toàn cục (tập trung) thành việc truy xuất dữ liệu được phân mảnh (phân bố).

Chương 6, trình bày các nguyên lý tối ưu hóa chiến lược truy xuất dữ liệu phân bố.

Mặc dù có nhiều cố gắng, nhưng quyển sách này không thể tránh khỏi những thiếu sót, tôi rất mong nhận được ý kiến đóng góp của các bạn đồng nghiệp và độc giả.

Địa chỉ liên hệ: Khoa Công nghệ Thông tin - Trường Đại học Bách khoa - Đại học Quốc gia Tp HCM 268 Lý Thường Kiệt, Quận 10.

ĐT: (08) 8.658.689 - (08)8.654.526

Tác giả

Nguyễn Trung Trực

TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU PHÂN BỐ

Ngày nay, cơ sở dữ liệu (CSDL) phân bố đã trở thành một lãnh vực quan trọng của xử lý thông tin và tầm quan trọng của nó ngày càng tăng nhanh. Có hai lý do về mặt công nghệ và về mặt tổ chức để đi theo hướng này: các CSDL phân bố loại trừ nhiều thiếu sót của các CSDL tập trung (*centralized database*) và thích hợp một cách tự nhiên với các cấu trúc không tập trung (*decentralized structure*) của nhiều tổ chức (*organization*).

1.1 ĐỊNH NGHĨA CƠ SỞ DỮ LIỆU PHÂN BỐ

Một định nghĩa tiêu biểu của CSDL phân bố (nhưng hơi mơ hồ) như sau: *Một cơ sở dữ liệu phân bố là sự tập hợp dữ liệu mà về mặt luận lý chúng thuộc cùng một hệ thống nhưng được trải rộng ở nhiều nơi (site) của một mạng máy tính.*

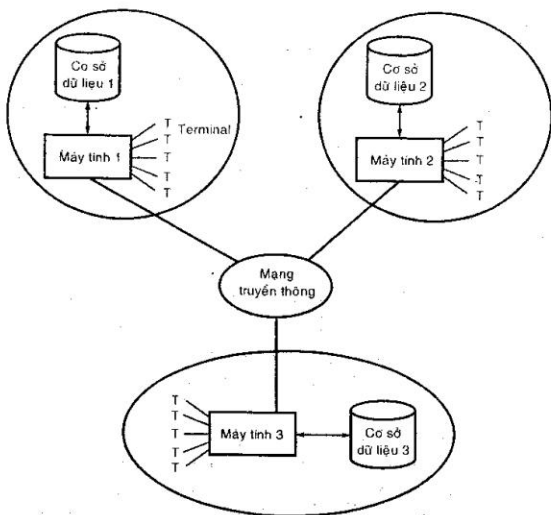
Định nghĩa này nhấn mạnh hai khía cạnh quan trọng như nhau của một CSDL phân bố là:

1- *Sự phân bố (distribution)*: nghĩa là dữ liệu không thường trú ở cùng một nơi (hiểu theo nghĩa bộ xử lý - *processor*), điều mà chúng ta có thể phân biệt một CSDL phân bố với một CSDL tập trung, đơn lẻ.

2- *Sự tương quan luận lý (logical correlation)*: nghĩa là dữ liệu có những tính chất liên hệ mật thiết với nhau, điều mà chúng ta có thể phân biệt một CSDL phân bố với một tập hợp các CSDL cục bộ (*local database*) hoặc các tập tin thường trú ở những nơi khác nhau của một mạng máy tính.

Một vấn đề xuất phát từ định nghĩa trên là các tính chất, sự phân bố, sự tương quan luận lý được định nghĩa một cách quá mơ hồ để luôn luôn phân biệt được giữa những trường hợp thực sự là những CSDL phân bố với những trường hợp không là CSDL phân bố. Để có một định nghĩa rõ ràng hơn, chúng ta xét một số ví dụ sau.

Ví dụ 1.1 Xét một ngân hàng có ba chi nhánh ở các vị trí địa lý khác nhau. Tại mỗi chi nhánh có máy tính kiểm soát các máy rút tiền của chi nhánh, và một CSDL tài khoản (*account database*) của chi nhánh (hình 1.1). Mỗi máy tính với một CSDL tài khoản cục bộ ở một chi nhánh tạo thành một *nơi* (*site*) của CSDL phân bố; các máy tính được nối với nhau thông qua một mạng máy tính truyền thông (*communication network*). Khi hoạt động bình thường, các ứng dụng được chạy tại các thiết bị đầu cuối của chi nhánh chỉ cần truy xuất CSDL của chi nhánh đó. Những ứng dụng này được thực hiện hoàn tất bởi máy tính của chi nhánh, nơi mà chúng được chạy, và được gọi là các *ứng dụng cục bộ* (*local application*). Một ví dụ của một ứng dụng cục bộ là ứng dụng ghi nợ hoặc ghi có được thực hiện trên một tài khoản được lưu trữ trong cùng một chi nhánh mà tại đây chạy ứng dụng này.



Hình 1.1 Một CSDL phân bố trên một mạng phân bố địa lý

Nếu chúng ta thử áp dụng định nghĩa của CSDL phân bố cho trường hợp này, thì chúng ta nhận thấy rằng khó nói là có tính tương quan luận lý hay không.

Có phải các chi nhánh chỉ chứa thông tin về các tài khoản của cùng một ngân hàng là đủ?

Có nên xem điều này là một ví dụ của một CSDL phân bố hoặc một tập hợp các CSDL cục bộ?

Để trả lời các câu hỏi này, chúng ta tập trung vào cái gì làm cho một tập hợp các CSDL cục bộ khác với một CSDL phân bố có cùng dữ liệu. Về quan điểm công nghệ (*technological viewpoint*), có một khía cạnh thực sự quan trọng là sự tồn tại của một vài ứng dụng truy xuất dữ liệu tại hai chi nhánh trở lên. Các ứng dụng này được gọi là *ứng dụng toàn cục* (*global application*) hoặc *ứng dụng phân bố* (*distributed application*). Sự tồn tại của các ứng dụng toàn cục là đặc điểm phân biệt của các CSDL phân bố so với một tập hợp các CSDL cục bộ.

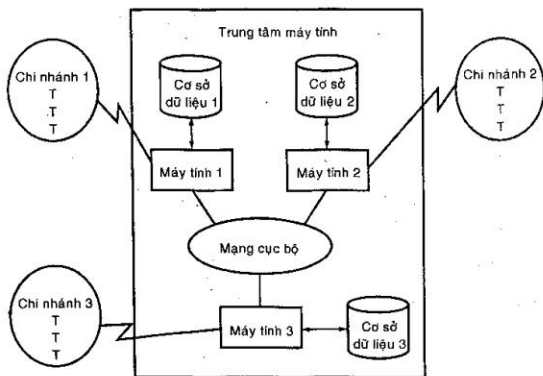
Một ứng dụng toàn cục tiêu biểu là một sự chuyển quỹ từ một tài khoản của một chi nhánh này vào một tài khoản của một chi nhánh khác. Ứng dụng này cập nhật các CSDL tại hai chi nhánh khác nhau. Lưu ý rằng ứng dụng này thực hiện hai cập nhật cục bộ tại hai chi nhánh riêng biệt (ghi nợ và ghi có), bởi vì cũng cần phải bảo đảm là cả hai cập nhật này đều được thực hiện hoặc đều không được thực hiện. Do đó, việc bảo đảm yêu cầu này của các ứng dụng toàn cục là một công việc khó khăn.

Trong ví dụ 1.1, các máy tính được đặt tại các vị trí địa lý khác nhau; tuy nhiên, các CSDL phân bố cũng có thể được xây dựng trên các mạng cục bộ.

Ví dụ 1.2 Xét cùng một ngân hàng của ví dụ trước, với cùng các ứng dụng, tuy nhiên với một cấu hình hệ thống được chỉ ra trong hình 1.2. Cùng các bộ xử lý với các CSDL của chúng đã được di chuyển từ các chi nhánh về một tòa nhà chung và bây giờ được kết nối với nhau bởi một mạng cục bộ băng thông cao (*high-bandwidth local network*). Các máy rút tiền của các chi nhánh được kết nối với các máy tính tương ứng bởi các tuyến điện thoại. Mỗi bộ xử lý và CSDL của nó tạo thành một nơi của mạng máy tính cục bộ.

Chúng ta thấy rằng cấu trúc vật lý của các kết nối đã được thay đổi so với ví dụ 1.1; tuy nhiên các khía cạnh đặc trưng của kiến trúc vẫn không thay đổi. Đặc biệt, cùng các máy tính thực hiện cùng các ứng dụng truy xuất trên cùng các CSDL. Một ứng dụng là cục bộ trong ví dụ trước thì vẫn là cục bộ, miễn là tính cục bộ (*locality*) không được định nghĩa đối với sự phân bố địa lý của các máy tính thực hiện ứng dụng này, nhưng chỉ đối với một máy tính và CSDL riêng của nó.

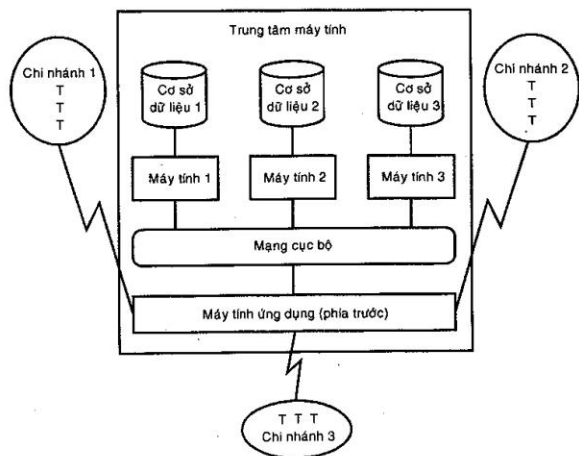
Nếu có các ứng dụng toàn cục, thì ví dụ này được xem là một CSDL phân bố, vì hầu hết các điểm đặc trưng của ví dụ trước vẫn còn hợp lệ. Tuy nhiên, CSDL phân bố được thực hiện trên một mạng cục bộ thay vì trên một mạng địa lý, với độ tin cậy (*reliability*) và thông lượng (*throughput*) của nó cao hơn, trong một số trường hợp sẽ thay đổi các giải pháp cho trước đối với một số vấn đề. Chúng ta xét một ví dụ của một hệ thống mà chúng ta không xem là một CSDL phân bố.



Hình 1.2 Một CSDL phân bố trên một mạng cục bộ

Ví dụ 1.3 Xét cùng một ngân hàng của ví dụ trước nhưng với cấu hình hệ thống được chỉ ra trong hình 1.3. Dữ liệu của các chi nhánh khác nhau được phân bố trên ba máy tính phía sau (*backend computer*), mà chúng thực hiện các chức năng quản trị CSDL. Chương trình ứng dụng được thực hiện bởi một máy tính khác, mà máy tính này yêu cầu các dịch vụ truy xuất CSDL tại các máy tính phía sau khi cần thiết.

Lý do mà hệ thống này không được xem là một CSDL phân bố, bởi vì mặc dù dữ liệu được phân bố về mặt vật lý trên các bộ xử lý khác nhau, nhưng sự phân bố của chúng là không thích hợp trên quan điểm ứng dụng. Điều mà chúng ta không có ở đây chính là sự tồn tại của các ứng dụng cục bộ, bởi vì không có một máy tính nào mà chính nó có khả năng thực hiện một ứng dụng.



Hình 1.3 Một hệ thống đa xử lý (*multiprocessor system*)

Bây giờ chúng ta có thể tổng kết các khảo sát mà chúng ta đã có được từ các ví dụ ở trên thành một định nghĩa chấp nhận được như sau:

Một cơ sở dữ liệu phân bố, là sự tập hợp dữ liệu được phân bố trên các máy tính khác nhau của một mạng máy tính. Mỗi nơi của mạng máy tính có khả năng xử lý tự trị (autonomous processing capability) và có thể thực hiện các ứng dụng cục bộ. Mỗi nơi cũng tham gia vào sự thực hiện của ít nhất một ứng dụng toàn cục, mà nơi này yêu cầu truy xuất dữ liệu ở nhiều nơi bằng cách dùng hệ thống truyền thông con (communication subsystem).

Định nghĩa này nhấn mạnh ba khía cạnh quan trọng của một CSDL phân bố, đó là:

- 1- *Sự phân bố dữ liệu*: dữ liệu phải được phân bố ở nhiều nơi.
- 2- *Ứng dụng cục bộ*: mỗi nơi phải thực hiện ít nhất một ứng dụng cục bộ.
- 3- *Ứng dụng toàn cục*: mỗi nơi phải tham gia vào sự thực hiện của ít nhất một ứng dụng toàn cục.

Các vấn đề công nghệ quan trọng nhất của CSDL phân bố có được từ việc khảo sát chúng là kết quả của sự hợp tác giữa các nơi tự trị; đặc điểm này được nhấn mạnh ở định nghĩa trên.

1.2 CÁC ĐẶC ĐIỂM CỦA CSDL PHÂN BỐ SO VỚI CSDL TẬP TRUNG

Các CSDL phân bố không đơn giản là sự thực hiện phân bố của các CSDL tập trung, bởi vì chúng cho phép thiết kế các hệ thống biểu thị các đặc điểm khác so với các hệ thống tập trung, truyền thống. Do đó, chúng ta sẽ xét các đặc điểm tiêu biểu của các CSDL truyền thống, và so sánh chúng với các đặc điểm tương ứng của các CSDL phân bố. Các đặc điểm đặc trưng cho cách tiếp cận CSDL truyền thống là:

- Điều khiển tập trung (*centralized control*)
- Độc lập dữ liệu (*data independence*)
- Giảm dư thừa (*reduction of redundancy*)
- Các cấu trúc vật lý phức tạp dùng để truy xuất hiệu quả
- Tính toàn vẹn (*integrity*)
- Phục hồi (*recovery*)
- Điều khiển đồng thời (*concurrency control*)
- Tính riêng biệt (*privacy*) và tính bảo mật (*security*).

1- Điều khiển tập trung

Khả năng cung cấp điều khiển tập trung trên các tài nguyên thông tin (*information resources*) của toàn bộ một xí nghiệp (hoặc một tổ chức) được xem là một trong những động cơ thúc đẩy tốt nhất cho việc giới thiệu các CSDL; chúng được phát triển như là một cuộc cách mạng của các hệ thống thông tin mà trong đó mỗi ứng dụng có các tập tin riêng biệt của nó. Chức năng cơ bản của *người quản trị CSDL (DBA – DataBase Administrator)* là bảo đảm sự an toàn của dữ liệu; bản thân dữ liệu đã được công nhận là sự đầu tư quan trọng của xí nghiệp mà dữ liệu cần phải được tập trung.

Trong các CSDL phân bố, ý tưởng điều khiển tập trung không được nhấn mạnh. Điều này cũng tùy thuộc vào kiến trúc, như chúng ta có thể thấy trong các ví dụ 1.1 và 1.2. Ví dụ 1.2 thích hợp với điều khiển tập trung hơn so với ví dụ 1.1.

Nói chung, trong các CSDL phân bố, chúng ta có thể nhận biết một cấu trúc điều khiển phân cấp (*hierarchical control structure*) gồm có *người quản trị CSDL toàn cục (global database administrator)*, mà người này có trách nhiệm chính về toàn bộ CSDL, và những *người quản trị CSDL cục bộ (local database administrator)*, mà họ có trách nhiệm về các CSDL cục bộ của họ.

Tuy nhiên, cần phải nhấn mạnh là những người quản trị CSDL cục bộ có thể có một mức độ tự trị cao, cao đến mức mà hoàn toàn không cần có người quản trị CSDL toàn cục, và sự phối hợp giữa các nơi (*intersite coordination*) được thực hiện bởi chính những người quản trị cục bộ. Đặc tính này thường được gọi là *tính tự trị vị trí (site autonomy)*.

Các CSDL phân bố có thể khác nhau rất nhiều về mức độ tự trị vị trí: từ tính tự trị vị trí hoàn toàn (*complete site autonomy*) không có người quản trị CSDL tập trung, đến điều khiển tập trung hầu như hoàn toàn.

2- Độc lập dữ liệu

Độc lập dữ liệu cũng được xem là một trong những động cơ thúc đẩy chính cho việc giới thiệu cách tiếp cận CSDL. Về cơ bản, độc lập dữ liệu có nghĩa là tổ chức hiện tại của dữ liệu là trong suốt đối với người lập trình ứng dụng. Các chương trình được viết dựa trên một *cái nhìn ý niệm (conceptual view)* về dữ liệu, được gọi là *lược đồ ý niệm (conceptual schema)*.

Ưu điểm chính của độc lập dữ liệu, là các chương trình không bị ảnh hưởng bởi những thay đổi về tổ chức vật lý của dữ liệu.

Trong các CSDL phân bố, độc lập dữ liệu có cùng tầm quan trọng như trong các CSDL truyền thống. Tuy nhiên, một khía cạnh mới được đưa vào trong khái niệm thông thường của độc lập dữ liệu, được gọi là *tính trong suốt phân bố* (*distribution transparency*). Nhờ tính trong suốt phân bố mà các chương trình có thể được viết giống như khi CSDL không được phân bố. Do đó, tính đúng đắn của các chương trình không bị ảnh hưởng bởi việc di chuyển dữ liệu từ một nơi này đến một nơi khác, nhưng tốc độ thực hiện thì có bị ảnh hưởng.

Độc lập dữ liệu đã được cung cấp trong các CSDL truyền thống thông qua một kiến trúc nhiều mức (*multilevel architecture*), có các mô tả khác nhau về dữ liệu và các ánh xạ giữa chúng; các khái niệm của lược đồ ý niệm, lược đồ lưu trữ (*storage schema*) đã được phát triển nhằm mục đích này. Một cách tương tự, tính trong suốt phân bố có được trong các CSDL phân bố bằng cách nêu ra các mức và những lược đồ mới.

3- Giảm dư thừa

Trong các CSDL truyền thống, dư thừa dữ liệu được giảm càng nhiều càng tốt vì hai lý do:

Thứ nhất, những sự không nhất quán (*inconsistency*) giữa nhiều bản sao của cùng dữ liệu luận lý sẽ tự động được tránh khỏi bằng cách chỉ có một bản sao.

Thứ hai, vùng nhớ lưu trữ được tiết kiệm bằng cách loại bỏ dư thừa.

Giảm dư thừa có được bằng cách dùng chung dữ liệu (*data sharing*), nghĩa là cho phép nhiều ứng dụng truy xuất đến cùng các tập tin và các mẫu tin.

Tuy nhiên, trong các CSDL phân bố, có nhiều lý do để xem dư thừa dữ liệu là một đặc điểm cần thiết:

Thứ nhất, tính cục bộ (*locality*) của các ứng dụng có thể được gia tăng nếu dữ liệu được nhân bản tại tất cả các nơi mà ứng dụng cần dữ liệu này;

Thứ hai, tính sẵn sàng (*availability*) của hệ thống có thể được gia tăng, bởi vì một nơi bị hỏng sẽ không làm ngưng thực hiện của các ứng dụng tại các nơi khác nếu dữ liệu được nhân bản.

Nói chung, các lý do về sự dư thừa này đối với môi trường truyền thống vẫn còn có giá trị, và do đó muốn đánh giá mức độ dư thừa tối ưu thì cần phải đánh giá về sự thỏa hiệp phức tạp hơn. Lợi ích của việc nhân bản một mục dữ liệu sẽ tăng theo tỉ số giữa các lần truy xuất lấy dữ liệu (*retrieval access*), với các lần truy xuất cập nhật dữ liệu (*update access*), được thực hiện bởi các ứng dụng truy xuất đến mục dữ liệu này. Lợi ích của nhân bản dữ liệu được gia tăng, bởi vì nếu chúng ta có nhiều bản sao của một mục dữ liệu thì việc lấy dữ liệu có thể được thực hiện trên bất kỳ bản sao nào, trong khi việc cập nhật phải được thực hiện một cách nhất quán trên tất cả các bản sao. Do đó, nhân bản dữ liệu cần phải được xem xét kỹ lưỡng dựa vào hai loại ứng dụng cơ bản, đó là *ứng dụng chỉ đọc* và *ứng dụng cập nhật*. Nhân bản dữ liệu giúp cho các ứng dụng chỉ đọc được thực hiện nhanh hơn, nhưng nó làm cho các ứng dụng cập nhật bị thực hiện lâu hơn vì phải cập nhật dữ liệu tại các nơi được nhân bản.

Như vậy, nhân bản dữ liệu sẽ là một ưu điểm nếu hệ thống có rất nhiều ứng dụng chỉ đọc và có rất ít ứng dụng cập nhật. Trong trường hợp ngược lại thì sự nhân bản dữ liệu là một nhược điểm.

4- Các cấu trúc vật lý phức tạp và truy xuất hiệu quả

Các cấu trúc truy xuất phức tạp, chẳng hạn như các chỉ mục thứ cấp (*secondary index*), các chuỗi kết nối giữa các tập tin (*interfile chain*), ..., là một khía cạnh chính của các CSDL truyền thống. Sự hỗ trợ cho các cấu trúc này là một phần quan trọng nhất của các hệ quản trị CSDL (*DBMS - DataBase Management System*). Lý do của việc cung cấp các cấu trúc truy xuất phức tạp là để truy xuất dữ liệu hiệu quả.

Trong các CSDL phân bố, các cấu trúc truy xuất phức tạp không phải là một công cụ đúng để truy xuất hiệu quả. Do đó, truy xuất hiệu quả vẫn là một vấn đề chính trong các CSDL phân bố, các cấu trúc vật lý không thích hợp về mặt công nghệ. Truy xuất hiệu quả đến một CSDL phân bố không thể được cung cấp bằng cách sử

dụng các cấu trúc vật lý giữa các nơi, bởi vì rất khó xây dựng và bảo trì các cấu trúc như vậy, và bởi vì nó không thuận lợi cho liên kết ở mức mẫu tin trong các CSDL phân bố. Chúng ta trình bày điều này bằng một ví dụ.

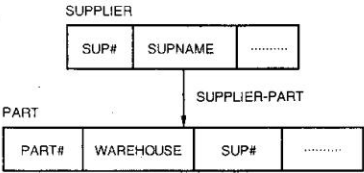
Ví dụ 1.4 Xét lược đồ CSDL tựa-Codasyl được chỉ ra trong hình 1.4. Có hai loại mẫu tin, SUPPLIER và PART, và một kiểu tập hợp SUPPLIER-PART kết nối các mẫu tin của nhà cung cấp với các mẫu tin mặt hàng được cung cấp bởi các nhà cung cấp này. Ứng dụng “tìm tất cả các mẫu tin của PART được cung cấp bởi nhà cung cấp SI” được viết bằng Codasyl-DML như trong hình 1.4b.

Bây giờ, chúng ta giả sử CSDL trên được phân bố tại ba nơi của một mạng máy tính và được chỉ ra trong hình 1.4c:

- Tập tin của nhà cung cấp được đặt ở nơi 1 (quản trị trung tâm - *central administration*)
- Tập tin mặt hàng được phân chia thành hai tập tin con khác nhau được đặt tại hai nơi 2 và 3 (các kho - *warehouse*).

Chúng ta giả sử thêm rằng hệ thống Codasyl được thực hiện phân bố, sao cho chúng ta có thể chạy cùng một chương trình (thông kết) của hình 1.4b trên CSDL phân bố. Giả sử ứng dụng được chạy tại nơi 1. Rõ ràng, hệ thống phải truy xuất một mẫu tin PART từ xa cho mỗi lần lặp của vòng lặp “repeat until”. Do đó, đối với mỗi truy xuất đến một mẫu tin thì không chỉ chính mẫu tin này được truyền đi, mà còn có nhiều thông điệp cũng được trao đổi với nhau.

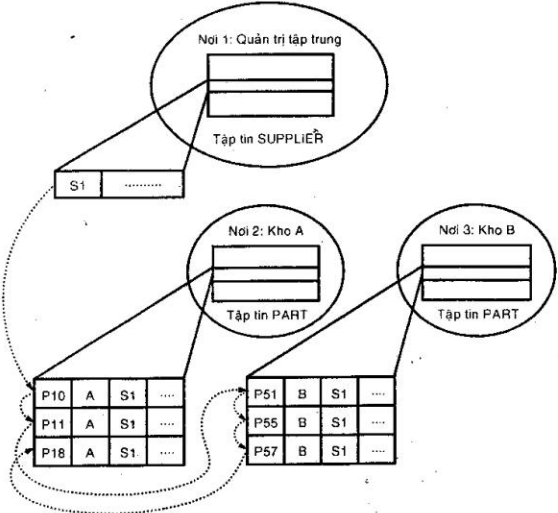
Thực hiện hiệu quả hơn cho ứng dụng này bao gồm việc gom nhóm càng nhiều càng tốt tất cả các truy xuất từ xa (*remote access*) và được chỉ ra trong hình 1.5.



a) Một lược đồ cơ sở dữ liệu Codasyl

Find SUPPLIER record with SUP# = S1;
Repeat until "no more members in set"
Find next PART record in SUPPLIER-PART set;
Output PART record;

b) Một chương trình tựa Codasyl-DBMS để tìm kiếm các mặt hàng được cung cấp bởi nhà cung cấp S1



c) Sự phân bố của tập SUPPLIER-PART

Hình 1.4 Một cơ sở dữ liệu phân bố tựa-Codasyl

So sánh các chương trình trong hình 1.4b và 1.5:

- Trong chương trình đầu, mệnh đề "*find next*" đòi hỏi các truy xuất trên cơ sở từng mẫu tin (*record-by-record*);

- Trong chương trình sau, mệnh đề "*find all*" gom nhóm tất cả các truy xuất được thực hiện trên cùng một nơi.

Thủ tục được chỉ ra trong hình 1.5 bao gồm hai loại tác vụ:

- Tác vụ thực hiện của các chương trình là cục bộ tại những nơi đơn lẻ.

- Tác vụ truyền các tập tin giữa các nơi.

Một thủ tục giống như ở trên được gọi là một *kế hoạch truy xuất phân bố* (*distributed access plan*).

1- At site 1

Send sites 2 and 3 the supplier number SN

2- At sites 2 and 3

Execute in parallel, upon receipt of the supplier number, the following program:

Find all PARTS records having

SUP# = SN;

Send result to site 1.

3- At site 1

Merge results from sites 2 and 3;

Output the result.

Hình 1.5 Ví dụ về kế hoạch truy xuất

Một kế hoạch truy xuất phân bố có thể được viết bởi người lập trình, hoặc được phát sinh tự động bởi một bộ tối ưu hóa (*optimizer*). Viết một kế hoạch truy xuất phân bố tương tự như lập trình thông kết (*navigational programming*) trong các CSDL tập trung, theo nghĩa là người lập trình phải chỉ định CSDL được truy xuất như thế nào. Tuy nhiên, sự thông kết giữa các nơi (*intersite navigation*) nên được thực hiện ở mức độ của các nhóm mẫu tin, trong khi sự thông kết thường dùng một lần một mẫu tin có thể được thực hiện cho việc xử lý cục bộ tại một nơi. Do đó, một ngôn ngữ thông kết (*navigational language*) là kém thích hợp hơn so với một ngôn ngữ hướng tập hợp (*set-oriented language*), phi thủ tục (*nonprocedural*) dùng để xây dựng các kế hoạch truy xuất.

Nhiều vấn đề được giải quyết trong thiết kế một bộ tối ưu hóa mà nó phát sinh tự động một kế hoạch truy xuất, chẳng hạn như kế hoạch được chỉ ra trong hình 1.5. Các vấn đề này được phân chia thành hai loại: *tối ưu hóa toàn cục* (*global optimization*) và *tối ưu hóa cục bộ* (*local optimization*).

Tối ưu hóa toàn cục bao gồm việc xác định dữ liệu nào phải được truy xuất tại các nơi nào, và từ đó các tập tin dữ liệu nào phải được truyền giữa các nơi. Thông số chính của tối ưu hóa là chi phí truyền thông (*communication cost*), mặc dù chi phí truy xuất các CSDL cục bộ cũng nên tính đến trong một số trường hợp. Tầm quan trọng tương đối của các yếu tố này tùy thuộc vào tỉ số giữa các chi phí truyền thông và các chi phí truy xuất đĩa (*disk access cost*), mà các chi phí này lại tùy thuộc vào loại mạng truyền thông.

Tối ưu hóa cục bộ bao gồm việc quyết định truy xuất CSDL cục bộ được thực hiện như thế nào tại mỗi nơi; các vấn đề tối ưu hóa cục bộ là tiêu biểu của các CSDL không phân bố, truyền thống và sẽ không được xét đến nữa trong quyển sách này.

5- Tính toàn vẹn, phục hồi, điều khiển đồng thời

Trong các CSDL, tính toàn vẹn, phục hồi và điều khiển đồng thời, mặc dù có các vấn đề khác nhau, nhưng chúng có liên quan chặt chẽ với nhau. Trong một phạm vi rộng, giải pháp của các vấn đề này bao gồm việc cung cấp các giao dịch (*transaction*).

Một *giao dịch* là một *đơn vị thực hiện nguyên tố*, nghĩa là một chuỗi các tác vụ hoặc tất cả đều được thực hiện hoặc tất cả đều không được thực hiện.

Ứng dụng *chuyển quỹ*, được cho trong hình 1.1, là một ứng dụng toàn cục và phải là một đơn vị nguyên tố: cả hai phần ghi nợ và phần ghi có đều được thực hiện hoặc đều không được thực hiện, không thể chấp nhận chỉ thực hiện một trong hai phần. Do đó, ứng dụng chuyển quỹ cũng là một *giao dịch toàn cục* (*global transaction*).

Rõ ràng, trong các CSDL phân bố, các vấn đề của tính nguyên tố của giao dịch có một điểm riêng biệt: hệ thống sẽ chạy như thế nào nếu nơi *ghi nợ* hoạt động và nơi *ghi có* không hoạt động khi

chuyển quỹ được yêu cầu? Giao dịch nên bị hủy bỏ (hủy bỏ tất cả các tác vụ đã được thực hiện khi có một nơi bị hỏng) hoặc nên có một hệ thống thông minh cố gắng thực hiện chuyển quỹ đúng đắn ngay cả khi cả hai nơi không bao giờ hoạt động đồng thời? Tất nhiên, người sử dụng sẽ ít bị ảnh hưởng bởi các hư hỏng này nếu áp dụng cách tiếp cận sau cùng.

Rõ ràng, các giao dịch nguyên tố là cách thức để có được tính toàn vẹn của CSDL, bởi vì chúng bảo đảm tất cả các tác vụ để chuyển CSDL từ trạng thái nhất quán này sang trạng thái nhất quán khác, hoặc trở về trạng thái nhất quán ban đầu.

Có hai điều nguy hiểm cho tính nguyên tố của giao dịch là các hư hỏng và tính đồng thời:

- Các hư hỏng có thể làm cho hệ thống ngưng hoạt động khi đang thực hiện giao dịch giữa chừng, do đó sẽ vi phạm tính nguyên tố.

- Thực hiện đồng thời các giao dịch khác nhau có thể cho phép một giao dịch đang thực hiện sẽ xem xét đến một trạng thái không nhất quán nhất thời được tạo ra bởi một giao dịch khác.

Trong một phạm vi rộng, sự phục hồi giải quyết vấn đề lưu giữ tính nguyên tố của giao dịch khi xảy ra hư hỏng. Trong các CSDL phân bố, khía cạnh này là đặc biệt quan trọng, bởi vì một số nơi liên quan đến sự thực hiện của giao dịch có thể bị hư hỏng, được chỉ ra trong ví dụ trước.

Điều khiển đồng thời liên quan đến việc bảo đảm tính nguyên tố của giao dịch khi có sự thực hiện đồng thời các giao dịch. Vấn đề này có thể được xem là một vấn đề đồng bộ hóa tiêu biểu. Trong các CSDL phân bố, cũng như trong tất cả các hệ thống phân bố, vấn đề đồng bộ hóa khó hơn so với các hệ thống tập trung.

6. Tính riêng biệt và tính bảo mật

Trong các CSDL truyền thống có điều khiển tập trung, người quản trị CSDL có thể bảo đảm rằng chỉ có truy xuất dữ liệu có thẩm quyền được thực hiện. Tuy nhiên, lưu ý rằng trong cách tiếp cận CSDL tập trung, không có các thủ tục điều khiển đặc biệt, tính riêng biệt và tính bảo mật dễ bị vi phạm hơn so với các cách tiếp cận cũ dựa trên các tập tin riêng biệt.

Trong các CSDL phân bố, những người quản trị cục bộ chủ yếu đối phó với cùng vấn đề giống như những người quản trị CSDL trong CSDL truyền thống. Tuy nhiên, hai khía cạnh đặc biệt của các CSDL phân bố đáng được đề cập là: thứ nhất, trong một CSDL phân bố có mức độ tự trị vị trí rất cao, các chủ nhân của dữ liệu cục bộ cảm thấy cần phải được bảo vệ hơn và họ có thể thực hiện các bảo vệ của riêng họ thay vì phụ thuộc vào người quản trị CSDL trung tâm; thứ hai, nói chung các vấn đề bảo mật thực chất là ở bên trong các hệ thống phân bố, bởi vì các mạng truyền thông có thể tiêu biểu cho một điểm yếu về sự bảo vệ.

1.3 TẠI SAO PHẢI SỬ DỤNG CSDL PHÂN BỐ?

Có nhiều lý do tại sao phát triển CSDL phân bố. Các động cơ thúc đẩy chính là:

- Các lý do về tổ chức (*organizational*) và kinh tế (*economic*)
- Kết nối lẫn nhau (*interconnection*) của các CSDL hiện tại
- Sự lớn mạnh gia tăng (*incremental growth*)
- Giảm chi phí truyền thông (*communication overhead*)
- Các nghiên cứu về hiệu suất (*performance consideration*)
- Độ tin cậy (*reliability*) và tính sẵn sàng (*availability*)

1- Các lý do về tổ chức và kinh tế

Nhiều tổ chức không được tập trung hóa, và do đó cách tiếp cận CSDL phân bố thỏa mãn một cách tự nhiên hơn với cơ cấu tổ chức này. Các vấn đề của cơ cấu tổ chức phân bố và của hệ thống thông tin tương ứng là chủ đề của nhiều quyển sách và bài báo. Với những phát triển hiện nay của công nghệ máy tính, các động cơ giảm bớt chi phí để có các trung tâm máy tính tập trung lớn trở nên có vấn đề. Tuy nhiên, các động cơ thúc đẩy về tổ chức và kinh tế có thể là lý do quan trọng nhất để phát triển CSDL phân bố.

2- Sự kết nối lẫn nhau của các cơ sở dữ liệu hiện tại

Các CSDL phân bố là giải pháp tự nhiên khi có nhiều CSDL đã tồn tại trong một tổ chức và cần phải thực hiện nhiều ứng dụng toàn cục hơn. Trong trường hợp này, CSDL được tạo từ dưới lên từ các CSDL cục bộ đã tồn tại trước. Tuy nhiên, việc tái cấu trúc này sẽ cần sự nỗ lực ít hơn so với việc tạo mới hoàn toàn một CSDL tập trung.

3- Sự lớn mạnh gia tăng

Khi một tổ chức lớn mạnh lên do có thêm các đơn vị tổ chức tương đối độc lập (các chi nhánh mới, các kho mới, ...), thì cách tiếp cận CSDL phân bố hỗ trợ sự lớn mạnh tăng dần với một mức độ ảnh hưởng tối thiểu đến các đơn vị đã tồn tại. Với cách tiếp cận tập trung, qui mô ban đầu của hệ thống cần phải thận trọng để mở rộng trong tương lai, mà điều này cũng khó nhìn thấy trước và tốn kém nhiều cho việc thực hiện, hoặc sự lớn mạnh sẽ ảnh hưởng nhiều đến các ứng dụng mới và các ứng dụng đang tồn tại.

4- Giảm chi phí truyền thông

Trong một CSDL phân bố về mặt địa lý, chẳng hạn như CSDL của ví dụ 1.1, rõ ràng nhiều ứng dụng cục bộ làm giảm chi phí truyền thông so với CSDL tập trung. Do đó, cực đại hóa tính cục bộ của các ứng dụng là một trong những mục tiêu chính trong thiết kế CSDL phân bố.

5- Các nghiên cứu về hiệu suất

Vì có nhiều bộ xử lý độc lập, do đó hiệu suất được nâng cao thông qua một cơ chế song song hóa mức độ cao. Nghiên cứu này có thể áp dụng cho bất kỳ hệ thống đa xử lý (*multiprocessor system*), chứ không chỉ áp dụng cho các CSDL phân bố. Chẳng hạn, cấu hình của ví dụ 1.3 cũng cho phép song song hóa, nhưng nó không được xem là một CSDL phân bố.

Tuy nhiên, các CSDL phân bố có ưu điểm trong trường hợp sự phân rã dữ liệu phản ánh các tiêu chí phụ thuộc ứng dụng, mà các tiêu chí này làm cực đại hóa tính cục bộ của ứng dụng. Bằng cách này, các bộ xử lý sẽ ít liên lạc với nhau. Tải được dùng chung giữa các bộ xử lý khác nhau, và các hiện tượng cổ chai nguy kịch, chẳng hạn như mạng truyền thông hoặc các dịch vụ chung của toàn bộ hệ thống, sẽ được tránh khỏi. Hiệu quả này là kết quả của khả năng xử lý tự trị đối với các ứng dụng cục bộ mà đã được nói đến trong định nghĩa của CSDL phân bố.

6- Độ tin cậy và tính sẵn sàng

Cách tiếp cận CSDL phân bố, nhất là với dữ liệu dư thừa, có thể được sử dụng để có được tính sẵn sàng và độ tin cậy cao. Tuy nhiên, để có được mục tiêu này là điều không phải dễ dàng, và cần phải sử dụng các kỹ thuật mà hoàn toàn vẫn còn chưa được thỏa thuận với nhau. Khả năng xử lý tự trị của các nơi khác nhau tuy không thể bảo đảm toàn bộ độ tin cậy cao của hệ thống, nhưng nó bảo đảm đặc tính suy biến (*degradation property*). Nói cách khác, các hư hỏng trong CSDL phân bố có thể xảy ra nhiều hơn so với trong CSDL tập trung, bởi vì số thành phần nhiều hơn, nhưng ảnh hưởng của mỗi hư hỏng bị hạn chế đối với các ứng dụng sử dụng dữ liệu của nơi hư hỏng và rất hiếm khi hệ thống bị ngưng hoàn toàn.

Các động cơ thúc đẩy ở trên đối với CSDL phân bố không phải là mới. Tại sao sự phát triển của các hệ thống CSDL chỉ mới bắt đầu? Có hai lý do chính:

Thứ nhất, sự phát triển hiện nay của các máy tính nhỏ, cung cấp nhiều khả năng có mức chi phí thấp so với các máy tính lớn trước đây, sẽ hỗ trợ phần cứng cần thiết để phát triển các hệ thống thông tin phân bố;

Thứ hai, công nghệ CSDL phân bố dựa trên hai công nghệ khác nhau đã được phát triển thành một nền tảng đủ vững chắc trong những thập niên 1970 là: công nghệ mạng máy tính, và công nghệ CSDL. Đó là một công việc phức tạp để xây dựng một CSDL phân bố ở trên một mạng máy tính, và một tập hợp các hệ quản trị CSDL cục bộ tại mỗi nơi; nó sẽ không khả thi nếu không có các khối xây dựng có sẵn này.

1.4 CÁC HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU PHÂN BỐ

Một hệ quản trị CSDL phân bố (DDBMS – *Distributed DBMS*) hỗ trợ tạo lập và bảo trì các CSDL phân bố. Để phân tích các đặc điểm của các DDBMS, cần phải phân biệt các hệ thống có sẵn được thương mại hóa và các mẫu (*prototype*) nghiên cứu phát triển. Rõ ràng, một số đặc điểm có được từ thực nghiệm hiện tại sẽ được kết hợp vào trong các hệ thống thương mại hóa trong tương lai.

Nhiều hệ thống CSDL phân bố được thương mại hóa đã được phát triển bởi các nhà cung cấp các hệ quản trị CSDL tập trung. Các hệ thống này bao gồm các thành phần mới để mở rộng các khả năng của các hệ quản trị CSDL tập trung bằng cách hỗ trợ sự truyền thông và sự hợp tác giữa các hệ quản trị CSDL được cài đặt tại các nơi khác nhau của một mạng máy tính. Trong trường hợp này, các thành phần phần mềm tiêu biểu cần thiết để xây dựng một CSDL phân bố là:

1- Thành phần *quản trị CSDL* (DB – *DataBase management component*)

2- Thành phần *truyền thông dữ liệu* (DC – *Data Communication component*)

3- *Từ điển dữ liệu* (DD – *Data Dictionary*) được mở rộng để biểu diễn thông tin về sự phân bố dữ liệu trong mạng.

4- Thành phần *CSDL phân bố* (DDB – *Distributed DataBase*)

Các thành phần này được kết nối như trong hình 1.6 cho một mạng gồm hai nơi.

Chúng ta sử dụng thuật ngữ “*hệ quản trị CSDL phân bố*” để nói đến tập hợp bốn thành phần trên, trong khi đó DDB chỉ là một thành phần CSDL phân bố đặc biệt.

Tương tự, chúng ta sẽ sử dụng thuật ngữ “*hệ quản trị CSDL*” để nói đến một tập hợp các thành phần phục vụ cho việc quản lý một CSDL không phân bố, tức là các thành phần DB, DC và DD.

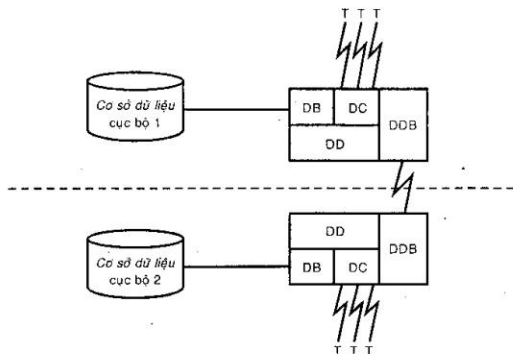
Các dịch vụ tiêu biểu được hỗ trợ bởi hệ thống trên là:

1- *Truy xuất CSDL từ xa bởi một chương trình ứng dụng*: Đây là đặc điểm quan trọng nhất, và được cung cấp bởi tất cả các hệ thống có thành phần CSDL phân bố.

2- *Một số mức trong suốt phân bố*: Đặc điểm này được hỗ trợ cho phần mở rộng khác nhau ở các hệ thống khác nhau, bởi vì cần phải có sự thỏa thuận chặt chẽ giữa trong suốt phân bố và hiệu suất.

3- *Hỗ trợ cho việc quản trị CSDL và điều khiển*: Đặc điểm này bao gồm các công cụ để giám sát CSDL, thu thập thông tin về việc sử dụng CSDL và cung cấp cái nhìn toàn cục về các tập tin dữ liệu đang có tại các nơi khác nhau.

4- *Một số hỗ trợ cho điều khiển đồng thời và phục hồi các giao dịch phân bố*.

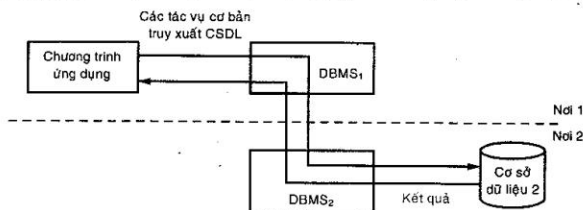


Hình 1.6 Các thành phần của DDBMS thương mại hóa

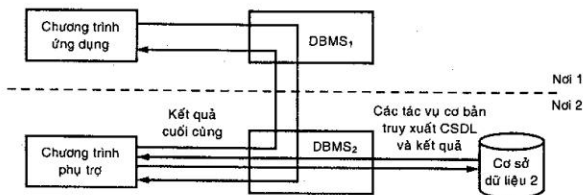
Truy xuất CSDL từ xa bởi một ứng dụng có thể được thực hiện bằng một trong hai cách cơ bản (xem hình 1.7a, b).

Hình 1.7a cho thấy ứng dụng phát ra một truy xuất CSDL cần phải tham chiếu dữ liệu từ xa. Yêu cầu này được gửi theo tuyến (*routed*) một cách tự động bởi DDBMS đến nơi đặt dữ liệu; sau đó nó được thực hiện tại nơi đó và kết quả được trả về. Bằng cách này, các đơn vị cơ bản được gủi giữa các hệ thống là tác vụ cơ bản truy xuất CSDL (*database access primitive*) và kết quả có được bằng cách thực hiện tác vụ cơ bản này. Nếu cách tiếp cận này được sử dụng để truy xuất từ xa, trong suốt phân bố có thể được thực hiện bằng cách cung cấp các tên tập tin toàn cục; sau đó các tác vụ cơ bản có thể được gửi một cách tự động đến các nơi từ xa tương ứng.

Hình 1.7b cho thấy một cách tiếp cận khác mà trong đó ứng dụng yêu cầu thực hiện một chương trình phụ trợ (*auxiliary program*) tại nơi xa. Chương trình phụ trợ này, được viết bởi người lập trình ứng dụng, truy xuất CSDL từ xa và trả về kết quả cho chương trình yêu cầu (*requesting program*).



a) Truy xuất từ xa thông qua các tác vụ cơ bản DBMS



b) Truy xuất từ xa thông qua một chương trình phụ trợ

Hình 1.7 Các loại truy xuất một cơ sở dữ liệu phân bố

Các DDBMS đều cung cấp cả hai loại truy xuất từ xa, bởi vì mỗi loại truy xuất có các ưu điểm và nhược điểm riêng. Giải pháp của hình 1.7a cung cấp nhiều trong suốt phân bố, trong khi đó giải pháp của hình 1.7b có thể hiệu quả hơn nếu nhiều truy xuất CSDL được yêu cầu, bởi vì chương trình phụ trợ có thể thực hiện tất cả các truy xuất được yêu cầu và chỉ gửi kết quả trả về.

Nếu khả năng của chương trình phụ trợ trong hình 1.7b nhiều hơn khả năng của các tác vụ cơ bản truy xuất CSDL của hình 1.7a thì giải pháp thứ hai có thể hiệu quả hơn. Do đó, các tác vụ cơ bản mạnh có khả năng một lần thao tác các tập mẫu tin là một đặc điểm rất có ích của DBMS cục bộ trong việc xây dựng một hệ thống CSDL. Tuy

nhien, hầu hết các DBMS có sẵn được thương mại hóa là các hệ thống một-lần-một-mẫu-tin (*one-record-at-a-time system*) và do đó các chương trình phụ trợ là cần thiết.

Một đặc tính quan trọng của các DDBMS là **đồng nhất** (*homogeneous*) hoặc **không đồng nhất** (*heterogeneous*). Tính đồng nhất (*homogeneity*) và tính không đồng nhất (*heterogeneity*) có thể được xem xét ở các mức khác nhau trong CSDL phân bố: phần cứng, hệ điều hành và các DBMS cục bộ. Tuy nhiên, sự khác biệt quan trọng đối với chúng ta là ở mức của các DBMS cục bộ, bởi vì sự khác nhau ở các mức thấp hơn được quản lý bởi phần mềm truyền thông. Do đó, thuật ngữ "**DDBMS đồng nhất**" nói đến một DDBMS có các DBMS tại mỗi nơi giống nhau, ngay cả khi các máy tính và các hệ điều hành khác nhau. Lưu ý rằng, trường hợp này không phải là hiếm gặp, bởi vì cùng một nhà cung cấp thông thường cung cấp cùng DBMS trên nhiều máy tính khác nhau và các hãng phần mềm cung cấp các DBMS chạy trên các máy tính khác nhau của các nhà cung cấp khác nhau.

Một DDBMS không đồng nhất phải sử dụng ít nhất hai DBMS khác nhau. Các DDBMS không đồng nhất có thêm vấn đề chuyển đổi giữa các mô hình dữ liệu khác nhau của các DBMS cục bộ khác nhau, phức tạp hơn so với các DDBMS đồng nhất. Với lý do này, nếu việc phát triển một CSDL phân bố được thực hiện từ trên xuống (*top-down*) mà không có một hệ thống đã tồn tại trước, thì sẽ thuận lợi cho việc phát triển một hệ thống đồng nhất. Tuy nhiên, như đã nói trong phần trước, trong một số trường hợp, động cơ thúc đẩy cho việc tạo lập một CSDL phân bố là sự cần thiết của việc tích hợp các CSDL đã có trước; trong trường hợp này, cần phải phát triển một DDBMS không đồng nhất, có khả năng xây dựng cái nhìn toàn cục về CSDL.

Mặc dù các vấn đề của DDBMS không đồng nhất là rất khó, do đó các DDBMS thật sự chỉ tồn tại trong các mẫu nghiên cứu, các DDBMS thương mại hóa chỉ cung cấp một số mức độ hỗ trợ cho tính

không đồng nhất. Trong khi đó không có một hệ thống nào trong các hệ thống này thực hiện sự chuyển đổi giữa các mô hình dữ liệu khác nhau, mà đó là một vấn đề thật sự khó, một số hệ thống có hỗ trợ truyền thông giữa các thành phần truyền thông dữ liệu (DC) khác nhau. Loại truyền thông này đã không được phát triển đặc biệt cho việc xây dựng các CSDL phân bố, nhưng đã được phát triển vì các lý do tương thích bên trong một hệ thống tập trung. Các ví dụ tiêu biểu của trường hợp này là nhiều DBMS đã được các hãng phần mềm sản xuất để chạy trên các máy tính IBM. Các DBMS này có một cặp thành phần tích hợp DB/DC, thành phần DC này có thể được thay thế bởi bộ kiểm tra xử lý từ xa chuẩn (*standard teleprocessing monitor*) CICS của IBM. CICS có một phương tiện truyền thông giữa các hệ thống (ISC - *InterSystem Communication*) cho phép truyền thông giữa các chương trình chạy trên các máy tính khác nhau. Vì các chương trình này có thể truy xuất CSDL cục bộ bằng cách sử dụng các thành phần DB cục bộ, điều này cho phép thực hiện các ứng dụng truy xuất dữ liệu từ một CSDL không đồng nhất, phân bố. Tuy nhiên, nhiệm vụ của những người lập trình là giải quyết các mô hình dữ liệu khác nhau.

TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU VÀ MẠNG MÁY TÍNH

Chương này giới thiệu sơ lược về các ý niệm (*concept*) của cơ sở dữ liệu (CSDL) không phân bố, truyền thống và của mạng máy tính, các ý niệm này sẽ được sử dụng trong các chương kế tiếp. Chương này không nhằm mục đích trình bày đầy đủ về các hệ CSDL, mà chỉ định nghĩa các ý niệm và các ký hiệu (*notation*) được sử dụng trong các chương sau.

Phần 2.1 giới thiệu các ý niệm của CSDL quan hệ, chuẩn hóa dữ liệu, các quy tắc toàn vẹn, các ngôn ngữ dữ liệu quan hệ và hệ quản trị CSDL quan hệ. Phần 2.2 giới thiệu các ý niệm truyền thông dữ liệu, các loại mạng, mạng diện rộng và mạng cục bộ.

2.1 TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU

2.1.1 Các ý niệm của cơ sở dữ liệu quan hệ

Cơ sở dữ liệu (database) là sự tập hợp có tổ chức (*organized collection*) các dữ liệu có liên quan luận lý với nhau (*logically related data*).

Thuật ngữ “có tổ chức” có nghĩa là dữ liệu có cấu trúc (*structured*) để người sử dụng dễ dàng lưu trữ, thao tác và truy xuất. Thuật ngữ “có liên quan” có nghĩa là dữ liệu mô tả một lĩnh vực mà một nhóm người sử dụng có quan tâm và người sử dụng có thể sử dụng dữ liệu này để trả lời các câu hỏi liên quan đến lĩnh vực đó.

Cơ sở dữ liệu quan hệ (relational database) là CSDL mà dữ liệu được lưu trữ trong các bảng.

Miền (domain) là một tập các giá trị. Thông thường, miền là một tập con của một kiểu dữ liệu và được ký hiệu bắt đầu bằng chữ *D* (ví dụ $D_1, D_2, \dots$).

Thuộc tính (attribute) là một đặc tính riêng của đối tượng dữ liệu. Một đối tượng dữ liệu có thể có nhiều thuộc tính. Thông thường, thuộc tính được ký hiệu bắt đầu bằng chữ *A* (ví dụ $A_1, A_2, \dots$). Đối với

một đối tượng dữ liệu, mỗi thuộc tính đều được đặt tên phân biệt, được gọi là *tên thuộc tính* (*attribute name*). Mỗi thuộc tính được nhận giá trị từ một miền, miền này được gọi là *miền trị của thuộc tính*. Miền trị của một thuộc tính A được ký hiệu là $dom(A)$. Mỗi miền trị của thuộc tính đều có chứa một giá trị đặc biệt và được gọi là *giá trị rỗng* (*null*). Giá trị này đặc trưng cho *một giá trị không thể xác định được* hoặc đặc trưng cho *một giá trị chưa thể xác định được* tại thời điểm đang xét nhưng có thể được xác định tại một thời điểm sau đó. Ví dụ 2.1 Một nhà cung cấp có các thuộc tính là mã nhà cung cấp (SNUM), tên nhà cung cấp (NAME), thành phố (CITY). Các tên ghi trong dấu ngoặc là các tên thuộc tính. Miền trị của thuộc tính CITY là $dom(CITY)$.

Lược đồ quan hệ R (*relation schema*) là một tập hữu hạn các thuộc tính $\{A_1, A_2, \dots, A_n\}$ và được ký hiệu là $R(A_1, A_2, \dots, A_n)$ với R là tên của lược đồ quan hệ.

Vị từ của lược đồ quan hệ R (*predicate*), ký hiệu là $||R||$, là một phát biểu dùng để chỉ ra ngữ nghĩa của lược đồ quan hệ R .

Bậc của lược đồ quan hệ (*degree, arity*) là số thuộc tính của lược đồ quan hệ. **Bậc của một quan hệ** trên lược đồ quan hệ bằng bậc của lược đồ quan hệ.

Ví dụ 2.2 Lược đồ quan hệ của nhà cung cấp là SUPPLIER (SNUM, NAME, CITY). Vị từ của lược đồ quan hệ SUPPLIER là "Mỗi nhà cung cấp có một mã nhà cung cấp SNUM phân biệt, tên nhà cung cấp NAME, thuộc thành phố CITY". Bậc của lược đồ quan hệ này là 3.

Gọi $D_1, D_2, \dots, D_n$ là các miền tương ứng với các thuộc tính $A_1, A_2, \dots, A_n$ và $D = D_1 \cup D_2 \cup \dots \cup D_n$.

Quan hệ r (*relation*) trên lược đồ R , ký hiệu là $r(R)$, là một tập hữu hạn các ánh xạ $\{t_1, t_2, \dots, t_p\}$ từ R vào D , với điều kiện mọi ánh xạ $t \in r$ thì $t(A_i) \in D_i$; $1 \leq i \leq n$. Mỗi ánh xạ được gọi là một **bộ** (*tuple*) của quan hệ. Một bộ của một quan hệ bậc n được gọi là **bộ- n** (*n -tuple*).

Quan hệ r là một **thể hiện quan hệ** (*relation instance*) của lược đồ quan hệ tại một thời điểm.

Một bộ của quan hệ bao gồm nhiều giá trị, mỗi giá trị được gọi là một **thành phần** (*component*) của bộ. Thành phần A của bộ u được ký hiệu là $u[A]$ hoặc $u.A$, các thành phần $X = \{A_1, A_2, \dots, A_k\}$ của bộ u được ký hiệu là $u[X]$.

Ví dụ 2.3 Một quan hệ SUPPLIER trên lược đồ quan hệ SUPPLIER (SNUM, NAME, CITY) là:

SNUM	NAME	CITY
S1	Nguyễn Trung Tiến	SF
S2	Hồ Thị Thu Trang	LA
S3	Nguyễn Ngọc Thanh Trúc	SF

Hình 2.1 Quan hệ SUPPLIER

$u = ('S1', 'Nguyễn Trung Tiến', 'SF')$ là một bộ của quan hệ SUPPLIER.
 $u[\text{SNUM}, \text{NAME}] = ('S1', 'Nguyễn Trung Tiến')$ là thành phần SNUM, NAME của bộ u .

Khóa (key) của lược đồ quan hệ R có tập thuộc tính $U = \{A_1, A_2, \dots, A_m\}$ là một tập con $K = \{A_{j_1}, A_{j_2}, \dots, A_{j_n}\}$, với $j_1, j_2, \dots, j_n$ là các số nguyên phân biệt và nằm trong khoảng từ 1 đến m , phải thỏa mãn đồng thời hai điều kiện sau đây:

- (1) $\forall r(R); \forall t_1, t_2 \in r$: nếu $t_1 \neq t_2$ thì $t_1[K] \neq t_2[K]$
- (2) Không tồn tại $K' \subset K$ sao cho K' thỏa mãn điều kiện (1).

Một tập thuộc tính $K \subseteq U$ thỏa mãn điều kiện (1) được gọi là một **siêu khóa (superkey)** của R . Một siêu khóa hiển nhiên của R là U .

Một lược đồ quan hệ R phải có ít nhất một khóa và có thể có nhiều khóa.

Các thuộc tính thuộc một khóa được gọi là **thuộc tính khóa**, các thuộc tính còn lại trong lược đồ quan hệ được gọi là các **thuộc tính không khóa**. Trong lược đồ quan hệ, các thuộc tính khóa được gạch dưới.

Ví dụ 2.4 Lược đồ quan hệ SUPPLIER (SNUM, NAME, CITY) có khóa là SNUM. SNUM là thuộc tính khóa, NAME và CITY là các thuộc tính không khóa. Một siêu khóa của SUPPLIER là {SNUM, NAME}.

Lược đồ cơ sở dữ liệu (database schema) là một tập hợp các lược đồ quan hệ.

Ví dụ 2.5 Ví dụ về một lược đồ CSDL như sau:

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)
 DEPT (DEPTNUM, NAME, AREA, MGRNUM)
 SUPPLIER (SNUM, NAME, CITY)
 SUPPLY (SNUM, PNUM, DEPTNUM, QUAN)

2.1.2 Chuẩn hóa

Chuẩn hóa (*normalization*) là một quá trình thuận nghịch từng bước (*step-by-step reversible process*) để thay thế tập hợp các quan hệ cho trước thành các quan hệ có cấu trúc càng đơn giản hơn và chuẩn hơn. Mục đích của chuẩn hóa là loại bỏ các bất thường (*anomaly*) của một quan hệ để có được các quan hệ "tốt hơn". Bốn vấn đề sau đây có thể tồn tại trong một lược đồ quan hệ.

1- Bất thường do sự lặp lại (*repetition anomaly*)

Thông tin có thể bị lặp lại không cần thiết. Ví dụ, xét lược đồ quan hệ SUPPLY (SNUM, SNAME, PNUM, DEPTNUM, QUAN). Tên nhà cung cấp SNAME bị lặp lại với mỗi mặt hàng PNUM của nhà cung cấp có mã SNUM. Điều này làm lãng phí vùng nhớ lưu trữ.

SUPPLY

SNUM	SNAME	PNUM	DEPTNUM	QUAN
S1	Nguyễn Trung Tiến	P1	D1	10
S1	Nguyễn Trung Tiến	P2	D1	20
S2	Hồ Thị Thu Trang	P1	D1	15
S2	Hồ Thị Thu Trang	P2	D2	30
S3	Nguyễn Ngọc Thanh Trúc	P3	D1	25

Hình 2.2 Sự lặp lại của SNAME trong quan hệ SUPPLY

2- Bất thường khi cập nhật (*update anomaly*)

Đây là hệ quả của sự lặp lại dữ liệu, thực hiện việc cập nhật có thể gặp nhiều rắc rối. Ví dụ, nếu thay đổi tên của một nhà cung cấp thì cần phải cập nhật nhiều bộ của nhà cung cấp này để phản ánh sự thay đổi này.

3- Bất thường khi thêm vào (*insertion anomaly*)

Đôi khi chúng ta không thể thêm thông tin mới vào CSDL. Ví dụ, khi có thông tin của một nhà cung cấp mới thì thông tin này không thể thêm vào quan hệ SUPPLY, bởi vì nếu chúng ta thêm thông tin này vào quan hệ SUPPLY thì PNUM phải có giá trị *null*, nhưng PNUM là thuộc tính khóa nên nó không thể có giá trị *null* (ràng buộc toàn vẹn về khóa).

4- Bất thường khi loại bỏ (*deletion anomaly*)

Cũng tương tự như thêm vào, việc loại bỏ có thể làm mất thông tin trong CSDL. Ví dụ, nếu nhà cung cấp có mã 'S3' không còn cung cấp mặt hàng có mã 'P3', việc xóa bộ này trong quan hệ SUPPLY sẽ làm mất thông tin của nhà cung cấp có mã 'S3', bởi vì nhà cung cấp này chỉ xuất hiện một lần trong quan hệ SUPPLY.

Chuẩn hóa sẽ biến đổi các lược đồ quan hệ thành các lược đồ không có các vấn đề này. Cách tiếp cận phổ biến nhất để chuẩn hóa một lược đồ CSDL quan hệ là *cách tiếp cận phân rã* (*decomposition approach*), chúng ta bắt đầu từ một quan hệ, được gọi là *quan hệ chung* (*universal relation*), chứa tất cả các thuộc tính (và có những bất thường) và rút gọn quan hệ này nhiều lần. Ở mỗi lần lặp, quan hệ được phân chia thành hai hoặc nhiều quan hệ có *dạng chuẩn* (NF – Normal Form) cao hơn. Một quan hệ ở một dạng chuẩn nếu nó thỏa mãn các điều kiện của dạng chuẩn này. Codd là người đầu tiên đã định nghĩa các dạng chuẩn 1 (1NF), dạng chuẩn 2 (2NF) và dạng chuẩn 3 (3NF). Sau đó, Boyce và Codd đã định nghĩa một phiên bản sửa đổi của dạng chuẩn 3 và được gọi là dạng chuẩn Boyce-Codd (BCNF). Đi sau các dạng chuẩn này là dạng chuẩn 4 (4NF) và dạng chuẩn 5 (5NF).

Có mối quan hệ phân cấp giữa các dạng chuẩn này. Mọi quan hệ đã được chuẩn hóa (*decomposed relation*) đều ở dạng chuẩn 1NF, một số quan hệ ở dạng chuẩn 1NF cũng ở dạng chuẩn 2NF, một số quan hệ ở dạng chuẩn 3NF và cứ như thế. Các dạng chuẩn cao hơn có các đặc tính tốt hơn các dạng chuẩn khác đối với bốn bất thường đã được bàn luận ở trên.

Một trong các yêu cầu của quá trình chuẩn hóa là phân rã phải bảo toàn. Điều này có nghĩa là thay thế một quan hệ bởi nhiều quan hệ khác thì không được làm mất thông tin. Nếu có thể kết các quan hệ đã được phân rã để tạo lại quan hệ ban đầu, thì quá trình phân rã này được gọi là *phân rã bảo toàn thông tin* (*lossless decomposition*).

Phép kết là một phép toán đại số quan hệ, có hai quan hệ toán hạng, kết nối mỗi bộ của quan hệ thứ hai với các bộ của quan hệ thứ nhất khi thỏa mãn một điều kiện xác định. Điều kiện này được định nghĩa trên các thuộc tính của hai quan hệ. Ví dụ, điều kiện có thể là giá trị của một thuộc tính của quan hệ thứ nhất bằng với giá trị của một thuộc tính của quan hệ thứ hai.

Một yêu cầu khác của quá trình chuẩn hóa là bảo toàn phụ thuộc (*dependency preservation*). Một phân rã là bảo toàn phụ thuộc nếu hợp các phụ thuộc trong các quan hệ được phân rã sẽ tương đương với bao đóng (*closure*) (đối với tập hợp các quy tắc suy diễn) của các phụ thuộc của quan hệ ban đầu.

Các cấu trúc phụ thuộc

Các dạng chuẩn được dựa trên các cấu trúc phụ thuộc (*dependency structure*). BCNF và các dạng chuẩn thấp hơn dựa trên các phụ thuộc hàm (*FD - Functional Dependency*). 4NF dựa trên các phụ thuộc đa trị (*multivalued dependency*) và 5NF dựa trên các phụ thuộc chiếu-kết (*projection-join dependency*).

Cho r là một quan hệ, X và Y là hai tập thuộc tính của r . Chúng ta nói " X xác định hàm Y " hoặc " Y phụ thuộc hàm vào X ", ký hiệu là $X \rightarrow Y$ và được gọi là phụ thuộc hàm, nếu:

$$\forall u, v \in r: u[X] = v[X] \Rightarrow u[Y] = v[Y]$$

tức là, với mỗi giá trị của X trong r chỉ tương ứng với một giá trị của Y .

Khóa của một quan hệ xác định hàm các thuộc tính không khóa của quan hệ này.

Trong phụ thuộc hàm $X \rightarrow Y$, X được gọi là *định thuộc* (*determinant*).

Ví dụ 2.6 Trong quan hệ SUPPLIER, chúng ta có phụ thuộc hàm:

$$f_1: \text{SNUM} \rightarrow \{\text{NAME}, \text{CITY}\}$$

Trong quan hệ SUPPLY của hình 2.2, chúng ta có các phụ thuộc hàm:

$$f_2: \{\text{SNUM}, \text{PNUM}\} \rightarrow \{\text{SNAME}, \text{DEPTNUM}, \text{QUAN}\}$$

$$f_3: \text{SNUM} \rightarrow \text{SNAME}$$

Lưu ý rằng, một số thuộc tính của vế phải (*right-hand side*) của phụ thuộc hàm f_2 chỉ phụ thuộc hàm vào tập con của các thuộc tính của vế trái (*left-hand side*) của f_2 , đó chính là phụ thuộc hàm f_3 . Chúng ta nói, thuộc tính SNAME phụ thuộc hàm riêng phần (*partially functionally dependent*) vào $\{\text{SNUM}, \text{PNUM}\}$, trong khi đó các thuộc tính DEPTNUM, QUAN phụ thuộc hàm đầy đủ (*fully functionally dependent*) vào $\{\text{SNUM}, \text{PNUM}\}$.

Cho X, Y, Z, W là các tập thuộc tính, A là một thuộc tính, và $X \cup Y$, được ký hiệu XY .

Chúng ta nói:

$X \rightarrow A$ là một phụ thuộc hàm đầy đủ nếu không tồn tại $Y \subset X$ để cho $Y \rightarrow A$.

$X \rightarrow A$ là một phụ thuộc hàm bắc cầu nếu tồn tại Y để cho $X \rightarrow Y$ và $Y \rightarrow A$, nhưng $A \notin XY$ và không tồn tại $Y \rightarrow X$.

Có một tập hợp các quy tắc suy diễn (*inference rule*) được dựa trên một tập hợp các tiên đề, được gọi là hệ tiên đề Armstrong (*Armstrong's axioms*), bao gồm các tiên đề suy diễn (*inference axiom*) sau đây:

- | | |
|--|--|
| F1. Phản xạ (<i>reflexivity</i>): | $X \rightarrow X$ |
| F2. Thêm vào (<i>augmentation</i>): | $X \rightarrow Y \Rightarrow XZ \rightarrow Y$ |
| F3. Hợp (<i>additivity</i>): | $X \rightarrow Y$ và $X \rightarrow Z \Rightarrow X \rightarrow YZ$ |
| F4. Chiếu (<i>projectivity</i>): | $X \rightarrow YZ \Rightarrow X \rightarrow Y$ |
| F5. Bắc cầu (<i>transitivity</i>): | $X \rightarrow Y$ và $Y \rightarrow Z \Rightarrow X \rightarrow Z$ |
| F6. Bắc cầu giả (<i>pseudotransitivity</i>): | $X \rightarrow Y$ và $YZ \rightarrow W \Rightarrow XZ \rightarrow W$ |

Hệ tiên đề Armstrong cho phép suy diễn một số phụ thuộc mới từ các phụ thuộc đã có, cho phép tìm phủ tối thiểu (*minimal cover*) của một tập phụ thuộc hàm, là một tập hợp tối thiểu gồm các phụ thuộc hàm mà từ đó có thể tạo ra các phụ thuộc hàm khác. Có các phương pháp cho phép tạo ra một lược đồ quan hệ ở các dạng chuẩn cao từ một phủ tối thiểu của một tập phụ thuộc hàm và một tập thuộc tính cho trước.

Các dạng chuẩn

1- Một lược đồ quan hệ R ở dạng chuẩn 1 nếu mọi thuộc tính của R đều chứa các giá trị nguyên tố (*atomic value*), giá trị này không là một danh sách các giá trị hoặc các giá trị phức hợp (*composite value*).

Các quan hệ ở dạng chuẩn 1 vẫn có các bất thường đã được đề cập ở trên. Để loại bỏ một số bất thường này, chúng cần phải được phân rã thành các quan hệ ở dạng chuẩn cao hơn.

2- Một lược đồ quan hệ R ở dạng chuẩn 2 đối với một tập phụ thuộc hàm F nếu R ở dạng chuẩn 1 và mọi thuộc tính không khóa (*nonprime attribute*) đều phụ thuộc hàm đầy đủ vào mọi khóa của R .

3- Một lược đồ quan hệ R ở dạng chuẩn 3 đối với một tập phụ thuộc hàm F nếu R ở dạng chuẩn 1 và mọi thuộc tính không khóa (*nonprime attribute*) không phụ thuộc hàm bắc cầu vào một khóa của R .

Một phát biểu khác của dạng chuẩn 3 là: một lược đồ quan hệ R ở dạng chuẩn 3 đối với một tập phụ thuộc hàm F nếu R ở dạng chuẩn 1 và mọi phụ thuộc hàm $X \rightarrow A$ với $A \notin X$ thì X là một siêu khóa của R hoặc A là một thuộc tính khóa (*prime attribute*).

4- Một lược đồ quan hệ R ở dạng chuẩn Boyce-Codd đối với một tập phụ thuộc hàm F nếu R ở dạng chuẩn 1 và mọi phụ thuộc hàm $X \rightarrow A$ với $A \notin X$ thì X là một siêu khóa của R .

Nếu một lược đồ quan hệ R ở dạng chuẩn Boyce-Codd thì nó cũng ở dạng chuẩn 3.

Phương pháp “*Phân rã*” cho phép phân rã bảo toàn thông tin (*lossless decomposition*) từ một quan hệ ở dạng chuẩn 1 thành các quan hệ ở dạng chuẩn 3.

Phương pháp “*Tổng hợp*” cho phép phân rã bảo toàn thông tin và bảo toàn phụ thuộc hàm (*dependency-preserving decomposition*) từ một quan hệ ở dạng chuẩn 1 thành các quan hệ ở dạng chuẩn 3.

S1

SNUM	SNAME
S1	Nguyễn Trung Tiến
S2	Hồ Thị Thu Trang
S3	Nguyễn Ngọc Thanh Trúc

S2

SNUM	PNUM	DEPTNUM	QUAN
S1	P1	D1	10
S1	P2	D1	20
S2	P1	D1	15
S2	P2	D2	30
S3	P3	D1	25

Hình 2.3 Các quan hệ đã được chuẩn hóa

2.1.3 Các quy tắc toàn vẹn

Các quy tắc toàn vẹn (*integrity rule*) là các ràng buộc dùng để định nghĩa các trạng thái nhất quán của CSDL. Chúng thường được biểu diễn như là các điều khẳng định (*assertion*). Các ràng buộc toàn vẹn (*integrity constraint*) có thể là ràng buộc cấu trúc (*structural constraint*) hoặc ràng buộc hành vi (*behavioral constraint*). Các ràng buộc cấu trúc được thừa kế từ mô hình dữ liệu theo nghĩa chúng cho biết thông tin về các mối quan hệ dữ liệu mà không thể được mô hình hóa trực tiếp. Các ràng buộc hành vi cho phép bảo đảm ngữ nghĩa của các ứng dụng. Các phụ thuộc đã được đề cập trong phần trên là các ràng buộc hành vi. Việc duy trì các ràng buộc toàn vẹn thông thường

rất tốn kém về mặt các tài nguyên của hệ thống. Về mặt lý tưởng, chúng cần phải được thẩm tra cho mỗi cập nhật CSDL, bởi vì các cập nhật có thể dẫn đến các trạng thái không nhất quán của CSDL.

Tối thiểu cần phải có hai ràng buộc cấu trúc trong mô hình quan hệ là *quy tắc thực thể (entity rule)* và *quy tắc toàn vẹn tham chiếu (referential integrity rule)*.

Theo định nghĩa, mọi quan hệ đều phải có một khóa chính (*primary key*). Quy tắc thực thể phát biểu rằng, mỗi thuộc tính của khóa không được có giá trị *null* (*null value*) và các giá trị của khóa không được trùng với nhau.

Ví dụ 2.7 Trong quan hệ EMP, khóa của quan hệ này là EMPNUM, do đó chúng ta có quy tắc thực thể là thuộc tính EMPNUM không được có giá trị *null* và các giá trị của EMPNUM không được trùng nhau.

Toàn vẹn tham chiếu có ích cho việc lưu giữ các mối quan hệ giữa các đối tượng mà mô hình quan hệ không thể biểu diễn được. Toàn vẹn tham chiếu liên quan đến hai quan hệ và ràng buộc này bắt buộc một nhóm thuộc tính của quan hệ này là khóa của quan hệ kia.

Ví dụ 2.8 Xét hai quan hệ EMP và DEPT, một toàn vẹn tham chiếu liên hệ đến hai quan hệ này là các giá trị của thuộc tính của DEPTNUM của quan hệ EMP phải có trong DEPTNUM của quan hệ DEPT.

2.1.4 Các ngôn ngữ dữ liệu quan hệ

1- Đại số quan hệ

Đại số quan hệ (*relational algebra*) bao gồm một tập hợp các phép toán thực hiện trên các quan hệ. Nó được phát triển từ lý thuyết tập hợp (các quan hệ tương ứng với các tập hợp). Mỗi phép toán (*operator / operation*) có một hoặc nhiều toán hạng (*operand*), mà mỗi toán hạng là một quan hệ. Kết quả của phép toán là một quan hệ và nó có thể là toán hạng của một phép toán khác. Các phép toán này cho phép truy vấn và cập nhật CSDL quan hệ.

a- Các phép toán đại số quan hệ

Có năm phép toán đại số quan hệ cơ bản (*fundamental relational algebra operator*) và các phép toán khác có thể được xác định theo các phép toán này.

Năm phép toán cơ bản là:

- Phép chọn (*selection*)
- Phép chiếu (*projection*)
- Phép hợp (*union*)
- Phép hiệu (*difference*)
- Phép tích Descartes (*cartesian product*).

Hai phép toán đầu tiên là các *phép toán một ngôi* (*unary operator*) và ba phép toán cuối cùng là các *phép toán hai ngôi* (*binary operator*).

Các phép toán khác được xác định theo các phép toán cơ bản này là:

- Phép giao (*intersection*)
- Phép kết-θ (*θ-join*)
- Phép kết tự nhiên (*natural join*)
- Phép nửa kết-θ (*θ-semijoin*)
- Phép nửa kết tự nhiên (*natural semijoin*)
- Phép chia (*quotient/division*).

Trên thực tế, đại số quan hệ được mở rộng với các phép toán để gom nhóm, sắp thứ tự kết quả, thực hiện các hàm số học (*arithmetic function*) và các hàm kết hợp (*aggregate function*).

Chúng ta xét các quan hệ sau đây:

Quan hệ SUPPLIER

SNUM	NAME	CITY
S1	Nguyễn Trung Tiến	SF
S2	Hồ Thị Thu Trang	LA
S3	Nguyễn Ngọc Thanh Trúc	SF

Quan hệ SUPPLIER2

SNUM	NAME	CITY
S1	Nguyễn Trung Tiến	SF
S4	Nguyễn Trung Trực	LA

Quan hệ SUPPLY

SNUM	PNUM	DEPTNUM	QUAN
S1	P1	D1	10
S1	P2	D1	20

Hình 2.4 Các quan hệ ví dụ

Phép chọn: Cho một quan hệ r trên một lược đồ quan hệ R và điều kiện F (là một biểu thức luận lý có giá trị là *true* hoặc *false*) bao gồm:

- Các toán hạng là các hằng, hoặc các tên thuộc tính của lược đồ quan hệ R .

- Các phép toán so sánh $=, \neq, <, \leq, >, \geq$.

- Các phép toán luận lý *not* ($\neg$), *and* ($\wedge$), *or* ($\vee$)

Phép chọn trên r theo điều kiện F , ký hiệu là $\sigma_F(r)$, cho kết quả là một quan hệ bao gồm các bộ của r thỏa mãn điều kiện F là *true*.

$$\sigma_F(r) = \{u \mid u \in r \text{ và } F(u) = \text{true}\}$$

trong đó $F(u)$ là điều kiện có được bằng cách thay thế các tên thuộc tính trong điều kiện F bởi các giá trị tương ứng trong bộ u .

Ví dụ 2.9 Phép chọn $\sigma_{\text{CITY}=\text{SF}}$ (SUPPLIER) cho kết quả là:

SNUM	NAME	CITY
S1	Nguyễn Trung Tiến	SF
S3	Nguyễn Ngọc Thanh Trúc	SF

Phép chiếu: Cho một quan hệ r trên lược đồ quan hệ $R (A_1, A_2, \dots, A_m)$ và một tập con các thuộc tính $X = \{A_{j_1}, A_{j_2}, \dots, A_{j_n}\}$ với $j_1, j_2, \dots, j_n$ là các số nguyên phân biệt và nằm trong khoảng từ 1 đến m .

Phép chiếu một quan hệ r trên một tập thuộc tính X , ký hiệu là $\Pi_X(r)$, cho kết quả là một quan hệ bao gồm các bộ $u (u_1, u_2, \dots, u_n)$ sao cho tồn tại một bộ $v (v_1, v_2, \dots, v_m)$ trong r để $u_i = v_{j_i}$ với $i = 1, 2, \dots, n$.

$$\Pi_X(r) = \{u \mid \exists v \in r : u = v[X]\}$$

Ví dụ 2.10 Phép chiếu $\Pi_{\text{SNUM}, \text{NAME}}$ (SUPPLIER) cho kết quả là:

SNUM	NAME
S1	Nguyễn Trung Tiến
S2	Hồ Thị Thu Trang
S3	Nguyễn Ngọc Thanh Trúc

Phép hợp: Cho hai quan hệ r và s trên cùng một lược đồ quan hệ R . Phép hợp của hai quan hệ r và s , ký hiệu là $r \cup s$, là một quan hệ bao gồm các bộ của r hay của s .

$$r \cup s = \{u \mid u \in r \text{ hay } u \in s\}$$

Phép hợp có tính giao hoán.

Ví dụ 2.11 Phép hợp $SUPPLIER \cup SUPPLIER2$ cho kết quả là:

SNUM	NAME	CITY
S1	Nguyễn Trung Tiến	SF
S2	Hồ Thị Thu Trang	LA
S3	Nguyễn Ngọc Thanh Trúc	SF
S4	Nguyễn Trung Trực	LA

Phép hiệu: Cho hai quan hệ r và s trên cùng một lược đồ quan hệ R . Phép hiệu của quan hệ r cho s , ký hiệu là $r - s$, là một quan hệ bao gồm các bộ của r không có trong s .

$$r - s = \{u \mid u \in r \text{ và } u \notin s\}$$

Phép hiệu không có tính giao hoán.

Ví dụ 2.12 Phép hiệu $SUPPLIER - SUPPLIER2$ cho kết quả là:

SNUM	NAME	CITY
S2	Hồ Thị Thu Trang	LA
S3	Nguyễn Ngọc Thanh Trúc	SF

Phép tích Descartes: Cho quan hệ r trên lược đồ quan hệ $R(A_1, A_2, \dots, A_m)$ và s trên lược đồ quan hệ $S(B_1, B_2, \dots, B_n)$.

Phép tích Descartes của hai quan hệ r và s , ký hiệu là $r \times s$, là một quan hệ trên lược đồ $T(A_1, A_2, \dots, A_m; B_1, B_2, \dots, B_n)$ bao gồm các bộ u sao cho m thành phần đầu tiên là một bộ của r và n thành phần cuối cùng là một bộ của s .

$$r \times s = \{(u_1, \dots, u_m, u_{m+1}, \dots, u_{m+n}) \mid (u_1, \dots, u_m) \in r \text{ và } (u_{m+1}, \dots, u_{m+n}) \in s\}$$

Phép tích Descartes có tính giao hoán.

Ví dụ 2.13 Phép tích $SUPPLIER \times SUPPLY$ cho kết quả là:

SNUM	NAME	CITY	SNUM	PNUM	DEPTNUM	QUAN
S1	Nguyễn Trung Tiến	SF	S1	P1	D1	10
S1	Nguyễn Trung Tiến	SF	S1	P2	D1	20
S2	Hồ Thị Thu Trang	LA	S1	P1	D1	10
S2	Hồ Thị Thu Trang	LA	S1	P2	D1	20
S3	Nguyễn Ngọc Thanh Trúc	SF	S1	P1	D1	10
S3	Nguyễn Ngọc Thanh Trúc	SF	S1	P2	D1	20

Phép giao: Cho hai quan hệ r và s trên cùng một lược đồ quan hệ R . Phép giao của hai quan hệ r và s , ký hiệu là $r \cap s$, là một quan hệ bao gồm các bộ có trong cả hai r và s .

$$r \cap s = \{u \mid u \in r \text{ và } u \in s\}$$

Phép giao của hai quan hệ r và s có thể được tính từ phép hiệu như sau:

$$r \cap s = r - (r - s) = s - (s - r)$$

Phép giao có tính giao hoán.

Ví dụ 2.14 Phép giao của $SUPPLIER \cap SUPPLIER2$ cho kết quả là:

SNUM	NAME	CITY
S1	Nguyễn Trung Tiến	SF

Phép kết-θ: Cho quan hệ r trên lược đồ quan hệ R , quan hệ s trên lược đồ quan hệ S và $T = R \cup S$ (các thuộc tính của R được xem là khác với các thuộc tính của S), θ là một phép so sánh ($=, \neq, <, \leq, >, \geq$), $A \in R$ và $B \in S$ là hai thuộc tính có thể so sánh với nhau bởi phép θ .

Phép kết- θ của r và s trên hai thuộc tính A và B , ký hiệu là $r \bowtie_{A \theta B} s$, cho kết quả là một quan hệ trên lược đồ quan hệ T bao gồm các bộ t như sau:

$$q(T) = \{t \mid \exists t_r \in r \text{ và } \exists t_s \in s \text{ với } t[R] = t_r \text{ và } t[S] = t_s \text{ và } t[A] \theta t[B]\}$$

Phép kết- θ của hai quan hệ r và s có thể được tính từ phép tích và phép chọn như sau:

$$r \bowtie_{A \theta B} s = \sigma_{A \theta B} (r \times s)$$

Phép kết- θ có tính giao hoán.

Ví dụ 2.15 Phép kết $SUPPLIER \bowtie_{SUPPLIER.SNUM > SUPPLY.SNUM} SUPPLY$ cho kết quả là:

SNUM	NAME	CITY	SNUM	PNUM	DEPTNUM	QUAN
S2	Hồ Thị Thu Trang	LA	S1	P1	D1	10
S2	Hồ Thị Thu Trang	LA	S1	P2	D1	20
S3	Nguyễn Ngọc Thanh Trúc	SF	S1	P1	D1	10
S3	Nguyễn Ngọc Thanh Trúc	SF	S1	P2	D1	20

Phép kết tự nhiên: Cho quan hệ r trên lược đồ quan hệ R , quan hệ s trên lược đồ quan hệ S và $T = R \cup S$ (R và S có thể có các thuộc tính chung).

Phép kết tự nhiên của r và s , ký hiệu là $r \bowtie s$, cho kết quả là một quan hệ q trên lược đồ quan hệ T , bao gồm các bộ t sao cho tồn tại các bộ $t_r \in r$ và $t_s \in s$ với $t_r = t[R]$ và $t_s = t[S]$.

$$r \bowtie s = \{t \mid \exists t_r \in r \text{ và } \exists t_s \in s \text{ với } t[R] = t_r \text{ và } t[S] = t_s\}$$

Định nghĩa của phép kết tự nhiên không đòi hỏi hai tập thuộc tính R và S giao nhau khác rỗng. Nếu $R \cap S = \emptyset$ thì $r \bowtie s$ là phép tích Descartes của r và s .

Phép kết tự nhiên của hai quan hệ r và s có thể được tính từ phép tích, phép chọn và phép chiếu như sau:

$$r \bowtie s = \Pi_T \sigma_F (r \times s)$$

với F là biểu thức $RA_1 = SA_1$ and $RA_2 = SA_2$ and ... and $RA_m = SA_m$ trong đó $R \cap S = \{A_1, A_2, \dots, A_m\}$.

Phép kết tự nhiên có tính giao hoán.

Ví dụ 2.16 Phép kết SUPPLIER $\bowtie$ SUPPLY cho kết quả là:

SNUM	NAME	CITY	PNUM	DEPTNUM	QUAN
S1	Nguyễn Trung Tiến	SF	P1	D1	10
S1	Nguyễn Trung Tiến	SF	P2	D1	20

Phép nửa kết- θ : Cho quan hệ r trên lược đồ quan hệ R , quan hệ s trên lược đồ quan hệ S , θ là một phép so sánh ($=, \neq, <, \leq, >, \geq$), $A \in R$ và $B \in S$ là hai thuộc tính có thể so sánh với nhau bởi phép θ .

Phép nửa kết- θ của r và s trên hai thuộc tính A và B , ký hiệu là $r \bowtie_{A\theta B} s$, cho kết quả là một quan hệ trên lược đồ quan hệ R như sau:

$$r \bowtie_{A\theta B} s = \{u \mid u \in r \text{ và } \exists v \in s \text{ với } u[A] \theta v[B]\}$$

Phép nửa kết- θ của hai quan hệ r và s có thể được tính từ phép kết- θ và phép chiếu như sau:

$$r \bowtie_{A\theta B} s = \Pi_R (r \bowtie s)$$

Phép nửa kết- θ không có tính giao hoán.

Ưu điểm của phép nửa kết là làm giảm số bộ cần được xử lý khi thực hiện phép kết. Trong các hệ thống CSDL tập trung, điều này là quan trọng bởi vì nó sẽ làm giảm số lần truy xuất vùng nhớ thứ cấp bằng cách sử dụng bộ nhớ chính. Điều này lại càng quan trọng hơn trong các CSDL phân bố bởi vì nó làm giảm khối lượng dữ liệu được truyền giữa các nơi để thực hiện truy vấn.

Ví dụ 2.17: Phép nửa kết $\text{SUPPLIER} \triangleright < \text{SUPPLIER.SNUM} \triangleright \text{SUPPLY.SNUM} \text{ SUPPLY}$ cho kết quả là:

SNUM	NAME	CITY
S2	Hồ Thị Thu Trang	LA
S3	Nguyễn Ngọc Thanh Trúc	SF

Phép nửa kết tự nhiên: Cho quan hệ r trên lược đồ quan hệ R , quan hệ s trên lược đồ quan hệ S và $X = R \cap S \neq \emptyset$.

Phép nửa kết tự nhiên của r và s , ký hiệu là $r \triangleright < s$, cho kết quả là một quan hệ trên lược đồ quan hệ R như sau:

$$r \triangleright < s = \{u \mid u \in r \text{ và } \exists v \in s \text{ với } u[X] = v[X]\}$$

Phép nửa kết tự nhiên của hai quan hệ r và s có thể được tính từ phép kết tự nhiên và phép chiếu như sau:

$$r \triangleright < s = \Pi_R (r \triangleright \triangleleft s)$$

Phép nửa kết tự nhiên không có tính giao hoán.

Ví dụ 2.18 Phép nửa kết $\text{SUPPLIER} \triangleright < \text{SUPPLY}$ cho kết quả là:

SNUM	NAME	CITY
S1	Nguyễn Trung Tiến	SF

Phép chia: Cho quan hệ r trên lược đồ quan hệ $R (A_1, A_2, \dots, A_k, A_{k+1}, \dots, A_m)$ và quan hệ s trên lược đồ quan hệ $S (A_1, A_2, \dots, A_k)$.

Phép chia quan hệ r cho s với $s \neq \emptyset$, ký hiệu là $r \div s$, cho kết quả là một quan hệ trên lược đồ quan hệ $T (A_{k+1}, \dots, A_m)$ bao gồm các bộ $(u_{k+1}, \dots, u_m)$ sao cho đối với tất cả các bộ $(u_1, \dots, u_k)$ thuộc s thì bộ $(u_1, \dots, u_k, u_{k+1}, \dots, u_m)$ thuộc r .

Phép chia quan hệ r cho s có thể được tính từ các phép chiếu, phép tích, phép hiệu.

$$r \div s = \Pi_{A_{k+1}, \dots, A_m} (r) - \Pi_{A_{k+1}, \dots, A_m} ((\Pi_{A_1, \dots, A_k} (r) \times s) - r)$$

Phép chia không có tính giao hoán.

Ví dụ 2.19 Cho quan hệ SUPPLY :

SNUM	PNUM	DEPTNUM	QUAN
S1	P1	D1	10
S1	P2	D1	10
S2	P1	D2	20

Quan hệ PART:

PNUM
P1
P2

Phép chia $SUPPLY \div PART$ cho kết quả là:

SNUM	DEPTNUM	QUAN
S1	D1	10

b. Các biểu thức đại số quan hệ

Bởi vì tất cả các phép toán đại số quan hệ đều có các toán hạng là quan hệ, do đó các phép toán đại số quan hệ có thể lồng nhau bằng cách sử dụng các dấu ngoặc và tạo thành một biểu thức đại số quan hệ. Các dấu ngoặc xác định thứ tự thực hiện của các phép toán.

Ví dụ 2.20 Xét truy vấn "Hãy cho biết các tên của các nhân viên làm việc trong một phòng ban có mã người quản lý là 373 nhưng họ không được lãnh lương nhiều hơn \$35.000". Truy vấn này có thể được trả lời bằng một biểu thức đại số quan hệ như sau:

$$\pi_{EMP\ NAME} ((EMP \triangleright \triangleleft_{DEPTNUM = DEPTNUM \ \sigma \ MGRNUM = 373} DEPT) - (\sigma_{SAL > 35000} EMP \triangleright \triangleleft_{DEPTNUM = DEPTNUM \ \sigma \ MGRNUM = 373} DEPT))$$

Một biểu thức đại số quan hệ khác của truy vấn này là:

$$\pi_{EMP\ NAME} (\sigma_{SAL \leq 35000} (EMP \triangleright \triangleleft_{DEPTNUM = DEPTNUM \ (\sigma_{MGRNUM = 373} DEPT))$$

2. Các phép tính quan hệ

Trong các ngôn ngữ phép tính quan hệ (*relational calculus-based language*), thay vì đặc tả kết quả có được bằng cách nào (*how*), chúng ta đặc tả kết quả là gì (*what*) bằng cách nêu ra mối liên hệ mà nó được dùng để tạo ra kết quả. Các ngôn ngữ phép tính quan hệ được chia thành hai nhóm: phép tính quan hệ bộ (*tuple relational calculus*) và phép tính quan hệ miền (*domain relational calculus*). Sự khác biệt giữa hai loại này dựa vào biến cơ bản (*primitive variable*) được sử dụng trong đặc tả truy vấn.

Các ngôn ngữ phép tính quan hệ có cơ sở lý thuyết vững chắc bởi vì chúng dựa trên luận lý vị từ cấp một (*first-order predicate logic*). Ngữ nghĩa của các công thức được xác định từ CSDL. CSDL quan hệ có thể được xem là sự tập hợp các bộ hoặc các miền. Phép tính

quan hệ bộ biểu diễn một biến trong công thức là một bộ của quan hệ, trong khi đó phép tính quan hệ miền biểu diễn một biến là một giá trị của miền.

a- Phép tính quan hệ bộ

Biến cơ bản được sử dụng trong phép tính quan hệ bộ là *biến bộ* (tuple variable) mà nó biểu diễn cho một bộ của quan hệ. Phép tính bộ là phép tính quan hệ ban đầu, do Codd nêu ra.

Trong phép tính quan hệ bộ, các truy vấn được đặc tả như sau:

$$\{t | F(t)\}$$

với t là một biến bộ và F là một công thức đúng đắn (well-formed formula). Công thức nguyên tố (atomic formula) có hai dạng:

1- Các biểu thức thành viên biến bộ (tuple-variable membership expression): Nếu t là một biến bộ lấy các bộ của quan hệ R , biểu thức "bộ t thuộc quan hệ R " là một công thức nguyên tố, thường được biểu diễn là $R.t$ hoặc $R(t)$.

2- Các điều kiện: Các điều kiện có thể được định nghĩa như sau:

- $s[A] \theta t[B]$ với s và t là các biến bộ, A và B là các thành phần tương ứng của s và t , θ là một trong các phép so sánh số học $<$, $>$, $=$, $\neq$, $\leq$ và $\geq$. Điều kiện này biểu diễn thành phần A của s có quan hệ θ với thành phần B của t ; ví dụ $s[\text{SAL}] > t[\text{SAL}]$.

- $s[A] \theta c$ với s , A và θ đã được định nghĩa ở trên và c là một hằng. Ví dụ: $s[\text{NAME}] = \text{'Nguyễn Trung Tiến'}$.

Lưu ý rằng, A được định nghĩa là một thành phần của biến bộ s . Bởi vì miền của s là một thể hiện quan hệ (relation instance), gọi là S , do đó thành phần A tương ứng với thuộc tính A của quan hệ S . Một cách tương tự cho B .

Có nhiều ngôn ngữ được dựa trên phép tính quan hệ bộ, ví dụ như SQL và QUEL và có nhiều hệ thống thương mại hóa, cung cấp các loại ngôn ngữ SQL khác nhau để làm phần giao tiếp với người sử dụng. Chúng ta sẽ tìm hiểu một loại SQL trong phần sau của chương này.

b- Phép tính quan hệ miền

Sự khác biệt cơ bản giữa ngôn ngữ quan hệ bộ với ngôn ngữ quan hệ miền là việc sử dụng biến miền (domain variable) trong ngôn ngữ quan hệ miền. Một biến miền lấy các giá trị trong một miền và đặc tả một thành phần của một bộ. Nói cách khác, phạm vi của một

biến miền bao gồm các miền mà trên đó quan hệ được định nghĩa. Các truy vấn được đặc tả theo dạng sau đây:

$$x_1, x_2, \dots, x_n \mid F(x_1, x_2, \dots, x_n)$$

với F là một công thức đúng dẫn và $x_1, x_2, \dots, x_n$ là các biến tự do (*free variable*).

Thành công của các ngôn ngữ phép tính quan hệ miền chủ yếu là QBE, đó là một ứng dụng trực quan của phép tính miền, được thiết kế chỉ dùng cho tương tác với các thiết bị đầu cuối trực quan và thân thiện với người sử dụng (*user friendly*). Ý niệm cơ bản là một ví dụ (*example*): người sử dụng tạo các câu truy vấn bằng cách cung cấp một ví dụ có thể có của câu trả lời. Việc đánh vào các tên quan hệ sẽ làm xuất hiện các lược đồ của chúng ở trên màn hình. Sau đó, người sử dụng đặc tả truy vấn bằng cách cung cấp các từ khóa cho các cột (các miền).

3- Ngôn ngữ SQL

Hệ quản trị CSDL (DBMS - *DataBase Management System*) cần phải có một ngôn ngữ cho phép người sử dụng truy xuất dữ liệu. Ngôn ngữ truy vấn có cấu trúc (SQL - *Structured Query Language*) là một ngôn ngữ được sử dụng trong hầu hết các hệ CSDL quan hệ.

Ngôn ngữ SQL được phát triển trong một hệ quản trị CSDL mẫu - System R - bởi IBM vào giữa những năm 1970. Năm 1979, Oracle Corporation đã giới thiệu về việc thực hiện SQL thương mại hóa đầu tiên.

Các đặc điểm của SQL

- SQL là một ngôn ngữ tựa tiếng Anh (*English-like language*). Nó sử dụng các từ như *select*, *insert*, *delete* trong tập lệnh.

- SQL là một ngôn ngữ phi thủ tục (*non-procedural language*): chúng ta chỉ ra các thông tin nào cần thiết (*what*) mà không cần phải chỉ ra cách thực hiện như thế nào (*how*) để có được các thông tin này. Nói cách khác, SQL không yêu cầu chỉ ra phương pháp truy xuất dữ liệu. Tất cả các lệnh SQL đều sử dụng bộ tối ưu hóa truy vấn (*query optimizer*) - là một thành phần của hệ quản trị CSDL quan hệ RDBMS - để xác định các cách truy xuất nhanh nhất lấy dữ liệu cần thiết. Đặc điểm này làm cho chúng ta tập trung dễ dàng hơn vào việc có được kết quả mong muốn.

• SQL xử lý các tập hợp mẫu tin hơn là mỗi lần một mẫu tin. Dạng chung nhất của tập hợp các mẫu tin là một *bảng* (*table*).

• SQL có thể được dùng cho nhiều người sử dụng, bao gồm người quản trị dữ liệu (DBA – *DataBase Administrator*), người lập trình ứng dụng, người quản lý và người sử dụng cuối cùng (*end user*).

• SQL cung cấp nhiều lệnh để thực hiện nhiều công việc khác nhau, bao gồm:

- Truy vấn dữ liệu.
- Thêm vào, cập nhật và hủy bỏ các hàng của bảng.
- Tạo lập, thay đổi và hủy bỏ các đối tượng CSDL.
- Điều khiển truy xuất đến CSDL và các đối tượng CSDL.
- Bảo đảm tính nhất quán của CSDL.

Các hệ quản trị CSDL trước kia thường dùng một ngôn ngữ riêng biệt cho mỗi loại công việc nêu trên. SQL hợp nhất các công việc này trong một ngôn ngữ.

SQL đã trở thành ngôn ngữ chuẩn cho các CSDL quan hệ. Năm 1986, Viện Tiêu chuẩn quốc gia Mỹ (ANSI – *American National Standards Institute*) đề nghị SQL là ngôn ngữ chuẩn cho các hệ quản trị CSDL quan hệ (RDBMS). Tổ chức Tiêu chuẩn quốc tế (ISO – *International Standards Organisation*) cũng đã đề nghị SQL là ngôn ngữ chuẩn cho các RDBMS. Đa số các RDBMS hỗ trợ một vài dạng SQL và hầu hết các nhà cung cấp phần mềm cũng đồng ý với chuẩn ANSI.

Tập lệnh SQL

Các lệnh SQL có thể được phân chia thành ba loại: ngôn ngữ định nghĩa dữ liệu, ngôn ngữ thao tác dữ liệu và ngôn ngữ điều khiển dữ liệu.

Ngôn ngữ định nghĩa dữ liệu (DDL – *Data Definition Language*) bao gồm các lệnh dùng để tạo lập, thay đổi và hủy bỏ các bảng.

Ngôn ngữ thao tác dữ liệu (DML – *Data Manipulation Language*) bao gồm các lệnh dùng để truy vấn, thêm vào, cập nhật và xóa bỏ dữ liệu trong CSDL. Các lệnh này có thể được đưa vào trực tiếp (chế độ tương tác) và kết quả được trả về ngay sau khi thực hiện lệnh này, hoặc có thể đưa vào trong các chương trình.

Ngôn ngữ điều khiển dữ liệu (DCL – Data Control Language) bao gồm các lệnh dùng để hỗ trợ cho DBA để điều khiển CSDL, đó là các lệnh dùng để gán và hủy bỏ các quyền (*privilege*) truy xuất CSDL và các đối tượng riêng biệt trong CSDL và lưu trữ hoặc hủy bỏ các giao dịch (*transaction*) ảnh hưởng đến CSDL.

Lệnh	Mô tả
SELECT	Đây là lệnh thường dùng nhất, nó được dùng để lấy dữ liệu từ CSDL.
INSERT UPDATE DELETE	Ba lệnh này được dùng để thêm vào các hàng mới, cập nhật và xóa bỏ các hàng của các bảng trong CSDL. Các lệnh này còn được gọi là các lệnh của ngôn ngữ thao tác dữ liệu (DML).
CREATE ALTER DROP	Ba lệnh này được dùng để tạo lập, thay đổi và xóa bỏ một cấu trúc dữ liệu nào đó ví dụ như bảng, khung nhìn (<i>view</i>), chỉ mục (<i>index</i>). Các lệnh này còn được gọi là các lệnh của ngôn ngữ định nghĩa dữ liệu (DDL).
GRANT REVOKE	Hai lệnh này được dùng để gán và hủy bỏ các quyền truy xuất đến CSDL và các cấu trúc bên trong. Các lệnh này còn được gọi là các lệnh của ngôn ngữ điều khiển dữ liệu (DCL).

Phần này giới thiệu các lệnh Oracle SQL, bao gồm: SELECT, INSERT, UPDATE, DELETE, CREATE TABLE và DROP TABLE.

Lệnh SELECT

Lệnh SELECT dùng để lấy dữ liệu trong CSDL, nó thực hiện các phép toán của đại số quan hệ: phép tích, phép chọn, phép chiếu, phép kết.

Ngữ pháp của lệnh SELECT như sau:

```
SELECT [DISTINCT] <list of expressions> [INTO <list of variables>]
FROM <list of tables>
[WHERE <row conditions>]
[GROUP BY <list of expressions>]
[HAVING <group of rows conditions>]]
[ORDER BY <list of expressions>];
```

Mệnh đề SELECT thực hiện phép chiếu của đại số quan hệ. *List of expression* là danh sách các biểu thức được chọn, các biểu thức này được phân cách nhau bởi dấu phẩy. Biểu thức có thể là một hằng, một biến (hoặc cột), hoặc sự kết hợp giữa các hằng, các biến với các phép

toán. Mỗi biểu thức có thể có một bí danh (*alias*) đứng ngay phía sau, được gọi là *bí danh cột* (*column alias*); bí danh cột chỉ được sử dụng trong mệnh đề SELECT. Nếu *list of expressions* là dấu * (*asterisk*) thì tất cả các cột được chọn. Cột có thể có dạng *table_name.column_name*.

Mệnh đề FROM thực hiện phép tích của đại số quan hệ, dùng để chỉ ra các bảng chứa dữ liệu cần lấy ra. *List of tables* là danh sách các bảng được phân cách nhau bởi dấu phẩy. Mỗi bảng có thể có bí danh đứng ngay phía sau.

Ví dụ 2.21 Cho biết tất cả các nhân viên:

```
SELECT empnum, name, sal, tax, mgrnum, deptnum  
FROM emp;
```

hoặc:

```
SELECT *  
FROM emp;
```

Trình bày tiền lương một năm của tất cả nhân viên, cột này được đặt tên là ANNSAL:

```
SELECT empnum, name, sal * 12 ANNSAL  
FROM emp;
```

Từ khóa DISTINCT loại bỏ các giá trị trùng trong kết quả của truy vấn. Nếu có nhiều cột được chọn thì DISTINCT ảnh hưởng đến toàn bộ các cột này. Từ khóa DISTINCT phải được đặt ngay sau từ khóa SELECT.

Ví dụ 2.22 Cho biết các mã phòng của tất cả các nhân viên:

```
SELECT DISTINCT deptnum  
FROM emp;
```

Cho biết các tiền lương của mỗi phòng:

```
SELECT DISTINCT deptnum, sal  
FROM emp;
```

Mệnh đề INTO chỉ được sử dụng trong PL/SQL, dùng để gán giá trị của các biểu thức cho các biến theo thứ tự tương ứng. *List of variables* là danh sách các biến được phân cách nhau bởi dấu phẩy.

Mệnh đề WHERE thực hiện các phép chọn, phép kết của đại số quan hệ. *Row conditions* là các điều kiện được xét trên mỗi hàng, các hàng nào thỏa mãn các điều kiện này thì được đưa vào kết quả của truy vấn.

Ví dụ 2.23 Gán mã và tên phòng của phòng có mã 10 cho các biến *v_deptnum*, *v_name* theo thứ tự tương ứng:

```
SELECT deptnum, name INTO v_deptnum, v_name
FROM emp
WHERE deptnum = 10;
```

Cho biết mã nhân viên, tên nhân viên và tên phòng của các nhân viên thuộc các phòng có mã lớn hơn 10:

```
SELECT empnum, e.name, dept.name
FROM emp e, dept
WHERE e.deptnum = dept.deptnum
AND dept.deptnum > 10;
```

trong đó:

e – bí danh của bảng EMP

e.deptnum = *dept.deptnum* – điều kiện kết

dept.deptnum > 10 – điều kiện chọn.

Mệnh đề GROUP BY được dùng để phân chia các hàng của một bảng thành các nhóm nhỏ hơn. Các hàm nhóm có thể được sử dụng để trả về thông tin chung cho mỗi nhóm. Mỗi nhóm chỉ xuất hiện ở một hàng trong kết quả của truy vấn. Trong trường hợp lệnh SELECT có cả hai mệnh đề WHERE và GROUP BY thì các hàng sẽ được chọn bởi điều kiện của mệnh đề WHERE, rồi sau đó phân chia các hàng được chọn thành các nhóm. Các cột trong mệnh đề GROUP BY không bắt buộc phải có trong mệnh đề SELECT. Tuy nhiên, để cho kết quả của truy vấn là có ngữ nghĩa thì các cột của mệnh đề GROUP BY nên có trong mệnh đề SELECT. Khi có mệnh đề GROUP BY, tất cả các cột có trong mệnh đề SELECT nếu chúng không nằm trong hàm nhóm thì chúng phải có trong mệnh đề GROUP BY. Trong trường hợp lệnh SELECT không có mệnh đề GROUP BY thì toàn bộ bảng được xem là một nhóm. Do đó, nếu mệnh đề SELECT có chứa hàm nhóm thì chúng ta không thể lấy được chi tiết của mỗi hàng.

Ví dụ 2.24 Tính tiền lương trung bình của mỗi phòng:

```
SELECT deptnum, AVG(sal)
FROM emp
GROUP BY deptnum;
```

Tính tiền lương trung bình của mỗi phòng có mã lớn hơn 10:

```
SELECT deptnum, AVG(sal)
FROM emp
WHERE deptnum > 10
GROUP BY deptnum;
```

Tính số lượng trung bình của mỗi mặt hàng tại mỗi phòng:

```
SELECT deptnum, pnum, AVG(quan)
FROM supply
GROUP BY deptnum, pnum;
```

Các lệnh SELECT sau đây là không hợp lệ:

```
SELECT deptnum, MIN(sal)
FROM emp;
SELECT deptnum, name, MAX(sal)
FROM emp
GROUP BY deptnum;
```

Mệnh đề HAVING được dùng để xác định các nhóm được đưa vào kết quả của truy vấn. *Group of rows conditions* là các điều kiện được xét cho mỗi nhóm. Mệnh đề HAVING có thể đứng trước mệnh đề GROUP BY, nhưng mệnh đề GROUP BY nên đứng trước để cho dễ hiểu. Các nhóm được tạo ra và các hàm nhóm được tính toán trước khi thực hiện mệnh đề HAVING để chọn ra các nhóm. Mệnh đề WHERE không được dùng để chọn các nhóm. Nếu lệnh SELECT có các mệnh đề WHERE, GROUP BY và HAVING thì thứ tự thực hiện của các mệnh đề này là WHERE (để chọn các hàng), kế tiếp GROUP BY (để phân chia nhóm) và sau cùng là HAVING (để chọn các nhóm).

Ví dụ 2.25 Cho biết tiền lương trung bình của các phòng có nhiều hơn ba nhân viên:

```
SELECT deptnum, AVG(sal)
FROM emp
GROUP BY deptnum
HAVING COUNT(*) > 3;
```

Cho biết tiền lương lớn nhất của các phòng mà tiền lương lớn nhất này lớn hơn hoặc bằng 35000:

```
SELECT deptnum, MAX(sal)
FROM emp
HAVING MAX(sal) >= 35000
GROUP BY deptnum;
```


Các lệnh SELECT sau đây là không hợp lệ:

```
SELECT deptnum, AVG(sal)
FROM emp
WHERE AVG(sal) >= 35000
GROUP BY deptnum;
```

Thông thường, thứ tự của các hàng được trả về trong kết quả của truy vấn là không xác định. Mệnh đề ORDER BY có thể được dùng để sắp thứ tự các hàng này. Mệnh đề ORDER BY luôn luôn là mệnh đề cuối cùng của lệnh SELECT. Thứ tự ngầm định là tăng dần (ASC - ascending), từ khóa DESC (descending) đứng ngay sau tên cột dùng để chỉ định thứ tự giảm dần. Các cột trong mệnh đề ORDER BY không bắt buộc phải có trong mệnh đề SELECT.

Ví dụ 2.26 Sắp thứ tự theo tên nhân viên tăng dần:

```
SELECT empnum, name, sal * 12, deptnum
FROM emp
ORDER BY name;
```

Sắp thứ tự theo tiền lương giảm dần:

```
SELECT empnum, name, sal
FROM emp
ORDER BY sal DESC;
```

Sắp thứ tự theo mã phòng tăng dần và tiền lương giảm dần:

```
SELECT empnum, name, sal
FROM emp
ORDER BY deptnum, sal DESC;
```

Các phép toán: Lệnh SELECT có chứa các phép toán và các hàm, chúng được ghi trong các mệnh đề SELECT, WHERE, GROUP BY, HAVING và ORDER BY.

Các phép toán số học (arithmetic operator) gồm có:

- + cộng
- trừ
- * nhân
- / chia

Phép nhân và phép chia được thực hiện trước phép cộng và phép trừ.

Các phép toán so sánh (comparison operator / relational operator) gồm có:

=	bằng
<> hoặc !=	khác, không bằng
>	lớn hơn
>=	lớn hơn hoặc bằng
<	nhỏ hơn
<=	nhỏ hơn hoặc bằng

Các phép toán luận lý (logical operator) gồm có:

NOT	phủ định
AND	và
OR	hoặc

Các phép toán SQL (SQL operator) gồm có:

BETWEEN <expr1> AND <expr2>: giữa hai giá trị

IN (<list of values>): một trong các giá trị

LIKE <skeleton> có dạng mẫu:

% là chuỗi bất kỳ,

_ là ký tự bất kỳ

IS NULL: là giá trị null

IS NOT NULL: là giá trị khác null

Các hàm nhóm gồm có:

AVG([DISTINCT | ALL] n): giá trị trung bình của n, bỏ qua các giá trị rỗng.

COUNT([DISTINCT | ALL] expr *): số hàng mà expr có giá trị khác rỗng. '*' làm cho COUNT đếm tất cả các hàng được chọn, bao gồm các hàng trùng nhau và các hàng có giá trị rỗng.

MAX([DISTINCT | ALL] expr): giá trị lớn nhất của expr.

MIN([DISTINCT | ALL] expr): giá trị nhỏ nhất của expr.

SUM([DISTINCT | ALL] n): tổng giá trị của n, bỏ qua các giá trị rỗng.

STDDEV([DISTINCT | ALL] n): độ lệch chuẩn (standard deviation) của n, bỏ qua các giá trị rỗng.

VARIANCE([DISTINCT | ALL] n): phương sai (variance) của n, bỏ qua các giá trị rỗng.

Tất cả các hàm nêu trên đều thao tác trên một số hàng hoặc trên một bảng, và do đó chúng được gọi là các hàm nhóm hoặc các hàm kết hợp.

DISTINCT làm cho hàm nhóm chỉ xét các giá trị khác biệt; ALL làm cho hàm nhóm xét mọi giá trị kể cả các giá trị trùng nhau. Mặc nhiên là ALL.

Kiểu dữ liệu của các tham số có thể là CHAR, NUMBER hoặc DATE tùy theo *expr*.

Tất cả các hàm nhóm, ngoại trừ COUNT(*), đều bỏ qua các giá trị rỗng.

Hàm NVL thường được dùng để xử lý các giá trị rỗng.

Ví dụ 2.27 Cho biết các nhân viên có tiền lương từ 10000 đến 30000:

```
SELECT empnum, name, sal
FROM emp
WHERE sal BETWEEN 10000 AND 30000;
```

Cho biết các nhân viên có mã sếp là một trong các mã 100, 200, 300:

```
SELECT empnum, name, mgrnum
FROM emp
WHERE mgrnum IN (100, 200, 300);
```

Cho biết các nhân viên có tên bắt đầu bằng chữ 'S':

```
SELECT empnum, name
FROM emp
WHERE name LIKE 'S%';
```

Cho biết các nhân viên có tên gồm bốn ký tự:

```
SELECT empnum, name
FROM emp
WHERE name LIKE '____';
```

Cho biết các nhân viên có sếp:

```
SELECT empnum, name
FROM emp
WHERE mgrnum IS NOT NULL;
```

Thứ tự thực hiện ưu tiên của các phép toán trong một biểu thức được liệt kê theo độ ưu tiên giảm dần, các phép toán có cùng độ ưu tiên sẽ được thực hiện từ trái qua phải:

- Biểu thức con trong dấu ngoặc
- Các phép toán số học *, /
- Các phép toán số học +, -
- Các phép toán so sánh và các phép toán SQL:

=, !=, <, <=, >, >=, BETWEEN... AND, IN, LIKE, IS NULL

- NOT
- AND
- OR

Hai lệnh SELECT có thể được kết nối với nhau bằng các phép toán tập hợp bao gồm phép hợp (*union*), phép giao (*intersection*) và phép hiệu (*minus*) và có dạng như sau:

```
SELECT statement_1  
UNION [ALL] | INTERSECT | MINUS  
SELECT statement_2;
```

trong đó:

UNION thực hiện phép hợp kết quả của hai truy vấn.

INTERSECT thực hiện phép giao kết quả của hai truy vấn.

MINUS thực hiện phép hiệu kết quả của hai truy vấn.

Từ khóa ALL cho phép các hàng trong kết quả có thể trùng nhau.

Ví dụ 2.28 Cho biết tên của các nhân viên thuộc các phòng có mã 10, 30:

```
SELECT name  
FROM emp  
WHERE deptnum = 10  
UNION  
SELECT name  
FROM emp  
WHERE deptnum = 30;
```

Cho biết tên của các nhân viên (một tên có thể xuất hiện ở nhiều hàng) thuộc các phòng có mã 10, 30:

```
SELECT name  
FROM emp  
WHERE deptnum = 10  
UNION ALL  
SELECT name  
FROM emp  
WHERE deptnum = 30;
```

Cho biết tên của các nhân viên thuộc cả hai phòng có mã 10, 30:

```
SELECT name  
FROM emp  
WHERE deptnum = 10
```

```

INTERSECT
SELECT name
FROM emp
WHERE deptnum = 30;

```

Cho biết tên của các nhân viên thuộc phòng có mã 10 mà không có trong phòng có mã 30:

```

SELECT name
FROM emp
WHERE deptnum = 10
MINUS
SELECT name
FROM emp
WHERE deptnum = 30;

```

Xử lý giá trị rỗng

Hàm NVL dùng để đổi giá trị rỗng (*null value*) thành một giá trị khác rỗng (*non-null value*). Hàm này có hai tham số: một biểu thức và một giá trị khác rỗng. Nếu giá trị của các cột trong một biểu thức là *null* thì giá trị của biểu thức này là *null*.

Ví dụ 2.29 Lệnh sau đây có các giá trị *null* trong cột ANNUAL_SAL nếu cột *sal* có chứa các giá trị *null*:

```

SELECT empnum, name, sal * 12 ANNUAL_SAL
FROM emp;

```

Trong lệnh sau đây, các giá trị *null* trong cột ANNUAL_SAL được đổi thành giá trị 0:

```

SELECT empnum, name, NVL(sal * 12, 0) ANNUAL_SAL
FROM emp;

```

Lệnh INSERT

Lệnh INSERT được dùng để thêm các hàng vào một bảng. Ngữ pháp của lệnh INSERT như sau:

```

INSERT INTO <table_name> [(<list of columns>)]
VALUES (<list of expressions>);

```

trong đó:

table_name – tên bảng cần được thêm vào

list of columns – danh sách các cột được phân cách nhau bởi dấu phẩy;

list of expressions – danh sách các biểu thức được phân cách nhau bởi dấu phẩy; kiểu dữ liệu của các biểu thức phải có cùng kiểu dữ liệu của các cột theo thứ tự tương ứng.

Lệnh INSERT ở trên được dùng để thêm một hàng mới vào một bảng, giá trị của các biểu thức được gán cho các cột theo thứ tự tương ứng. Các cột không được gán giá trị thì các cột này sẽ có giá trị *null*.

Trong trường hợp chúng ta muốn lấy kết quả của một truy vấn đưa thêm vào một bảng, thì chúng ta sử dụng lệnh INSERT có ngữ pháp sau đây:

```
INSERT INTO table_name [(<list of columns>)]
```

```
SELECT statement;
```

Ví dụ 2.30 Lệnh sau đây thêm một phòng mới có mã là 50, tên là 'MARKETING', thuộc vùng 'NORTH' vào bảng DEPT:

```
INSERT INTO dept (deptnum, name, area)
```

```
VALUES (50, 'MARKETING', 'NORTH');
```

Lệnh sau đây lấy kết quả của truy vấn đưa thêm vào bảng EMP_HISTORY (bảng này đã có sẵn trong CSDL và có cấu trúc giống với bảng EMP):

```
INSERT INTO emp_history (empnum, name, sal, tax, mgrnum, deptnum)
```

```
SELECT empnum, name, sal, tax, mgrnum, deptnum
```

```
FROM emp
```

```
WHERE deptnum = 10;
```

Lệnh UPDATE

Lệnh UPDATE được dùng để cập nhật dữ liệu của các hàng của một bảng. Ngữ pháp của lệnh UPDATE như sau:

```
UPDATE <table name> [<alias>]
```

```
SET <column name> = <value>
```

```
[, <column name> = <value> ...]
```

```
[WHERE <row conditions>];
```

trong đó:

table name – tên bảng cần cập nhật

alias – bí danh của bảng

column name – tên cột cần cập nhật

value – có thể là một biểu thức hoặc là một truy vấn con (lệnh SELECT), và có cùng kiểu dữ liệu với *column name*.

row conditions – điều kiện dùng để chọn các hàng cần được cập nhật, nếu không có mệnh đề WHERE thì cập nhật tất cả các hàng của bảng.

Ví dụ 2.31 Cập nhật dữ liệu của nhân viên có mã 100:

```
UPDATE emp
SET sal = 30000,
    tax = 1000
WHERE empnum = 100;
```

Cập nhật tiền lương của các nhân viên có trong bảng EMP_SALARY:

```
UPDATE emp
SET sal = (SELECT sal
           FROM emp_salary s
           WHERE s.empnum = emp.empnum)
WHERE empnum IN (SELECT empnum FROM emp_salary);
```

Lệnh DELETE

Lệnh DELETE được dùng để xóa các hàng của một bảng. Ngữ pháp của lệnh DELETE như sau:

```
DELETE [FROM] <table name> [<alias>]
[WHERE <row conditions>];
```

trong đó:

table_name – tên bảng cần xóa dữ liệu

alias – bí danh của bảng

row conditions – điều kiện dùng để chọn các hàng cần xóa, nếu không có mệnh đề WHERE thì xóa tất cả các hàng bảng.

Ví dụ 2.32 Xóa các nhân viên thuộc phòng có mã 10:

```
DELETE FROM emp
WHERE deptnum = 10;
```

Xóa tất cả các nhân viên, kết quả là bảng EMP rỗng:

```
DELETE FROM emp;
```

Lệnh CREATE TABLE

Lệnh CREATE TABLE được dùng để tạo lập một bảng mới. Ngữ pháp của lệnh CREATE TABLE như sau:

```
CREATE TABLE <table name>
(<column name> <type> (<size>)
[, <column name> <type> (<size>) ...]);
```

trong đó:

table name – tên bảng mới

column name – tên cột của bảng

type – tên kiểu dữ liệu của cột

size – chiều dài của cột

Kiểu dữ liệu	Diễn giải
VARCHAR2(<i>n</i>)	Chuỗi ký tự có chiều dài thay đổi đến <i>n</i> ký tự. Chiều dài tối đa là 2000 ký tự.
CHAR(<i>n</i>)	Chuỗi ký tự có chiều dài cố định là <i>n</i> ký tự. Mặc nhiên là 1, tối đa là 255.
NUMBER	Số dấu chấm động có 38 ký số có nghĩa.
NUMBER(<i>n</i>)	Số nguyên có <i>n</i> ký số.
NUMBER(<i>n,m</i>)	Số thập phân có <i>n</i> ký số (không vượt quá 38 ký số) với <i>m</i> số số lẻ.
DATE	Giá trị ngày từ 1/1/4712 trước Công nguyên đến 31/12/4712 sau Công nguyên, có chứa thông tin về giờ.
LONG	Chuỗi ký tự có chiều dài thay đổi đến 2 gigabyte hoặc $2^{31} - 1$. Chỉ có một cột kiểu LONG trong một bảng.
RAW và LONGRAW	Tương đương với VARCHAR2 và LONG theo thứ tự tương ứng, nhưng dùng để lưu dữ liệu dạng byte hoặc dữ liệu nhị phân như hình ảnh đồ họa hay âm thanh kỹ thuật số.

Ví dụ 2.33

- Tạo lập bảng DEPT:

```
CREATE TABLE dept
  (deptnum NUMBER(2) NOT NULL,
   name VARCHAR2(20),
   area VARCHAR2(10));
```

Tạo lập bảng EMP:

```
CREATE TABLE emp
  (empnum NUMBER(4) NOT NULL,
   name VARCHAR2(20),
   sal NUMBER (7,2),
   tax NUMBER(7,2),
   mgrnum NUMBER(4),
   deptnum NUMBER(2) NOT NULL);
```

Chúng ta có thể tạo lập một bảng mới từ các bảng khác. Ngữ pháp của lệnh CREATE TABLE như sau:

```
CREATE TABLE <table name> [(<column name>, ...)]
AS SELECT statement;
```

trong đó:

table name - tên bảng mới

column name - tên cột của bảng

Ví dụ 2.34 Tạo lập bảng DEPT10 từ bảng EMP, bao gồm các nhân viên thuộc phòng có mã 10:

```
CREATE TABLE dept10
AS
SELECT empnum, name, sal, tax, mgrnum, deptnum
FROM emp
WHERE deptnum = 10;
```

Tạo lập bảng EMP_SAL từ bảng EMP, bao gồm các nhân viên có tiền lương lớn hơn 10000:

```
CREATE TABLE emp_sal (empnum, name, salary)
AS
SELECT empnum, name, sal
FROM emp
WHERE sal > 10000;
```

Lệnh DROP TABLE

Lệnh DROP TABLE được dùng để hủy bỏ một bảng trong CSDL. Ngữ pháp của lệnh DROP TABLE như sau:

```
DROP TABLE <table name>;
```

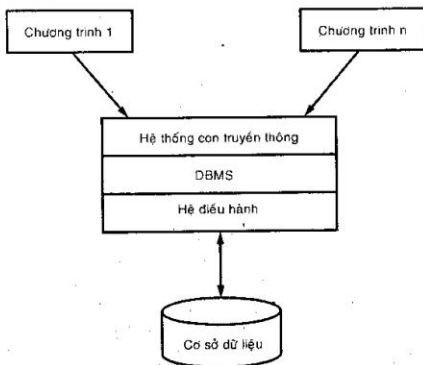
trong đó *table name* là tên bảng cần hủy bỏ.

Ví dụ 2.35 Hủy bỏ bảng EMP_SAL trong CSDL:

```
DROP TABLE emp_sal;
```

2.1.5 Hệ quản trị CSDL quan hệ

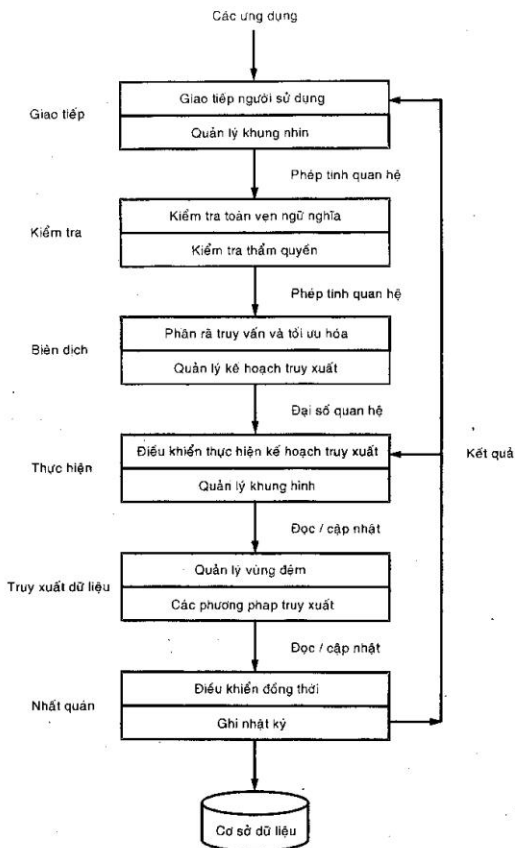
Hệ quản trị CSDL quan hệ (RDBMS – *Relational DataBase Management System*) là một hệ thống quản lý dữ liệu mà hệ thống này cung cấp mô hình dữ liệu quan hệ (*relational data model*), dữ liệu được chứa trong các bảng (*table*) và các mối quan hệ của dữ liệu được biểu diễn bởi các giá trị chung trong các bảng có quan hệ với nhau. Một DBMS giao tiếp với hai thành phần khác: hệ thống con truyền thông và hệ điều hành. Kiến trúc chung của một DBMS (tập trung) được trình bày trong hình 2.5. Hệ thống con truyền thông cho phép DBMS giao tiếp với các hệ thống con khác để liên lạc với các ứng dụng. Ví dụ, thiết bị đầu cuối cần liên lạc với DBMS để chạy các giao dịch tương tác. Hệ điều hành cung cấp giao tiếp giữa DBMS và các nguồn tài nguyên của máy tính (bộ xử lý, bộ nhớ, các ổ đĩa,...).



Hình 2.5 Kiến trúc chung của một DBMS tập trung

Các chức năng của một DBMS có thể được phân lớp như trong hình 2.6, trong đó các mũi tên cho biết chiều của dòng dữ liệu và dòng điều khiển. Theo cách tiếp cận từ trên xuống, các lớp gồm giao tiếp, điều khiển, biên dịch, thực hiện, truy xuất dữ liệu và quản lý nhất quán.

Lớp giao tiếp dùng để quản lý giao tiếp với các ứng dụng. Có thể có nhiều giao tiếp chẳng hạn như SQL nhúng trong COBOL và QBE (*Query By Example*). Các chương trình ứng dụng được thực hiện trên các khung nhìn ngoài (*external view*) của CSDL. Đối với một ứng dụng, khung nhìn có ích cho việc biểu diễn sự hiểu biết riêng biệt về CSDL (được dùng chung cho nhiều ứng dụng). Một khung nhìn quan hệ là một quan hệ ảo được lấy từ các quan hệ thường (*base relation*) bằng cách áp dụng các phép toán đại số quan hệ. Quản lý khung nhìn bao gồm việc chuyển đổi truy vấn của người sử dụng từ dữ liệu ngoài (*external data*) thành dữ liệu ý niệm (*conceptual data*) trong các quan hệ thường.



Hình 2.6 Các lớp chức năng của DBMS quan hệ

Lớp kiểm tra dùng để kiểm tra truy vấn bằng cách đưa thêm vào các vị từ toàn vẹn ngữ nghĩa và các vị từ thẩm quyền. Các ràng buộc toàn vẹn ngữ nghĩa và các thẩm quyền được đặc tả dưới dạng phép tính quan hệ. Phần xuất của lớp này là một truy vấn dưới dạng phép tính quan hệ.

Lớp xử lý truy vấn dùng để biến đổi truy vấn thành một chuỗi tối ưu, gồm các phép toán mức thấp hơn. Lớp này liên quan đến hiệu suất. Lớp này phân rã truy vấn thành một cây bao gồm các phép toán đại số quan hệ và thử tìm thứ tự thực hiện tối ưu cho các phép toán này. Kết quả được lưu trong một kế hoạch truy xuất (*access plan*). Phần xuất của lớp này là một truy vấn, được biểu diễn dưới dạng đại số quan hệ (hoặc dưới dạng mã mức thấp).

Lớp thực hiện dùng để điều khiển sự thực hiện của kế hoạch truy xuất, bao gồm quản lý giao dịch (hoàn tất, bắt đầu lại) và đồng bộ hóa các phép toán đại số quan hệ. Lớp này diễn dịch các phép toán quan hệ bằng cách gọi lớp truy xuất dữ liệu thông qua các yêu cầu đọc và cập nhật.

Lớp truy xuất dữ liệu dùng để quản lý các cấu trúc dữ liệu biểu diễn các quan hệ (các tập tin, các chỉ mục). Lớp này cũng quản lý các vùng đệm (*buffer*) bằng cách lưu trữ các dữ liệu được truy xuất thường xuyên nhất. Sử dụng cẩn thận lớp này sẽ làm giảm số lần truy xuất đĩa để lấy và ghi dữ liệu.

Lớp nhất quán dùng để quản lý điều khiển đồng thời và ghi nhật ký cho các yêu cầu cập nhật. Lớp này cho phép khôi phục giao dịch, hệ thống và các phương tiện sau khi khắc phục hư hỏng.

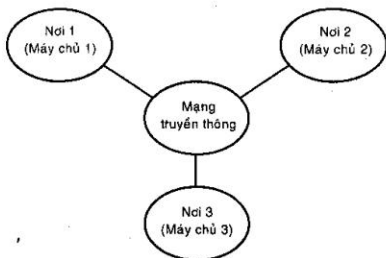
2.2 TỔNG QUAN VỀ MẠNG MÁY TÍNH

Mô hình của một mạng máy tính mà chúng ta sẽ xét trong quyển sách này được trình bày trong hình 2.7, các máy tính có khả năng thực hiện công việc một cách độc lập và được kết nối với nhau bởi một mạng truyền thông. Các máy tính được gọi là các máy chủ (*host*) trong các tài liệu về mạng máy tính, chúng ta sẽ sử dụng thuật ngữ *nơi* (*site*) để biểu diễn cho máy tính chủ (*host computer*). Mạng truyền thông bao gồm các liên kết truyền thông (*communication link*) của nhiều loại khác nhau (dây điện thoại, cáp đồng trục, liên kết vệ

tin, ...) và thường bao gồm nhiều máy tính. Các máy tính chuyên thực hiện các chức năng truyền thông và không phải là các máy chủ thì sẽ không bao giờ được xét đến trong quyển sách này. Chúng chỉ có ích cho việc thực hiện mạng truyền thông.

Mạng truyền thông cung cấp một khả năng cơ bản là một quá trình đang chạy tại một nơi bất kỳ có thể gửi một thông điệp cho một quá trình đang chạy tại một nơi khác của mạng.

Lưu ý rằng, khả năng cơ bản này độc lập với cấu trúc thực sự của mạng truyền thông: hai nơi bất kỳ đều có thể liên lạc với nhau.



Hình 2.7 Mô hình mạng máy tính

Các thông số đặc trưng cho chức năng cơ bản ở trên và sẽ được xét trong phần còn lại của quyển sách này là:

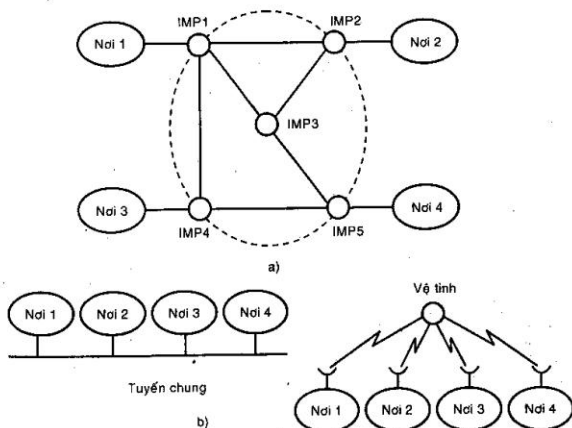
- *Trì hoãn của thông điệp được truyền cho nơi đến*: khi việc sử dụng mạng còn thấp, trì hoãn phụ thuộc chủ yếu vào các đặc tính hiệu suất của các thành phần tạo nên mạng và do đó có thể xem là đã biết trước về mặt thống kê. Tuy nhiên, nếu mạng được sử dụng quá nhiều, thì trì hoãn sẽ lâu hơn do thông điệp phải chờ truyền các thông điệp khác trước thông điệp này; trong trường hợp này, cần phải phân tích cách truyền các thông điệp để đánh giá trì hoãn.

- *Chi phí truyền thông điệp*: trong quyển sách này, chúng ta giả sử có một chi phí cố định cho mỗi thông điệp, cộng thêm chi phí tỉ lệ với chiều dài của thông điệp.

- *Độ tin cậy của mạng*: chủ yếu là xác suất mà thông điệp được gửi đến đúng nơi đến.

Trong nhiều ứng dụng, một quá trình tại một nơi nào đó cần gửi cùng thông điệp cho tất cả các nơi khác, tác vụ này được gọi là *truyền rộng (broadcast)*. Có thể thực hiện việc truyền rộng bằng cách đơn giản là gửi từng thông điệp riêng biệt cho các nơi của mạng, trong trường hợp này, tác vụ truyền rộng là sự lặp lại của tác vụ gửi thông điệp cơ bản. Tuy nhiên, trong nhiều loại mạng truyền thông, việc truyền rộng một thông điệp là ít tốn kém hơn so với việc gửi nhiều lần cùng một thông điệp. Trong thực tế, có những mạng truyền thông mà chi phí của việc truyền rộng một thông điệp bằng chi phí gửi cùng thông điệp này đến đúng một nơi.

1- Các loại mạng truyền thông



Hình 2.8 Các loại mạng truyền thông

a) Mạng điểm - điểm; b) Hai loại mạng truyền thông

Có nhiều cách phân loại các mạng, ví dụ xét mạng trong hình 2.8a. Một mạng truyền thông bao gồm các bộ xử lý chuyên dụng, được gọi là các bộ xử lý thông điệp giao tiếp IMP (*Interface Message Processor*) mà chúng được kết nối từng cặp với nhau bởi các kênh

truyền thông, ví dụ như các đường điện thoại. Mỗi nơi được kết nối với một IMP. Khi một nơi gửi một thông điệp đến một nơi khác, thông điệp này được gửi đến một IMP để lưu trữ và sau đó IMP này gửi nó đến một IMP khác mà tại đó thông điệp này cũng được lưu trữ và sau đó gửi nó đến một IMP khác nữa, và cứ như thế cho đến khi cuối cùng thông điệp đến IMP của nơi đến. Loại mạng này được gọi là *mạng điểm-điểm (point-to-point network)* hoặc *mạng lưu trữ-và-chuyển tiếp (store-and-forward network)*. Trong mạng điểm-điểm, các IMP có nhiệm vụ chọn đường đi để truyền một thông điệp. Ví dụ, trong hình 2.8a, nếu nơi 1 gửi một thông điệp đến nơi 4, thông điệp có thể đi theo đường IMP1 - IMP3 - IMP5, hoặc đường IMP1 - IMP2 - IMP3 - IMP5 hoặc nhiều đường khác. Công việc chọn một đường đi được gọi là *chọn tuyến (routing)*. Theo định nghĩa của khả năng cơ bản của mạng truyền thông thì chúng ta ngầm định một mạng có khả năng thực hiện việc chọn tuyến này, bởi vì các nơi gửi chỉ đơn giản cho biết nơi đến của mỗi thông điệp.

Trong mạng điểm-điểm, chi phí truyền rộng cho một thông điệp đến nhiều nơi sẽ tương đương với việc gửi nhiều thông điệp riêng biệt. Cuối cùng, lưu ý rằng sự tách biệt giữa các nơi với các IMP chỉ xét về mặt luận lý và không nhất thiết về mặt vật lý: cùng một máy tính có thể thực hiện cả hai chức năng này.

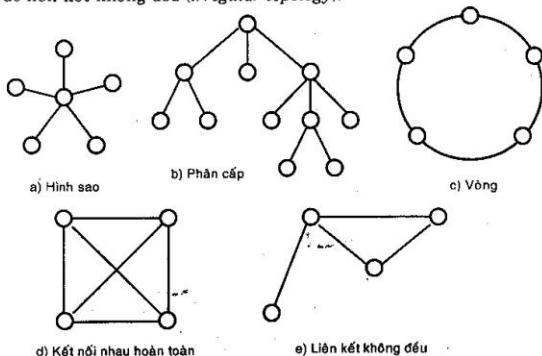
Bây giờ, xét hai ví dụ của mạng được chỉ ra trong hình 2.8b, một mạng dùng tuyến chung (*common bus*) và một mạng khác dùng vệ tinh. Đặc điểm chung của hai mạng truyền thông này là tất cả các nơi nhận tất cả các thông điệp được gửi từ nơi khác bất kỳ. Tất nhiên, đây là cơ chế cho phép mỗi nơi sẽ nhận các thông điệp được gửi đến nó và bỏ qua các thông điệp khác. Cơ chế tiêu biểu bao gồm việc thêm vào đầu mỗi thông điệp một định danh (*identification*) của nơi đến. Một mạng truyền thông thuộc loại này được gọi là *mạng truyền rộng (broadcast network)*. Trong một mạng truyền rộng, việc truyền rộng một thông điệp đến nhiều nơi sẽ không tốn kém hơn so với việc truyền chỉ một thông điệp đến một nơi.

Các mạng máy tính cũng được phân loại thành các mạng cục bộ (*local network*) và các mạng phân bố về mặt địa lý (*geographically distributed network*) hoặc mạng đường dài (*long-haul network*). Sự khác biệt này phụ thuộc vào các loại kênh truyền thông được sử dụng khi khoảng cách giữa các nơi gần khác với các kênh truyền thông

được sử dụng cho việc truyền thông khoảng cách xa. Một mạng cục bộ tiêu biểu không được vượt quá khoảng cách 1000 mét. Các nơi của một mạng cục bộ phải thuộc cùng tòa nhà, hoặc khuôn viên, hoặc xí nghiệp. Các kênh truyền thông có thể được thực hiện bằng cáp đồng trục (*coaxial cable*) hoặc cáp quang (*optical fiber*). Cách này có thông lượng 10 megabit trên giây và đạt độ tin cậy rất cao. Các mạng cục bộ tiêu biểu là các mạng truyền rộng.

Các mạng phân bố về mặt địa lý vẫn chủ yếu dùng các đường điện thoại. Các đường điện thoại chuyên dụng, tốt, có thể đạt thông lượng từ 50 đến 100 kilobit trên giây, tuy nhiên thông lượng này sẽ thấp hơn trong một số trường hợp. Hơn nữa, độ tin cậy sẽ thấp hơn so với các mạng cục bộ.

Cuối cùng, trong một số trường hợp, chúng ta cần phân biệt các mạng trên cơ sở cấu trúc liên kết (*topology*). Mặc dù chúng ta giả sử chức năng chọn tuyến cho chúng ta biết các mối liên kết vật lý tạo nên mạng được cấu tạo như thế nào, điều này đặc biệt có ích cho chúng ta phát triển một số giải thuật. Các cấu trúc liên kết cơ bản được chỉ ra trong hình 2.9 là hình sao (*star*), phân cấp (*hierarchical*), vòng (*ring*), kết nối nhau hoàn toàn (*completely interconnected*) và cấu trúc liên kết không đều (*irregular topology*).



Hình 2.9 Các cấu trúc liên kết mạng

2- Các giao thức và phiên

Hai quá trình muốn liên lạc trao đổi các thông điệp với nhau cần phải thỏa mãn một số quy tắc để đạt được truyền thông thỏa đáng. Các quy tắc mà hai hoặc nhiều quá trình truyền thông phải thỏa mãn được gọi là *giao thức (protocol)*. Cụ thể, một giao thức cho biết bên gửi (*sender*) và bên nhận (*receiver*) đạt được thỏa thuận như thế nào về việc trao đổi thông điệp, chúng có thể nhận biết với nhau như thế nào, có bao nhiêu thông điệp được trao đổi, một thông điệp có cần trả lời hay không và nhiều khía cạnh khác.

Một ví dụ về giao thức rất quen thuộc với chúng ta là tập hợp các quy tắc cần phải có để thực hiện một cuộc gọi điện thoại. Các quy tắc này bao gồm quay số của nơi đến, ghi nhận tín hiệu của điện thoại, nói "hello", tự giới thiệu, ... Rất khó thực hiện cuộc gọi nếu không biết các quy tắc này.

Trong một cuộc gọi điện thoại, mỗi lần người gọi và người bị gọi đã thực hiện các tác vụ mở đầu, họ có thể nói chuyện qua lại với nhau cho đến khi không còn muốn nói gì với nhau nữa và sau đó họ gác máy. Bây giờ xét hai quá trình tại hai nơi khác nhau của một mạng cũng có trao đổi thông tin nào đó bằng cách gửi các thông điệp với nhau. Cũng giống như hai người ở trên, mỗi khi hai quá trình thực hiện các tác vụ mở đầu cần thiết để nhận biết với nhau, chúng có thể trao đổi tất cả các thông điệp mà không cần phải lặp lại các tác vụ này cho mỗi thông điệp. Khái niệm *phiên (session)* được nêu ra cho mục đích này. Một phiên được thiết lập giữa hai quá trình muốn liên lạc và được duy trì cho đến khi tất cả các thông điệp đã được trao đổi với nhau. Việc kết thúc phiên tương tự với việc gác máy.

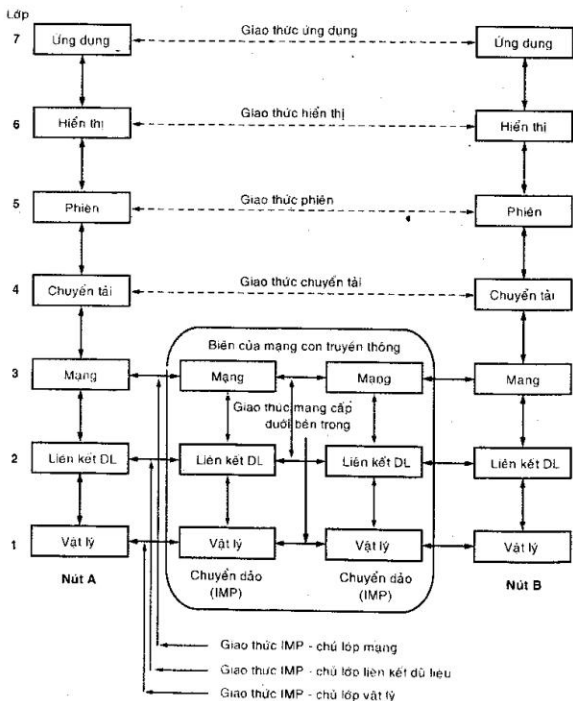
Cũng giống với cuộc gọi điện thoại giữa hai người, khoảng thời gian của một phiên tùy thuộc vào sự thỏa thuận sau đây. Khi hai người đang liên tục trao đổi thông tin với nhau thì không được gác máy. Tuy nhiên, nếu một người nghĩ về một vấn đề nào đó trước khi họ có thể trả lời thì nên gác máy và sau đó gọi lại. Bởi vì các phiên chiếm giữ các tài nguyên, do đó cần phải có thỏa thuận khi một số quá trình không cần liên lạc trong một khoảng thời gian nào đó.

Có một số điểm khác nhau cơ bản giữa một cuộc gọi điện thoại và một phiên giữa hai quá trình. Trong ví dụ cuộc gọi điện thoại, việc thiết lập liên lạc không chỉ bao gồm việc trao đổi các thông tin định danh giữa những người liên quan mà còn có việc thiết lập kết nối luận lý giữa họ. Xét mạng lưu trữ-và-chuyển tiếp được chỉ ra trong hình 2.8a và hai quá trình tại nơi 1 và nơi 4. Hai quá trình có thể thiết lập một phiên, do đó việc nhận biết với nhau và quyết định trao đổi nhiều thông điệp không cần phải có kết nối vật lý trực tiếp được thiết lập trước đó giữa chúng. Trong thực tế, các thông điệp được trao đổi giữa các quá trình trong cùng một phiên có thể đi theo những đường đi khác nhau, tùy thuộc vào các quyết định của giải thuật chọn tuyến. Do đó, khái niệm phiên là một khả năng mức cao được cung cấp bởi mạng truyền thông và không đề cập đến kết nối vật lý trực tiếp.

Trong một số trường hợp, việc trao đổi các thông điệp giữa các quá trình theo kiểu bất quy tắc và không có lợi cho việc thiết lập các phiên, toàn bộ giao thức mở đầu sẽ được gửi đi khi thực hiện mỗi thông điệp. Cách tiếp cận này được gọi là *gam dữ liệu (datagam)*. Trong cùng một mạng truyền thông, hai quá trình có thể trao đổi thông tin bằng cách thiết lập các phiên và/hoặc bằng cách gửi các thông điệp gam dữ liệu lẫn nhau. Việc chọn lựa tùy thuộc vào kiểu truyền thông.

3- Kiến trúc tham khảo ISO/ OSI

Cách tiếp cận phân lớp được sử dụng một cách tiêu biểu để đơn giản hóa thiết kế và thực hiện các mạng truyền thông. Mục đích của mỗi lớp là để cung cấp khả năng liên lạc ảo cho lớp ở mức cao hơn. Các mạng khác nhau sử dụng các cách tiếp cận khác nhau cho việc phân lớp của mạng. Ở đây chúng ta sẽ mô tả sơ lược mô hình tham khảo ISO (*International Standards Organization*) được gọi là OSI (*Open Systems Interconnection*). Hầu hết các mạng hiện tại đều có cấu trúc tương tự, mặc dù thường có một số lớp được kết hợp lại.



Hình 2.10 Kiến trúc tham khảo ISO/OSI

Mô hình tham khảo ISO / OSI có bảy lớp và được chỉ ra trong hình 2.10. Hình này cho thấy mỗi lớp được thực hiện tại tất cả các nơi liên lạc. Tại mỗi lớp, hai lớp tương ứng tại hai nơi khác nhau liên lạc với nhau bằng cách sử dụng giao thức của mức tương ứng.

Hầu hết các giải thuật và các giao thức được mô tả trong quyển sách này thuộc lớp ứng dụng. Theo quan điểm của các mạng máy tính truyền thông, trong thực tế CSDL phân bố là một ứng dụng đặc biệt.

Ba lớp thấp nhất là lớp vật lý (*physical layer*), lớp liên kết dữ liệu (*data link layer*) và lớp mạng (*network layer*) được gọi chung là lớp mạng con truyền thông (*communication subnet*). Mạng con truyền thông có nhiệm vụ truyền thông mức vật lý đáng tin cậy giữa hai nơi.

Để thực hiện lớp ứng dụng, các lớp nằm dưới phải cung cấp nhiều khả năng, trong đó khả năng cơ bản là truyền các thông điệp giữa các quá trình tại các nơi. Bây giờ, chúng ta xét các lớp để có được khả năng cơ bản này như thế nào.

Lớp hiển thị (presentation layer) giải quyết việc biến đổi thông tin giữa các dạng thức khác nhau của việc hiển thị dữ liệu được sử dụng tại các nơi khác nhau. Một ví dụ của chức năng này là biến đổi từ mã ký tự được sử dụng tại nơi gửi thành mã ký tự được sử dụng tại nơi đến. Theo cách này, quá trình bên gửi sẽ gửi một thông điệp được mã hóa theo dạng thức của bên gửi và quá trình bên nhận sẽ nhận thông điệp này đã được biến đổi theo dạng thức của bên nhận.

Lớp phiên (session layer) có nhiệm vụ thiết lập và duy trì các phiên giữa các quá trình. Tầm quan trọng phải có các phiên đã được trình bày.

Lớp chuyển tải (transport layer) là lớp nguồn-đích (*source-to-destination layer*) thực sự. Lớp này thực hiện kênh điểm-điểm ảo (*virtual point-to-point channel*) giữa nơi nguồn và nơi đích. Các thông điệp được gửi từ nơi nguồn sẽ được truyền không bị lỗi và theo đúng thứ tự tại nơi đích.

Để thực hiện chức năng này một cách hiệu quả, lớp chuyển tải sử dụng các dịch vụ của các lớp mạng, lớp liên kết dữ liệu và lớp vật lý nằm dưới. Chúng ta không quan tâm đến các lớp mức dưới này. Chúng phụ thuộc nhiều vào các loại kênh truyền thông được sử dụng và các khía cạnh kỹ thuật của truyền dữ liệu.

Lưu ý rằng, các đơn vị dữ liệu được truyền trong mạng là khác nhau giữa các thông điệp luận lý được gửi từ một quá trình cho một quá trình khác ở lớp ứng dụng. Một thông điệp đơn có thể được phân chia thành nhiều đơn vị truyền thông hoặc nhiều thông điệp có thể được kết hợp lại thành một đơn vị. Lớp chuyển tải cung cấp khả năng cơ bản cho các lớp cao hơn là trao đổi các thông điệp giữa các nơi. Kích thước của các thông điệp được sử dụng ở các lớp nằm trên lớp chuyển tải phụ thuộc vào nội dung thông tin của chúng, và kích thước của các đơn vị được truyền ở các mức thấp hơn phụ thuộc chủ yếu vào các đặc tính của các kênh truyền thông.

CÁC MỨC TRONG SUỐT PHÂN BỐ

Chương này giải quyết các mức khác nhau mà người lập trình ứng dụng có thể nhìn thấy CSDL phân bố, điều này phụ thuộc vào DDBMS cung cấp bao nhiêu mức trong suốt phân bố. Tính trong suốt phân bố (*distribution transparency*) đã được định nghĩa một cách không chính qui ở trong chương 1 là tính độc lập của chương trình ứng dụng đối với việc phân bố dữ liệu và đã được xem tương đương về mặt luận lý với tính độc lập dữ liệu trong các CSDL tập trung.

Trong chương này, chúng ta nêu ra nhiều mức trong suốt phân bố khác nhau. Để làm điều này, trong phần 3.1 chúng ta định nghĩa một kiến trúc tham khảo phân lớp (*layered reference architecture*) cho một CSDL phân bố. Kiến trúc này cho phép chúng ta xác định một cách dễ dàng các mức trong suốt phân bố khác nhau. Các mức này thích hợp về mặt ý niệm để hiểu các CSDL phân bố, mặc dù các hệ thống cụ thể có thể không thực hiện chúng một cách tường minh.

Phần 3.2 cho thấy ánh xạ giữa các mức khác nhau được định nghĩa như thế nào. Chúng ta sử dụng mô hình quan hệ và đại số quan hệ cho mục đích này.

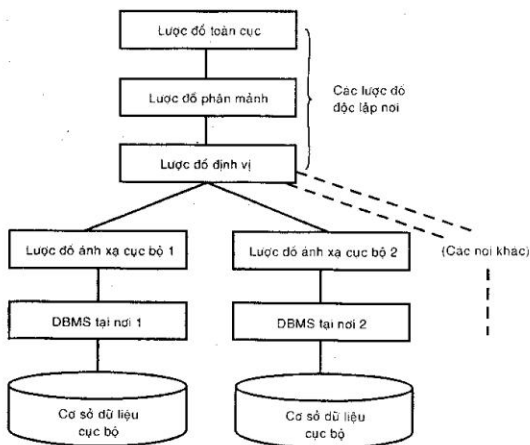
Các phần còn lại cho thấy các ứng dụng có thể được viết như thế nào ở các mức khác nhau được định nghĩa trong kiến trúc tham khảo. Các ứng dụng được viết bằng ngôn ngữ tựa Pascal (*Pascal-like language*) với các lệnh SQL nhúng (*embedded SQL statement*). Các lệnh SQL biểu diễn các tác vụ cơ bản truy xuất CSDL phân bố; SQL đã được chọn là một ngôn ngữ thao tác dữ liệu (*data manipulation language*) bởi vì nó diễn giải các truy xuất đến một CSDL theo một cách thức thân thiện với người sử dụng, phi thủ tục (*nonprocedural*), hướng tập hợp (*set-oriented*).

Điều nhấn mạnh trong chương này là các ứng dụng nhìn CSDL như thế nào, nghĩa là các thao tác cơ bản SQL tham chiếu đến các đối tượng tạo nên CSDL, và việc thay đổi phân bố dữ liệu sẽ ảnh hưởng đến bao nhiêu chương trình ứng dụng. Ở đây, chúng ta không quan tâm đến tính hiệu quả của các chiến lược truy xuất CSDL của các ứng dụng. Vấn đề xác định, đánh giá và chọn lựa giữa các chiến lược truy xuất khác nhau sẽ được giải quyết trong Chương 5 và 6. Tuy nhiên, chúng ta sẽ cho thấy rằng cùng một ứng dụng có thể được viết theo nhiều cách khác nhau, và cách mà theo đó một ứng dụng được viết ra có thể ảnh hưởng đến các chiến lược truy xuất có thể có.

Phần 3.3 giải quyết các ứng dụng chỉ đọc (*read-only application*). Trước tiên, phần này nêu ra một ứng dụng chỉ đọc rất đơn giản và sau đó là một ứng dụng phức tạp hơn với các mức trong suốt phân bố khác nhau. Phần 3.4 thực hiện cùng một phân tích giống như phần 3.3 trên các ứng dụng cập nhật (*update application*). Cả hai phần trên đều dựa trên các ứng dụng truy xuất CSDL để lấy ra hoặc cập nhật chỉ một bộ; theo cách này, chúng ta có thể bàn luận về tính trong suốt phân bố mà không liên quan đến các vấn đề về truy xuất các tập hợp bộ. Trong phần 3.5, chúng ta giải quyết vấn đề sau cùng và cho thấy rằng ngay cả ở mức trong suốt phân bố cao nhất thì người lập trình ứng dụng có thể sử dụng khả năng mạnh của các tác vụ cơ bản mà chúng được gửi cho DBMS. Phần 3.6 giải quyết các ràng buộc toàn vẹn (*integrity constraint*) và việc thực hiện chúng trong các CSDL phân bố.

3.1 KIẾN TRÚC THAM KHẢO DÙNG CHO CÁC CSDL PHÂN BỐ

Hình 3.1 cho thấy một kiến trúc tham khảo dùng cho một CSDL phân bố. Kiến trúc tham khảo này không được thực hiện một cách tường minh trong tất cả CSDL phân bố; tuy nhiên, các mức của nó thích hợp về mặt ý niệm để hiểu tổ chức của một CSDL phân bố bất kỳ. Do đó, chúng ta sẽ phân tích và nắm bắt tất cả các thành phần của kiến trúc tham khảo này.



Hình 3.1 Một kiến trúc tham khảo dùng cho các CSDL phân bố

Ở mức cao nhất của hình 3.1 là *lược đồ toàn cục* (*global schema*). Lược đồ toàn cục định nghĩa tất cả các dữ liệu được chứa trong CSDL phân bố và cũng giống như CSDL không được phân bố gì cả. Do đó, lược đồ toàn cục có thể được định nghĩa hoàn toàn giống như trong một CSDL không phân bố. Tuy nhiên, mô hình dữ liệu dùng để định nghĩa một lược đồ toàn cục nên thuận tiện cho việc định nghĩa các ánh xạ giữa các mức khác nhau của CSDL phân bố. Chúng ta sẽ sử dụng mô hình quan hệ cho mục đích này. Bằng cách sử dụng mô hình này, lược đồ toàn cục bao gồm định nghĩa một tập hợp các *quan hệ toàn cục* (*global relation*).

Mỗi quan hệ toàn cục có thể được phân thành nhiều phần không phủ lấp nhau, các phần này được gọi là các *mảnh* (*fragment*). Có nhiều cách khác nhau để thực hiện phép phân mảnh và được mô tả trong phần kế tiếp. Ánh xạ giữa các quan hệ toàn cục và các mảnh được định nghĩa trong *lược đồ phân mảnh* (*fragmentation schema*).

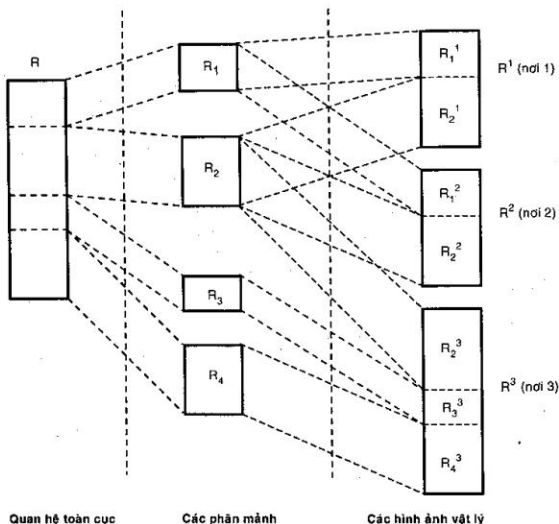
Ánh xạ này là một-nhiều (*one-to-many*); nghĩa là một quan hệ toàn cục được phân thành nhiều mảnh, nhưng một mảnh chỉ thuộc một quan hệ toàn cục. Các mảnh được chỉ ra bởi một tên quan hệ toàn cục với một chỉ số (chỉ số mảnh - *fragment index*); ví dụ R_i là mảnh thứ i của quan hệ toàn cục R .

Các mảnh là những phần luận lý của các quan hệ toàn cục, mà về mặt vật lý chúng được đặt tại một hoặc nhiều nơi của mạng. **Lược đồ định vị** (*allocation schema*) xác định một mảnh được đặt tại các nơi nào. Lưu ý rằng loại ánh xạ được định nghĩa trong lược đồ định vị sẽ xác định CSDL phân bố là **dư thừa** (*redundant*) hoặc **không dư thừa** (*nonredundant*); trong trường hợp đầu thì ánh xạ là một-nhiều, trong trường hợp sau thì ánh xạ là một-một (*one-to-one*). Tất cả các mảnh thuộc cùng một quan hệ toàn cục R và được đặt tại cùng một nơi j sẽ tạo thành **hình ảnh vật lý** (*physical image*) của quan hệ toàn cục R tại nơi j . Do đó có một ánh xạ một-một giữa một hình ảnh vật lý với một cặp (quan hệ toàn cục, nơi); các hình ảnh vật lý có thể được chỉ ra bởi một tên quan hệ toàn cục và một chỉ số nơi (*site index*). Để phân biệt chúng với các mảnh, chúng ta sẽ sử dụng chỉ số trên (*superscript*); ví dụ R^j để chỉ ra hình ảnh vật lý của quan hệ toàn cục R tại nơi j .

Hình 3.2 cho thấy một ví dụ về mối liên kết giữa các loại đối tượng được định nghĩa ở trên. Một quan hệ toàn cục R được phân thành bốn mảnh R_1 , R_2 , R_3 và R_4 . Bốn mảnh này được đặt một cách dư thừa tại ba nơi của một mạng máy tính, do đó tạo thành ba hình ảnh vật lý R^1 , R^2 và R^3 .

Một mảnh đặt tại một nơi cho trước được gọi là **bản nhân** (*replica*) của mảnh này và được ký hiệu bằng cách dùng tên quan hệ toàn cục và hai chỉ số (một chỉ số mảnh và một chỉ số nơi). Ví dụ trong Hình 3.2, ký hiệu R_2^3 là bản nhân của mảnh R_2 được đặt tại nơi 3.

Cuối cùng, lưu ý rằng hai hình ảnh vật lý có thể giống nhau. Trong trường hợp này, chúng ta sẽ nói rằng một hình ảnh vật lý là một bản nhân của một hình ảnh vật lý khác. Ví dụ trong hình 3.2, R^1 là một bản nhân của R^2 .



Hình 3.2 Các mảnh và các hình ảnh vật lý của một quan hệ toàn cục

Bây giờ, chúng ta trở lại kiến trúc tham khảo của hình 3.1. Chúng ta đã mô tả các mối liên kết giữa các đối tượng ở ba mức trên cùng của kiến trúc này. Ba mức này là độc lập vị trí (*site independent*); do đó, chúng không phụ thuộc vào mô hình dữ liệu của các DBMS cục bộ. Ở mức thấp hơn, cần phải ánh xạ các hình ảnh vật lý vào các đối tượng mà chúng được thao tác bởi các DBMS cục bộ. Ánh xạ này được gọi là *lược đồ ánh xạ cục bộ (local mapping schema)* và phụ thuộc vào loại DBMS cục bộ; do đó, trong một hệ thống không đồng nhất, chúng ta có nhiều loại ánh xạ cục bộ khác nhau tại các nơi khác nhau.

Kiến trúc này cung cấp một cơ cấu tổ chức ý niệm (*conceptual framework*) rất tổng quát để hiểu các CSDL phân bố. Ba mục tiêu quan trọng nhất, mà do đó có được các đặc điểm của kiến trúc này, là

sự tách biệt giữa phân mảnh dữ liệu (*data fragmentation*) và định vị dữ liệu (*data allocation*), điều khiển dư thừa (*control of redundancy*) và sự độc lập giữa các DBMS cục bộ.

1- Khái niệm phân mảnh dữ liệu tách biệt với khái niệm định vị dữ liệu: Sự tách biệt này cho phép chúng ta phân biệt hai mức trong suốt phân bố khác nhau, được gọi là *trong suốt phân mảnh* (*fragmentation transparency*) và *trong suốt vị trí* (*location transparency*). Trong suốt phân mảnh là mức trong suốt cao nhất cho phép người sử dụng hoặc người lập trình ứng dụng làm việc trên các quan hệ toàn cục. Trong suốt vị trí là mức trong suốt thấp hơn và đòi hỏi người sử dụng hoặc người lập trình ứng dụng làm việc trên các mảnh thay vì trên các quan hệ toàn cục; tuy nhiên, họ không biết các mảnh được đặt tại các nơi nào. Sự tách biệt giữa khái niệm về sự phân mảnh và sự định vị sẽ rất thuận lợi trong thiết kế CSDL phân bố, bởi vì việc xác định các phần dữ liệu thích hợp sẽ khác với vấn đề định vị tối ưu (*optimal allocation*), và sẽ được trình bày trong chương kế tiếp.

2- Điều khiển dư thừa tường minh: Kiến trúc tham khảo cung cấp điều khiển dư thừa tường minh ở mức phân mảnh. Ví dụ, trong hình 3.2, hai hình ảnh vật lý R^2 và R^3 phủ lấp lên nhau; nghĩa là chúng chứa dữ liệu chung. Định nghĩa các mảnh tách biệt (*disjoint fragment*) khi xây dựng các khối hình ảnh vật lý cho phép chúng ta tham chiếu tường minh đến phần phủ lấp này: mảnh được nhân bản R_2 . Như chúng ta sẽ thấy, điều khiển dư thừa tường minh có ích lợi trong nhiều khía cạnh của quản lý CSDL phân bố.

3- Sự độc lập giữa các DBMS cục bộ: Đặc điểm này, được gọi là *trong suốt ánh xạ cục bộ* (*local mapping transparency*), cho phép chúng ta nghiên cứu nhiều vấn đề về quản lý CSDL phân bố mà không cần phải đưa vào các mô hình dữ liệu cụ thể của các DBMS cục bộ. Rõ ràng, trong hệ thống đồng nhất (*homogenous system*), các lược đồ độc lập vị trí (*site-independent schema*) có thể được định nghĩa bằng cách sử dụng cùng mô hình dữ liệu của các DBMS cục bộ, do đó làm giảm độ phức tạp của ánh xạ này.

Một loại trong suốt liên quan chặt chẽ với trong suốt vị trí là *trong suốt nhân bản* (*replication transparency*). Trong suốt nhân bản

có nghĩa là người sử dụng không biết có sự nhân bản các mảnh. Rõ ràng, trong suốt nhân bản được bao hàm trong suốt vị trí; tuy nhiên, trong một số trường hợp, người sử dụng không có trong suốt vị trí nhưng lại có trong suốt nhân bản (do đó, họ sử dụng một bản nhân riêng biệt và hệ thống làm các tác vụ tương ứng trên các bản nhân khác). Trong quyển sách này, chúng ta không phân biệt trong suốt nhân bản với trong suốt vị trí.

Một lưu ý cuối cùng về kiến trúc tham khảo là chúng ta có thể nói rằng nó thực hiện cùng một chức năng như kiến trúc ANSI/SPARC trong các DBMS truyền thống: nó không cần thiết phải được thực hiện bởi các hệ thống hiện tại, nhưng nó cho thấy các mức nào và các lược đồ nào là thích hợp về ý niệm.

3.2 CÁC LOẠI PHÂN MẢNH DỮ LIỆU

Phân rã các quan hệ toàn cục thành các mảnh có thể được thực hiện bằng cách áp dụng hai loại phân mảnh khác nhau:

- *Phân mảnh ngang (horizontal fragmentation)*
- *Phân mảnh dọc (vertical fragmentation)*

Trước tiên, chúng ta sẽ xét hai loại phân mảnh này một cách riêng biệt và sau đó xét sự phân mảnh phức tạp hơn có thể có được bằng cách áp dụng sự kết hợp của hai loại này.

Trong tất cả các loại phân mảnh, một mảnh có thể được xác định bởi một biểu thức trong một ngôn ngữ quan hệ (chúng ta sẽ dùng đại số quan hệ) mà các quan hệ toàn cục là toán hạng và tạo ra kết quả là một mảnh. Ví dụ, nếu một quan hệ toàn cục chứa dữ liệu của các nhân viên, một mảnh chỉ chứa dữ liệu của các nhân viên làm việc ở phòng ban D_1 có thể được xác định rõ ràng bằng một phép chọn (*selection operation*) trên quan hệ toàn cục.

Trong chuẩn hóa dữ liệu, chúng ta có một số qui tắc để đảm bảo tính nhất quán của CSDL. Lưu ý quan trọng là việc phân mảnh dữ liệu nhằm để phân bố dữ liệu (đặc biệt là phân mảnh dọc) tương tự với việc chuẩn hóa các quan hệ. Do đó, chúng ta có thể định nghĩa các qui tắc phân mảnh tương tự như các nguyên lý chuẩn hóa dữ liệu. Phân mảnh dữ liệu phải thỏa mãn ba qui tắc sau đây để bảo đảm CSDL không bị thay đổi ngữ nghĩa trong quá trình phân mảnh. Các qui tắc này được gọi là các điều kiện đúng đắn (*correctness condition*).

a- Điều kiện đầy đủ (completeness condition)

Nếu một quan hệ R được phân rã thành các mảnh $R_1, R_2, \dots, R_n$, mỗi mục dữ liệu (*data item*) có thể được tìm thấy trong R thì nó cũng có thể được tìm thấy trong một hoặc nhiều mảnh R_i . Đặc tính này giống với đặc tính *phân rã bảo toàn nội dung (lossless decomposition)* của chuẩn hóa dữ liệu và cũng quan trọng trong việc phân mảnh, bởi vì nó bảo đảm dữ liệu trong một quan hệ toàn cục được ánh xạ vào trong các mảnh mà không bị mất. Lưu ý rằng trong trường hợp phân mảnh ngang, mục dữ liệu là một bộ (*tuple*), và trong trường hợp phân mảnh dọc, mục dữ liệu là một thuộc tính.

- *Phân mảnh ngang:*

$$\forall u \in R, \exists i \in [1, n]: u \in R_i$$

- *Phân mảnh dọc:*

$$\forall A \in \text{Attr}(R), \exists i \in [1, n]: A \in \text{Attr}(R_i)$$

với $\text{Attr}(R)$ là tập thuộc tính của R .

b- Điều kiện tái tạo (reconstruction condition)

Nếu một quan hệ R được phân rã thành các mảnh $R_1, R_2, \dots, R_n$, luôn luôn có thể xác định một phép toán quan hệ ∇ sao cho:

$$R = \nabla R_i, \forall R_i \in F_R$$

Phép toán ∇ sẽ khác nhau đối với mỗi loại phân mảnh khác nhau; tuy nhiên, điều quan trọng là nó phải được xác định. Trong trường hợp phân mảnh ngang, phép toán ∇ là phép hợp ($\cup$), trong trường hợp phân mảnh dọc, phép toán ∇ là phép kết ($\bowtie$).

- *Phân mảnh ngang:*

$$R = R_1 \cup R_2 \cup \dots \cup R_n$$

- *Phân mảnh dọc:*

$$R = R_1 \bowtie R_2 \bowtie \dots \bowtie R_n$$

Tính tái tạo (*reconstructability*) của quan hệ từ các mảnh của nó bảo đảm các ràng buộc về phụ thuộc dữ liệu sẽ được bảo toàn. Trong thực tế, chỉ có các mảnh được chứa trong CSDL phân bố, và quan hệ toàn cục phải được tạo lại thông qua phép toán ∇ này, nếu cần.

c- Điều kiện tách biệt (disjointness condition)

Nếu một quan hệ R được phân mảnh ngang thành các mảnh $R_1, R_2, \dots, R_n$ và mục dữ liệu d_i có trong R_i thì nó không có trong bất kỳ mảnh R_k khác ($k \neq i$). Điều kiện này bảo đảm các mảnh ngang là tách biệt nhau. Nếu quan hệ R được phân mảnh dọc, thông thường các thuộc tính khóa chính đều có trong tất cả các mảnh của nó; do đó, điều kiện này thường bị vi phạm. Điều kiện tách biệt bảo đảm việc nhân bản dữ liệu có thể được điều khiển một cách tường minh ở mức định vị (allocation level).

Phân mảnh ngang:

$$\forall i \neq k \text{ và } i, k \in [1, n]: R_i \cap R_k = \emptyset$$

Các loại phân mảnh dữ liệu gồm có:

- 1- Phân mảnh ngang (horizontal fragmentation): gồm có hai loại là
 - Phân mảnh ngang chính (primary horizontal fragmentation) và
 - Phân mảnh ngang dẫn xuất (derived horizontal fragmentation)
- 2- Phân mảnh dọc (vertical fragmentation): gồm có hai loại là
 - Phân mảnh dọc gom tụ (vertical clustering fragmentation) và
 - Phân mảnh dọc tách biệt (vertical partitioning fragmentation)
- 3- Phân mảnh hỗn hợp (mixed/hybrid fragmentation) là sự kết hợp của phân mảnh ngang với phân mảnh dọc.

1- Phân mảnh ngang chính

Phân mảnh ngang chính (primary horizontal fragmentation) là sự phân chia các bộ của một quan hệ toàn cục thành các tập hợp con dựa vào các thuộc tính của quan hệ này, mỗi tập hợp con được gọi là một **mảnh ngang** (horizontal fragment). Rõ ràng, điều này là có ích trong các CSDL phân bố, mỗi tập hợp con có thể chứa dữ liệu có các tính chất địa lý chung. Nó có thể được xác định bằng cách biểu diễn mỗi mảnh như là một phép chọn trên quan hệ toàn cục. Ví dụ, cho một quan hệ toàn cục là:

SUPPLIER (SNUM, NAME, CITY)

Sau đó, phân mảnh ngang có thể được xác định theo cách sau đây:

$SUPPLIER_1 = \sigma_{CITY = 'SF'} (SUPPLIER)$

$SUPPLIER_2 = \sigma_{CITY = 'LA'} (SUPPLIER)$

Sự phân mảnh ở trên thỏa mãn điều kiện đầy đủ nếu 'SF' và 'LA' chỉ là các giá trị có thể có của thuộc tính CITY; ngược lại thì chúng ta sẽ không biết các bộ ứng với các giá trị khác của CITY thuộc về mảnh nào.

Điều kiện tái tạo được thỏa mãn dễ dàng, bởi vì luôn luôn có thể tái tạo quan hệ toàn cục SUPPLIER thông qua phép hợp (*union operation*) sau đây:

$$\text{SUPPLIER} = \text{SUPPLIER}_1 \cup \text{SUPPLIER}_2$$

Rõ ràng, điều kiện tách biệt được thỏa mãn vì SNUM là khóa của SUPPLIER.

Chúng ta gọi vị từ (*predicate*) được sử dụng trong phép chọn để xác định một mảnh là *vị từ định tính* (*qualification*) hoặc *điều kiện chọn* của nó. Chẳng hạn, trong ví dụ trên, các vị từ định tính là:

$$q_1: \text{CITY} = \text{'SF'}$$

$$q_2: \text{CITY} = \text{'LA'}$$

Chúng ta có thể tổng quát hóa từ ví dụ trên như sau:

- Để thỏa mãn điều kiện đầy đủ thì tập hợp các vị từ định tính của tất cả các mảnh phải là đầy đủ, ít nhất là đối với tập hợp các giá trị cho phép.

- Điều kiện tái tạo luôn luôn được thỏa mãn thông qua phép hợp.

- Để thỏa mãn điều kiện tách biệt thì các vị từ định tính phải loại trừ nhau (*mutual exclusive*).

2- Phân mảnh ngang dẫn xuất

Trong một số trường hợp, phân mảnh ngang của một quan hệ không thể dựa trên một đặc điểm của các thuộc tính riêng của nó, nhưng được dẫn xuất từ việc phân mảnh ngang của một quan hệ khác và được gọi là *phân mảnh ngang dẫn xuất* (*derived horizontal fragmentation*). Ví dụ, xét quan hệ toàn cục:

$$\text{SUPPLY}(\text{SNUM}, \text{PNUM}, \text{DEPTNUM}, \text{QUAN})$$

trong đó SNUM là mã nhà cung cấp. Điều có ý nghĩa là phân mảnh quan hệ này sao cho một mảnh chứa các bộ thuộc các nhà cung cấp ở một thành phố cho trước. Tuy nhiên, thành phố không phải là một thuộc tính của quan hệ SUPPLY, mà nó là thuộc tính của quan hệ SUPPLIER được xét trong ví dụ trên. Do đó, chúng ta sử dụng một

phép nửa kết (*semi-join operation*) để xác định các bộ của SUPPLY thuộc các nhà cung cấp trong một thành phố cho trước. Phần mảnh ngang dẫn xuất có thể được xác định như sau:

$$\text{SUPPLY}_1 = \text{SUPPLY} \bowtie_{\text{SNUM} = \text{SNUM}} \text{SUPPLIER}_1$$

$$\text{SUPPLY}_2 = \text{SUPPLY} \bowtie_{\text{SNUM} = \text{SNUM}} \text{SUPPLIER}_2$$

Các phép nửa kết dùng để chọn từ SUPPLY các bộ thỏa mãn điều kiện kết giữa SUPPLIER₁ hoặc SUPPLIER₂ với SUPPLY; do đó, chúng xác định các bộ của SUPPLY thuộc các nhà cung cấp ở San Francisco hoặc Los Angeles theo thứ tự tương ứng.

Quan hệ toàn cục SUPPLY có thể được tái tạo bằng cách thực hiện phép hợp SUPPLY₁ và SUPPLY₂.

$$\text{SUPPLY} = \text{SUPPLY}_1 \cup \text{SUPPLY}_2$$

Khi một quan hệ toàn cục R được phân mảnh ngang dẫn xuất, các vị từ định tính của các mảnh của nó không thể được biểu diễn bằng các vị từ chỉ sử dụng các thuộc tính của R ; hơn nữa, điều kiện để một bộ t thuộc về một mảnh cho trước R_i của R là phải tồn tại một bộ t' (trong một mảnh S_i nào đó của S) sao cho t và t' thỏa mãn điều kiện của phép nửa kết của phân mảnh ngang dẫn xuất. Xét ví dụ trên, chúng ta đưa ra một biểu diễn đơn giản của điều kiện này như sau:

$$q_1: \text{SUPPLY.SNUM} = \text{SUPPLIER.SNUM AND SUPPLIER.CITY} = \text{'SF'}$$

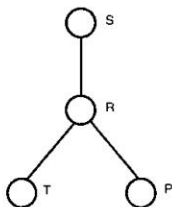
$$q_2: \text{SUPPLY.SNUM} = \text{SUPPLIER.SNUM AND SUPPLIER.CITY} = \text{'LA'}$$

Ý nghĩa của các vị từ định tính ở trên được phát biểu như sau: đối với một bộ bất kỳ của SUPPLY₁ (SUPPLY₂), tồn tại một nhà cung cấp ở San Francisco (Los Angeles) có cùng mã nhà cung cấp.

Để thỏa mãn điều kiện đầy đủ của phân mảnh ở trên, thì tất cả các mã nhà cung cấp trong quan hệ SUPPLY đều phải có trong quan hệ SUPPLIER. Đây là một ràng buộc toàn vẹn, tiêu biểu và hợp lý cho CSDL này, và thường được gọi là *ràng buộc toàn vẹn tham chiếu* (*referential integrity constraint*).

Điều kiện tách biệt được thỏa mãn nếu một bộ của quan hệ SUPPLY không tương ứng với hai bộ của quan hệ SUPPLIER thuộc hai mảnh khác nhau. Trong trường hợp này, điều kiện này dễ dàng được thỏa mãn, bởi vì các mã nhà cung cấp là các khóa duy nhất của quan hệ SUPPLIER; tuy nhiên, nói chung rất khó chứng minh rằng điều kiện này được thỏa mãn.

Lưu ý rằng nếu một quan hệ toàn cục R được phân mảnh ngang dẫn xuất theo một quan hệ toàn cục S , thì cũng có thể có hai quan hệ toàn cục T và P được phân mảnh ngang dẫn xuất theo R và cứ như thế. Nói chung, phân mảnh ngang dẫn xuất của các quan hệ toàn cục có thể được biểu diễn bằng một *cây phân mảnh dẫn xuất* (*derived fragmentation tree*); ví dụ quan hệ R là nút con của quan hệ S , các quan hệ T và P là các nút con của quan hệ R , xem hình 3.3.



Hình 3.3

Cây phân mảnh dẫn xuất

3- Phân mảnh dọc

Phân mảnh dọc một quan hệ toàn cục là sự phân chia các thuộc tính của quan hệ này thành các nhóm; các mảnh có được bằng cách chiếu quan hệ toàn cục trên mỗi nhóm. Điều này có thể có ích trong các CSDL phân bố, mỗi nhóm thuộc tính có thể chứa dữ liệu mà chúng có những tính chất địa lý chung. Sự phân mảnh là đúng đắn nếu mỗi thuộc tính được ánh xạ vào ít nhất một thuộc tính của các mảnh; hơn nữa, có thể tái tạo quan hệ toàn cục bằng cách kết các mảnh với nhau. Ví dụ, xét một quan hệ toàn cục:

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)

Phân mảnh dọc quan hệ này có thể được xác định như sau:

$EMP_1 = \Pi_{EMPNUM, NAME, MGRNUM, DEPTNUM} (EMP)$

$EMP_2 = \Pi_{EMPNUM, SAL, TAX} (EMP)$

Sự phân mảnh này có thể phản ánh một tổ chức mà tiền lương và thuế được quản lý riêng biệt. Tái tạo quan hệ EMP có thể được thực hiện bằng phép kết như sau:

$EMP = EMP_1 \bowtie_{EMPNUM = EMPNUM} EMP_2$

trong đó EMPNUM là khóa của EMP. Nói chung, việc đặt một khóa của một quan hệ toàn cục vào trong mỗi mảnh là một cách trực tiếp nhất để bảo đảm việc tái tạo quan hệ toàn cục bằng cách thực hiện phép kết. Một cách khác để tái tạo quan hệ toàn cục là tạo ra các

danh hiệu bộ (*tuple identifier*) mà chúng được sử dụng như là các khóa được điều khiển bởi hệ thống (*system-controlled key*). Điều này nhằm để tránh nhân bản các khóa dài; hơn nữa, người sử dụng không thể thay đổi các danh hiệu bộ.

Lưu ý rằng công thức ở trên để tái tạo quan hệ toàn cục EMP là không đầy đủ, bởi vì kết quả của phép kết EMP₁ với EMP₂ sẽ chứa cột EMPNUM hai lần. Việc nhân bản không mong muốn này có thể bị loại bỏ bởi một phép chiếu mà chúng ta không nêu ra hoặc sử dụng phép kết tự nhiên (*natural join operation*).

Cuối cùng, chúng ta xét vấn đề về tính tách biệt mảnh (*fragment disjointness*). Trước tiên, chúng ta thấy rằng ít nhất là khóa nên được nhân bản trong tất cả các mảnh để tái tạo quan hệ toàn cục. Nói chung, trong phân mảnh dọc, động cơ thúc đẩy chính để có các mảnh tách biệt (nghĩa là để điều khiển nhân bản) là không quan trọng như trong phân mảnh ngang. Trong thực tế, nếu chúng ta đưa cùng một thuộc tính vào trong hai mảnh dọc khác nhau, chúng ta biết chính xác rằng dữ liệu được nhân bản sẽ được chứa trong cột tương ứng với thuộc tính này; nếu chúng ta cho phép hai mảnh ngang phủ lấp, bằng cách cho phép các vị từ định tính không loại trừ nhau, thì chúng ta không thể tham chiếu trực tiếp đến phần phủ lấp này.

Ví dụ, xét phân mảnh dọc sau đây của quan hệ EMP:

$$EMP_1 = \Pi_{EMPNUM, NAME, MGRNUM, DEPTNUM}(EMP)$$

$$EMP_2 = \Pi_{EMPNUM, NAME, SAL, TAX}(EMP)$$

Thuộc tính NAME được nhân bản trong cả hai mảnh. Chúng ta có thể loại trừ thuộc tính này một cách tường minh khi chúng ta tái tạo quan hệ EMP bằng cách thực hiện một phép chiếu:

$$EMP = EMP_1 \bowtie EMPNUM = EMPNUM \Pi_{EMPNUM, SAL, TAX}(EMP_2)$$

4- Phân mảnh hỗn hợp

Các mảnh có được từ các phép phân mảnh ở trên chính là các quan hệ, do đó chúng ta có thể áp dụng các phép phân mảnh này một cách đệ qui, sao cho mỗi lần thực hiện phân mảnh phải thỏa mãn các điều kiện đúng đắn. Sự tái tạo có thể có được bằng cách áp dụng các qui tắc theo thứ tự ngược. Trong trường hợp này, các biểu thức định nghĩa các mảnh sẽ phức tạp hơn; ví dụ, xét cùng quan hệ toàn cục:

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)

Phân mảnh sau đây là phân mảnh hỗn hợp có được bằng cách áp dụng phân mảnh dọc của ví dụ trước, rồi kế tiếp là phân mảnh ngang dựa trên DEPTNUM:

$EMP_1 = \sigma_{DEPTNUM \leq 10} \Pi_{EMPNUM, NAME, MGRNUM, DEPTNUM}(EMP)$

$EMP_2 = \sigma_{DEPTNUM > 10 \text{ AND } DEPTNUM \leq 20} \Pi_{EMPNUM, NAME, MGRNUM, DEPTNUM}(EMP)$

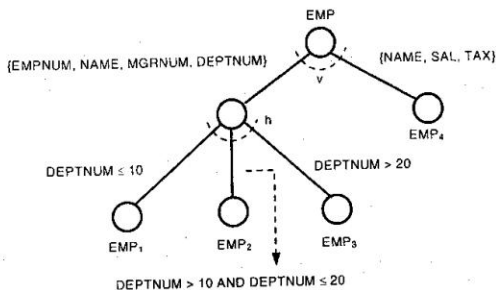
$EMP_3 = \sigma_{DEPTNUM > 20} \Pi_{EMPNUM, NAME, MGRNUM, DEPTNUM}(EMP)$

$EMP_4 = \Pi_{EMPNUM, NAME, SAL, TAX}(EMP)$

Tái tạo quan hệ EMP được xác định bởi biểu thức sau đây:

$EMP = (EMP_1 \cup EMP_2 \cup EMP_3) \bowtie_{EMPNUM = EMPNUM} \Pi_{EMPNUM, SAL, TAX}(EMP_4)$

Phân mảnh hỗn hợp có thể được biểu diễn bằng một *cây phân mảnh* (fragmentation tree). Trong cây phân mảnh, nút gốc tương ứng với một quan hệ toàn cục, các nút lá tương ứng với các mảnh và các nút trung gian tương ứng với các kết quả trung gian của các biểu thức định nghĩa mảnh (fragment-defining expression). Tập hợp các nút con của một nút đã cho biểu diễn sự phân rã (decomposition) của nút này bởi một phép phân mảnh (dọc hoặc ngang). Ví dụ, hình 3.4 cho thấy cây phân mảnh của quan hệ EMP. Nút gốc (quan hệ EMP) được phân mảnh dọc thành hai mảnh: một mảnh tương ứng với nút trái của cây (EMP_4); mảnh khác được phân mảnh ngang tạo ra ba nút lá khác tương ứng với các mảnh EMP_1 , EMP_2 và EMP_3 .



Hình 3.4 Cây phân mảnh của quan hệ EMP

EXAMPLE.DDB

Hình 3.5 cho thấy các lược đồ toàn cục và lược đồ phân mảnh của EXAMPLE.DDB được sử dụng trong phần còn lại của quyển sách này để làm các ví dụ. Hầu hết các quan hệ toàn cục của EXAMPLE.DDB và sự phân mảnh của chúng đã được giới thiệu. Quan hệ DEPT, được phân thành ba mảnh ngang dựa vào giá trị của thuộc tính DEPTNUM. Các đặc điểm của EXAMPLE.DDB sẽ được bàn đến khi chúng được sử dụng làm ví dụ cho các chủ đề cụ thể.

– *Lược đồ toàn cục:*

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)

DEPT (DEPTNUM, NAME, AREA, MGRNUM)

SUPPLIER (SNUM, NAME, CITY)

SUPPLY (SNUM, PNUM, DEPTNUM, QUAN)

– *Lược đồ phân mảnh:*

$EMP_1 = \sigma_{DEPTNUM = 10} \Pi_{EMPNUM, NAME, MGRNUM, DEPTNUM} (EMP)$

$EMP_2 = \sigma_{DEPTNUM > 10 \text{ AND } DEPTNUM = 20} \Pi_{EMPNUM, NAME, MGRNUM, DEPTNUM} (EMP)$

$EMP_3 = \sigma_{DEPTNUM > 20} \Pi_{EMPNUM, NAME, MGRNUM, DEPTNUM} (EMP)$

$EMP_4 = \Pi_{EMPNUM, NAME, SAL, TAX} (EMP)$

$DEPT_1 = \sigma_{DEPTNUM = 10} (DEPT)$

$DEPT_2 = \sigma_{DEPTNUM > 10 \text{ AND } DEPTNUM = 20} (DEPT)$

$DEPT_3 = \sigma_{DEPTNUM > 20} (DEPT)$

$SUPPLIER_1 = \sigma_{CITY = 'SF'} (SUPPLIER)$

$SUPPLIER_2 = \sigma_{CITY = 'LA'} (SUPPLIER)$

$SUPPLY_1 = SUPPLY \bowtie_{SNUM = SNUM} SUPPLIER_1$

$SUPPLY_2 = SUPPLY \bowtie_{SNUM = SNUM} SUPPLIER_2$

Hình 3.5 Lược đồ toàn cục và lược đồ phân mảnh của EXAMPLE.DDB

Lưu ý rằng bốn quan hệ toàn cục có nhiều thuộc tính chung mà các thuộc tính này sẽ được dùng để thực hiện các phép kết, đặc biệt DEPTNUM trong các quan hệ EMP, DEPT và SUPPLY; MGRNUM trong các quan hệ EMP và DEPT; và SNUM trong các quan hệ SUPPLIER và SUPPLY. Trong tất cả các trường hợp này, cùng tên thuộc tính sẽ tham chiếu đến các miền quan hệ đồng nhất về mặt ngữ nghĩa. Điều này không đúng đối với cùng thuộc tính NAME, nó là tên nhân viên trong

quan hệ EMP, là tên phòng ban trong quan hệ DEPT và là tên nhà cung cấp trong quan hệ SUPPLIER. Cũng lưu ý rằng phân mảnh ngang của DEPT cũng giống với phân mảnh ngang của một phần của EMP; cũng như thế, quan hệ SUPPLY được phân mảnh ngang dẫn xuất theo SUPPLIER. Các điều này có tầm rất quan trọng trong việc xác định các chiến lược thích hợp để thực hiện các phép kết mà chúng ta sẽ thấy.

3.3 TÍNH TRONG SUỐT PHÂN MẢNH DÙNG CHO CÁC ỨNG DỤNG CHỈ ĐỌC

Trong phần này, chúng ta phân tích các mức trong suốt khác nhau có thể được cung cấp bởi một DBMS cho các ứng dụng chỉ đọc.

3.3.1 Một ứng dụng đơn giản

Trước tiên, chúng ta xét một ứng dụng đơn giản, được gọi là SUPINQUIRY, bao gồm việc nhập vào một mã nhà cung cấp từ một thiết bị đầu cuối, tìm kiếm tên của nhà cung cấp này và trình bày tên này tại thiết bị đầu cuối. Chúng ta sẽ phân tích ứng dụng này nhìn thấy CSDL như thế nào ở các mức trong suốt thấp hơn.

Ứng dụng được viết theo ngôn ngữ tựa-Pascal (*Pascal-like language*) có các lệnh SQL nhúng (*embedded SQL statement*) dùng để truy xuất CSDL. Sự kết hợp của các ngôn ngữ này sẽ làm cho chúng ta dễ hiểu các chương trình mà không cần người đọc hiểu quá nhiều chi tiết về ngôn ngữ lập trình. Một số ít chi tiết cần thiết sẽ được diễn giải khi chúng được sử dụng trong các ví dụ. Chúng ta sẽ bỏ qua phần khai báo các biến trong các ví dụ tựa-Pascal. Tất cả các biến được giả sử là các chuỗi ký tự. Phần nhập - xuất sẽ được thực hiện bằng cách sử dụng các thủ tục chuẩn *read (terminal, variable)* và *write (terminal, variable)*. Nếu việc nhập - xuất được thực hiện tại thiết bị đầu cuối thì tên tập tin *terminal* sẽ được sử dụng.

Một lệnh SQL trong ngôn ngữ này định nghĩa một tác vụ cơ bản truy xuất CSDL; nó được xem là một yêu cầu của một thủ tục: nhận một số tham số từ chương trình tựa-Pascal, truy xuất CSDL và trả về kết quả như là một tham số. Để chỉ ra các tham số nhập và tham số xuất cho các thủ tục truy xuất CSDL, các biến Pascal được dùng làm tham số

trong các lệnh SQL có tiếp đầu ngữ là '\$'. Lưu ý rằng các tham số nhập của một truy vấn SQL xuất hiện trong mệnh đề *where*, trong khi đó các tham số xuất xuất hiện trong mệnh đề *select*. Ví dụ, truy vấn:

```
select NAME into $NAME
from SUPPLIER
where SNUM = $SNUM;
```

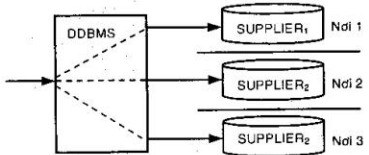
có thể được xem là một thủ tục nhận biến \$SNUM như là một tham số, chọn tên của nhà cung cấp có mã là giá trị của \$SNUM và trả về tên này cho chương trình tựa-Pascal thông qua biến \$NAME. Trong phần 3.5, chúng ta sẽ xét một trường hợp tổng quát mà lệnh SQL trả về các tập hợp (hoặc các quan hệ) mà không là các biến.

Trong một số trường hợp, ứng dụng cũng có liên lạc với DDBMS để trao đổi một số thông tin điều khiển, ví dụ thông tin về sự thành công hoặc thất bại của một tác vụ CSDL được yêu cầu. Các biến Pascal được sử dụng cho loại liên lạc này có tiếp đầu ngữ là '#'. Ví dụ, ngay sau khi thực hiện một truy vấn, chúng ta thường kiểm tra biến luận lý #FOUND, là *true* chỉ khi kết quả của truy vấn là khác rỗng. Một biến tiêu biểu khác của loại này là #OK, là *true* chỉ khi tác vụ đã được thực hiện đúng bởi DBMS (các biến này thường được gọi là 'các mã điều kiện' trong các DBMS).

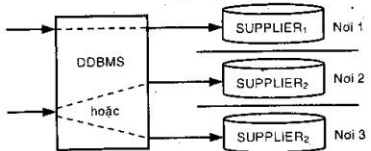
Mức 1 – Trong suốt phân mảnh

Hình 3.6a cho thấy cách thức mà ứng dụng truy xuất CSDL khi DDBMS cung cấp trong suốt phân mảnh. Trước tiên, ứng dụng nhận một mã nhà cung cấp từ thiết bị đầu cuối; sau đó nó truy xuất CSDL. Toàn bộ lệnh SQL biểu diễn một tác vụ cơ bản truy xuất CSDL, nhận tham số nhập là biến \$SNUM và trả về tham số xuất là biến \$NAME. DDBMS diễn dịch tác vụ cơ bản này bằng cách truy xuất các CSDL tại một nơi bất kỳ trong ba nơi, mà nơi này hoàn toàn được xác định bởi hệ thống. Xuất phát từ quan điểm của tính trong suốt phân bố, lưu ý rằng ứng dụng tham chiếu đến tên quan hệ toàn cục SUPPLIER mà hoàn toàn không quan tâm đến CSDL được phân bố. Bằng cách này, ứng dụng hoàn toàn không bị ảnh hưởng bởi bất kỳ thay đổi nào trong tất cả các lược đồ nằm dưới lược đồ toàn cục trong kiến trúc tham khảo.

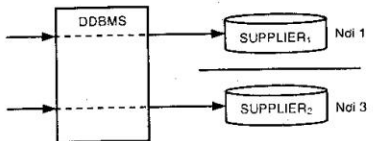
```
read (terminal, $SNUM);
select NAME into $NAME
from SUPPLIER
where SNUM = $SNUM;
if #FOUND then
    write (terminal, $NAME)
else write (terminal, 'Not found');
(a) Trong suốt phân mảnh (mức 1)
```



```
read (terminal, $SNUM);
select NAME into $NAME
from SUPPLIER1
where SNUM = $SNUM;
if not #FOUND then
    select NAME into $NAME
    from SUPPLIER2
    where SNUM = $SNUM;
if #FOUND then
    write (terminal, $NAME)
else write (terminal, 'Not found');
(b) Trong suốt vị trí (mức 2)
```



```
read (terminal, $SNUM);
select NAME into $NAME
from SUPPLIER1 at site 1
where SNUM = $SNUM;
if not #FOUND then
    select NAME into $NAME
    from SUPPLIER2 at site 3
    where SNUM = $SNUM;
if #FOUND then
    write (terminal, $NAME)
else write (terminal, 'Not found');
(c) Trong suốt ánh xạ cục bộ (mức 3)
```



Hình 3.6 Ứng dụng chỉ đọc SUPINQUIRY ở các mức trong suốt khác nhau

Mức 2 - Trong suốt vị trí

Nếu DDBMS cung cấp tính trong suốt vị trí mà không có tính trong suốt phân mảnh, cùng ứng dụng này có thể được viết như trong hình 3.6b. Trước tiên, mã nhà cung cấp được tìm kiếm trong mảnh SUPPLIER₁, và nếu DDBMS trả về biến điều khiển #FOUND là *false* thì việc tìm kiếm này được thực hiện tương tự đối với mảnh SUPPLIER₂. Tất nhiên, việc tìm kiếm này có thể được thực hiện theo nhiều cách khác nhau; ví dụ, có thể tìm kiếm mã nhà cung cấp một cách đồng thời trong SUPPLIER₁, và trong SUPPLIER₂; điều này nhằm để khai thác cơ chế song song hóa của hệ thống phân bố; tuy nhiên, điều này không thay đổi đặc tính phân bố trong suốt. Rõ ràng, ứng dụng này độc lập với những thay đổi trong lược đồ định vị, nhưng nó phụ thuộc vào những thay đổi trong lược đồ phân mảnh, bởi vì cấu trúc phân mảnh được kết hợp chặt chẽ trong ứng dụng. Tuy nhiên, tính trong suốt vị trí rất có ích, bởi vì nó cho phép các ứng dụng không xét đến mỗi mảnh có các bản nhân nào, do đó nó cho phép các bản nhân được di chuyển từ nơi này đến nơi khác và cho phép tạo các bản nhân mới mà không ảnh hưởng đến ứng dụng.

Khi có tính trong suốt vị trí mà không có tính trong suốt phân mảnh thì các ứng dụng sẽ chạy rất hiệu quả trên cấu trúc phân mảnh. Ví dụ, cùng ứng dụng của Hình 3.6b có thể được viết như sau khi biết thêm thành phố của nhà cung cấp:

SUPINQUIRY:

read (terminal, \$SNUM);

read (terminal, \$CITY);

case \$CITY of

'SF': select NAME into \$NAME
from SUPPLIER1

where SNUM = \$SNUM;

'LA': select NAME into \$NAME
from SUPPLIER2

where SNUM = \$SNUM

end;

if #FOUND then write (terminal, \$NAME)

else write (terminal, 'Not found');

Trong trường hợp này, các tác vụ cơ bản truy xuất CSDL cũng giống như trong hình 3.6b; tuy nhiên, ứng dụng chỉ tham chiếu đến một trong hai mảnh. Theo cách này, ứng dụng được viết dựa trên cùng mức trong suốt phân bố được cung cấp bởi DDBMS như trong hình 3.6b (nghĩa là mức trong suốt vị trí); tuy nhiên, nó phụ thuộc nhiều vào lược đồ phân mảnh, bởi vì nó liên quan chặt chẽ với các hàng 'SF' và 'LA', là các giá trị của thuộc tính được sử dụng trong lược đồ phân mảnh. Do đó, một thay đổi của các giá trị này sẽ ảnh hưởng đến ứng dụng này, trong khi đó nó không ảnh hưởng đến ứng dụng của hình 3.6b.

Mức 3 - Trong suốt ánh xạ cục bộ

Tại mức này, giả sử ứng dụng vẫn tham chiếu đến các đối tượng bằng cách sử dụng các tên độc lập với các hệ thống cục bộ riêng biệt; tuy nhiên, cần phải chỉ định các đối tượng được đặt tại các nơi nào. Điều này được chỉ ra trong hình 3.6c. Tên của các nơi được chỉ ra trong các lệnh SQL bằng cách thêm mệnh đề *at* vào trong mệnh đề *from*. Rõ ràng, trong trường hợp này, mỗi tác vụ cơ bản truy xuất CSDL được dẫn đường (*routed*) bởi DDBMS để đến nơi chỉ định. Tuy nhiên, các tác vụ cơ bản này sử dụng các tên mảnh độc lập vị trí (*site-independent*). Nếu ánh xạ này không được cung cấp, ứng dụng sẽ sử dụng trực tiếp các tên tập tin trong các hệ thống cục bộ.

Khía cạnh quan trọng nhất của tính trong suốt ánh xạ cục bộ không phải là ánh xạ giữa các tên mảnh với các tên tập tin cục bộ, mà là ánh xạ từ các tác vụ cơ bản được sử dụng bởi chương trình ứng dụng thành các tác vụ cơ bản được sử dụng bởi DBMS cục bộ. Do đó, tính trong suốt cục bộ là một đặc điểm quan trọng trong DDBMS không đồng nhất. Ví dụ, giả sử DBMS cục bộ tại nơi 1 là IMS và DBMS cục bộ tại nơi 3 là hệ thống Codasyl. Để cung cấp tính trong suốt ánh xạ cục bộ, DDBMS phải chuyển đổi các tác vụ cơ bản truy xuất CSDL được phát ra bởi ứng dụng đưa vào trong các chương trình IMS và

Codasyl tương ứng. Các vấn đề biên dịch tương tự cũng có thể thấy trong một DDBMS không đồng nhất, bởi vì ngay cả trong các hệ thống đồng nhất thì cũng có sự khác biệt giữa các tác vụ cơ bản được sử dụng ở mức toàn cục với các tác vụ cơ bản của DBMS cục bộ.

Mức 4 – Không có trong suốt

Chúng ta đã thấy rằng tính trong suốt ánh xạ cục bộ là đặc biệt quan trọng nhưng khó có được trong một hệ thống không đồng nhất. Do đó, ứng dụng SUPINQUIRY có thể được viết như thế nào trên CSDL phân bố với IMS tại nơi 1 và Codasyl tại nơi 3 nếu DDBMS không hỗ trợ bất kỳ tính trong suốt phân bố nào. Người lập trình ứng dụng phải viết các chương trình IMS và Codasyl để thực hiện các chức năng cần thiết và cài đặt các chương trình phụ trợ này tại các nơi mà chúng được yêu cầu; ứng dụng gửi một yêu cầu chạy các chương trình phụ trợ từ xa này thay vì các lệnh SQL, như đã được chỉ ra trong hình 3.7. Trong ví dụ này, chúng ta đã giả sử DDBMS chấp nhận một tác vụ cơ bản *execute* để xác định chương trình phải được chạy tại nơi nào và các tham số nào được truyền đi. Chi tiết của các chương trình IMS và Codasyl không được nêu ra, bởi vì điều này đòi hỏi phải định nghĩa các lược đồ ánh xạ cục bộ và đưa vào nhiều chi tiết không cần thiết của các hệ thống IMS và Codasyl. Lưu ý rằng biến #FOUND, là mã điều kiện trong các ví dụ trước, đã được thay đổi thành tham số \$FOUND. Trong trường hợp này, thay đổi này là do DDBMS không đủ thông tin về ngữ nghĩa của các tác vụ cơ bản để trả về kiểu của mã điều kiện này; các chương trình ứng dụng phụ trợ sẽ phải xác định các điều kiện này và trả chúng về một cách tường minh. Một lần nữa, không phụ thuộc vào các chi tiết ngôn ngữ lập trình, ví dụ này cho thấy một ưu điểm tiêu biểu khác của tính trong suốt phân bố: DDBMS có thể xác định và quản lý nhiều điều kiện hơn một cách độc lập ở các mức trong suốt cao hơn.

SUPINQUIRY:

read (terminal, \$\$SUPNUM);

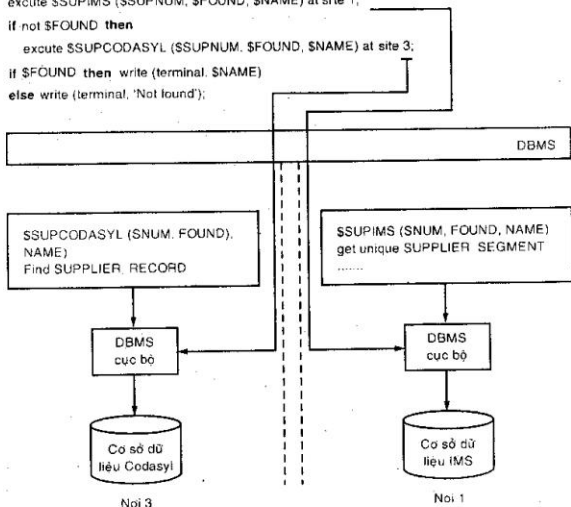
excute \$\$SUPIMS (\$\$SUPNUM, \$FOUND, \$NAME) at site 1;

if not \$FOUND then

excute \$\$SUPCODASYL (\$\$SUPNUM, \$FOUND, \$NAME) at site 3;

if \$FOUND then write (terminal, \$NAME)

else write (terminal, 'Not found');



Hình 3.7 Một ứng dụng trên một cơ sở dữ liệu phân bố không đồng nhất không có trong suốt

Trong ví dụ ở trên, DDBMS có một công việc không hoàn toàn hiển nhiên, đó là chạy các chương trình từ xa và truyền các tham số giữa các chương trình chạy trên các DBMS cục bộ khác nhau. Lưu ý rằng DDBMS có thể rất có ích trong việc phát triển các ứng dụng CSDL phân bố, ngay cả khi nó không cung cấp bất kỳ tính trong suốt nào.

3.3.2 Một ứng dụng phức tạp hơn

Các mức trong suốt phân bố dùng cho các ứng dụng đã được giới thiệu thông qua một ví dụ rất đơn giản mà nó thực hiện việc lấy dữ liệu từ một quan hệ mà thôi; sự khác biệt giữa các mức trong suốt khác nhau sẽ rõ ràng hơn nếu chúng ta xét một ứng dụng phức

tạp hơn mà nó thực hiện một phép kết giữa hai quan hệ khác nhau. Do đó, chúng ta xét một ứng dụng thứ hai, SUPOFPART dùng để lấy tên của nhà cung cấp mà họ cung cấp một mặt hàng cho trước; mã mặt hàng được nhập vào bởi người sử dụng chạy ứng dụng này. Ứng dụng được viết với giả sử mỗi mặt hàng chỉ được cung cấp bởi một nhà cung cấp.

Mức 1 – Trong suốt phân mảnh

Hình 3.8a cho thấy ứng dụng được viết ở mức này. Không có gì mới so với ứng dụng trước ở cùng mức này, ngoại trừ truy vấn toàn cục chứa một phép kết.

Mức 2 – Trong suốt vị trí

Nếu hệ thống chỉ cung cấp tính trong suốt vị trí thì ứng dụng có thể được viết theo nhiều cách khác nhau với hiệu quả khác nhau. Một giải pháp đơn giản bao gồm bốn truy vấn khác nhau, mỗi truy vấn tương ứng với mỗi cặp mảnh khác nhau của SUPPLIER và SUPPLY, do đó áp dụng một chiến lược vét cạn: mỗi mảnh SUPPLY được kết với mỗi mảnh SUPPLIER để tìm ra tên của nhà cung cấp mà họ cung cấp mà mặt hàng cho trước.

Một giải pháp thông minh hơn, được chỉ ra trong hình 3.7b, có ưu điểm là xét nội dung thông tin của lược đồ phân mảnh, đó là các nhà cung cấp của các mặt hàng được chứa trong SUPPLY₁ (SUPPLY₂) thì được chứa trong SUPPLIER₁ (SUPPLIER₂). Bằng cách này, chỉ cần có hai truy vấn SQL thay vì bốn truy vấn. Trước tiên, chương trình này nhập một mã mặt hàng từ thiết bị đầu cuối, và sau đó kết mảnh SUPPLIER₁ với các bộ của mảnh SUPPLY₁ để tìm mã mặt hàng này. Nếu kết quả của truy vấn đầu tiên là rỗng (*not #FOUND*) thì một tác vụ tương tự được áp dụng cho các mảnh SUPPLIER₂ và SUPPLY₂.

Ví dụ này cho thấy rằng trong một hệ thống không cung cấp tính trong suốt phân mảnh, người lập trình ứng dụng phải xác định chiến lược để thực hiện các tác vụ, chẳng hạn như các phép kết (trong trường hợp này, người lập trình ứng dụng đã quyết định chỉ có hai phép kết giữa các mảnh thay vì bốn phép kết); ngược lại, ví dụ cho thấy rằng để cung cấp tính trong suốt phân mảnh, hệ thống phải có khả năng tự động chọn các chiến lược truy xuất khác nhau. Do đó, một vấn đề thật sự quyết định trong việc cung cấp tính

trong suốt không phải là sự diễn giải của một ánh xạ (lược đồ phân mảnh) mà là việc chọn một chiến lược truy xuất tốt. Ví dụ, một DDBMS cung cấp tính trong suốt phân mảnh và cho phép viết ứng dụng SUPOFPART như trong Hình 3.8a, nhưng thực hiện bốn phép kết giữa các mảnh thì rất kém hiệu quả so với hiệu quả có được bởi một người lập trình (thông minh).

Mức 3 – Trong suốt ánh xạ cục bộ

Chúng ta xét cùng ứng dụng SUPOFPART sẽ được viết như thế nào nếu DDBMS không cung cấp tính trong suốt vị trí. Giả sử lược đồ định vị của các mảnh của SUPPLY như sau:

SUPPLIER₁: site 1

SUPPLIER₂: site 2

SUPPLY₁: site 3

SUPPLY₂: site 4

Bây giờ, ứng dụng có thể được viết như trong hình 3.8c. Chương trình này nhập một mã nhà cung cấp và tìm nhà cung cấp của mặt hàng này trong mảnh SUPPLY₁ ở nơi 3. Nếu tìm thấy mã nhà cung cấp (#FOUND) thì ứng dụng gửi mã nhà cung cấp đến nơi 1 (lệnh *send*) để tìm tên nhà cung cấp trong mảnh SUPPLIER₁ đặt tại nơi 1; nếu không tìm thấy mã nhà cung cấp tại nơi 3 thì cùng những tác vụ này được thực hiện trên mảnh SUPPLY₂ tại nơi 4 và trên mảnh SUPPLIER₂ tại nơi 2.

```
read (terminal, $PNUM);
select NAME into $NAME
from SUPPLIER, SUPPLY
where SUPPLIER.SNUM = SUPPLY.SNUM
and SUPPLY.PNUM = $PNUM;
If #FOUND then write (terminal, $NAME)
else write (terminal, 'Not found');
```

a) Trong suốt phân mảnh (mức 1)

```
read (terminal, $PNUM);
select NAME into $NAME
from SUPPLIER1, SUPPLY1
where SUPPLIER1.SNUM = SUPPLY1.SNUM
and SUPPLY1.PNUM = $PNUM;
```

```

if not #FOUND then
    select NAME into $NAME
    from SUPPLIER2, SUPPLY2
    where SUPPLIER2.SNUM = SUPPLY2.SNUM
    and SUPPLY2.PNUM = $PNUM;
if #FOUND then write (terminal, $NAME)
else write (terminal, 'Not found');

```

b) Trong suốt vị trí (mức 2)

```

read (terminal, $PNUM);
select SNUM into $SNUM
from SUPPLY1 at site 3
where PNUM = $PNUM;
if #FOUND then
    begin
        send $SNUM from site 3 to site 1;
        select NAME into $NAME
        from SUPPLIER1 at site 1
        where SNUM = $SNUM
    end
else
    begin
        select SNUM into $SNUM
        from SUPPLY2 at site 4
        where PNUM = $PNUM;
        if #FOUND then
            begin
                send $SNUM from site 4 to site 2;
                select NAME into $NAME
                from SUPPLIER2 at site 2
                where SNUM = $SNUM
            end
        end;
    end;
if #FOUND then write (terminal, $NAME)
else write (terminal, 'Not found');

```

c) Trong suốt ánh xạ cục bộ (mức 3)

Hình 3.8 Ứng dụng chỉ đọc SUPQFPART ở các mức trong suốt phân bố khác nhau

Lưu ý rằng các truy vấn SQL được sử dụng trong ứng dụng của ví dụ trước, để thực hiện một phép kết giữa hai mảnh có thể được đặt tại các nơi khác nhau, đã được tách ra trong ví dụ này. Trên thực tế, trong ví dụ này, chúng ta giả sử DDBMS yêu cầu mỗi tác vụ cơ bản được thực hiện chỉ tại một nơi, và do đó nó không thể thực hiện một phép kết giữa hai mảnh đặt tại các nơi khác nhau; điều này là một giả sử có lý đối với một DDBMS không cung cấp tính trong suốt vị trí.

Các lệnh *send* thật sự không cần thiết; nếu các tác vụ cơ bản *send* không được định nghĩa tường minh thì chúng ta giả sử các biến \$SNUM, được tạo ra bởi truy vấn thứ nhất, được trả về ngầm định cho ứng dụng và sau đó nó được gửi cho các nơi thực hiện của truy vấn thứ hai. Lệnh *send* cho phép liên lạc trực tiếp giữa các nơi mà hai truy vấn được thực hiện. Điều này có thể rất quan trọng nếu kết quả trung gian (trong trường hợp này là \$SNUM) không là biến đơn mà là cả một tập tin.

Ứng dụng sau cùng này rõ ràng phụ thuộc nhiều vào nội dung của các lược đồ; mọi thay đổi trong phân mảnh dữ liệu và định vị dữ liệu đều dẫn đến sự thay đổi của ứng dụng.

Chúng ta sẽ không giới thiệu ứng dụng SUPOFPART trên hệ thống không đồng nhất và không có tính trong suốt ánh xạ cục bộ, bởi vì điều này đòi hỏi phải giải quyết nhiều chi tiết về ngôn ngữ lập trình mà nó không thích hợp ở đây.

Bằng cách so sánh ứng dụng trong ví dụ thứ nhất (SUPINQUIRY) với ứng dụng trong ví dụ thứ hai (SUPOFPART) ở mức trong suốt ánh xạ cục bộ, chúng ta lưu ý rằng tính trong suốt phân bố là rất quan trọng đối với ứng dụng có phép kết. Viết lại các ứng dụng, chẳng hạn SUPOFPART, do hệ quả của sự di chuyển của một mảnh, là một công việc rất khó, hầu như là viết một ứng dụng hoàn toàn mới.

3.4 TÍNH TRONG SUỐT PHÂN MẢNH DÙNG CHO CÁC ỨNG DỤNG CẬP NHẬT

Trong phần trước, chúng ta chỉ xét các ứng dụng thực hiện việc lấy dữ liệu từ CSDL. Trong phần này, chúng ta xét các ứng dụng thực hiện việc cập nhật CSDL phân bố. Ở đây, vấn đề cập nhật chỉ được

xét trên quan điểm của tính trong suốt phân bố đối với người lập trình ứng dụng, trong khi đó chúng ta không đề cập đến các vấn đề về việc bảo đảm tính nguyên tố của các giao dịch cập nhật.

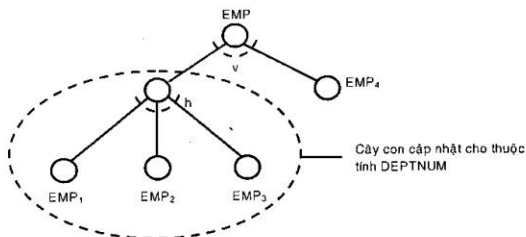
Cùng các mức trong suốt phân bố, mà chúng ta đã phân tích đối với các ứng dụng chỉ đọc, cũng có thể được xét đối với các ứng dụng cập nhật. Tuy nhiên, một cập nhật phải được thực hiện trên tất cả các bản sao của một mục dữ liệu (*data item*), trong khi đó việc lấy dữ liệu có thể được thực hiện trên bất kỳ một bản sao nào (*read-one write-all*). Điều này có nghĩa là nếu một DDBMS không cung cấp tính trong suốt vị trí và tính trong suốt nhân bản thì người lập trình ứng dụng phải có nhiệm vụ thực hiện tất cả các cập nhật cần thiết. Do đó, tính trong suốt vị trí và tính trong suốt nhân bản là quan trọng hơn đối với các ứng dụng cập nhật so với các ứng dụng chỉ đọc.

Trong việc cung cấp tính trong suốt phân bố cho ứng dụng cập nhật, có một vấn đề khác phức tạp hơn so với việc thực hiện các cập nhật trên tất cả các bản sao của một mục dữ liệu. Ví dụ, điều gì sẽ xảy ra nếu thuộc tính CITY của một nhà cung cấp bị thay đổi. Rõ ràng, một bộ (*tuple*) của nhà cung cấp này phải bị di chuyển từ một mảnh này qua một mảnh khác. Hơn nữa, trong CSDL của ví dụ, các bộ của quan hệ SUPPLY thuộc nhà cung cấp này cũng phải thay đổi mảnh, bởi vì quan hệ SUPPLY được phân mảnh dẫn xuất theo quan hệ SUPPLIER. Rõ ràng một cách trực quan, việc thay đổi giá trị của một thuộc tính được sử dụng trong định nghĩa lược đồ phân mảnh (được gọi là *thuộc tính phân mảnh*) có thể có nhiều ảnh hưởng tương đối phức tạp và các ảnh hưởng này được quản lý bởi DDBMS. Mức độ của các ảnh hưởng này là điểm đặc trưng của mức trong suốt phân bố dùng cho các cập nhật.

Để hiểu cần phải có các tác vụ di chuyển dữ liệu nào khi cập nhật một thuộc tính phân mảnh (*fragmentation attribute*) thì chúng ta cần nhìn vào cây phân mảnh (*fragmentation tree*).

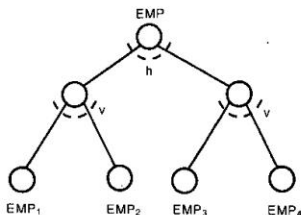
Xét thuộc tính A được sử dụng trong vị từ chọn (*selection predicate*) của phân mảnh ngang. *Cây cập nhật* (*update tree*) của A là

cây con có nút gốc là nút biểu diễn phân mảnh ngang ở trên. Ví dụ, hình 3.9 cho thấy cây con cập nhật cho DETPNUM. Các ảnh hưởng của việc thay đổi một thuộc tính được giới hạn cho các mảnh là các nút lá của cây con cập nhật của nó. Ví dụ, việc thay đổi thuộc tính DEPTNUM chỉ ảnh hưởng đến EMP₁, EMP₂ và EMP₃, nhưng không ảnh hưởng đến EMP₄. Một bộ có thể di chuyển (thay đổi các mảnh) giữa hai trong ba mảnh do hệ quả của việc cập nhật.



Hình 3.9 Cây con cập nhật cho thuộc tính DEPTNUM trong cây phân mảnh của quan hệ EMP

Nói chung, cây con cập nhật của một thuộc tính A có thể phức tạp hơn so với cây con cập nhật đơn giản được chỉ ra trong Hình 3.9 và do đó, ảnh hưởng của việc thay đổi một giá trị của thuộc tính A là phức tạp hơn. Ví dụ, giả sử quan hệ EMP có cây phân mảnh được chỉ ra trong hình 3.10a. Trong trường hợp này, cây con cập nhật của thuộc tính DEPTNUM là toàn bộ cây phân mảnh. Ảnh hưởng của việc thay đổi giá trị DEPTNUM của một bộ có EMPNUM = 100 từ 3 thành 15 được chỉ ra trong Hình 3.10b. Việc cập nhật này làm cho một bộ, ban đầu thuộc về các mảnh của cây con bên trái của Hình 3.10a, sau đó nó thuộc về cây con bên phải. Chúng ta thấy rằng không chỉ dữ liệu bị di chuyển giữa các mảnh mà một bộ cũng bị sắp đặt lại theo một cách khác.



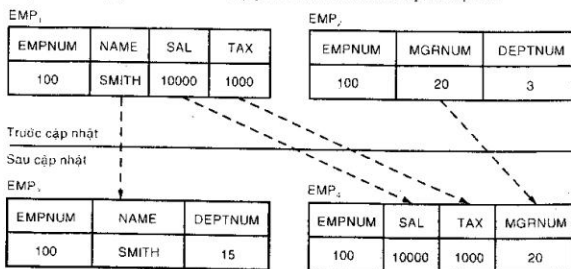
$$EMP_1 = \Pi_{EMPNUM, NAME, SAL, TAX} \sigma_{DEPTNUM = 10} (EMP)$$

$$EMP_2 = \Pi_{EMPNUM, MGRNUM, DEPTNUM} \sigma_{DEPTNUM \leq 10} (EMP)$$

$$EMP_3 = \Pi_{EMPNUM, NAME, DEPTNUM} \sigma_{DEPTNUM > 10} (EMP)$$

$$EMP_4 = \Pi_{EMPNUM, SAL, TAX, MGRNUM} \sigma_{DEPTNUM > 10} (EMP)$$

a) Sự phân mảnh và cây phân mảnh khác của quan hệ EMP



b) Ảnh hưởng của việc cập nhật DEPTNUM của nhân viên có EMPNUM = 100

Hình 3.10 Một ứng dụng cập nhật

Bây giờ, chúng ta có thể xét các mức trong suốt phân bố dùng cho một ứng dụng cập nhật đơn giản.

Mức 1 – Trong suốt phân mảnh

Ở mức này, người lập trình ứng dụng thực hiện việc cập nhật giống như khi CSDL không được phân bố; do đó, người lập trình ứng dụng không cần biết một thuộc tính nào được sử dụng trong định

nghĩa của lược đồ phân mảnh hay không. Để thay đổi DEPTNUM của nhân viên có EMPNUM = 100, người lập trình viết chương trình ứng dụng được chỉ ra trong hình 3.11a. Nhiệm vụ của hệ thống là thực hiện tất cả các tác vụ được yêu cầu một cách ngầm định bởi lược đồ phân mảnh và lược đồ định vị.

```
update EMP
set DEPTNUM = 15
where EMPNUM = 100;
```

a) Trong suốt phân mảnh (mức 1)

```
select NAME, SAL TAX into $NAME, $SAL, $TAX
from EMP1
where EMPNUM = 100;
if #FOUND then
begin
    select MGRNUM into $MGRNUM
    from EMP2
    where EMPNUM = 100;
    insert into EMP3 (EMPNUM, NAME, DEPTNUM)
    values (100, $NAME, 15);
    insert into EMP4 (EMPNUM, SAL, TAX, MGRNUM)
    values (100, $SAL, $TAX, $MGRNUM);
    delete from EMP1
    where EMPNUM = 100;
    delete from EMP2
    where EMPNUM = 100
end;
```

b) Trong suốt vị trí (mức 2)

```
select NAME, SAL TAX into $NAME, $SAL, $TAX
from EMP1 at site 1
where EMPNUM = 100;
if #FOUND then
begin
    select MGRNUM into $MGRNUM
    from EMP2 at site 2
```

```

where EMPNUM = 100;
insert into EMP3 (EMPNUM, NAME, DEPTNUM) at site 3
values (100, $NAME, 15);
insert into EMP3 (EMPNUM, NAME, DEPTNUM) at site 7
values (100, $NAME, 15);
insert into EMP4 (EMPNUM, SAL, TAX, MGRNUM) at site 4
values (100, $SAL, $TAX, $MGRNUM);
insert into EMP4 (EMPNUM, SAL, TAX, MGRNUM) at site 8
values (100, $SAL, $TAX, $MGRNUM);
delete from EMP1 at site 1
where EMPNUM = 100;
delete from EMP1 at site 5
where EMPNUM = 100;
delete from EMP2 at site 2
where EMPNUM = 100;
delete from EMP2 at site 6
where EMPNUM = 100
end;

```

c) Trong suốt ánh xạ cục bộ (mức 3)

Hình 3.11 Một ứng dụng cập nhật ở các mức trong suốt phân bố khác nhau

Mức 2 – Trong suốt vị trí

Ở mức này, người lập trình ứng dụng phải giải quyết tường minh với các mảnh. Giả sử sự phân mảnh như trong hình 3.10, cùng ứng dụng này được viết như trong hình 3.11b. Hai lệnh *select* đầu tiên thu thập bộ EMP cần thiết từ các mảnh chứa nó. Hai lệnh *insert* sắp đặt lại bộ này theo một cách khác và đưa nó vào trong hai mảnh khác. Hai lệnh *delete* xóa bản sao cũ của bộ này. Lưu ý rằng bốn lệnh cuối cùng có thể được thực hiện một cách song song hoặc theo một thứ tự bất kỳ.

Ứng dụng ở trên được viết với giả sử rất đơn giản là chỉ thay đổi DEPTNUM của một nhân viên có EMPNUM = 100; tuy nhiên, một ứng dụng thật sự phải có tham số tương ứng với EMPNUM và DEPTNUM, thực hiện việc nhập dữ liệu chẳng hạn như: nhập mã

nhân viên và mã phòng ban từ thiết bị đầu cuối và cập nhật mã phòng ban của nhân viên có mã nhân viên này. Trong trường hợp này, ứng dụng sẽ phức tạp hơn, phải tìm mảnh nào có chứa bộ của nhân viên, đọc DEPTNUM cũ của nhân viên, xác định sự thay đổi của DEPTNUM để thay đổi mảnh mới chứa bộ này, và cuối cùng thực hiện các lệnh thêm vào và lệnh loại bỏ cần thiết. Các lệnh thêm vào và lệnh loại bỏ cần thiết là khác nhau đối với các mảnh đầu tiên và các mảnh cuối cùng chứa bộ này; do đó, ứng dụng phải chứa mã lệnh để giải quyết tất cả các cách làm có thể có. Ví dụ này cho thấy tính trong suốt phân mảnh có ích lợi như thế nào đối với người lập trình ứng dụng trong trường hợp ứng dụng cập nhật, nhưng nó cũng ngầm định cho thấy sự phức tạp như thế nào đối với một DDMBS để cung cấp đặc tính này.

Mức 3 – Trong suốt ánh xạ cục bộ

Ở mức này, ứng dụng phải giải quyết tường minh với vị trí của các mảnh. Trong trường hợp ứng dụng cập nhật, điều này có nghĩa là nó phải quan tâm đến sự nhân bản. Giả sử các mảnh của quan hệ EMP được đặt theo cách sau:

EMP₁: nơi 1 và 5

EMP₂: nơi 2 và 6

EMP₃: nơi 3 và 7

EMP₄: nơi 4 và 8

Ứng dụng trên phải được thay đổi như trong hình 3.11c; hai lệnh *select* đầu tiên có chứa mệnh đề *at* để chọn nơi truy xuất EMP₁ và EMP₂; hai lệnh *insert* và hai lệnh *delete* dùng để thêm các bộ mới vào tất cả các bản nhân của các mảnh chứa các bộ này và để loại bỏ các bộ cũ trong tất cả các bản nhân chứa các bộ này.

Cuối cùng, chúng ta không giới thiệu một ví dụ cho trường hợp không có tính trong suốt ánh xạ cục bộ, bởi vì cùng các nghiên cứu đã được nói đến trong phần trước được áp dụng cho trường hợp này.

Lưu ý quan trọng là khi cập nhật dữ liệu, chúng ta phải bảo đảm dữ liệu mới phải thỏa mãn tất cả các ràng buộc toàn vẹn có trong CSDL, kể cả CSDL tập trung và CSDL phân bố. Trong CSDL phân bố, việc cập nhật dữ liệu sẽ dẫn đến việc di chuyển dữ liệu từ

các mảnh này qua các mảnh khác chỉ xảy ra khi thuộc tính bị cập nhật có trong vị từ định tính của mảnh có dữ liệu bị cập nhật và giá trị mới phải làm thay đổi giá trị của vị từ này đang là *true* bị trở thành *false*; các mảnh mới, chứa dữ liệu được di chuyển đến, là các mảnh mà giá trị mới làm cho vị từ định tính của chúng có giá trị *true*. Ngoài ra, trong ví dụ trên, nếu chúng ta có phụ thuộc hàm $DEPTNUM \rightarrow MGRNUM$, thì chương trình cập nhật ở trên chưa thay đổi giá trị $MGRNUM$ của nhân viên có mã 100 thành giá trị mới.

3.5 CÁC TÁC VỤ CƠ BẢN TRUY XUẤT CSDL PHÂN BỐ

Trong các ví dụ của các phần trước, chúng ta luôn luôn giả sử mỗi tác vụ cơ bản truy xuất CSDL (truy vấn) đều trả về kết quả là một giá trị đơn (*single value*) được gán vào một biến đơn (*simple variable*). Tuy nhiên, chúng ta muốn sử dụng các tác vụ cơ bản mạnh hơn mà chúng trả về kết quả là các quan hệ; trong trường hợp này, chương trình thủ tục tựa-Pascal và các tác vụ cơ bản SQL thay thế các tham số là các biến đơn thành các tập tin (các tập tin tham số - *parameter file*). Để tránh vấn đề mô tả các kiểu, chúng ta sẽ sử dụng cách tiếp cận sau đây trong các ví dụ của chúng ta: các tham số có tên bắt đầu bằng tiếp vị ngữ REL được xem là thuộc kiểu tập tin trong chương trình tựa-Pascal và thuộc kiểu quan hệ trong các lệnh SQL, giả sử cấu trúc của chúng đã được khai báo. Ví dụ, lệnh SQL:

```
select EMPNUM, NAME into $EMP_REL ($EMPNUM, $NAME)
from EMP;
```

trả về cho chương trình tựa-Pascal một tập tin gồm các mẫu tin; tên tập tin là \$EMP_REL, và các mẫu tin gồm có hai trường là \$EMPNUM và \$NAME. Tất nhiên, \$EMP_REL chứa kết quả của phép chiếu của quan hệ EMP trên các thuộc tính EMPNUM và NAME.

Chúng ta xét các cách thức khác nhau mà người lập trình có thể viết cùng ứng dụng này, và phân tích hiệu quả của các cách này về mặt hiệu suất thực hiện ứng dụng. Xét chương trình ứng dụng lấy tất cả các mặt hàng được cung cấp bởi một tập hợp các nhà cung cấp. Người sử dụng chỉ ra tập hợp các nhà cung cấp bằng cách nhập vào các mã của họ tại thiết bị đầu cuối. Giả sử mỗi nhà cung cấp có thể cung cấp nhiều mặt hàng.

repeat

```

read (terminal, $SNUM);
delete from $PNUM_REL;
insert into $PNUM_REL ($PNUM)
select PNUM
from SUPPLY
where SNUM = $SNUM;
repeat
    read ($PNUM_REL, $PNUM);
    write (terminal, $SNUM, $PNUM)

```

until END-OF-\$PNUM_REL

until END-OF-TERMINAL-INPUT;

a) Cơ sở dữ liệu được truy xuất ứng với mỗi giá trị của \$SNUM

repeat

```

read (terminal, $SNUM);
insert into $SNUM_REL ($SNUM)
values ($SNUM)

```

until END-OF-TERMINAL-INPUT;

```

insert into $PNUM_REL ($SNUM, $PNUM)

```

```

select $SNUM, PNUM

```

```

from SUPPLY, $SNUM_REL

```

```

where SUPPLY.SNUM = $SNUM_REL.$SNUM;

```

repeat

```

read ($PNUM_REL, $SNUM, $PNUM);
write (terminal, $SNUM, $PNUM)

```

until END-OF-\$PNUM_REL;

b) Cơ sở dữ liệu được truy xuất sau khi nhập vào tất cả các giá trị của \$SNUM

```

insert into $TEMP_REL ($TEMP_PNUM, $TEMP_SNUM)

```

```

select PNUM, SNUM

```

```

from SUPPLY;

```

repeat

```

read (terminal, $SNUM);
delete from $TEMP2_REL;
insert into $TEMP2_REL ($TEMP2_PNUM)
select $TEMP_PNUM
from $TEMP_REL
where $TEMP_SNUM = $SNUM;

```

repeat

read (\$TEMP2_REL, \$TEMP2_PNUM);

write (terminal, \$\$NUM, \$TEMP2_PNUM)

until END-OF-\$TEMP2_REL

until END-OF-TERMINAL-INPUT;

c) Cơ sở dữ liệu được truy xuất trước khi nhập vào các giá trị của \$\$NUM

Hình 3.12 Các cách viết khác nhau của một chương trình ứng dụng với tính trong suốt phân mảnh

Cách viết thứ nhất của ứng dụng này được chỉ ra trong hình 3.12a. Cơ sở dữ liệu phân bố được truy xuất ứng với mỗi lần lặp của vòng lặp ngoài; đối với mỗi nhà cung cấp có mã \$\$NUM được người sử dụng nhập vào, các mặt hàng cung cấp sẽ được lấy ra và đưa vào tập tin tham số \$PNUM_REL. Vòng lặp bên trong quét tập tin tham số này và trình bày mỗi mã mặt hàng \$PNUM của các mặt hàng được chọn.

Cùng ứng dụng này có thể được viết như trong hình 3.12b. Trước tiên, ứng dụng này thu thập tất cả các mã nhà cung cấp và sau đó truy xuất CSDL; do đó việc lấy dữ liệu chỉ một lần với tập tin tham số lớn hơn so với ví dụ trước. Bằng cách này, số tác vụ truy xuất CSDL phân bố giảm, cùng một lượng dữ liệu kết quả được truyền về, nhưng ít thông điệp điều khiển được sử dụng. Ví dụ này cho thấy người lập trình có thể điều khiển việc truy xuất CSDL ngay cả khi họ làm việc ở mức toàn cục. Sự chọn lựa giữa hai giải pháp ở trên phụ thuộc vào số lượng các mặt hàng được cung cấp bởi mỗi nhà cung cấp và vào số lượng các nhà cung cấp mà ứng dụng dùng đến. Rõ ràng, một sự thỏa thuận (*trade-off*) giữa thời gian đáp ứng (*response time*) và toàn bộ thời gian tiêu tốn của hệ thống (*system overhead*) phải được xem xét: nếu muốn tối thiểu hóa (*minimization*) số lượng các thông điệp thì giải pháp thứ hai là tốt hơn; nếu muốn đáp ứng nhanh sau mỗi lần nhập mã nhà cung cấp thì giải pháp đầu tiên là tốt hơn. Nói cách khác, một ứng dụng có thể truy xuất các tập hợp lớn hơn nếu nó được kết bó (*batched*) và ít tương tác; sự thỏa thuận này giữa các giải pháp kết bó hơn và trực tuyến hơn cũng đã được biết nhiều trong các ứng dụng CSDL truyền thống.

Một giải pháp cuối cùng được minh họa bằng một chương trình trong hình 3.12c. Ở đây, truy vấn đầu tiên không có tham số nhập và lấy tất cả các dữ liệu cần thiết để thực hiện ứng dụng rồi đưa vào trong một quan hệ tạm thời \$TEMP_REL. Giả sử chỉ có một số ít bộ SUPPLY được lấy ra trong hai trường hợp trước, giải pháp này đòi hỏi truyền nhiều dữ liệu hơn. Tuy nhiên, giải pháp này có thể có ích nếu ứng dụng lấy nhiều bộ SUPPLY, bởi vì nó chỉ thực hiện một truy xuất CSDL phân bố và sau đó thực hiện việc lấy dữ liệu trong quan hệ tạm thời. Giải pháp này thường thích hợp nếu ứng dụng không đòi hỏi dữ liệu tức thời. Trong trường hợp này, quan hệ tạm thời biểu diễn một hình ảnh tĩnh của CSDL xảy ra tại một thời điểm đã cho trong quá khứ mà nó được sử dụng để thực hiện một ứng dụng cục bộ.

Một vấn đề liên quan trong việc viết các tác vụ cơ bản truy xuất CSDL là việc phát hiện sự lặp lại của các *biểu thức con chung* (*common subexpression*). Ví dụ, xét một ứng dụng lấy các tên nhân viên làm việc trong các phòng ban thuộc một vùng cho trước và các mã mặt hàng của các mặt hàng được lưu trữ tại các phòng ban thuộc cùng một vùng. Rõ ràng không có mối liên kết giữa nhân viên và các mặt hàng; do đó, cách thức tự nhiên nhất để viết ứng dụng này là viết hai truy vấn độc lập được thực hiện tuần tự với tham số nhập là cùng một vùng. Giải pháp này được chỉ ra trong hình 3.13a. Nhược điểm chính của giải pháp này là tập hợp các phòng ban thuộc một vùng cho trước được xác định hai lần và có thể được truyền đi hai lần giữa các nơi. Rất khó xác định một cách tự động từ chương trình của hình 3.13a một tác vụ chung trong việc thực hiện cả hai truy vấn. Để xác định thì cần phải tách biểu thức con chung và xác định biến \$AREA không bị thay đổi giá trị của nó giữa hai truy vấn. Điều này đòi hỏi sự phân tích phức tạp của chương trình thủ tục ở thời gian biên dịch (lưu ý rằng vấn đề tìm các biểu thức con chung trong một truy vấn đơn thì đơn giản hơn).

Để giải quyết vấn đề ở trên thì cần phải viết ứng dụng theo một cách khác, ví dụ như trong hình 3.13b. Tham số \$AREA chỉ được truyền một lần cho DDBMS và do đó rõ ràng chỉ có một giá trị chung cho cả hai truy vấn. Truy vấn đầu tiên trong giải pháp này thực hiện việc lấy dữ liệu giống với cách thức đã thực hiện bởi mỗi truy vấn trong giải pháp trước.

Cuối cùng, nếu giao tiếp ngôn ngữ của DDBMS không cung cấp các tác vụ kết hợp, hướng tập hợp (*set-oriented, associative operation*) mà chỉ cung cấp một ngôn ngữ thủ tục (*procedural language*) thì có thể thực hiện cái gọi là **biên dịch ngược** (*decompilation*). Biên dịch ngược bao gồm việc chuyển đổi một chương trình được viết bằng một ngôn ngữ thủ tục thành một tập hợp tương đương các truy vấn phi thủ tục. Biên dịch ngược là một vấn đề rất khó, bởi vì cần phải xuất phát từ cấu trúc tĩnh của các chương trình mà dữ liệu được truy xuất ở thời gian chạy (*run time*).

```
read (terminal, $AREA);
```

```
.....
insert into $NAME_REL ($NAME)
select NAME
from EMP, DEPT
where EMP.DEPTNUM = DEPT.DEPTNUM
and AREA = $AREA;
```

```
.....
insert into $PNUM_REL ($PNUM)
select PNUM
from SUPPLY, DEPT
where SUPPLY.DEPTNUM = DEPT.DEPTNUM
and AREA = $AREA;
```

a) Các truy vấn độc lập

```
read (terminal, $AREA);
```

```
.....
insert into $NUM_REL ($DEPTNUM)
select DEPTNUM
from DEPT
where AREA = $AREA;
```

```
.....
insert into $NAME_REL ($NAME)
select NAME
from EMP, $NUM_REL
where EMP.DEPTNUM = $NUM_REL.$DEPTNUM;
```

```

insert into $PNUM_REL ($PNUM)
select PNUM
from SUPPLY, $NUM_REL
where SUPPLY.DEPTNUM = $NUM_REL.$DEPTNUM;

```

b) Các truy vấn với sự định giá trước của một biểu thức con chung

Hình 3.13 Sử dụng các biểu thức con chung trong ứng dụng

3.6 CÁC RÀNG BUỘC TOÀN VỆN TRONG CSDL PHÂN BỐ

Trong một số hệ quản trị CSDL, chúng ta có thể đặc tả các *ràng buộc toàn vẹn* (*integrity constraint*); nghĩa là các qui định về các giá trị cho phép của dữ liệu. Các ràng buộc toàn vẹn cho biết các giá trị dữ liệu nào được cho phép (ví dụ tuổi là một số nguyên từ 0 đến 100) hoặc các biến đổi được cho phép (ví dụ tuổi không thể giảm), chúng có thể liên quan đến một quan hệ hoặc nhiều quan hệ. Khi một cập nhật được thực hiện bởi một ứng dụng CSDL vi phạm một ràng buộc toàn vẹn, ứng dụng sẽ không được chấp thuận và do đó tính đúng đắn của dữ liệu vẫn được bảo đảm.

Một ví dụ tiêu biểu của ràng buộc toàn vẹn mà chúng ta đã đề cập trong chương này là ràng buộc tham chiếu (*referential constraint*), đòi hỏi tất cả các giá trị của một thuộc tính cho trước của một quan hệ cũng tồn tại trong một quan hệ khác nào đó. Ràng buộc này đặc biệt có ích trong các CSDL phân bố để bảo đảm tính đúng đắn của phân mảnh ngang dẫn xuất. Ví dụ, bởi vì quan hệ SUPPLY được phân mảnh ngang dẫn xuất theo quan hệ SUPPLIER bởi phép nửa kết trên thuộc tính SNUM, do đó tất cả các giá trị của SNUM trong SUPPLY phải có trong SNUM của SUPPLIER.

Việc quản lý các ràng buộc toàn vẹn trong các hệ thống tập trung là điều rất khó và tốn kém. Do đó, rất ít hệ thống có các khả năng tạo sẵn (*built-in capability*) để định nghĩa và kích hoạt các ràng buộc toàn vẹn; vì thế, thông thường nhiệm vụ của người lập trình ứng dụng phải viết các chương trình bảo đảm tính đúng đắn của dữ liệu. Tuy nhiên, chúng ta cần phải đặc tả các ràng buộc toàn

ven một cách rõ ràng, ngay cả khi chúng không được các hệ thống hỗ trợ một cách trực tiếp để viết các ứng dụng một cách đúng đắn.

Các ràng buộc toàn vẹn có thể được kích hoạt một cách tự động bằng cách thêm vào các chương trình ứng dụng một mã lệnh nào đó để kiểm tra ràng buộc nào bị vi phạm. Nếu ràng buộc bị vi phạm thì chương trình bị ngừng thực hiện và tất cả các tác vụ đã được thực hiện sẽ bị hủy bỏ, nếu cần. Ví dụ, chúng ta xét việc loại bỏ một bộ trong quan hệ SUPPLIER. Trong SQL, chúng ta có thể viết ứng dụng này như sau:

```
delete from SUPPLIER
where SNUM = $SNUM;
```

Tác vụ này có thể vi phạm ràng buộc tham chiếu ở trên. Để cho ràng buộc này không bị vi phạm thì có thể sửa chương trình như sau:

```
select 1
from SUPPLY
where SNUM = $SNUM;
if not #FOUND then
    delete from SUPPLIER
    where SNUM = $SNUM;
```

Truy vấn SQL đầu tiên lấy các bộ của quan hệ SUPPLY có cùng mã nhà cung cấp như trong chương trình ban đầu, nếu #FOUND là *false* thì không có bộ nào trong quan hệ SUPPLY tham chiếu đến các bộ bị loại bỏ; do đó vẫn an toàn để tiếp tục loại bỏ các bộ được xác định trong quan hệ SUPPLIER giống như đã thực hiện không điều kiện trong ứng dụng ban đầu.

Ví dụ này được áp dụng cho cả CSDL tập trung và CSDL phân bố mà trong đó hệ thống cung cấp tính trong suốt phân mảnh. Trong trường hợp này, các ràng buộc toàn vẹn được định nghĩa trong các quan hệ toàn cục; sự kết hợp của các chương trình ứng dụng với mã lệnh để kiểm tra ràng buộc được thực hiện ở mức toàn cục. Nếu các ứng dụng toàn cục được viết ở các mức trong suốt thấp hơn thì sự kết hợp này sẽ khó hơn và phụ thuộc vào phân mảnh dữ liệu và định vị dữ liệu.

Một trong các nhược điểm nghiêm trọng của các ràng buộc toàn vẹn là hiệu quả bị giảm do việc thực hiện các kiểm tra ràng buộc; hiệu quả bị giảm là một điều rất quan trọng trong CSDL phân bố. Ví dụ, giả sử SUPPLY và SUPPLIER được định vị như trong phần 3.3 mục 2 (ứng dụng SUPOFPART) thì truy vấn SQL được thêm vào để thực hiện việc kiểm tra, chạy trên SUPPLY, phải được truyền cho các nơi 3 và 4, trong khi đó ứng dụng ban đầu chỉ cần truy xuất quan hệ SUPPLIER tại các nơi 1 và 2.

Việc quản lý các ràng buộc toàn vẹn trong các CSDL phân bố chưa được nghiên cứu hoàn toàn; các kỹ thuật cần thiết cho một môi trường phân bố không khác nhiều so với các kỹ thuật trong một hệ thống tập trung, đặc biệt khi CSDL phân bố cung cấp tính trong suốt phân mảnh (và rất vô lý để cố gắng phát triển các ràng buộc toàn vẹn ở mức trong suốt thấp hơn). Vấn đề chính trong việc áp dụng kiểm tra ràng buộc cho các CSDL phân bố là toàn bộ hiệu quả của hệ thống. Bởi vì việc kiểm tra ràng buộc có thể làm tăng yêu cầu truy xuất các nơi từ xa (như trong ví dụ trên) nên cũng cần xét việc kiểm tra ràng buộc trong việc thiết kế phân bố của CSDL.

THIẾT KẾ CƠ SỞ DỮ LIỆU PHÂN BỐ

Thiết kế CSDL phân bố là một việc rất khó, bởi vì nhiều vấn đề về tổ chức và về kỹ thuật, mà chúng đóng vai trò chủ yếu trong thiết kế CSDL đơn nơi (*single-site database*), trở nên khó hơn trong một hệ thống đa nơi (*multiple-site system*). Về quan điểm kỹ thuật, các vấn đề mới phát sinh, chẳng hạn như việc kết nối giữa các nơi nhờ vào một mạng máy tính và phân bố tối ưu về dữ liệu và các ứng dụng cho các nơi để thỏa mãn các yêu cầu của ứng dụng và để tối ưu hóa hiệu quả. Về quan điểm tổ chức, vấn đề phi tập trung hóa là chủ yếu, bởi vì các hệ thống phân bố thay thế cho các hệ thống tập trung, lớn, và trong trường hợp này, việc phân bố một ứng dụng có ảnh hưởng lớn đến tổ chức.

Mặc dù các kinh nghiệm thiết kế hệ thống phân bố còn bị hạn chế, vấn đề này đã được nghiên cứu một cách bao quát, phần lớn xuất phát từ quan điểm kỹ thuật, và nhiều đóng góp có thể được tìm thấy trong các tài liệu. Vấn đề về mặt toán học của việc phân bố dữ liệu một cách tối ưu trên một mạng máy tính đã được phân tích nhiều trong ngữ cảnh của các hệ thống tập tin phân bố và trong các CSDL phân bố. Các kết quả chủ yếu của nghiên cứu này bao gồm hai phần:

- 1- Nhiều tiêu chuẩn thiết kế đã được thiết lập về vấn đề dữ liệu có thể được phân bố một cách thuận lợi như thế nào.
- 2- Cơ sở toán học dùng để trợ giúp thiết kế sẽ giúp cho người thiết kế trong việc xác định phân bố dữ liệu.

Chương này gồm ba phần:

Phần 4.1, giới thiệu các bước để thiết kế các CSDL phân bố bằng cách nhấn mạnh cái gì nên được thiết kế. Chúng ta cũng nêu ra các mục tiêu của thiết kế phân bố dữ liệu và giới thiệu cách tiếp cận từ trên xuống (*top-down approach*) và cách tiếp cận từ dưới lên (*bottom-up approach*). Trong phần còn lại của chương này, chúng ta sẽ tập trung vào cách tiếp cận từ trên xuống.

Phần 4.2, chúng ta giải quyết thiết kế các mảnh ngang và các mảnh dọc, nêu ra các tiêu chuẩn có tính kinh nghiệm cho phép xác định chúng. Trong ngữ cảnh này, chúng ta giới thiệu khái niệm phép kết phân bố giữa các quan hệ được phân mảnh, và cho thấy việc chọn các tiêu chuẩn phân mảnh có thể rất có ích cho việc đơn giản các phép kết phân bố.

Phần 4.3, chúng ta giải quyết việc định vị các mảnh theo mô hình tham khảo của chúng ta, vấn đề này tương ứng với việc xác định ánh xạ giữa các mảnh và các hình ảnh vật lý. Chúng ta giới thiệu các nguyên lý và các khái niệm trong định vị mảnh, sau đó là các công thức đơn giản để đánh giá các chi phí và lợi ích của các định vị.

4.1 CÁC BƯỚC THIẾT KẾ CƠ SỞ DỮ LIỆU PHÂN BỐ

Thuật ngữ *thiết kế cơ sở dữ liệu phân bố* có một ý nghĩa rất rộng và không chính xác. Trong chương này, chúng ta tập trung vào các khía cạnh đó mà chúng chỉ có riêng cho các CSDL phân bố, trong khi đó chúng ta không giới thiệu các vấn đề thiết kế mà chúng cũng có trong thiết kế CSDL tập trung. Thiết kế một CSDL tập trung bao gồm:

- *Thiết kế lược đồ ý niệm* để mô tả CSDL được tích hợp (nghĩa là tất cả dữ liệu được sử dụng bởi các ứng dụng CSDL).

- *Thiết kế cơ sở dữ liệu vật lý*, nghĩa là ánh xạ lược đồ ý niệm vào các vùng lưu trữ và xác định các phương pháp truy xuất thích hợp.

Trong một CSDL phân bố, hai vấn đề này trở thành thiết kế lược đồ toàn cục và thiết kế các CSDL vật lý cục bộ tại mỗi nơi, các kỹ thuật có thể được áp dụng cho các vấn đề này thì giống như trong các CSDL tập trung. Cùng với các vấn đề trên, CSDL phân bố còn có thêm hai vấn đề mới.

- *Thiết kế phân mảnh*, nghĩa là xác định các quan hệ toàn cục được phân chia thành các mảnh ngang, dọc hoặc hỗn hợp như thế nào.

- *Thiết kế định vị mảnh*, nghĩa là xác định các mảnh được ánh xạ vào các hình ảnh vật lý như thế nào và xác định việc nhân bản các mảnh.

Hai vấn đề này hoàn toàn đặc trưng cho thiết kế phân bố dữ liệu. Thiết kế phân mảnh đã được nghiên cứu, bởi vì sự phân mảnh được xem là một đặc tính riêng biệt của các CSDL phân bố. Tuy

nhien, nó đã được phân tích từng phần trong ngữ cảnh của các hệ thống tập trung với nhiều thiết bị lưu trữ. Vấn đề định vị đã được nghiên cứu một cách bao quát do sự phát triển của các hệ thống tập tin phân bố.

Sự khác biệt giữa hai vấn đề này là hợp lý về mặt ý niệm, bởi vì vấn đề đầu tiên giải quyết "các tiêu chuẩn luận lý" là cơ sở của việc phân mảnh một quan hệ toàn cục, trong khi đó, vấn đề thứ hai giải quyết "sự sắp đặt vật lý" của dữ liệu đặt tại nhiều nơi khác nhau. Tuy nhiên, sự khác biệt này phải được nêu ra một cách thận trọng. Nói chung, không thể xác định sự phân mảnh và sự định vị tối ưu bằng cách giải quyết hai vấn đề này một cách độc lập, bởi vì chúng có liên quan với nhau.

Mặc dù thiết kế các chương trình ứng dụng được thực hiện sau việc thiết kế các lược đồ, nhưng các hiểu biết về các yêu cầu của ứng dụng ảnh hưởng đến thiết kế các lược đồ, bởi vì các lược đồ phải có khả năng hỗ trợ ứng dụng một cách hiệu quả. Do đó, trong thiết kế CSDL phân bố, cần phải hiểu biết thật chính xác về các yêu cầu của ứng dụng; rõ ràng, sự hiểu biết này chỉ cần thiết đối với các ứng dụng "quan trọng" hơn, nghĩa là các ứng dụng được thực hiện thường xuyên hoặc các ứng dụng cần phải được chạy một cách hiệu quả. Trong các yêu cầu của ứng dụng, chúng ta quan tâm đến:

- Nơi chạy ứng dụng (còn được gọi là *nơi gốc* của ứng dụng).
- Tần suất chạy ứng dụng (nghĩa là số lần chạy trong một đơn vị thời gian); trong trường hợp tổng quát mà các ứng dụng có thể được chạy tại nhiều nơi, chúng ta cần phải biết tần suất chạy của mỗi ứng dụng tại mỗi nơi.
- Số lượng, loại và sự phân bố của các truy xuất trong mỗi ứng dụng đến mỗi đối tượng dữ liệu cần thiết.

Xác định rõ các đặc điểm này là điều quan trọng. Hơn nữa, cần phải xem xét các dữ liệu này được cho trước đối với các quan hệ toàn cục và phải được biến thành các điều kiện áp dụng cho các mảnh, bởi vì kết quả của thiết kế là tạo ra sự phân mảnh, các dữ liệu này phải được biết đến đối với tất cả các cách phân mảnh khác nhau mà chúng được xét trong khi thiết kế.

4.1.1 Các mục tiêu của thiết kế phân bố dữ liệu

Trong thiết kế phân bố dữ liệu, chúng ta nên quan tâm đến các mục tiêu sau đây:

Tính cục bộ xử lý

Việc phân bố dữ liệu để làm cực đại hóa tính cục bộ xử lý (*processing locality*) tương ứng với nguyên tắc cơ bản là đặt dữ liệu càng gần các ứng dụng sử dụng các dữ liệu này càng tốt. Trong chương 1, chúng ta đã chỉ ra rằng đạt được tính cục bộ xử lý là một trong các mục tiêu chủ yếu của CSDL phân bố.

Đối với phân mảnh, vấn đề quan trọng là *đơn vị phân bố* (*unit of distribution*) thích hợp. Một quan hệ không là một đơn vị phân bố thích hợp vì nhiều lý do:

- Thứ nhất, các khung nhìn ứng dụng thông thường là các tập con của các quan hệ. Do đó, tính cục bộ xử lý của các ứng dụng không được xác định trên các quan hệ mà là trên các tập con của các quan hệ này. Vì thế, chỉ có thể xem các tập con của các quan hệ là các đơn vị phân bố.

- Thứ hai, nếu các ứng dụng có các khung nhìn được định nghĩa trên một quan hệ cho trước được đặt tại các nơi khác nhau, thì có thể có hai cách khác nhau để xem toàn bộ quan hệ là một đơn vị phân bố. Quan hệ không được nhân bản và được lưu trữ chỉ tại một nơi, hoặc quan hệ được nhân bản tại tất cả hoặc một số nơi có chạy các ứng dụng. Cách thứ nhất dẫn đến một số lượng lớn không cần thiết các truy xuất dữ liệu từ xa. Mặt khác, cách thứ hai có sự nhân bản không cần thiết và gây ra các vấn đề không mong muốn trong việc thực hiện cập nhật nếu vùng lưu trữ bị giới hạn.

Cách đơn giản nhất để xác định tính cục bộ xử lý là xét hai loại tham chiếu dữ liệu: các *tham chiếu cục bộ* (*local reference*) và các *tham chiếu từ xa* (*remote reference*). Rõ ràng, khi biết các nơi gốc của các ứng dụng, tính cục bộ (*locality*) và tính từ xa (*remoteness*) của các tham chiếu chỉ phụ thuộc vào sự phân bố dữ liệu.

Thiết kế phân bố dữ liệu để cực đại hóa tính cục bộ xử lý (tức là làm cực tiểu hóa các tham chiếu từ xa) có thể được thực hiện bằng cách xét các tham chiếu cục bộ và các tham chiếu từ xa tương ứng với mỗi cách phân mảnh dự kiến (*candidate fragmentation*) và chọn ra giải pháp tốt nhất giữa các cách phân mảnh này.

Mở rộng tiêu chuẩn tối ưu hóa đơn giản này cần được quan tâm đến khi một ứng dụng có *tính cục bộ hoàn toàn* (*complete locality*). Chúng ta sử dụng thuật ngữ này để nói đến các ứng dụng mà chúng có thể được thực hiện hoàn toàn tại nơi gốc của chúng. Ưu điểm của tính cục bộ hoàn toàn không chỉ là giảm bớt các truy xuất từ xa mà còn làm tăng tính đơn giản trong việc kiểm soát việc thực hiện ứng dụng.

Tính sẵn sàng và độ tin cậy của dữ liệu phân bố

Trong chương 1, chúng ta đã thấy tính sẵn sàng và độ tin cậy được xem là ưu điểm của các hệ thống phân bố đối với các hệ thống không phân bố. Mức độ sẵn sàng cao đối với các ứng dụng chỉ đọc sẽ đạt được bằng cách lưu trữ nhiều bản nhân của cùng một thông tin, hệ thống phải có khả năng chuyển đổi qua một bản khác khi một bản được truy xuất dưới các điều kiện bình thường trở thành không hiệu lực.

Độ tin cậy cũng đạt được bằng cách lưu trữ nhiều bản nhân của cùng một thông tin, bởi vì có thể phục hồi khi máy tính bị ngừng hoặc có hư hỏng vật lý của một trong các bản nhân bằng cách sử dụng các bản nhân khác vẫn còn hiệu lực. Vì hư hỏng vật lý có thể xảy ra bởi các biến cố (chẳng hạn như lửa, động đất, phá hoại, ...), do đó việc lưu trữ các bản nhân tại các vị trí địa lý phân tán là thích hợp.

Điều phối tải làm việc

Điều phối tải làm việc tại các nơi là một đặc điểm quan trọng của các hệ thống máy tính phân bố. Thực hiện điều phối tải làm việc để tận dụng ưu điểm của các nguồn lực khác nhau hoặc tính năng của các máy tính tại mỗi nơi, và cực đại hóa mức độ thực hiện song song các ứng dụng. Vì điều phối tải làm việc có thể ảnh hưởng ngược lại với tính cục bộ xử lý, do đó cần phải cân nhắc giữa chúng trong thiết kế phân bố dữ liệu. Phân rã một quan hệ thành các mảnh, mỗi mảnh được xử lý như là một đơn vị, cho phép nhiều giao dịch được thực hiện đồng thời. Hơn nữa, phân mảnh các quan hệ dẫn đến việc thực hiện đồng thời một truy vấn đơn bằng cách chia truy vấn này thành các truy vấn con để thực hiện trên các mảnh, đặc tính này được gọi là *tính đồng thời nội truy vấn* (*intraquery concurrency*). Do đó, sự phân mảnh làm tăng mức độ đồng thời và từ đó làm tăng thông lượng của hệ thống (*system throughput*).

Các chi phí lưu trữ và khả năng lưu trữ có sẵn

Sự phân bố CSDL nên phản ánh chi phí và khả năng lưu trữ tại các nơi khác nhau. Có thể có các nơi chuyên dụng trong mạng để lưu trữ dữ liệu, hoặc ngược lại có các nơi không hỗ trợ vùng lưu trữ lớn. Chi phí lưu trữ dữ liệu là không thích đáng so với các chi phí CPU, nhập/xuất và truyền thông của các ứng dụng, nhưng phải xét giới hạn lưu trữ có sẵn tại mỗi nơi.

Sử dụng cùng lúc tất cả các tiêu chuẩn ở trên là điều vô cùng khó khăn, bởi vì điều này dẫn đến các mô hình tối ưu hóa phức tạp. Có thể xem một số đặc điểm ở trên như là các ràng buộc, hơn là các mục tiêu (chẳng hạn, có thể nêu ra các ràng buộc về số lượng tải làm việc tối đa hoặc vùng lưu trữ tối đa có sẵn tại mỗi nơi). Mặt khác, có thể xét tiêu chuẩn quan trọng nhất trong thiết kế ban đầu và đưa ra các tiêu chuẩn khác trong hậu tối ưu hóa.

Lưu ý rằng, sự phân mảnh vẫn có các nhược điểm. Nếu các ứng dụng có các yêu cầu mâu thuẫn nhau không cho phép phân rã các quan hệ thành các mảnh riêng biệt nhau, các ứng dụng này có các khung nhìn được định nghĩa trên nhiều mảnh, có thể làm giảm hiệu quả. Chẳng hạn, cần phải lấy dữ liệu từ hai mảnh và sau đó thực hiện phép hợp hoặc phép kết của chúng mà điều này làm tốn chi phí. Tránh điều này là một vấn đề cơ bản của phân mảnh. Vấn đề thứ hai liên quan đến việc kiểm tra dữ liệu ngữ nghĩa (*semantic data control*), đặc biệt là việc kiểm tra tính toàn vẹn (*integrity checking*). Do kết quả của sự phân mảnh, các thuộc tính tham gia vào một phụ thuộc có thể bị phân rã trong các mảnh khác nhau và có thể được định vị tại các nơi khác nhau. Trong trường hợp này, ngay cả công việc đơn giản như kiểm tra các phụ thuộc sẽ dẫn đến việc kết nối dữ liệu tại nhiều nơi.

Sau đây chúng ta sẽ sử dụng cách tiếp cận đơn giản để cực đại hóa tính cục bộ xử lý, mục tiêu này là thích đáng để đưa ra các tiêu chuẩn thiết kế thường dùng và thực tế.

4.1.2 Cách tiếp cận từ trên xuống và cách tiếp cận từ dưới lên để thiết kế phân bố dữ liệu

Có hai cách tiếp cận khác nhau để thiết kế phân bố dữ liệu: cách tiếp cận từ trên xuống và cách tiếp cận từ dưới lên.

1- Cách tiếp cận từ trên xuống

Trong cách tiếp cận từ trên xuống (*top-down approach*), chúng ta bắt đầu bằng cách thiết kế lược đồ toàn cục, thiết kế phân mảnh của CSDL, định vị các mảnh tại các nơi, tạo ra các hình ảnh vật lý. Kết thúc cách tiếp cận này là việc thực hiện thiết kế dữ liệu vật lý đặt tại mỗi nơi.

Quá trình thiết kế từ trên xuống được chỉ ra trong hình 4.1. Trước tiên, chúng ta phân tích các yêu cầu để xác định môi trường của hệ thống và từ đó xác định các nhu cầu về dữ liệu và xử lý của người sử dụng CSDL. Nghiên cứu các yêu cầu cũng để xác định hệ thống cuối cùng sẽ thỏa mãn các mục tiêu nào của một DBMS phân bố. Các mục tiêu này được xác định từ hiệu suất, độ tin cậy và tính sẵn sàng, tính kinh tế và tính mở rộng (tính linh hoạt).

Tài liệu các yêu cầu là phần nhập cho hai hoạt động song song: *thiết kế khung nhìn* (*view design*) và *thiết kế ý niệm* (*conceptual design*). Thiết kế khung nhìn định nghĩa các giao diện cho những người sử dụng cuối cùng. Mặt khác, thiết kế ý niệm là quá trình khảo sát hoạt động kinh doanh (*enterprise*) để xác định các kiểu thực thể (*entity type*) và các mối liên kết (*relationship*) giữa các kiểu thực thể này. Người ta có thể chia quá trình này thành hai nhóm hoạt động liên quan với nhau: phân tích thực thể (*entity analysis*) và phân tích chức năng (*functional analysis*). **Phân tích thực thể** liên quan đến việc xác định các thực thể, các thuộc tính của chúng và các mối liên kết giữa chúng. Mặt khác, **phân tích chức năng** liên quan đến việc xác định các chức năng cơ bản của hoạt động kinh doanh được mô hình hóa. Các kết quả của hai bước này cần được bổ sung những lời chỉ dẫn tham khảo để hiểu biết tốt hơn về các chức năng nào được giải quyết với các thực thể nào.

Có một mối quan hệ giữa thiết kế ý niệm và thiết kế khung nhìn. Theo một nghĩa nào đó, thiết kế ý niệm có thể được diễn giải như là sự tích hợp các khung nhìn của người sử dụng. Mặc dù hoạt động tích hợp khung nhìn này (*view integration*) là rất quan trọng, mô hình ý niệm (*conceptual model*) nên hỗ trợ không chỉ cho các ứng dụng hiện tại mà còn cho các ứng dụng trong tương lai. Tích hợp khung nhìn nên được sử dụng để bảo đảm rằng các yêu cầu thực thể và mối liên kết cho tất cả các khung nhìn đã được đưa vào trong lược đồ ý niệm.

Trong các hoạt động thiết kế ý niệm và thiết kế khung nhìn, người sử dụng cần phải chỉ rõ các thực thể dữ liệu và phải xác định các ứng dụng sẽ chạy trên CSDL cũng như thông tin thống kê về các ứng dụng này. Thông tin thống kê bao gồm việc xác định tần suất của các ứng dụng, khối lượng của các thông tin khác nhau và các số liệu khác tương tự. Lưu ý rằng, từ bước thiết kế ý niệm sẽ cho ra định nghĩa *lược đồ ý niệm toàn cục* (GCS – *Global Conceptual Schema*). Chúng ta chưa xét đến sự liên quan của môi trường phân bố; thực tế là cho đến lúc này, quá trình này giống hệt với quá trình trong thiết kế CSDL tập trung.

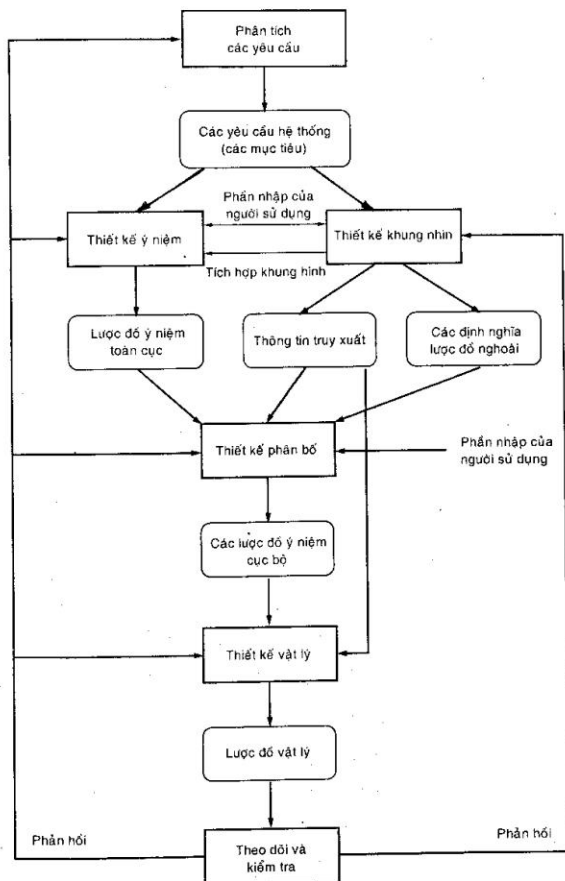
Lược đồ ý niệm toàn cục và thông tin kiểu truy xuất (*access pattern information*) là kết quả của thiết kế khung nhìn và là phần nhập cho bước thiết kế phân bố (*distribution design*). Mục tiêu của giai đoạn này cũng là trọng tâm của chương này, đó là thiết kế các *lược đồ ý niệm cục bộ* (LCS – *Local Conceptual Schema*) bằng cách phân bố các thực thể tại các nơi của hệ thống phân bố. Tất nhiên, có thể xem mỗi thực thể là một đơn vị phân bố. Trong mô hình quan hệ, các thực thể tương ứng với các quan hệ.

Thay vì phân bố các quan hệ, thông thường chúng ta phân chia chúng thành các quan hệ con, được gọi là các *mảnh* (*fragment*) và sau đó phân bố các mảnh này. Do đó, hoạt động thiết kế phân bố bao gồm hai bước: *phân mảnh* (*fragmentation*) và *định vị* (*allocation*). Đây là hai vấn đề chính được giải quyết trong chương này, do đó chúng ta sẽ bàn đến chúng trong các phần sau.

Bước cuối cùng trong quá trình thiết kế là *thiết kế vật lý* (*physical design*) để ánh xạ các lược đồ ý niệm cục bộ vào các thiết bị lưu trữ vật lý (*physical storage device*) có sẵn tại các nơi tương ứng. Phần nhập cho bước này là các lược đồ ý niệm cục bộ và thông tin kiểu truy xuất các mảnh trong các lược đồ này.

Hoạt động thiết kế và phát triển là một quá trình liên tục, nó đòi hỏi phải giám sát thường xuyên và điều chỉnh định kỳ. Do đó, chúng ta phải xem việc theo dõi và giám sát như là một hoạt động chính trong quá trình này. Lưu ý rằng, chúng ta không chỉ giám sát cách thực hiện CSDL mà còn phải giám sát tính thích hợp của các khung nhìn của người sử dụng. Kết quả là một số dạng phản hồi và có thể dẫn đến việc quay trở lại một trong các bước trước trong thiết kế.

Cách tiếp cận từ trên xuống là cách thường dùng nhất đối với các hệ thống được xây dựng từ đầu, bởi vì nó cho phép thực hiện thiết kế một cách hợp lý. Do đó, cách tiếp cận này sẽ được đề cập đến trong phần còn lại của chương này. Tuy nhiên, chúng ta sẽ không bàn đến các kỹ thuật có thể được sử dụng để thiết kế lược đồ toàn cục và lược đồ vật lý, bởi vì chúng không phải được dùng riêng cho CSDL phân bố. Chúng ta sẽ giải quyết thiết kế phân mảnh và thiết kế định vị mảnh trong các phần 4.2 và 4.3.



Hình 4.1 Quá trình thiết kế từ trên xuống

2- Cách tiếp cận từ dưới lên

Khi CSDL phân bố được phát triển từ sự kết hợp của các CSDL hiện tại, thì không cần thiết phải sử dụng cách tiếp cận từ trên xuống. Trong thực tế, trong trường hợp này, lược đồ toàn cục thường được tạo ra từ sự kết hợp giữa các mô tả dữ liệu hiện tại. Cũng có thể sử dụng một lược đồ chuyển đổi khác nhau cho từng cặp CSDL hiện tại và khi đó không có khái niệm lược đồ toàn cục. Tuy nhiên, điều này sẽ làm cho các hệ thống khác với kiến trúc tham khảo của chúng ta.

Khi các CSDL hiện tại được kết hợp lại, chúng ta có thể sử dụng cách tiếp cận từ dưới lên (*bottom-up approach*) để thiết kế phân bố dữ liệu. Cách tiếp cận này dựa trên sự tích hợp của các lược đồ hiện tại thành một lược đồ toàn cục. Bằng cách tích hợp, chúng ta muốn hợp nhất các định nghĩa dữ liệu chung và giải quyết các mâu thuẫn giữa các biểu diễn khác nhau cho cùng một dữ liệu.

Lưu ý rằng, bản thân cách tiếp cận từ dưới lên không dễ dàng được áp dụng cho các quan hệ được phân mảnh ngang. Trong thực tế, các mảnh ngang của cùng một quan hệ toàn cục phải có cùng một lược đồ quan hệ, đặc điểm này dễ dàng được áp dụng trong thiết kế từ trên xuống, trong khi đó khó phát hiện ra đặc điểm này trong các CSDL được thiết kế một cách độc lập rồi sau đó tích hợp lại. Vì các mảnh ngang có một đặc điểm thích hợp và có ích cho một CSDL phân bố, quá trình tích hợp nên biến đổi các định nghĩa của các quan hệ cục bộ sao cho chúng có thể được xem là các mảnh ngang của một quan hệ toàn cục chung.

Khi các CSDL hiện tại được kết hợp thành một CSDL phân bố, có thể chúng sử dụng các hệ DBMS khác nhau. Như chúng ta đã bàn ngắn gọn trong chương 1 và chương 3, một hệ thống không đồng nhất làm tăng độ phức tạp của việc tích hợp dữ liệu và cần phải có *sự chuyển đổi* giữa các biểu diễn khác nhau. Trong trường hợp này, có thể có sự chuyển đổi một - một giữa từng cặp DBMS khác nhau. Tuy nhiên, cách tiếp cận được sử dụng chủ yếu trong các nguyên mẫu của các hệ thống không đồng nhất là chọn ra một mô hình dữ liệu chung, và sau đó chuyển đổi tất cả các lược đồ khác nhau của các DBMS phức tạp thành một dạng biểu diễn duy nhất này.

Tóm lại, thiết kế từ dưới lên của một CSDL phân bố đòi hỏi:

– **Chọn một mô hình cơ sở dữ liệu chung** để mô tả lược đồ toàn cục của CSDL.

– **Chuyển đổi** mỗi lược đồ cục bộ thành mô hình dữ liệu chung.

– **Tích hợp** các lược đồ cục bộ thành một lược đồ toàn cục chung.

Do đó, cách tiếp cận từ dưới lên đòi hỏi phải giải quyết ba vấn đề mà không phải của riêng các CSDL phân bố, mà chúng cũng có trong các hệ thống tập trung. Vì thế, thiết kế từ dưới lên không được giải quyết trong chương này.

4.1.3 Các yêu cầu thông tin

Một khía cạnh của thiết kế phân bố là có quá nhiều yếu tố trong thiết kế tối ưu. Tổ chức luận lý của CSDL, vị trí của các ứng dụng, các đặc điểm truy xuất CSDL của các ứng dụng và các đặc tính của các hệ thống máy tính tại mỗi nơi, tất cả đều ảnh hưởng đến các quyết định phân bố. Điều này làm cho việc phát biểu vấn đề phân bố rất phức tạp.

Thông tin dùng để thiết kế phân bố có thể được chia thành bốn loại:

- Thông tin về CSDL
- Thông tin về ứng dụng
- Thông tin về mạng truyền thông
- Thông tin về hệ thống máy tính.

Hai loại sau hoàn toàn có tính chất định lượng và được sử dụng trong các mô hình định vị hơn là trong các giải thuật phân mảnh. Ở đây, chúng ta không xét chúng một cách chi tiết. Thay vào đó, các yêu cầu thông tin chi tiết của các giải thuật phân mảnh và định vị sẽ được bàn đến trong các phần tương ứng của chúng.

4.2 THIẾT KẾ PHÂN MẢNH CƠ SỞ DỮ LIỆU

Thiết kế phân mảnh là vấn đề đầu tiên phải được giải quyết trong thiết kế phân bố dữ liệu theo cách tiếp cận từ trên xuống. Mục đích của thiết kế phân mảnh là xác định các mảnh không giao nhau mà chúng là các **đơn vị phân bố**, còn được gọi là **đơn vị định vị luận lý** (*logical unit of allocation*), thích hợp cho vấn đề định vị dữ liệu sau này.

Rõ ràng, các bộ hoặc các thuộc tính của các quan hệ không thể được xem là các đơn vị phân bố riêng biệt, bởi vì các vấn đề ứng dụng sẽ trở nên khó quản lý. Thiết kế các mảnh bao gồm việc gom nhóm các bộ (trong trường hợp phân mảnh ngang) hoặc các thuộc tính (trong trường hợp phân mảnh dọc) mà chúng có "cùng các đặc tính" theo quan điểm định vị của chúng. Mỗi nhóm gồm các bộ hoặc các thuộc tính có cùng các đặc tính tạo thành một mảnh. Ý tưởng cơ bản là nếu bất kỳ hai phần tử nào thuộc cùng một mảnh có cùng các đặc tính theo quan điểm định vị của chúng, thì bất kỳ phương pháp nào được dùng để định vị dữ liệu sẽ đặt chúng chung với nhau. Do đó, các mảnh có được theo cách này là các đơn vị phân bố thích hợp. Bây giờ chúng ta làm rõ nghĩa của "cùng các đặc tính" của các phần tử của mảnh.

Ví dụ 4.1 Xét việc xác định các mảnh ngang của quan hệ toàn cục EMP. EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM, JOB)

Giả sử các ứng dụng quan trọng của CSDL phân bố này yêu cầu thông tin về các thành viên của các dự án và được lấy từ quan hệ EMP. Cho mỗi phòng ban là một nơi của CSDL phân bố.

Ứng dụng có thể được chạy tại bất kỳ phòng ban nào. Tuy nhiên, khi chúng được chạy tại một phòng ban cho trước, chúng sẽ tham chiếu đến các bộ của các nhân viên của phòng ban đó với xác suất cao hơn so với các bộ của các nhân viên khác. Điều này xảy ra bởi vì các nhân viên hiện tại được phân bố trong các phòng ban, và bởi vì thông tin về các dự án được yêu cầu bởi các nhân viên làm trong dự án. Do đó, trong trường hợp đơn giản, các mảnh được thiết kế bằng cách tập hợp trong mỗi mảnh gồm các bộ của các nhân viên có cùng đặc tính theo quan điểm định vị của chúng.

Chúng ta cũng nêu ra một ví dụ đơn giản về phân mảnh dọc quan hệ EMP. Giả sử các thuộc tính SAL và TAX chỉ được sử dụng cho các ứng dụng quản trị mà các ứng dụng này luôn luôn sử dụng các thuộc tính này chung với nhau. Do đó, cặp thuộc tính SAL, TAX là một mảnh dọc thích hợp.

Lưu ý rằng, nơi gốc của mỗi ứng dụng là thích hợp cho việc xác định các đặc tính cục bộ để định nghĩa các mảnh tương ứng. Do đó, chúng ta xem các ứng dụng được chạy tại các nơi khác nhau như là các ứng dụng khác nhau, ngay cả khi chúng thực hiện cùng một chức năng.

Sau đây, chúng ta xét phân mảnh ngang, phân mảnh dọc và phân mảnh hỗn hợp một cách độc lập nhau. Như chúng ta đã khảo sát, điều phải làm trước tiên về mặt ý niệm là xác định phân mảnh và sau đó định vị các mảnh.

4.2.1 Phân mảnh ngang

Trong chương 3, chúng ta đã giới thiệu hai loại phân mảnh ngang là phân mảnh ngang chính và phân mảnh ngang dẫn xuất.

Việc xác định phân mảnh ngang của một CSDL có nghĩa là xác định các đặc tính luận lý của dữ liệu, chẳng hạn như các vị từ phân mảnh, và các đặc tính thống kê của dữ liệu, chẳng hạn như số tham chiếu của các ứng dụng đến các mảnh. Sự kết hợp của các mặt luận lý và thống kê sẽ khó hơn.

Phân mảnh ngang chia một quan hệ theo các bộ của nó. Do đó, mỗi mảnh là một tập hợp con gồm các bộ của quan hệ. Phân mảnh ngang chính của một quan hệ được thực hiện bằng cách sử dụng các vị từ được định nghĩa trên quan hệ đó. Phân mảnh ngang dẫn xuất là phân chia một quan hệ dựa vào các vị từ được định nghĩa trên một quan hệ khác.

Trước tiên, chúng ta phải tìm hiểu thông tin cần thiết để thực hiện phân mảnh ngang.

1- Các yêu cầu thông tin của phân mảnh ngang

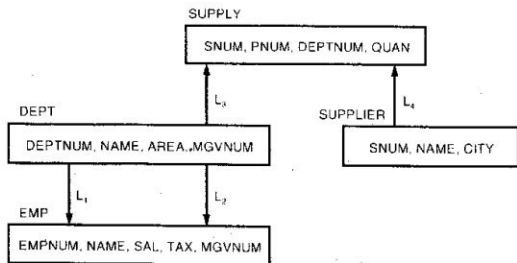
a- Thông tin về cơ sở dữ liệu

Thông tin về CSDL liên quan đến lược đồ ý niệm toàn cục. Trong ngữ cảnh này, lưu ý quan trọng là các quan hệ của CSDL được kết nối với nhau như thế nào, đặc biệt là các phép kết. Trong mô hình quan hệ, các mối liên kết này cũng được biểu diễn bằng các quan hệ. Tuy nhiên, trong các mô hình dữ liệu khác, chẳng hạn như mô hình thực thể kết hợp (*entity-relationship model E-R*), các mối liên kết này giữa các đối tượng CSDL được biểu diễn tường minh. Mỗi liên kết cũng được mô hình hóa tường minh trong cơ cấu tổ chức quan hệ (*relational framework*) cho các mục đích thiết kế phân bố. Về mặt ký hiệu, các đường liên kết có hướng (*directed link*) được vẽ giữa các quan hệ có liên quan với nhau và biểu diễn cho một phép kết bằng.

Ví dụ 4.2 Hình 4.2 cho thấy cách biểu diễn các đường liên kết giữa các quan hệ của CSDL trong hình 3.4. Lưu ý rằng, chiều của đường liên kết cho thấy mỗi liên kết một - nhiều (*one-to-many relationship*). Ví dụ, có nhiều nhân viên làm tại mỗi phòng ban. Do đó có một đường liên kết giữa các quan hệ DEPT và EMP. Theo cách này, mỗi liên kết nhiều - nhiều (*many-to-many relationship*) giữa các quan hệ SUPPLIER và DEPT được biểu diễn bởi hai đường liên kết đến SUPPLY.

Các đường liên kết giữa các đối tượng CSDL (tức là các quan hệ trong trường hợp này) hoàn toàn quen thuộc với những ai đã làm trên các mô hình dữ liệu mạng (*network data model*), nhưng chiều của đường liên kết được quy định ngược lại. Trong các mô hình quan hệ, chúng được giới thiệu dưới dạng các đồ thị kết. Ở đây, các đường liên kết sẽ giúp cho việc biểu diễn các mô hình phân bố được đơn giản.

Quan hệ ở gốc của đường liên kết được gọi là *quan hệ chủ* (*owner*) của đường liên kết và quan hệ ở ngọn được gọi là *quan hệ bộ phận* (*member*). Trong cơ cấu tổ chức quan hệ, các thuật ngữ thường được sử dụng là *quan hệ nguồn* (*source relation*) dùng cho quan hệ chủ và *quan hệ đích* (*target relation*) dùng cho quan hệ bộ phận. Chúng ta định nghĩa hai hàm *owner* và *member*, cả hai hàm này quy định các ánh xạ từ tập hợp các đường liên kết vào tập hợp các quan hệ. Cho trước một đường liên kết, hàm *owner* trả về quan hệ chủ và hàm *member* trả về quan hệ bộ phận của đường liên kết.



Hình 4.2 Biểu diễn các mối liên kết giữa các quan hệ dùng các đường liên kết

Ví dụ 4.3 Xét đường liên kết L_1 của hình 4.2, các hàm *owner* và *member* cho các kết quả như sau:

$$\text{owner}(L_1) = \text{DEPT}$$

$$\text{member}(L_1) = \text{EMP}$$

Thông tin định lượng cần có về CSDL là số bộ của mỗi quan hệ R , ký hiệu là $\text{card}(R)$.

b- Thông tin về ứng dụng

Như đã nói đến trước kia về hình 4.1, cần phải có cả hai thông tin định lượng và định tính về các ứng dụng. Thông tin định tính giúp ích cho việc phân mảnh và thông tin định lượng liên quan chủ yếu đến các mô hình định vị.

Thông tin định tính cơ bản bao gồm các vị từ được sử dụng trong các truy vấn của người sử dụng. Nếu không thể phân tích tất cả các ứng dụng để xác định các vị từ này, chúng ta chỉ nghiên cứu các ứng dụng quan trọng. Một đề nghị là 20% các truy vấn thiết thực nhất cần đến 80% tổng số lần truy xuất dữ liệu [Wiederhold, 1982]. Quy tắc 80/20 này có thể được sử dụng để thực hiện phân tích này.

Bây giờ, giả sử chúng ta cần phân mảnh ngang một quan hệ $R(A_1, A_2, \dots, A_n)$, với A_i là một thuộc tính được định nghĩa trên miền D_i . Chúng ta có các định nghĩa sau đây:

1- Một vị từ đơn giản (*simple predicate*) p_j được định nghĩa trên R có dạng:

$$p_j: A_i \theta \text{value}$$

trong đó θ là một trong các phép so sánh $=, \neq, <, \leq, >, \geq$ và *value* được chọn từ miền của A_i ($\text{value} \in D_i$). Chúng ta ký hiệu P_j là tập hợp các vị từ đơn giản được định nghĩa trên quan hệ R . Các phần tử của P_j được ký hiệu là p_j .

2- Mặc dù vị từ đơn giản được giải quyết dễ dàng, nhưng các truy vấn của người sử dụng thường bao gồm nhiều vị từ phức tạp, chúng là các tổ hợp luận lý của các vị từ đơn giản. Một tổ hợp mà chúng ta đặc biệt quan tâm đến được gọi là vị từ giao tối thiểu. Bởi vì luôn luôn có thể biến đổi một biểu thức luận lý về dạng chuẩn giao (*conjunctive normal form*), do đó việc sử dụng các vị từ giao tối thiểu trong các giải thuật thiết kế sẽ không làm mất tính tổng quát.

Một vị từ giao tối thiểu (*minterm predicate*) m_j đối với một tập hợp các vị từ đơn giản $P_r = \{p_1, p_2, \dots, p_m\}$ là một tổ hợp giao của tất cả các vị từ xuất hiện trong P_r , các vị từ này ở dạng thông thường hoặc ở dạng phủ định, để sao cho m_j không bị mâu thuẫn. Do đó:

$$m_j = \bigwedge p^*_{i}, 1 \leq i \leq m$$

với $p^*_{i} = p_i$ hoặc $p^*_{i} = \neg p_i$ và $m_j \neq \text{false}$

Chúng ta gọi tập hợp các vị từ giao tối thiểu là:

$$M = \{m_1, m_2, \dots, m_r\}$$

Có một điều lưu ý quan trọng ở đây. Việc xét phủ định của một vị từ sẽ có ý nghĩa đối với các vị từ đẳng thức (*equality predicate*) có dạng:

$$\text{Attribute} = \text{Value}$$

Đối với các vị từ bất đẳng thức (*inequality predicate*), phủ định nên được xử lý như phần bù (*complement*). Ví dụ, phủ định của vị từ đơn giản $\text{Attribute} \leq \text{Value}$ là $\text{Attribute} > \text{Value}$.

Ngoài các vấn đề lý thuyết về phép lấy bù trong các tập hợp vô hạn, cũng có một vấn đề thực tế là có thể khó xác định được phần bù. Ví dụ, nếu hai vị từ đơn giản có dạng:

$$\text{Lower_bound} \leq \text{Attribute}$$

$$\text{Attribute} \leq \text{Upper_bound}$$

thì phần bù của chúng là:

$$\neg (\text{Lower_bound} \leq \text{Attribute})$$

và:

$$\neg (\text{Attribute} \leq \text{Upper_bound})$$

Tuy nhiên, hai vị từ trên có thể được viết như sau:

$$\text{Lower_bound} \leq \text{Attribute} \leq \text{Upper_bound}$$

với phần bù là:

$$\neg (\text{Lower_bound} \leq \text{Attribute} \leq \text{Upper_bound})$$

thì không thể dễ xác định. Do đó, nghiên cứu trong lãnh vực này chỉ xét các vị từ đẳng thức đơn giản.

Về mặt thông tin định lượng về các ứng dụng của người sử dụng, chúng ta cần có hai tập hợp dữ liệu.

1. *Độ chọn giao tối thiểu (minterm selectivity)* là số bộ của quan hệ sẽ được truy xuất bởi một truy vấn tương ứng với một vị từ giao tối thiểu cho trước. Chúng ta ký hiệu độ chọn của vị từ giao tối thiểu m_j là $\text{sel}(m_j)$.

2. Tần số truy xuất (*access frequency*) là tần số của các ứng dụng truy xuất dữ liệu. Nếu $Q = \{q_1, q_2, \dots, q_q\}$ là tập hợp các truy vấn, $acc(q_i)$ cho biết tần số truy xuất của truy vấn q_i trong một khoảng thời gian cho trước.

Lưu ý rằng, các tần số truy xuất giao tối thiểu (*minterm access frequency*) có thể được xác định từ các tần số truy vấn. Chúng ta ký hiệu tần số truy xuất của vị từ giao tối thiểu m_j là $acc(m_j)$.

2. Phân mảnh ngang chính

Các mảnh ngang chính được xác định bằng cách sử dụng các phép chọn trên các quan hệ toàn cục. Tính đúng đắn của phân mảnh ngang chính đòi hỏi mỗi bộ của quan hệ toàn cục được đưa vào trong một và chỉ một mảnh. Do đó, việc xác định phân mảnh ngang chính của một quan hệ toàn cục đòi hỏi xác định một tập hợp các vị từ chọn (*selection predicate*) đầy đủ và tách biệt. Đặc tính cần có cho mỗi mảnh là các bộ của một mảnh phải được tham chiếu giống nhau trong tất cả các ứng dụng.

Chúng ta thực hiện phân mảnh ngang chính cho các quan hệ chủ bằng cách sử dụng các phép chọn. Do đó, cho trước một quan hệ R thì các mảnh ngang R_i của nó là:

$$R_i = \sigma_{F_i}(R); \quad 1 \leq i \leq n$$

với F_i là điều kiện chọn hoặc công thức chọn (*selection formula*) của mảnh R_i . Lưu ý rằng nếu F_i ở dạng chuẩn giao thì nó là một vị từ giao tối thiểu m_i .

Ví dụ 4.4 Chúng ta phân mảnh ngang chính cho quan hệ EMP thành hai mảnh ngang EMP_1 và EMP_2 như sau:

$$EMP_1 = \sigma_{SAL \leq 35000}(EMP)$$

$$EMP_2 = \sigma_{SAL > 35000}(EMP)$$

Ví dụ 4.4 cho thấy một trong các vấn đề của phân mảnh ngang. Nếu các miền của các thuộc tính có trong các điều kiện chọn là liên tục và vô hạn, thì chúng ta khó định nghĩa một tập hợp các điều kiện $F = \{F_1, F_2, \dots, F_n\}$ dùng để phân mảnh quan hệ một cách đúng đắn. Một cách giải quyết là định nghĩa các vùng (*range*) như chúng ta đã làm trong ví dụ 4.4. Tuy nhiên luôn luôn có vấn đề xử lý hai điểm giới hạn. Ví dụ, nếu thêm một bộ có $SAL = 50000$ vào quan hệ EMP

thì chúng ta phải xét lại việc phân mảnh này để quyết định thêm bộ mới vào mảnh EMP_2 hoặc phân mảnh lại và định nghĩa thêm một mảnh mới EMP_3 :

$$EMP_2 = \sigma \text{ } 35000 < \text{SAL AND SAL} \leq 40000 \text{ (EMP)}$$

$$EMP_3 = \sigma \text{ SAL} > 40000 \text{ (EMP)}$$

Chúng ta giả sử tính không âm của các giá trị của SAL là một đặc điểm của quan hệ và là một ràng buộc toàn vẹn. Nếu không, chúng ta phải thêm một vị từ đơn giản $0 \leq \text{SAL}$ vào tập P_r .

Trong thực tế, vấn đề này có thể được giải quyết bằng cách giới hạn miền của các thuộc tính phù hợp với các yêu cầu của ứng dụng.

Bây giờ, chúng ta có thể định nghĩa một mảnh ngang một cách chính xác hơn. Một **mảnh ngang** (*horizontal fragment*) R_i của quan hệ R bao gồm tất cả các bộ của R thỏa mãn vị từ giao tối thiểu m_i . Do đó, cho trước một tập hợp các vị từ giao tối thiểu M thì số mảnh ngang bằng số vị từ giao tối thiểu. Tập hợp các mảnh ngang này cũng thường được gọi là tập hợp các **mảnh giao tối thiểu** (*minterm fragment*).

Chúng ta vẫn chưa nêu ra được một giải thuật có thể phân mảnh một quan hệ R và một tập hợp các ứng dụng sẽ tham chiếu đến nó. Việc chọn lựa các vị từ không thể dựa vào các quy tắc chính xác, bởi vì việc nhận biết một vị từ có ích cho việc mô tả phân mảnh sẽ dựa vào trực giác của người thiết kế. Nhưng chúng ta biết rằng, định nghĩa các mảnh ngang dựa vào các vị từ giao tối thiểu. Do đó, bước đầu tiên của bất kỳ giải thuật phân mảnh nào là xác định một tập hợp các vị từ đơn giản có các đặc tính nào đó. Chúng ta có thể định nghĩa hai đặc tính đặc trưng cho sự phân mảnh thích hợp, đó là **tính đầy đủ** (*completeness*) và **tính tối thiểu** (*minimality*) của các vị từ đơn giản. Chúng ta có các định nghĩa sau:

1- Một vị từ đơn giản p_i được gọi là **thích hợp** (*relevant*) đối với một tập P_r các vị từ đơn giản, nếu tồn tại ít nhất hai vị từ giao tối thiểu m_i và m_j của P_r mà các biểu thức của chúng chỉ khác nhau ở p_i (tức là m_i chứa p_i và m_j chứa $\neg p_i$) và tồn tại ít nhất một ứng dụng tham chiếu khác nhau đến hai mảnh f_i và f_j (tương ứng m_i và m_j). Do đó, p_i là vị từ thích hợp nếu và chỉ nếu:

$$\frac{\text{acc}(m_i)}{\text{card}(f_i)} \neq \frac{\text{acc}(m_j)}{\text{card}(f_j)}$$

2- Một tập hợp các vị từ đơn giản P_r được gọi là **đầy đủ** (complete) nếu và chỉ nếu bất kỳ hai bộ nào thuộc bất kỳ mảnh giao tối thiểu nào được định nghĩa theo P_r thì bất kỳ ứng dụng nào đều tham chiếu đến hai bộ này với cùng một xác suất.

3- Một tập hợp các vị từ đơn giản P_r được gọi là **tối thiểu** (minimal) nếu tất cả các vị từ của nó là các vị từ thích hợp.

Ví dụ 4.5 Xét phân mảnh ngang ở ví dụ 4.1. Giả sử một số ứng dụng quan trọng yêu cầu thông tin về các nhân viên là thành viên của các dự án và cũng có các ứng dụng quan trọng khác chỉ yêu cầu dữ liệu của các nhân viên là người lập trình (JOB = 'P'). Các ứng dụng sau cùng này có thể được chạy tại bất kỳ nơi nào của CSDL phân bố và chúng tham chiếu đến tất cả người lập trình với cùng xác suất. Để đơn giản, chúng ta giả sử chỉ có hai phòng ban có DEPTNUM là 1 và 2, do đó DEPTNUM = 1 tương đương với DEPTNUM $\neq$ 2 và ngược lại. Trong ví dụ này, tập hợp các vị từ đơn giản P_r là:

$$P_r = \{ \text{DEPTNUM} = 1, \text{JOB} = 'P' \}$$

Các vị từ giao tối thiểu là:

$$m_1: \text{DEPTNUM} = 1 \wedge \text{JOB} = 'P'$$

$$m_2: \text{DEPTNUM} = 1 \wedge \text{JOB} \neq 'P'$$

$$m_3: \text{DEPTNUM} \neq 1 \wedge \text{JOB} = 'P'$$

$$m_4: \text{DEPTNUM} \neq 1 \wedge \text{JOB} \neq 'P'$$

Tất cả các vị từ đơn giản ở trên là các vị từ thích hợp, trong khi đó vị từ SAL > 35000 không, là vị từ thích hợp. Trên thực tế, nếu chúng ta phân chia bất kỳ một trong các mảnh tương ứng với một trong các vị từ giao tối thiểu ở trên thành hai phần có SAL $\leq$ 35000 và SAL > 35000, thì tất cả các ứng dụng đều tham chiếu đến hai phần này theo cùng một cách thức.

Bây giờ, chúng ta có thể phát biểu như sau:

Cho $P_r = \{p_1, p_2, \dots, p_m\}$ là một tập hợp các vị từ đơn giản. Để cho P_r biểu diễn phân mảnh đúng đắn và hiệu quả thì P_r phải **đầy đủ** và **tối thiểu**.

Ví dụ 4.6 Có thể sử dụng các ví dụ 4.1 và 4.5 để hiểu rõ các định nghĩa này.

Tập hợp $P_1 = \{\text{DEPTNUM} = 1\}$ là không đầy đủ, bởi vì các ứng dụng tham chiếu đến các bộ của những người lập trình ở trong mỗi mảnh được tạo ra bởi P_1 với xác suất lớn hơn.

Tập hợp $P_2 = \{\text{DEPTNUM} = 1; \text{JOB} = 'P'\}$ là đầy đủ và tối thiểu.

Tập hợp $P_2 = \{\text{DEPTNUM} = 1; \text{JOB} = 'P', \text{SAL} > 35000\}$ là đầy đủ nhưng không tối thiểu, bởi vì $\text{SAL} > 35000$ không là vị từ thích hợp.

Bây giờ, chúng ta có thể trình bày một giải thuật tạo ra một tập hợp các vị từ P_r là đầy đủ và tối thiểu. Giải thuật này được gọi là COM_MIN và được trình bày trong giải thuật 4.1.

Bắt đầu: Xét một vị từ p_i phân chia các bộ của của R thành hai phần và tồn tại ít nhất một ứng dụng tham chiếu khác nhau đến hai phần này. Cho $P_r = p_i$.

Phương pháp: Xét một vị từ đơn giản mới p_j mà phân chia ít nhất một mảnh f_k của P_r thành hai phần và tồn tại ít nhất một ứng dụng tham chiếu khác nhau đến hai phần này. Cho $P_r \leftarrow P_r \cup p_j$. Loại bỏ các vị từ không thích hợp ra khỏi P_r . Lặp lại bước này cho đến khi tập hợp các mảnh giao tối thiểu của P_r là đầy đủ.

Để tránh diễn đạt dài dòng, chúng ta chấp nhận các ký hiệu sau đây:

Quy tắc 1: là quy tắc cơ bản về tính đầy đủ và tính tối thiểu mà một quan hệ hoặc một mảnh được phân chia thành ít nhất hai phần và tồn tại ít nhất một ứng dụng tham chiếu khác nhau đến hai phần này.

f_i của P_r : mảnh f_i được xác định theo vị từ giao tối thiểu được định nghĩa trên các vị từ của P_r .

Giải thuật 4.1 COM_MIN

Phần nhập: R : quan hệ; P_r : tập hợp các vị từ đơn giản

Phần xuất: P_r : tập hợp các vị từ đơn giản là đầy đủ và tối thiểu

declare

F : tập hợp các mảnh giao tối thiểu;

begin

 Tìm $p_i \in P_r$ sao cho p_i phân chia R theo quy tắc 1;

$P_r \leftarrow p_i$

$P_r \leftarrow P_r - p_i$

$F \leftarrow f_i$ $\{f_i \text{ là mảnh giao tối thiểu theo } p_i\}$

```

repeat
  begin
    Tìm  $p_i \in P_r$  sao cho  $p_i$  phân chia mảnh  $f_k$  nào đó của  $P_r$  theo quy
    tắc 1;
     $P_r \leftarrow P_r \cup p_i$ 
     $P_r \leftarrow P_r - p_i$ 
     $F \leftarrow F \cup f_i$ 
    while  $\exists p_k \in P_r$  là một vị từ không thích hợp do
      begin
         $P_r \leftarrow P_r - p_k$ 
         $F \leftarrow F - f_k$ 
      end
    end
  until  $P_r$  là đầy đủ
end. {COM_MIN}

```

Ví dụ 4.7 Ví dụ này là phần tiếp của các ví dụ 4.1, 4.5 và 4.6, mục đích của ví dụ này cho thấy áp dụng của phương pháp trên và cũng cho thấy việc loại bỏ các vị từ không thích hợp ra khỏi tập hợp P_r .

Xét vị từ đầu tiên là $SAL > 35000$, chúng ta có thể giả sử những người lập trình có tiền lương trung bình lớn hơn 35000, và do đó vị từ này xác định hai tập hợp nhân viên được tham chiếu khác nhau bởi các ứng dụng (lưu ý rằng chúng ta đang xác nhận vị từ $SAL > 35000$ là thích hợp đối với tập hợp $P_1 = \{SAL > 35000\}$).

Sau đó, xét $DEPTNUM = 1$, vị từ này là thích hợp và được đưa vào tập hợp trên, cho ra tập hợp $P_2 = \{SAL > 35000; DEPTNUM = 1\}$.

Cuối cùng, xét $JOB = 'P'$. Vị từ này là thích hợp và chúng ta xây dựng tập hợp cuối cùng $P_3 = \{SAL > 35000; DEPTNUM = 1; JOB = 'P'\}$. Sau đó, chúng ta phát hiện ra $SAL > 35000$ là không thích hợp đối với P_1 (xem ví dụ 4.6). Do đó, chúng ta có tập hợp cuối cùng $P_4 = \{DEPTNUM = 1, JOB = 'P'\}$ là đầy đủ và tối thiểu.

Bước thứ hai trong quá trình thiết kế phân mảnh ngang chính là tìm tập hợp các vị từ giao tối thiểu có thể được định nghĩa trên các vị từ trong tập hợp P_r . Các vị từ giao tối thiểu này xác định các mảnh được sử dụng trong bước định vị. Lưu ý rằng, việc xác định một tập hợp đầy đủ các vị từ có thể rất tốn kém trong một số trường hợp, bởi vì tập hợp này có thể rất lớn (trong thực tế, là lũy

thừa của số vị từ đơn giản). Chẳng hạn, điều này xảy ra khi đưa ra các vị từ để mô tả các tham chiếu khác nhau của các ứng dụng khác nhau, sử dụng các tiêu chí khác nhau. Trong trường hợp này, các mảnh sẽ tương ứng với các kết hợp của các tiêu chí này. Trong thực tế, việc tìm một tập hợp đầy đủ các vị từ phải được thực hiện theo một cách "hợp lý", bằng cách:

- Tập trung vào các ứng dụng khá quan trọng
- Không phân biệt các mảnh có các đặc điểm rất giống nhau.

Trong tập hợp đầy đủ các vị từ, chúng ta loại bỏ các vị từ vô nghĩa mà chúng mâu thuẫn với tập hợp các phép suy diễn I để tìm ra các vị từ giao tối thiểu.

Ví dụ, nếu $P_r = \{p_1, p_2\}$ với:

$$p_1: att = value_1$$

$$p_2: att = value_2$$

và miền của att là $\{value_1, value_2\}$, thì tập hợp I bao gồm hai phép suy diễn (implication) là:

$$i_1: (att = value_1) \Rightarrow \neg (att = value_2)$$

$$i_2: \neg (att = value_1) \Rightarrow (att = value_2)$$

Bốn vị từ sau đây được định nghĩa theo P_r :

$$m_1: (att = value_1) \wedge (att = value_2)$$

$$m_2: (att = value_1) \wedge \neg (att = value_2)$$

$$m_3: \neg (att = value_1) \wedge (att = value_2)$$

$$m_4: \neg (att = value_1) \wedge \neg (att = value_2)$$

Trong trường hợp này, các vị từ m_1 và m_4 mâu thuẫn với tập hợp các phép suy diễn I và do đó có thể loại bỏ chúng ra khỏi tập hợp M . Trong tập hợp I , các vị từ m_2 và m_3 có thể được rút gọn:

$$m_2: att = value_1$$

$$m_3: att = value_2$$

Giải thuật để phân mảnh ngang chính được trình bày trong giải thuật 4.2 và được gọi là PHORIZONTAL. Phần nhập là một quan hệ R_i cần được phân mảnh ngang chính, và P_r là tập hợp các vị từ đơn giản được xác định theo các ứng dụng tham chiếu đến R_i . Phần xuất là tập hợp các vị từ giao tối thiểu M_i dùng để tạo ra các mảnh giao tối thiểu (là các mảnh ngang của R_i).

Giải thuật 4.2 PHORIZONTAL

Phần nhập: R_i : quan hệ; P_i : tập hợp các vị từ đơn giản.

Phần xuất: M_i : tập hợp các mảnh giao tối thiểu.

begin

$P_{ri} \leftarrow \text{COM_MIN}(R_i, R_{ri});$

Xác định M_i là tập hợp các vị từ giao tối thiểu;

Xác định I là tập các phép suy diễn giữa $p_i \in P_{ri}$

for each $m_i \in M_i$ **do**

if m_i là mâu thuẫn với I **then**

$M_i \leftarrow M_i - m_i$

end. {PHORIZONTAL}

Ví dụ 4.8 Trong phần này và các phần sau, chúng ta trình bày một ví dụ tổng quát dựa trên các quan hệ toàn cục của EXAMPLE.DDB của chương 3. Xét CSDL phân bố của một công ty tại California có ba nơi là San Francisco (nơi 1), Fresno (nơi 2) và Los Angeles (nơi 3). Fresno nằm ở chính giữa San Francisco và Los Angeles. Có 30 phòng ban được gom nhóm về mặt vật lý như sau: các phòng ban có mã từ 1 đến 10 được đặt gần San Francisco, các phòng ban có mã từ 11 đến 20 được đặt gần Fresno và các phòng ban có mã từ 21 đến 30 được đặt gần Los Angeles. Tất cả các nhà cung cấp của công ty đều ở thành phố San Francisco hoặc Los Angeles. Vùng phía Bắc bao gồm San Francisco, vùng phía Nam bao gồm Los Angeles, và Fresno ở ngay ranh giới giữa hai vùng, với một số phòng ban gần Fresno thuộc vùng phía Bắc và một số phòng ban gần Fresno thuộc vùng phía Nam.

Lược đồ toàn cục trong EXAMPLE.DDB bao gồm các quan hệ EMP, DEPT, SUPPLIER và SUPPLY. Chúng ta thiết kế phân mảnh ngang chính cho SUPPLIER và DEPT.

Chúng ta bắt đầu phân tích quan hệ SUPPLIER (SNUM, NAME, CITY). Giả sử thành phố chỉ có hai giá trị là 'SF' và 'LA'; số nhà cung cấp ở 'SF' và 'LA' gần bằng nhau. Giả sử chỉ có một ứng dụng quan trọng, yêu cầu NAME của nhà cung cấp có mã SNUM cho trước. Truy vấn SQL của ứng dụng này là:

```
select NAME
from SUPPLIER
where SNUM = $X;
```

Ứng dụng này được chạy tại bất kỳ nơi nào, nếu nó được chạy tại nơi 1 thì nó tham chiếu đến các nhà cung cấp có CITY là 'SF' với xác suất 80%, nếu nó được chạy tại nơi 2 thì nó tham chiếu đến các nhà cung cấp của 'SF' và 'LA' với xác suất bằng nhau, nếu nó được chạy tại nơi 3 thì nó tham chiếu đến các nhà cung cấp có CITY là 'LA' với xác suất 80%. Điều này xảy ra là do các phòng ban xung quanh một thành phố có xu hướng giao dịch với các nhà cung cấp ở gần các phòng ban này.

Chúng ta có thể áp dụng phương pháp để tạo ra các vị từ cho ví dụ đơn giản này:

$$p_1: \text{CITY} = \text{'SF'}$$

$$p_2: \text{CITY} = \text{'LA'}$$

Bởi vì tập hợp $\{p_1, p_2\}$ là đầy đủ và tối thiểu, do đó kết thúc việc tìm kiếm.

Mặc dù đơn giản nhưng ví dụ này cho thấy hai đặc điểm quan trọng:

1- Các vị từ mà chúng thích hợp cho việc mô tả phân mảnh này không thể được suy ra bằng việc phân tích mã lệnh (*code*) của một ứng dụng. Trong ví dụ này, các giá trị của CITY tạo ra các vị từ và trong truy vấn không có tham chiếu nào đến chúng.

2- Các phép suy diễn giữa các vị từ làm giảm số mảnh. Trong trường hợp này, chúng ta nên xét các mảnh tương ứng với các vị từ sau đây:

$$m_1: (\text{CITY} = \text{'SF'}) \wedge (\text{CITY} = \text{'LA'})$$

$$m_2: (\text{CITY} = \text{'SF'}) \wedge \neg (\text{CITY} = \text{'LA'})$$

$$m_3: \neg (\text{CITY} = \text{'SF'}) \wedge (\text{CITY} = \text{'LA'})$$

$$m_4: \neg (\text{CITY} = \text{'SF'}) \wedge \neg (\text{CITY} = \text{'LA'})$$

nhưng chúng ta biết rằng:

$$(\text{CITY} = \text{'LA'}) \Rightarrow \neg (\text{CITY} = \text{'SF'})$$

$$(\text{CITY} = \text{'SF'}) \Rightarrow \neg (\text{CITY} = \text{'LA'})$$

và do đó chúng ta có thể suy ra m_1 và m_4 là mâu thuẫn, và m_2 và m_3 được rút gọn thành hai vị từ đơn giản p_1 và p_2 .

Bây giờ chúng ta xét quan hệ toàn cục:

DEPT (DEPTNUM, NAME, AREA, MGRNUM)

Chúng ta tập trung vào các ứng dụng quan trọng sau đây:

Các ứng dụng quản trị được chạy tại nơi 1 và nơi 3: các ứng dụng quản trị về các phòng ban thuộc vùng phía Bắc được chạy tại nơi 1. Các ứng dụng quản trị về các phòng ban thuộc vùng phía Nam được chạy tại nơi 3.

Các ứng dụng về các công việc thuộc mỗi phòng ban: chúng có thể được chạy tại bất kỳ phòng ban nào, nhưng chúng tham chiếu đến các bộ của các phòng ban ở gần nơi gốc với xác suất cao hơn so với các bộ của các phòng ban khác.

Bằng cách áp dụng phương pháp này, chúng ta tạo ra tập hợp các vị từ sau đây:

$$p_1: \text{DEPTNUM} \leq 10$$

$$p_2: 10 < \text{DEPTNUM} \wedge \text{DEPTNUM} \leq 20$$

$$p_3: \text{DEPTNUM} > 20$$

$$p_4: \text{AREA} = \text{'NORTH'}$$

$$p_5: \text{AREA} = \text{'SOUTH'}$$

Nếu theo giải thuật COM_MIN thì tập hợp $P_r = \{p_1, p_2, p_3, p_4, p_5\}$ là đầy đủ và tối thiểu.

Dựa vào P_r , 6 vị từ sau đây có thể được định nghĩa để tạo ra M :

$$m_1: (\text{DEPTNUM} \leq 10) \wedge (\text{AREA} = \text{'NORTH'})$$

$$m_2: (\text{DEPTNUM} \leq 10) \wedge (\text{AREA} = \text{'SOUTH'})$$

$$m_3: (10 < \text{DEPTNUM} \wedge \text{DEPTNUM} \leq 20) \wedge (\text{AREA} = \text{'NORTH'})$$

$$m_4: (10 < \text{DEPTNUM} \wedge \text{DEPTNUM} \leq 20) \wedge (\text{AREA} = \text{'SOUTH'})$$

$$m_5: (\text{DEPTNUM} > 20) \wedge (\text{AREA} = \text{'NORTH'})$$

$$m_6: (\text{DEPTNUM} > 20) \wedge (\text{AREA} = \text{'SOUTH'})$$

Các vị từ này không phải là các vị từ duy nhất có thể được tạo ra. Ví dụ, có thể nêu ra các vị từ có dạng:

$$p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge p_5$$

Tuy nhiên, các phép suy diễn hiển nhiên:

$$i_1: p_1 \Rightarrow \neg p_2 \wedge \neg p_3$$

$$i_2: p_2 \Rightarrow \neg p_1 \wedge \neg p_3$$

$$i_3: p_3 \Rightarrow \neg p_1 \wedge \neg p_2$$

$$i_4: p_4 \Rightarrow \neg p_5$$

$$i_5: p_5 \Rightarrow \neg p_4$$

$$i_6: \neg p_4 \Rightarrow p_5$$

$$i_7: \neg p_5 \Rightarrow p_4$$

loại bỏ các vị từ này và chúng ta giữ lại m_1 đến m_6 .

Các phép suy diễn có thể được xác định theo ngữ nghĩa của CSDL, mà không theo các giá trị hiện tại. Chúng ta có các phép suy diễn sau đây:

$$i_8: \text{DEPTNUM} \leq 10 \Rightarrow \text{AREA} = \text{'NORTH'}$$

$$i_9: \text{DEPTNUM} > 20 \Rightarrow \text{AREA} = \text{'SOUTH'}$$

Do đó, các vị từ m_2 và m_5 là mâu thuẫn và cuối cùng chúng ta có bốn vị từ giao tối thiểu đã được rút gọn là:

$$m_1: \text{DEPTNUM} \leq 10$$

$$m_3: (10 < \text{DEPTNUM} \wedge \text{DEPTNUM} \leq 20) \wedge (\text{AREA} = \text{'NORTH'})$$

$$m_4: (10 < \text{DEPTNUM} \wedge \text{DEPTNUM} \leq 20) \wedge (\text{AREA} = \text{'SOUTH'})$$

$$m_6: \text{DEPTNUM} > 20$$

Một nhận xét cuối cùng từ ví dụ này là bàn luận về định vị mảnh. Rõ ràng, mảnh tương ứng với m_1 được đặt tại nơi 1 và mảnh tương ứng với m_4 được đặt tại nơi 3. Việc định vị các mảnh tương ứng với m_3 và m_4 là sự thỏa thuận (*trade-off*) giữa các ứng dụng quản trị (mà các mảnh được đặt tại nơi 1 hoặc 3) với các ứng dụng về các công việc của phòng ban (mà các mảnh được đặt tại nơi 2). Sự thỏa thuận này sẽ được giải quyết trong phần định vị mảnh kế tiếp. Một lần nữa chúng ta nhấn mạnh các mảnh tương ứng với m_2 và m_5 là các đơn vị thích hợp cho vấn đề định vị.

3- Phân mảnh ngang dẫn xuất

Phân mảnh ngang dẫn xuất của một quan hệ toàn cục R không dựa trên các đặc tính của các thuộc tính riêng của nó, mà được suy dẫn từ phân mảnh ngang của một quan hệ khác.

Phân mảnh ngang dẫn xuất được định nghĩa trên các quan hệ bộ phận của đường liên kết theo phép chọn trên quan hệ chủ của đường liên kết này. Điều quan trọng cần nhớ hai điểm:

Thứ nhất, đường liên kết giữa quan hệ chủ và quan hệ bộ phận được định nghĩa là một phép kết bằng.

Thứ hai, một phép kết bằng có thể được thực hiện bằng các phép nửa kết.

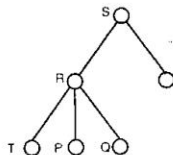
Điểm thứ hai là đặc biệt quan trọng đối với các mục đích của chúng ta, bởi vì chúng ta muốn phân chia quan hệ bộ phận theo sự phân mảnh của quan hệ chủ của nó, nhưng chúng ta cũng muốn mảnh kết quả chỉ được định nghĩa trên các thuộc tính của quan hệ bộ phận.

Do đó, cho trước một đường liên kết L với $owner(L) = S$ và $member(L) = R$, các mảnh ngang dẫn xuất của R được định nghĩa như sau:

$$R_i = R \triangleright_{<F} S_i, \quad 1 \leq i \leq n$$

với n là số lượng lớn nhất các mảnh sẽ được định nghĩa trên R , và $S_i = \sigma_{F_i}(S)$ với F_i là công thức dùng để định nghĩa mảnh ngang chính S_i , và F là điều kiện nửa kết (*semi-join condition*).

Quan hệ R có thể là quan hệ chủ của một đường liên kết khác là L' và quan hệ bộ phận là T , và cứ như thế. Trong trường hợp tổng quát, chúng ta có thể có một **cây phân mảnh ngang dẫn xuất** (*derived horizontal fragmentation tree*), trong đó nút cha là quan hệ chủ và các nút con của nó là các quan hệ bộ phận (H.4.3).



Hình 4.3 Cây phân mảnh ngang dẫn xuất

Để thực hiện phân mảnh ngang dẫn xuất, có ba dữ liệu vào là:

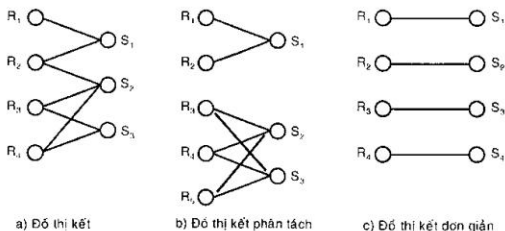
- Tập hợp các mảnh của quan hệ chủ
- Quan hệ bộ phận
- Tập hợp các vị từ nửa kết giữa quan hệ chủ và quan hệ bộ phận.

Phân mảnh ngang dẫn xuất được dùng để tạo điều kiện thuận lợi cho phép kết giữa các mảnh.

Một **phép kết phân bố** (*distributed join*) là một phép kết giữa các quan hệ được phân mảnh ngang. Khi một ứng dụng cần thực hiện phép kết giữa hai quan hệ toàn cục R và S , thì tất cả các bộ của R và của S phải được so sánh với nhau. Do đó theo nguyên tắc, cần phải so sánh tất cả các mảnh R_i của R với tất cả các mảnh S_j của S . Tuy nhiên, đôi khi có thể suy diễn để xác định một số phép kết từng phần $R_i \triangleright_{<F} S_j$ là rỗng. Đối với sự phân bố dữ liệu cho trước, điều này xảy ra khi các giá trị của thuộc tính kết của R_i và S_j là khác nhau.

Một phép kết phân bố được biểu diễn một cách hiệu quả bằng **đồ thị kết** (*join graph*). Đồ thị kết G của phép kết phân bố $R \triangleright_{<F} S$ là một đồ thị (N, E) với các nút N biểu diễn các mảnh của R và S và các cạnh vô hướng giữa các nút biểu diễn các phép kết giữa các mảnh mà

kết quả của phép kết là khác rỗng. Để đơn giản, chúng ta không đưa vào trong N các mảnh của R hoặc của S mà chúng kết với tất cả các mảnh của quan hệ khác đều cho ra kết quả rỗng. Một ví dụ của đồ thị kết được trình bày trong hình 4.4a.



Hình 4.4 Các đồ thị kết

Đồ thị kết được gọi là **hoàn toàn** (total) nếu nó chứa tất cả các cạnh có thể có giữa các mảnh của R và S . Đồ thị kết được gọi là **suy giảm** (reduced) nếu không có một số cạnh giữa các mảnh của R và S . Đồ thị kết suy giảm gồm có hai loại:

1- Đồ thị kết suy giảm được gọi là **phân tách** (partitioned) nếu đồ thị bao gồm hai hoặc nhiều đồ thị con và không có các cạnh giữa chúng (H.4.4b).

2- Đồ thị kết suy giảm được gọi là **đơn giản** (simple) nếu nó là phân tách và mỗi đồ thị con có đúng một cạnh (H.4.4c).

Việc xác định một phép kết có đồ thị kết đơn giản là rất quan trọng trong thiết kế CSDL. Một cặp mảnh được nối với nhau bằng một cạnh trong đồ thị kết đơn giản có một tập hợp các giá trị chung của các thuộc tính kết. Do đó có thể xác định phân mảnh và định vị của hai quan hệ toán hạng R và S sao cho đồ thị kết là đồ thị đơn giản và các cặp mảnh tương ứng được đặt tại cùng một nơi, khi đó phép kết có thể được thực hiện theo cách phân bố bằng cách thực hiện phép kết cục bộ các cặp mảnh và sau đó tập hợp các kết quả của các phép kết từng phần này. Do đó, điều quan trọng để thiết kế CSDL phân bố là các phép kết được thực hiện thường xuyên phải có đồ thị kết đơn giản.

Các đồ thị kết phân tách cũng có ích lợi, bởi vì chúng có thể được tối ưu hóa một cách dễ dàng hơn bằng cách sử dụng các kỹ thuật sẽ được trình bày trong chương 6. Do đó, điều thuận lợi để định nghĩa phân mảnh là: đối với các phép kết được thực hiện rất thường xuyên, nếu chúng không có đồ thị kết đơn giản thì chúng phải có đồ thị kết phân tách.

Ví dụ 4.9 Bây giờ chúng ta xét quan hệ SUPPLY (SNUM, PNUM, DEPTNUM, QUAN). Cho các ứng dụng sử dụng quan hệ này, sao cho SUPPLY luôn luôn được sử dụng chung với quan hệ khác, cụ thể:

- Một số ứng dụng yêu cầu thông tin về việc cung cấp hàng của các nhà cung cấp cho trước, do đó các ứng dụng này kết SUPPLY và SUPPLIER dựa trên thuộc tính SNUM.

- Các ứng dụng khác yêu cầu thông tin về việc cung cấp tại một phòng ban cho trước, do đó các ứng dụng này kết SUPPLY và DEPT dựa trên thuộc tính DEPTNUM.

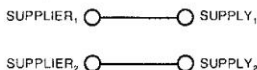
Chúng ta giả sử quan hệ toàn cục DEPT được phân mảnh ngang theo các giá trị của thuộc tính DEPTNUM và SUPPLIER được phân mảnh ngang theo các giá trị của thuộc tính SNUM. Có hai cách phân mảnh ngang dẫn xuất cho quan hệ SUPPLY: thông qua phép nối kết với SUPPLIER dựa trên SNUM và thông qua phép nối kết với DEPT dựa trên DEPTNUM, cả hai cách này đều đúng.

Trong lược đồ CSDL, nói chung có thể có nhiều đường liên kết đến một quan hệ R (ví dụ có hai đường liên kết đến quan hệ SUPPLY trong hình 4.2). Trong trường hợp này, có nhiều cách phân mảnh ngang dẫn xuất cho R . Việc quyết định chọn cách phân mảnh nào được dựa trên hai tiêu chuẩn:

- 1- Sự phân mảnh có các đặc điểm kết (*join characteristic*) tốt hơn.
- 2- Sự phân mảnh được sử dụng trong nhiều ứng dụng hơn.

Trước tiên, chúng ta bàn luận đến tiêu chuẩn thứ hai. Tiêu chuẩn này hoàn toàn không phức tạp nếu chúng ta xét đến tần số mà các ứng dụng truy xuất dữ liệu nào đó. Nếu có thể được, chúng ta nên tạo điều kiện thuận lợi cho các truy xuất của những người sử dụng *quan trọng* để cho toàn bộ các tác động của họ trên hiệu suất hệ thống là tối thiểu.

Tuy nhiên, việc áp dụng tiêu chuẩn thứ nhất thì không đơn giản. Ví dụ, xét phân mảnh dẫn xuất của SUPPLY theo SUPPLIER dựa trên thuộc tính SNUM. Kết quả (và mục tiêu) của phân mảnh này là phép kết của các quan hệ SUPPLIER và SUPPLY để trả lời truy vấn sẽ được trợ giúp (1) bằng cách thực hiện nó trên các quan hệ nhỏ hơn (tức là các mảnh), và (2) bằng cách thực hiện các phép kết theo kiểu phân bố. Cách đầu tiên là hiển nhiên. Các mảnh của SUPPLY thì nhỏ hơn SUPPLY. Do đó, phép kết một mảnh bất kỳ của SUPPLY với một mảnh bất kỳ của SUPPLIER sẽ nhanh hơn so với phép kết giữa các quan hệ. Tuy nhiên, cách thứ hai là quan trọng hơn và là điều chủ yếu của các CSDL phân bố. Ngoài việc thực hiện các truy vấn tại các nơi khác nhau, nếu chúng ta có thể thực hiện song song một truy vấn thì thời gian đáp ứng hoặc thông lượng của hệ thống có thể được cải tiến. Trong trường hợp các phép kết, điều này có thể có trong một số tình huống. Ví dụ, xét đồ thị kết giữa các mảnh của SUPPLIER và SUPPLY trong hình 4.5, đồ thị kết này là đồ thị đơn giản. Do đó, mỗi cặp mảnh có thể được đặt tại cùng một nơi và các phép kết giữa các cặp mảnh khác nhau có thể được thực hiện một cách độc lập và song song.



Hình 4.5 Đồ thị kết SUPPLIER với SUPPLY

4- Kiểm tra tính đúng đắn

Bây giờ, chúng ta kiểm tra các giải thuật phân mảnh đối với ba điều kiện đúng đắn đã được trình bày trong phần 3.2.

a- Điều kiện đầy đủ

Tính đầy đủ của phân mảnh ngang chính được dựa trên các vị từ chọn được sử dụng. Với điều kiện các vị từ chọn là đầy đủ thì kết quả của việc phân mảnh cũng đảm bảo là đầy đủ. Bởi vì cơ sở của giải thuật phân mảnh là một tập hợp P_r các vị từ đầy đủ và tối thiểu, do đó điều kiện đầy đủ được đảm bảo nếu không xảy ra các sai lầm trong việc định nghĩa P_r .

Tính đầy đủ của phân mảnh ngang dẫn xuất hơi khó xác định hơn. Điều khó khăn là do vị từ xác định phân mảnh liên quan đến hai quan hệ. Trước tiên, chúng ta định nghĩa quy tắc đầy đủ một cách hình thức.

Cho R là quan hệ bộ phận của một đường liên kết mà quan hệ chủ là S , và S được phân mảnh thành $F_S = \{S_1, S_2, \dots, S_n\}$. Cho A là thuộc tính kết giữa R và S . Khi đó, đối với mỗi bộ t của R , tồn tại một bộ t' của S sao cho:

$$t[A] = t'[A]$$

Ví dụ, mã nhà cung cấp $SNUM$ của một bộ bất kỳ của $SUPPLY$ phải có trong $SUPPLIER$. Quy tắc này được gọi là ràng buộc toàn vẹn tham chiếu (*referential integrity constraint*) và bảo đảm tất cả các giá trị của thuộc tính kết của quan hệ bộ phận đều có trong quan hệ chủ.

b- Điều kiện tái tạo

Sự tái tạo quan hệ toàn cục từ các mảnh của nó được thực hiện bằng phép hợp cho cả hai loại phân mảnh ngang chính và phân mảnh ngang dẫn xuất. Do đó, nếu quan hệ R được phân mảnh ngang thành $F_R = \{R_1, R_2, \dots, R_n\}$ thì:

$$R' = \cup R_i, \quad \forall R_i \in F_R$$

c- Điều kiện tách biệt

Tính tách biệt của phân mảnh ngang chính được chứng minh dễ dàng hơn so với phân mảnh ngang dẫn xuất. Đối với phân mảnh ngang chính, tính tách biệt được bảo đảm với điều kiện các vị từ giao tối thiểu xác định phân mảnh là loại trừ lẫn nhau.

Tuy nhiên, trong phân mảnh ngang dẫn xuất, phép nối kết làm tăng độ phức tạp đáng kể. Tính tách biệt có thể được bảo đảm nếu đồ thị kết là đồ thị đơn giản. Nếu đồ thị kết không là đồ thị đơn giản thì cần phải xét các giá trị hiện tại của các bộ. Nói chung, chúng ta không muốn một bộ của quan hệ bộ phận kết với hai hoặc nhiều bộ của quan hệ chủ khi các bộ này thuộc các mảnh khác nhau của quan hệ chủ. Điều này không thể dễ dàng đạt được và cho thấy tại sao luôn luôn chúng ta mong muốn các lược đồ phân mảnh dẫn xuất tạo ra đồ thị kết đơn giản.

4.2.2 Phân mảnh dọc

Việc xác định phân mảnh dọc một quan hệ toàn cục R đòi hỏi việc gom nhóm các thuộc tính của R thành các tập thuộc tính mà chúng được các ứng dụng tham chiếu theo cùng một cách thức. Chúng ta phân biệt *phân mảnh dọc tách biệt* (*vertical partitioning fragmentation*), mà các tập thuộc tính phải không giao nhau, với *phân mảnh dọc gom tụ* (*vertical clustering fragmentation*), mà các tập thuộc tính có thể giao nhau. Các điều kiện đúng đắn đối với phân mảnh dọc tách biệt đòi hỏi mỗi thuộc tính của R phải thuộc ít nhất một tập thuộc tính và mỗi tập thuộc tính phải chứa khóa của R hoặc *danh hiệu bộ* (*tuple identifier*).

Mục đích của phân mảnh dọc là xác định các mảnh R , sao cho nhiều ứng dụng có thể được thực hiện bằng cách chỉ sử dụng một mảnh. Trong ngữ cảnh này, *phân mảnh tối ưu* (*optimal fragmentation*) là phân mảnh tạo ra một lược đồ phân mảnh mà lược đồ này tối thiểu hóa thời gian thực hiện của các ứng dụng chạy trên các mảnh này. Chúng ta xét một quan hệ toàn cục R được phân mảnh dọc thành R_1 và R_2 . Một ứng dụng có được ưu điểm của phân mảnh dọc này nếu nó có thể được thực hiện bằng cách sử dụng mảnh R_1 hoặc R_2 , bởi vì trong trường hợp này nên tránh truy xuất quan hệ R (lớn). Tuy nhiên, nếu một ứng dụng cần truy xuất cả R_1 và R_2 , thì việc phân mảnh là không có lợi, bởi vì cần phải thực hiện phép kết để tạo lại R . Tiêu chuẩn này cũng được áp dụng cho các CSDL không tập trung; trong CSDL phân bố, phân mảnh dọc có ưu điểm hơn khi nhiều ứng dụng sử dụng R_1 và nhiều ứng dụng sử dụng R_2 chạy tại nhiều nơi khác nhau. Trong trường hợp này, việc đặt R tại một nơi nào đó sẽ không có lợi cho các ứng dụng tại các nơi khác, trong khi đó việc phân mảnh R cho phép thỏa mãn cả hai.

Việc xác định phân mảnh dọc một quan hệ toàn cục R là không dễ dàng, bởi vì số cách phân mảnh có thể có sẽ tăng tổ hợp theo số thuộc tính của R , và ngay cả số nhóm (*cluster*) có thể có còn lớn hơn. Do đó, trong trường hợp một quan hệ lớn, cần phải có các cách tiếp cận mang tính kinh nghiệm (*heuristic approach*) để xác định phân mảnh hoặc gom nhóm. Chúng ta trình bày ngắn gọn các phương pháp này thực hiện như thế nào.

Hai cách tiếp cận mang tính kinh nghiệm để phân mảnh dọc một quan hệ toàn cục là:

Cách tiếp cận phân tách (*split approach*) mà theo đó các quan hệ toàn cục được phân tách dần dần để tạo thành các mảnh. Bắt đầu từ một quan hệ và quyết định chọn các cách phân mảnh có lợi, dựa trên cách truy xuất của các ứng dụng đến các thuộc tính.

Cách tiếp cận gom nhóm (*grouping approach*) mà theo đó các thuộc tính được kết hợp lại dần dần để tạo thành các mảnh. Bắt đầu bằng cách gán mỗi thuộc tính cho mỗi mảnh, và ở mỗi bước, kết hợp một số mảnh với nhau cho đến khi thỏa mãn một tiêu chuẩn nào đó.

Có thể phân loại cả hai cách tiếp cận này là các *phương pháp kinh nghiệm tham lam* (*greedy heuristics*), bởi vì chúng tiến hành bằng cách thực hiện việc chọn lựa *tốt nhất* có thể có ở mỗi lần lặp vòng. Trong cả hai trường hợp, các công thức được sử dụng để cho thấy cách phân tách hoặc cách gom nhóm nào là tốt nhất. Một số dạng lẩn ngược (*backtracking*) có thể được thực hiện bằng cách thử di chuyển các thuộc tính từ một tập hợp này qua một tập hợp khác trong cách phân mảnh cuối cùng.

Phân mảnh dọc gom tụ cho thấy **sự nhân bản** trong phạm vi các mảnh, bởi vì các giá trị của các thuộc tính trùng nhau sẽ được nhân bản. Ở đây, chúng ta bàn đến các ảnh hưởng của nhân bản mà nó sẽ được trình bày đầy đủ trong phần kế tiếp. Nhân bản có một ảnh hưởng khác đến các ứng dụng chỉ đọc và các ứng dụng cập nhật. Nhân bản có lợi cho các ứng dụng chỉ đọc, bởi vì chúng ta có thể tham chiếu dữ liệu cục bộ. Nhân bản không có lợi cho các ứng dụng cập nhật, bởi vì chúng ta phải cập nhật tất cả các bản nhân để duy trì tính nhất quán.

Chúng ta xét điều gì sẽ xảy ra khi hai mảnh R_1 và R_2 phủ lấp nhau, nghĩa là tồn tại một tập thuộc tính I thuộc cả R_1 và R_2 . Giả sử R_1 và R_2 đều đặt tại cả hai nơi là nơi 1 và 2. Sau đó, các ứng dụng đọc chạy tại nơi 1, sử dụng các thuộc tính của I chung với các thuộc tính của R_1 , là cục bộ tại nơi 1. Cũng như vậy, các ứng dụng đọc chạy tại nơi 2, sử dụng các thuộc tính của I chung với các thuộc tính của R_2 , là cục bộ tại nơi 2. Tuy nhiên, các ứng dụng cập nhật mà chúng thay đổi giá trị của các thuộc tính của I phải tham chiếu đến R_1 và R_2 tại cả hai nơi.

Do đó, trong việc đánh giá thuận lợi của phân mảnh dọc gom tụ, điều quan trọng là các thuộc tính phủ lấp phải *không được cập nhật thường xuyên*.

Ví dụ 4.10 Bây giờ, chúng ta xét quan hệ toàn cục:

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)

Giả sử các ứng dụng sau đây sử dụng quan hệ EMP:

1- Các ứng dụng quản trị, tập trung tại nơi 3, yêu cầu NAME, SAL và TAX của các nhân viên.

2- Các ứng dụng về các công việc của mỗi phòng ban, yêu cầu NAME, MGRNUM và DEPTNUM của các nhân viên. Các ứng dụng này được chạy tại tất cả các nơi và tham chiếu các bộ của các nhân viên thuộc cùng nhóm phòng ban với xác suất 80%.

Do đó, phân mảnh dọc EMP thành hai mảnh gồm các thuộc tính *quản trị* và các thuộc tính *mô tả công việc* là một cách làm tự nhiên. Chúng ta đưa khóa EMPNUM vào cả hai mảnh. Nếu chúng ta sử dụng phân mảnh dọc tách biệt, thì chúng ta phải quyết định mảnh nào trong hai mảnh có thuộc tính NAME. Trong trường hợp đơn giản, phân mảnh dọc tách biệt sẽ không hiệu quả, bởi vì một trong hai ứng dụng sẽ phải thực hiện phép kết và nhiều tham chiếu không cục bộ. Vì thuộc tính NAME tương đối ít bị thay đổi (bởi vì các nhân viên không thường xuyên thay đổi tên của họ), do đó phân mảnh dọc gom tụ là thích hợp, xác định phân mảnh như sau:

EMP₁ (EMPNUM, NAME, SAL, TAX)

EMP₂ (EMPNUM, NAME, MGRNUM, DEPTNUM)

Sau đây, chúng ta chỉ bàn luận đến kỹ thuật phân tách, bởi vì nó phù hợp một cách tự nhiên với phương pháp luận thiết kế từ trên xuống, và bởi vì một giải pháp tối ưu gắn gũi với quan hệ thực sự hơn là một tập hợp các mảnh mà mỗi mảnh chỉ có một thuộc tính. Hơn nữa, việc phân tách tạo ra các mảnh không phủ lấp nhau, trong khi đó việc gom nhóm tạo ra các mảnh phủ lấp nhau. Trong ngữ cảnh các hệ thống CSDL phân bố, chúng ta quan tâm đến các mảnh không phủ lấp nhau, vì các lý do hiển nhiên. Dĩ nhiên, việc không phủ lấp nhau chỉ xét đến các thuộc tính không là thuộc tính khóa chính.

Bây giờ, chúng ta xét sự nhân bản của khóa của quan hệ toàn cục ở trong các mảnh. Đây là một đặc tính của phân mảnh dọc để tái tạo quan hệ toàn cục. Do đó, việc phân tách chỉ được xét đối với các thuộc tính không tham gia vào khóa chính.

Một số phụ thuộc dữ liệu có liên quan đến các thuộc tính khóa của quan hệ. Nếu chúng ta thiết kế CSDL mà các thuộc tính khóa là thành phần của một mảnh được đặt tại một nơi, và các thuộc tính phụ thuộc là thành phần của một mảnh khác được đặt tại một nơi thứ hai, thì mọi yêu cầu cập nhật sẽ gây ra một kiểm tra tính toàn vẹn và cần phải truyền thông giữa các nơi. Nhân bản các thuộc tính khóa ở mỗi mảnh sẽ hạn chế điều này nhưng không thể loại bỏ điều này một cách hoàn toàn, bởi vì sự truyền thông như thế có thể cần thiết do các ràng buộc toàn vẹn không liên quan đến khóa chính, cũng như do điều khiển đồng thời.

Một cách khác để nhân bản các thuộc tính khóa là sử dụng các danh hiệu bộ (TID), đó là các giá trị duy nhất được gán bởi hệ thống cho các bộ của một quan hệ. Bởi vì các danh hiệu bộ được hệ thống duy trì, do đó các mảnh sẽ tách biệt nhau đối với người sử dụng.

1- Các yêu cầu thông tin của phân mảnh dọc

Thông tin chính cần thiết cho phân mảnh dọc có liên quan đến các ứng dụng. Do đó, bàn luận sau đây chỉ đề cập đến cái gì cần được xác định về các ứng dụng sẽ chạy trên CSDL phân bố. Bởi vì việc phân mảnh dọc đặt các thuộc tính thường được truy xuất chung với nhau vào trong một mảnh, cần phải có một số đo nào đó để định nghĩa chính xác hơn khái niệm *tính chất chung với nhau* (*togetherness*). Số đo này được gọi là *ái lực* (*affinity*) của các thuộc tính và nó cho thấy các thuộc tính liên hệ chặt chẽ với nhau như thế nào. Trên thực tế, người thiết kế hoặc người sử dụng không dễ dàng xác định được các giá trị này. Bây giờ, chúng ta trình bày một cách thức để có được các giá trị này từ nhiều dữ liệu nguyên thủy.

Yêu cầu dữ liệu chính liên quan đến các ứng dụng là tần số truy xuất của chúng. Cho $Q = \{q_1, q_2, \dots, q_i\}$ là tập hợp các truy vấn của người sử dụng (các ứng dụng) sẽ chạy trên quan hệ $R(A_1, A_2, \dots, A_n)$. Đối với mỗi truy vấn q_i và mỗi thuộc tính A_j , chúng ta gán một *giá trị sử dụng thuộc tính* (*attribute usage value*), được ký hiệu là $use(q_i, A_j)$ và được định nghĩa như sau:

$$use(q_i, A_j) = \begin{cases} 1 & \text{nếu truy vấn } q_i \text{ tham chiếu thuộc tính } A_j \\ 0 & \text{nếu truy vấn } q_i \text{ không tham chiếu thuộc tính } A_j \end{cases}$$

Các vectơ $use(q_i, \bullet)$ cho mỗi ứng dụng được xác định dễ dàng nếu người thiết kế biết các ứng dụng sẽ chạy trên CSDL. Quy tắc 80-20 được bàn luận trong phần 4.2.1 có ích trong công việc này.

Ví dụ 4.11 Xét quan hệ DEPT. (DEPTNUM, NAME, MGRNUM, AREA). Giả sử các ứng dụng sau đây được xác định chạy trên quan hệ này. Trong mỗi trường hợp, chúng ta cũng nêu ra đặc tả SQL.

q_1 : Tìm mã trưởng phòng của một phòng ban có mã cho trước

```
select MGRNUM
from DEPT
where DEPTNUM = Value;
```

q_2 : Tìm tên và mã trưởng phòng của tất cả các phòng ban

```
select NAME, MGRNUM
from DEPT;
```

q_3 : Tìm tên của các phòng ban thuộc một vùng cho trước

```
select NAME
from DEPT
where AREA = Value;
```

q_4 : Tìm mã phòng ban lớn nhất thuộc một vùng cho trước

```
select MAX(DEPTNUM)
from DEPT
where AREA = Value;
```

Theo bốn ứng dụng này, chúng ta có thể xác định các giá trị sử dụng thuộc tính. Để cho thuận tiện, chúng ta ký hiệu:

$A_1 = \text{DEPTNUM}$, $A_2 = \text{NAME}$, $A_3 = \text{MGRNUM}$, $A_4 = \text{AREA}$.

Các giá trị sử dụng được xác định trong ma trận hình 4.6, trong đó ô (i, j) biểu thị $use(q_i, A_j)$.

	A_1	A_2	A_3	A_4
q_1	1	0	1	0
q_2	0	1	1	0
q_3	0	1	0	1
q_4	0	0	1	1

Hình 4.6 Ma trận sử dụng thuộc tính của ví dụ

Các giá trị sử dụng thuộc tính không đủ tổng quát để làm cơ sở cho việc phân tách thuộc tính và phân mảnh, bởi vì các giá trị này không tiêu biểu cho tác dụng của các tần số ứng dụng. Số đo tần số có thể được đưa vào trong định nghĩa của số đo ái lực thuộc tính $aff(A_i, A_j)$, nó cho biết độ liên kết giữa hai thuộc tính của một quan hệ tùy theo các ứng dụng truy xuất chúng như thế nào.

Số đo ái lực thuộc tính giữa hai thuộc tính A_i và A_j của một quan hệ $R(A_1, A_2, \dots, A_n)$ đối với tập hợp các ứng dụng $Q = \{q_1, q_2, \dots, q_q\}$ được định nghĩa như sau:

$$aff(A_i, A_j) = \sum_{kluse(q_k, A_i)=1} \sum_{use(q_k, A_j)=1 \forall S_l} ref_l(q_k) acc_l(q_k)$$

trong đó $ref_l(q_k)$ là số lần truy xuất đến các thuộc tính (A_i, A_j) đối với mỗi lần thực hiện ứng dụng q_k tại nơi S_l và $acc_l(q_k)$ là số đo tần số truy xuất của ứng dụng đã được định nghĩa trước đó và được sửa đổi để bao gồm các tần số tại các nơi khác nhau.

Kết quả của tính toán này là một ma trận $n \times n$, mỗi phần tử của ma trận là một trong các số đo được định nghĩa ở trên. Chúng ta gọi ma trận này là **ma trận ái lực thuộc tính (AA) (Attribute Affinity matrix)**.

Ví dụ 4.12 Chúng ta tiếp tục trường hợp đã xét trong ví dụ 4.11. Để đơn giản, chúng ta giả sử $ref_l(q_k) = 1$ đối với tất cả q_k và S_l . Nếu các tần số ứng dụng là:

$$\begin{array}{lll} acc_1(q_1) = 15 & acc_2(q_1) = 20 & acc_3(q_1) = 10 \\ acc_1(q_2) = 5 & acc_2(q_2) = 0 & acc_3(q_2) = 0 \\ acc_1(q_3) = 25 & acc_2(q_3) = 25 & acc_3(q_3) = 25 \\ acc_1(q_4) = 3 & acc_2(q_4) = 0 & acc_3(q_4) = 0 \end{array}$$

thì số đo ái lực giữa các thuộc tính A_1 và A_3 có thể được tính như sau:

$$aff(A_1, A_3) = \sum_{k=1}^1 \sum_{l=1}^3 acc_l(q_k) = acc_1(q_1) + acc_2(q_1) + acc_3(q_1) = 45$$

bởi vì chỉ có ứng dụng truy xuất cả hai thuộc tính này là q_1 . Ma trận ái lực thuộc tính đầy đủ được chỉ ra trong hình 4.7. Lưu ý rằng, để cho đầy đủ, các giá trị đường chéo cũng được tính toán, mặc dù chúng không có ý nghĩa.

	A_1	A_2	A_3	A_4
A_1	45	0	45	0
A_2	0	80	5	75
A_3	45	5	53	3
A_4	0	75	3	78

Hình 4.7 Ma trận ái lực thuộc tính

Ma trận ái lực thuộc tính sẽ được sử dụng trong phần còn lại của chương này để dựa vào đó phân mảnh dữ liệu. Quá trình này trước tiên gom các thuộc tính có ái lực lớn chung với nhau và dựa vào đó để phân chia quan hệ.

2- Giải thuật gom tụ

Công việc cơ bản trong thiết kế giải thuật phân mảnh dọc là tìm kiếm một số cách gom nhóm các thuộc tính của quan hệ dựa trên các giá trị ái lực thuộc tính trong AA. *Giải thuật năng lượng liên kết* (BEA – Bond Energy Algorithm) được sử dụng cho mục đích này vì những lý do sau đây:

1- Nó được thiết kế đặc biệt để xác định các nhóm bao gồm các mục tương tự nhau, ngược với thứ tự tuyến tính của các mục (nghĩa là nó gom các thuộc tính có các giá trị ái lực lớn chung với nhau và các thuộc tính có giá trị ái lực nhỏ chung với nhau).

2- Các nhóm cuối cùng không bị ảnh hưởng bởi thứ tự của các mục được đưa vào trong giải thuật.

3- Thời gian tính toán của giải thuật là hợp lý, $O(n^2)$ với n là số thuộc tính.

4- Có thể nhận biết các mối liên kết phụ giữa các nhóm thuộc tính gom tụ.

Giải thuật năng lượng liên kết có phần nhập là ma trận ái lực thuộc tính, hoán vị các hàng và các cột của nó và tạo ra *ma trận ái lực gom tụ* (CA – Clustered Affinity matrix). Thực hiện hoán vị sao cho *số đo ái lực chung* (AM – global Affinity Measure) sau đây là lớn nhất:

$$AM = \sum_{i=1}^n \sum_{j=1}^n aff(A_i, A_j) [aff(A_i, A_{j-1}) + aff(A_i, A_{j+1}) + aff(A_{i-1}, A_j) + aff(A_{i+1}, A_j)]$$

với:

$$aff(A_0, A_j) = aff(A_i, A_0) = aff(A_{n+1}, A_j) = aff(A_i, A_{n+1}) = 0$$

Tập hợp các điều kiện cuối cùng biểu diễn các trường hợp mà một thuộc tính được đặt vào CA ở bên trái của thuộc tính cực trái hoặc ở bên phải của thuộc tính cực phải trong các hoán vị cột, và trước hàng trên cùng và sau hàng cuối cùng trong các hoán vị hàng. Trong các trường hợp này, chúng ta gán 0 cho các giá trị *aff* giữa thuộc tính được xét để đặt vào và các lân cận bên trái hoặc bên phải của nó (trên cùng hoặc cuối cùng) mà chúng không tồn tại trong CA.

Hàm cực đại hóa chỉ xét các phần tử gần nhất, do đó kết quả là gom các giá trị lớn với các giá trị lớn, các giá trị nhỏ với các giá trị nhỏ. Cũng vậy, ma trận ái lực thuộc tính (AA) là đối xứng, do đó hàm mục tiêu ở trên được rút gọn thành:

$$AM = \sum_{i=1}^n \sum_{j=1}^n aff(A_i, A_j) [aff(A_i, A_{j-1}) + aff(A_i, A_{j+1})]$$

Chi tiết của giải thuật năng lượng liên kết được trình bày trong giải thuật 4.3. Quá trình tạo ma trận ái lực gom tụ (CA) được thực hiện qua ba bước:

Bắt đầu: Đặt và cố định một cột nào đó trong các cột của AA vào trong CA. Giải thuật chọn cột 1.

Lặp: Chọn mỗi cột trong $n - i$ cột còn lại (i là số cột đã được đặt trong CA) và thử đặt chúng trong $i + 1$ vị trí còn lại trong ma trận CA. Chọn sắp đặt nào làm cho số đo ái lực chung có giá trị lớn nhất như đã được mô tả ở trên.

Sắp thứ tự hàng: Mỗi lần xác định thứ tự cột, chúng ta nên sắp đặt lại các hàng sao cho các vị trí tương đối của chúng khớp với các vị trí tương đối của các cột.

Giải thuật 4.3 BEA

Phần nhập: AA: ma trận ái lực thuộc tính

Phần xuất: CA: ma trận ái lực gom tụ

begin

{Bắt đầu; nên nhớ rằng AA là ma trận $n \times n$ }

$CA(\bullet, 1) \leftarrow AA(\bullet, 1);$

$CA(\bullet, 2) \leftarrow AA(\bullet, 2);$

$index \leftarrow 3;$

while $index \leq n$ **do** {chọn vị trí "tốt nhất" cho thuộc tính AA_{index} }

begin

for i **from** 1 **to** $index - 1$ **do**

Tính $cont(A_{i-1}, A_{index}, A_i)$;

Tính $cont(A_{index-1}, A_{index}, A_{index+1})$; {điều kiện biên}

$loc \leftarrow$ sắp đặt được cho bởi giá trị $cont$ lớn nhất;

for j from $index$ to loc step -1 do {di chuyển hai ma trận}

$CA(\bullet, j) \leftarrow CA(\bullet, j-1)$;

$CA(\bullet, loc) \leftarrow AA(\bullet, index)$;

$index \leftarrow index + 1$

end;

Sắp thứ tự các hàng theo thứ tự tương đối của các cột

end. {BEA}

Trong bước hai của giải thuật, chúng ta cần xác định sự góp phần của một thuộc tính vào số đo ái lực sẽ có ý nghĩa gì. Sự góp phần này có thể được suy diễn như sau. Nên nhớ rằng, số đo ái lực chung AM đã được định nghĩa trước như sau:

$$AM = \sum_{i=1}^n \sum_{j=1}^n aff(A_i, A_j) [aff(A_i, A_{j-1}) + aff(A_i, A_{j+1})]$$

và có thể được viết lại như sau:

$$\begin{aligned} AM &= \sum_{i=1}^n \sum_{j=1}^n [aff(A_i, A_j) aff(A_i, A_{j-1}) + aff(A_i, A_j) aff(A_i, A_{j+1})] \\ &= \sum_{j=1}^n \left[\sum_{i=1}^n aff(A_i, A_j) aff(A_i, A_{j-1}) + \sum_{i=1}^n aff(A_i, A_j) aff(A_i, A_{j+1}) \right] \end{aligned}$$

Chúng ta định nghĩa *liên kết (bond)* giữa hai thuộc tính A_x và A_y như sau:

$$bond(A_x, A_y) = \sum_{z=1}^n aff(A_z, A_x) aff(A_z, A_y)$$

Do đó, AM có thể được viết như sau:

$$AM = \sum_{j=1}^n [bond(A_j, A_{j-1}) + bond(A_j, A_{j+1})]$$

Bây giờ xét n thuộc tính sau đây:

$$\underbrace{A_1 A_2 \dots A_{i-1}}_{AM'} A_i A_j \underbrace{A_{j+1} \dots A_n}_{AM''}$$

Số đo ái lực chung cho các thuộc tính này có thể được viết như sau:

$$\begin{aligned} AM_{old} &= AM' + AM'' + bond(A_i, A_j) + bond(A_j, A_i) = \\ &= \sum_{i=1}^n [bond(A_i, A_{i-1}) + bond(A_i, A_{i+1})] + \\ &+ \sum_{i=i+2}^n [bond(A_i, A_{i-1}) + bond(A_i, A_{i+1})] + 2bond(A_i, A_j) \end{aligned}$$

Bây giờ, xét việc đặt thuộc tính mới A_k giữa thuộc tính A_i và A_j trong ma trận ái lực gom tụ. Số đo ái lực chung mới có thể được viết tương tự như sau:

$$\begin{aligned} AM_{new} &= AM' + AM'' + bond(A_i, A_k) + bond(A_k, A_j) + bond(A_k, A_j) \\ &\quad + bond(A_i, A_k) \\ &= AM' + AM'' + 2 bond(A_i, A_k) + 2 bond(A_k, A_j) \end{aligned}$$

Do đó, sự góp phần thực (*net contribution*) vào số đo ái lực chung khi đặt thuộc tính A_k giữa A_i và A_j là:

$$\begin{aligned} cont(A_i, A_k, A_j) &= AM_{new} - AM_{old} \\ &= 2 bond(A_i, A_k) + 2 bond(A_k, A_j) - 2 bond(A_i, A_j) \end{aligned}$$

Ví dụ 4.13 Xét ma trận AA được cho trong hình 4.7 và nghiên cứu sự góp phần khi di chuyển thuộc tính A_4 vào giữa A_1 và A_2 , được cho bởi công thức:

$$cont(A_1, A_4, A_2) = 2 bond(A_1, A_4) + 2 bond(A_4, A_2) - 2 bond(A_1, A_2)$$

Tính mỗi số hạng, chúng ta được:

$$bond(A_1, A_4) = 45 \times 0 + 0 \times 75 + 45 \times 3 + 0 \times 78 = 135$$

$$bond(A_4, A_2) = 11865$$

$$bond(A_1, A_2) = 225$$

Do đó:

$$cont(A_1, A_4, A_2) = 2 \times 135 + 2 \times 11865 - 2 \times 225 = 23550$$

Lưu ý rằng, việc tính toán liên kết giữa hai thuộc tính cần phải nhân các phần tử tương ứng của hai cột biểu diễn các thuộc tính này và lấy tổng theo hàng.

Giải thuật và bàn luận của chúng ta cho đến bây giờ đều tập trung vào các cột của ma trận ái lực thuộc tính. Chúng ta có thể làm những điều tương tự và thiết kế lại giải thuật để thực hiện trên các hàng. Vì ma trận AA là đối xứng nên cả hai cách tiếp cận này đều tạo ra cùng kết quả.

Một điểm nữa của giải thuật 4.3 là cải tiến hiệu quả, cột thứ hai cũng được cố định và được đặt kế cột thứ nhất trong bước bắt đầu. Điều này có thể chấp nhận được, bởi vì theo giải thuật, A_2 có thể được đặt ở bên trái hoặc ở bên phải của A_1 . Tuy nhiên, liên kết giữa hai thuộc tính này không phụ thuộc vào vị trí tương đối giữa chúng.

Cuối cùng, chúng ta chỉ ra vấn đề tính toán $cont$ tại các điểm cực. Nếu một thuộc tính A_k được xét để đặt vào bên trái của thuộc tính cực trái, thì một trong các phương trình liên kết được tính toán là giữa phần tử không tồn tại ở bên trái và A_k , tức là $bond(A_0, A_k)$. Do đó, chúng ta cần xét đến các điều kiện tác động đến định nghĩa của số đo ái lực chung AM , mà $CA(0, k) = 0$. Một điểm cực khác là nếu A_j là thuộc tính cực phải đã được đặt trong ma trận CA và chúng ta đang kiểm tra sự góp phần khi đặt thuộc tính A_k vào bên phải của A_j . Trong trường hợp này, cần phải tính $bond(k, k+1)$. Tuy nhiên, vì không có thuộc tính được đặt trong cột $k + 1$ của CA cho nên không xác định được số đo ái lực. Do đó, theo các điều kiện cực, giá trị $bond$ này cũng là 0.

Ví dụ 4.14 Xét việc gom nhóm các thuộc tính của quan hệ $DEPT$ và sử dụng ma trận ái lực AA của hình 4.7.

Theo bước bắt đầu, chúng ta chép cột 1 và cột 2 của ma trận AA vào ma trận CA (H.4.8a) và bắt đầu từ cột 3 (tức là thuộc tính A_3). Có ba nơi khác nhau để đặt cột 3: đặt bên trái của cột 1, cho thứ tự (3-1-2), đặt giữa cột 1 và cột 2, cho thứ tự (1-3-2) và đặt bên phải của cột 2, cho thứ tự (1-2-3). Lưu ý rằng, để tính toán sự góp phần của thứ tự cuối cùng, chúng ta phải tính $cont(A_2, A_3, A_4)$ mà không phải là $cont(A_1, A_2, A_3)$. Hơn nữa, trong ngữ cảnh này, A_4 nói đến vị trí chỉ mục thứ tư trong ma trận CA mà vị trí này là rỗng (H.4.8c), không phải là cột thuộc tính A_4 của ma trận AA . Chúng ta tính toán sự góp phần vào số đo ái lực chung của mỗi cách này.

Thứ tự (0-3-1)

$$cont(A_0, A_3, A_1) = 2 \, bond(A_0, A_3) + 2 \, bond(A_3, A_1) - 2 \, bond(A_0, A_1)$$

Chúng ta biết rằng:

$$bond(A_0, A_1) = bond(A_0, A_3) = 0$$

$$bond(A_3, A_1) = 45 \times 45 + 5 \times 0 + 53 \times 45 + 3 \times 0 = 4410$$

$$\text{Do đó: } cont(A_0, A_3, A_1) = 8820$$

Thứ tự (1-3-2)

$$cont(A_1, A_3, A_2) = 2 \, bond(A_1, A_3) + 2 \, bond(A_3, A_2) - 2 \, bond(A_1, A_2)$$

$$bond(A_1, A_3) = bond(A_3, A_1) = 4410$$

$$bond(A_3, A_2) = 890$$

$$bond(A_1, A_2) = 225$$

$$\text{Do đó: } cont(A_1, A_3, A_2) = 10150$$

Thứ tự (2-3-4)

$$\text{cont}(A_2, A_3, A_4) = 2 \text{ bond}(A_2, A_3) + 2 \text{ bond}(A_3, A_4) - 2 \text{ bond}(A_2, A_4)$$

$$\text{bond}(A_2, A_3) = 890$$

$$\text{bond}(A_3, A_4) = 0$$

$$\text{bond}(A_2, A_4) = 0$$

$$\text{Do đó: } \text{cont}(A_2, A_3, A_4) = 1780$$

Vì sự góp phần của thứ tự (1-3-2) là lớn nhất, chúng ta chọn đặt A_3 ở bên phải của A_1 (H.4.8b). Các tính toán tương tự cho A_4 cho thấy rằng A_4 nên được đặt ở bên phải của A_2 (H.4.8c).

Cuối cùng, các hàng được tổ chức theo cùng thứ tự như các cột và kết quả được chỉ ra trong hình 4.8d.

	A_1	A_2	A_3	A_4
A_1	45	0		
A_2	0	80		
A_3	45	5		
A_4	0	75		

a)

	A_1	A_3	A_2	A_4
A_1	45	45	0	
A_2	0	5	80	
A_3	45	53	5	
A_4	0	3	75	

b)

	A_1	A_3	A_2	A_4
A_1	45	45	0	0
A_2	0	5	80	75
A_3	45	53	5	3
A_4	0	3	75	78

c)

	A_1	A_3	A_2	A_4
A_1	45	45	0	0
A_3	45	53	5	3
A_2	0	5	80	75
A_4	0	3	75	78

d)

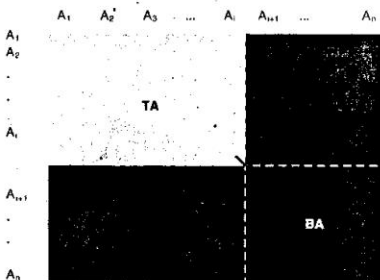
Hình 4.8 Tính toán ma trận ái lực gom tụ (CA)

Trong hình 4.8d, chúng ta thấy hình thành hai nhóm: một nhóm ở góc trái trên chứa các giá trị ái lực nhỏ hơn và một nhóm ở góc dưới phải chứa các giá trị ái lực lớn hơn. Việc gom nhóm này cho thấy các thuộc tính của quan hệ DEPT được phân chia như thế nào. Tuy nhiên nói chung, ranh giới trong việc phân chia này không rõ ràng. Khi ma trận CA có kích thước lớn, thông thường hình thành hơn hai nhóm và có nhiều cách phân chia. Do đó, cần phải tiếp cận vấn đề này một cách có hệ thống.

3- Giải thuật phân tách

Mục tiêu của việc phân tách là tìm các tập hợp thuộc tính được truy xuất đơn độc hoặc phần lớn, bởi các tập hợp ứng dụng riêng biệt. Ví dụ, nếu hai thuộc tính A_1 và A_2 chỉ được truy xuất bởi ứng dụng q_1 và các thuộc tính A_3 và A_4 được truy xuất bởi hai ứng dụng q_2 và q_3 thì dễ dàng quyết định chọn các mảnh như thế nào. Vấn đề đặt ra là tìm phương pháp mang tính giải thuật để xác định các nhóm này.

Xét ma trận thuộc tính gom tụ của hình 4.9. Nếu một điểm dọc theo đường chéo được cố định thì hai tập hợp thuộc tính được xác định. Một tập hợp $\{A_1, A_2, \dots, A_i\}$ ở góc trái trên và tập hợp thứ hai $\{A_{i+1}, \dots, A_n\}$ ở góc phải dưới của điểm này. Chúng ta gọi tập hợp đầu tiên là *tập hợp đỉnh (top)*, được ký hiệu là TA , và tập hợp thứ hai là *tập hợp đáy (bottom)*, được ký hiệu là BA .



Hình 4.9 Định vị điểm phân tách

Bây giờ, chúng ta đề cập đến tập hợp các ứng dụng $Q = \{q_1, q_2, \dots, q_r\}$ và định nghĩa tập hợp các ứng dụng chỉ truy xuất TA , chỉ truy xuất BA , hoặc cả hai. Các tập hợp này được định nghĩa như sau:

$$AQ(q_i) = \{A_j \mid use(q_i, A_j) = 1\}$$

$$TQ = \{q_i \mid AQ(q_i) \subseteq TA\}$$

$$BQ = \{q_i \mid AQ(q_i) \subseteq BA\}$$

$$OQ = Q - \{TQ \cup BQ\}$$

Công thức đầu tiên định nghĩa tập hợp các thuộc tính được truy xuất bởi ứng dụng q_i . TQ là tập hợp các ứng dụng chỉ truy xuất TA và BQ là tập hợp các ứng dụng chỉ truy xuất BA , và OQ là tập hợp các ứng dụng truy xuất cả hai TA và BA .

Ở đây có một vấn đề tối ưu hóa. Nếu một quan hệ có n thuộc tính thì có thể có $n - 1$ vị trí để đặt điểm phân chia dọc theo đường chéo của ma trận thuộc tính gom tụ cho quan hệ này. Vị trí tốt nhất cho việc phân chia này là vị trí mà nó tạo ra các tập hợp TQ và BQ sao cho toàn bộ các truy xuất *chỉ trên một mảnh* là lớn nhất, trong khi đó toàn bộ các truy xuất *trên cả hai mảnh* là nhỏ nhất. Do đó, chúng ta định nghĩa các công thức chi phí sau đây:

$$CQ = \sum_{q_i \in Q} \sum_{j \in S_j} ref_j(q_i) acc_j(q_i)$$

$$CTQ = \sum_{q_i \in TQ} \sum_{j \in S_j} ref_j(q_i) acc_j(q_i)$$

$$CBQ = \sum_{q_i \in BQ} \sum_{j \in S_j} ref_j(q_i) acc_j(q_i)$$

$$COQ = \sum_{q_i \in OQ} \sum_{j \in S_j} ref_j(q_i) acc_j(q_i)$$

Mỗi công thức ở trên sẽ tính tổng số truy xuất đến các thuộc tính bởi các ứng dụng trong các lớp tương ứng của chúng. Dựa trên các số đo này, vấn đề tối ưu hóa là xác định điểm x ($1 \leq x \leq n$) sao cho biểu thức:

$$z = CTQ \times CBQ - COQ^2$$

là lớn nhất. Đặc điểm quan trọng của biểu thức này là nó định nghĩa hai mảnh sao cho các giá trị của CTQ và CBQ càng gần bằng nhau càng tốt. Điều này cho phép cân bằng các tải xử lý khi các mảnh được phân bố tại các nơi khác nhau. Rõ ràng, giải thuật phân mảnh có độ phức tạp tuyến tính theo số thuộc tính của quan hệ, đó là $O(n)$.

Có hai điều phức tạp cần phải được đề cập đến. Điều thứ nhất là về việc phân chia. Lưu ý rằng, thủ tục này sẽ phân chia tập hợp các thuộc tính thành hai phần. Đối với các tập hợp thuộc tính lớn hơn, có thể cần phải phân chia thành n phần.

Có thể thiết kế phân chia n phần nhưng sẽ tốn kém về mặt tính toán. Dọc theo đường chéo của ma trận CA , cần phải xét các điểm phân chia 1, 2, ..., $n - 1$ và đối với mỗi giá trị này, cần phải

kiểm tra vị trí nào làm cho z lớn nhất. Do đó, độ phức tạp của giải thuật là $O(2^n)$. Tất nhiên, định nghĩa của z phải được sửa đổi cho các trường hợp có nhiều điểm phân chia. Một giải pháp khác là áp dụng giải thuật phân tách nhị phân (*binary partitioning algorithm*) một cách đệ quy cho mỗi mảnh có được trong lần lặp vòng trước. Chúng ta tính TQ , BQ và OQ cũng như các số đo truy xuất liên quan cho mỗi mảnh và tiếp tục phân mảnh chúng.

Điều phức tạp thứ hai liên quan đến vị trí của khối thuộc tính để tạo một mảnh. Cho đến lúc này, bàn luận của chúng ta đã giả sử điểm phân chia là duy nhất, đơn lẻ và chia ma trận CA thành một phần ở bên trái trên và phần thứ hai được tạo bởi các thuộc tính còn lại. Tuy nhiên, việc phân chia cũng có thể được hình thành ở giữa ma trận. Trong trường hợp này, chúng ta cần sửa đổi giải thuật một chút. Cột cực trái của ma trận CA được dịch chuyển để trở thành cột cực phải và hàng trên cùng được dịch chuyển thành hàng cuối cùng. Sau khi dịch chuyển, cần phải kiểm tra $n - 1$ vị trí thuộc đường chéo để tìm z lớn nhất. Ý tưởng của việc dịch chuyển là di chuyển khối thuộc tính tạo nhóm đến góc trái trên cùng của ma trận, mà nó có thể được xác định dễ dàng. Với việc thêm vào tác vụ dịch chuyển, độ phức tạp của giải thuật phân tách tăng theo hệ số n và là $O(n^2)$.

Giả sử thủ tục dịch chuyển đã có sẵn, được gọi là **SHIFT**, giải thuật phân chia được cho trong Giải thuật 4.4. Phần nhập của **PARTITION** là ma trận ái lực gom tụ CA và quan hệ R cần được phân mảnh. Phần xuất là tập hợp các mảnh $F_R = \{R_1, R_2\}$, với $R_i \subseteq \{A_1, A_2, \dots, A_n\}$ và $R_1 \cap R_2 = \emptyset$ các thuộc tính khóa của quan hệ R . Lưu ý rằng, đối với việc phân chia n -phần, thủ tục này cần được thực hiện lặp đi lặp lại, hoặc được thực hiện như là một thủ tục đệ quy.

Giải thuật 4.4 PARTITION

Phần nhập: CA : ma trận ái lực gom tụ; R : quan hệ

Phần xuất: F : tập hợp các mảnh

begin

{Xác định giá trị z cho cột thứ nhất}

{Các chỉ số trong các công thức chỉ phí cho biết điểm phân chia}

Tính CTQ_{n-1} ;

Tính CBQ_{n-1} ;

```

    Tính  $COQ_{n-1}$ ;
     $best \leftarrow CTQ_{n-1} * CBQ_{n-1} - (COQ_{n-1})^2$ ;
    repeat          {xác định cách phân mảnh tốt nhất}
        for  $i$  from  $n - 2$  to  $1$  by  $-1$  do
            begin
                Tính  $CTQ_i$ ;
                Tính  $CBQ_i$ ;
                Tính  $COQ_i$ ;
                 $best \leftarrow CTQ_i * CBQ_i - COQ_i^2$ ;
                if  $z > best$  then
                    begin
                         $best \leftarrow z$ ;
                        ghi nhận vị trí dịch chuyển
                    end
            end;
        end;
        Gọi SHIFT(CA)
    until không thể SHIFT được nữa;
    Xây dựng lại ma trận theo vị trí dịch chuyển;
     $R_1 \leftarrow \Pi_{TA}(R) \cup K$ ;    ( $K$  là tập hợp các thuộc tính khóa của  $R$ )
     $R_2 \leftarrow \Pi_{BA}(R) \cup K$ ;
     $F \leftarrow \{R_1, R_2\}$ 
end. {PARTITION}

```

Ví dụ 4.15 Khi áp dụng giải thuật PARTITION cho ma trận CA dùng cho quan hệ DEPT (ví dụ 4.14), kết quả là định nghĩa các mảnh $F_{DEPT} = \{DEPT_1, DEPT_2\}$, với $DEPT_1 = \{A_1, A_3\}$ và $DEPT_2 = \{A_1, A_2, A_4\}$. Do đó:

$DEPT_1 = \{DEPTNUM, MGRNM\}$
 $DEPT_2 = \{DEPTNUM, NAME, AREA\}$

Lưu ý rằng, trong ví dụ này, chúng ta đã thực hiện việc phân mảnh trên toàn bộ tập hợp các thuộc tính mà không chỉ trên các thuộc tính không khóa. Lý do là để cho ví dụ được đơn giản. Vì lý do này, chúng ta đưa DEPTNUM, là khóa của DEPT, vào DEPT₁ và DEPT₂.

4- Kiểm tra tính đúng đắn

Chúng ta cũng theo các lý lẽ tương tự của phân mảnh ngang để chứng minh rằng giải thuật PARTITION tạo ra phân mảnh dọc đúng đắn.

a- Điều kiện đầy đủ

Giải thuật PARTITION bảo đảm điều kiện đầy đủ bởi vì mỗi thuộc tính của quan hệ toàn cục được đưa vào một trong các mảnh. Đối với quan hệ R được phân mảnh dọc thành $F_R = \{R_1, R_2, \dots, R_r\}$, gọi $Attr(R_i)$ là tập hợp các thuộc tính của R_i , nếu điều kiện tập hợp các thuộc tính A của quan hệ R bao gồm:

$$A = \cup Attr(R_i), \quad i = 1, 2, \dots, r$$

thì điều kiện đầy đủ của phân mảnh dọc được bảo đảm.

b- Điều kiện tái tạo

Việc tái tạo quan hệ toàn cục ban đầu được thực hiện bởi phép kết. Do đó, đối với quan hệ R được phân mảnh dọc thành $F_R = \{R_1, R_2, \dots, R_r\}$ và các thuộc tính khóa là K thì:

$$R = \bowtie_K R_i, \quad \forall R_i \in F_R$$

Do đó, với điều kiện mỗi R_i là đầy đủ, phép kết sẽ tạo lại đúng R . Một điểm quan trọng là mỗi R_i nên chứa các thuộc tính khóa của R hoặc chứa các danh hiệu bộ (TID - Tuple ID) được gán bởi hệ thống.

b- Điều kiện tách biệt

Như chúng ta đã trình bày trước kia, tính tách biệt của các mảnh trong phân mảnh dọc là không quan trọng như trong phân mảnh ngang. Ở đây có hai trường hợp:

1- Sử dụng các TID, trong trường hợp này, các mảnh là tách biệt bởi vì các TID được nhân bản trong mỗi mảnh là các thực thể được quản lý và được gán bởi hệ thống và người sử dụng hoàn toàn không nhìn thấy được.

2- Các thuộc tính khóa được nhân bản trong mỗi mảnh, trong trường hợp này, chúng ta không thể khẳng định rằng chúng tách biệt nhau theo đúng nghĩa. Tuy nhiên, điều quan trọng để thấy rằng việc trùng lặp của các thuộc tính khóa đã được biết và được quản lý bởi hệ thống và không giống với việc trùng lặp bộ trong các mảnh được phân chia ngang. Nói cách khác, với điều kiện các mảnh tách biệt nhau ngoại trừ các thuộc tính khóa, thì điều kiện tách biệt được xem là thỏa mãn.

4.2.3 Phân mảnh hỗn hợp

Trong hầu hết các trường hợp, phân mảnh ngang hoặc phân mảnh dọc đơn giản của lược đồ CSDL không đủ thỏa mãn các yêu cầu của các ứng dụng. Trong trường hợp này, đi sau phân mảnh dọc là phân mảnh ngang, hoặc ngược lại, tạo ra phân mảnh có cấu trúc cây (H.3.4), cây này được gọi là **cây phân mảnh** (*fragmentation tree*) của quan hệ. Bởi vì có hai loại chiến lược phân chia được áp dụng xen kẽ nhau, do đó phân mảnh này được gọi là phân mảnh hỗn hợp (*hybrid fragmentation, mixed fragmentation, nested fragmentation*).

Số mức lồng nhau có thể lớn nhưng chắc chắn hữu hạn. Trong trường hợp phân mảnh ngang, chúng ta phải dừng lại khi mỗi mảnh chỉ bao gồm một bộ, trong khi đó đối với phân mảnh dọc, chúng ta phải dừng lại khi mỗi bộ chỉ có một thuộc tính. Tuy nhiên, các giới hạn này hoàn toàn không thực tế, bởi vì các mức lồng nhau trong hầu hết các ứng dụng thực tế đều không vượt quá 2. Điều này là do các quan hệ toàn cục đã được chuẩn hóa đều có bậc nhỏ và chúng ta không thể thực hiện quá nhiều phân mảnh dọc sẽ làm cho chi phí của các phép kết rất cao.

Chúng ta sẽ không bàn luận chi tiết về các quy tắc đúng đắn và các điều kiện cho phân mảnh hỗn hợp, bởi vì phân mảnh hỗn hợp phải tuân theo các quy tắc đúng đắn và các điều kiện cho phân mảnh ngang và phân mảnh dọc. Để tạo lại quan hệ toàn cục ban đầu trong trường hợp phân mảnh hỗn hợp, chúng ta bắt đầu từ các nút lá đến nút gốc của cây phân mảnh và thực hiện các phép kết (đối với phân mảnh dọc) và các phép hợp (đối với phân mảnh ngang). Phân mảnh hỗn hợp là đầy đủ nếu các mảnh lá và các mảnh trung gian là đầy đủ. Tương tự, tính tách biệt được bảo đảm nếu các mảnh lá và các mảnh trung gian là tách biệt.

4.3 THIẾT KẾ ĐỊNH VỊ CƠ SỞ DỮ LIỆU

Bài toán định vị dữ liệu đã được phân tích nhiều trong ngữ cảnh của "bài toán định vị tập tin". Nhiều mô hình phân tích dùng để định vị tối ưu cho các tập tin với nhiều giả thiết khác nhau và với nhiều mục tiêu khác nhau. Cách đơn giản nhất để áp dụng cách làm

này cho bài toán định vị mảnh là xem mỗi mảnh như là một tập tin riêng biệt. Tuy nhiên, cách tiếp cận này là không thuận tiện. Một số bài toán này đã không được giải quyết một cách thỏa đáng, bởi vì cách tiếp cận đúng đắn là đánh giá sự phân bố dữ liệu như thế nào để cho các ứng dụng được thực hiện "tối ưu". Tuy nhiên, điều này đòi hỏi việc tối ưu hóa tất cả các ứng dụng quan trọng đối với mỗi định vị dữ liệu có thể có, việc giải quyết tối ưu hóa truy vấn là bài toán con của định vị dữ liệu và rất khó.

1- Bài toán định vị

Giả sử có một tập hợp các mảnh $F = \{F_1, F_2, \dots, F_n\}$, một mạng bao gồm các nơi $S = \{S_1, S_2, \dots, S_m\}$ và một tập hợp các ứng dụng $Q = \{q_1, q_2, \dots, q_q\}$ chạy tại các nơi này. Bài toán định vị là tìm sự phân bố "tối ưu" của F tại S .

Một trong các vấn đề quan trọng cần được bàn luận là định nghĩa tính tối ưu (*optimality*). Tính tối ưu có thể được định nghĩa theo hai số đo:

Chi phí tối thiểu: Hàm chi phí bao gồm chi phí lưu trữ mỗi mảnh F_i tại nơi S_j , chi phí truy vấn F_i tại nơi S_j , chi phí cập nhật F_i tại tất cả các nơi lưu trữ mảnh này và chi phí truyền dữ liệu. Do đó, bài toán định vị là tìm một lược đồ định vị để cực tiểu hóa hàm chi phí này.

Hiệu suất: Chiến lược định vị được thiết kế để duy trì hiệu suất. Hai chiến lược nổi tiếng là cực tiểu hóa thời gian đáp ứng và cực đại hóa thông lượng hệ thống tại mỗi nơi.

Hầu hết các mô hình đã được đề nghị đều theo hai hướng tối ưu khác biệt này. Tuy nhiên, nếu chúng ta thật sự nghiên cứu sâu vào bài toán này, rõ ràng cần phải đưa hiệu suất và các hệ số chi phí vào số đo "tính tối ưu". Nói cách khác, chúng ta nên tìm một lược đồ định vị, ví dụ trả lời các truy vấn của người sử dụng trong thời gian nhanh nhất mà vẫn giữ được chi phí xử lý nhỏ nhất. Một điều tương tự có thể được thực hiện cho cực đại hóa thông lượng hệ thống. Do đó, tại sao các mô hình như vậy không được phát triển. Câu trả lời hoàn toàn đơn giản: tính phức tạp.

Xét một trường hợp rất đơn giản cho bài toán này. Cho F và S đã được định nghĩa ở trên. Bây giờ chúng ta chỉ xét một mảnh đơn là F_k . Chúng ta nêu ra một số giả thiết và định nghĩa để mô hình hóa bài toán định vị này.

1- Giả sử Q có thể được sửa đổi sao cho có thể xác định các truy vấn cập nhật (*update query*) và các truy vấn chỉ đọc (*read-only query*) và định nghĩa các tập hợp sau đây đối với mảnh đơn F_k :

$$T = \{t_1, t_2, \dots, t_m\}$$

với t_i là lưu lượng chỉ đọc (*read-only traffic*) tại nơi S_i đối với F_k , và:

$$U = \{u_1, u_2, \dots, u_m\}$$

với u_i là lưu lượng cập nhật (*update traffic*) tại nơi S_i đối với F_k .

2- Giả sử chi phí truyền thông giữa hai nơi bất kỳ S_i và S_j là cố định đối với một đơn vị truyền thông. Hơn nữa, giả sử rằng có khác nhau giữa các lần cập nhật và các lần đọc và có thể định nghĩa các tập hợp sau đây:

$$C(T) = \{c_{12}, c_{13}, \dots, c_{1m}, \dots, c_{m-1,m}\}$$

$$C'(T) = \{c'_{12}, c'_{13}, \dots, c'_{1m}, \dots, c'_{m-1,m}\}$$

với c_{ij} là chi phí truyền thông đơn vị đối với các yêu cầu đọc giữa các nơi S_i và S_j , và c'_{ij} là chi phí truyền thông đơn vị đối với các yêu cầu cập nhật giữa các nơi S_i và S_j .

3- Gọi chi phí lưu trữ mảnh ở S_i là d_i . Do đó, chúng ta có thể định nghĩa $D = \{d_1, d_2, \dots, d_m\}$ cho chi phí lưu trữ của mảnh F_k tại tất cả các nơi.

4- Giả sử không có các ràng buộc về khả năng cho các nơi hoặc các liên kết truyền thông.

Do đó, bài toán định vị có thể được mô tả như là bài toán cực tiểu hóa chi phí (*cost-minimization problem*) mà chúng ta tìm tập hợp $I \subseteq S$ xác định nơi sẽ chứa các bản sao của mảnh này. Trong phần sau, x_i là biến quyết định (*decision variable*) để đặt một mảnh sao cho:

$$x_i = \begin{cases} 1 & \text{nếu mảnh } F_k \text{ được đặt tại } S_j \\ 0 & \text{nếu mảnh } F_k \text{ không được đặt tại } S_j \end{cases}$$

Hàm mục tiêu của chi phí như sau:

$$\min \left[\sum_{i=1}^m \left(\sum_{j|S_j \in I} x_j u_j c'_{ij} + t_j \min_{j|S_j \in I} c_{ij} \right) + \sum_{j|S_j \in I} x_j d_j \right]$$

với $x_j = 1$ hoặc 0 .

Số hạng thứ hai của hàm mục tiêu dùng để tính toán chi phí tổng cộng của việc lưu trữ tất cả các bản nhân của mảnh này. Số hạng thứ nhất tương ứng với chi phí của việc truyền các cập nhật đến tất cả các nơi chứa các bản nhân của mảnh này, và tương ứng với chi phí của việc thực hiện các yêu cầu chỉ đọc tại nơi mà có chi phí truyền thông dữ liệu nhỏ nhất.

Điều này rất đơn giản và không thích hợp đối với thiết kế CSDL phân bố và đã được chứng minh là thuộc bài toán *NP*-đầy đủ (*NP-complete*). Tất nhiên, đối với các bài toán lớn (rất nhiều mảnh và rất nhiều nơi), để có được các giải pháp tối ưu thì chắc chắn không khả thi về mặt tính toán. Do đó, có nhiều nghiên cứu đáng kể nhằm tìm các phương pháp mang tính kinh nghiệm (*heuristics*) tốt để đạt được các giải pháp gần tối ưu (*suboptimal solution*).

Có một số lý do tại sao đặt vấn đề rất đơn giản như chúng ta đã bàn luận lại không thích hợp cho thiết kế CSDL phân bố. Các lý do này vốn có ở tất cả các mô hình định vị tập tin trước kia đối với các mạng máy tính.

- Chúng ta không thể xử lý các mảnh như là các tập tin riêng biệt mà mỗi lần chỉ định vị một tập tin một cách riêng biệt. Việc đặt một mảnh thường có ảnh hưởng đến các quyết định đặt các mảnh khác mà chúng được truy xuất chung với nhau, bởi vì các chi phí truy xuất đến các mảnh còn lại có thể bị thay đổi (ví dụ do phép kết phân bố). Do đó, cần phải quan tâm đến mối quan hệ giữa các mảnh.

- Truy xuất đến dữ liệu trong các ứng dụng được mô hình hóa rất đơn giản. Một yêu cầu của người sử dụng được phát sinh từ một nơi và tất cả dữ liệu dùng để trả lời yêu cầu này đều được truyền đến nơi này. Trong các hệ thống CSDL phân bố, truy xuất đến dữ liệu sẽ phức tạp hơn nhiều so với mô hình "truy xuất tập tin từ xa" này. Do đó, mối quan hệ giữa việc định vị và xử lý truy vấn cần phải được mô hình hóa một cách đúng đắn.

- Các mô hình này không xét đến chi phí duy trì tính toàn vẹn, việc đặt hai quan hệ có liên quan đến cùng một ràng buộc toàn vẹn tại hai nơi khác nhau có thể tốn kém chi phí.

- Tương tự, cần phải xét đến chi phí để duy trì các cơ chế điều khiển đồng thời.

Bởi vì việc định vị là chính yếu, do đó mối quan hệ của nó với các giải thuật được thực hiện cho các bài toán khác cần phải được biểu diễn trong mô hình định vị. Tuy nhiên, chính điều này làm cho chúng ta rất khó giải quyết các mô hình này. Để phân biệt bài toán định vị tập tin truyền thống với việc định vị mảnh trong thiết kế CSDL phân bố, chúng ta gọi bài toán thứ nhất là *bài toán định vị tập tin* (FAP - *File Allocation Problem*) và bài toán thứ hai là *bài toán định vị cơ sở dữ liệu* (DAP - *Database Allocation Problem*).

Không có các mô hình chung mang tính kinh nghiệm mà phần nhập là một tập hợp các mảnh và cho ra việc định vị gần tối ưu (*near-optimal allocation*) cho các loại ràng buộc được bàn luận ở đây. Các mô hình đã được phát triển cho đến nay có một số giả thiết đơn giản và có thể được áp dụng cho một số bài toán cụ thể nào đó. Do đó, thay vì trình bày một hoặc nhiều giải thuật định vị này, chúng ta giới thiệu một mô hình tương đối tổng quát và sau đó bàn luận một số giải thuật mang tính kinh nghiệm có thể có để giải quyết bài toán định vị.

2- Các yêu cầu về thông tin

Ở giai đoạn định vị, chúng ta cần có dữ liệu định lượng về CSDL, các ứng dụng chạy trên CSDL này, mạng truyền thông, các khả năng xử lý và các giới hạn lưu trữ của mỗi nơi trong mạng. Chúng ta sẽ bàn luận chi tiết về các yếu tố này.

a- Thông tin về cơ sở dữ liệu

Để thực hiện phân mảnh ngang, chúng ta đã định nghĩa hệ số chọn của các vị từ giao tối thiểu. Bây giờ, chúng ta cần mở rộng định nghĩa này cho các mảnh, và định nghĩa hệ số chọn của mảnh F_j đối với truy vấn q_i . Đây là số bộ của F_j cần được truy xuất để xử lý q_i . Giá trị này được ký hiệu là $sel_i(F_j)$.

Một thông tin cần thiết khác về các mảnh của CSDL là kích thước của chúng. Kích thước của mảnh F_j được cho bởi:

$$size(F_j) = card(F_j) \times length(F_j)$$

với $length(F_j)$ là chiều dài (số byte) của một bộ của mảnh F_j .

b- Thông tin về ứng dụng

Hầu hết các thông tin liên quan đến ứng dụng đã được thể hiện trong quá trình phân mảnh, nhưng cũng cần thêm một số thông tin cho mô hình định vị. Hai số đo quan trọng là số lần truy xuất đọc mảnh F_j trong quá trình thực hiện truy vấn q_i (ký hiệu là RR_{ij}), và tương tự cho các lần truy xuất cập nhật cần thiết cho việc thực hiện truy vấn (ký hiệu là UR_{ij}). Ví dụ, đếm số lần truy xuất khối cần thiết cho truy vấn.

$$u_{ij} = \begin{cases} 1 & \text{nếu truy vấn } q_i \text{ cập nhật mảnh } F_j \\ 0 & \text{nếu truy vấn } q_i \text{ không cập nhật mảnh } F_j \end{cases}$$

$$r_{ij} = \begin{cases} 1 & \text{nếu truy vấn } q_i \text{ đọc mảnh } F_j \\ 0 & \text{nếu truy vấn } q_i \text{ không đọc mảnh } F_j \end{cases}$$

Chúng ta định nghĩa vectơ O của các giá trị $o(i)$, với $o(i)$ là nơi gốc của truy vấn q_i . Cuối cùng, để định nghĩa ràng buộc thời gian đáp ứng, cần phải xác định thời gian đáp ứng tối đa cho phép.

c- Thông tin về nơi

Đối với mỗi nơi của mạng máy tính, chúng ta cần biết về khả năng lưu trữ và khả năng xử lý của nó. Rõ ràng, các giá trị này có thể được tính từ các hàm phức tạp hoặc bởi các đánh giá đơn giản. Chi phí đơn vị cho việc lưu trữ dữ liệu tại nơi S_k được ký hiệu là USC_k . Cũng cần xác định số đo chi phí LPC_k là chi phí xử lý một đơn vị công việc tại nơi S_k . Một đơn vị công việc nên giống với đơn vị của các số đo RR và UR .

d- Thông tin về mạng

Trong mô hình, chúng ta giả sử tồn tại một mạng đơn giản mà chi phí truyền thông được xác định theo gói dữ liệu (*frame of data*). Do đó, g_{ij} là chi phí truyền mỗi gói dữ liệu giữa nơi S_i và S_j . Để có

thể tính số thông điệp, chúng ta sử dụng f_{size} là kích thước (tính theo byte) của một gói. Nhiều mô hình mạng phức tạp hơn có quan tâm xem xét đến các khả năng kênh truyền, khoảng cách giữa các nơi, chi phí giao thức....

3- Mô hình định vị

Chúng ta bàn luận một mô hình định vị để cực tiểu hóa chi phí tổng cộng của việc xử lý và lưu trữ mà vẫn bảo đảm các ràng buộc thời gian đáp ứng. Chúng ta sử dụng mô hình có dạng:

min (chi phí tổng cộng)

với ràng buộc thời gian đáp ứng, ràng buộc lưu trữ và ràng buộc xử lý.

Chúng ta mở rộng các thành phần của mô hình này dựa trên các yêu cầu thông tin đã được đề cập trong phần 2. Biến quyết định là x_{ij} được định nghĩa như sau:

$$x_{ij} = \begin{cases} 1 & \text{nếu mảnh } F_i \text{ được đặt tại nơi } S_j \\ 0 & \text{nếu mảnh } F_i \text{ không được đặt tại nơi } S_j \end{cases}$$

α - Chi phí tổng cộng

Hàm chi phí tổng cộng có hai thành phần: xử lý truy vấn và lưu trữ. Do đó, hàm này được biểu diễn như sau:

$$TOC = \sum_{\forall q_i \in Q} QPC_i + \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} STC_{jk}$$

với QPC_i là chi phí xử lý truy vấn của ứng dụng q_i và STC_{jk} là chi phí lưu trữ mảnh F_j tại nơi S_k .

Trước tiên, chúng ta xét chi phí lưu trữ, nó đơn giản là:

$$STC_{jk} = USC_k \times size(F_j) \times x_{ij}$$

và hai tổng cộng để tính các chi phí lưu trữ tổng cộng tại tất cả các nơi đối với tất cả các mảnh.

Chi phí xử lý truy vấn thì khó mô tả hơn. Hầu hết các mô hình của bài toán định vị tập tin (FAP) chia chi phí này thành hai thành phần: chi phí xử lý chỉ đọc và chi phí xử lý cập nhật. Chúng ta chọn cách tiếp cận khác trong mô hình của chúng ta cho bài toán định vị CSDL (DAP) và chi phí này bao gồm chi phí xử lý (PC) và chi phí truyền thông (TC). Do đó, chi phí xử lý truy vấn (QPC) cho ứng dụng q_i là:

$$QPC_i = PC_i + TC_i$$

Theo các hướng dẫn đã được trình bày trong phần 1, thành phần xử lý *PC* bao gồm ba hệ số chi phí là chi phí truy xuất (*AC*), chi phí duy trì toàn vẹn (*IE*) và chi phí điều khiển đồng thời (*CC*):

$$PC_i = AC_i + IE_i + CC_i$$

Mô tả chi tiết cho mỗi hệ số chi phí này phụ thuộc vào các giải thuật để hoàn tất các công việc này. Tuy nhiên, để minh họa điều này, chúng ta nêu chi tiết cho *AC*:

$$AC_i = \sum_{\substack{q_j \in S \\ \forall F_j \in F}} (u_{ij} \times UR_{ij} + r_{ij} \times RR_{ij}) \times x_{jk} \times LPC_k$$

Hai số hạng đầu tiên trong công thức ở trên để tính số lần truy xuất của truy vấn q_i đến mảnh F_j . Lưu ý rằng $(UR_{ij} + RR_{ij})$ là tổng số lần truy xuất đọc và cập nhật. Chúng ta giả sử các chi phí cục bộ của việc xử lý chúng là giống nhau. Số hạng tổng cho biết tổng số lần truy xuất đối với tất cả các mảnh được tham chiếu bởi q_i . Số hạng tổng này nhân với LPC_k cho biết chi phí của truy xuất này tại nơi S_k . Chúng ta lại sử dụng x_{jk} để chỉ chọn các giá trị chi phí này đối với các nơi có đặt các mảnh.

Một vấn đề rất quan trọng cần được nêu ra ở đây. Hàm chi phí truy xuất đã giả sử việc xử lý một truy vấn bao gồm việc phân rã truy vấn này thành nhiều truy vấn con, mỗi truy vấn con thực hiện trên một mảnh được đặt tại một nơi, sau đó truyền các kết quả về nơi gốc của truy vấn. Như chúng ta đã bàn luận trước kia, đây là một cái nhìn rất đơn giản mà không quan tâm xem xét đến độ phức tạp của việc xử lý CSDL. Ví dụ, hàm chi phí không quan tâm đến chi phí thực hiện các phép kết (nếu cần) mà các phép kết này có thể được thực hiện theo nhiều cách khác nhau.

Hệ số chi phí duy trì toàn vẹn có thể được mô tả giống như thành phần xử lý, ngoại trừ chi phí xử lý cục bộ đơn vị (*unit local processing cost*) phải sửa đổi để phản ánh chi phí thực của việc duy trì toàn vẹn.

Hàm chi phí truyền thông có thể được thiết lập theo cách của hàm chi phí truy xuất. Tuy nhiên, chi phí truyền thông dữ liệu đối với các yêu cầu cập nhật và đối với yêu cầu đọc thì hoàn toàn khác nhau. Trong các truy vấn cập nhật, cần phải thực hiện tại tất cả các nơi có chứa bản nhân, còn trong các truy vấn đọc, chỉ cần truy xuất một trong các bản nhân. Hơn nữa, ở bước cuối của yêu cầu cập nhật,

không có truyền dữ liệu trở về nơi gốc ngoài một thông báo xác nhận, trong khi đó các truy vấn chỉ đọc phải truyền dữ liệu trả về.

Thành phần cập nhật của hàm truyền thông là:

$$TCU_k = \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} u_{ij} \times x_{jk} \times g_{o(i),k} + \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} u_{ij} \times x_{jk} \times g_{k,o(i)}$$

Số hạng thứ nhất tương ứng với việc gửi thông điệp cập nhật từ nơi gốc $o(i)$ của q_i đến tất cả các bản nhân cần được cập nhật. Số hạng thứ hai tương ứng với việc xác nhận.

Chi phí đọc có thể được tính như sau:

$$TCR_i = \sum_{\forall F_j \in F} \min(u_{ij} \times x_{jk} \times g_{o(i),k} + r_{ij} \times x_{jk} \times \frac{sel_i(F_j)}{f_{size}} \times g_{k,o(i)})$$

Số hạng thứ nhất trong TCR biểu diễn chi phí truyền yêu cầu đọc đến các nơi lưu trữ các bản nhân của các mảnh cần được truy xuất. Số hạng thứ hai tương ứng với việc truyền các kết quả từ các nơi này đến nơi gốc. Công thức này phát biểu rằng trong tất cả các nơi lưu trữ các bản nhân của cùng một mảnh thì chỉ chọn nơi có chi phí truyền thông tổng cộng nhỏ nhất để thực hiện tác vụ này.

Bây giờ, hàm chi phí truyền thông cho truy vấn q_i có thể được tính như sau:

$$TC_i = TCU_i + TCR_i$$

đó là hàm chi phí tổng cộng.

b- Các ràng buộc

Các hàm ràng buộc có thể được mô tả chi tiết một cách tương tự. Tuy nhiên, thay vì mô tả chi tiết các hàm này, chúng ta chỉ cho thấy dạng thức của chúng. Ràng buộc thời gian đáp ứng có thể được mô tả như sau:

thời gian thực hiện của $q_i \leq$ thời gian đáp ứng lớn nhất của q_i

với $\forall q_i \in Q$.

Tốt nhất là số đo chi phí trong hàm mục tiêu nên được mô tả theo thời gian, bởi vì nó mô tả ràng buộc thời gian thực hiện tương đối dễ hiểu.

Ràng buộc lưu trữ là:

$$\sum_{\forall F_j \in F} STC_{jk} \leq \text{khả năng lưu trữ tại nơi } S_k, \forall S_k \in S$$

trong khi đó, ràng buộc xử lý là:

$$\sum_{\forall q_i \in Q} \text{tải xử lý của } q_i \text{ tại nơi } S_k \leq \text{khả năng xử lý của } S_k, \forall S_k \in S$$

Đến đây hoàn tất phần trình bày của chúng ta về mô hình định vị. Mặc dù chúng ta đã không trình bày mô hình này một cách trọn vẹn, tính chính xác trong một số thuật ngữ cho thấy một vấn đề như thế sẽ được trình bày như thế nào. Ngoài khía cạnh này, chúng ta đã nêu ra các vấn đề quan trọng cần phải được xác định trong các mô hình định vị.

4- Các giải pháp

Trong phần trước, chúng ta đã trình bày một mô hình định vị chung, phức tạp nhiều hơn so với mô hình FAP đã được trình bày trong phần 1 ở trên. Bởi vì mô hình FAP thuộc NP–đầy đủ, người ta cho rằng, giải pháp của bài toán định vị CSDL (DAP) cũng thuộc NP–đầy đủ. Mặc dù chúng ta sẽ không chứng minh điều ước đoán này, nhưng điều này quả thực là đúng. Do đó, chúng ta cần phải tìm các phương pháp mang tính kinh nghiệm cho các lời giải gần tối ưu. Kiểm tra “lời giải tốt” trong trường hợp này là xét các kết quả của các giải thuật mang tính kinh nghiệm có gần với định vị tối ưu không.

Một số giải thuật mang tính kinh nghiệm đã được áp dụng để giải quyết các mô hình FAP và DAP. Sự tương ứng giữa FAP và bài toán định vị nhà máy (*plant location problem*) đã được khảo sát từ lâu và đã được nghiên cứu trong vận trù học (*operations research*). Các giải thuật mang tính kinh nghiệm được các nhà vận trù học phát triển và thường được sử dụng để giải quyết các bài toán FAP và DAP.

Có nhiều nỗ lực khác nhau nhằm để giảm độ phức tạp của bài toán. Một chiến lược giả sử rằng tất cả các cách phân chia có thể có đã được xác định cùng với các chi phí và lợi ích của chúng theo việc xử lý truy vấn. Do đó, bài toán được mô hình hóa là chọn cách phân chia tối ưu và xếp đặt đối với mỗi quan hệ. Một đơn giản khác thường được áp dụng là trước tiên bỏ qua sự nhân bản và tìm lời giải tối ưu không nhân bản. Sự nhân bản được giải quyết ở bước thứ hai bằng cách áp dụng giải thuật tham lam (*greedy algorithm*) bắt đầu từ lời giải không nhân bản như là lời giải khả thi ban đầu và sau đó cải tiến lời giải này. Tuy nhiên, đối với các giải thuật mang tính kinh nghiệm này, chúng ta không có đủ dữ liệu để xác định các kết quả gần với lời giải tối ưu.

BIẾN ĐỔI CÁC TRUY VẤN TOÀN CỤC THÀNH CÁC TRUY VẤN MẢNH

Như chúng ta đã thấy trong chương 3, một tác vụ truy xuất trong một ứng dụng có thể được biểu diễn như là một truy vấn tham chiếu đến các quan hệ toàn cục. DDBMS phải biến đổi truy vấn này thành các truy vấn đơn giản hơn mà chúng chỉ tham chiếu đến các mảnh. Chương này giải quyết phép biến đổi này.

Nói chung, có nhiều cách khác nhau để biến đổi một truy vấn trên các quan hệ toàn cục (được gọi là *truy vấn toàn cục* - *global query*) thành các truy vấn trên các mảnh (được gọi là *truy vấn mảnh* - *fragment query*). Các biến đổi khác nhau này tạo ra các truy vấn mảnh tương đương theo nghĩa chúng tạo ra cùng kết quả. Vì lý do này, chương này cũng giải quyết các *phép biến đổi tương đương* (*equivalence transformation*), nghĩa là các quy tắc có thể được áp dụng cho một truy vấn để viết truy vấn này thành một biểu thức tương đương.

Các quy tắc tương đương được sử dụng để đơn giản hóa biểu thức truy vấn (*query expression*): ví dụ xác định các biểu thức con chung (và loại bỏ), và các phép toán được "phân bổ" cho các mảnh. Tuy nhiên, điều nhấn mạnh trong chương này là *tính đầy đủ* (*completeness*) và *tính đúng đắn* (*correctness*) của phép biến đổi. Mục tiêu của chúng ta là đưa ra một tập hợp các quy tắc biến đổi tương đương và bao quát tất cả các khía cạnh có liên quan đến phép biến đổi truy vấn.

Phần 5.1 nhắc lại các kỹ thuật được sử dụng trong các hệ thống tập trung để biến đổi truy vấn. Trước tiên, chúng ta đưa ra cách biểu diễn truy vấn bằng cách sử dụng một *cây truy vấn* (*query tree*), sau đó chúng ta đưa ra một cách tiếp cận về ngữ nghĩa cho các phép biến đổi tương đương và cuối cùng cho thấy cách biến đổi một cây truy vấn thành một *đồ thị truy vấn* (*query graph*) để xác định các biểu thức con chung trong một truy vấn. Ở đây, chúng ta đưa ra điều nhắc nhở này bởi vì nó liên kết chặt chẽ với những gì đi theo sau. Hơn nữa, các khía

cạnh này trong các CSDL phân bố càng quan trọng hơn so với trong các CSDL tập trung và phép biến đổi truy vấn được đưa vào trong các môi trường phân bố.

Phần 5.2 cho thấy các truy vấn toàn cục được biến đổi như thế nào thành các truy vấn mảnh. Chúng ta nêu ra một ánh xạ chuẩn tắc (*canonical mapping*) và cho thấy ánh xạ chuẩn tắc là đúng đắn. Sau đó, chúng ta sử dụng các phép biến đổi tương đương của phần 5.1 để biến đổi **biểu thức chuẩn tắc** (*canonical expression*) của truy vấn. Cơ sở của các phép biến đổi này là áp dụng các phép toán đại số (*algebraic operation*), chẳng hạn phép chiếu và phép chọn, để làm giảm kích thước của các toán hạng của chúng càng nhiều càng tốt trên mỗi mảnh trước khi truyền dữ liệu giữa các nơi.

Trong phần 5.3, chúng ta giải quyết các truy vấn có liên quan đến cách đánh giá của việc gom nhóm từng phần (mệnh đề GROUP BY của lệnh SELECT) và các hàm kết hợp (*aggregate function*). Chúng ta mở rộng đại số quan hệ (*relational algebra*) để bao quát các vấn đề này, và sau đó nêu ra một số phép biến đổi tương đương áp dụng cho các phép toán mới. Cơ sở của các phép toán này là phân bố việc xử lý đến các mảnh.

Trong phần 5.4, chúng ta giải quyết các **truy vấn tham số** (*parametric query*), nghĩa là các truy vấn này có chứa các điều kiện chọn bao gồm các tham số mà giá trị sẽ được xác định ở thời gian thực hiện. Các truy vấn là các tác vụ truy xuất cơ bản tiêu biểu trong các ứng dụng tham số (*parametric application*) như được nêu ra trong chương 3. Chúng ta nêu ra các phép biến đổi tương đương có thể được sử dụng như thế nào để cải tiến tính hiệu quả của chúng.

5.1 CÁC PHÉP BIẾN ĐỔI TƯƠNG ĐƯƠNG DÙNG CHO CÁC TRUY VẤN

Một truy vấn quan hệ (*relational query*) có thể được biểu diễn bằng cách sử dụng các ngôn ngữ khác nhau. Trong sách này, chúng ta sử dụng đại số quan hệ và ngôn ngữ SQL cho mục đích này. Hầu hết các truy vấn SQL đều có thể được biến đổi thành các biểu thức đại số quan hệ tương đương và ngược lại. Do đó, bất kỳ các ngôn ngữ nêu trên có thể được sử dụng để biểu diễn ngữ nghĩa của truy vấn. Tuy nhiên, chúng ta có thể diễn dịch một biểu thức đại số quan hệ (*expression of relational algebra*) không chỉ là sự đặc tả ngữ nghĩa của

một truy vấn, mà còn là sự đặc tả của một *chuỗi các phép toán* (sequence of operations). Từ quan điểm này, hai biểu thức có cùng ngữ nghĩa có thể mô tả hai chuỗi phép toán khác nhau. Ví dụ:

$$\Pi_{\text{NAME, DEPTNUM}} \sigma_{\text{DEPTNUM} = 15} (\text{EMP})$$

và $\sigma_{\text{DEPTNUM} = 15} \Pi_{\text{NAME, DEPTNUM}} (\text{EMP})$

là các biểu thức tương đương nhưng định nghĩa hai chuỗi phép toán khác nhau.

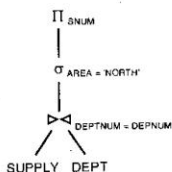
Trong chương này, vì chúng ta quan tâm đến thứ tự thực hiện của các phép toán, chúng ta bắt đầu từ việc sử dụng các biểu thức đại số quan hệ ban đầu và phân tích các phép biến đổi tương đương của chúng.

1- Cây toán tử của một truy vấn

Để biểu diễn thực tế hơn cho các truy vấn mà trong đó chúng ta dễ dàng theo dõi các thao tác trên biểu thức, chúng ta sử dụng các *cây toán tử* (operator tree). Xét truy vấn Q1 cho biết mã của các nhà cung cấp có đơn hàng cung cấp ở phía Bắc của công ty. Truy vấn Q1 tương ứng với biểu thức đại số quan hệ sau đây:

$$Q1: \Pi_{\text{SNUM}} \sigma_{\text{AREA} = \text{'NORTH'}} (\text{SUPPLY} \bowtie \sigma_{\text{DEPTNUM} = \text{DEPTNUM}} \text{DEPT})$$

Hình 5.1 cho thấy cây toán tử của truy vấn Q1. Lưu ý rằng các nút lá của cây là các quan hệ toàn cục và mỗi nút biểu diễn phép toán một ngôi (unary operation) hoặc phép toán hai ngôi (binary operation). Một cây xác định một thứ tự bộ phận mà trong đó các phép toán phải được thực hiện để cho ra kết quả của truy vấn. Do đó trong trường hợp này,



Hình 5.1

Cây toán tử của truy vấn Q1

phép kết được thực hiện trước tiên, đi sau là một phép chọn và một phép chiếu. Lưu ý rằng, phép chọn được thực hiện trên quan hệ toàn cục DEPT (bởi vì AREA là một thuộc tính của DEPT). Do đó, một thứ tự khác của các phép toán, tương ứng với một cây toán tử khác, có thể là phép chọn (được áp dụng cho nút lá DEPT), phép kết, phép chiếu. Đảo ngược thứ tự của các nút của một cây toán tử tương ứng với một phép biến đổi tương đương.

Cây toán tử của một biểu thức đại số quan hệ có thể được xem như một cây phân tích cú pháp (*parse tree*) của chính biểu thức đó, giả sử có văn phạm sau đây:

$$R \rightarrow \text{identifier}$$

$$R \rightarrow (R)$$

$$R \rightarrow \text{un_op } R$$

$$R \rightarrow R \text{ bin_op } R$$

$$\text{un_op} \rightarrow \sigma_F \mid \Pi_A$$

$$\text{bin_op} \rightarrow \times \mid \cup \mid - \mid \triangleright \triangleleft_F \mid \triangleright \triangleleft \mid \triangleright \triangleleft_F \mid \triangleright \triangleleft$$

Để dễ biểu diễn, chúng ta loại bỏ trong cây phân tích cú pháp các dấu ngoặc trong luật sinh thứ hai và chúng được thể hiện bởi cấu trúc của cây toán tử.

2- Các phép biến đổi tương đương dùng cho đại số quan hệ

Trước khi đưa ra các phép biến đổi tương đương, chúng ta cần phải nói rõ khái niệm về *sự tương đương* (*equivalence*). Chúng ta đã chấp nhận một định nghĩa của quan hệ mà không phụ thuộc vào thứ tự. Nghĩa là một định nghĩa mà trong đó thứ tự của các thuộc tính trong lược đồ thuộc tính là không quan trọng. Do đó, hai quan hệ là tương đương khi các bộ của chúng biểu diễn cùng ánh xạ từ các tên thuộc tính vào các giá trị, ngay cả khi thứ tự của các thuộc tính là khác nhau.

Bây giờ, chúng ta xét hai biểu thức đại số quan hệ E_1 và E_2 . Các tên quan hệ xuất hiện trong mỗi biểu thức tương ứng với các biến quan hệ (*relation variable*). Chúng ta nói rằng hai biểu thức là tương đương, viết là $E_1 \leftrightarrow E_2$ hoặc $E_1 = E_2$ nếu thay thế cùng các quan hệ cho các tên giống nhau trong hai biểu thức, chúng ta có các kết quả tương đương.

Các phép biến đổi tương đương có thể có được từ các biểu thức đơn giản, nghĩa là các biểu thức có hai hoặc ba quan hệ toán hạng. Các phép biến đổi này được phân loại theo loại toán tử. Cho U là phép toán đại số một ngôi và B là phép toán đại số hai ngôi. Chúng ta có:

Tính giao hoán (*commutativity*) của các phép toán một ngôi:

$$U_1 U_2 R \leftrightarrow U_2 U_1 R$$

Tính giao hoán của các toán hạng của các phép toán hai ngôi:

$$R B S \leftrightarrow S B R$$

Tính kết hợp (associativity) của các phép toán hai ngôi:

$$R B (S B T) \leftrightarrow (R B S) B T$$

Tính lũy đẳng (idempotence) của các phép toán một ngôi:

$$U R \leftrightarrow U, U_2 R$$

trong đó U_1, U_2 thuộc cùng loại phép toán.

Tính phân phối (distributivity) của các phép toán một ngôi đối với các phép toán hai ngôi:

$$U (R B S) \rightarrow U(R) B U(S)$$

Tính rút thừa số (factorization) của các phép toán một ngôi (phép biến đổi này ngược với tính phân phối):

$$U(R) B U(S) \rightarrow U(R B S)$$

Từ các tính chất trên của các phép toán đại số quan hệ, chúng ta có thể liệt kê các phép biến đổi tương đương sau đây, còn được gọi là **quy tắc tạo sinh** (generation rule) mà chúng cho thấy phần mô tả bên vế phải được suy diễn như thế nào từ phần mô tả bên vế trái. Các quy tắc tạo sinh không được thể hiện khi các phần mô tả của vế trái và vế phải giống nhau. Sau đây là một số phép biến đổi tương đương:

1- Hai phép chọn kề nhau

$$\sigma_{F_1}(\sigma_{F_2}(R)) \equiv \sigma_{F_2}(\sigma_{F_1}(R))$$

$$\sigma_{F_1}(\sigma_{F_2}(R)) \equiv \sigma_{F_1 \wedge F_2}(R)$$

2- Hai phép chiếu kề nhau

$$\Pi_{X_1}(\Pi_{X_2}(R)) \equiv \Pi_{X_1}(R) \quad \text{với } X_1 \subseteq X_2$$

3- Phép chiếu kề với phép chọn

$$\Pi_X(\sigma_F(R)) \leftrightarrow \sigma_F(\Pi_X(R)) \quad (\rightarrow \text{nếu } \text{Attr}(F) \subseteq X)$$

$$\Pi_X(\sigma_F(R)) \equiv \Pi_X(\sigma_F(\Pi_{X \cup \text{Attr}(F)}(R)))$$

4- Phép hợp kề với phép chọn

$$\sigma_F(R \cup S) \equiv \sigma_F(R) \cup \sigma_F(S)$$

5- Phép kết kề với phép chọn

$$\sigma_F(R \bowtie_{F_1} S) \leftrightarrow \sigma_F(R) \bowtie_{F_1} S \quad (\rightarrow \text{nếu } \text{Attr}(F) \subseteq \text{Attr}(R))$$

$$\sigma_{F_1 \wedge F_2}(R \bowtie_{F_3} S) \leftrightarrow \sigma_{F_1}(R) \bowtie_{F_3} \sigma_{F_2}(S)$$

$$(\rightarrow \text{nếu } \text{Attr}(F_1) \subseteq \text{Attr}(R) \text{ và } \text{Attr}(F_2) \subseteq \text{Attr}(S))$$

$$\sigma_F(R \bowtie_{F_3} S) \leftrightarrow \sigma_{F_2}(\sigma_{F_1}(R) \bowtie_{F_3} S)$$

$$(\rightarrow \text{nếu } F = F_1 \wedge F_2 \text{ và}$$

$$\text{Attr}(F_1) \subseteq \text{Attr}(R) \text{ và } \text{Attr}(F_2) \subseteq \text{Attr}(R) \cup \text{Attr}(S))$$

6- Phép nửa kết kề với phép chọn

$$\sigma_F(R \bowtie_{F_1} S) \leftrightarrow \sigma_F(R) \bowtie_{F_1} S \quad (\rightarrow \text{nếu } \text{Attr}(F) \subseteq \text{Attr}(R))$$

7- Phép tích Descartes kể với phép chọn

$$\sigma_F(R \times S) \leftrightarrow \sigma_F(R) \times S \quad (\rightarrow \text{nếu } \text{Attr}(F) \subseteq \text{Attr}(R))$$

$$\sigma_{F_1 \wedge F_2}(R \times S) \leftrightarrow \sigma_{F_1}(R) \times \sigma_{F_2}(S)$$

$$(\rightarrow \text{nếu } \text{Attr}(F_1) \subseteq \text{Attr}(R) \text{ và } \text{Attr}(F_2) \subseteq \text{Attr}(S))$$

$$\sigma_F(R \times S) \leftrightarrow \sigma_{F_2}(\sigma_{F_1}(R) \times S) \quad (\rightarrow \text{nếu } F = F_1 \wedge F_2 \text{ và } \text{Attr}(F_1) \subseteq \text{Attr}(R) \text{ và } \text{Attr}(F_2) \subseteq \text{Attr}(R) \cup \text{Attr}(S))$$

8- Phép giao kể với phép chọn

$$\sigma_F(R \cap S) \equiv \sigma_F(R) \cap \sigma_F(S)$$

$$\sigma_{F_1 \vee F_2}(R \cap S) \leftrightarrow \sigma_{F_1}(R) \cap \sigma_{F_2}(S)$$

$$(\rightarrow \text{nếu } \text{Attr}(F_1) \subseteq \text{Attr}(R) \text{ và } \text{Attr}(F_2) \subseteq \text{Attr}(S))$$

9- Phép hiệu kể với phép chọn

$$\sigma_F(R - S) \equiv \sigma_F(R) - \sigma_F(S)$$

10- Phép chia kể với phép chọn

$$\sigma_F(R \div S) \leftrightarrow \sigma_F(R) \div S \quad (\rightarrow \text{nếu } \text{Attr}(F) \subseteq \text{Attr}(R))$$

11- Phép hợp kể với phép chiếu

$$\Pi_X(R \cup S) \equiv \Pi_X(R) \cup \Pi_X(S)$$

12- Phép kết kể với phép chiếu

$$\Pi_X(R \bowtie_F S) \leftrightarrow \Pi_X(R) \bowtie_F S \quad (\rightarrow \text{nếu thỏa hai điều kiện:}$$

$$1. \text{Attr}(F_R) \subseteq X; \quad 2. X \subseteq \text{Attr}(R))$$

$$\Pi_{X_1 \cup X_2}(R \bowtie_F S) \leftrightarrow \Pi_{X_1}(R) \bowtie_F \Pi_{X_2}(S)$$

$$(\rightarrow \text{nếu thỏa ba điều kiện:}$$

$$1. \text{Attr}(F) \subseteq X_1 \cup X_2; \quad 2. X_1 \subseteq \text{Attr}(R); \quad 3. X_2 \subseteq \text{Attr}(S))$$

13- Phép nửa kết kể với phép chiếu

$$\Pi_{X_1 \cup X_2}(R \ltimes_F S) \leftrightarrow \Pi_{X_1}(R) \ltimes_F \Pi_{X_2}(S)$$

$$(\rightarrow \text{nếu thỏa ba điều kiện:}$$

$$1. \text{Attr}(F) \subseteq X_1 \cup X_2; \quad 2. X_1 \subseteq \text{Attr}(R); \quad 3. X_2 \subseteq \text{Attr}(S) \cap \text{Attr}(F))$$

14- Phép tích Descartes kể với phép chiếu

$$\Pi_{X_1 \cup X_2}(R \times S) \leftrightarrow \Pi_{X_1}(R) \times \Pi_{X_2}(S)$$

$$(\rightarrow \text{nếu thỏa hai điều kiện:}$$

$$1. X_1 \subseteq \text{Attr}(R); \quad 2. X_2 \subseteq \text{Attr}(S))$$

15- Các phép hợp, phép giao, phép tích Descartes và phép kết

$$(R \cup S) \cup T \equiv (R \cup T) \cup S$$

$$(R \cap S) \cap T \equiv (R \cap T) \cap S$$

$$(R \times S) \times T \equiv (R \times T) \times S$$

$$(R \bowtie_{F_1} S) \bowtie_{F_2} T \leftrightarrow (R \bowtie_{F_2} T) \bowtie_{F_1} S$$

$$(\rightarrow \text{ nếu } \text{Attr}(F2) \subseteq \text{Attr}(R) \cup \text{Attr}(T))$$

$$(\leftarrow \text{ nếu } \text{Attr}(F1) \subseteq \text{Attr}(R) \cup \text{Attr}(S))$$

16- Các phép hiệu và phép chia

$$R - S - T \equiv R - T - S$$

$$R \div S \div T \equiv R \div T \div S$$

17- Phép kết đi sau phép hợp

$$(R \cup S) \bowtie_F T \equiv (R \bowtie_F T) \cup (S \bowtie_F T)$$

18- Phép giao đi sau phép hợp

$$(R \cup S) \cap T \equiv (R \cap T) \cup (S \cap T)$$

19- Phép hiệu đi sau phép hợp

$$(R \cup S) - T \equiv (R - T) \cup (S - T)$$

20- Phép hiệu đi trước phép hợp

$$R - (S \cup T) \equiv R - S - T \equiv R - T - S \equiv (R - S) \cap (R - T)$$

21- $(R \bowtie_F S) \cap T \equiv (R \cap T) \bowtie_F S \equiv (R \bowtie_F S) \cap (S \bowtie_F T)$

$$\equiv (T \bowtie_F S) \cap R$$

$$(R \bowtie_F S) - T \equiv (R - T) \bowtie_F S \equiv (R \bowtie_F S) - (T \bowtie_F S)$$

22- Phép giao

$$(R \cap S) \cup T \equiv (R \cup T) \cap (S \cup T)$$

$$(R \cap S) \bowtie_F T \equiv (R \bowtie_F T) \cap (S \bowtie_F T)$$

$$(R \cap S) - T \equiv (R - T) \cap (S - T)$$

$$R - (S \cap T) \equiv (R - S) \cup (R - T)$$

$$(R \cap S) \div T \equiv (R \div T) \cap (S \div T)$$

23- Phép hiệu

$$(R - S) \bowtie_F T \equiv (R \bowtie_F T) - (S \bowtie_F T)$$

$$(R - S) \cap T \equiv (R \cap T) - (S \cap T)$$

Trong các CSDL không phân bố, chúng ta có các tiêu chuẩn tổng quát cho việc áp dụng các phép biến đổi tương đương để đơn giản hóa việc thực hiện các truy vấn.

Tiêu chuẩn 1: Sử dụng tính lũy đẳng của phép chọn và phép chiếu để tạo ra các phép chọn và các phép chiếu thích hợp đối với mỗi quan hệ toán hạng.

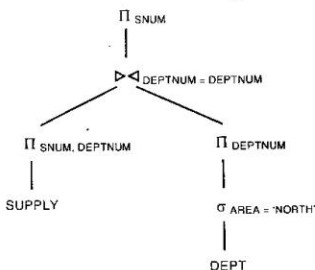
Tiêu chuẩn 2: Đẩy các phép chọn và các phép chiếu xuống phía dưới cây nếu có thể được.

Các tiêu chuẩn này xuất phát từ sự khảo sát các phép toán hai ngôi và đặc biệt là các phép kết, đó là các phép toán tốn kém nhất trong các hệ thống CSDL; do đó cần phải giảm kích thước của các toán hạng của các phép toán hai ngôi trước khi thực hiện chúng. Trong các CSDL phân bố, các tiêu chuẩn này còn quan trọng hơn: các phép toán hai ngôi đòi hỏi so sánh các toán hạng mà chúng có thể được đặt tại các nơi khác nhau. Truyền dữ liệu là một trong các thành phần chính của các chi phí và các trì hoãn gắn liền với việc thực hiện truy vấn. Do đó, việc làm giảm kích thước của các toán hạng của các phép toán hai ngôi là một điều quan tâm chính.

Hình 5.2 cho thấy cây toán tử của truy vấn Q_1 đã bị thay đổi. Trong đó các biến đổi sau đây đã được áp dụng:

1- Phép chọn được phân phối đối với phép kết. Do đó, phép chọn được áp dụng trực tiếp cho quan hệ DEPT.

2- Hai phép chiếu mới được tạo ra và được phân phối đối với phép kết.



Hình 5.2 Cây toán tử của truy vấn Q_1 đã bị thay đổi

Lưu ý rằng, tiêu chuẩn 1 (sử dụng tính lũy đẳng) đã được áp dụng cho phép chiếu trước khi phân phối nó. Phép chọn đã được di chuyển trực tiếp đến quan hệ DEPT.

Khi một phép chiếu di chuyển xuống dưới qua một phép kết thì các thuộc tính trong điều kiện kết phải có trong tập thuộc tính chiếu của phép chiếu này.

3- Đồ thị toán tử và xác định các biểu thức con chung

Một vấn đề quan trọng trong việc áp dụng các phép biến đổi cho một biểu thức truy vấn là tìm ra các *biểu thức con chung* (*common subexpression*) của nó, nghĩa là các biểu thức con xuất hiện nhiều lần trong truy vấn. Rõ ràng, điều này sẽ tiết kiệm thời gian thực hiện của truy vấn nếu các biểu thức con chung chỉ được định trị một lần. Một phương pháp để nhận biết chúng bao gồm việc biến đổi cây toán tử tương ứng thành một đồ thị toán tử bằng cách trước tiên gộp các nút lá giống nhau của cây (nghĩa là các quan hệ toán hạng giống nhau), và sau đó gộp các nút trung gian khác của cây tương ứng với cùng các phép toán và có cùng các toán hạng.

Chúng ta dùng một ví dụ để trình bày phương pháp này. Xét truy vấn Q2: hãy cho biết các tên của các nhân viên làm việc trong một phòng ban có mã người quản lý là 373 nhưng họ không được lãnh lương nhiều hơn \$35.000. Một biểu thức của truy vấn này là:

Q2: $\Pi_{EMP.NAME} ((EMP \triangleright \triangleleft_{DEPTNUM = DEPTNUM \wedge MGRNUM = 373} DEPT) -$
 $(\sigma_{SAL > 35000} EMP \triangleright \triangleleft_{DEPTNUM = DEPTNUM \wedge MGRNUM = 373} DEPT))$

Cây toán tử tương ứng được chỉ ra trong hình 5.3a. Chúng ta bắt đầu bằng cách gộp các nút lá tương ứng với các quan hệ EMP và DEPT. Sau đó, chúng ta đặt thừa số là phép chọn trên SAL đối với phép kết (để làm điều này, chúng ta di chuyển phép chọn lên phía trên). Bây giờ, chúng ta có thể gộp các nút tương ứng với phép chọn trên MGRNUM và cuối cùng các nút tương ứng với phép kết. Chúng ta có được cây toán tử trong hình 5.3b. Chúng ta nhận được biểu thức con sau đây:

$EMP \triangleright \triangleleft_{DEPTNUM = DEPTNUM \wedge MGRNUM = 373} DEPT$

Mỗi khi các biểu thức con đã được xác định, chúng ta có thể sử dụng các đặc tính sau đây để đơn giản hóa một cây toán tử:

$$R \triangleright \triangleleft R \leftrightarrow R$$

$$R \cup R \leftrightarrow R$$

$$R - R \leftrightarrow \emptyset$$

$$R \triangleright \triangleleft \sigma_F R \leftrightarrow \sigma_F R$$

$$R \cup \sigma_F R \leftrightarrow R$$

$$R - \sigma_F R \leftrightarrow \sigma_{\text{NOT } F} R$$

$$(\sigma_{F_1} R) \triangleright \triangleleft (\sigma_{F_2} R) \leftrightarrow \sigma_{F_1 \wedge F_2} R$$

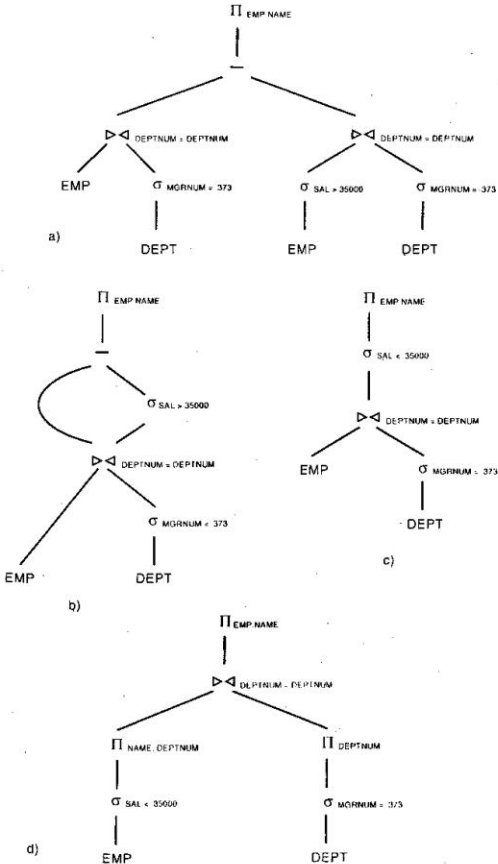
$$(\sigma_{F_1} R) \cup (\sigma_{F_2} R) \leftrightarrow \sigma_{F_1 \vee F_2} R$$

$$(\sigma_{F_1} R) - (\sigma_{F_2} R) \leftrightarrow \sigma_{F_1 \wedge \neg F_2} R$$

Chúng ta có thể áp dụng đặc tính thứ 6 trong danh sách trên vào ví dụ này, rút gọn cây toán tử thành cây trong hình 5.3c. Cuối cùng, chúng ta có thể áp dụng tính lũy đẳng cho phép chiếu và đẩy các phép chọn và các phép chiếu xuống phía các nút lá của cây, tạo ra cây toán tử trong hình 5.3d. Lưu ý rằng, cây toán tử mới này tương ứng với một truy vấn mà ban đầu một người lập trình chuyên nghiệp có thể đưa ra. Do đó, các phép biến đổi đã được giới thiệu trong phần này nhằm mục đích đơn giản hóa việc mô tả một truy vấn mà nó không được viết một cách trực tiếp ở dạng tốt nhất.

5.2 BIẾN ĐỔI CÁC TRUY VẤN TOÀN CỤC THÀNH CÁC TRUY VẤN MẢNH

Bây giờ chúng ta trở về các khía cạnh riêng biệt của các CSDL phân bố. Trong chương 3, chúng ta đã giới thiệu sự phân mảnh như là một đặc điểm thích hợp của các CSDL phân bố. Trong phần này chúng ta nêu ra một phép biến đổi chuẩn mà nó biến đổi một biểu thức đại số trên lược đồ toàn cục thành một biểu thức đại số trên lược đồ phân mảnh. Sau đó chúng ta nêu ra các đặc tính thích hợp của các biểu thức này.



Hình 5.3 Các bước đơn giản hóa của truy vấn Q2

1- Biểu thức chuẩn tắc của một truy vấn mảnh

Cho một biểu thức đại số trên lược đồ toàn cục, **biểu thức chuẩn tắc** (*canonical expression*) của nó có được bằng cách thay thế mỗi tên quan hệ toàn cục xuất hiện trong nó bởi biểu thức đại số tái tạo các quan hệ toàn cục từ các mảnh. Bằng cách tương tự, chúng ta biến đổi một cây toán tử trên lược đồ toàn cục thành một cây toán tử trên lược đồ phân mảnh bằng cách thay thế các nút lá của cây đầu tiên bởi các biểu thức tương ứng của phép biến đổi ngược của lược đồ phân mảnh (tức là các biểu thức tái tạo tương ứng).

Nên nhớ rằng, phép biến đổi ngược của lược đồ phân mảnh đưa ra quy tắc cho việc tái tạo một lược đồ toàn cục từ các mảnh của nó. Do đó, biểu thức mới (trên lược đồ phân mảnh) sẽ cho ra cùng kết quả so với biểu thức cũ (trên lược đồ toàn cục). Điều quan trọng là bây giờ các nút lá của cây toán tử của biểu thức chuẩn tắc là các mảnh thay vì là các quan hệ toàn cục.

Hình 5.4a cho thấy phép biến đổi của cây toán tử của truy vấn Q1 được biểu diễn trong hình 5.2 thành cây toán tử của biểu thức chuẩn tắc của Q1. Trong hình 5.4a, hai cây con được khoanh tròn thay thế cho các quan hệ toàn cục SUPPLY và DEPT của hình 5.2.

Một biểu thức chuẩn tắc tương ứng với cách thực hiện bảo toàn một truy vấn, trước tiên bao gồm việc tập hợp các mảnh của tất cả các quan hệ toàn cục thành các quan hệ tạm thời, và sau đó áp dụng truy vấn toàn cục trên các quan hệ tạm thời này. Rõ ràng, điều này không phải là một cách tiếp cận hiệu quả. Do đó, biểu thức chuẩn tắc cũng là trường hợp xấu đối với một truy vấn cho trước. Nhưng một biểu thức chuẩn tắc là một biểu thức đại số, và chúng ta có thể áp dụng bất kỳ một phép biến đổi tương đương nào cho nó. Đặc biệt, chúng ta sẽ sử dụng tính phân phối của phép chọn và phép chiếu đối với phép hợp và phép kết để phân phối việc xử lý đến các mảnh. Do đó, tiêu chuẩn 1 và 2 của phần 5.1 mục 2 càng thích hợp hơn trong ngữ cảnh của các truy vấn trên lược đồ phân mảnh. Hình 5.4b cho thấy ứng dụng của hai tiêu chuẩn này cho cây toán tử của hình 5.4a.

Trong chương 3, khi giới thiệu các mảnh ngang, chúng ta cũng đã giới thiệu vị từ định tính của mảnh, nghĩa là một vị từ biểu diễn một đặc tính của tất cả các bộ trong mảnh. Bây giờ, chúng ta sử dụng vị từ định tính trong việc đơn giản hóa các truy vấn phân bố.

2- Đại số quan hệ định tính

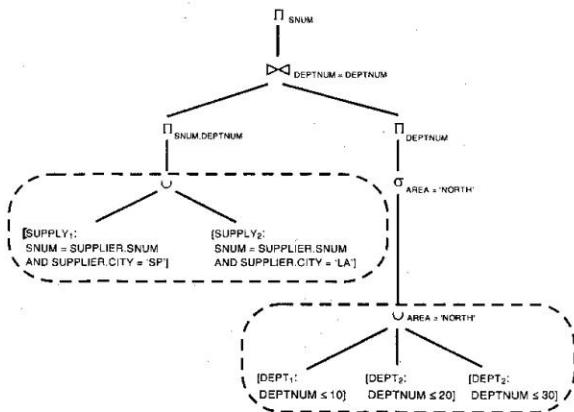
Một quan hệ định tính là một quan hệ được mở rộng bởi một vị từ định tính. Vị từ định tính có thể được xem là một đặc tính ngữ nghĩa (*intensional property*) của tất cả các bộ của quan hệ. Chúng ta ký hiệu một **quan hệ định tính** (*qualified relation*) là một cặp $[R: q_R]$, trong đó R là một quan hệ được gọi là *thân* (*body*) của quan hệ định tính và q_R là một vị từ được gọi là *vị từ định tính* (*qualification*) của quan hệ định tính. Các mảnh ngang là các ví dụ tiêu biểu của các quan hệ định tính, trong đó vị từ định tính tương ứng với vị từ phân mảnh.

Vị từ định tính của một quan hệ không cần phải được định trị trên các bộ của quan hệ. Tuy nhiên, nếu vị từ định tính được định trị thì vị từ định tính phải đúng. Cụ thể, một vị từ định tính không thể được định trị khi chúng ta không định trị được một số thuộc tính trong biểu thức của vị từ định tính.

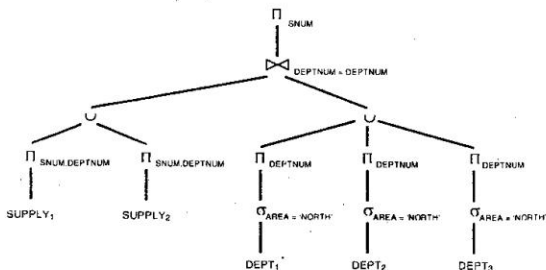
Đại số các quan hệ định tính là sự mở rộng của đại số quan hệ, trong đó sử dụng các quan hệ định tính như là các toán hạng. Rõ ràng, đại số này đòi hỏi việc thao tác trên các điều kiện định tính cũng như trên các quan hệ. Các quy tắc sau đây xác định kết quả của việc áp dụng các phép toán đại số quan hệ cho các quan hệ định tính.

Quy tắc 1 $\sigma_F[R: q_R] \Rightarrow [\sigma_F R: F \text{ AND } q_R]$

Quy tắc này phát biểu rằng việc áp dụng một phép chọn σ_F cho một quan hệ định tính $[R: q_R]$ tạo ra một quan hệ định tính có quan hệ $\sigma_F R$ là thân của nó và vị từ $F \text{ AND } q_R$ là vị từ định tính của nó. Việc mở rộng vị từ định tính $F \text{ AND } q_R$ sau khi chọn phản ánh rằng F và q_R đều được thỏa mãn trên tất cả các bộ được chọn.



a) Dạng chuẩn tắc của truy vấn Q1



b) Đẩy các phép chọn và phép chiếu xuống phía dưới cây toán tử

Hình 5.4 Các phép biến đổi của cây toán tử của truy vấn Q1

Quy tắc 2 $\Pi_A[R: q_R] \Rightarrow [\Pi_A R: q_R]$

Vị từ định tính của kết quả của một phép chiếu vẫn không thay đổi, ngay cả khi phép chiếu loại bỏ một số thuộc tính dùng để định trị vị từ định tính. Lý do là vì các vị từ định tính mang thông tin ngữ nghĩa (*intensional information*). Do đó, điều này vẫn đúng khi loại bỏ các thuộc tính được dùng để biểu diễn vị từ định tính, mà không loại bỏ chính vị từ định tính.

Quy tắc 3 $[R: q_R] \times [S: q_S] \Rightarrow [R \times S: q_R \text{ AND } q_S]$

Việc mở rộng vị từ định tính $q_R \text{ AND } q_S$ là trực quan. Lưu ý rằng hai vị từ định tính áp dụng riêng cho các thuộc tính của $R \times S$.

Quy tắc 4 $[R: q_R] - [S: q_S] \Rightarrow [R - S: q_R]$

Việc mở rộng phép hiệu là hơi trực quan. Khi chúng ta loại bỏ các bộ của quan hệ S có trong quan hệ R , vị từ định tính của kết quả vẫn giống với vị từ định tính của R . Điều không đúng khi thay thế nó thành $q_R \text{ AND NOT } q_S$, bởi vì có thể có các bộ của R không thỏa mãn $q_R \text{ AND NOT } q_S$, nhưng vẫn có trong kết quả bởi vì S không chứa các bộ này.

Tuy nhiên, vấn đề sau đây được đặt ra với quy tắc 4:

Xét phép giao của đại số quan hệ truyền thống, được định nghĩa là $R \cap S = R - (R - S)$. Từ định nghĩa, dễ dàng thấy rằng phép giao có tính giao hoán, nghĩa là:

$$R \cap S = S \cap R$$

Nếu chúng ta áp dụng định nghĩa của đại số mở rộng, chúng ta đi đến một kết quả đáng ngạc nhiên:

$$\begin{aligned} [R: q_R] \cap [S: q_S] &\Rightarrow [R: q_R] - ([R: q_R] - [S: q_S]) \Rightarrow \\ [R: q_R] - [R - S: q_R] &\Rightarrow [R - (R - S): q_R] \Rightarrow [R \cap S: q_R] \end{aligned} \quad (5.4a)$$

$$\begin{aligned} [S: q_S] \cap [R: q_R] &\Rightarrow [S: q_S] - ([S: q_S] - [R: q_R]) \Rightarrow \\ [S: q_S] - [S - R: q_S] &\Rightarrow [S - (S - R): q_S] \Rightarrow [S \cap R: q_S] \end{aligned} \quad (5.4b)$$

Trên thực tế, kết quả mà chúng ta muốn tìm là

$$[R \cap S: q_R \text{ AND } q_S]$$

bởi vì các bộ của phép giao là những bộ thuộc cả hai quan hệ thỏa mãn cả hai q_R và q_S . Lưu ý rằng, $q_R \text{ AND } q_S$ bao hàm cả hai q_R và q_S . Do đó, các kết quả (5.4a) và (5.4b) là không sai, nhưng dĩ nhiên một số thông tin bị mất.

Quy tắc 5 $[R: q_R] \cup [S: q_S] \Rightarrow [R \cup S: q_R \text{ OR } q_S]$

Phép hợp được mở rộng bằng cách lấy OR của các vị từ định tính.

Cho trước các quy tắc trên, chúng ta thấy các phép toán đại số được suy dẫn như thế nào, chẳng hạn phép kết và các phép nửa kết. Chúng ta có thêm hai quy tắc:

Quy tắc 6 $[R: q_R] \triangleright \triangleleft_F [S: q_S] \Rightarrow [R \triangleright \triangleleft_F S: q_R \text{ AND } q_S \text{ AND } F]$

Quy tắc 7 $[R: q_R] \triangleright \triangleleft_F [S: q_S] \Rightarrow [R \triangleright \triangleleft_F S: q_R \text{ AND } q_S \text{ AND } F]$

Chứng minh hai quy tắc này sẽ cho chúng ta thấy cách thao tác các quan hệ định tính như thế nào.

Chứng minh quy tắc 6 (nên nhớ rằng phép kết được suy dẫn từ việc sử dụng phép chọn và tích Descartes).

$$\begin{aligned} & [R: q_R] \triangleright \triangleleft_F [S: q_S] \Rightarrow \\ & \sigma_F([R: q_R] \times [S: q_S]) \Rightarrow \\ & \sigma_F[R \times S: q_R \text{ AND } q_S] \Rightarrow \\ & \sigma_F(R \times S): q_R \text{ AND } q_S \text{ AND } F \Rightarrow \\ & [R \triangleright \triangleleft_F S: q_R \text{ AND } q_S \text{ AND } F] \end{aligned}$$

Chứng minh quy tắc 7 (nên nhớ rằng phép nửa kết được suy dẫn từ việc sử dụng phép chiếu và phép kết).

$$\begin{aligned} & [R: q_R] \triangleright \triangleleft_F [S: q_S] \Rightarrow \\ & \Pi_{Attr(R)}([R: q_R] \triangleright \triangleleft_F [S: q_S]) \Rightarrow \\ & \Pi_{Attr(R)}[R \triangleright \triangleleft_F S: q_R \text{ AND } q_S \text{ AND } F] \Rightarrow \\ & [\Pi_{Attr(R)}(R \triangleright \triangleleft_F S): q_R \text{ AND } q_S \text{ AND } F] \Rightarrow \\ & [R \triangleright \triangleleft_F S: q_R \text{ AND } q_S \text{ AND } F] \end{aligned}$$

Với định nghĩa đại số quan hệ định tính, bây giờ chúng ta nêu ra sự mở rộng của các phép biến đổi tương đương cho nó. Hai quan hệ định tính là tương đương nếu các thân của chúng là các quan hệ tương đương và các vị từ định tính của chúng biểu diễn cùng hàm chân trị (nghĩa là, nếu chúng ta áp dụng cả hai vị từ định tính cho cùng một bộ thì chúng ta có cùng một chân trị). Đề nghị sau đây được áp dụng cho đại số quan hệ định tính: tất cả các phép biến đổi tương đương của đại số quan hệ vẫn còn đúng đối với đại số quan hệ định tính.

Chúng ta sử dụng các vị từ định tính để loại bỏ các mệnh không liên quan đến truy vấn. Xét vị từ định tính được tạo ra sau một phép chọn hoặc một phép chiếu. Trong những trường hợp này, vị từ định

tính sẽ có dạng chuẩn giao các vị từ (*conjunction of predicates*). Vị từ định tính này có thể bị mâu thuẫn. Một ví dụ về một vị từ định tính mâu thuẫn là $DEPTNUM = 1 \text{ AND } DEPTNUM = 5$. Các quan hệ định tính với các vị từ định tính mâu thuẫn về thực chất là rỗng. Điều nhận biết này là có ích hơn so với việc phát hiện kết quả của một biểu thức là rỗng khi thực hiện truy vấn. Trên thực tế, sự mâu thuẫn làm rút giảm các đặc tính ngữ nghĩa (*intensional property*) của các mảnh, và có thể được sử dụng trong lúc biên dịch truy vấn.

Bằng cách sử dụng các vị từ định tính ở trên, đại số quan hệ định tính sẽ cho các kết quả nhất quán mà các vị từ định tính của các toán hạng của nó được cho một cách đúng đắn, và sẽ không biết trước kết quả như thế nào nếu các vị từ định tính là không đúng. Vì chúng ta áp dụng đại số này cho các mảnh của các quan hệ, điều cảnh báo này cho thấy các vị từ định tính của các mảnh cần phải được cho trước một cách chính xác.

Các ví dụ về các biểu thức con dẫn đến quan hệ rỗng là:

$$\sigma_{CITY = 'LA'} [SUPPLIER_1; CITY = 'SF']$$

hoặc

$$[DEPT_1; DEPTNUM \leq 10] \triangleright \triangleleft DEPTNUM = DEPTNUM [EMP_3; DEPTNUM > 20]$$

Việc nhận biết một trong các biểu thức con của một truy vấn thực chất là rỗng sẽ dẫn đến các phép đơn giản hóa đáng kể của cây truy vấn. Các phép biến đổi tương đương sau đây là có ích, trong đó R có thể được xem là một quan hệ thông thường hoặc một quan hệ định tính:

$$\sigma_F(\emptyset) \leftrightarrow \emptyset$$

$$\Pi_A(\emptyset) \leftrightarrow \emptyset$$

$$R \times \emptyset \leftrightarrow R$$

$$R \cup \emptyset \leftrightarrow R$$

$$R - \emptyset \leftrightarrow R$$

$$\emptyset - R \leftrightarrow \emptyset$$

$$R \triangleright \triangleleft_F \emptyset \leftrightarrow \emptyset$$

$$R \triangleright \triangleleft_F \emptyset \leftrightarrow \emptyset$$

$$\emptyset \triangleright \triangleleft_F R \leftrightarrow \emptyset$$

Bây giờ, chúng ta nêu ra các tiêu chuẩn mới để đơn giản hóa các biểu thức trên một lược đồ phân mảnh. Chúng nên được sử dụng chung với các tiêu chuẩn 1 và 2 của phần 5.1. Các tiêu chuẩn mới là:

Tiêu chuẩn 3: Đẩy các phép chọn xuống phía các nút lá của cây, và sau đó thực hiện chúng bằng cách dùng đại số quan hệ định tính. Thay thế kết quả của phép chọn bởi quan hệ rỗng nếu vị từ định tính của kết quả bị mâu thuẫn.

Tiêu chuẩn 4: Sử dụng đại số quan hệ định tính để định trị vị từ định tính của các toán hạng của các phép kết. Thay thế cây con, bao gồm phép kết và các toán hạng của nó, bởi quan hệ rỗng nếu vị từ định tính của kết quả của phép kết bị mâu thuẫn.

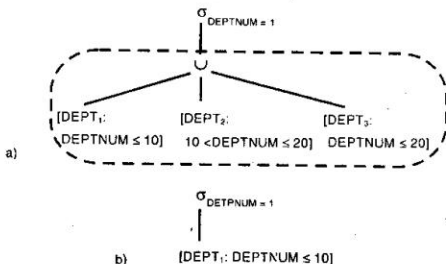
Các tiêu chuẩn trên được dùng để đơn giản hóa các quan hệ được phân mảnh ngang và để đơn giản các phép kết giữa các quan hệ được phân mảnh ngang.

3. Đơn giản hóa các quan hệ được phân mảnh ngang

Chúng ta trình bày phép đơn giản hóa này thông qua một ví dụ. Xét truy vấn Q3 trên quan hệ DEPT được phân mảnh ngang:

Q3: $\sigma_{\text{DEPTNUM} = 1} \text{ DEPT}$

Dạng chuẩn tắc của truy vấn được chỉ ra trong hình 5.5a. Chúng ta đẩy phép chọn hướng xuống các nút lá bằng cách phân phối nó đối với phép hợp. Sau đó, chúng ta thực hiện các phép chọn dựa theo tiêu chuẩn 3. Vị từ định tính của kết quả của các phép chọn trên DEPT_2 và DEPT_3 bị mâu thuẫn. Do đó, hai biểu thức con này được thay thế bởi quan hệ rỗng ở trong cây. Cuối cùng, chúng ta đơn giản phép hợp. Kết quả cuối cùng là cây toán tử đơn giản trong hình 5.5b.



Hình 5.5 Đơn giản hóa các quan hệ được phân mảnh ngang

a) Dạng chuẩn tắc của truy vấn Q3; b) Đơn giản hóa của truy vấn Q3

Lưu ý rằng, nếu các quan hệ toàn cục được phân mảnh ngang một cách đúng đắn, các phép đơn giản hóa ở trên nên được áp dụng cho hầu hết các truy vấn trên chúng.

4- Đơn giản hóa các phép kết giữa các quan hệ được phân mảnh ngang

Để đơn giản, chúng ta xét phép kết giữa hai quan hệ được phân mảnh ngang là R và S . Có hai giải pháp riêng biệt để kết chúng:

- Giải pháp thứ nhất đòi hỏi việc tập hợp tất cả các mảnh của R và của S trước khi thực hiện phép kết.

$$R \bowtie_F S = (\cup R_i) \bowtie_F (\cup S_j)$$

- Giải pháp thứ hai bao gồm việc thực hiện phép kết giữa các mảnh và sau đó tập hợp tất cả các kết quả vào trong cùng một quan hệ kết quả. Chúng ta gọi trường hợp thứ hai này là **phép kết phân bố** (*distrubuted join*).

$$R \bowtie_F S = \cup (R_i \bowtie_F S_j)$$

Trong hai giải pháp trên thì không có giải pháp nào hay hơn. Một cách tổng quát, chúng ta chọn giải pháp thứ nhất nếu có nhiều cặp mảnh được kết với nhau. Chúng ta chọn giải pháp thứ hai nếu có một số cặp mảnh được kết với nhau.

Trong chương 4, chúng ta đã bàn luận đến các tiêu chuẩn cho phép thiết kế phân mảnh các quan hệ để tạo ra một phép kết phân bố hiệu quả. Đối với các phép kết cho trước, các đặc tính được tóm lược trong các đồ thị kết cho thấy các mảnh nào cần được kết với nhau. Các đồ thị kết trợ giúp cho việc thiết kế phân mảnh, nhưng chúng ta cũng cần chỉ ra cách xây dựng các đồ thị kết như thế nào cho các truy vấn cụ thể. Mọi cố gắng thực hiện một thiết kế tốt sẽ vô ích nếu chúng ta không nêu ra các quy tắc xây dựng như thế khi chúng ta ánh xạ các truy vấn toàn cục trên lược đồ phân mảnh.

Phép biến đổi tương đương sau đây cho phép biến đổi một truy vấn không có các phép kết phân bố thành một truy vấn có phép kết phân bố: các phép kết xuất hiện trong truy vấn toàn cục phải được phân phối đối với các phép hợp các mảnh. Lưu ý rằng, điều này tương ứng với việc đẩy phép hợp lên phía trên cây toán tử, nghĩa là phép hợp các mảnh được thực hiện sau. Chúng ta có thể tóm lược phép biến đổi này thành tiêu chuẩn 5.

Tiêu chuẩn 5: Để phân phối các phép kết xuất hiện trong một truy vấn toàn cục, các phép hợp (biểu diễn việc tập hợp các mảnh) phải được đẩy lên phía trên các phép kết mà chúng ta muốn phân phối.

Sau đó, việc xây dựng một đồ thị kết cần phải áp dụng tiêu chuẩn 5 (để phân phối phép kết) tiếp theo sau là tiêu chuẩn 4 (để loại bỏ các phép kết giữa các mảnh mà kết quả là rỗng). Đồ thị kết, đã được định nghĩa trong chương 4, được tạo bởi tất cả các cặp mảnh của R và S có trong cây toán tử.

Chúng ta xét một ví dụ về một phép kết phân bố. Chúng ta bắt đầu từ truy vấn Q_4 cho biết tên NAME của tất cả các nhà cung cấp có đơn hàng cung cấp. Biểu thức đại số của truy vấn trên lược đồ toàn cục là:

$$Q_4: \Pi_{NAME} (SUPPLY \bowtie SUPPLIER)$$

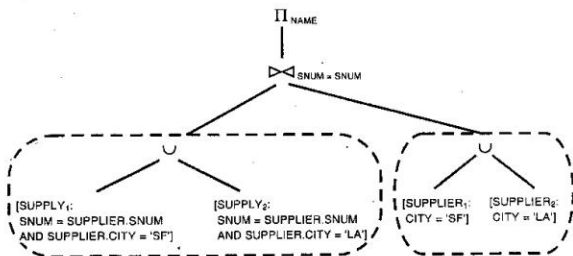
Hình 5.6a cho thấy dạng chuẩn tắc của truy vấn. Chúng ta nhớ rằng sự phân mảnh của SUPPLY được suy từ sự phân mảnh của SUPPLIER. Nghĩa là mỗi bộ của SUPPLY được chứa trong mảnh SUPPLY₁, nếu nó tham chiếu đến một nhà cung cấp ở San Francisco, hoặc được chứa trong mảnh SUPPLY₂ nếu nó tham chiếu đến một nhà cung cấp ở Los Angeles. Bằng cách áp dụng tiêu chuẩn 5, chúng ta đẩy hai phép hợp lên phía trên phép kết. Do đó, chúng ta tạo ra bốn phép kết giữa các mảnh. Sau đó, chúng ta áp dụng tiêu chuẩn 4, và chúng ta phát hiện ra hai phép kết cho ra kết quả rỗng bởi vì vị từ định tính của chúng là mâu thuẫn. Các phép kết rỗng là phép kết SUPPLIER₁ (trong 'SF') với SUPPLY₂ (của các nhà cung cấp ở 'LA'), và phép kết SUPPLIER₂ (trong 'LA') với SUPPLY₁ (của các nhà cung cấp ở 'SF'). Do đó, cây toán tử được rút gọn như trong hình 5.6b. Giả sử các mảnh có cùng chỉ số được đặt ở cùng một nơi (nghĩa là dữ liệu về SUPPLY được chứa chung với dữ liệu về SUPPLIER), cây toán tử này tương ứng với một cách thức hiệu quả để đánh giá truy vấn, bởi vì mỗi phép kết được thực hiện cục bộ ở một nơi.

Lưu ý rằng, phép kết phân bố này có một đồ thị thuộc loại *đơn giản*. Đồ thị kết là:

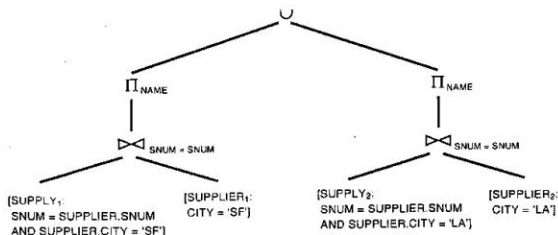
$$SUPPLIER_1 \text{ — } SUPPLY_1$$

$$SUPPLIER_2 \text{ — } SUPPLY_2$$

Cuối cùng, lưu ý rằng ví dụ này cũng đã được giới thiệu trong chương 3 và điều nhấn mạnh là người lập trình viết ứng dụng như thế nào. Ở đây, chúng ta cho thấy chiến lược kết phân bố có thể được xác định như thế nào bởi các phép biến đổi truy vấn.



a) Dạng chuẩn tắc của truy vấn Q4



b) Phép kết phân bố dùng cho Q4

Hình 5.6 Đơn giản hóa các phép kết giữa các quan hệ được phân mảnh ngang

5- Sử dụng phép suy diễn cho các phép đơn giản hóa

Các phần trước đã chỉ ra cách xác định các điều mâu thuẫn giữa các điều kiện chọn của các truy vấn và các vị từ định tính của các mảnh là đặc biệt có ích. Các mâu thuẫn của các trường hợp trên rất dễ dàng bị phát hiện khi phát hiện các giá trị được yêu cầu bởi một truy vấn cho một thuộc tính hoặc một nhóm các thuộc tính là không tương hợp với các tiêu chuẩn phân mảnh cũng được biểu diễn trên cùng một thuộc tính hoặc một nhóm các thuộc tính. Các kiểm tra đơn giản có thể đủ để làm điều này. Tuy nhiên, việc xác định một công thức là mâu thuẫn có thể được thực hiện bằng cách sử dụng các thông tin ngữ nghĩa phức tạp và nói chung đòi hỏi sử dụng một bộ chứng minh định lý (*theorem prover*).

Chúng ta sẽ nêu ra các ví dụ cho thấy các thông tin thêm vào có thể được sử dụng như thế nào để đơn giản hóa các truy vấn. Chúng ta lại xét truy vấn Q1 cho biết mã của các nhà cung cấp có đơn hàng cung cấp ở phía Bắc, mà cây toán tử được chỉ ra trong hình 5.4. Giả sử các điều hiểu biết sau đây có sẵn cho bộ tối ưu hóa truy vấn:

1- Phía Bắc chỉ bao gồm các phòng ban có mã từ 1 đến 10.

2- Tất cả các đơn hàng của các phòng ban có mã từ 1 đến 10 đều gửi đến các nhà cung cấp ở San Francisco.

Chúng ta sử dụng các điều hiểu biết ở trên để suy ra các mâu thuẫn cho phép loại bỏ các biểu thức con.

a- Từ (1) ở trên, chúng ta có thể viết các điều liên quan sau đây:

$AREA = 'NORTH' \Rightarrow NOT (10 < DEPTNUM \text{ AND } DEPTNUM \leq 20)$

$AREA = 'NORTH' \Rightarrow NOT (DEPTNUM > 20)$

Sử dụng tiêu chuẩn 3, chúng ta thực hiện phép chọn trên các mảnh $DEPT_1$, $DEPT_2$ và $DEPT_3$ và định trị vị từ định tính của các kết quả. Vì những điều liên quan ở trên, hai trong các vị từ định tính này là mâu thuẫn. Điều này cho phép chúng ta loại bỏ các biểu thức con đối với các mảnh $DEPT_2$ và $DEPT_3$. Do đó, cây toán tử của hình 5.4b được rút gọn thành cây trong hình 5.7a.

b- Sau đó, chúng ta áp dụng tiêu chuẩn 5 cho việc phân phối phép kết; về nguyên tắc, chúng ta sẽ cần kết cây con bao gồm $DEPT_1$ với các cây con bao gồm $SUPPLY_1$ và $SUPPLY_2$. Nhưng từ (1) ở trên, chúng ta biết rằng:

$AREA = 'NORTH' \Rightarrow DEPTNUM \leq 10$

và từ (2) ở trên, chúng ta biết rằng:

$DEPTNUM \leq 10 \Rightarrow$

$NOT (SNUM \neq SUPPLIER.SNUM \text{ AND } SUPPLIER.CITY = 'LA')$

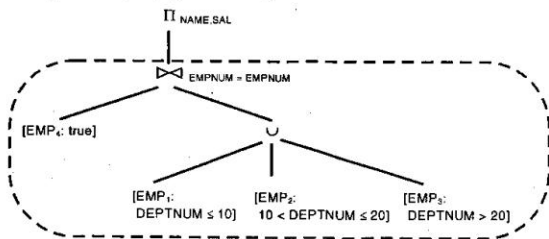
Bằng cách áp dụng tiêu chuẩn 4, có thể rút gọn thành chỉ một cây con bao gồm $SUPPLY_1$ dùng để kết. Cây toán tử cuối cùng của Q1 được chỉ ra trong hình 5.7b.

Các điều ở trên cho thấy các phép đơn giản hóa lũy tiến (*progressive simplification*) đã được áp dụng cho cây toán tử của truy vấn Q1 mà bây giờ nó chỉ bao gồm hai mảnh (có thể lưu trữ ở cùng một nơi).

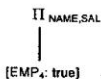
Q5: $\Pi_{NAME, SAL} EMP$

Cây toán tử chuẩn tắc của biểu thức được chỉ ra trong hình 5.8a. Nên nhớ rằng trước tiên EMP được phân mảnh dọc thành EMP₄ (mô tả tiền lương của các nhân viên) và một mảnh thứ hai mà sau này được phân mảnh ngang thành EMP₁, EMP₂ và EMP₃. Chúng ta lưu ý các thuộc tính của EMP₄ bao gồm NAME và SAL cần cho truy vấn. Sau đó, có thể trả lời truy vấn bằng cách chỉ dùng mảnh EMP₄, và cây toán tử có thể được đơn giản hóa bằng cách không quan tâm đến các mảnh khác và phép kết. Cây toán tử cuối cùng của truy vấn Q5 được chỉ ra trong hình 5.8b.

Lưu ý rằng, nếu thiết kế phân mảnh dọc được thực hiện một cách đúng đắn thì phép đơn giản hóa ở trên nên áp dụng cho hầu hết các truy vấn hoạt động trên chúng.



a) Dạng chuẩn tắc của truy vấn Q5



b) Truy vấn đã đơn giản hóa

Hình 5.8 Đơn giản hóa các quan hệ được phân mảnh dọc

7- Các chương trình nửa kết

Một phép nửa kết (*semi-join*) là một phép toán đại số quan hệ được suy dẫn mà nó thích hợp đặc biệt trong tối ưu hóa các truy vấn phân bố. Trong phần này, với một phép kết cho trước, chúng ta sẽ cho thấy ánh xạ phép kết này thành một **chương trình nửa kết** (*semi-join*

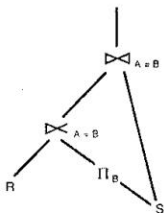
program) như thế nào, nghĩa là một tập hợp các phép toán tạo ra cùng một kết quả với phép kết. Chúng ta cũng nêu ra một giải thích trực giác rằng tại sao các phép nửa kết có thể có ích.

Cho một phép kết bằng (equi-join) $R \bowtie_{A=B} S$, trong đó A và B là các thuộc tính (hoặc tổng quát hơn, tập hợp các thuộc tính) của R và S , chương trình nửa kết ứng với phép kết này được cho bởi:

$$S \bowtie_{A=B} (R \bowtie_{A=B} \Pi_B S)$$

Lưu ý rằng, quan hệ S xuất hiện hai lần trong biểu thức này. Bằng cách gộp các nút lá tương ứng với S thành cùng một nút, chúng ta có được đồ thị toán tử của chương trình nửa kết được chỉ ra trong hình 5.9. Chương trình nửa kết là tương đương với phép kết.

Để hiểu một cách trực giác tại sao các chương trình nửa kết là có ích, giả sử R và S được đặt tại các nơi khác nhau. Việc thực hiện chương trình nửa kết chính là chiếu S trên B và gộp kết quả của phép chiếu đến nơi của R . Thực hiện phép nửa kết tại nơi của R . Gửi kết quả của phép nửa kết đến nơi của S . Và thực hiện phép kết tại nơi đó. Khía cạnh quan trọng của chương trình nửa kết là chỉ có một tập hợp con gồm các bộ của R sẽ tồn tại trong phép nửa kết. Chỉ các bộ này của R , mà chúng tạo ra kết quả của phép kết cuối cùng, được truyền đi giữa các nơi.



Hình 5.9 Đồ thị toán tử của chương trình nửa kết

5.3 GOM NHÓM PHÂN BỐ VÀ ĐỊNH TRỊ HÀM KẾT HỢP

Các ứng dụng CSDL thông thường đòi hỏi việc thực hiện các tác vụ truy xuất CSDL mà chúng không thể được biểu diễn bởi đại số quan hệ. Do đó, các ngôn ngữ truy vấn dùng cho các CSDL quan hệ cho phép thành lập các truy vấn mà chúng không thể được rút gọn thành các biểu thức đại số quan hệ. Điều quan trọng nhất của các đặc điểm thêm vào này là khả năng gom nhóm (grouping) các bộ thành các tập hợp không giao nhau của các quan hệ và định trị các hàm kết hợp

(*aggregate function*) trên chúng. Trong phần này, bằng cách nêu ra các đặc tính thích hợp, chúng ta sẽ cho thấy các truy vấn này có thể được xử lý một cách hiệu quả như thế nào trong một CSDL phân bố.

Trước tiên, bằng cách sử dụng SQL, chúng ta nêu ra một vài ví dụ về các truy vấn mà chúng sử dụng các đặc điểm trên.

Q6: `select AVG(QUAN)`
`from SUPPLY`
`where PNUM = 'P1';`

Truy vấn này lấy dữ liệu đưa vào quan hệ kết quả chỉ có một thuộc tính và một bộ chứa giá trị trung bình của các đơn hàng cung cấp của mặt hàng 'P1'.

Q7: `select PNUM, SNUM, SUM(QUAN)`
`from SUPPLY`
`group by SNUM, PNUM;`

Truy vấn này tương ứng với việc phân chia quan hệ SUPPLY thành các nhóm có cùng giá trị của SNUM và PNUM (nhưng các giá trị của DEPTNUM và QUAN là khác nhau), lấy SNUM, PNUM và tổng số lượng của mỗi nhóm đưa vào trong quan hệ kết quả.

Q8: `select SNUM, PNUM, SUM(QUAN)`
`from SUPPLY`
`group by SNUM, PNUM`
`having SUM(QUAN) > 300;`

Truy vấn này chia quan hệ SUPPLY thành các nhóm, nhưng chỉ xử lý các nhóm mà có tổng số lượng lớn hơn 300. Sau đó lấy ra các thông tin giống như truy vấn Q7.

Lưu ý rằng, không thể biểu diễn trực tiếp các truy vấn này bằng đại số quan hệ. Do đó, chúng ta cần phải mở rộng đại số quan hệ trước khi giải quyết chúng.

1- Mở rộng đại số quan hệ

Đại số quan hệ được mở rộng với phép *gom nhóm* (*group by*) $\Psi_{G, AF} R$ trong đó:

G – các thuộc tính dùng để xác định việc gom nhóm của R , được gọi là tập thuộc tính gom nhóm

AF – các hàm kết hợp được định trị trên mỗi nhóm.

$\Psi_{GAF} R$ là một quan hệ có:

- Lực đồ quan hệ được tạo ra bởi các thuộc tính của G và các hàm kết hợp của AF .
- Nhiều bộ mà mỗi bộ là một nhóm trong R . Các thuộc tính của G lấy giá trị của nhóm. Các thuộc tính của AF lấy giá trị của các hàm kết hợp được định trị trên nhóm.

Hoặc G hoặc AF có thể không có.

Với phép toán ở trên, có thể viết các truy vấn Q6, Q7, và Q8 trong đại số quan hệ. Chúng ta có:

Q6: $\Psi_{AVG(QUAN)} \sigma_{PNUM = 'P1'} SUPPLY$

Q7: $\Psi_{SNUM, PNUM, SUM(QUAN)} SUPPLY$

Q8: $\sigma_{SUM(QUAN) > 300} \Psi_{SNUM, PNUM, SUM(QUAN)} SUPPLY$

Lưu ý rằng, phần G tương ứng với mệnh đề *GROUP BY*, và phần AF tương ứng với các hàm kết hợp cần được tính toán. Trong các truy vấn này, các thuộc tính mà dựa vào chúng để gom nhóm cũng phải được lấy ra. Do đó, các thuộc tính xuất hiện trong mệnh đề *GROUP BY* cũng phải xuất hiện trong mệnh đề *SELECT*.

Trong truy vấn Q6, phần G là rỗng và hàm được áp dụng cho tất cả các bộ của *SUPPLY*. Trong truy vấn Q8, phép chọn thông thường được áp dụng cho kết quả của phép Ψ . Phép chọn này tương ứng với mệnh đề *HAVING* của SQL.

2- Các đặc tính của phép gom nhóm

Trong phần này, chúng ta nêu ra một đặc tính tương đương cho phép toán mới, và nói chung, chúng ta bàn luận khả năng định trị các hàm kết hợp theo một cách thức phân bố.

Đặc tính mà chúng ta quan tâm đến là tính phân bố của Ψ đối với phép hợp:

$$\Psi_{GAF}(R_1 \cup R_2) \rightarrow (\Psi_{GAF} R_1) \cup (\Psi_{GAF} R_2)$$

Điều kiện cần và đủ cho đặc tính này đòi hỏi mỗi nhóm G_i hoặc được chứa hoặc không được giao nhau với mọi toán hạng R_j .

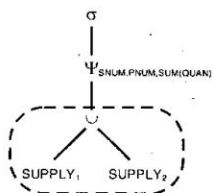
Điều kiện cần và đủ: Đối với mọi i, j thì hoặc $(G_i \subseteq R_j)$ hoặc $(G_i \cap R_j = \emptyset)$.

Ý nghĩa của điều kiện này là mỗi nhóm phải được chứa hoàn toàn trong một mảnh, nghĩa là việc gom nhóm phải nhỏ hơn việc

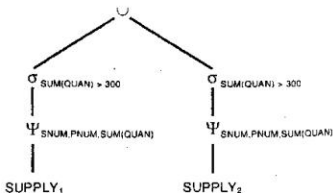
phân mảnh. Rõ ràng, việc thực hiện phép gom nhóm trên các toán hạng của phép hợp và sau đó hợp các kết quả này thì tương đương với việc thực hiện phép gom nhóm trực tiếp trên kết quả của phép hợp. Tất nhiên, tính phân bố của phép gom nhóm dẫn đến việc định trị phân bố của truy vấn mà điều này vô cùng có ích, bởi vì các kết quả (nhỏ) của các phép gom nhóm được tập hợp lại thay vì của các quan hệ toàn cục (lớn). Bây giờ, chúng ta có thể phát biểu một tiêu chuẩn mới áp dụng cho phép biến đổi ở trên.

Tiêu chuẩn 6: Để phân bố việc gom nhóm và định trị hàm kết hợp xuất hiện trong một truy vấn toàn cục, các phép hợp (biểu diễn việc tập hợp các mảnh) phải được đẩy lên phía trên phép gom nhóm tương ứng.

Tiêu chuẩn này áp dụng có kết quả cho các truy vấn Q7 và Q8. Chúng ta xét truy vấn Q8. Dạng chuẩn tắc của truy vấn này được chỉ ra trong hình 5.10a. Các bộ của SUPPLY phải được gom nhóm lại theo các giá trị khác nhau của các thuộc tính SNUM và PNUM, và chúng ta biết rằng, từ sự phân mảnh của SUPPLY, các giá trị bằng nhau của SNUM thuộc cùng một mảnh. Sau đó, chúng ta có thể áp dụng phép biến đổi theo tiêu chuẩn 6 để đẩy phép hợp lên phía trên đối với phép Ψ . Chúng ta phân phối phép chọn đối với phép hợp. Cây toán tử cuối cùng của truy vấn Q8 được chỉ ra trong hình 5.10b. Lưu ý rằng, phép Ψ được áp dụng riêng biệt cho mỗi mảnh và phép chọn được áp dụng cho kết quả.



a) Dạng chuẩn tắc của truy vấn Q8



b) Bản phân bố của truy vấn Q8

Hình 5.10 Một truy vấn với việc gom nhóm và các hàm kết hợp

Bằng phép biến đổi và tiêu chuẩn ở trên, chúng ta có thể đơn giản nhiều truy vấn, nhưng chúng ta vẫn phải xét đến các truy vấn mà không thể áp dụng điều kiện cần và đủ ở trên. Ví dụ, điều này là trường hợp của truy vấn Q6, mà trong đó không có gom nhóm. Chúng ta có thể nêu ra một đặc tính khác có liên quan đến một số hàm kết hợp.

Chúng ta nói rằng, một hàm kết hợp F có tính toán phân bố nếu đối với một đa tập (*multiset*) S bất kỳ và một phân rã bất kỳ của S thành các đa tập $S_1, S_2, \dots, S_n$, chúng ta có thể xác định một tập hợp các hàm kết hợp $F_1, \dots, F_m$ và một biểu thức $E(F_1, \dots, F_m)$ sao cho:

$$F(S) = E(F_1(S_1), \dots, F_1(S_n), F_2(S_1), \dots, F_2(S_n), \dots, F_m(S_1), \dots, F_m(S_n))$$

Chúng ta nhớ rằng, một đa tập có các phần tử được nhân bản. Trong định nghĩa ở trên, phân rã của S phải đảm bảo mỗi phần tử của S được ánh xạ đến một và chỉ một phần tử của một trong các đa tập $S_1, \dots, S_n$ (nghĩa là bậc nhân bản của các phần tử là không thay đổi). Khái niệm phân rã này là đúng khi $S_1, S_2, \dots, S_n$ là các đa tập được cho bởi các giá trị của một cột cho trước trên các mảnh $R_1, R_2, \dots, R_n$ và S là một đa tập được cho bởi các giá trị của cột này trên quan hệ toàn cục R .

Một hàm kết hợp mà chúng ta có thể tìm ra các hàm F_i và biểu thức $E(F_i)$ đó là hàm tính trung bình:

$$AVG(S) = \frac{SUM(SUM(S_1), SUM(S_2), \dots, SUM(S_n))}{SUM(COUNT(S_1), COUNT(S_2), \dots, COUNT(S_n))}$$

Tương tự, chúng ta có:

$$MIN(S) = MIN(MIN(S_1), MIN(S_2), \dots, MIN(S_n))$$

$$MAX(S) = MAX(MAX(S_1), MAX(S_2), \dots, MAX(S_n))$$

$$COUNT(S) = SUM(COUNT(S_1), COUNT(S_2), \dots, COUNT(S_n))$$

$$SUM(S) = SUM(SUM(S_1), SUM(S_2), \dots, SUM(S_n))$$

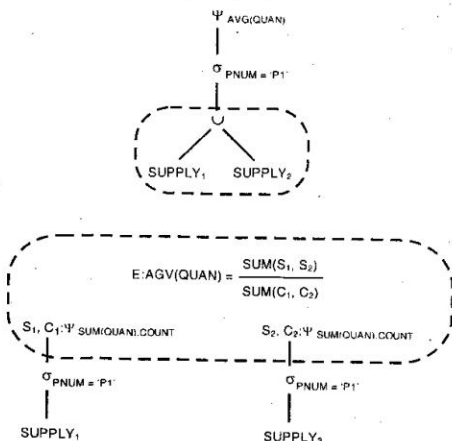
Tính toán phân bố của các hàm kết hợp là một đặc tính quan trọng trong các CSDL phân bố, bởi vì các kết quả từng phần của các hàm $F_1(S_1), \dots, F_m(S_n)$ có thể được truyền đến một nơi chung, mà tại đó biểu thức E có thể được định trị, thay vì truyền tất cả dữ liệu đến nơi đó và tính toán hàm kết hợp tại nơi đó.

Đặc tính trên có thể được áp dụng cho truy vấn Q6. Dạng chuẩn tắc của truy vấn được chỉ ra trong hình 5.11a. Trong trường hợp này, chúng ta không thể áp dụng tiêu chuẩn 6, bởi vì điều kiện cần và đủ không có giá trị. Để giải quyết truy vấn này, chúng ta tạo ra hai truy vấn con độc lập, thực hiện trên hai mảnh $SUPPLY_1$ và $SUPPLY_2$:

$$\Psi_{SUM(QUAN), COUNT} \sigma_{PNUM = 'P1'} SUPPLY_i, \quad \text{với } i = 1, 2$$

Gọi S_1 và C_1 là các giá trị được trả về tương ứng với các thuộc tính $SUM(QUAN)$ và $COUNT$ của biểu thức trên $SUPPLY_1$, và S_2 và C_2 là các giá trị tương ứng trên $SUPPLY_2$. Khi đó, giá trị số lượng trung bình $AVG(QUAN)$ được cho bởi $(S_1 + S_2)/(C_1 + C_2)$. Do đó, S_1 , S_2 , C_1 và C_2 được truyền đến nơi tạo ra kết quả của truy vấn, và giá trị $AVG(QUAN)$ được tính tại nơi đó.

Trong trường hợp không có phép biến đổi này, việc định trị không phân bố của hàm sẽ đòi hỏi việc tập hợp các bộ thật sự của mảnh $SUPPLY_1$ và $SUPPLY_2$ tại cùng một nơi.



Hình 5.11 Định trị phân bố của các hàm kết hợp

5.4 CÁC TRUY VẤN CÓ THAM SỐ

Bây giờ, chúng ta xét các truy vấn có tham số. Truy vấn tham số (*parametric query*) là truy vấn mà trong đó các công thức trong các điều kiện chọn của truy vấn bao gồm các tham số mà các giá trị của chúng chưa được biết khi biên dịch truy vấn. Khi thực hiện các truy vấn tham số, người sử dụng cung cấp các giá trị để gán cho các tham số. Do đó, các truy vấn tham số cho phép thực hiện nhiều lần các truy vấn với nhiều giá trị khác nhau của các tham số, tất nhiên ở mỗi lần thực hiện sẽ trả về các kết quả khác nhau.

1- Đơn giản hóa các truy vấn tham số và mở rộng đại số

Chúng ta xét một ví dụ về truy vấn tham số là truy vấn Q9, chọn các bộ của quan hệ toàn cục DEPT có các mã phòng ban cho trước. Phép chọn trên DEPTNUM có tham số:

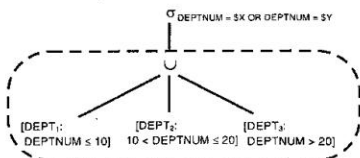
Q9: $\sigma_{DEPTNUM = \$X \text{ OR } DEPTNUM = \$Y} \text{ DEPT}$

Ở thời gian chạy, các giá trị thực sự được gán cho các tham số \$X và \$Y bởi chương trình chạy truy vấn này.

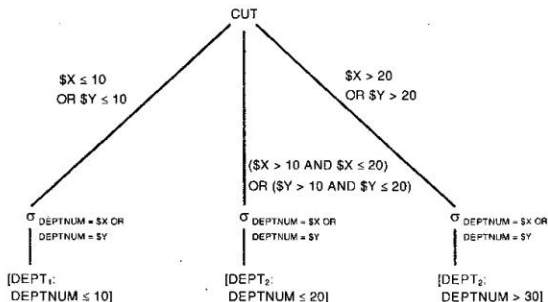
Phép đơn giản hóa của các truy vấn tham số có một số đặc tính khác biệt. Xét truy vấn ở trên mà dạng chuẩn tắc của nó được chỉ ra trong hình 5.12a. Ở thời gian biên dịch, chúng ta không biết các mảnh nào của quan hệ toàn cục DEPT sẽ được sử dụng. Tuy nhiên, chúng ta biết rằng *nhiều nhất là hai mảnh* của nó sẽ có liên quan đến truy vấn. Ở thời gian chạy, khi truy vấn được tác động, cho biết giá trị của các tham số, chúng ta sẽ xác định được các mảnh nào có liên quan đến truy vấn. Điều này cho thấy một phần của phép đơn giản hóa của một truy vấn tham số có thể được thực hiện ở thời gian biên dịch, nhưng một phần của nó cần được thực hiện ở thời gian chạy.

Lưu ý rằng, phép đơn giản hóa các truy vấn tham số có liên quan đến việc áp dụng đại số quan hệ định tính để xác định các vị từ định tính của các biểu thức con là mâu thuẫn với nhau.

Trong ví dụ trên, chúng ta giả sử khi truy vấn được thực hiện, các tham số có các giá trị sau đây: $X = 1$ và $Y = 13$. Sau đó, ở thời gian chạy, chúng ta có thể đơn giản hóa cây con của DEPT_3 bởi vì công thức chọn $\text{DEPTNUM} = 1 \text{ OR } \text{DEPTNUM} = 13$ mâu thuẫn với vị từ định tính của nó.



a) Dạng chuẩn tắc của truy vấn Q9



b) Cây truy vấn với phép CUT

Hình 5.12 Sử dụng CUT trong một truy vấn có tham số

Do đó, về nguyên tắc, chúng ta không cần giải quyết các phép đơn giản hóa ở thời gian chạy một cách khác so với các phép đơn giản hóa ở thời gian biên dịch. Tuy nhiên, chúng ta muốn hầu hết tối ưu hóa được thực hiện ở thời gian biên dịch, đặc biệt là đối với các truy vấn tham số thường được sử dụng. Cũng vậy, chúng ta không muốn sử dụng các kỹ thuật phức tạp (giống như áp dụng các bộ chứng minh định lý) ở thời gian chạy vì các lý do hiệu quả. Do đó, điều này thích hợp cho việc giới hạn tối ưu hóa ở thời gian chạy cho các kiểm tra đơn giản mà chúng cho phép đơn giản hóa việc thực hiện truy vấn. Để có được các biểu thức cho các kiểm tra này, cần phải có một số thao tác đại số (*algebraic manipulation*) trên các định danh (*qualifier*) và các điều kiện chọn mà chúng được thực hiện đúng đắn ở thời gian biên dịch hơn là ở thời gian chạy.

Tối ưu hóa ở thời gian chạy có cùng hiệu quả với việc loại bỏ các biểu thức con trong cây toán tử được tạo ra ở thời gian biên dịch. Để biểu diễn sự hiện diện của các kiểm tra trên cây toán tử đối với phép đơn giản hóa ở thời gian chạy, chúng ta thay thế các phép hợp bởi một phép toán mới n -ngôi, được gọi là CUT. Phép toán mới thực hiện phép hợp của chỉ một số toán hạng của nó. Đối với mỗi toán hạng, phép toán CUT có một mô tả gồm một công thức sử dụng các tham số của truy vấn và vị từ định tính của các mảnh. Mỗi công thức được chuẩn bị ở thời gian biên dịch và được định trị ở thời gian chạy. Nếu công thức trả về *true*, thì toán hạng tương ứng được đưa vào phép hợp. Nếu nó trả về *false*, thì toán hạng tương ứng bị loại bỏ ra khỏi cây. Lưu ý rằng, trong trường hợp này, toàn bộ cây con tạo ra toán hạng không cần được thực hiện. Do đó, điều này thuận lợi cho việc định trị các công thức ngay sau khi các tham số được xác định, để cho cây toán tử được đơn giản hóa trước khi bắt đầu thực hiện thật sự. Trên thực tế, mỗi công thức đòi hỏi biểu thức con tương ứng phải có một vị từ định tính không bị mâu thuẫn.

Hình 5.12b cho thấy cây toán tử mới của truy vấn tham số. Trong ví dụ này, phép chọn được đẩy xuống dưới phép hợp, áp dụng tiêu chuẩn 2, và sau đó phép hợp được thay thế bởi phép CUT. Ba công thức cho phép CUT (mỗi công thức cho mỗi toán hạng của phép hợp) là:

$$F1: \$X \leq 10 \text{ OR } \$Y \leq 10$$

$$F2: (\$X > 10 \text{ OR } \$X \leq 20) \text{ OR } (\$Y > 10 \text{ OR } \$Y \leq 20)$$

$$F3: \$X > 20 \text{ OR } \$Y > 20$$

$F1$, $F2$ và $F3$ được chuẩn bị ở thời gian biên dịch bằng cách xác định các công thức mà chúng phải thỏa mãn với các tham số của truy vấn để cho các vị từ định tính của các toán hạng tương ứng không bị mâu thuẫn. Việc kiểm tra ba công thức trên đòi hỏi phải so sánh vị từ của truy vấn với vị từ định tính của các biểu thức con mà nó được định trị bằng cách sử dụng đại số quan hệ định tính.

Ở thời gian chạy, phép CUT phải được định trị trước tiên. Đối với $\$X = 1$ và $\$Y = 13$, truy vấn được rút gọn thành nhánh thứ nhất và nhánh thứ hai của cây toán tử. Đối với $\$X = 1$ và $\$Y = 5$, truy vấn được rút gọn thành nhánh thứ nhất của cây.

2- Sử dụng vùng nhớ tạm thời khi thực hiện nhiều lần các truy vấn tham số

Các truy vấn tham số có đặc điểm là chúng được sử dụng nhiều lần, với các giá trị khác nhau của các tham số. Trong một số trường hợp, những lần thực hiện lặp lại được tập trung trong những khoảng thời gian ngắn. Ví dụ, điều này xảy ra khi truy vấn là một phần của một vòng lặp của một chương trình ứng dụng, mà ở mỗi lần lặp nó đòi hỏi một giá trị mới và gán giá trị này cho tham số của truy vấn.

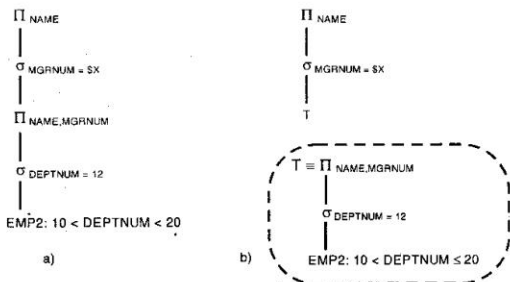
Như chúng ta đã thấy trong chương 3, việc thực hiện lặp lại một truy vấn ở mỗi lần tác động có một chi phí nào đó. Để làm giảm chi phí này, chúng ta có thể sử dụng các quan hệ tạm thời ở nơi gốc của truy vấn, mà nơi này lưu trữ một tập cha (*superset*) chứa dữ liệu cần thiết cho mỗi lần lặp. Chúng ta không thể đi vào vấn đề khá phức tạp về việc đánh giá sự thuận lợi của việc tạo lập các bản sao tạm thời của thông tin. Thay vào đó, bằng một ví dụ, chúng ta cho thấy làm sao có thể tạo một quan hệ tạm thời có ích bằng cách thao tác cây toán tử của truy vấn. Chúng ta xét truy vấn Q10 cho biết tên của các nhân viên đang làm việc ở phòng ban 12 mà có mã sếp là \$X (nghĩa là mã sếp là tham số của truy vấn). Chúng ta có:

Q10: Π NAME σ MGRNUM = SX AND DEPTNUM = 12 EMP

Trước tiên, chúng ta thay đổi truy vấn sao cho có được cây toán tử như trong hình 5.13a. Mục đích tổng quát của phép biến đổi này là cô lập các cây con không có tham số, bởi vì chúng sẽ tạo ra các vùng tạm thời cần thiết.

Trong ví dụ này, chúng ta đã sử dụng tính lũy đẳng để tạo ra hai phép chọn riêng biệt. Một phép chọn dựa trên MGRNUM có tham số và một phép chọn dựa trên DEPTNUM không có tham số. Chúng ta cũng đã sử dụng tính lũy đẳng để tạo ra một phép chiếu trên NAME và MGRNUM. Phép chọn dựa trên DEPTNUM và phép chiếu được đẩy xuống phía các nút lá của cây toán tử như thông thường, trong khi đó phép chọn dựa trên MGRNUM được đẩy lên phía trên. Do đó, chúng ta đã cô lập một cây con không có tham số tương ứng với biểu thức:

Π NAME, MGRNUM σ DEPTNUM = 12 EMP



Hình 5.13 Sử dụng các quan hệ tạm thời cho các truy vấn tham số

Sau đó, chúng ta gọi T là vùng tạm thời được cho bởi biểu thức trên, và phân chia cây toán tử thành hai phần tương ứng với T (xem hình 5.13b). T biểu diễn quan hệ tạm thời mà nó sẽ được sử dụng cho những lần thực hiện lặp lại của truy vấn $Q10$.

TỐI ƯU HÓA CÁC CHIẾN LƯỢC TRUY XUẤT

Trong chương trước, chúng ta đã thấy một truy vấn được cải thiện như thế nào dựa vào các phép biến đổi thích hợp. Trong chương này, chúng ta đề cập đến tối ưu hóa các chiến lược truy xuất, nghĩa là, chúng ta chọn một trong các khả năng khác nhau để có chi phí thấp nhất. Trong thực tế, thuật ngữ *tối ưu hóa* (*optimization*) là không chính xác, bởi vì trên thực tế, các kỹ thuật để tối ưu hóa việc thực hiện không có được tính tối ưu (*optimality*) và chỉ tìm kiếm các chiến lược truy xuất tốt. Hơn nữa, chúng chỉ dựa vào việc đơn giản các giả sử về môi trường xử lý. Do đó, bất kỳ yêu cầu về tính tối ưu cần phải được xem xét cẩn thận. Tuy nhiên, chúng ta vẫn dùng thuật ngữ tối ưu hóa này.

Hầu hết các phép biến đổi đã được nêu trong chương trước, trong khi đó việc thiết lập một tiền đề cơ bản cho tối ưu hóa truy vấn không đòi hỏi thực sự việc chọn lựa giữa các cách khác nhau, bởi vì chúng chắc chắn đều có ích. Việc rút ra các biểu thức con chung và việc loại bỏ các biểu thức con không cần thiết trong truy vấn là các ví dụ về các phép biến đổi chắc chắn có ích. Cũng vậy, việc đẩy các phép toán một ngôi về phía các nút lá của các cây toán tử là có ích, bởi vì điều này cho phép giảm kích thước của các toán hạng ở thời điểm sớm nhất; điều này cũng có ích trong các môi trường không phân bố (mà các nhân tố chi phí chính có liên quan đến việc thực hiện các phép kết), và đặc biệt có ích trong các môi trường phân bố (mà chúng ta đã đưa vào các chi phí truyền dữ liệu). Điều quan trọng là các phép biến đổi của chương 5 được thực hiện trên một cơ sở lý luận và không cần có bất kỳ giả sử nào về môi trường xử lý.

Có hai ngoại lệ chính là:

- Tính giao hoán của các phép kết và các phép hợp.
- Phép biến đổi của các phép kết thành các chương trình nửa kết.

Các phép biến đổi này tạo ra các chiến lược xử lý truy vấn khác nhau mà phải được so sánh trên cơ sở chi phí. Do đó, các phép biến đổi này thuộc về cả chương 5 và chương 6 và chúng ta sẽ giải quyết chúng một lần nữa.

6.1 MỘT CƠ CẤU CHO TỐI ƯU HÓA TRUY VẤN

Trong phần này, chúng ta nêu ra sự phân loại của các vấn đề xử lý truy vấn, của các giả sử cần có để mô hình hóa và giải quyết chúng, và của các điều kiện được sử dụng trong tối ưu hóa. Sau đó, chúng ta xây dựng một mô hình mới của các truy vấn và nêu ra tất cả các tham số định lượng thích hợp mà chúng là một phần của mô hình.

6.1.1 Các vấn đề của tối ưu hóa truy vấn

Chọn một chiến lược xử lý truy vấn bao gồm:

a- Cho trước một biểu thức truy vấn trên các mảnh, xác định các bản nhân vật lý của các mảnh mà chúng được dùng để thực hiện truy vấn này. Thuật ngữ *bản thực hiện* (*materialization*) được sử dụng trong tài liệu dùng để biểu thị một bản sao không dư thừa của toàn bộ cơ sở dữ liệu (CSDL) phân bố mà truy vấn được thực hiện trên đó. Theo mô hình tham khảo trong chương 3, đối với mỗi mảnh, một bản thực hiện tương ứng với việc chọn lựa một trong các bản nhân của mảnh. Lưu ý rằng, nói chung, các truy vấn khác nhau sẽ sử dụng các bản thực hiện khác nhau.

Chọn một bản thực hiện cho một truy vấn là một hạn chế về tính tổng quát của vấn đề, bởi vì, về mặt nguyên tắc, không có lý do gì để chọn một bản nhân khác của mỗi mảnh để thực hiện cùng một truy vấn. Chẳng hạn, nếu các biểu thức con khác nhau cùng chứa một mảnh thì chúng có thể được thực hiện trên các bản nhân khác nhau của mảnh này. Tuy nhiên, hầu hết các giải thuật xử lý truy vấn đều hoạt động trong ngữ cảnh của bản thực hiện.

b- Chọn thứ tự thực hiện của các phép toán, như chúng ta đã thấy, điều này bao hàm việc xác định một chuỗi các phép kết, phép nối kết và phép hợp có lợi, bởi vì việc xác định thứ tự thực hiện của các phép toán khác là không khó. Điều đáng lưu ý là cây toán tử được tạo ra sau các phép biến đổi của chương 5 ngầm định một thứ tự bộ

phần của các phép toán, bao gồm việc thực hiện chúng đi từ các nút lá đến nút gốc. Tuy nhiên, điều này không phải hoàn toàn xác định một giải pháp của vấn đề tối ưu hóa, bởi vì cũng cần phải chỉ ra thứ tự định trị các biểu thức con mà chúng được thực hiện ở cùng một mức của cây. Hơn nữa, việc đi từ các nút lá đến nút gốc không nhất thiết là một giải pháp tốt nhất.

c- Chọn phương pháp để thực hiện mỗi phép toán; điều này bao hàm việc chọn các phép toán đại số được thực hiện chung với nhau ở trong cùng một truy xuất cơ sở dữ liệu (chẳng hạn, thực hiện các phép chọn và các phép chiếu trên cùng một toán hạng ở cùng một thời điểm), và chọn một phương pháp để thực hiện mỗi truy xuất CSDL giữa các phương pháp khác nhau có thể có. Vấn đề khó nhất là xác định phương pháp tốt nhất để đánh giá các phép kết. Vấn đề cuối cùng này là khác biệt cho mỗi hệ thống riêng biệt, và chúng ta sẽ không nêu ra một giải pháp chung cho nó. Trong phần 6.2.4 chúng ta sẽ nêu ra các phương pháp để đánh giá các phép kết mà chúng được hỗ trợ bởi nguyên mẫu R^* (R^* prototype). Nói chung, chúng ta xét tất cả các chuỗi phép toán được thực hiện trên các toán hạng, mà chúng được lưu trữ tại cùng một nơi CSDL khi thiết lập một chương trình đơn, nhưng chúng ta không chỉ ra cách thực hiện như thế nào cho các truy xuất CSDL tương ứng.

Các vấn đề ở trên đều phụ thuộc nhau. Chẳng hạn, chọn bản thực hiện tốt nhất cho một truy vấn phụ thuộc vào thứ tự mà các phép toán được thực hiện. Do đó, cách tiến hành bằng cách giải quyết chúng một cách độc lập sẽ dẫn đến các sai lầm. Tuy nhiên, trong các phương pháp tối ưu hóa, điều đơn giản là xem xét ba vấn đề độc lập nhau là:

- Thừa nhận bản thực hiện đối với một truy vấn cho trước.
- Thứ tự thực hiện các phép toán đã được tối ưu hóa.
- Các phép toán được gom lại thành các chương trình cục bộ.

Trên thực tế, vấn đề đầu tiên thường được bỏ qua, vấn đề thứ ba cũng không được quan tâm đến (bởi vì nó phụ thuộc vào hệ thống) và điều nhấn mạnh là vấn đề thứ hai. Chúng ta sẽ đi theo cách tiếp cận này.

Khi chúng ta tập trung vào vấn đề xác định thứ tự thực hiện của các phép toán, các truy vấn có thể vẫn được mô hình hóa theo các mảnh, như trong chương 5, ở trong bản thực hiện, mỗi mảnh được ánh xạ thành một trong các bản nhân vật lý của nó ở trong các hình ảnh vật lý và được gọi là **mảnh được định vị** (*allocated fragment*). Chọn bản thực hiện tối ưu cho một truy vấn cho trước đòi hỏi phải hiểu biết đầy đủ về ánh xạ giữa các mảnh và các hình ảnh vật lý.

6.1.2 Các mục tiêu của tối ưu hóa xử lý truy vấn

Chọn các chiến lược thực hiện truy vấn khác nhau, trong cả hai môi trường tập trung và môi trường phân bố, được thực hiện bằng việc đo lường các hiệu suất của chúng. Các số đo tiêu biểu được áp dụng trong CSDL tập trung là số các tác vụ I/O và việc sử dụng CPU để thực hiện truy vấn. Trong các CSDL phân bố, chúng ta cũng phải xét đến khối lượng dữ liệu được truyền giữa các nơi. Tuy nhiên, không có sự thỏa thuận nào về tầm quan trọng tương đối của chi phí của việc truyền thông đối với I/O cục bộ.

Chỉ có các chi phí truyền thông liên hệ chặt chẽ với CSDL phân bố được phân tán về mặt địa lý, mà các hạn chế về băng thông (*bandwidth*) là nghiêm ngặt. Trong các mạng cục bộ, băng thông giữa các nơi (*intersite bandwidth*) gần với băng thông của các tác vụ I/O, và do đó việc truyền thông giữa các nơi (*intersite communication*) được so sánh với các tác vụ CPU và I/O cục bộ.

Chỉ có các yêu cầu truyền thông là đáng quan tâm, bởi vì:

- Các yêu cầu truyền thông không phụ thuộc vào các hệ thống; chúng là một hàm theo khối lượng dữ liệu được truyền giữa các nơi (điều này không được áp dụng cho các số đo I/O mà chúng phụ thuộc vào phương pháp được sử dụng để thực hiện các tác vụ).

- Tối ưu hóa một truy vấn phân bố có thể được phân chia thành hai vấn đề độc lập nhau: phân bố chiến lược truy xuất giữa các nơi được thực hiện bằng cách chỉ xét việc truyền thông, và xác định các chiến lược truy xuất cục bộ tại mỗi nơi bằng cách sử dụng các phương pháp truyền thống của các CSDL tập trung. Hai vấn đề này được giải quyết một cách tuần tự bởi vì vấn đề đầu tiên quan trọng hơn vấn đề sau cùng.

Nói chung, trong chương này, chúng ta giả sử tối ưu hóa toàn cục (*global optimization*) có thể được thực hiện một cách độc lập với tối ưu hóa cục bộ (*local optimization*), và nó có thể chỉ dựa trên các chi phí truyền thông, với ngoại lệ của phần 6.2.4 mà ở đó chúng ta sẽ bàn luận về một giải thuật tối ưu hóa truy vấn cũng bao gồm việc chọn các phương pháp truy xuất cục bộ và xét các chi phí I/O và CPU của việc tối ưu hóa.

Tuy nhiên, cơ sở của các phương pháp xử lý truy vấn được giới thiệu trong chương này vẫn còn có giá trị nếu chúng ta loại bỏ giả sử trên. Trong trường hợp này, chúng ta có thể có một mô hình phức tạp của các chiến lược xử lý truy vấn, nhưng cấu trúc cơ sở của các giải thuật được giới thiệu trong chương này vẫn không bị thay đổi.

Các yêu cầu truyền thông có thể được đánh giá theo chi phí và các trì hoãn (*delay*):

- Khi xét các chi phí, số đo hiệu suất của một ứng dụng là tổng của tất cả chi phí truyền thông của mỗi lần thực hiện truyền thông. Cách tiếp cận này tương ứng với tối thiểu hóa tổng chi phí truyền thông trong mạng truyền thông.

- Khi xét các trì hoãn, số đo hiệu suất của một ứng dụng được cho bởi thời gian trôi qua giữa lúc tác động và lúc hoàn tất ứng dụng. Giảm các trì hoãn bằng cách tăng mức độ song song hóa (*degree of parallelism*) trong việc thực hiện và không nhất thiết tối thiểu hóa tổng chi phí truyền thông.

Các chi phí truyền thông TC và các trì hoãn truyền thông TD của một lần truyền thông được mô hình hóa một cách tiêu biểu bởi một hàm tuyến tính theo kích thước x của dữ liệu được truyền:

$$TC(x) = C_0 + x \times C_1$$

$$TD(x) = D_0 + x \times D_1$$

trong đó C_0 , C_1 , D_0 và D_1 là các hằng số phụ thuộc hệ thống (*system-dependent constant*); C_0 tương ứng với chi phí cố định để khởi động một lần truyền thông giữa hai nơi; C_1 là chi phí truyền thông đơn vị mạng (*networkwide unitary transmission cost*); D_0 là thời gian cố định để thiết lập một kết nối; D_1 là tốc độ truyền đơn vị mạng (*networkwide unitary transfer rate*).

Các mô hình khác thừa nhận một đặc trưng chi tiết hơn của các chi phí và các trì hoãn bằng cách kết hợp các hệ số khác nhau cho mỗi cặp nơi. Trong trường hợp này, chúng ta có:

$$TC(x) = C_o^{ij} + x \times C_I^{ij}$$

$$TD(x) = D_o^{ij} + x \times D_I^{ij}$$

trong đó: i - chỉ số biểu thị nơi nguồn

j - chỉ số biểu thị nơi đích của lần truyền thông.

Đặc trưng chi tiết này đưa thêm thông tin có nghĩa, bởi vì:

- Trong các mạng được phân tán về mặt địa lý, các chi phí và các trì hoãn phụ thuộc vào các tuyến dẫn (*routing*) (nghĩa là chuỗi các nối kết nối nơi nguồn và nơi đích) mà chúng được xác định một cách động. Do đó, khó có thể gán các giá trị chính xác cho các hệ số trên.

- Trong các mạng cục bộ, việc truyền thông giữa các nơi được thực hiện tiêu biểu bằng cách sử dụng các mối liên kết đồng nhất, và do đó nó không có ích cho các chi phí đơn vị giữa các nơi độc lập.

Trong một số trường hợp, các chi phí đơn vị giữa các nơi khác nhau có sức thuyết phục hơn:

- Khi các chi phí được thay đổi theo những người sử dụng của hệ thống CSDL phân bố dựa trên cơ sở khoảng cách (do đó, cần phải có một bộ tối ưu hóa tốt để người sử dụng tiết kiệm tiền bạc).

- Khi cấu trúc liên kết mạng (*network topology*) giúp ích trong việc phân biệt các lớp kết nối giữa các nơi (chẳng hạn như trong một mạng hình sao, chúng ta phân biệt việc truyền thông giữa các nơi bao gồm nơi trung tâm và một nơi xa hoặc giữa hai nơi xa khác nhau).

6.1.3 Mô hình mới của các truy vấn

Trong phần trước, chúng ta đã nêu ra dạng biểu diễn cây của các truy vấn mà trong đó các nút của cây biểu diễn các phép toán khác nhau (bao gồm các phép toán đại số truyền thống và các phần đại số mở rộng). Chúng ta cũng đã biến đổi các cây thành các đồ thị để xác định các biểu thức con chung. Trong phần này, chúng ta nêu ra một dạng biểu diễn đơn giản hơn của các truy vấn bằng cách chỉ xét các phép toán quan trọng trong việc xử lý truy vấn. Chúng ta cũng đưa vào mô hình này tất cả các thông tin định lượng (*quantitative information*) có thể có ích cho việc đánh giá các chiến lược xử lý truy vấn khác nhau.

1- Các bản tóm lược cơ sở dữ liệu

Chúng ta bắt đầu bằng việc xác định mô tả thống kê (*statistical description*) của CSDL mà nó cần cho việc phân tích định lượng (*quantitative analysis*). Chúng ta sử dụng thuật ngữ *bản tóm lược* (*profile*) cho mô tả này. Các bản tóm lược sẽ mô tả các mảnh và chứa các thông tin sau đây:

- Số bộ của mỗi mảnh R_i , ký hiệu là $card(R_i)$.
- Kích thước (số byte) của mỗi thuộc tính A , được ký hiệu là $size(A)$ (để đơn giản, các thuộc tính có cùng tên trong lược đồ CSDL được giả sử có cùng kích thước); kích thước của một mảnh, $size(R_i)$, là tổng của các kích thước của các thuộc tính của mảnh.

- Đối với mỗi thuộc tính A trong mỗi mảnh R_i , số giá trị khác biệt xuất hiện trong R_i , ký hiệu là $val(A/R_i)$.

Thông tin này được dùng để mô tả cho các quan hệ toàn cục hơn là cho các mảnh. Ví dụ, hình 6.1a cho thấy các bản tóm lược của hai quan hệ toàn cục và hình 6.1b cho thấy các bản tóm lược của hai mảnh. Hình 6.1b cho thấy, trong các ví dụ của chương này, đặc trưng của mỗi mảnh R , bao gồm các bản tóm lược của mảnh và nơi đặt mảnh này, ký hiệu là $site(R_i)$.

$card(SUPPLY) = 50.000$

	SNUM	PNUM	DEPTNUM	QUAN
size	6	7	2	10
val	3000	1000	30	500

$card(DEPT) = 30$

	DEPTNUM	NAME	AREA	MGRNUM
size	2	15	1	7
val	30	30	6	30

a) Bản tóm lược của các quan hệ *SUPPLY* và *DEPT*

$card(SUPPLY_i) = 30.000$

$site(SUPPLY_i) = 1$

	SNUM	PNUM	DEPTNUM	QUAN
size	6	7	2	10
val	1800	1000	20	500

$card(DEPT_i) = 10$

$site(DEPT_i) = 2$

	DEPTNUM	NAME	AREA	MGRNUM
size	2	15	1	7
val	10	10	2	10

b) Bản tóm lược của các mảnh *SUPPLY_i* và *DEPT_i*

Hình 6.1 Ví dụ về các bản tóm lược

2- Ước lượng bản tóm lược của kết quả của các phép toán đại số

Ước lượng bản tóm lược của kết quả của các phép toán đại số sẽ có ích cho việc đánh giá hiệu quả của việc áp dụng các phép toán khác nhau trong tối ưu hóa. Nó cũng có ích cho việc đánh giá các bản tóm lược của các mảnh khi cho trước các bản tóm lược của các quan hệ toàn cục. Cho S là kết quả của việc thực hiện một phép toán một ngôi trên một quan hệ R , và cho T là kết quả của việc áp dụng một phép toán hai ngôi trên hai quan hệ R và S .

a- Phép chọn

1- *Số phần tử (cardinality)*. Chúng ta gán một hệ số chọn (*selectivity*) p cho mỗi phép chọn mà kết quả bao gồm các bộ thỏa mãn điều kiện chọn. Trong các phép chọn đơn giản $attribute = value$ ($A = v$), p có thể được ước lượng là $1/val(A[R])$ với giả sử các giá trị được phân bố đều và giá trị v xuất hiện trong R . Chúng ta có:

$$card(S) = p \times card(R)$$

2- *Kích thước*. Phép chọn không ảnh hưởng đến kích thước của các quan hệ:

$$size(S) = size(R)$$

3- *Các giá trị khác biệt*. Xét một thuộc tính B không có trong điều kiện chọn. Với giả sử B độc lập đối với các điều kiện chọn và đối với sự phân bố đều các giá trị, việc xác định $val(B[S])$ được suy dẫn từ bài toán thống kê sau đây: Cho $n = card(R)$ các đối tượng được phân bố đều trên $m = val(B[R,])$ màu, có bao nhiêu màu khác nhau $c = val(B[R,])$ được chọn nếu chúng ta lấy đúng $r = card(R,)$ đối tượng?

Bài toán này được giải bằng cách sử dụng các phép tính toán gần đúng khác nhau. Ở đây, chúng ta nêu ra một phép tính toán gần đúng trong thực tế của giá trị $c(n, m, r)$:

$$c(n, m, r) = \begin{cases} r & \text{với } r < \frac{m}{2} \\ \frac{r+m}{3} & \text{với } \frac{m}{2} \leq r < 2m \\ m & \text{với } r \geq 2m \end{cases} \quad (6.1)$$

Đôi khi, các giả sử về sự độc lập và về sự phân bố đều của các giá trị là sai, và công thức (6.1) sẽ không được chấp nhận. Đặc biệt, đối với các thuộc tính có trong điều kiện chọn, cần phải đánh giá hoàn toàn khác. Ví dụ, trong trường hợp quan trọng là phép chọn đơn giản trên A , $val(A[S]) = 1$. Mở rộng cho các phép chọn phức tạp hơn thì không khó lắm.

b- Phép chiếu

1- *Số phần tử*: phép chiếu ảnh hưởng đến số phần tử của các toán hạng, bởi vì các giá trị bị trùng sẽ bị loại bỏ trong kết quả. Rất khó đánh giá ảnh hưởng này.

Ba quy tắc sau đây có thể được áp dụng:

- Nếu phép chiếu bao gồm một thuộc tính đơn A , cho

$$card(S) = val(A[R])$$

- Nếu tích $\prod_{A_i \in Attr(S)} val(A_i[R])$ nhỏ hơn $card(R)$, với $Attr(S)$ là các thuộc tính trong kết quả của phép chiếu, cho

$$card(S) = \prod_{A_i \in Attr(S)} val(A_i[R])$$

- Nếu phép chiếu bao gồm một khóa của R , cho

$$card(S) = card(R)$$

Lưu ý rằng, một số hệ thống không thực hiện việc loại bỏ các giá trị bị trùng sau khi thực hiện phép chiếu. Trong trường hợp này, số phần tử của kết quả bằng số lượng phần tử của quan hệ toán hạng.

2- *Kích thước*: Kích thước của kết quả của phép chiếu bị giảm và bằng tổng các kích thước của các thuộc tính trong tập thuộc tính chiếu.

3- *Các giá trị khác biệt*: Các giá trị khác biệt của các thuộc tính chiếu (*projected attribute*) là các giá trị khác biệt trong quan hệ toán hạng.

c- Gom nhóm

Nên nhớ rằng phép $\Psi_{G,AF}$ thực hiện việc phân chia các quan hệ thành các nhóm và định trị các hàm kết hợp trên chúng.

G chỉ ra các thuộc tính dùng để gom nhóm, AF chỉ ra các hàm kết hợp được định trị.

1- *Số phần tử*. Chúng ta nêu ra cận trên của số phần tử của S :

$$\text{card}(S) = \prod_{A_i \in G} \text{val}(A_i[R])$$

Cận trên này tương ứng với trường hợp mỗi giá trị của một thuộc tính cho trước của tập thuộc tính gom nhóm kết hợp với mỗi giá trị của tất cả các thuộc tính khác trong tập thuộc tính gom nhóm.

2- *Kích thước*. Đối với tất cả các thuộc tính A có trong G :

$$\text{size}(R, A) = \text{size}(S, A)$$

Kích thước của S bằng tổng các kích thước của các thuộc tính trong G và AF .

3- *Các giá trị khác biệt*. Đối với tất cả các thuộc tính A trong G :

$$\text{val}(A[S]) = \text{val}(A[R])$$

d- Phép hợp

1- *Số phần tử*. Chúng ta có:

$$\text{card}(T) \leq \text{card}(R) + \text{card}(S)$$

Trường hợp bằng xảy ra khi các bộ bị trùng không bị loại bỏ.

2- *Kích thước*. Chúng ta có:

$$\text{size}(T) = \text{size}(R) = \text{size}(S)$$

bởi vì phép hợp được áp dụng cho các quan hệ thuộc cùng một lược đồ quan hệ.

3- *Các giá trị khác biệt*. Một cận trên là:

$$\text{val}(A[T]) \leq \text{val}(A[R]) + \text{val}(A[S])$$

e- Phép hiệu

1- *Số phần tử*. Chúng ta có:

$$\max(0, \text{card}(R) - \text{card}(S)) \leq \text{card}(T) \leq \text{card}(R)$$

2- *Kích thước*. Chúng ta có:

$$\text{size}(T) = \text{size}(R) = \text{size}(S)$$

bởi vì phép hiệu được áp dụng cho các quan hệ thuộc cùng một lược đồ quan hệ.

3- *Các giá trị khác biệt*. Một cận trên là:

$$\text{val}(A[T]) \leq \text{val}(A[R])$$

f- Phép tích Descartes

1- *Số phần tử*. Chúng ta có:

$$\text{card}(T) = \text{card}(R) \times \text{card}(S)$$

2- *Kích thước*. Chúng ta có:

$$\text{size}(T) = \text{size}(R) + \text{size}(S)$$

3- *Các giá trị khác biệt*. Các giá trị khác biệt của các thuộc tính giống với các giá trị khác biệt trong các quan hệ toán hạng.

g- Phép kết

Xét phép kết $T = R \bowtie_{A=B} S$.

1- *Số phần tử*. Việc đánh giá chính xác số phần tử của T khá phức tạp; chúng ta có thể nêu ra một cận trên cho $\text{card}(T)$ là $\text{card}(T) \leq \text{card}(R) \times \text{card}(S)$, nhưng giá trị này thông thường lớn hơn số phần tử thực sự. Giả sử tất cả các giá trị của A trong R cũng giống với các giá trị của B trong S và ngược lại (với $\text{val}(A[R]) = \text{val}(B[S])$) và các giá trị của cả hai thuộc tính này được phân bố đều trên các bộ của R và S , chúng ta có:

$$\text{card}(T) = (\text{card}(R) \times \text{card}(S)) / \text{val}(A[R])$$

Cũng với giả sử trên, nếu một trong hai thuộc tính, gọi là A , là một khoá của R , thì:

$$\text{card}(T) = \text{card}(S)$$

Các giá trị này có thể được xem là các cận trên nếu các giả sử trên không được thỏa mãn.

2- *Kích thước*. Chúng ta có:

$$\text{size}(T) = \text{size}(R) + \text{size}(S)$$

Trong trường hợp phép kết tự nhiên, vì phép kết tự nhiên cũng loại bỏ một trong các thuộc tính kết, kích thước của thuộc tính này phải được trừ cho kích thước của kết quả.

3- *Các giá trị khác biệt*. Chúng ta có thể nêu ra một cận trên của số giá trị khác biệt được tạo ra bởi một phép kết. Nếu A là một thuộc tính kết, chúng ta có:

$$\text{val}(A[T]) \leq \min(\text{val}(A[R]), \text{val}(B[S]))$$

Nếu A không là một thuộc tính kết, chúng ta có:

$$val(A[T]) \leq val(A[R]) + val(B[S])$$

h- Phép nửa kết

Xét phép nửa kết $T = R \triangleright_{A=B} S$.

1- *Số phần tử*. Việc đánh giá số phần tử của T cũng tương tự như của phép chọn; chúng ta ký hiệu p là hệ số chọn của phép nửa kết, là số lượng các bộ của R có trong kết quả. Chúng ta có:

$$p = val(A[S]) / val(dom(A))$$

với $val(dom(A))$ là số giá trị khác biệt trong miền trị của A . Cho trước p , chúng ta xác định $card(T)$ như sau:

$$card(T) = p \times card(R)$$

2- *Kích thước*. Kích thước của kết quả của phép nửa kết bằng kích thước của toán hạng thứ nhất.

3- *Các giá trị khác biệt*. Số giá trị khác biệt của các thuộc tính, mà chúng không thuộc tập thuộc tính nửa kết, có thể được đánh giá bằng công thức (6.1) với

$$n = card(R), m = val(A[R]) \text{ và } r = card(T).$$

Nếu chỉ có một thuộc tính nửa kết là A , thì:

$$val(A[T]) = p \times val(A[R])$$

Hình 6.1b cho thấy bản tóm lược của các mảnh ngang SUPPLY, và DEPT,. Bởi vì chúng tương ứng với các phép chọn của các quan hệ toàn cục SUPPLY và DEPT theo thứ tự tương ứng, các bản tóm lược được định trị theo các quy tắc áp dụng cho phép chọn, giả sử $p(SUPPLY_i) = 0,6$ và $p(DEPT_i) = 0,33$. Lưu ý rằng, giả sử về sự độc lập của các thuộc tính đối với các điều kiện chọn là không chắc chắn trong việc xác định số giá trị khác biệt của các thuộc tính SNUM, AREA và DEPTNUM. Trong trường hợp này, các giá trị khác biệt được giả sử tỷ lệ với các hệ số chọn của các phép chọn.

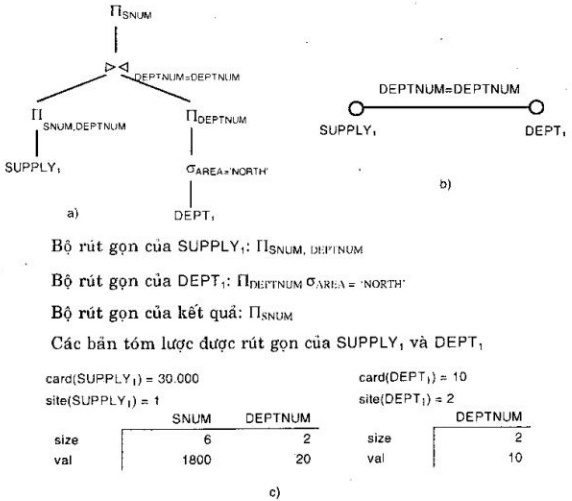
3- Một mô hình cho tối ưu hóa truy vấn

Bây giờ, chúng ta trình bày một mô hình mới của truy vấn thuận tiện hơn so với mô hình cây toán tử để mô tả tối ưu hóa truy vấn. Trong mô hình mới này, chúng ta chỉ đưa vào các phép toán chính cho vấn đề tối ưu hóa truy vấn; chúng ta chỉ xét tập con của các truy vấn thường dùng.

Các phép toán một ngôi là không quan trọng, bởi vì chúng ảnh hưởng đến việc rút gọn các toán hạng của chúng về số phần tử và kích thước, và không cần phải truyền dữ liệu. Do đó các phép toán một ngôi được thực hiện càng sớm càng tốt. Các phép toán một ngôi được thực hiện trên cùng một mảnh sẽ được gom chung lại thành các chương trình, được gọi là các *bộ rút gọn mảnh* (*fragment reducer*). Các bộ rút gọn được áp dụng cho các mảnh trước khi thực hiện các phép toán hai ngôi. Do việc áp dụng chương trình rút gọn (*reducer program*), bản tóm lược mảnh có thể được tính toán lại bằng cách sử dụng các quy tắc trong phần trước.

Các phép toán hai ngôi là quan trọng, bởi vì chúng đòi hỏi phải so sánh hai toán hạng. Do đó, khi các toán hạng không ở cùng một nơi, chúng đòi hỏi phải truyền dữ liệu. Trong chương này, chúng ta không xét phép tích Descartes và phép hiệu vì chúng hiếm có trong các truy vấn của người sử dụng. Chúng ta cũng giả sử rằng phép nối kết không có trong truy vấn; tuy nhiên, chúng được đưa vào trong quá trình tối ưu hóa truy vấn. Do đó, chúng ta chỉ cần xét các phép toán hai ngôi là phép kết và phép hợp.

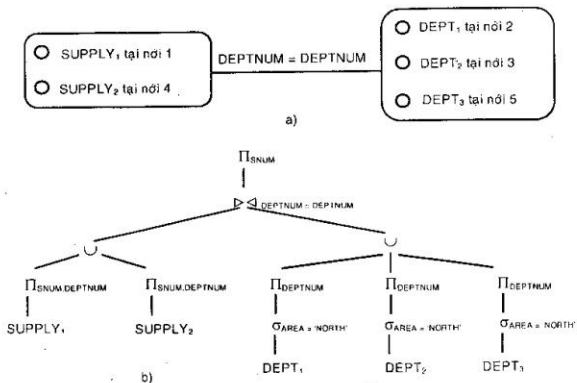
Vì chúng ta giới hạn các phép toán hai ngôi là các phép kết và các phép hợp, trong khi đó các phép toán một ngôi là không thích đáng trong vấn đề tối ưu hóa truy vấn, điều này thuận lợi cho việc giới thiệu một mô hình mới của truy vấn, ở một mức trừu tượng cao hơn. Mô hình này được gọi là *đồ thị tối ưu hóa* (*optimization graph*). Trong các đồ thị tối ưu hóa, các nút biểu diễn các mảnh được rút gọn (*reduced fragment*), các phép kết được biểu diễn bởi các cạnh giữa các nút và có nhãn là điều kiện kết, và các phép hợp được biểu diễn bởi các nút bao (*hypernode*) bao các toán hạng của chúng.



Hình 6.2 Biểu diễn một cây truy vấn thành một đồ thị tối ưu hóa với các chương trình rút gọn và các bản tóm lược của các mảnh

- Cây toán tử của truy vấn Q1
- Đồ thị tối ưu hóa của truy vấn Q1
- Các bộ rút gọn và các bản tóm lược sau khi rút gọn của truy vấn Q1

Hình 6.2a cho thấy cây toán tử của truy vấn Q1 sau khi thực hiện tất cả các phép biến đổi trong chương 5. Đồ thị tối ưu hóa của truy vấn này được chỉ ra trong hình 6.2b. Đồ thị chỉ bao gồm hai nút được nối với nhau bởi một cạnh biểu diễn phép kết. Các chương trình rút gọn của hai mảnh của truy vấn được chỉ ra trong hình 6.2c. Chúng ta cần phải tính lại các bản tóm lược của các mảnh bằng cách áp dụng các chương trình rút gọn trước khi bắt đầu xét vấn đề tối ưu hóa.



Hình 6.3 Các đồ thị tối ưu hóa của truy vấn Q1

a) Đồ thị tối ưu hóa của truy vấn Q1

b) Một cây toán tử của truy vấn Q1

Hình 6.3a cho thấy một ví dụ phức tạp hơn của đồ thị tối ưu hóa, tương ứng với cây toán tử của hình 5.4 được vẽ lại trong hình 6.3b. Cây toán tử này là cây toán tử của truy vấn Q1, trước khi thực hiện các phép biến đổi của chương 5. Đồ thị tối ưu hóa của hình 6.3a cho thấy hai phép hợp được thực hiện trước phép kết.

Việc sử dụng cây toán tử sẽ bị trở ngại trong tối ưu hóa truy vấn, bởi vì mỗi lần các phép toán được đẩy xuống phía các nút lá, thực sự không quan tâm đến việc duy trì các thứ tự bộ phận giữa các nút. Trong khi đó, các đồ thị tối ưu hóa thì đơn giản hơn và độc lập hơn, nghĩa là không có các thứ tự bộ phận được ngầm định như trong cấu trúc cây.

6.1.4 Tóm lược các giả sử dùng cho tối ưu hóa truy vấn phân bố

Ở đây, chúng ta tóm lược các giả sử được sử dụng trong chương này; tóm lược này cũng được sử dụng cho các phần sau.

– Đối với mỗi truy vấn, chúng ta giả sử có một bản thực hiện cho trước và chúng ta làm việc trên các mảnh được định vị.

– Ban đầu, chúng ta chỉ xét các yêu cầu truyền thông và tối thiểu hóa các chi phí truyền thông (phần 6.2.1 và 6.2.2); các trị hoàn truyền thông được tối thiểu hóa trong phần 6.2.3, và một mô hình chi phí toàn diện hơn được nêu ra trong phần 6.2.4.

– Các chi phí truyền thông không phụ thuộc vào các nơi mà giữa các nơi này cần phải truyền thông với nhau.

– Để định trị bản tóm lược khi cần, chúng ta giả sử các giá trị được phân bố đều trên các bộ dữ liệu của các mảnh và các thuộc tính là độc lập nhau.

– Chúng ta không xét các truy vấn có chứa phép tích và phép hiệu.

– Trong phần 6.2, chúng ta khảo sát các **truy vấn kết** (*join query*), nghĩa là các truy vấn mà đồ thị tối ưu hóa chỉ bao gồm các phép kết và không có phép hợp. Giả sử này bị bỏ qua trong phần 6.3, mà chúng ta sẽ đề cập đến trường hợp phức tạp hơn của các truy vấn mà đồ thị tối ưu hóa bao gồm các phép kết và các phép hợp. Lý do để giải quyết riêng biệt các truy vấn kết là vì lớp các truy vấn này đã được nghiên cứu nhiều hơn trong tài liệu và đang có nhiều kỹ thuật để tối ưu hóa chúng; các truy vấn kết thường được gọi là các **truy vấn chọn-chiều-kết** (*select-project-join query*) trong tài liệu, bởi vì rõ ràng các phép toán một ngôi cũng có thể có trong các biểu thức của chúng.

6.1.5 Tầm quan trọng của tối ưu hóa truy vấn trong cơ sở dữ liệu phân bố

Phần này nhấn mạnh tầm quan trọng của tối ưu hóa truy vấn trong môi trường phân bố. Phát triển một chiến lược xử lý truy vấn tốt là điều bắt buộc trong môi trường này, cũng như cần phải tránh các chiến lược xử lý truy vấn tệ hại. Do đó, tính hiệu quả của hệ thống cuối cùng phụ thuộc rất nhiều vào chất lượng của bộ tối ưu hóa truy vấn (*query optimizer*), hoặc trong trường hợp không có bộ tối ưu hóa truy vấn thì nó phụ thuộc vào kỹ năng của người lập trình ứng dụng. Nên nhớ rằng, tất cả các phép biến đổi của các biểu thức truy vấn được nêu trong chương 5 và một số giải thuật của chương này phải trở thành một phần của các bộ tối ưu hóa trong một hệ thống có

cung cấp tính trong suốt phân bố, và dùng làm tài liệu hướng dẫn cho những người lập trình ứng dụng để viết các truy vấn hiệu quả đối với các hệ thống có các mức trong suốt thấp hơn.

Chúng ta lấy truy vấn Q_1 của chương 5 làm ví dụ. Truy vấn cho biết các nhà cung cấp có đơn hàng cung cấp ở phía Bắc của công ty; kết quả cần có tại nơi 2. Chúng ta phân tích các yêu cầu truyền thông cho các chiến lược thực hiện khác nhau của truy vấn này; để đơn giản, chúng ta không để ý đến C_o và D_o trong biểu thức của các chi phí và các trì hoãn. Bàn luận sau đây được dựa trên các cây toán tử, các đồ thị tối ưu hóa và các bản tóm lược của truy vấn Q_1 được chỉ ra trong hình 5.1, 5.4, 5.7, 6.1, 6.2 và 6.3.

Chiến lược 1: Một chiến lược thực hiện đầu tiên, rất tệ hại, bao gồm việc thực hiện các phép toán theo thứ tự được xác định trong truy vấn (và được biểu diễn trong hình 5.4a), không có bất kỳ đơn giản hóa và tối ưu hóa nào. Hình 6.3b cho thấy đồ thị tối ưu hóa của trường hợp này, hình 6.1 cho thấy các bản tóm lược của $SUPPLY_1$ và $DEPT_1$. Giả sử các bản tóm lược của $SUPPLY_2$, $DEPT_2$ và $DEPT_3$ là tương tự nhau, với $card(SUPPLY_2) = 20.000$, $card(DEPT_2) = card(DEPT_3) = 10$. Chúng ta tập hợp tất cả các mảnh tại nơi 2 để thực hiện truy vấn tại đó và việc tập hợp này được thực hiện một cách song song. Do đó, $DEPT_1$ không cần được truyền đi. $SUPPLY_1$ có $30.000 \times 25 \times 8$ bit, và $DEPT_2$ và $DEPT_3$ có $10 \times 25 \times 8$ bit. Do đó, chúng ta có chi phí truyền thông là $10.002.000 \times C_1$ và trì hoãn truyền thông là $6.000.000 \times D_1$ (tương ứng với việc truyền thông của mảnh lớn nhất, $SUPPLY_1$). Nếu D_1 là 10.000 bit / giây, chiến lược thực hiện này cần đến 10 phút.

Chiến lược 2: Một chiến lược thực hiện thứ hai bao gồm việc thực hiện xử lý cục bộ trên các mảnh (nghĩa là áp dụng các bộ rút gọn mảnh) và sau đó gửi tất cả các mảnh được rút gọn đến nơi 2, nơi thực hiện truy vấn. Cây toán tử tương ứng với phép biến đổi truy vấn này được chỉ ra trong hình 6.3b. Đồ thị tối ưu hóa vẫn không thay đổi như trong trường hợp trước và được chỉ ra trong hình 6.3a. $SUPPLY_1$ và $SUPPLY_2$ được rút gọn bởi một phép chiếu có $30.000 \times 8 \times 8$ bit ứng với $SUPPLY_1$ và $20.000 \times 8 \times 8$ bit ứng với $SUPPLY_2$. Các mảnh của $DEPT$ được rút gọn nhiều bởi một phép chọn và một phép chiếu. Chúng có thể không được xét đến trong tính toán các chi phí và các trì hoãn. Do đó,

tổng chi phí truyền thông giảm còn khoảng $3.200.000 \times C_1$ và chi phí trì hoãn giảm còn $1.920.000 \times D_1$. Chúng ta vẫn còn cần khoảng 3 phút để thực hiện truy vấn này.

Chiến lược 3: Một chiến lược thực hiện thứ ba sử dụng nhiều phép đơn giản hóa hơn mà chúng đã được áp dụng cho truy vấn Q_1 trong phần 5.2.5. Cây toán tử được chỉ ra trong hình 6.2a, và đồ thị tối ưu hóa được rút gọn như trong hình 6.2b. Chúng ta nhận thấy chỉ có các mảnh $SUPPLY_1$ và $DEPT_1$ cho ra kết quả của truy vấn. Hơn nữa, chúng ta truyền tất cả các mảnh cần thiết (nghĩa là chỉ có $SUPPLY_1$) đến nơi 2, nơi thực hiện truy vấn. Do đó, tổng chi phí giảm còn $1.920.000 \times C_1$ so với chiến lược trước. Trì hoãn vẫn là $1.920.000 \times D_1$, bởi vì $SUPPLY_2$ không được truyền đi cũng không ảnh hưởng đến trì hoãn, mà trì hoãn được xác định bởi mảnh lớn nhất $SUPPLY_1$.

Các chiến lược 2 và 3 vẫn được xem là các chiến lược thực hiện tệ hại bởi vì chúng di chuyển mảnh $SUPPLY_1$ trước khi rút gọn mảnh này.

Chiến lược 4: Một chiến lược thực hiện thứ tư phản ánh tầm quan trọng có được thông qua tối ưu hóa truy vấn và bao gồm một thứ tự thực hiện khác của các phép toán. Một cách chính xác, mảnh được rút gọn $DEPT_1$ được gửi đến nơi 1, phép kết được thực hiện tại đó, và sau đó kết quả của truy vấn được gửi trở lại nơi 2. Rõ ràng, đồ thị tối ưu hóa cũng giống với đồ thị tối ưu hóa trong chiến lược 3. Vì $DEPT_1$ chỉ bao gồm các thuộc tính kết, phương pháp này thuộc loại phép nửa kết. Để đo lường các lần truyền thông cần thiết, chúng ta cần định trị bán tóm lược của kết quả của phép nửa kết và của phép chọn trên thuộc tính $SNUM$; áp dụng các công thức của phần 6.1.3 mục 2, chúng ta có $card(DEPT_1) = 5$, $card(RESULT) = 1800$ (nghĩa là $card(RESULT) = val(SNUM[SUPPLY_1])$). Do đó, các yêu cầu truyền thông có $1800 \times 2 \times 8$ bit. Chi phí truyền thông là $28.800 \times C_1$ và chi phí trì hoãn là $28.800 \times D_1$; thời gian cần thiết là 2.88 giây.

Trì hoãn của chiến lược 1 lớn hơn 200 lần trì hoãn của chiến lược 4 khi thực hiện cùng một truy vấn. Ví dụ này cho thấy tầm quan trọng của việc chọn một chiến lược thực hiện tốt trong các CSDL phân bố.

6.2 CÁC TRUY VẤN KẾT

Trong phần này, chúng ta giới hạn phân tích các truy vấn kết, nghĩa là các truy vấn mà đồ thị tối ưu hóa chỉ bao gồm các phép kết, các giải thuật được trình bày trong phần này không xét đến một cách tường minh sự phân mảnh của các quan hệ. Do đó, trong phần này, chúng ta không sử dụng khái niệm phân mảnh, và chúng ta sử dụng thuật ngữ chung là quan hệ bao hàm cả các mảnh hoặc các quan hệ không bị phân mảnh và chúng là các nút của các đồ thị tối ưu hóa. Các giải thuật tối ưu hóa có thể được phân loại thành hai lớp:

- Các giải thuật sử dụng các phép nửa kết để rút gọn các quan hệ trước khi truyền chúng.

- Các giải thuật không xét các phép nửa kết.

Trước tiên, chúng ta giới thiệu cơ sở chung của các giải thuật sử dụng các phép nửa kết, và sau đó chúng ta nêu ra hai cách tiếp cận cụ thể sử dụng chúng để tối ưu hóa. Cả hai cách tiếp cận này chỉ xét hàm mục tiêu (*goal function*) là các chi phí truyền thông. Cuối cùng, chúng ta nêu ra một cách tiếp cận để tối ưu hóa truy vấn mà không sử dụng các phép nửa kết, và cũng tính đến các chi phí cục bộ.

Đồ thị tối ưu hóa của một truy vấn kết chỉ bao gồm các nút biểu diễn các quan hệ và các cạnh biểu diễn các phép kết; cách biểu diễn này đã được giới thiệu trong Chương 4 và được gọi là *đồ thị kết* (*join graph*). Trong thực tế, vì chúng ta cho phép có nhiều hơn hai phép kết giữa hai quan hệ bất kỳ, do đó đồ thị kết là một đa đồ thị (*multigraph*).

6.2.1 Sử dụng các chương trình nửa kết cho các truy vấn kết

Chúng ta bắt đầu bằng cách xét một ví dụ đơn giản của một chương trình nửa kết và các chi phí truyền thông của nó. Trong chương 5 phần 5.2 mục 7, một chương trình nửa kết giữa hai quan hệ R và S trên hai thuộc tính A và B được định nghĩa là $(R \triangleright_{A=B} S) \triangleright_{A=B} S$ và tương đương với $R \triangleright_{A=B} S$. Giả sử R được đặt tại nơi r và S được đặt tại nơi s . Khi đó, việc thực hiện chương trình nửa kết tương ứng với:

- 1- Gửi $\Pi_B(S)$ đến nơi r , với chi phí:

$$C_0 + C_1 \times \text{size}(B) \times \text{val}(B[S])$$

- 2- Tính phép nửa kết tại nơi r , với chi phí bằng 0. Cho:

$$R' = R \triangleright_{A=B} S$$

3- Gửi R' đến nơi s , với chi phí:

$$C_o + C_1 \times \text{size}(R) \times \text{card}(R')$$

4- Tính phép kết tại nơi s , với chi phí bằng 0

Chi phí tổng cộng là:

$$C_{SJ} = 2 \times C_o + C_1 \times (\text{size}(B) \times \text{val}(B[S]) + \text{size}(R) \times \text{card}(R'))$$

Phép nửa kết không có tính giao hoán, và rõ ràng chương trình nửa kết khác ($S \triangleright_{B=A} R \triangleright_{B=A} R$) có chi phí khác. Việc so sánh hai chi phí của chúng sẽ giải quyết vấn đề xác định chương trình nửa kết tối ưu cho phép kết của R và S . Lưu ý rằng, hai chương trình cho ra các kết quả được đặt tại các nơi khác nhau. Do đó, cũng nên xét các chi phí truyền thông khác nhau của các kết quả này.

Hơn nữa, điều quan trọng là phải kiểm tra lại chương trình nửa kết có thuận lợi cho việc sử dụng các phép kết như là một cách xử lý truy vấn, mà nó bao gồm việc thực hiện phép kết một cách trực tiếp mà không có các phép nửa kết đi trước. Chi phí sử dụng các phép kết trong cách xử lý truy vấn bao gồm chi phí truyền một trong các toán hạng từ nơi này đến nơi khác. Nếu chúng ta thực hiện phép kết tại nơi s , chi phí tương ứng là:

$$C_{JN} = C_o + C_1 \times \text{size}(R) \times \text{card}(R)$$

So sánh C_{JN} và C_{SJ} , chương trình nửa kết là thuận lợi hơn nếu:

$$C_o + C_1 \times (\text{size}(B) \times \text{val}(B[S]) + \text{size}(R) \times \text{card}(R')) < C_1 \times \text{size}(R) \times \text{card}(R)$$

Trong nhiều trường hợp thì bất đẳng thức trên được thỏa mãn, bởi vì $\text{size}(B)$ và $\text{val}(B[S])$ nhỏ, trong khi đó $\text{card}(R)$ lớn hơn $\text{card}(R')$. Do đó, các giải thuật được giới thiệu trong các phần 6.2.2 và phần 6.2.3 bỏ qua việc sử dụng các phép kết trong cách xử lý truy vấn.

1- Rút gọn các quan hệ bằng cách sử dụng các phép nửa kết

Các phép nửa kết có thể được xem là các **bộ rút gọn**, nghĩa là các phép toán có thể được áp dụng, tương tự như các phép toán một ngôi, để rút gọn số phần tử của các toán hạng của chúng. Rõ ràng, đối với bất kỳ R và S nào đều có thể được nửa kết với nhau, chúng ta có $R \supseteq (R' \equiv R \triangleright S)$. Việc mở rộng đặc tính này cho trường hợp của ba quan hệ cho phép các phép nửa kết giữa mỗi cặp có thể có. Chúng ta có, đối với bất kỳ R, S và T :

$$R \supseteq (R' = R \triangleright \langle (S \triangleright \langle T) \rangle)$$

nhưng cũng có: $R \supseteq (R'' = R \triangleright \langle (T \triangleright \langle S) \rangle)$

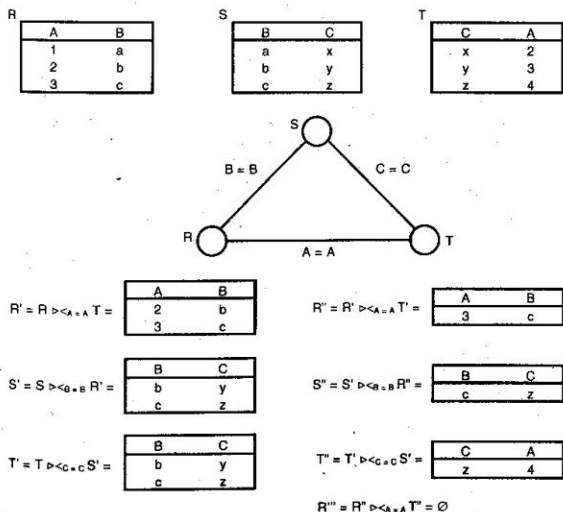
Chúng ta gọi các **chương trình rút gọn** (*reducer program*) cho R là một chuỗi các phép nửa kết, chẳng hạn như R' và R'' .

Cho $RED(Q, R)$ là một tập hợp các chương trình rút gọn có thể được xây dựng cho một quan hệ R cho trước trong một truy vấn Q cho trước. Rõ ràng, có một chương trình rút gọn, là phần tử của $RED(Q, R)$, rút gọn nhiều hơn so với tất cả các phần tử khác; chúng ta gọi chương trình rút gọn này là **bộ rút gọn hoàn toàn** (*full reducer*) của R . Do đó, một vấn đề quan tâm là xác định các bộ rút gọn hoàn toàn cho các quan hệ của một truy vấn. Tuy nhiên, lưu ý rằng điều này không giải quyết vấn đề xác định chiến lược tốt nhất cho một truy vấn, bởi vì chi phí của một chương trình rút gọn hoàn toàn, về các lần truyền thông, có thể lớn hơn so với các ưu điểm của nó.

Việc xác định các bộ rút gọn hoàn toàn là một công việc khó khăn. Nói chung, ví dụ hình 6.4 cho thấy rằng không thể nêu ra một chiều dài giới hạn của chương trình rút gọn, nghĩa là số phép nửa kết có trong chương trình. Trong ví dụ này, mỗi quan hệ có thể được rút gọn thành quan hệ rỗng. Tuy nhiên, điều này cần phải có một chương trình rút gọn mà chiều dài tăng tuyến tính theo số bộ của các quan hệ, một cách chính xác, nếu m là số bộ thì chiều dài là $3 \times (m - 1)$. Lưu ý rằng, cần phải lưu trữ các kết quả của mỗi phép nửa kết bộ phận vào trong các quan hệ tạm thời.

Theo ví dụ trên, một câu hỏi sau đây được đặt ra là: có thể nêu ra chiều dài giới hạn của bộ rút gọn hoàn toàn, ít nhất là cho một lớp các truy vấn? Câu trả lời cho câu hỏi này là có thể: tồn tại một chiều dài cho các truy vấn mà đồ thị kết của nó là một cây, được gọi là **truy vấn cây** (*tree query*) (như chúng ta đã biết, cây là trường hợp đặc biệt của đồ thị không có các chu trình). Chiều dài giới hạn của bộ rút gọn hoàn toàn là $n - 1$, với n là số nút của cây. Một định lý suy diễn một cách trực quan từ ví dụ của hình 6.4 phát biểu rằng, trong trường hợp tổng quát của các **truy vấn vòng** (*cyclic query*), chiều dài của bộ rút gọn tốt nhất bị giới hạn tuyến tính bởi số bộ của một số quan hệ

trong truy vấn. Hơn nữa, không có bảo đảm nào về tính hiệu quả của các chương trình rút gọn. Nghĩa là bộ rút gọn tốt nhất không nhất thiết rút gọn mỗi quan hệ toán hạng thành các bộ mà chúng sẽ tạo nên kết quả cuối cùng. Chẳng hạn, trong một phép kết mà nó tạo ra kết quả là một quan hệ rỗng, có thể không tìm ra bộ rút gọn nào cho bất kỳ các toán hạng mà nó trả về một quan hệ rỗng. Do đó, ý tưởng về một chương trình rút gọn hoàn toàn là không chắc chắn trong ngữ cảnh của các truy vấn vòng.



Hình 6.4 Một chương trình rút gọn hoàn toàn cho quan hệ R

Bây giờ, chúng ta cần giải quyết đồ thị kết nào có thể được xem là cây. Việc xét đồ thị có các chu trình có thể dẫn đến các cách giải quyết sai lầm, bởi vì có nhiều trường hợp các chu trình có thể bị phá mất mà không làm thay đổi nghĩa của truy vấn. Các trường hợp này là:

- Trong chu trình $(R.A = S.B)$, $(S.B = T.C)$, $(T.C = R.A)$, trong đó R, T, S là tên quan hệ và A, B, C là tên thuộc tính, bất kỳ một trong các cạnh có thể bị bỏ, cũng như bất kỳ cạnh nào đều có thể được tạo ra từ các cạnh còn lại nhờ vào tính bắc cầu.

- Trong chu trình $(R.A = S.B)$, $(S.B = T.C)$, $(T.C = R.D)$, chúng ta có thể thay thế $(T.C = R.D)$ thành $(R.A = R.D)$ bởi vì, từ tính bắc cầu, $T.C$ phải bằng $R.A$. Do đó, đồ thị chỉ chứa hai cạnh (R, S) và (S, T) và không có chu trình, bởi vì một mệnh đề liên quan hệ (*interrelation clause*) đã được thay thế bởi một mệnh đề nội quan hệ (*intrarelational clause*).

Một định lý khẳng định rằng việc kiểm tra một đồ thị truy vấn có thể được rút gọn thành một cây truy vấn bởi các loại đơn giản ở trên sẽ có độ phức tạp tuyến tính theo số nút. Từ đó, việc tìm kiếm các bộ rút gọn hoàn toàn, mà chúng cũng là các chiến lược thực hiện truy vấn tốt, có xác suất thành công cao nếu chúng ta giải quyết các truy vấn cây hơn là các truy vấn vòng.

Có một phương pháp để biến đổi các truy vấn vòng thành các truy vấn cây bằng cách sử dụng các **phép nửa kết tổng quát hóa** (*generalized semi-join*). Phép nửa kết tổng quát hóa, ký hiệu là GSJ , được định nghĩa là:

$$R \text{ GSJ}_{X,F} S = \Pi_{Attr(R),X} (R \bowtie_F S)$$

trong đó $Attr(R)$ là các thuộc tính của R và X là tập thuộc tính của S .

Lưu ý rằng kết quả của phép nửa kết tổng quát hóa có nhiều thuộc tính hơn R . Các phép nửa kết tổng quát hóa có thể được áp dụng thuận lợi cho các truy vấn vòng.

Một lần nữa, xét đồ thị kết có chu trình của hình 6.4, các cạnh tương ứng với các phép kết:

$$R.B = S.B, S.C = T.C, T.A = R.A.$$

Phương pháp bao gồm việc chọn một cạnh của đồ thị mà khi loại bỏ cạnh này sẽ biến đổi truy vấn thành một truy vấn cây (chọn cạnh (S, T) tương ứng với phép kết trên thuộc tính C), và gửi thuộc tính kết C của quan hệ T cho các quan hệ khác. Việc gửi thuộc tính C tương ứng với việc thực hiện một phép nửa kết tổng quát hóa, mà thuộc tính X của định nghĩa tương ứng với thuộc tính C . Trong ví dụ đơn giản của hình 6.4, việc sử dụng một phép nửa kết tổng quát hóa đủ để nhận thấy rằng kết quả của phép kết là rỗng, đó là:

$$R' = R \text{ GSJ}_{C,A=A} T$$

Bàn luận ở trên cho thấy điểm bắt đầu hợp lý để thực hiện một truy vấn bao gồm việc xác định các chương trình rút gọn dùng các phép nửa kết cho mỗi quan hệ liên quan ở trong truy vấn. Do đó, điều thích đáng là cần phải biết tồn tại bao nhiêu chương trình như vậy và độ phức tạp của giải thuật để tìm kiếm một chương trình tốt nhất trong các chương trình này. Thật không may có sự bùng nổ tổ hợp (*combinational explosion*) của các chương trình khi số quan hệ nhiều. Bài toán này thuộc NP-đầy đủ (*NP-complete*), và chỉ có một lớp giới hạn các truy vấn chấp nhận giải pháp đa thức (*polynomial solution*). Đây là một lý do chính đáng để không xét các cách tiếp cận vét cạn (*exhaustive approach*) để chọn các chương trình nửa kết, và để tập trung vào các giải thuật mang tính kinh nghiệm (*heuristic algorithm*).

6.2.2 Xác định các chương trình nửa kết trong SDD-1

Trong cách tiếp cận SDD-1, các phép nửa kết được sử dụng để rút gọn số phần tử của các quan hệ. Sau đó, tất cả các quan hệ được tập hợp lại tại cùng một nơi mà nơi này thực hiện truy vấn. Giải thuật không xét chi phí của lần truyền thông cuối cùng để truyền kết quả từ nơi xử lý đến nơi gốc của truy vấn.

Cách tiếp cận SDD-1 bao gồm một giải thuật cơ bản để ban đầu xác định một giải thuật không hiệu quả nhưng khả thi và sau đó sử dụng các tiêu chuẩn hậu tối ưu hóa (*postoptimization criterion*) để cải thiện nó.

1- Giải thuật SDD-1 cơ bản

Giải thuật SDD-1 cơ bản được dùng để xây dựng các chương trình rút gọn cho các quan hệ. Các bộ rút gọn bao gồm các phép toán một ngôi và các phép nửa kết, mà các phép toán này được chọn dựa trên cơ sở chi phí của chúng. Khi một phép toán mới được đưa vào trong bộ rút gọn của một quan hệ, bản tóm lược của quan hệ này bị thay đổi. Rõ ràng, các phép toán một ngôi không có chi phí và chúng trở thành một phần của các chương trình rút gọn. Thứ tự xuất hiện của chúng trong các chương trình rút gọn là không quan trọng, bởi vì khi giải thuật được áp dụng, tất cả các phép toán cục bộ sẽ được đưa vào các bộ rút gọn, và các bản tóm lược sẽ bị thay đổi một cách tương ứng.

Chúng ta xét phép nửa kết $R \triangleright_{A=B} S$. Phép này không có chi phí khi R và S được lưu trữ tại cùng một nơi. Khi các nơi của R và S là khác nhau, chúng ta nhớ rằng chi phí có thể được đánh giá như sau:

$$\text{cost}(R \triangleright_{A=B} S) = C_0 + \text{val}(B[S]) \times \text{size}(B) \times C_1$$

Lợi ích của chương trình nửa kết có thể được đánh giá như sau:

$$\text{benefit}(R \triangleright_{A=B} S) = (1 - p) \times \text{size}(R) \times \text{card}(R) \times C_1$$

với p là hệ số chọn của phép nửa kết. Cho một tập hợp gồm tất cả các phép nửa kết có thể có, các giá trị này cho phép xây dựng một tập P các **phép nửa kết có ích** (*profitable semi-join*), nghĩa là các phép nửa kết mà lợi ích lớn hơn chi phí. Vòng lặp chính của giải thuật sẽ chọn thứ tự của ứng dụng của các phép nửa kết có ích.

Ở mỗi lần lặp vòng, giải thuật sẽ chọn **phép nửa kết có ích nhất** trong P . Phép nửa kết này trở thành một phần của bộ rút gọn của R ; bản tóm lược của R và chi phí của các phép nửa kết dùng các thuộc tính của R sẽ bị thay đổi. Hơn nữa, tất cả các phép nửa kết khác đã được phân tích để xem xét nếu một số phép toán này trở nên có ích. Các lần lặp vòng được kết thúc khi các phép nửa kết có ích đã được vét hết. Trong ví dụ của hình 6.4, việc thực hiện lặp lại của phép nửa kết có thể có ích cho các truy vấn vòng. Tuy nhiên, điều này có thể dẫn đến các giải pháp không hiệu quả với các bộ rút gọn rất dài. Có thể tránh điều nguy hiểm này bằng cách đặt ra ràng buộc là mỗi phép nửa kết không được thực hiện quá một lần.

Cuối cùng, giải thuật chọn nơi thích hợp nhất để tập hợp các quan hệ rút gọn. Điều này được thực hiện một cách đơn giản bằng cách đánh giá nơi mà nó đòi hỏi số lần truyền thông ít nhất.

Lưu ý rằng, ở mỗi lần lặp vòng cũng có thể chọn **phép nửa kết ít tổn kém nhất** thay vì chọn phép nửa kết thích hợp nhất. Việc chọn lựa này cũng hợp lý, bởi vì phép nửa kết có ích nhất, gọi là SJ , có thể hơi tổn kém và việc đưa nó vào bộ rút gọn tương ứng với việc tốn thêm chi phí của nó. Nếu các phép nửa kết khác ít tổn kém hơn được thực hiện trước tiên thì chi phí của SJ có thể bị giảm ở các lần lặp vòng sau.

Bây giờ, chúng ta có thể tóm lược giải thuật như sau:

1- **Bắt đầu:** Cho trước một đồ thị G , thực hiện tất cả các phép rút gọn cục bộ cho các quan hệ xuất hiện trong G .

2- *Phương pháp*: Trong khi có các phép nửa kết có ích, đưa phép nửa kết có ích nhất hoặc phép nửa kết ít tổn kém nhất vào chương trình rút gọn của quan hệ để áp dụng phép toán này. Đánh giá lại các lợi ích và các chi phí của các phép nửa kết bị ảnh hưởng.

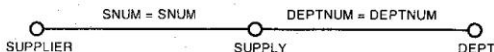
3- *Kết thúc*: Chọn nơi có ít lần truyền thông để tập hợp tất cả các quan hệ.

Ví dụ 6.1 Xét vấn đề tối ưu hóa được xác định trong hình 6.5a, mà hình này cho thấy đồ thị kết, các bản tóm lược của các quan hệ và một số giả sử cơ bản trên các giá trị thuộc tính. Giả sử các bản tóm lược của hình 6.5a chỉ đề cập đến hiệu quả của các bộ rút gọn cục bộ (nghĩa là các bộ rút gọn bao gồm các phép toán một ngôi), để cho chúng ta có thể xem chúng là điểm bắt đầu cho giải thuật tối ưu hóa SDD-1.

Có bốn phép nửa kết có thể có, được chỉ ra trong hình 6.5b, nhưng chỉ có phép nửa kết là có ích để rút gọn quan hệ SUPPLY. Hình 6.5c cho thấy diễn biến tính toán số phần tử, kích thước và các giá trị khác biệt của các quan hệ sau khi chọn một phép nửa kết mới ở mỗi lần lặp vòng, và các biến đổi của các chi phí và lợi ích của các phép nửa kết khác. Để cho đơn giản, giả sử tất cả các giá trị của SNUM trong SUPPLIER và của DEPTNUM trong DEPT đều có trong SUPPLY. Điều này nhằm để đơn giản cho việc đánh giá các hệ số chọn $p(p_i)$. Khi một phép nửa kết được chọn, nó sẽ bị loại bỏ khỏi tập hợp các phép nửa kết được xét ở lần lặp vòng kế tiếp.

Sau hai lần lặp vòng đầu tiên, giải thuật cơ bản bao gồm hai phép nửa kết có thể có của hình 6.5b trong bộ rút gọn của SUPPLY. Kết quả của việc rút gọn quan hệ SUPPLY làm cho phép nửa kết thứ ba SUPPLIER $\bowtie$ SUPPLY trở nên có ích và nó được chọn trong lần lặp vòng thứ ba.

Cuối cùng, chọn một nơi để thực hiện truy vấn. Các chi phí truyền thông liên quan với mỗi lần chọn được chỉ ra trong hình 6.5c. Nơi 1 được chọn bởi vì chi phí của nó là nhỏ nhất.

Đồ thị kết*Các bản tóm lược*

SUPPLIER
 card(SUPPLIER) = 200
 site(SUPPLIER) = 1

	SNUM	NAME
size	4	20
val	200	200

SUPPLY
 card(SUPPLY) = 5000
 site(SUPPLY) = 2

	SNUM	DEPTNUM
size	4	2
val	1000	100

DEPT
 card(DEPT) = 20
 site(DEPT) = 3

	DEPTNUM	AREA
size	2	3
val	20	5

Các giả sử

Tất cả các giá trị của SNUM trong SUPPLIER đều có trong SUPPLY

Tất cả các giá trị của DEPTNUM trong DEPT đều có trong SUPPLY

$C_0 = 0, C_1 = 1$

a) Định nghĩa vấn đề tối ưu hóa

Các phép nửa kết	Hệ số chọn	Lợi ích	Chi phí
p1: SUPPLY $\bowtie$ SUPPLIER	$p(p1) = 0,2$	$0,8 \times 6 \times 5000$	4×200
p2: SUPPLY $\bowtie$ DEPT	$p(p2) = 0,2$	$0,8 \times 6 \times 5000$	2×20
p3: SUPPLIER $\bowtie$ SUPPLY	$p(p3) = 1$	-	4×1000
p4: DEPT $\bowtie$ SUPPLY	$p(p4) = 1$	-	2×100

Các phép nửa kết có ích: p1, p2

b) Mô tả tất cả các phép nửa kết có thể có

Lần lập vòng 1: chọn p2

Ảnh hưởng trên bản tóm lược của SUPPLY

card(SUPPLY) = 1000
 site(SUPPLY) = 2

	SNUM	DEPTNUM
size	4	2
val	666 ↑	20

↑ $c(n, m, r)$ được sử dụng cho SNUM,

với $n = 5000, r = 1000, m = \text{val}(\text{SNUM}[\text{SUPPLY}]) = 1000$

Ảnh hưởng trên các phép nửa kết khác	Hệ số chọn	Lợi ích	Chi phí
p1: SUPPLY ▷< SUPPLIER	$p(p1) = 0.2$	$0.8 \times 6 \times 1000$	4×200
p3: SUPPLIER ▷< SUPPLY	$p(p3) = 0.666 \uparrow$	$0.333 \times 24 \times 200$	4×666
p4: DEPT ▷< SUPPLY	$p(p4) = 1 \downarrow$	-	2×20

↑ Bởi vì 666 giá trị trong 1000 giá trị của SNUM đã được chọn ra.

↓ Bởi vì tất cả giá trị DEPTNUM của SUPPLY đều có trong DEPT.

Các phép nửa kết có ích: p1, p3

Lần lập vòng 2: chọn p1

Ảnh hưởng trên bản tóm lược của SUPPLY

$\text{card}(\text{SUPPLY}) = 200$

$\text{site}(\text{SUPPLY}) = 2$

	SNUM	DEPTNUM
size	4	2
val	133	20 ↑

↑ $c(n, m, r)$ được sử dụng cho DEPTNUM,

với $n = 1000$, $r = 200$, $m = \text{val}(\text{DEPTNUM}[\text{SUPPLY}]) = 20$

Ảnh hưởng trên các phép nửa kết khác	Hệ số chọn	Lợi ích	Chi phí
p3: SUPPLIER ▷< SUPPLY	$p(p3) = 0.666 \uparrow$	$0.333 \times 24 \times 200$	4×133
p4: DEPT ▷< SUPPLY	$p(p4) = 1$	-	2×20

↑ Trong việc rút gọn DEPT, hệ số chọn là 1, bởi vì tất cả giá trị DEPTNUM của SUPPLY đều có trong DEPT.

Các phép nửa kết có ích: p3

Lần lập vòng 3: chọn p3

Ảnh hưởng trên bản tóm lược của SUPPLIER

$\text{card}(\text{SUPPLIER}) = 123$

$\text{site}(\text{SUPPLIER}) = 1$

	SNUM	NAME
size	4	20
val	133	133

Ảnh hưởng trên các phép nửa kết khác	Hệ số chọn	Lợi ích	Chi phí
p4: DEPT $\triangleright$ < SUPPLY	p(p4) = 1	-	2 x 20

Không tồn tại các phép nửa kết có ích khác, do đó kết thúc việc chọn phép nửa kết

Chọn nơi để tập hợp tất cả các quan hệ

$$\text{Cost}(\text{nơi 1}) = 6 \times 200 + 5 \times 20 = 1300$$

$$\text{Cost}(\text{nơi 2}) = 133 \times 24 + 5 \times 20 = 3292$$

$$\text{Cost}(\text{nơi 3}) = 133 \times 24 + 6 \times 200 = 4392$$

Nơi 1 được chọn.

c) Giải thuật SDD-1 cơ bản

Hậu tối ưu hóa

Bởi vì phép nửa kết p3 chỉ ảnh hưởng đến việc rút gọn quan hệ SUPPLIER mà quan hệ này đặt tại nơi 1, p3 không còn có ích nữa.

Tóm tắt

DEPT được gửi đến nơi 1 mà không có rút gọn bằng phép nửa kết, với chi phí là 100. SUPPLY được rút gọn bởi hai phép nửa kết với SUPPLIER và DEPT, với chi phí là 840, sau đó được gửi đến nơi 1, với chi phí là 1200. Chi phí truyền thông tổng cộng là 2140.

d) Hậu tối ưu hóa và tóm tắt

Hình 6.5 Tối ưu hóa truy vấn bằng giải thuật SDD-1

2- Hậu tối ưu hóa

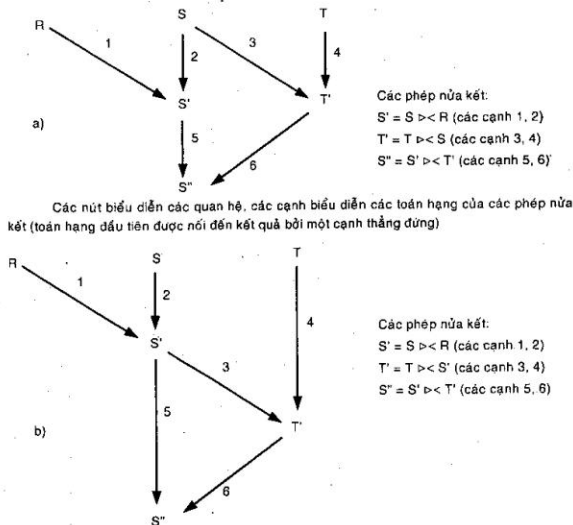
Giải thuật SDD-1 áp dụng một phương pháp mang nhiều tính kinh nghiệm tham lam (*greedy heuristic*), bởi vì ở mỗi lần lặp vòng, nó tìm kiếm cách cải tiến tốt nhất mà không quan tâm đến hậu quả cho các lần lặp vòng sau. Để cải tiến giải pháp này, chúng ta có thể thực hiện hậu tối ưu hóa (*postoptimization*). Hậu tối ưu hóa tuân theo hai tiêu chuẩn sau đây:

- Loại bỏ các phép nửa kết mà tác dụng của chúng là rút gọn các quan hệ đã có tại nơi được chọn để thực hiện truy vấn.

- Trì hoãn các phép nửa kết tốn kém $R \triangleright < S$ sau khi rút gọn S bởi các phép nửa kết khác; điều này đòi hỏi phải thay đổi thứ tự thực hiện của các phép nửa kết.

Hậu tối ưu hóa đầu tiên có thể được áp cho ví dụ 6.1.

Ví dụ 6.1 (tiếp theo) Phép nửa kết $SUPPLIER \triangleright< SUPPLY$ bị tạm dừng, bởi vì quan hệ $SUPPLIER$, sẽ bị rút gọn bởi phép nửa kết, được lưu trữ tại nơi 1 mà nơi này tập hợp tất cả các quan hệ. Kết quả cuối cùng của toàn bộ tối ưu hóa truy vấn $Q1$ được chỉ ra trong hình 6.5d.



Hình 6.6 Hậu tối ưu hóa trong giải thuật SDD-1

- a) Các chương trình nửa kết trước khi áp dụng tiêu chuẩn 2 của hậu tối ưu hóa
 b) Các chương trình nửa kết sau khi áp dụng tiêu chuẩn 2 của hậu tối ưu hóa

Hậu tối ưu hóa thứ hai được thực hiện bằng cách trong các lần lặp vòng chính của giải thuật cơ bản, chúng ta xây dựng một đồ thị luồng (*flow graph*) chỉ ra thứ tự chọn các phép nửa kết. Các quan hệ rút gọn tương ứng với các nút và các phép nửa kết tương ứng với các cạnh của đồ thị luồng. Giải thuật hậu tối ưu hóa bao gồm việc chọn các phép nửa kết tốn kém trong đồ thị luồng và trì hoãn chúng nếu có thể được.

Cho $\langle T, S \rangle$ là một cung của đồ thị luồng mô tả một phép nửa kết $T \triangleright S$ được thực hiện, và cho S' là một nút bất kỳ biểu diễn việc rút gọn của S . Hậu tối ưu hóa chính là việc thay thế $\langle T, S \rangle$ bởi $\langle T, S' \rangle$, với điều kiện nó không tạo ra các chu trình trong đồ thị (một đồ thị có các chu trình tương ứng với một chương trình không khả thi). Cách đơn giản hóa này được chỉ ra trong hình 6.6. Ưu điểm của cách đơn giản hóa này bao gồm hai phần:

- Một số ít bộ được gửi từ nơi S đến nơi T , bởi vì S được rút gọn thành S' .

- Hệ số chọn của phép nửa kết $T \triangleright S$ được tăng lên bởi kết quả của phép nửa kết $S \triangleright R$ trước đó.

6.2.3 Xác định các chương trình nửa kết bằng các giải thuật Apers, Hevner và Yao

Một cách tiếp cận khác mang tính kinh nghiệm là cơ sở của các giải thuật được trình bày trong phần này. Cách tiếp cận này dựa trên việc phân rã truy vấn thành các truy vấn đơn giản hơn mà chúng có thể được giải quyết một cách dễ dàng và tối ưu. Do đó, nhiều vấn đề nhỏ hơn được giải quyết một cách độc lập, và sau đó các kết quả của chúng được tích hợp thành các chiến lược đầy đủ (có thể không theo một cách tối ưu). Theo Apers, Hevner và Yao, cách tiếp cận này tập trung vào các nguyên tắc hơn là vào các chi tiết của các giải thuật.

Cách tiếp cận AHY (Apers-Hevner-Yao) thích hợp cho cả hai tối thiểu hóa chi phí và trì hoãn; bởi vì chúng ta đã mô tả một phương pháp để tối thiểu hóa chi phí (giải thuật SDD-1), trong phần này chúng ta chỉ giới thiệu tối thiểu hóa trì hoãn. Cách tiếp cận AHY bao gồm việc thực hiện xử lý cục bộ trên các quan hệ, sau đó rút gọn các quan hệ bằng cách sử dụng các phép nửa kết, và cuối cùng thực hiện đánh giá truy vấn tại một nơi, được gọi là *nơi thực hiện* (execution site) của truy vấn. Việc xác định một chiến lược xử lý truy vấn trong cách tiếp cận này bao gồm việc đưa ra các *hế hoạch* (schedule) cho mỗi quan hệ trong đồ thị kết.

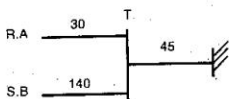
Kế hoạch của quan hệ T là một cây ưu tiên (precedence tree) mà trong đó tất cả các nút, ngoại trừ nút gốc và nút con duy nhất của nút gốc, tương ứng với các thuộc tính của các quan hệ. Chúng được ghi

nhãn là $R.A$ (với R là tên quan hệ và A là tên thuộc tính). Nút đặc biệt được nối trực tiếp với nút gốc của cây tương ứng với quan hệ T và được ghi nhãn là T . Nút gốc biểu diễn nơi thực hiện truy vấn và nó không được ghi nhãn.

Mỗi cung nối một nút bất kỳ khác nút gốc với nút T tương ứng với một phép nửa kết được áp dụng cho T .

Ví dụ, cung $\langle R.A, T \rangle$ biểu diễn một phép nửa kết $T \triangleright_{<A} R$ được thực hiện. Khi các kế hoạch có chiều sâu lớn hơn 2, điều này có nghĩa là các chuỗi phép nửa kết được sử dụng; ví dụ cung $\langle U.C, R.A \rangle$ biểu diễn phép nửa kết $R \triangleright_{<C} U$ được thực hiện để rút gọn R trước khi chiếu nó trên thuộc tính A và sử dụng nó trong các phép nửa kết theo sau. Các cạnh được ghi nhãn với các trị hoãn truyền dữ liệu cần thiết để thực hiện các phép nửa kết tương ứng; cạnh đặc biệt giữa T và nút gốc được ghi nhãn là trị hoãn cần thiết để truyền quan hệ rút gọn T đến nơi thực hiện.

Ví dụ, xét kế hoạch cho quan hệ T được biểu diễn trong hình 6.7. T được rút gọn bằng cách sử dụng phép nửa kết với R và S trên các thuộc tính A và B tương ứng. Sau đó, các bộ của T đã được chọn bởi phép nửa kết, được gửi đến nơi thực hiện truy vấn.



Hình 6.7 Kế hoạch cho quan hệ T

Kế hoạch tương ứng với:

- Tổng chi phí là 205 đơn vị (là tổng của tất cả các chi phí)
- Tổng trị hoãn là 185 đơn vị (xét đường đi dài nhất đến nút gốc)

Chúng ta biểu thị *kế hoạch đơn giản* (simple schedule) cho T là một kế hoạch chỉ bao gồm hai nút, T và nút gốc, và một cạnh giữa chúng, được ghi nhãn bởi chi phí truyền T mà không có rút gọn. Kế hoạch này tương ứng với việc gửi trực tiếp quan hệ T đến nơi thực hiện. Hệ số chọn cố định, được gọi là *hệ số chọn đầu vào* (incoming selectivity), được liên kết với mỗi thuộc tính và có thể được sử dụng để thực hiện các phép nửa kết. Ví dụ, cho hệ số chọn đầu vào của $R.A$ là p_A và của $S.B$ là p_B . Vì phép nửa kết được áp dụng cho T , với giả sử hai thuộc tính độc lập nhau, hệ số chọn đầu vào cho T là $p_A \times p_B$.

Để đơn giản, trong cách tiếp cận AHY, tất cả các quan hệ được giả sử đặt tại các nơi khác nhau; việc loại bỏ giả sử này đòi hỏi cũng xét đến sự định vị của các quan hệ trong giải thuật, bởi vì các phép nửa kết giữa các quan hệ tại cùng một nơi không có tổn chi phí.

1- Các truy vấn đơn giản

Truy vấn đơn giản (*simple query*) là một truy vấn mà sau khi xử lý cục bộ ban đầu thì các quan hệ chỉ chứa một thuộc tính mà thuộc tính này được dùng để kết chúng. Các truy vấn đơn giản được sử dụng để phân rã vấn đề. Chúng có thể được giải quyết một cách tối ưu bởi các giải thuật rất đơn giản. Lưu ý rằng, các truy vấn đơn giản chắc chắn phải thuộc về lớp các truy vấn cây, ngay cả khi đồ thị kết của chúng có chu trình, bởi vì luôn luôn có thể loại bỏ một số cạnh của chúng và rút gọn chúng thành các truy vấn cây.

Chúng ta liên kết tổng số byte của R_i với mỗi quan hệ R_i xuất hiện trong đồ thị kết, được ký hiệu là s_i :

$$s_i = \text{card}(R_i) \times \text{size}(R_i)$$

Giải thuật PARALLEL diễn biến như sau:

1- *Bắt đầu*: Xét các quan hệ theo thứ tự tăng của s_i ; để đơn giản chúng ta giả sử chỉ số quan hệ i phản ánh thứ tự này (tức $i < j$ nghĩa là $s_i < s_j$). Chọn kế hoạch đơn giản cho R_1 .

2- *Phương pháp*: Ở lần lặp vòng j ($j > 1$), xây dựng kế hoạch cho R_j bằng cách chọn kế hoạch với trị hoãn tối thiểu giữa:

- a- Kế hoạch đơn giản cho R_j
- b- Đối với mỗi $i < j$, kế hoạch được xây dựng là sự kết hợp của:
 - Kế hoạch được chọn cho R_j ở các lần lặp vòng trước đó, mà trong đó chúng ta đặt quan hệ R_j thay thế cho nút gốc
 - Tất cả các kế hoạch được chọn cho R_k ở các lần lặp vòng trước đó (với $k < i$) mà trong đó chúng ta cũng đặt quan hệ R_j thay thế cho nút gốc
 - Cạnh giữa R_j và nút gốc, được ghi nhãn bởi chi phí truyền thông của R_j được rút gọn bởi các hệ số chọn đầu vào của tất cả các phép nửa kết được áp dụng cho nó.

Lý do của bước bắt đầu là kế hoạch đơn giản chắc chắn phải là kế hoạch tốt nhất có thể có cho quan hệ nào mà có kế hoạch đơn giản ngắn nhất. Do đó, quan hệ R_1 được gửi đến nút thực hiện mà không có rút gọn. Đối với các quan hệ $R_2, \dots, R_n$, có thể cải tiến kế hoạch đơn giản bằng cách sử dụng các phép nửa kết.

Lý do của quy tắc (b) là nếu chúng ta xem kế hoạch của R_i như là một phần của kế hoạch của R_j , thì sự kết hợp của nó với bất kỳ kế hoạch nào trong các kế hoạch của R_k , mà các kế hoạch này được xây dựng ngắn hơn, sẽ phát sinh một hệ số chọn đầu vào cộng thêm cho R mà không làm tăng trì hoãn, và do đó phải được đưa vào.

Trong việc định trị các bản tóm lược của các quan hệ, giải thuật AHY giả sử các thuộc tính kết độc lập nhau, và việc định trị các hệ số chọn đầu vào do bởi có nhiều phép nửa kết trên cùng thuộc tính là tích của các hệ số chọn đầu vào.

Ví dụ 6.2 Chúng ta minh họa giải thuật PARALLEL bằng một ví dụ có thêm một quan hệ mới:

PROJECT (SNUM, DEPTNUM, PROJNUM)

Xét truy vấn sau đây:

II (SUPPLIER $\bowtie$ SUPPLY $\bowtie$ PROJECT)
SUPPLIER.NAME, PROJNUM

Các bản tóm lược của các quan hệ sau khi xử lý ban đầu và đồ thị kết của truy vấn này được chỉ ra trong hình 6.8a. Các bản tóm lược bao gồm các hệ số chọn đầu vào của các thuộc tính, nghĩa là hệ số chọn p của các phép nửa kết có thể được áp dụng nếu các thuộc tính này được truyền đi. Mặc dù đồ thị kết có chu trình, dễ thấy rằng đây là một truy vấn cây bởi vì bất kỳ cạnh nào được ghi nhãn là SNUM đều có thể bị loại bỏ.

Bây giờ, chúng ta xét các thuộc tính kết một cách riêng biệt. Do đó, chúng ta biến đổi truy vấn này thành hai truy vấn con đơn giản. Để dễ ký hiệu, chúng ta không nêu ra tên thuộc tính tương ứng trong các kế hoạch của các truy vấn đơn giản. Trước tiên, chúng ta áp dụng giải thuật PARALLEL cho truy vấn đơn giản được xây dựng đối với thuộc tính SNUM. Chúng ta bắt đầu bằng cách sắp thứ tự các quan hệ SUPPLY, PROJECT và SUPPLIER, mà chúng đều có thuộc tính SNUM, theo thứ tự tăng dần của các giá trị của s_i (xem bảng trong hình 6.8a); lưu ý rằng:

$$S_i = \text{size}(\text{SNUM}) \times \text{val}(\text{SNUM})$$

trong tất cả ba quan hệ. Sau đó, chúng ta xây dựng các kế hoạch đơn giản cho các quan hệ nhỏ hơn PROJECT và SUPPLY, tương ứng với bước bắt đầu của giải thuật PARALLEL. Lưu ý rằng, các cạnh của các kế hoạch đơn giản đều được ghi nhãn là 420, bởi vì chúng ta có trì hoãn truyền tổng cộng là:

$$D_0 + 400 \times D_1 = 420$$

Các bản tóm lược ban đầu

SUPPLIER

card(SUPPLIER) = 225

site(SUPPLIER) = 1

	SNUM	NAME
Size	4	20
val	225	225
p	0.9	-

SUPPLY

card(SUPPLY) = 100

site(SUPPLY) = 2

	SNUM	DEPTNUM	QTY
	4	5	7
	100	20	50
	0.4	0.2	-

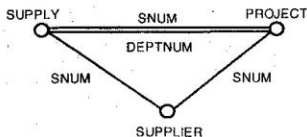
PROJECT

card(PROJECT) = 1000

site(PROJECT) = 3

	SNUM	DEPTNUM	PROJNUM
	4	5	4
	100	90	30
	0.4	0.9	-

Đồ thị kết

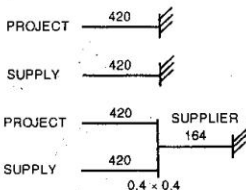


Các trị hoàn

 $D_0 = 20$ $D_1 = 1$

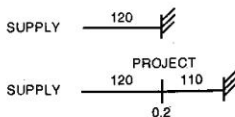
Giải thuật PARALLEL trên SNUM

Quan hệ	S
SUPPLY	400
PROJECT	400
SUPPLIER	900



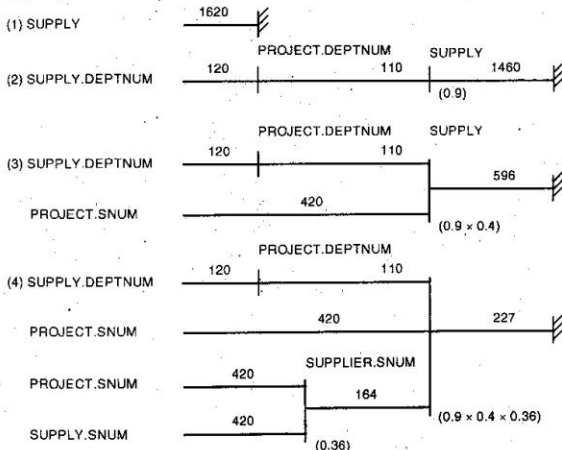
Giải thuật PARALLEL trên DEPTNUM

Quan hệ	S
SUPPLY	100
PROJECT	450



Các kế hoạch cho SUPPLY	Trị hoàn	Hệ số chọn
PROJECT.DEPTNUM	230	0.9
PROJECT.SNUM	420	0.4
SUPPLIER.SNUM	584	0.36

Giải thuật RESPONSE cho SUPPLY



Kế hoạch	Trị hoàn tổng cộng
1	1620
2	1690
3	1016
4	811

Hình 6.8 Tối ưu hóa truy vấn sử dụng giải thuật AHY

Sau đó, chúng ta xét kế hoạch của SUPPLIER và nhận thấy rằng kế hoạch đơn giản cho nó tương ứng với trị hoàn 920 đơn vị. Chúng ta xét kế hoạch sử dụng các phép nửa kết với PROJECT và SUPPLY. Kế hoạch này có được bằng cách kết hợp các kế hoạch cho PROJECT và SUPPLY với cạnh nối nút SUPPLIER với nút gốc. Các hệ số chọn đầu vào do các phép nửa kết này được chỉ ra ở dưới nút SUPPLIER. Trị hoàn để truyền quan hệ rút gọn SUPPLIER đến nơi thực hiện được tính như sau:

$$D_0 + 900 \times 0.16 \times D_1 = 164 \text{ đơn vị}$$

Kế hoạch này ngắn hơn so với kế hoạch đơn giản, bởi vì nó tương ứng với một tri hoãn tổng cộng là 584 đơn vị; do đó nó được chọn và giải thuật PARALLEL kết thúc.

Giải thuật PARALLEL áp dụng một cách tương tự cho thuộc tính DEPTNUM. Kết quả của áp dụng này được chỉ ra trong hình 6.8a. PROJECT.DEPTNUM được rút gọn với hệ số đầu vào $p = 0.2$ trước khi nó được truyền đi.

Lưu ý quan trọng là giải thuật PARALLEL đánh giá chi phí truyền một quan hệ chỉ có một thuộc tính đơn bằng cách rút gọn bởi các phép nửa kết trên thuộc tính đó. Do đó, giải pháp mà chúng ta có được là rất khác so với giải pháp của vấn đề ban đầu. Tuy nhiên, không có gì được quyết định tại lúc này. Chúng ta chỉ đơn giản xây dựng dữ liệu vào cho việc tích hợp các kế hoạch.

2- Tích hợp các kế hoạch

Bây giờ, chúng ta tập trung vào vấn đề tích hợp các kế hoạch cho tối thiểu hóa tri hoãn. Tối thiểu hóa tri hoãn có được bằng cách áp dụng giải thuật RESPONSE mà giải thuật này rất tương tự với giải thuật PARALLEL.

Giải thuật RESPONSE áp dụng riêng biệt cho mỗi quan hệ của đồ thị kết và tạo ra kế hoạch ngắn nhất cho quan hệ này; kế hoạch bao gồm lần truyền cuối cùng của quan hệ đến nơi thực hiện. Giải thuật, được áp dụng cho một quan hệ R , diễn biến như sau:

1- *Bắt đầu*: Xét tất cả các thuộc tính kết có thể có của R . Sắp thứ tự chúng tương ứng với tri hoãn cần phải có để nhận các thuộc tính kết tại nơi của R . Gọi S_j biểu diễn kế hoạch tương ứng.

2- *Phương pháp*: Xây dựng kế hoạch cho R bằng cách chọn kế hoạch với chi phí tối thiểu giữa:

a- Kế hoạch đơn giản cho R

b- Đối với $j \geq 1$, kế hoạch được xây dựng là sự kết hợp của:

- Kế hoạch S_j mà trong đó chúng ta đặt quan hệ R thay cho nút gốc.

- Tất cả các kế hoạch S_k , với $k < j$ mà trong đó chúng ta cũng đặt quan hệ R thay cho nút gốc.

- Cạnh giữa R và nút gốc, được ghi nhãn bởi chi phí truyền của R được rút gọn theo các hệ số chọn đầu vào của tất cả các phép nửa kết được áp dụng cho nó.

Lý do của quy tắc (b) của giải thuật này là bất kỳ lần truyền song song nào của kế hoạch S_k ngắn hơn sẽ tạo ra thêm một hệ số chọn đầu vào mà không làm tăng trì hoãn, và do đó phải được đưa vào.

Phần cuối cùng của cách tiếp cận AHY, tương tự với cách tiếp cận SDD-1, bao gồm việc xác định nơi thực hiện, bằng cách chọn một nơi với kế hoạch liên kết dài nhất; mỗi lần kế hoạch và nơi này được xác định, các lần truyền thông tương ứng với nó sẽ không còn cần thiết nữa, và trì hoãn tổng cộng liên kết với truy vấn trở thành trì hoãn của kế hoạch dài thứ hai.

Ví dụ 6.2 (tiếp tục) Hình 6.8b cho thấy áp dụng của giải thuật RESPONSE cho quan hệ SUPPLY. Các thuộc tính có thể có dùng cho phép rút gọn nửa kết của SUPPLY là DEPTNUM và SNUM. Chúng có thể được truyền từ quan hệ PROJECT, trong khi đó quan hệ SUPPLIER có thể chỉ truyền thuộc tính SNUM. Từ các kế hoạch của PROJECT.DEPTNUM, PROJECT.SNUM và SUPPLIER.SNUM trong hình 6.8a, chúng ta biết được các trì hoãn cần thiết. Sau đó, chúng ta thực hiện:

- Kế hoạch đơn giản cho SUPPLY.
- Kế hoạch mà nó sử dụng thuộc tính PROJECT.DEPTNUM. với trì hoãn ngắn nhất. để rút gọn SUPPLY (hệ số chọn đầu vào là 0.9).
- Kế hoạch mà nó sử dụng cả hai PROJECT.DEPTNUM và PROJECT.SNUM để rút gọn SUPPLY (hệ số chọn đầu vào tổng cộng là 0.36).
- Kế hoạch sử dụng tất cả các thuộc tính để rút gọn SUPPLY (hệ số chọn đầu vào tổng cộng là 0.1296).

Trong các kế hoạch này thì kế hoạch thứ tư có trì hoãn ngắn nhất (696 đơn vị) và nó được chọn:

6.2.4 Xử lý truy vấn bằng cách sử dụng các phép kết

Các phương pháp tối ưu hóa truy vấn mà chúng ta đã giới thiệu dựa trên ba pha khác biệt theo trình tự:

- Thực hiện xử lý cục bộ
- Sử dụng các phép nửa kết để làm giảm các nơi của các quan hệ, mà các quan hệ được gửi đến một nơi chung
- Thực hiện truy vấn dựa trên các quan hệ rút gọn.

Một cách tiếp cận khác bao gồm việc thực hiện luân phiên các phép toán cục bộ và một số lần truyền dữ liệu. Do đó, một truy vấn gồm phép kết của ba quan hệ mà chúng được lưu trữ tại các nơi khác

nhau được giải quyết bằng cách trước tiên gửi quan hệ đầu tiên đến nơi thứ hai, sau đó thực hiện phép kết của chúng, gửi kết quả của phép kết này đến quan hệ thứ ba, và cuối cùng thực hiện phép kết của chúng và tạo ra kết quả của truy vấn. Phương pháp này xử lý truy vấn (*query processing tactic*) bằng cách sử dụng các phép kết.

Xử lý truy vấn bằng cách sử dụng các phép nửa kết hoặc các phép kết cũng phụ thuộc vào các giả sử trên các chi phí truyền thông tương đối và xử lý cục bộ. Nói chung, ưu điểm của việc sử dụng các phép nửa kết là nhiều hơn so với các phép kết nếu các chi phí truyền thông được xem là quan trọng và các chi phí xử lý cục bộ được xem là không đáng kể. Nếu chúng ta cũng xét chi phí xử lý cục bộ trong việc đánh giá các chiến lược xử lý truy vấn khác nhau thì việc xử lý truy vấn bằng cách sử dụng các phép kết thường có ưu điểm hơn so với việc sử dụng các phép nửa kết. Một phép nửa kết $R \triangleright_{A=B} S$ cho việc rút gọn R đòi hỏi:

- Thực hiện phép chiếu S trên các thuộc tính nửa kết
- Loại bỏ các bộ bị trùng lặp từ phép chiếu này trước khi truyền chúng (điều này đòi hỏi phải sắp thứ tự các thuộc tính được chiếu).
- Thực hiện một phép kết ngoại (*extra-join*) (nghĩa là một phép kết không xuất hiện trong truy vấn) $R' = R \triangleright_{A=B} \Pi_B S$ để rút gọn kích thước của R .

Việc rút gọn bằng phép nửa kết chỉ có ưu điểm nếu số lần truyền thông vượt quá tất cả các chi phí này.

Có thể các phép kết thực hiện tốt hơn các phép nửa kết, cũng như khi chúng ta chỉ xét các chi phí truyền thông. Trong trường hợp này, các điều kiện sau đây phải thỏa mãn:

- Có ít thuộc tính trong danh sách kết quả của truy vấn (nghĩa là các thuộc tính xuất hiện trong quan hệ kết quả) mà các thuộc tính này cũng không xuất hiện trong các thuộc tính kết của phép nửa kết.
- Các phép nửa kết có hệ số chọn thấp (nghĩa là chúng không thực hiện các phép rút gọn một cách hiệu quả).

Các khảo sát ở trên đã cho thấy các kỹ thuật xử lý truy vấn sử dụng các phép kết và các phép nửa kết nên được xem xét chung với nhau. Trong ví dụ trên xét truy vấn với ba quan hệ, các phép nửa kết nên được sử dụng để rút gọn chỉ một trong các toán hạng của phép kết đầu tiên, và

phần xử lý còn lại có thể được thực hiện bằng cách sử dụng các phép kết. Tuy nhiên, có ít bộ tối ưu hóa kết hợp việc sử dụng các phép kết với các phép nữa kết, bởi vì vấn đề tối ưu hóa này rất phức tạp.

Các truy vấn kết trong cách tiếp cận R^*

Một hệ thống sử dụng cách xử lý truy vấn dùng các phép kết là R^* ; tối ưu hóa truy vấn của R^* cũng quan tâm đến các chi phí xử lý cục bộ. Ở đây, chúng ta giải quyết các phép kết của nhiều quan hệ.

Bởi vì bộ tối ưu hóa R^* cũng xét đến các chi phí xử lý cục bộ, chúng ta phải nghiên cứu có những phương pháp nào để kết cục bộ hai quan hệ và đánh giá chi phí của phép kết này. Một trong hai quan hệ được xem là **quan hệ ngoài** (*outer relation*) O , và quan hệ kia được xem là **quan hệ trong** (*inner relation*) I , phụ thuộc vào thứ tự mà chúng được quét; quan hệ ngoài có thể là kết quả của phép kết trước đó. Có hai phương pháp cơ bản:

a- Phương pháp vòng lặp lồng nhau (*nested-loop method*)

Quan hệ ngoài O được quét một cách tuần tự. Đối với mỗi bộ của O , quan hệ trong được quét, tìm kiếm các bộ kết được (*matching tuple*) dựa vào các thuộc tính kết. Các bộ kết được sẽ được kết hợp lại và được đưa vào trong kết quả của phép kết. Phương pháp này đòi hỏi phải quét toàn bộ quan hệ O và $card(O)$ lần để tìm kiếm các bộ kết được của I . Cho N_{out} là số trang lấy ra khi quét O và N_{in} là số trang trung bình của I được lấy ra đối với mỗi bộ của O . Giả sử các chi phí nhập/xuất tỉ lệ với số trang của I được lấy ra và các chi phí CPU tỷ lệ với số phần tử của kết quả của phép kết, thì chi phí của phương pháp vòng lặp lồng nhau là:

$$C(nested-loop) = (N_{out} + card(O) \times N_{in}) \times C_{IO} + card(RESULT) \times C_{CPU}$$

b- Phương pháp quét - trộn (*merge-scan method*)

Cả hai quan hệ được quét theo thứ tự thuộc tính kết. Các bộ kết được sẽ được đưa vào trong kết quả của phép kết. Các bộ của quan hệ trong có cùng giá trị thuộc tính kết sẽ được xét chung, sao cho nếu các bộ của O có cùng giá trị kết thì các bộ kết được có thể được kết hợp lại mà không cần phải lấy ra từng trang. Phương pháp này đòi hỏi phải sắp thứ tự cả hai quan hệ. Chi phí sắp thứ tự phụ thuộc vào các phương pháp sắp thứ tự được sử dụng. Chi phí của phương pháp quét - trộn là:

$$C(merge-scan) = (N_{out} + N_{in}) \times C_{IO} + C_{sort}(I) + C_{sort}(O) + card(RESULT) \times C_{CPU}$$

Trong R^* , việc kết các quan hệ I và O , mà chúng được lưu trữ tại các nơi khác nhau, có thể được thực hiện tại nơi của O , tại nơi của I , hoặc tại nơi thứ ba. Các chi phí truyền thông phải được cộng vào các chi phí cục bộ để xác định phương pháp kết tốt nhất. Chúng được đo lường dựa trên các thông điệp trao đổi giữa các nơi. Đo lường này còn chi tiết hơn so với đo lường dựa trên C_o và C_1 . Nó ở cùng mức với các đo lường xuất/nhập và CPU. Bộ tối ưu hóa R^* sử dụng hai phương pháp để truyền các quan hệ:

a- Di chuyển toàn bộ (shipped whole)

Các chi phí truyền thông được tính bởi số thông điệp cần thiết mà số lượng này tỉ lệ với số byte của quan hệ. Nếu quan hệ trong được truyền, nó phải được lưu trữ trong một quan hệ tạm thời tại nơi đến của nó. Nếu quan hệ ngoài được truyền, quan hệ trong có thể sử dụng các bộ được truyền đến và không cần lưu trữ các bộ này.

b- Lấy ra khi cần (fetched as needed)

Trong trường hợp này, nơi xa sẽ phối hợp với việc truyền các bộ và sử dụng chúng một cách trực tiếp mà không cần sử dụng vùng nhớ tạm thời. Chi phí truyền thông sẽ cao hơn khi mỗi lần lấy ra đòi hỏi phải trao đổi các thông điệp. Lưu ý rằng việc xử lý "mỗi-lần-một-bộ" (*tuple-at-a-time*) là hợp lý trong các mạng có băng thông cao (chẳng hạn như các mạng cục bộ) và không hiện thực trong các mạng phân tán về mặt địa lý. Nguyên mẫu (prototype) R^* hoạt động trong môi trường đầu tiên.

Nếu các quan hệ được di chuyển toàn bộ, chúng có thể bị hạn chế về mặt cục bộ. Nếu chúng được lấy ra khi cần, thông điệp đến có thể chứa một giá trị của thuộc tính kết, và sau đó các bộ tương ứng với giá trị này sẽ được lấy ra. Trong thực tế, hiệu quả có được do việc thực hiện phép kết mỗi-lần-một-bộ. Tuy nhiên, có sự khác biệt quan trọng giữa phương pháp này với cách tiếp cận phép nửa kết; trong khi cách làm đầu tiên chủ yếu được sử dụng để đồng bộ hóa việc xử lý các quan hệ trong và quan hệ ngoài và đòi hỏi nhiều lần truyền thông, thì cách làm sau chỉ được sử dụng để tiết kiệm các lần truyền thông và đòi hỏi nhiều xử lý cục bộ.

Về nguyên tắc, mỗi phương pháp truyền thông có thể được sử dụng trong ngữ cảnh của mỗi phương pháp kết, các khảo sát trong thực tế chỉ làm một số ít các kết hợp cho R^* . Chẳng hạn, trong phương pháp vòng lặp lồng nhau, không cho phép truyền toàn bộ I , bởi vì I phải được quét $card(O)$ lần, và điều này chỉ hiệu quả nếu có các phương pháp truy xuất hiệu quả cho thuộc tính kết. Việc truyền quan hệ sẽ làm mất hiệu lực các phương pháp như thế. Điều này và các khảo sát tương tự giới hạn các trường hợp có thể có:

1- Vòng lặp lồng nhau, "di chuyển toàn bộ quan hệ ngoài và không lưu trữ". Chi phí là tổng của các chi phí kết cục bộ trong phương pháp này và lần truyền quan hệ ngoài O :

$$C_1 = C(nested-loop) + C_{mes} \times [card(O) \times size(O) / m]$$

2- Quét - trộn, "di chuyển toàn bộ quan hệ ngoài và không lưu trữ". Tương tự, chi phí là:

$$C_2 = C(merge-scan) + C_{mes} \times [card(O) \times size(O) / m]$$

3- Vòng lặp lồng nhau, "quan hệ trong được lấy ra khi cần". Trong trường hợp này, đối với mỗi bộ của O , chúng ta gửi một yêu cầu cho I và NI bộ (tính trung bình) thỏa mãn yêu cầu này sẽ được gửi trả về. Chúng ta có:

$$C_3 = C(nested-loop) + C_{mes} \times card(O) \times (1 + [NI \times size(I) / m])$$

4- Quét - trộn, "quan hệ trong được lấy ra khi cần". Ở đây, chúng ta giả sử gửi một thông điệp yêu cầu từ O đến I đối với mỗi giá trị khác nhau của thuộc tính kết, và tất cả các bộ kết được của I sẽ được gửi trả về. Các giả sử này trái ngược với điều đã làm trong R^* , mà một thông điệp sẽ được gửi đối với một bộ bất kỳ của O . Nếu A là thuộc tính kết của O , chúng ta có:

$$C_4 = C(merge-scan) + C_{mes} \times val(A[O]) \times (1 + [NI \times size(I) / m])$$

5- Quét - trộn, "di chuyển toàn bộ quan hệ trong và lưu trữ trước khi sử dụng". Trong trường hợp này, chúng ta mất thêm chi phí để lưu trữ và lấy quan hệ tại nơi xa:

$$C_5 = C(merge-scan) + C_{mes} \times [card(I) \times size(I) / m] + 2 \times N_{in} \times C_{IO}$$

6- Di chuyển cả hai quan hệ đến nơi thứ ba. Trong trường hợp này, các chi phí truyền thông là tổng chi phí của việc truyền cả hai quan hệ trong đến quan hệ ngoài và quan hệ ngoài đến quan hệ trong; chi phí kết là $C(merge-scan)$ hoặc $C(nested-loop)$.

Bộ tối ưu hóa R^* liệt kê tất cả các chuỗi phép kết với tất cả các phương pháp kết có thể có và các phép kết được định vị tại mỗi nơi có thể có. Sau đó, nó xác định kế hoạch truy xuất ít tốn kém nhất cho một truy vấn kết. Quy hoạch động được áp dụng để làm tăng tốc độ tính toán, bởi vì việc tính toán chỉ phí thỏa mãn các quy tắc có thể áp dụng được cho kỹ thuật này.

6.3 CÁC TRUY VẤN TỔNG QUÁT

Trong phần này, chúng ta xét trường hợp tổng quát của các truy vấn với các phép kết và các phép hợp trong đồ thị tối ưu hóa. Chúng ta cho thấy phần mở rộng đi từ các truy vấn kết đến các truy vấn tổng quát là quan trọng, và rất nhiều chiến lược thực hiện khác nhau có thể được sử dụng cho cùng một truy vấn tổng quát. Phép biến đổi cơ bản được sử dụng là tính giao hoán của phép kết và phép hợp (tức là tiêu chuẩn 5 trong chương trước) dùng để tạo ra các đồ thị kết phân bố. Chúng ta tập trung vào vấn đề của một phép kết đơn giữa các quan hệ được phân mảnh, các phép đa kết có thể được giải quyết bằng cách áp dụng liên tiếp các khảo sát dưới đây.

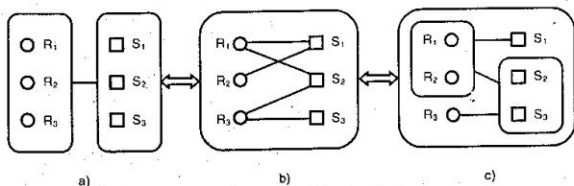
Việc xác định chiến lược thực hiện một phép hợp là điều dễ dàng, bởi vì chúng ta không quan tâm đến thứ tự của các bộ của các toán hạng được đưa vào trong quan hệ kết quả. Do đó, các lần truyền các mảnh không cục bộ đến các nơi (mà phép hợp được thực hiện) có thể được thực hiện một cách song song. Theo các yêu cầu truyền thông, phép hợp có trì hoãn tương ứng với lần truyền thông của mảnh lớn nhất, và chi phí là tổng của tất cả các chi phí truyền thông.

Tác dụng của việc giao hoán các phép kết và các phép hợp

Việc giao hoán các phép kết và các phép hợp được biểu diễn trong hình 6.9, có ba đồ thị tối ưu hóa khác nhau của cùng một truy vấn. Trong hình 6.9 và trong tất cả các hình khác của phần này, các vòng tròn biểu diễn các mảnh của quan hệ R , và các hình vuông biểu diễn các mảnh của quan hệ S . Xét hai đồ thị tối ưu hóa đầu tiên của hình 6.9a và 6.9b. Trong hình 6.9a, trước tiên các mảnh được tập hợp lại và sau đó được kết với nhau. Trong hình 6.9b trước tiên các mảnh được kết với nhau và sau đó được tập hợp lại. Chúng ta gọi cách tiếp cận thứ nhất là phép kết không phân bố (*nondistributed join*) và cách tiếp cận thứ hai là phép kết phân bố (*distributed join*). Hai trường hợp này đặt ra nhiều vấn đề tối ưu hóa khác nhau:

a- Phép kết không phân bố: Vấn đề tối ưu hóa này thì đơn giản hơn, chỉ cần xác định cặp nơi (có thể cùng một nơi) mà tại đó các phép hợp được thực hiện. Nếu hai nơi khác nhau, thì truy vấn được rút gọn thành một truy vấn kết đơn giản giữa hai quan hệ mà nó có thể được tối ưu hóa như đã chỉ ra trong phần trước.

b- Phép kết phân bố: Vấn đề tối ưu hóa này thì khó hơn. Lưu ý rằng, trong hình 6.9b chúng ta thấy đồ thị kết của phép kết giữa R và S ở bên trong nút bao (*hypernode*) mà nút này biểu diễn phép hợp. Sử dụng các tiêu chuẩn về phân mảnh để loại bỏ các cạnh của đồ thị kết mà chúng tương ứng với các phép kết rỗng, như được chỉ ra trong chương 5 phần 5.2 mục 4. Khi xác định được đồ thị kết tối thiểu, việc thực hiện các phép kết xuất hiện trong đồ thị kết phải được tối ưu hóa bằng cách sử dụng các kỹ thuật đã được trình bày trong phần trước. Cuối cùng, các kết quả kết được gửi đến cùng một nơi để thực hiện phép hợp. Đồ thị tối ưu hóa của hình 6.9b cho thấy phép kết phân bố mà trong đó phép hợp được thực hiện sau tất cả các phép kết. Tuy nhiên, cũng có thể thực hiện các ***phép hợp từng phần (partial union)*** mà chúng liên quan đến các mảnh của cùng một quan hệ, trước khi thực hiện phép kết. Một ví dụ được chỉ ra trong hình 6.9c.



Hình 6.9 Giao hoán phép kết và phép hợp

Trong việc xây dựng đồ thị tối ưu hóa G' của hình 6.9c, bắt đầu từ đồ thị tối ưu hóa G của hình 6.9b, áp dụng các quy tắc sau đây:

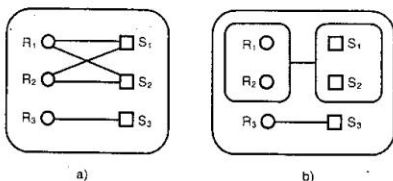
- Các mảnh dùng để thực hiện các phép hợp từng phần sẽ được bao quanh bởi một nút bao (trọng ví dụ, quy tắc này áp dụng cho $\{R_1, R_2\}$ và $\{S_2, S_3\}$).

- Nếu hai mảnh R_i và S_j được nối với nhau bởi một cạnh trong G , thì các nút (bao) mà chúng nằm trong cũng được nối với nhau bởi một cạnh trong G' (ví dụ, cạnh giữa R_1 và S_2 trong G tạo ra cạnh giữa $\{R_1, R_2\}$ và $\{S_2, S_3\}$ trong G').

Các đồ thị phân tách (*partitioned graph*) là một lớp thích hợp các đồ thị kết theo quan điểm tối ưu hóa việc thực hiện chúng. Trong đồ thị kết phân tách, chúng ta có thể phân biệt các đồ thị con rời rạc. Mỗi đồ thị con như thế có thể được tối ưu hóa một cách độc lập. Đặc tính này độc lập với phương pháp được sử dụng để tính toán các hiệu suất của truy vấn, và dựa vào các cơ sở lập luận sau đây:

- Tối ưu hóa các phép kết của các đồ thị con rời rạc có thể được thực hiện một cách độc lập.
- Phép hợp không bị ảnh hưởng bởi thứ tự của các toán hạng được tập hợp lại.

Ví dụ, trong hình 6.10a chúng ta có một đồ thị phân tách, vấn đề tối ưu hóa có thể được phân rã thành hai vấn đề bao gồm các mảnh $\{R_1, R_2, S_1, S_2\}$ và $\{R_3, S_3\}$. Trong hình 6.10b cho thấy một đồ thị tối ưu hóa cuối cùng có thể có cho truy vấn, bao gồm việc thực hiện các phép hợp từng phần của $\{R_1, R_2\}$ và $\{S_1, S_2\}$, các phép kết giữa chúng, phép kết giữa R_3 và S_3 , và phép hợp cuối cùng của các kết quả kết.



Hình 6.10 Tối ưu hóa độc lập của một đồ thị kết phân tách

Điều thuận lợi của phép biến đổi từ đồ thị tối ưu hóa của hình 6.10a thành đồ thị tối ưu hóa của hình 6.10b phụ thuộc vào sự định vị và kích thước của các mảnh. Xét ví dụ sau đây: giả sử một mạng có ba nơi, cho mảnh thứ i đặt tại nơi i , kết quả đặt tại nơi 1, các toán hạng và các kết quả của mỗi phép kết từng phần đều có cùng kích thước. Rất thuận tiện để thực hiện việc tập hợp từng phần tại nơi 1, phép kết giữa R_3 và S_3 tại nơi 3, và phép hợp cuối cùng tại nơi 1.

Các đặc tính ở trên cho phép xây dựng nhiều chiến lược bao gồm các phép hợp từng phần đối với mỗi phần trong đồ thị kết; việc tìm ra chiến lược thực hiện tốt nhất cho một truy vấn cho trước đòi hỏi:

– Tạo ra tất cả các đồ thị tối ưu hóa truy vấn có thể có.
 – Áp dụng các phương pháp truy vấn kết để tối ưu hóa các phép kết và cộng thêm các chi phí của các phép hợp. Do đó, mỗi đồ thị kết tương ứng với một chiến lược truy vấn tối ưu và một chi phí.

– Chọn chiến lược xử lý truy vấn tốt nhất trong các chiến lược ở trên.

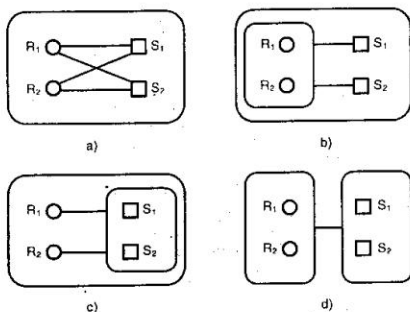
Chúng ta kết thúc phần này bằng cách minh họa một ví dụ trong hình 6.11, có bốn cách khác nhau để tính toán chi phí cho một đồ thị kết đầy đủ của hai quan hệ R và S mà mỗi quan hệ có hai mảnh.

– Thực hiện phép kết phân bố. Vấn đề này chỉ thực hiện bốn phép kết của hình 6.11a và tập hợp các kết quả của chúng. Lưu ý rằng, tối ưu hóa của bốn phép kết không thể được thực hiện riêng biệt. Bởi vì, chẳng hạn, nếu giảm kích thước của một trong các mảnh thì điều này có thể có ích cho ít nhất hai phép kết.

– Thực hiện phép hợp từng phần các mảnh của R . Vấn đề chỉ thực hiện hai phép kết, và được chỉ ra trong hình 6.11b.

– Thực hiện phép hợp từng phần các mảnh của S . Vấn đề chỉ thực hiện hai phép kết, và được chỉ ra trong hình 6.11c.

– Thực hiện phép hợp các mảnh, sau đó thực hiện phép kết của chúng; điều này tương ứng với việc thực hiện không phân bố của phép kết, được chỉ ra trong hình 6.11d.



Hình 6.11 Các đồ thị tối ưu hóa khác nhau cho cùng một truy vấn

THUẬT NGỮ VIỆT - ANH

Ái lực	<i>Affinity</i>	Ngôn ngữ thủ tục	<i>Procedural language</i>
Ảnh xạ chuẩn tắc	<i>Canonical mapping</i>	Nguyên mẫu	<i>Prototype</i>
Bao đóng	<i>Closure</i>	Người quản trị cơ sở dữ liệu	<i>Database administrator</i>
Bản nhân	<i>Replica</i>	Người quản trị cơ sở dữ liệu cục bộ	<i>Local database administrator</i>
Bản sao	<i>Copy</i>	Người quản trị cơ sở dữ liệu toàn cục	<i>Global database administrator</i>
Bản thực hiện	<i>Materialization</i>	Nhà cung cấp, người bán	<i>Vendor</i>
Bản tóm lược	<i>Profile</i>	Nơi thực hiện	<i>Execution site</i>
Bảo toàn phụ thuộc	<i>Dependency preservation</i>	Nơi, vị trí	<i>Site</i>
Băng thông	<i>Bandwidth</i>	Nút bao, siêu nút	<i>Hypernode</i>
Biến bộ	<i>Tuple variable</i>	Phân mảnh dọc	<i>Vertical fragmentation</i>
Biến cơ bản	<i>Primitive variable</i>	Phân mảnh dọc gom tụ	<i>Vertical clustering fragmentation</i>
Biến miền	<i>Domain variable</i>	Phân mảnh dọc tách biệt	<i>Vertical partitioning fragmentation</i>
Biến quan hệ	<i>Relation variable</i>	Phần mềm	<i>Software</i>
Biến tự do	<i>Free variable</i>	Phần mềm truyền thông	<i>Communication software</i>
Biểu thức chuẩn tắc	<i>Canonical expression</i>	Phân rã bảo toàn nội dung (thông tin)	<i>Lossless decomposition</i>
Biểu thức con chung	<i>Common subexpression</i>	Phân rã bảo toàn phụ thuộc hàm	<i>Dependency-preserving decomposition</i>
Biểu thức đại số quan hệ	<i>Expression of relational algebra</i>	Phân rã bảo toàn thông tin	<i>Lossless decomposition</i>
Biểu thức định nghĩa mảnh	<i>Fragment-defining expression</i>	Phân tích chức năng	<i>Functional analysis</i>
Biểu thức thành viên biến bộ	<i>Tuple-variable membership expression</i>	Phân tích định lượng	<i>Quantitative analysis</i>
Biểu thức tương đương	<i>Equivalent expression</i>	Phân tích thực thể	<i>Entity analysis</i>
Biểu thức truy vấn	<i>Query expression</i>	Phần bù	<i>Complement</i>
Bộ	<i>Tuple</i>	Phần cứng	<i>Hardware</i>
Bộ chứng minh định lý	<i>Theorem prover</i>	Phép biến đổi tương đương	<i>Equivalence transformation</i>
Bộ rút gọn hoàn toàn	<i>Full reducer</i>	Phép chọn	<i>Selection operation</i>
Bộ rút gọn mảnh	<i>Fragment reducer</i>	Phép hợp	<i>Union operation</i>
Bộ tối ưu hóa	<i>Optimizer</i>	Phép kết	<i>Join operation</i>
Bộ tối ưu hóa truy vấn	<i>Query optimizer</i>	Phép kết bằng	<i>Equijoin</i>
Bộ xử lý thông điệp giao tiếp	<i>Interface Message Processor</i>	Phép kết không phân bố	<i>Nondistributed join</i>

Cách nhìn ý niệm	<i>Conceptual view</i>	Phép kết phân bố	<i>Distributed join</i>
Cách tiếp cận gom nhóm	<i>Grouping approach</i>	Phép kết tự nhiên	<i>Natural join operation</i>
Cách tiếp cận mang tính kinh nghiệm	<i>Heuristic approach</i>	Phép nửa kết	<i>Semi-join operation</i>
Cách tiếp cận phân rã	<i>Decomposition approach</i>	Phép nửa kết tổng quát hóa	<i>Generalized semi-join</i>
Cách tiếp cận phân tách	<i>Split approach</i>	Phép phân mảnh	<i>Fragmentation operation</i>
Cách tiếp cận từ dưới lên	<i>Bottom-up approach</i>	Phép suy diễn	<i>Implication</i>
Cách tiếp cận từ trên xuống	<i>Top-down approach</i>	Phép tính quan hệ	<i>Relational calculus</i>
Cách tiếp cận vét cạn	<i>Exhaustive approach</i>	Phép tính quan hệ bộ	<i>Tuple relational calculus</i>
Cáp đồng trục	<i>Coaxial cable</i>	Phép tính quan hệ miền	<i>Domain relational calculus</i>
Cáp quang	<i>Optical fiber</i>	Phép tính toán gần đúng	<i>Mathematical approximation</i>
Cận trên	<i>Upper bound</i>	Phép toán hai ngôi	<i>Binary operation</i>
Cấu trúc liên kết mạng, topo mạng	<i>Network topology</i>	Phép toán một ngôi	<i>Unary operation</i>
Cấu trúc điều khiển phân cấp	<i>Hierarchical control structure</i>	Phép toán đại số	<i>Algebraic operation</i>
Cấu trúc phụ thuộc	<i>Dependency structure</i>	Phép toán so sánh	<i>Comparison operator</i>
Cây ưu tiên	<i>Precedence tree</i>	Phép toán số học	<i>Arithmetic operator</i>
Cây phân mảnh	<i>Fragmentation tree</i>	Phép toán, toán tử	<i>Operator</i>
Cây phân mảnh dẫn xuất	<i>Derived fragmentation tree</i>	Phiên làm việc	<i>Session</i>
Cây phân mảnh ngang dẫn xuất	<i>Derived horizontal fragmentation tree</i>	Phụ thuộc chiếu - kết	<i>Projection - join dependency</i>
Cây toán tử	<i>Operator tree</i>	Phụ thuộc hàm	<i>Functional dependency</i>
Cây truy vấn	<i>Query tree</i>	Phụ thuộc hàm đầy đủ	<i>Full functional dependency</i>
Chỉ mục thứ cấp	<i>Secondary index</i>	Phụ thuộc hàm riêng phần	<i>Partial functional dependency</i>
Chỉ số mảnh	<i>Fragment index</i>	Phụ thuộc đa trị	<i>Multivalued dependency</i>
Chỉ số nơi	<i>Site index</i>	Phủ tối thiểu	<i>Minimal cover</i>
Chi phí truy xuất đĩa	<i>Disk access cost</i>	Phương pháp kinh nghiệm tham lam	<i>Greedy heuristics</i>
Chi phí truyền thông	<i>Communication cost</i>	Phương tiện truyền thông giữa các hệ thống	<i>Intersystem communication facility</i>
Chi phí truyền thông	<i>Communication overhead</i>	Quan hệ	<i>Relation</i>
Chiến lược truy xuất	<i>Access strategy</i>	Quan hệ ban đầu	<i>Original relation</i>

Chiến lược vét cạn	<i>Exhaustive strategy</i>	Quan hệ chung, quan hệ ban đầu	<i>Universal relation</i>
Chọn tuyến	<i>Routing</i>	Quan hệ con	<i>Subrelation</i>
Chương trình phụ trợ	<i>Auxiliary program</i>	Quan hệ ngoài	<i>Outer relation</i>
Chương trình rút gọn	<i>Reducer program</i>	Quan hệ định tính	<i>Qualified relation</i>
Chương trình yêu cầu	<i>Requesting program</i>	Quan hệ được phân rã	<i>Decomposed relation</i>
Chuẩn hóa	<i>Normalization</i>	Quan hệ tạm thời	<i>Temporary relation</i>
Chuẩn hóa dữ liệu	<i>Data normalization</i>	Quan hệ toàn cục	<i>Global relational</i>
Chuỗi kết nối giữa các tập tin	<i>Interfile chain</i>	Quan hệ trong	<i>Inner relation</i>
Cơ cấu tổ chức quan hệ	<i>Relational framework</i>	Quy tắc suy diễn	<i>Inference rule</i>
Cơ cấu tổ chức ý niệm	<i>Conceptual framework</i>	Quy tắc tạo sinh	<i>Generation rule</i>
Cơ sở dữ liệu	<i>Database</i>	Quy tắc thực thể	<i>Entity rule</i>
Cơ sở dữ liệu phân bố (phân tán)	<i>Distributed database</i>	Quy tắc toàn vẹn	<i>Integrity rule</i>
Cơ sở dữ liệu quan hệ	<i>Relational database</i>	Quy tắc toàn vẹn tham chiếu	<i>Referential integrity rule</i>
Cơ sở dữ liệu tập trung	<i>Centralized database</i>	Ràng buộc toàn vẹn	<i>Integrity constraint</i>
Cơ sở dữ liệu từ xa	<i>Remote database</i>	Ràng buộc toàn vẹn tham chiếu	<i>Referential integrity constraint</i>
Cơ sở dữ liệu truyền thống	<i>Traditional database</i>	Số lượng phần tử	<i>Cardinality</i>
Công thức nguyên tử	<i>Atomic formula</i>	Số đo ái lực chung	<i>Global affinity measure</i>
Công thức đúng đắn	<i>Well-formed formula</i>	Sự (tính) trong suốt ánh xạ cục bộ	<i>Local mapping transiency</i>
Danh hiệu bộ	<i>Tuple identifier</i>	Sự (tính) trong suốt nhân bản	<i>Replication transparency</i>
Dạng chuẩn	<i>Normal form</i>	Sự (tính) trong suốt phân mảnh	<i>Fragmentation transparency</i>
Dạng chuẩn giao	<i>Conjunctive normal form</i>	Sự bất thường	<i>Anomaly</i>
Đa đồ thị	<i>Multigraph</i>	Sự biến dịch ngược	<i>Decompilation</i>
Đặc tính ngữ nghĩa	<i>Intensional property</i>	Sự giảm dư thừa	<i>Reduction of redundancy</i>
Đặc tính suy biến	<i>Degradation property</i>	Sự kết nối lẫn nhau	<i>Interconnection</i>
Đại số quan hệ	<i>Relational algebra</i>	Sự không nhất quán	<i>Inconsistency</i>
Đại số quan hệ, định tính	<i>Algebra of qualified relations</i>	Sự định vị dữ liệu	<i>Data allocation</i>
Điều khiển dư thừa	<i>Control of redundancy</i>	Sự phân bố đều	<i>Homogeneous distribution</i>
Điều khiển đồng thời	<i>Concurrency control</i>	Sự phân mảnh dẫn xuất	<i>Derived fragmentation</i>
Điều khiển tập trung	<i>Centralized control</i>	Sự phân mảnh dữ liệu	<i>Data fragmentation</i>
Điều kiện cần và đủ	<i>Sufficient and necessary condition</i>	Sự phân mảnh hỗn hợp	<i>Mixed (hybrid) fragmentation</i>
Điều kiện đầy đủ	<i>Completeness condition</i>	Sự phân mảnh ngang	<i>Horizontal fragmentation</i>
Điều kiện đúng đắn	<i>Correctness condition</i>	Sự phân mảnh ngang chính	<i>Primary horizontal fragmentation</i>

Điều kiện tách biệt	<i>Disjointness condition</i>	Sự phân mảnh ngang dẫn xuất	<i>Derived horizontal fragmentation</i>
Điều kiện tái tạo	<i>Reconstruction condition</i>	Sự phân rã	<i>Decomposition</i>
Định danh	<i>Qualifier</i>	Sự phối hợp giữa các nơi	<i>Intersite coordination</i>
Định thuộc	<i>Determinant</i>	Sự phục hồi	<i>Recovery</i>
Định vị tối ưu	<i>Optimal allocation</i>	Sự quản trị trung tâm	<i>Central administration</i>
Đơn vị định vị	<i>Unit of allocation</i>	Sự thông kết giữa các nơi	<i>Intersite navigation</i>
Đơn vị định vị luận lý	<i>Logical unit of allocation</i>	Sự tích hợp	<i>Integration</i>
Đơn vị phân bố	<i>Unit of distribution</i>	Sự tìm (rút) thừa số	<i>Factorization</i>
Độ chọn giao tối thiểu	<i>Minterm selectivity</i>	Sự tương quan luận lý	<i>Logical correlation</i>
Đồ thị kết	<i>Join graph</i>	Tác vụ cơ bản truy xuất cơ sở dữ liệu	<i>Database access primitive</i>
Đồ thị kết hoàn toàn	<i>Total join graph</i>	Tác vụ, toán tử, phép toán	<i>Operation</i>
Đồ thị kết đơn	<i>Simple join graph</i>	Tải làm việc	<i>Workload</i>
Đồ thị kết phân tách	<i>Partitioned join graph</i>	Tần số ứng dụng	<i>Application frequency</i>
Đồ thị luồng	<i>Flow graph</i>	Tần số truy xuất	<i>Access frequency</i>
Đồ thị phân tách	<i>Partitioned graph</i>	Tần số truy xuất giao tối thiểu	<i>Minterm access frequency</i>
Đồ thị suy giảm	<i>Reduced join graph</i>	Tham chiếu cục bộ	<i>Local reference</i>
Đồ thị toán tử	<i>Operator graph</i>	Tham chiếu từ xa	<i>Remote reference</i>
Đồ thị truy vấn	<i>Query graph</i>	Tham số	<i>Parameter</i>
Độ tin cậy	<i>Reliability</i>	Tham số nhập	<i>Input parameter</i>
Gam dữ liệu	<i>Datagram</i>	Tham số xuất	<i>Output parameter</i>
Giao dịch	<i>Transaction</i>	Thành phần	<i>Component</i>
Giao dịch toàn cục	<i>Global transaction</i>	Thành phần quản trị cơ sở dữ liệu	<i>Database management component</i>
Giao thức	<i>Protocol</i>	Thể hiện quan hệ	<i>Relation instance</i>
Giá trị sử dụng thuộc tính	<i>Attribute usage value</i>	Thiết bị lưu trữ vật lý	<i>Physical storage device</i>
Giải thuật mạng tính kinh nghiệm	<i>Heuristic algorithm</i>	Thiết kế khung nhìn	<i>View design</i>
Giải thuật năng lượng liên kết	<i>Bond Energy Algorithm</i>	Thiết kế phân bố	<i>Distribution design</i>
Gói dữ liệu	<i>Frame of data</i>	Thiết kế vật lý	<i>Physical design</i>
Hàm chân trị	<i>Truth function</i>	Thiết kế ý niệm	<i>Conceptual design</i>
Hàm kết hợp	<i>Aggregate function</i>	Thời gian biên dịch	<i>Compile time</i>
Hàm mục tiêu	<i>Goal function</i>	Thời gian chạy	<i>Run time</i>
Hàm mục tiêu	<i>Objective function</i>	Thời gian đáp ứng	<i>Response time</i>
Hàm số học	<i>Arithmetic function</i>	Thông lượng	<i>Throughput</i>
Hệ điều hành	<i>Operating system</i>	Thông lượng của hệ thống	<i>System throughput</i>
Hệ quản trị cơ sở dữ liệu	<i>Database management system</i>	Thông điệp	<i>Message</i>

Hệ quản trị cơ sở dữ liệu phân bố	<i>Distributed database management system</i>	Thông tin kiểu truy xuất	<i>Access pattern information</i>
Hệ số	<i>Coefficient</i>	Thuộc tính	<i>Attribute</i>
Hệ số chọn	<i>Selectivity</i>	Thuộc tính chiếu	<i>Projected attribute</i>
Hệ số chọn đầu vào	<i>Incoming selectivity</i>	Thuộc tính kết	<i>Join attribute</i>
Hệ thống không đồng nhất	<i>Heterogeneous system</i>	Thuộc tính không khóa	<i>Nonprime attribute</i>
Hệ thống một lần một mẫu tin	<i>One-record-at-a-time system</i>	Thuộc tính khóa	<i>Prime attribute</i>
Hệ thống đa xử lý	<i>Multiprocessor system</i>	Thuộc tính phân mảnh	<i>Fragmentation attribute</i>
Hệ thống đồng nhất	<i>Homogeneous system</i>	Tích hợp khung nhìn	<i>View integration</i>
Hệ thống phân bố	<i>Distributed system</i>	Tính cục bộ	<i>Locality</i>
Hệ thống tập trung	<i>Centralized system</i>	Tính cục bộ của ứng dụng	<i>Application locality</i>
Hệ tiên đề Armstrong	<i>Armstrong's axioms</i>	Tính cục bộ hoàn toàn	<i>Complete locality</i>
Hiện tượng cổ chai	<i>Bottleneck</i>	Tính cục bộ xử lý	<i>Processing locality</i>
Hiệu suất	<i>Performance</i>	Tính giao hoán	<i>Commutativity</i>
Hoán vị	<i>Permutation</i>	Tính hiệu quả	<i>Efficiency</i>
Kế hoạch	<i>Schedule</i>	Tính kết hợp	<i>Associativity</i>
Kế hoạch đơn giản	<i>Simple schedule</i>	Tính không đồng nhất	<i>Heterogeneity</i>
Kế hoạch truy xuất	<i>Access plan</i>	Tính kinh tế	<i>Economics</i>
Kế hoạch truy xuất phân bố	<i>Distributed access plan</i>	Tính linh hoạt	<i>Flexibility</i>
Khả năng xử lý tự trị (độc lập)	<i>Autonomous processing capability</i>	Tính lũy đẳng	<i>Idempotence</i>
Khái niệm, ý niệm	<i>Concept</i>	Tính mở rộng	<i>Expandability</i>
Khóa được điều khiển bởi hệ thống	<i>System-controlled key</i>	Tính đầy đủ	<i>Completeness</i>
Khung nhìn	<i>View</i>	Tính nguyên tố của giao dịch	<i>Transaction Atomicity</i>
Kịch hoạt	<i>Enforce</i>	Tính độc lập dữ liệu	<i>Data independence</i>
Kiểm tra dữ liệu ngữ nghĩa	<i>Semantic data control</i>	Tính đồng nhất	<i>Homogeneity</i>
Kiến trúc nhiều mức	<i>Multilevel architecture</i>	Tính đồng thời	<i>Concurrency</i>
Kiến trúc tham khảo phân lớp	<i>Layered reference architecture</i>	Tính đồng thời nội truy vấn	<i>Intraquery concurrency</i>
Kiểu thực thể	<i>Entity type</i>	Tính đúng đắn	<i>Correctness</i>
Ký hiệu	<i>Notation</i>	Tính phân phối	<i>Distributivity</i>
Lập trình thông kết	<i>Navigational programming</i>	Tính sẵn sàng	<i>Availability</i>
Liên kết truyền thông	<i>Communication link</i>	Tính tách biệt mảnh	<i>Fragment disjointness</i>
Lớp hiển thị	<i>Presentation layer</i>	Tính tái tạo	<i>Reconstructability</i>
Lớp liên kết dữ liệu	<i>Data link layer</i>	Tính tối ưu	<i>Optimality</i>
Lớp mạng	<i>Network layer</i>	Tính tối thiểu	<i>Minimality</i>
Lớp ứng dụng	<i>Application layer</i>	Tính toàn vẹn	<i>Integrity</i>
Lớp phiên	<i>Session layer</i>	Tính toàn vẹn của cơ sở dữ liệu	<i>Database integrity</i>

Lớp vật lý	<i>Physical layer</i>	Tính toàn vẹn ngữ nghĩa	<i>Semantic integrity</i>
Lược đồ ánh xạ cục bộ	<i>Local mapping schema</i>	Tính toán phân bố	<i>Distributed computation</i>
Lược đồ lưu trữ	<i>Storage schema</i>	Tính tự trị vị trí	<i>Site autonomy</i>
Lược đồ độc lập vị trí	<i>Site-independent schema</i>	Tính từ xa	<i>Remoteness</i>
Lược đồ định vị	<i>Allocation schema</i>	Tính trong suốt phân bố	<i>Distribution transparency</i>
Lược đồ phân mảnh	<i>Fragmentation schema</i>	Tính trong suốt vị trí	<i>Location transparency</i>
Lược đồ quan hệ	<i>Relation schema</i>	Toán hạng	<i>Operand</i>
Lược đồ toàn cục	<i>Global schema</i>	Tối ưu hóa cục bộ	<i>Local optimization</i>
Lược đồ ý niệm	<i>Conceptual schema</i>	Tối ưu hóa toàn cục	<i>Global optimization</i>
Lược đồ ý niệm cục bộ	<i>Local conceptual schema</i>	Tối ưu hóa truy vấn	<i>Query optimization</i>
Lược đồ ý niệm toàn cục	<i>Global conceptual schema</i>	Tối thiểu hóa	<i>Minimization</i>
Lưu lượng cập nhật	<i>Update traffic</i>	Từ điển dữ liệu	<i>Data dictionary</i>
Lưu lượng chỉ đọc	<i>Read-only traffic</i>	Truy vấn cập nhật	<i>Update query</i>
Mã trận ái lực gom tụ	<i>Clustered Affinity matrix</i>	Truy vấn cây	<i>Tree query</i>
Mã trận ái lực thuộc tính	<i>Attribute affinity matrix</i>	Truy vấn chỉ đọc	<i>Read-only query</i>
Mạng con truyền thông	<i>Communication subnet</i>	Truy vấn chọn-chiều-kết	<i>Select-project-join query</i>
Mạng cục bộ	<i>Local network</i>	Truy vấn kết	<i>Join query</i>
Mạng cục bộ băng thông cao	<i>High-bandwidth local network</i>	Truy vấn mảnh	<i>Fragment query</i>
Mạng lưu trữ-và-chuyển tiếp	<i>Store-and-forward network</i>	Truy vấn quan hệ	<i>Relational query</i>
Mạng điểm - điểm	<i>Point-to-point network</i>	Truy vấn tham số	<i>Parametric query</i>
Mạng đường dài	<i>Long-haul network</i>	Truy vấn toàn cục	<i>Global query</i>
Mạng phân bố	<i>Distributed network</i>	Truy vấn vòng	<i>Cyclic query</i>
Mạng truyền rộng	<i>Broadcast network</i>	Truy xuất cập nhật	<i>Update access</i>
Mã điều kiện	<i>Condition code</i>	Truy xuất có thẩm quyền	<i>Authorized access</i>
Mảnh	<i>Fragment</i>	Truy xuất lấy dữ liệu	<i>Retrieval access</i>
Mảnh giao tối thiểu	<i>Minitem fragment</i>	Truy xuất từ xa	<i>Remote access</i>
Mảnh ngang	<i>Horizontal fragment</i>	Truyền rộng	<i>Broadcast</i>
Mảnh được rút gọn	<i>Reduced fragment</i>	Ứng dụng cập nhật	<i>Update application</i>
Mảnh tách biệt	<i>Disjoint fragment</i>	Ứng dụng chỉ đọc	<i>Read-only application</i>
Máy tính ứng dụng	<i>Application computer</i>	Ứng dụng cục bộ	<i>Local application</i>
Máy tính phía sau	<i>Backend computer</i>	Ứng dụng phân bố	<i>Distributed application</i>
Máy tính phía trước	<i>Front-end computer</i>	Ứng dụng tham số	<i>Parametric application</i>
Miền	<i>Domain</i>	Ứng dụng toàn cục	<i>Global application</i>
Mô hình dữ liệu mạng	<i>Network data model</i>	Vấn đề định vị cơ sở dữ liệu	<i>Database allocation problem</i>

Mô hình dữ liệu quan hệ	<i>Relational data model</i>	Vấn đề định vị tập tin	<i>File allocation problem</i>
Mô hình thực thể kết hợp	<i>Entity-relationship model</i>	Vận trù học	<i>Operations research</i>
Mô hình ý niệm	<i>Conceptual model</i>	Việc dùng chung dữ liệu	<i>Data sharing</i>
Mối liên kết	<i>Relationship</i>	Vị từ	<i>Predicate</i>
Môi trường phân bố	<i>Distributed environment</i>	Vị từ bất đẳng thức	<i>Inequality predicate</i>
Môi trường xử lý	<i>Processing environment</i>	Vị từ chọn	<i>Selection predicate</i>
Mục dữ liệu	<i>Data item</i>	Vị từ giao tối thiểu	<i>Minterm predicate</i>
Ngôn ngữ hướng tập hợp	<i>Set-oriented language</i>	Vị từ đẳng thức	<i>Equality predicate</i>
Ngôn ngữ phép tính quan hệ	<i>Relational calculus-based language</i>	Vị từ đơn giản	<i>Simple predicate</i>
Ngôn ngữ phi thủ tục	<i>Non-procedural language</i>	Vị từ định tính, điều kiện phân mảnh	<i>Qualification</i>
Ngôn ngữ thao tác dữ liệu	<i>Data manipulation language</i>	Vùng nhớ lưu trữ	<i>Storage space</i>
Ngôn ngữ thông kết	<i>Navigational language</i>	Xí nghiệp, tổ chức kinh doanh	<i>Enterprise</i>

TỪ VIẾT TẮT

AC	Access Cost
AHY	Apers, Hevner and Yao approach
ANSI	American National Standards Institute
DAP	Database Allocation Problem
DB	DataBase management component
DBA	DataBase Administrator
DBMS	DataBase Management System
DC	Data Communication component
DD	Data Dictionary
DDB	Distributed DataBase
DDBMS	Distributed DataBase Management System
DDL	Data Definition Language
DML	Data Manipulation Language
FAP	File Allocation Problem
FD	Functional Dependency
GCS	Global Conceptual Schema
IE	Integrity Enforcement
IMP	Interface Message Processor
ISC	InterSystem Communication
ISO	International Standards Organization
LCS	Local Conceptual Schema
OSI	Open Systems Interconnection
PC	Processing Cost
QPC	Query Processing Cost
RDBMS	Relational DataBase Management System
SQL	Structured Query Language
STC	STorage Cost
TC	Transmission Cost

TÀI LIỆU THAM KHẢO

1. Angelo R. Bokak, *Distributed and Multi-Database Systems*, Multiscience Press, 1993.
2. C. J. Date, *Database Systems*, Addison Wesley, 2004.
3. Jeffrey A. Hoffer, Mary B. Prescott, Fred R. McFadden, *Modern Database Management*, Prentice Hall, 2002.
4. M. Tamez Ozu, Patrick Valduriez, *Principles of Distributed Database Systems Analysis and Design*, Prentice Hall, 1991, 1999.
5. Philip M. Lewis, Arthur Bernstein, Michael Kifer, *Databases and Transaction Processing*, Addison Wesley, 2002.
6. Ramez Elmasri, Shamkant B. Navathe, *Fundamentals of Database Systems*, Addison-Wesley, 1994.
7. Ramez Elmasri, Shamkant B. Navathe, *Fundamentals of Database Systems*, Addison Wesley, 2004.
8. Stefano Ceri, Giuseppe Pelagatti, *Distributed Databases - Principles and Systems*, McGraw-Hill, 1984.

CƠ SỞ DỮ LIỆU PHÂN BỐ

Nguyễn Trung Trực

NHÀ XUẤT BẢN

ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH

KP 6, P. Linh Trung, Q. Thủ Đức, TPHCM

Số 3 Công trường Quốc tế, Q.3, TPHCM

ĐT: 38239172, 38239170

Fax: 38239172; Email: vnuhp@vnuhcm.edu.vn

★★★

Chịu trách nhiệm xuất bản

TS HUỖNH BÁ LÂN

Tổ chức bản thảo và chịu trách nhiệm về tác quyền

TRƯỜNG ĐẠI HỌC BÁCH KHOA – ĐHQG TPHCM

Biên tập

TRẦN VĂN THẮNG

Sửa bản in

PHẠM THỊ ANH TÚ

Trình bày bìa

TRƯƠNG NGỌC TUẤN

In tái bản 1.000 cuốn, khổ 16 x 24 cm

Số đăng ký KHXB: 126-2013/CXB/32-07/ĐHQG-TPHCM

Quyết định xuất bản số: 255/QĐ-ĐHQG-TPHCM/TB

ngày 14/5/2013 của Nhà xuất bản ĐHQG TPHCM

In tại Xưởng in Đại học Bách khoa - ĐHQG TP.HCM

Nộp lưu chiếu tháng 6 năm 2013.