MUC LUC

Lời nói đầu

Các hệ cơ sở dữ liệu (hệ CSDL) đầu tiên được xây dựng theo các mô hình phân cấp và mô hình mạng, đã xuất hiện vào những năm 1960, được xem là thế hệ thứ nhất của các hệ quản trị cơ sở dữ liệu (hệ QTCSDL).

Tiếp theo là thế hệ thứ hai, các hệ QTCSDL quan hệ, được xây dựng theo mô hình dữ liệu quan hệ do E.F. Codd đề xuất vào năm 1970.

Các hệ QTCSDL có mục tiêu tổ chức dữ liệu, truy cập và cập nhật những khối lượng lớn dữ liệu một cách thuận lợi, an toàn và hiệu quả.

Hai thế hệ đầu các hệ QTCSDL đã đáp ứng được nhu cầu thu thập và tổ chức các dữ liệu của các cơ quan, xí nghiệp và tổ chức kinh doanh.

Tuy nhiên, với sự phát triển nhanh chóng của công nghệ truyền thông và sự bành trướng mạnh mẽ của mạng Internet, cùng với xu thế toàn cầu hoá trong mọi lĩnh vực, đặc biệt là về thương mại, đã làm nảy sinh nhiều ứng dụng mới trong đó phải quản lý những đối tượng có cấu trúc phức tạp (văn bản, âm thanh, hình ảnh) và động (các chương trình, các mô phỏng). Trong những năm 1990 đã xuất hiện một thế hệ thứ ba các hệ QTCSDL – các hệ "hướng đối tượng", có khả năng hỗ trợ các ứng dụng đa phương tiện (multimedia).

Trước nhu cầu về tài liệu và sách giáo khoa của sinh viên chuyên nghành công nghệ thông tin, nhất là các tài liệu về CSDL phân tán, CSDL suy diễn, CSDL hướng đối tượng, chúng tôi đưa ra giáo trình môn học "Cơ sở dữ liệu 2".

Mục đích của giáo trình "Cơ sở dữ liệu 2" nhằm trình bày các khái niệm và thuật toán cơ sở của CSDL bao gồm: các mô hình dữ liệu và các hệ CSDL tương ứng, các ngôn ngữ CSDL, tổ chức lưu trữ và tìm kiếm, xử lý và tối ưu hoá câu hỏi, quản lý giao dịch và điềukhiển tương tranh, thiết kế các CSDL.

Trong quá trình biên soạn, chúng tôi đã dựa vào nội dung chương trình của môn học hiện đang được giảng dạy tại các trường Đại học trong nước, đồng thời cũng cố gắng phản ánh một số thành tựu mới của công nghệ CSDL.

Giáo trình "Cơ sở dữ liệu 2" được chia thành 2 phần

Phần 1: Cở sở dữ liệu phân tán

Phần 2: Cơ sở dữ liệu suy diễn

Sau mỗi chương đều có những phần tóm tắt cuối chương, câu hỏi ôn tập và bài tập nhằm giúp sinh viên nắm vững nội dung chính của từng chương và kiểm tra trình độ của chính mình trong việc giải các bài tập.

Tuy đã rất gố gắng, giáo trình chắc chắn còn có những thiếu sót. Rất mong nhận được ý kiến đóng góp của độc giả để trong lần tái bản sau, giáo trình sẽ hoàn chỉnh hơn.

Thái Nguyên tháng 10 năm 2009

Các tác giả

PHẦN 1

CƠ SỞ DỮ LIỆU PHÂN TÁN

CHƯƠNG 1. TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU PHÂN TÁN

Với việc phân bố ngày càng rộng rãi của các công ty, xí nghiệp, dữ liệu bài toán là rất lớn và không tập trung được. Các CSDL thuộc thế hệ một và hai không giải quyết được các bài toán trong môi trường mới không tập trung mà phân tán, song song với các dữ liệu và hệ thống không thuần nhất, thế hệ thứ ba của hệ quản trị CSDL ra đời vào những năm 80 trong đó có CSDL phân tán để đáp ứng những nhu cầu mới.

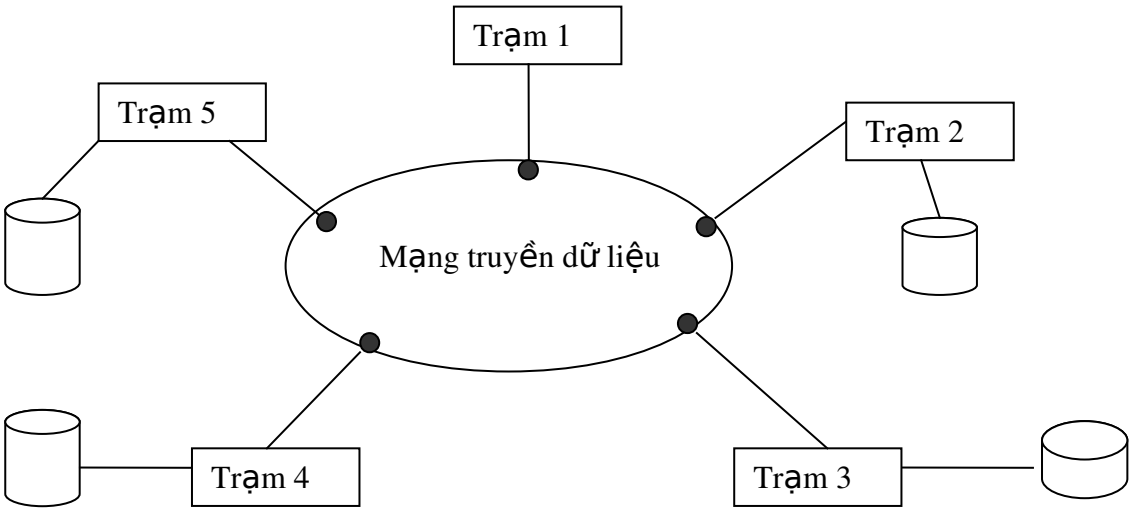
1.1. Hệ CSDL phân tán

1.1.1. Định nghĩa CSDL phân tán

Một CSDL phân tán là một tập hợp nhiều CSDL có liên đới logic và được phân bố trên một mạng máy tính

- *Tính chất phân tán:* Toàn bộ dữ liệu của CSDL phân tán không được cư trú ở một nơi mà cư trú ra trên nhiều trạm thuộc mạng máy tính, điều này giúp chúng ta phân biệt CSDL phân tán với CSDL tập trung đơn lẻ.

- *Tương quan logic*: Toàn bộ dữ liệu của CSDL phân tán có một số các thuộc tính ràng buộc chúng với nhau, điều này giúp chúng ta có thể phân biệt một CSDL phân tán với một tập hợp CSDL cục bộ hoặc các tệp cư trú tại các vị trí khác nhau trong một mạng máy tính.



Hình 1.1 Môi trường hệ CSDL phân tán

Trong hệ thống cơ sở dữ liệu phân tán gồm nhiều trạm, mỗi trạm có thể khai thác các giao tác truy nhập dữ liệu trên nhiều trạm khác.

Ví dụ 1.1: Với một ngân hàng có 3 chi nhánh đặt ở các vị trí khác nhau. Tại mỗi chi nhánh có một máy tính điều khiển một số máy kế toán cuối cùng (Teller terminal). Mỗi máy tính với cơ sở dữ liệu thống kê địa phương của nó tại mỗi chi nhánh được đặt ở một vị trí của cơ sở dữ liệu phân tán. Các máy tính được nối với nhau bởi một mạng truyền thông.

1.1.2. Các đặc điểm chính của cơ sở dữ liệu phân tán

(1) Chia sẻ tài nguyên

Việc chia sẻ tài nguyên của hệ phân tán được thực hiện thông qua mạng truyền thông. Để chia sẻ tài nguyên một cách có hiệu quả thì mỗi tài nguyên cần được quản lý bởi một chương trình có giao diện truyền thông, các tài nguyên có thể được truy cập, cập nhật một cách tin cậy và nhất quán. Quản lý tài nguyên ở đây là lập kế hoạch dự phòng, đặt tên cho các lớp tài nguyên, cho phép tài nguyên được truy cập từ nơi này đến nơi khác, ánh xạ lên tài nguyên vào địa chỉ truyền thông, ...

(2) Tính mở

Tính mở của hệ thống máy tính là dễ dàng mở rộng phần cứng (thêm các thiết bị ngoại vi, bộ nhớ, các giao diện truyền thông ...) và các phần mềm (các mô hình hệ điều hành, các giao thức truyền tin, các dịch vụ chung tài nguyên, ...)

Một hệ phân tán có tính mở là hệ có thể được tạo từ nhiều loại phần cứng và phần mềm của nhiều nhà cung cấp khác nhau với điều kiện là các thành phần này phải theo một tiêu chuẩn chung.

Tính mở của hệ phân tán được xem xét thao mức độ bổ sung vào các dịch vụ dùng chung tài nguyên mà không phá hỏng hay nhân đôi các dịch vụ đang tồn tại. Tính mở được hoàn thiện bằng cách xác định hay phân định rõ các giao diện chính của một hệ và làm cho nó tương thích với các nhà phát triển phần mềm.

Tính mở của hệ phân tán dựa trên việc cung cấp cơ chế truyền thông giữa các tiến trình và công khai các giao diện dùng để truy cập các tài nguyên chung.

(3) Khả năng song song

Hệ phân tán hoạt động trên một mạng truyền thông có nhiều máy tính, mỗi máy có thể có 1 hay nhiều CPU. Trong cùng một thời điểm nếu có N tiến trình cùng

tồn tại, ta nói chúng thực hiện đồng thời. Việc thực hiện tiến trình theo cơ chế phân chia thời gian (một CPU) hay song song (nhiều CPU)

Khả năng làm việc song song trong hệ phân tán được thực hiện do hai tình huống sau:

- Nhiều người sử dụng đồng thời ra các lệnh hay các tương tác với các chương trình ứng dụng
- Nhiều tiến trình Server chạy đồng thời, mỗi tiến trình đáp ứng các yêu cầu từ các tiến trình Client khác.

(4) Khả năng mở rộng

Hệ phân tán có khả năng hoạt động tốt và hiệu quả ở nhiều mức khác nhau. Một hệ phân tán nhỏ nhất có thể hoạt động chỉ cần hai trạm làm việc và một File Server. Các hệ lớn hơn tới hàng nghìn máy tính.

Khả năng mở rộng được đặc trưng bởi tính không thay đổi phần mềm hệ thống và phần mềm ứng dụng khi hệ được mở rộng. Điều này chỉ đạt được mức độ nào đó với hệ phân tán hiện tại. Yêu cầu việc mở rộng không chỉ là sự mở rộng về phần cứng, về mạng mà nó trải trên các khía cạnh khi thiết kế hệ phân tán.

(5) Khả năng thứ lỗi

Việc thiết kế khả năng thứ lỗi của các hệ thống máy tính dựa trên hai giải pháp:

- Dùng khả năng thay thế để đảm bảo sự hoạt động liên tục và hiệu quả.
- Dùng các chương trình hồi phục khi xảy ra sự cố.

Xây dựng một hệ thống có thể khắc phục sự cố theo cách thứ nhất thì người ta nối hai máy tính với nhau để thực hiện cùng một chương trình, một trong hai máy chạy ở chế độ Standby (không tải hay chờ). Giải pháp này tốn kém vì phải nhân đôi phần cứng của hệ thống. Một giải pháp để giảm phí tổn là các Server riêng lẻ được cung cấp các ứng dụng quan trọng để có thể thay thế nhau khi có sự cố xuất hiện. Khi không có các sự cố các Server hoạt động bình thường, khi có sự cố trên một Server nào đó, các ứng dụng Clien tự chuyển hướng sang các Server còn lại.

Cách hai thì các phần mềm hồi phục được thiết kế sao cho trạng thái dữ liệu hiện thời (trạng thái trước khi xảy ra sự cố) có thể được khôi phục khi lỗi được phát hiện.

Các hệ phân tán cung cấp khả năng sẵn sàng cao để đối phó với các sai hỏng phần cứng.

(6) Tính trong suốt

Tính trong suốt của một hệ phân tán được hiểu như là việc che khuất đi các thành phần riêng biệt của hệ đối với người sử dụng và những người lập trình ứng dụng.

Tính trong suốt về vị trí: Người sử dụng không cần biết vị trí vật lý của dữ liệu. Người sử dụng có quyền truy cập tới đến cơ sở dữ liệu nằm bất kỳ tại vị trí nào. Các thao tác lấy, cập nhật dữ liệu tại một điểm dữ liệu ở xa được tự động thực hiện bởi hệ thống tại điểm đưa ra yêu cầu, người sử dụng không cần biết đến sự phân tán của cơ sở dữ liệu trên mạng.

Tính trong suốt trong việc sử dụng: Việc chuyển đổi của một phần hay toàn bộ cơ sở dữ liệu do thay đổi về tổ chức hay quản lý, không ảnh hưởng tới thao tác người sử dụng.

Tính trong suốt của việc phân chia: Nếu dữ liệu được phân chia do tăng tải, nó không được ảnh hưởng tới người sử dụng.

Tính trong suốt của sự trùng lặp: Nếu dữ liệu trùng lặp để giảm chi phí truyền thông với cơ sở dữ liệu hoặc nâng cao độ tin cậy, người sử dụng không cần biết đến điều đó.

(7) Đảm bảo tin cậy và nhất quán

Hệ thống yêu cầu độ tin cậy cao: sự bí mật của dữ liệu phải được bảo vệ, các chức năng khôi phục hư hỏng phải được đảm bảo. Ngoài ra yêu cầu của hệ thống về tính nhất quán cũng rất quan trọng trong thể hiện: không được có mâu thuẫn trong nội dung dữ liệu. Khi các thuộc tính dữ liệu là khác nhau thì các thao tác vẫn phải nhất quán.

1.1.3. Mục đích của việc sử dụng cơ sở dữ liệu phân tán

Xuất phát từ yêu cầu thực tế về tổ chức và kinh tế: Trong thực tế nhiều tổ chức là không tập trung, dữ liệu ngày càng lớn và phục vụ cho đa người dùng nằm phân tán, vì vậy cơ sở dữ liệu phân tán là con đường thích hợp với cấu trúc tự nhiên của các tổ chức đó. Đây là một trong những yếu tố quan trọng thức đẩy việc phát triển cơ sở dữ liệu phân tán.

Sự liên kết các cơ sở dữ liệu địa phương đang tồn tại: cơ sở dữ liệu phân tán là giải pháp tự nhiên khi có các cơ sở dữ liệu đang tồn tại và sự cần thiết xây dựng một ứng dụng toàn cục. Trong trường hợp này cơ sở dữ liệu phân tán được tạo từ

dưới lên dựa trên nền tảng cơ sở dữ liệu đang tồn tại. Tiến trình này đòi hỏi cấu trúc lại các cơ sở dữ liệu cục bộ ở một mức nhất định. Dù sao, những sửa đổi này vẫn là nhỏ hơn rất nhiều so với việc tạo lập một cở sở dữ liệu tập trung hoàn toàn mới.

Làm giảm tổng chi phí tìm kiếm: Việc phân tán dữ liệu cho phép các nhóm làm việc cục bộ có thể kiểm soát được toàn bộ dữ liệu của họ. Tuy vậy, tại cùng thời điểm người sử dụng có thể truy cập đến dữ liệu ở xa nếu cần thiết. Tại các vị trí cục bộ, thiết bị phần cứng có thể chọn sao cho phù hợp với công việc xử lý dữ liệu cục bộ tại điểm đó.

Sự phát triển mở rộng: Các tổ chức có thể phát triển mở rộng bằng cách thêm các đơn vị mới, vừa có tính tự trị, vừa có quan hệ tương đối với các đơn vị tổ chức khác. Khi đó giải pháp cơ sở dữ liệu phân tán hỗ trợ một sự mở rộng uyển chuyển với một mức độ ảnh hưởng tối thiểu tới các đơn vị đang tồn tại

Trả lời truy vấn nhanh: Hầu hết các yêu cầu truy vấn dữ liệu từ người sử dụng tại bất kỳ vị trí cục bộ nào đều thoả mãn dữ liệu ngay tại thời điểm đó.

Độ tin cậy và khả năng sử dụng nâng cao: nếu có một thành phần nào đó của hệ thống bị hỏng, hệ thống vẫn có thể duy trì hoạt động.

Khả năng phục hồi nhanh chóng: Việc truy nhập dữ liệu không phụ thuộc vào một máy hay một đường nối trên mạng. Nếu có bất kỳ một lỗi nào hệ thống có thể tự động chọn đường lại qua các đường nối khác.

1.1.4. Kiến trúc cơ bản của CSDL phân tán

Đây không là kiến trúc tường minh cho tất cả các CSDL phân tán, tuy vậy kiến trúc này thể hiện tổ chức của bất kỳ một CSDL phân tán nào

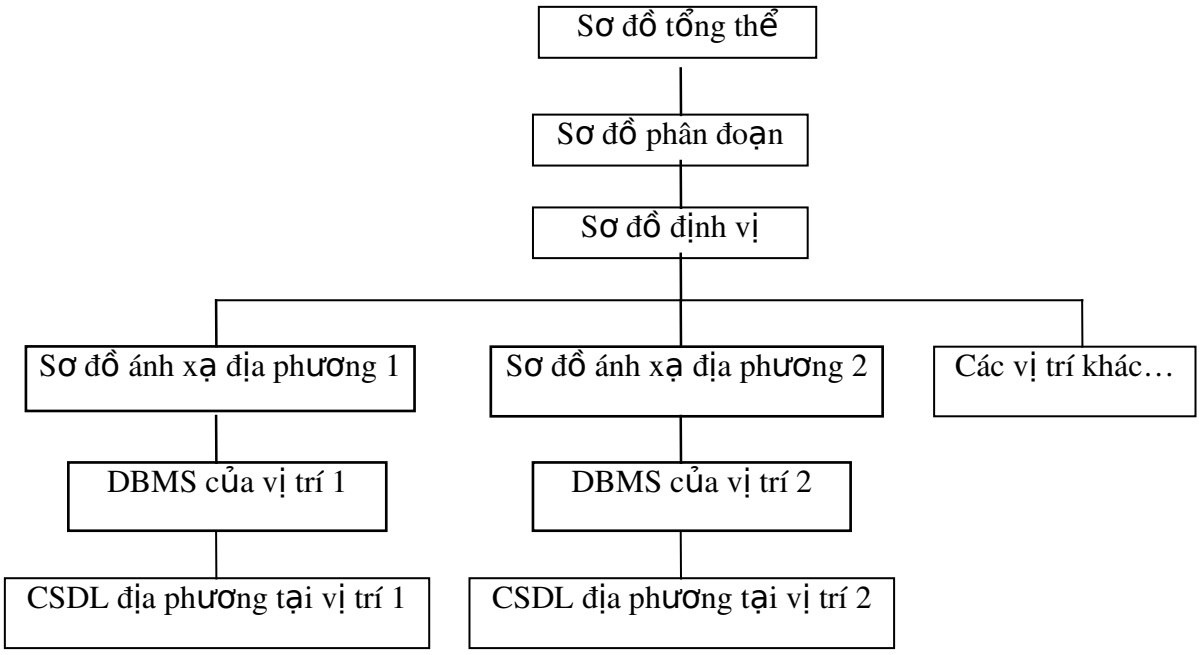
- Sơ đồ tổng thể: Định nghĩa tất cả các dữ liệu sẽ được lưu trữ trong CSDL phân tán. Trong mô hình quan hệ, sơ đồ tổng thể bao gồm định nghĩa của các tập quan hệ tổng thể.

- **Sơ đồ phân đoạn:** Mỗi quan hệ tổng thể có thể chia thành một vài phần không gối lên nhau được gọi là đoạn (fragments). Có nhiều cách khác nhau để thực hiện việc phân chia này. Ánh xạ (một - nhiều) giữa sơ đồ tổng thể và các đoạn được định nghĩa trong sơ đồ phân đoạn.

- **Sơ đồ định vị**: Các đoạn là các phần logic của quan hệ tổng thể được định vị vật lý trên một hoặc nhiều vị trí trên mạng. Sơ đồ định vị định nghĩa đoạn nào

định vị tại các vị trí nào. Lưu ý rằng kiểu ánh xạ được định nghĩa trong sơ đồ định vị quyết định CSDL phân tán là dư thừa hay không.

- **Sơ đồ ánh xạ địa phương:** ánh xạ các ảnh vật lý và các đối tượng được lưu trữ tại một trạm (tất cả các đoạn của một quan hệ tổng thể trên cùng một vị trí tạo ra một ảnh vật lý)



Hình1.2 Kiến trúc cơ bản của CSDL phân tán

1.1.5. Hệ quản trị CSDL phân tán

Hệ quản trị CSDL phân tán (Distributed Database Management System-DBMS) được định nghĩa là một hệ thống phần mềm cho phép quản lý các hệ CSDL (tạo lập và điều khiển các truy nhập cho các hệ CSDL phân tán) và làm cho việc phân tán trở nên trong suốt với người sử dụng.

Đặc tính vô hình muốn nói đến sự tách biệt về ngữ nghĩa ở cấp độ cao của một hệ thống với các vấn đề cài đặt ở cấp độ thấp. Sự phân tán dữ liệu được che dấu với người sử dụng làm cho người sử dụng truy nhập vào CSDL phân tán như hệ CSDL tập trung. Sự thay đổi việc quản trị không ảnh hưởng tới người sử dụng.

Hệ quản trị CSDL phân tán gồm 1 tập các phần mềm (chương trình) sau đây:

Các chương trình quản trị các dữ liệu phân tán

Chứa các chương trình để quản trị việc truyền thông dữ liệu

Các chương trình để quản trị các CSDL địa phương.

Các chương trình quản trị từ điển dữ liệu.

Để tạo ra một hệ CSDL phân tán (Distributed Database System-DDBS) các tập tin không chỉ có liên đới logic chúng còn phải có cấu trúc và được truy xuất qua một giao diện chung.

Môi trường hệ CSDL phân tán là môi trường trong đó dữ liệu được phân tán trên một số vị trí.

1.2. Kiến trúc hệ quản trị Cơ sở dữ liệu phân tán

1.2.1. Các hệ khách / đại lý

Các hệ quản trị CSDL khách / đại lý xuất hiện vào đầu những năm 90 và có ảnh hưởng rất lớn đến công nghệ DBMS và phương thức xử lý tính toán. Ý tưởng tổng quát hết sức đơn giản: phân biệt các chức năng cần được cung cấp và chia những chức năng này thành hai lớp: chức năng đại lý (server function) và chức năng khách hàng (client function). Nó cung cấp kiến trúc hai cấp, tạo dễ dàng cho việc quản lý mức độ phức tạp của các DBMS hiện đại và độ phức tạp của việc phân tán dữ liệu.

Đại lý thực hiện phần lớn công việc quản lý dữ liệu. Điều này có nghĩa là tất cả mọi việc xử lý và tối ưu hoá vấn tin, quản lý giao dịch và quản lý thiết bị lưu trữ được thực hiện tại đại lý. Khách hàng, ngoài ứng dụng và giao diện sẽ có modun DBMS khách chịu trách nhiệm quản lý dữ liệu được gửi đến cho bên khách và đôi khi việc quản lý các khoá chốt giao dịch cũng có thể giao cho nó. Kiến trúc được mô tả bởi hình dưới rất thông dụng trong các hệ thống quan hệ, ở đó việc giao tiếp giữa khách và đại lý nằm tại mức câu lệnh SQL. Nói cách khác, khách hàng sẽ chuyển các câu vấn tin SQL cho đại lý mà không tìm hiểu và tối ưu hoá chúng. Đại lý thực hiện hầu hết công việc và trả quan hệ kết quả về cho khách hàng.

Có một số loại kiến trúc khách/ đại lý khác nhau. Loại đơn giản nhất là trường hợp có một đại lý được nhiều khách hàng truy xuất. Chúng ta gọi loại này là nhiều khách một đại lý. Một kiến trúc khách/ đại lý phức tạp hơn là kiến trúc có nhiều đại lý trong hệ thống (được gọi là nhiều khách nhiều đại lý). Trong trường hợp này chúng ta có hai chiến lược quản lý: hoặc mỗi khách hàng tự quản lý nối kết của nó với đại lý hoặc mỗi khách hàng chỉ biết đại lý "ruột" của nó và giao tiếp với các đại lý khác qua đại lý đó khi cần. Lối tiếp cận thứ nhất làm đơn giản cho các chương trình đại lý nhưng lại đặt gánh nặng lên các máy khách cùng với nhiều trách nhiệm khác. Điều này dẫn đến tình huống được gọi là các hệ thống khách tự phục vụ. Lối tiếp cận sau tập trung chức năng quản lý dữ liệu tại đại lý. Vì thế sự vô hình của truy xuất dữ liệu được cung cấp qua giao diện của đại lý.

Từ góc độ tính logíc cả dữ liệu, DBMS khách/ đại lý cung cấp cùng một hình ảnh dữ liệu như các hệ ngang hàng sẽ được thảo luận ở phần tiếp theo. Nghĩa

là chúng cho người sử dụng thấy một hình ảnh về một CSDL logic duy nhất, còn tại mức vật lý nó có thể phân tán. Vì thế sự phân biệt chủ yếu giữa các hệ khách/đại lý và ngang hàng không phải ở mức vô hình được cung cấp cho người dùng và cho ứng dụng mà ở mô hình kiến trúc được dùng để nhận ra mức độ vô hình này.

1.2.2. Các hệ phân tán ngang hàng

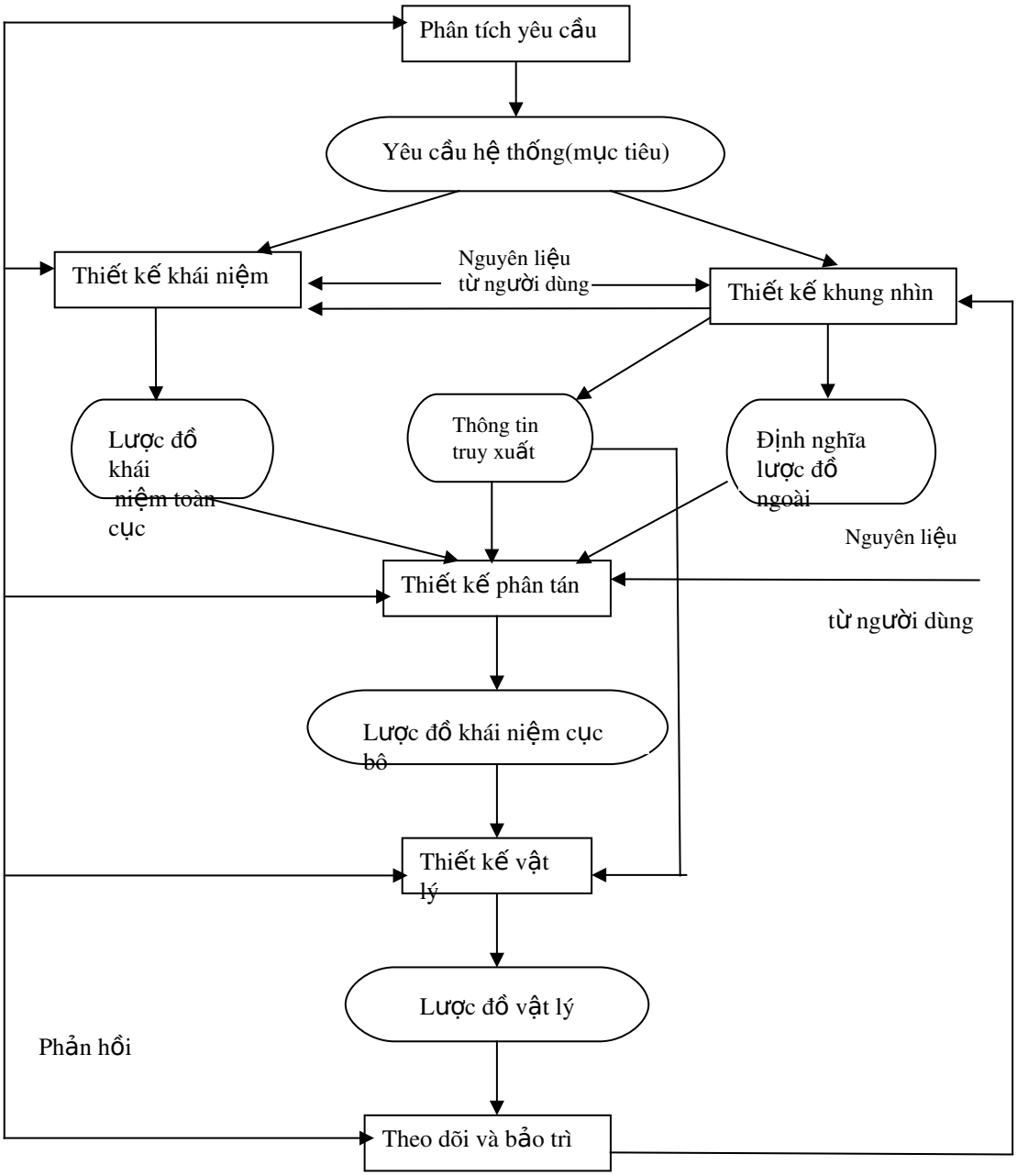
Mô hình client / server phân biệt client (nơi yêu cầu dịch vụ) và server (nơi phục vụ các yêu cầu). Nhưng mô hình xử lý ngang hàng, các hệ thống tham gia có vai trò như nhau. Chúng có thể yêu cầu vừa dịch vụ từ một hệ thống khác hoặc vừa trở thành nơi cung cấp dịch vụ. Một cách lý tưởng, mô hình tính toán ngang hàng cung cấp cho xử lý hợp tác giữa các ứng dụng có thể nằm trên các phần cứng hoặc hệ điều hành khác nhau. Mục đích của môi trường xử lý ngang hàng là để hỗ trợ các CSDL được nối mạng. Như vậy người sử dụng DBMS sẽ có thể truy cập tới nhiều CSDL không đồng nhất.

CHƯƠNG 2. CÁC PHƯƠNG PHÁP PHÂN TÁN DỮ LIỆU

2.1.Thiết kế cơ sở dữ liệu phân tán

2.1.1.Các chiến lược thiết kế

Quá trình thiết kế từ trên xuống (top-down)



Hình 2.1. Quá trình thiết kế từ trên xuống

Phân tích yêu cầu: nhằm định nghĩa môi trường hệ thống và thu thập các nhu cầu về dữ liệu và nhu cầu xử lý của tất cả mọi người có sử dụng CSDL

Thiết kế khung nhìn: định nghĩa các giao-diện cho người sử dụng cuối (end-user)

Thiết kế khái niệm: xem xét tổng thể xí nghiệp nhằm xác định các loại thực thể và mối liên hệ giữa các thực thể.

Thiết kế phân tán: chia các quan hệ thành nhiều quan hệ nhỏ hơn gọi là phân mảnh và cấp phát chúng cho các vị trí.

Thiết kế vật lý: ánh xạ lược đồ khái niệm cục bộ sang các thiết bị lưu trữ vật lý có sẵn tại các vị trí tương ứng.

Quá trình thiết kế từ dưới lên (bottom-up)

Thiết kế từ trên xuống thích hợp với những CSDL được thiết kế từ đầu. Tuy nhiên chúng ta cũng hay gặp trong thực tế là đã có sẵn một số CSDL, nhiệm vụ thiết kế là phải tích hợp chúng thành một CSDL. Tiếp cận từ dưới lên sẽ thích hợp cho tình huống này. Khởi điểm của thiết kế từ dưới lên là các lược đồ khái niệm cục bộ . Quá trình này sẽ bao gồm việc tích hợp các lược đồ cục bộ thành khái niệm lược đồ toàn cục.

2.2. Các vấn đề thiết kế

2.2.1. Lý do phân mảnh

Khung nhìn của các ứng dụng thường chỉ là một tập con của quan hệ. Vì thế đơn vị truy xuất không phải là toàn bộ quan hệ nhưng chỉ là các tập con của quan hệ. Kết quả là xem tập con của quan hệ là đơn vị phân tán sẽ là điều thích hợp duy nhất.

Việc phân rã một quan hệ thành nhiều mảnh, mỗi mảnh được xử lý như một đơn vị, sẽ cho phép thực hiện nhiều giao dịch đồng thời. Ngoài ra việc phân mảnh các quan hệ sẽ cho phép thực hiện song song một câu vấn tin bằng cách chia nó ra thành một tập các câu vấn tin con hoạt tác trên các mảnh. Vì thế việc phân mảnh sẽ làm tăng mức độ hoạt động đồng thời và như thế làm tăng lưu lượng hoạt động của hệ thống.

2.2.2. Các kiểu phân mảnh

Các quy tắc phân mảnh đúng đắn

Chúng ta sẽ tuân thủ ba quy tắc trong khi phân mảnh mà chúng bảo đảm rằng CSDL sẽ không có thay đổi nào về ngữ nghĩa khi phân mảnh.

a) Tính đầy đủ (completeness).

Nếu một thể hiện quan hệ R được phân rã thành các mảnh R_1, R_2,…,R_n, thì mỗi mục dữ liệu có thể gặp trong R cũng có thể gặp một trong nhiều mảnh R_i. Đặc tính này giống như tính chất phân rã nối không mất thông tin trong chuẩn hoá, cũng quan trọng trong phân mảnh bởi vì nó bảo đảm rằng dữ liệu trong quan hệ R được ánh xạ vào các mảnh và không bị mất. Chú ý rằng trong trường hợp phân mảnh ngang "mục dữ liệu" muốn nói đến là một bộ, còn trong trường hợp phân mảnh dọc, nó muốn nói đến một thuộc tính.

b) Tính tái thiết được (reconstruction).

Nếu một thể hiện quan hệ R được phân rã thành các mảnh R_1, R_2,…,R_n, thì cần phải định nghĩa một toán tử quan hệ sao cho

R= R_i, R_i F_r

Toán tử thay đổi tuỳ theo từng loại phân mảnh, tuy nhiên điều quan trọng là phải xác định được nó. Khả năng tái thiết một quan hệ từ các mảnh của nó bảo đảm rằng các ràng buộc được định nghĩa trên dữ liệu dưới dạng các phụ thuộc sẽ được bảo toàn.

c) Tính tách biệt (disjointness).

Nếu quan hệ R được phân rã ngang thành các mảnh R_1, R_2,…,R_n, và mục dữ liệu d_i nằm trong mảnh R_j, thì nó sẽ không nằm trong mảnh R_k khác ($k \neq j$). Tiêu chuẩn này đảm bảo các mảnh ngang sẽ tách biệt (rời nhau). Nếu quan hệ được phân rã dọc, các thuộc tính khoá chính phải được lặp lại trong mỗi mảnh. Vì thế trong trường hợp phân mảnh dọc, tính tách biệt chỉ được định nghĩa trên các trường không phải là khoá chính của một quan hệ.

Các yêu cầu thông tin

Một điều cần lưu ý trong việc thiết kế phân tán là quá nhiều yếu tố có ảnh hưởng đến một thiết kế tối ưu. tổ chức logic của CSDL, vị trí các ứng dụng, đặc tính truy xuất của các ứng dụng đến CSDL, và các đặc tính của hệ thống máy tính tại mỗi vị trí đều có ảnh hưởng đến các quyết định phân tán. Điều này khiến cho việc diễn đạt bài toán phân tán trở nên hết sức phức tạp.

Các thông tin cần cho thiết kế phân tán có thể chia thành bốn loại:

- Thông tin CSDL

- Thông tin ứng dụng
- Thông tin về mạng
- Thông tin về hệ thống máy tính

Hai loại sau có bản chất hoàn toàn định lượng và được sử dụng trong các mô hình cấp phát chứ không phải trong các thuật toán phân mảnh

2.2.3. Phân mảnh ngang

Trong phần này, chúng ta bàn đến các khái niệm liên quan đến phân mảnh ngang (phân tán ngang). Có hai chiến lược phân mảnh ngang cơ bản:

- Phân mảnh nguyên thuỷ (primary horizontal fragmentation) của một quan hệ được thực hiện dựa trên các vị từ được định nghĩa trên quan hệ đó.

- Phân mảnh ngang dẫn xuất (derived horizontal fragmentation) là phân mảnh một quan hệ dựa vào các vị từ được định trên một quan hệ khác.

Hai kiểu phân mảnh ngang

Phân mảnh ngang chia một quan hệ r theo các bộ, vì vậy mỗi mảnh là một tập con các bộ t của quan hệ r.

Phân mảnh nguyên thuỷ (primary horizontal fragmentation) của một quan hệ được thực hiện dựa trên các vị từ được định nghĩa trên quan hệ đó. Ngược lại phân mảnh ngang dẫn xuất (derived horizontal fragmentation) là phân mảnh một quan hệ dựa vào các vị từ được định trên một quan hệ khác. Như vậy trong phân mảnh ngang *tập các vị từ* đóng vai trò quan trọng.

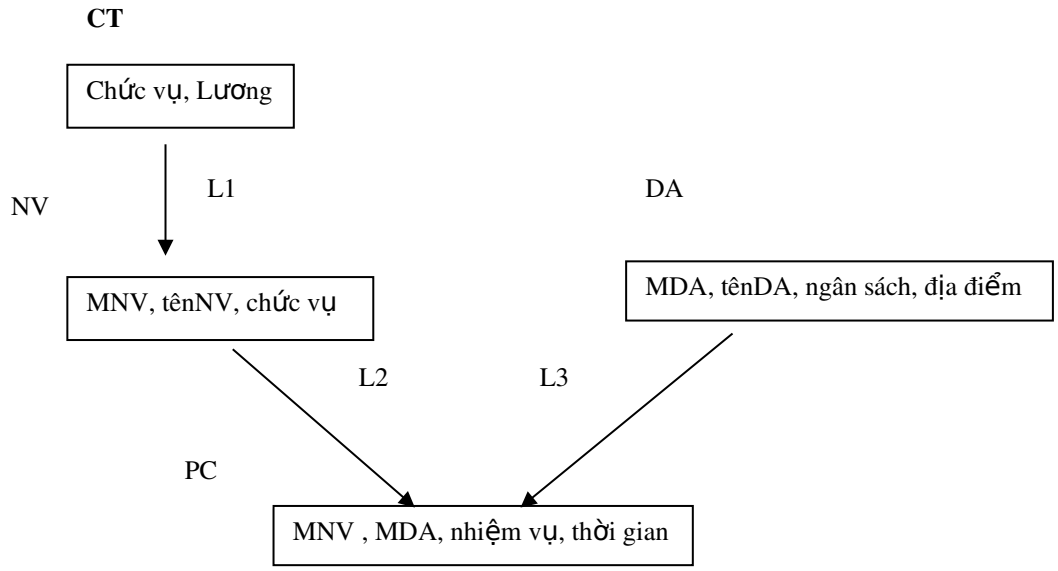
Trong phần này sẽ xem xét các thuật toán thực hiện các kiểu phân mảnh ngang. Trước tiên chúng ta nêu các thông tin cần thiết để thực hiện phân mảnh ngang.

Yêu cầu thông tin của phân mảnh ngang

a) Thông tin về cơ sở dữ liệu

Thông tin về CSDL muốn nói đến là lược đồ toàn cục và quan hệ gốc, các quan hệ con. Trong ngữ cảnh này, chúng ta cần biết được các quan hệ sẽ kết lại với nhau bằng phép nối hay bằng phép tính khác. với mục đích phân mảnh dẫn xuất, các vị từ được định nghĩa trên quan hệ khác, ta thường dùng mô hình thực thể - liên hệ (entity-relatiónhip model), vì trong mô hình này các mối liên hệ được biểu diễn bằng các đường nối có hướng (các cung) giữa các quan hệ có liên hệ với nhau qua một nối.

Thí dụ 1:



Hình 2.2. Biểu diễn mối liên hệ giữa các quan hệ nhờ các đường nối.

Hình trên trình bày một cách biểu diễn các đường nối giữa các quan hệ. chú ý rằng hướng của đường nối cho biết mối liên hệ một -nhiều. Chẳng hạn với mỗi chức vụ có nhiều nhân viên giữ chức vụ đó, vì thế chúng ta sẽ vẽ một đường nối từ quan hệ **CT (chi trả)** hướng đến **NV (nhân viên)**. Đồng thời mối liên hệ nhiều-nhiều giữa **NV** và **DA(dự án)** được biểu diễn bằng hai đường nối đến quan hệ **PC (phân công**).

Quan hệ nằm tại đầu (không mũi tên) của đường nối được gọi là chủ nhân (owner) của đường nối và quan hệ tại cuối đường nối (đầu mũi tên) gọi là thành viên (member).

Thí dụ 2:

Cho đường nối L1 của hình 2.2, các hàm owner và member có các giá trị sau:

Owner(L1) = CT

Member (L1) = NV

Thông tin định lượng cần có về CSDL là lực lượng (cardinality) của mỗi quan hệ R, đó là số bộ có trong R, được ký hiệu là card (R)

b) Thông tin về ứng dụng

Để phân tán ngoài thông tin định lượng Card(R) ta còn cần thông tin định tính cơ bản gồm các vị từ được dùng trong các câu vấn tin. Lượng thông tin này phụ thuộc bài toán cụ thể.

Nếu không thể phân tích được hết tất cả các ứng dụng để xác định những vị từ này thì ít nhất cũng phải nghiên cứu được các ứng dụng" quan trọng" nhất.

Vậy chúng ta xác định các vị từ đơn giản (simple predicate). Cho quan hệ R (A_1, A_2,…, A_n), trong đó A_i là một thuộc tính được định nghĩa trên một miền biến thiên $D(A_i)$ hay D_i..

Một vị từ đơn giản P được định nghĩa trên R có dạng:

P: A_i θ Value

Trong đó θ {=,<,≠, ≤, >, ≥} và

value được chọn từ miền biến thiên của A_i (value D_i).

Như vậy, cho trước lược đồ R, các miền trị D_i chúng ta có thể xác định được tập tất cả các vị từ đơn giản P_r trên R.

Vậy P_r ={P: A_i θ Value }. Tuy nhiên trong thực tế ta chỉ cần những tập con thực sự của P_r.

Thí dụ 3: Cho quan hệ Dự án như sau:

P_1 : TênDA = "thiết bị điều khiển"

P_2 : Ngân sách ≤ 200000

Là các vị từ đơn giản..

Chúng ta sẽ sử dụng ký hiệu Pr_i để biểu thị tập tất cả các vị từ đơn giản được định nghĩa trên quan hệ R_i. Các phần tử của Pr_i được ký hiệu là p_{ij}.

Các vị từ đơn giản thường rất dễ xử lý, các câu vấn tin thường chứa nhiều vị từ phức tạp hơn, là tổ hợp của các vị từ đơn giản. Một tổ hợp cần đặc biệt chú ý, được gọi là vị từ hội sơ cấp (minterm predicate), đó là hội (conjunction) của các vị từ đơn giản. Bởi vì chúng ta luôn có thể biến đổi một biểu thức Boole thành dạng chuẩn hội, việc sử dụng vị từ hội sơ cấp trong một thuật toán thiết kế không làm mất đi tính tổng quát.

Cho một tập Pr_i = {p_{i1}, p_{i2}, …, p_{im} } là các vị từ đơn giản trên quan hệ R_i, tập các vị từ hội sơ cấp M_i={m_{i1}, m_{i2}, …, m_{iz} } được định nghĩa là:

M_i={m_{ij} | m_{ij}=Λ p^*_{ik}} với $1 \le k \le m$, $1 \le j \le z$

Trong đó $p^*_{ik}=p_{ik}$ hoặc $p^*_{ik}=\neg p_{ik}$. Vì thế mỗi vị từ đơn giản có thể xuất hiện trong vị từ hội sơ cấp dưới dạng tự nhiên hoặc dạng phủ định.

Thí dụ 4:

Xét quan hệ CT:

chức vụ	Lương
Kỹ sư điện	40000
Phân tích hệ thống	34000
Kỹ sư cơ khí	27000
Lập trình	24000

Dưới đây là một số vị từ đơn giản có thể định nghĩa được trên PAY.

p1: chức vụ=" Kỹ sư điện"

p2: chức vụ=" Phân tích hệ thống "

p3: chức vụ=" Kỹ sư cơ khí "

p4: chức vụ=" Lập trình "

p5: Lương ≤ 30000

p6: Lương > 30000

Dưới đây là một số các vị từ hội sơ cấp được định nghĩa dựa trên các vị từ đơn giản này

m1: chức vụ=" Kỹ sư điện "Λ Lương ≤ 30000

m2: chức vụ =" Kỹ sư điện "Λ Lương > 30000

m3: ¬(chức vụ=" Kỹ sư điện ")Λ Lương ≤ 30000

m4: ¬(chức vụ=" Kỹ sư điện ")Λ Lương> 30000

m5: chức vụ=" Lập trình "Λ Lương ≤ 30000

m6: chức vụ=" Lập trình "Λ Lương > 30000

Chú ý:+ Phép lấy phủ định không phải lúc nào cũng thực hiện được. Thí dụ:xét hai vị từ đơn giản sau: Cận_dưới ≤ A; A Cận_trên. Tức là thuộc tính A có miền trị nằm trong cận dưới và cận trên, khi đó phần bù của chúng là:

¬(Cận_dưới ≤ A);

¬(A Cận_trên) không xác định được. Giá trị của A trong các phủ định này đã ra khỏi miền trị của A.

Hoặc hai vị từ đơn giản trên có thể được viết lại là:

Cận_dưới ≤ A Cận_trên có phần bù là: ¬(Cận_dưới ≤ A ≤ Cận_trên) không định nghĩa được. Vì vậy khi nghiên cứu những vấn đề này ta chỉ xem xét các vị từ đẳng thức đơn giản.

=> Không phải tất cả các vị từ hội sơ cấp đều có thể định nghĩa được.

+ Một số trong chúng có thể vô nghĩa đối với ngữ nghĩa của quan hệ Chi trả. Ngoài ra cần chú ý rằng m3 có thể được viết lại như sau:

m3: chức vụ ≠ "Kỹ sư điện " Λ Lương ≤ 30000

Theo những thông tin định tính về các ứng dụng, chúng ta cần biết hai tập dữ liệu.

Độ tuyển hội sơ cấp (minterm selectivity): số lượng các bộ của quan hệ sẽ được truy xuất bởi câu vấn tin được đặc tả theo một vị từ hội sơ cấp đã cho. chẳng hạn độ tuyển của m1 trong Thí dụ 4 là zero bởi vì không có bộ nào trong CT thỏa vị từ này. Độ tuyển của m2 là 1. Chúng ta sẽ ký hiệu độ tuyển của một hội sơ cấp m_i là sel (m_i).

Tần số truy xuất (access frequency): tần số ứng dụng truy xuất dữ liệu. Nếu $Q=\{q_1, q_2,....,q_q\}$ là tập các câu vấn tin, acc (q_i) biểu thị cho tần số truy xuất của q_i trong một khoảng thời gian đã cho.

Chú ý rằng mỗi hội sơ cấp là một câu vấn tin. Chúng ta ký hiệu tần số truy xuất của một hội sơ cấp là acc(m_i)

Phân mảnh ngang nguyên thuỷ

Phân mảnh ngang nguyên thuỷ được định nghĩa bằng một phép toán chọn trên các quan hệ chủ nhân của một lược đồ của CSDL. Vì thế cho biết quan hệ R, các mảnh ngang của R là các R_i:

$R_i = \sigma_{Fi}(R), \ 1 \le i \le z.$

Trong đó F_i là công thức chọn được sử dụng để có được mảnh R_i. Chú ý rằng nếu F_i có dạng chuẩn hội, nó là một vị từ hội sơ cấp (m_j).

Thí dụ 5: Xét quan hệ DA

MDA	TênDA	Ngân sách	Địa điểm
P1	Thiết bị đo đạc	150000	Montreal
P2	Phát triển dữ liệu	135000	New York
P3	CAD/CAM	250000	New York
P4	Bảo dưỡng	310000	Paris

Chúng ta có thể định nghĩa các mảnh ngang dựa vào vị trí dự án. Khi đó các mảnh thu được, được trình bày như sau:

$DA_1=\sigma_{\text{Địa điểm="Montreal"}}$ (DA)

$DA_2=\sigma_{\text{Địa điểm="New York"}}$ (DA)

$DA_3=\sigma_{\text{Địa điểm="Paris"}}$ (DA)

DA_1

MDA	TDA	Ngân sách	Địa điểm
P1	Thiết bị đo đạc	150000	Montreal

DA_2

MDA	TênDA	Ngân sách	Địa điểm
P2	Phát triển dữ liệu	135000	New York
P3	CAD/CAM	250000	New York

DA_3

MDA	TênDA	Ngân sách	Địa điểm
P4	thiết bị đo đạc	310000	Paris

Bây giờ chúng ta có thể định nghĩa một mảnh ngang chặt chẽ và rõ ràng hơn

Mảnh ngang Ri của quan hệ R có chứa tất cả các bộ R thỏa vị từ hội sơ cấp m_i

Một đặc tính quan trọng của các vị từ đơn giản là tính đầy đủ và tính cực tiểu.

- Tập các vị từ đơn giản Pr được gọi là đầy đủ nếu và chỉ nếu xác suất mỗi ứng dụng truy xuất đến một bộ bất kỳ thuộc về một mảnh hội sơ cấp nào đó được định nghĩa theo Pr đều bằng nhau.

Thí dụ 6: Xét quan hệ phân mảnh **DA** được đưa ra trong Thí dụ 5. Nếu tập ứng dụng Pr={Địa điểm=”Montreal”, Địa điểm=”New York ”, Địa điểm=”Paris”, Ngân sách 200000 } thì Pr không đầy đủ vì có một số bộ của **DA** không được truy xuất bởi vị từ **Ngân sách** 200000. Để cho tập vị từ này đầy đủ, chúng ta cần phải xét thêm vị từ **Ngân sách** > 200000 vào Pr. Vậy Pr={Địa điểm=”Montreal”, Địa điểm=”New York ”, Địa điểm=”Paris”, Ngân sách 200000 , Ngân sách> 200000 } là đầy đủ bởi vì mỗi bộ được truy xuất bởi đúng hai vị từ p của Pr. Tất nhiên nếu ta bớt đi một vị từ bất kỳ trong Pr thì tập còn lại không đầy đủ.

Lý do cần phải đảm bảo tính đầy đủ là vì các mảnh thu được theo tập vị từ đầy đủ sẽ nhất quán về mặt logic do tất cả chúng đều thoả vị từ hội sơ cấp. Chúng cũng đồng nhất và đầy đủ về mặt thống kê theo cách mà ứng dụng truy xuất chúng.

Vì thế chúng ta sẽ dùng một tập hợp gồm các vị từ đầy đủ làm cơ sở của phân mảnh ngang nguyên thủy.

- Đặc tính thứ hai của tập các vị từ là tính cực tiểu. Đây là một đặc tính cảm tính. Vị từ đơn giản phải có liên đới (relevant) trong việc xác định một mảnh. Một vị từ không tham gia vào một phân mảnh nào thì có thể coi vị từ đó là thừa. Nếu tất cả các vị từ của Pr đều có liên đới thì Pr là cực tiểu.

Thí dụ 7: Tập Pr được định nghĩa trong Thí dụ 6 là đầy đủ và cực tiểu. Tuy nhiên nếu chúng ta thêm vị từ **TênDA** =”thiết bị đo đạc” vào Pr, tập kết quả sẽ không còn cực tiểu bởi vì vị từ mới thêm vào không có liên đới ứng với Pr. Vị từ mới thêm vào không chia thêm mảnh nào trong các mảnh đã được tạo ra.

Khái niệm đầy đủ gắn chặt với mục tiêu của bài toán. Số vị từ phải đầy đủ theo yêu cầu của bài toán chúng ta mới thực hiện được những vấn đề đặt ra của bài toán. Khái niệm cực tiểu liên quan đến vấn đề tối ưu của bộ nhớ, tối ưu của các thao tác trên tập các câu vấn tin. Vậy khi cho trước một tập vị từ Pr để xét tính cực tiểu chúng ta có thể kiểm tra bằng cách vứt bỏ những vị từ thừa để có tập vị từ Pr’ là cực tiểu và tất nhiên Pr’ cũng là tập đầy đủ với Pr.

Thuật toán COM_MIN: Cho phép tìm tập các vị từ đầy đủ và cực tiểu Pr’ từ Pr. Chúng ta tạm quy ước:

Quy tắc 1: *Quy tắc cơ bản về tính đầy đủ và cực tiểu , nó khẳng định rằng một quan hệ hoặc một mảnh được phân hoạch ” thành ít nhất hai phần và chúng được truy xuất khác nhau bởi ít nhất một ứng dụng* “.

Thuật toán 1.1 COM_MIN

Input : R: quan hệ; Pr: tậpcác vị từ đơn giản;

Output: Pr': tập các vị từ cực tiểu và đầy đủ;

Declare

F: tập các mảnh hội sơ cấp;

Begin

Pr'= ; F = ;

For each vị từ p Pr if p phân hoạch R theo Quy tắc 1 then

Begin

Pr': = Pr' p;

Pr: = Pr – p;

F: = F p; {f_i là mảnh hội sơ cấp theo p_i }

End; {Chúng ta đã chuyển các vị từ có phân mảnh R vào Pr'}

Repeat

For each p Pr if p phân hoạch một mảnh f_k của Pr' theo quy tắc 1 then

Begin

Pr': = Pr' p;

Pr: = Pr – p;

F: = F p;

End;

Until Pr' đầy đủ {Không còn p nào phân mảnh f_k của Pr'}

For each p Pr', if p' mà p<=>p' then

Begin

Pr':= Pr'-p;

F:= F - f;

End;

End. {COM_MIN}

Thuật toán bắt dầu bằng cách tìm một vị từ có liên đới và phân hoạch quan hệ đã cho. Vòng lặp **Repeat-until** thêm các vị từ có phân hoạch các mảnh vào tập này, bảo đảm tính đầy đủ của Pr'. Đoạn cuối kiểm tra tính cực tiểu của Pr'. Vì thế cuối cùng ta có tập Pr' là cực tiểu và đầy đủ.

Bước hai của việc thiết kế phân mảnh nguyên thủy là suy dẫn ra tập các vị từ hội sơ cấp có thể được định nghĩa trên các vị từ trong tập Pr'. Các vị từ hội sơ cấp này xác định các mảnh "ứng cử viên" cho bước cấp phát. Việc xác định các vị từ hội sơ cấp là tầm thường; khó khăn chính là tập các vị từ hội sơ cấp có thể rất lớn (thực sự chúng tỷ lệ hàm mũ theo số lượng các vị từ đơn giản). trong bước kế tiếp chúng ta sẽ tìm cách làm giảm số lượng vị từ hội sơ cấp cần được định nghĩa trong phân mảnh.

Bước ba của quá trình thiết kế là loại bỏ một số mảnh vô nghĩa. Điều này được thực hiện bằng cách xác định những vị từ mâu thuẫn với tập các phép kéo theo (implication) I. Chẳng hạn nếu Pr'={p_1, p_2}, trong đó

P_1: att= value_1

P2: att=value_2

Và miền biến thiên của att là {value_1, value_2}, rõ ràng I chứa hai phép kéo theo với khẳng định:

I_1: (att=value_1) (att=value_2)

I_2: (att=value_1) (att=value_2)

Bốn vị từ hội sơ cấp sau đây được định nghĩa theo Pr':

M_1: (att=value_1) (att=value_2)

M2: (att=value_1) (att=value_2)

M3: (att=value_1) (att=value_2)

M4: (att=value_1) (att=value_2)

Trong trường hợp này các vị từ hội sơ cấp m_1, m_4 mâu thuẫn với các phép kéo theo I và vì thế bị loại ra khỏi M.

Thuật toán phân mảnh ngang nguyên thủy được trình bày trong thuật toán 1.2.

Thuật toán 1.2 PHORIZONTAL

Input: R: quan hệ; Pr: tập các vị từ đơn giản;

Output: M: tập các vị từ hội sơ cấp;

Begin

Pr':= COM_MIN(R, Pr);

Xác định tập M các vị từ hội sơ cấp;

Xác định tập I các phép kéo theo giữa các p_i Pr';

For each m_i M **do**

Begin

IF m_i mâu thuẫn với I **then**

M:= M-m_i

End;

End. {PHORIZONTAL}

Thí dụ 8: Chúng ta hãy xét quan hệ **DA**. Giả sử rằng có hai ứng dụng. Ứng dụng đầu tiên được đưa ra tại ba vị trí và cần tìm tên và ngân sách của các dự án khi cho biết vị trí. Theo ký pháp SQL câu vấn tin được viết là:

SELECT TênDA, Ngân sách

FROM DA

WHERE địa điểm=giá trị

Đối với ứng dụng này, các vị từ đơn giản có thể được dùng là:

P1: Địa điểm="Montreal"

P2: Địa điểm="New York"

P3: Địa điểm="Paris"

Ứng dụng thứ hai là những dự án có ngân sách dưới 200.000 đô la được quản lý tại một vị trí, còn những dự án có ngân sách lớn hơn được quản lý tại một vị trí thứ hai. Vì thế các vị từ đơn giản phải được sử dụng để phân mảnh theo ứng dụng thứ hai là:

P4: ngân sách≤200000

P5: ngân sách>200000

Nếu kiểm tra bằng thuật toán COM_MIN, tập Pr'={p1, p2, p3, p4, p5} rõ ràng đầy đủ và cực tiểu

Dựa trên Pr' chúng ta có thể định nghĩa sáu vị từ hội sơ cấp sau đây tạo ra M:

M1: (Địa điểm="Montreal") (ngân sách≤200000)

M2: (Địa điểm="Montreal") (ngân sách>200000)

M3: (Địa điểm="New York") (ngân sách≤200000)

M4: (Địa điểm="New York") (ngân sách>200000)

M5: (Địa điểm="Paris") (ngân sách≤200000)

M6: (Địa điểm="Paris") (ngân sách>200000)

Đây không phải là các vị từ hội sơ cấp duy nhất có thể được tạo ra. Chẳng hạn vẫn có thể định nghĩa các vị từ:

p1 p2 p3 p4 p5

Tuy nhiên các phép kéo hiển nhiên là:

I_1: p1 p2 p3

I_2: p2 p1 p3

I_3: p3 p1 p2

I_4: p4 p5

I_5: p5 p4

I_6: p4 p5

I_7: p5 p4

Cho phép loại bỏ những vị từ hội sơ cấp này và chúng ta còn lại m1 đến m6.

Cần nhớ rằng các phép kéo theo phải được định nghĩa theo ngữ nghĩa của CSDL, không phải theo các giá trị hiện tại. Một số mảnh được định nghĩa theo M={m1,…,m6} có thể rỗng nhưng chúng vẫn là các mảnh. Kết quả phân mảnh nguyên thuỷ cho **DA** là tạo ra sáu mảnh F_{DA}={DA_1, DA_2, DA_3, DA_4, DA_5, DA_6}, ở đây có hai mảnh rỗng là {DA_2, DA_5 }

DA_1

MDA	TênDA	Ngân sách	Địa điểm
P1	Thiết bị đo	150000	Montreal

	đạc		

DA_3

MDA	TênDA	Ngân sách	Địa điểm
P2	Phát triển dữ liệu	135000	New York

DA_4

MDA	TênDA	Ngân sách	Địa điểm
P3	CAD/CAM	250000	New York

DA_6

MDA	TênDA	Ngân sách	Địa điểm
P4	bảo dưỡng	310000	Paris

Phân mảnh ngang dẫn xuất

Phân mảnh ngang dẫn xuất được định nghĩa trên một quan hệ thành viên của đường nối dựa theo phép toán chọn trên quan hệ chủ nhân của đường nối đó.

Như thế nếu cho trước một đường nối L, trong đó owner (L)=S và member(L)=R, và các mảnh ngang dẫn xuất của R được định nghĩa là:

$R_i = R |>< S_i$, $1 \le i \le w$

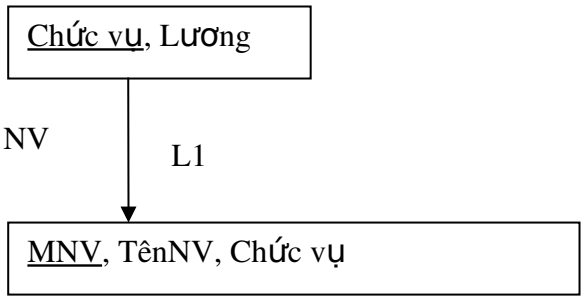
Trong đó w là số lượng các mảnh được định nghĩa trên R, và $S_i = {}_{Fi}(S)$ với F_i là công thức định nghĩa mảnh ngang nguyên thuỷ S_i

Thí dụ 9: Xét đường nối

NV

MNV	TênNV	Chức vụ
E1	J.Doe	Kỹ sư điện
E2	M.Smith	Phân tích
E2	M.Smith	Phân tích
E3	A.Lee	Kỹ sư cơ khí
E3	A.Lee	Kỹ sư cơ khí
E4	J.Miller	Programmer
E5	B.Casey	Phân tích hệ thống
E6	L.Chu	Kỹ sư điện
E7	R.david	Kỹ sư cơ khí
E8	J.Jones	Phân tích hệ thống

CT



thế thì chúng ta có thể nhóm các kỹ sư thành hai nhóm tùy theo lương: nhóm có lương từ 30.000 đôla trở lên và nhóm có lương dưới 30.000 đô la. Hai mảnh Nhân viên$_1$ và Nhân viên$_2$ được định nghĩa như sau:

$NV_1 = NV \mid >< CT_1$

$NV_2 = NV \mid >< CT_2$

Trong đó $CT_1 = {}_{Luong\ 30000}(CT)$

$CT_2 = {}_{Luong>30000}(CT)$

CT_1

Chức vụ	Lương
Kỹ sư cơ khí	27000
Lập trình	24000

CT_2

Chức vụ	Lương
Kỹ sư điện	40000
Phân tích hệ thống	34000

Kết quả phân mảnh ngang dẫn xuất của quan hệ **NV** như sau**:**

NV_1

MNV	TênNV	Chức vụ
E3	A.Lee	Kỹ sư cơ khí
E4	J.Miller	Lập trình viên
E7	R.David	Kỹ sư cơ khí

NV_2

MNV	TênNV	Chức vụ
E1	J.Doe	Kỹ sư điện
E2	M.Smith	Phân tích
E5	B.Casey	Phân tích hệ thống
E6	L.Chu	Kỹ sư điện
E8	J.Jones	Phân tích hệ thống

Chú ý:

+ Muốn thực hiện phân mảnh ngang dẫn xuất, chúng ta cần ba nguyên liệu (input): 1. Tập các phân hoạch của quan hệ chủ nhân (Thí dụ: CT1, CT2).

2. Quan hệ thành viên

3. Tập các vị từ nối nửa giữa chủ nhân và thành viên (Chẳng hạn CT.Chucvu = NV.Chucvu).

+ Vấn đề phức tạp cần chú ý: Trong lược đồ CSDL, chúng ta hay gặp nhiều đường nối đến một quan hệ R. Như thế có thể có nhiều cách phân mảnh cho quan hệ R. Quyết định chọn cách phân mảnh nào cần dựa trên hai tiêu chuẩn sau:

1. Phân mảnh có đặc tính nối tốt hơn

2. Phân mảnh được sử dụng trong nhiều ứng dụng hơn.

Tuy nhiên, việc áp dụng các tiêu chuẩn trên còn là một vấn đề rắc rối.

Thí dụ 10: Chúng ta tiếp tục với thiết kế phân tán cho CSDL đã bắt đầu từ Thí dụ 9. Và quan hệ NV phân mảnh theo CT. Bây giờ xét ASG. Giả sử có hai ứng dụng sau:

1. Ứng dụng 1: Tìm tên các kỹ sư có làm việc tại một nơi nào đó. Ứng dụng này chạy ở cả ba trạm và truy xuất cao hơn các kỹ sư của các dự án ở những vị trí khác.

2. Ứng dụng 2: Tại mỗi trạm quản lý, nơi quản lý các mẫu tin nhân viên, người dùng muốn truy xuất đến các dự án đang được các nhân viên này thực hiện và cần biết xem họ sẽ làm việc với dự án đó trong bao lâu.

Kiểm định tính đúng đắn

Bây giờ chúng ta cần phải kiểm tra tính đúng của phân mảnh ngang.

a. Tính đầy đủ

+ Phân mảnh ngang nguyên thuỷ: Với điều kiện các vị từ chọn là đầy đủ, phân mảnh thu cũng được đảm bảo là đầy đủ, bởi vì cơ sở của thuật toán phân mảnh là tập các vị từ cực tiểu và đầy đủ Pr', nên tính đầy đủ được bảo đảm với điều kiện không có sai sót xảy ra.

+ Phân mảnh ngang dẫn xuất: Có khác chút ít, khó khăn chính ở đây là do vị từ định nghĩa phân mảnh có liên quan đến hai quan hệ. Trước tiên chúng ta hãy định nghĩa qui tắc đầy đủ một cách hình thức.

R là quan hệ thành viên của một đường nối mà chủ nhân là quan hệ S. Gọi A là thuộc tính nối giữa R và S, thế thì với mỗi bộ t của R, phải có một bộ t' của S sao cho

$$t.A=t'.A$$

Quy tắc này được gọi là **ràng buộc toàn vẹn** hay **toàn vẹn tham chiếu**, bảo đảm rằng mọi bộ trong các mảnh của quan hệ thành viên đều nằm trong quan hệ chủ nhân.

b. Tính tái thiết được

Tái thiết một quan hệ toàn cục từ các mảnh được thực hiện bằng toán tử hợp trong cả phân mảnh ngang nguyên thủy lẫn dẫn xuất, Vì thế một quan hệ R với phân mảnh $F_r=\{R_1, R_2,\ldots,R_m\}$ chúng ta có

$$\mathbf{R = R_i\ ,\quad R_i\quad F_R}$$

c. Tính tách rời

Với phân mảnh nguyên thuỷ tính tách rời sẽ được bảo đảm miễn là các vị từ hội sơ cấp xác định phân mảnh có tính loại trừ tương hỗ (mutually exclusive). Với phân mảnh dẫn xuất tính tách rời có thể bảo đảm nếu đồ thị nối thuộc loại đơn giản.

2.3. Phân mảnh dọc

Một phân mảnh dọc cho một quan hệ R sinh ra các mảnh $R_1, R_2,..,R_r$, mỗi mảnh chứa một tập con thuộc tính của R và cả khoá của R. Mục đích của phân mảnh dọc là phân hoạch một quan hệ thành một tập các quan hệ nhỏ hơn để nhiều ứng dụng chỉ cần chạy trên một mảnh. Một phân mảnh "tối ưu"là phân mảnh sinh

ra một lược đồ phân mảnh cho phép giảm tối đa thời gian thực thi các ứng dụng chạy trên mảnh đó.

Phân mảnh dọc tất nhiên là phức tạp hơn so với phân mảnh ngang. Điều này là do tổng số chọn lựa có thể của một phân hoạch dọc rất lớn.

Vì vậy để có được các lời giải tối ưu cho bài toán phân hoạch dọc thực sự rất khó khăn. Vì thế lại phải dùng các phương pháp khám phá (heuristic). Chúng ta đưa ra hai loại heuristic cho phân mảnh dọc các quan hệ toàn cục.

- Nhóm thuộc tính: Bắt đầu bằng cách gán mỗi thuộc tính cho một mảnh, và tại mỗi bước, nối một số mảnh lại cho đến khi thỏa một tiêu chuẩn nào đó. Kỹ thuật này được được đề xuất lần đầu cho các CSDL tập trung và về sau được dùng cho các CSDL phân tán.

- Tách mảnh: Bắt đầu bằng một quan hệ và quyết định cách phân mảnh có lợi dựa trên hành vi truy xuất của các ứng dụng trên các thuộc tính.

Bởi vì phân hoạch dọc đặt vào một mảnh các thuộc tính thường được truy xuất chung với nhau, chúng ta cần có một giá trị đo nào đó để định nghĩa chính xác hơn về khái niệm "chung với nhau". Số đo này gọi là tụ lực hay lực hút (affinity) của thuộc tính, chỉ ra mức độ liên đới giữa các thuộc tính.

Yêu cầu dữ liệu chính có liên quan đến các ứng dụng là tần số truy xuất của chúng. gọi $Q=\{q_1, q_2,\ldots,q_q\}$ là tập các vấn tin của người dùng (các ứng dụng) sẽ chạy trên quan hệ $R(A_1, A_2,\ldots,A_n)$. Thế thì với mỗi câu vấn tin q_i và mỗi thuộc tính A_j, chúng ta sẽ đưa ra một giá trị sử dụng thuộc tính, ký hiệu $use(q_i, A_j)$ được định nghĩa như sau:

$$use(q_i, A_j)= \begin{cases} 1 & \text{nếu thuộc tính } A_j \text{ được vấn tin } q_i \text{ tham chiếu} \\ 0 & \text{trong trường hợp ngược lại} \end{cases}$$

Các véctơ use(q_i,) cho mỗi ứng dụng rất dễ định nghĩa nếu nhà thiết kế biết được các ứng dụng sẽ chạy trên CSDL.

Thí dụ 11:

Xét quan hệ **DA**, giả sử rằng các ứng dụng sau đây chạy trên các quan hệ đó. Trong mỗi trường hợp chúng ta cũng đặc tả bằng SQL.

q1: Tìm ngân sách của một dự án, cho biết mã của dự án

SELECT Ngân sách

FROM DA

WHERE MDA=giá trị

q2: Tìm tên và ngân sách của tất cả mọi dự án

SELECT TênDA, ngân sách

FROM DA

q3: Tìm tên của các dự án được thực hiện tại một thành phố đã cho

SELECT tênDA

FROM DA

WHERE địa điểm=giá trị

q4: Tìm tổng ngân sách dự án của mỗi thành phố

SELECT SUM (ngân sách)

FROM DA

WHERE Địa điểm=giá trị

Dựa theo bốn ứng dụng này, chúng ta có thể định nghĩa ra các giá trị sử dụng thuộc tính. Để cho tiện về mặt ký pháp, chúng ta gọi A_1=MDA, A_2=TênDA, A_3=Ngân sách, A_4=địa điểm. Giá trị sử dụng được định nghĩa dưới dạng ma trận, trong đó mục (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

Tụ lực của các thuộc tính

Giá trị sử dụng thuộc tính không đủ để làm cơ sở cho việc tách và phân mảnh. Điều này là do chúng không biểu thị cho độ lớn của tần số ứng dụng. Số đo lực hút (affinity) của các thuộc tính aff(A_i, A_j), biểu thị cho cầu nối (bond) giữa hai thuộc tính của một quan hệ theo cách chúng được các ứng dụng truy xuất, sẽ là một đại lượng cần thiết cho bài toán phân mảnh.

Xây dựng công thức để đo lực hút của hai thuộc tính A_i, A_j.

Gọi k là số các mảnh của R được phân mảnh. Tức là R = R_1 ….R_k.

Q= {q_1, q_2,…,q_m} là tập các câu vấn tin (tức là tập các ứng dụng chạy trên quan hệ R). Đặt Q(A, B) là tập các ứng dụng q của Q mà use(q, A).use(q, B) = 1.

Nói cách khác:

Q(A, B) = {q Q: use(q, A) =use(q, B) = 1}

Thí dụ dựa vào ma trận trên ta thấy $Q(A_1,A_1)$ = {q_1}, $Q(A_2,A_2)$ = {q_2, q_3}, $Q(A_3,A_3)$ = {q_1,q_2, q_4}, $Q(A_4,A_4)$ = {q_3, q_4}, $Q(A_1,A_2)$ = rỗng, $Q(A_1,A_3)$ = {q_1}, $Q(A_2,A_3)$ = {q_2},..

Số đo lực hút giữa hai thuộc tính A_i, A_j được định nghĩa là:

$$aff(A_i, A_j) = \sum_{q_k \in Q(A_i, A_j)} \sum_{l=1}^{R_l} ref_l(q_k) acc_l(q_k)$$

Hoặc:

$$aff(A_i, A_j) = \sum_{Use(q_k, A_i)=1} \sum_{Use(q_k, A_j)=1} \sum_{R_l} ref_l(q_k) acc_l(q_k)$$

Trong đó $ref_l(q_k)$ là số truy xuất đến các thuộc tính (A_i, A_j) cho mỗi ứng dụng q_k tại vị trí R_l và $acc_l(q_k)$ là số đo tần số truy xuất ứng dụng q_k đến các thuộc tính A_i, A_j tại vị trí l. Chúng ta cần lưu ý rằng trong công thức tính aff (A_i, A_j) chỉ xuất hiện các ứng dụng q mà cả A_i và A_j đều sử dụng.

Kết quả của tính toán này là một ma trận đối xứng n x n, mỗi phần tử của nó là một số đo được định nghĩa ở trên. Chúng ta gọi nó là ma trận lực tụ (lực hút hoặc ái lực) thuộc tính (AA) (attribute affinity matrix).

Thí dụ 12: Chúng ta hãy tiếp tục với Thí dụ 11. Để cho dơn giản chúng ta hãy giả sử rằng $ref_l(q_k)$ =1 cho tất cả q_k và R_l. Nếu tần số ứng dụng là:

$Acc_1(q1) = 15$ $Acc_2(q1) = 20$ $Acc_3(q1) = 10$

$Acc_1(q2) = 5$ $Acc_2(q2) = 0$ $Acc_3(q2) = 0$

$Acc_1(q3) = 25$ $Acc_2(q3) = 25$ $Acc_3(q3) = 25$

$Acc_1(q4) = 3$ $Acc_2(q4) = 0$ $Acc_3(q1) = 0$

Số đo lực hút giữa hai thuộc tính A_1 và A_3 là:

$$Aff(A_1, A_3) = \sum_{k=1}^{1} \sum_{t=1}^{3} acc_t(q_k) = acc_1(q_1)+acc_2(q_1)+acc_3(q_1) = 45$$

Tương tự tính cho các cặp còn lại ta có ma trận ái lực sau:

	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

Thuật toán năng lượng nối BEA (Bond Energy Algorithm)

Đến đây ta có thể phân R làm các mảnh của các nhóm thuộc tính dựa vào sự liên đới (lực hút) giữa các thuộc tính, thí dụ tụ lực của A_1, A_3 là 45, của A_2, A_4 là 75, còn của A_1, A_2 là 0, của A_3, A_4 là 3… Tuy nhiên, phương pháp tuyến tính sử dụng trực tiếp từ ma trận này ít được mọi người quan tâm và sử dụng. Sau đây chúng ta xét một phương pháp dùng thuật toán năng lượng nối BEA của Hoffer and Severance, 1975 và Navathe., 1984.

1. Nó được thiết kế đặc biệt để xác định các nhóm gồm các mục tương tự, khác với một sắp xếp thứ tự tuyến tính của các mục.

2. Các kết quả tụ nhóm không bị ảnh hưởng bởi thứ tự đưa các mục vào thuật toán.

3. Thời gian tính toán của thuật toán có thể chấp nhận được là $O(n^2)$, với n là số lượng thuộc tính.

4. Mối liên hệ qua lại giữa các nhóm thuộc tính tụ có thể xác định được.

Thuật toán BEA nhận nguyên liệu là một ma trận ái lực thuộc tính (AA), hoán vị các hàng và cột rồi sinh ra một ma trận ái lực tụ (CA) (Clustered affinity matrix). Hoán vị được thực hiện sao cho số đo ái lực chung AM (Global Affinity Measure) là lớn nhất. Trong đó AM là đại lượng:

$$AM= \sum_{i=1}^{n} \sum_{j=1}^{n} aff(A_i, A_j)[aff(A_i, A_{j-1})+aff(A_i, A_{j+1})+aff(A_{i-1}, A_j)+ aff(A_{i+1}, A_j)]$$

Với $aff(A_0, A_j)=aff(A_i, A_0)=aff(A_{n+1}, A_j)=aff(A_i, A_{n+1})=0$ cho i,j

Tập các điều kiện cuối cùng đề cập đến những trường hợp một thuộc tính được đặt vào CA ở về bên trái của thuộc tính tận trái hoặc ở về bên phải của thuộc tính tận phải trong các hoán vị cột, và bên trên hàng trên cùng và bên dưới hàng cuối cùng trong các hoán vị hàng. Trong những trường hợp này, chúng ta cho 0 là giá trị lực hút aff giữa thuộc tính đang được xét và các lân cận bên trái hoặc bên phải (trên cùng hoặc dưới đáy) của nó hiện chưa có trong CA.

Hàm cực đại hoá chỉ xét những lân cận gần nhất, vì thế nó nhóm các giá trị lớn với các giá trị lớn , giá trị nhỏ với giá trị nhỏ. Vì ma trận lực hút thuộc tính AA có tích chất đối xứng nên hàm số vừa được xây dựng ở trên thu lại thành:

$AM= \sum_{i=1}^{n} \sum_{j=1}^{n} aff(A_i, A_j)[aff(A_i, A_{j-1})+aff(A_i, A_{j+1})]$

Quá trình sinh ra ma trận tụ lực (CA) được thực hiện qua ba bước:

Bước 1: Khởi gán:

Đặt và cố định một trong các cột của AA vào trong CA. Thí dụ cột 1, 2 được chọn trong thuật toán này.

Bước 2: Thực hiện lặp

Lấy lần lượt một trong n-i cột còn lại (trong đó i là số cột đã được đặt vào CA) và thử đặt chúng vào trong i+1 vị trí còn lại trong ma trận CA. Chọn nơi đặt sao cho cho ái lực chung AM lớn nhất. Tiếp tục lặp đến khi không còn cột nào để dặt.

Bước 3: Sắp thứ tự hàng

Một khi thứ tự cột đã được xác định, các hàng cũng được đặt lại để các vị trí tương đối của chúng phù hợp với các vị trí tương đối của cột.

Thuật toán BEA

Input: AA - ma trận ái lực thuộc tính;

Output: CA - ma trận ái lực tụ sau khi đã sắp xếp lại các hàng các cột;

Begin

{Khởi gán: cần nhớ rằng Â là một ma trận n x n}

CA(, 1) AA(, 1)

CA(, 2) AA(, 2)

Index:=3

while index <= n do *{chọn vị trí "tốt nhất" cho thuộc tính A_{index}}*

begin

for i :=1 to index-1 by 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 nơi đặt, được cho bởi giá trị cont lớn nhất;

for i: = index downto loc do *{xáo trộn hai ma trận}*

CA(, j) CA(, j-1);

CA(, loc) AA(, index);

index index+1;

end-while

Sắp thứ tự các hàng theo thứ tự tương đối của cột.

end. {BEA}

Để hiểu rõ thuật toán chúng ta cần biết cont(*,*,*). Cần nhắc lại số đo ái lực chung AM đã được định nghĩa là:

$AM= \sum_{i=1}^{n} \sum_{j=1}^{n} aff(A_i, A_j)[aff(A_i, A_{j-1})+aff(A_i, A_{j+1})]$

Và có thể viết lại:

$AM = \sum_{i=1}^{n} \sum_{j=1}^{n} [aff(A_i, A_j)\ aff(A_i, A_{j-1})+aff(A_i, A_j)\ aff(A_i, A_{j+1})]$

$= \sum_{j=1}^{n}[\sum_{i=1}^{n} aff(A_i, A_j)\ aff(A_i, A_{j-1})+ \sum_{i=1}^{n} aff(A_i, A_j)\ aff(A_i, A_{j+1})]$

Ta định nghĩa cầu nối (Bond) giữa hai thuộc tính A_x, và A_y là:

$$Bond(A_x, A_y)= \sum_{z=1}^{n} aff(A_z, A_x)aff(A_z, A_y)$$

Thế thì có thể viết lại AM là:

$$AM = \sum_{j=1}^{n}[Bond(A_i, A_{j-1})+Bond(A_i, A_{j+1})]$$

Bây giờ xét n thuộc tính sau:

$$A_1\ A_2\ \ldots A_{i-1}\quad A_iA_j\quad A_{j+1}\ \ldots A_n$$

Với $A_1\ A_2\ \ldots A_{i-1}$ thuộc nhóm AM' và $A_iA_j\quad A_{j+1}\ \ldots A_n$ thuộc nhóm AM"

Khi đó số đo lực hút chung cho những thuộc tính này có thể viết lại:

$AM_{old} = AM' + AM"+ bond(A_{i-1}, A_i) + bond(A_i, A_j) + bond(A_j, A_i)+$

$bond(bond(A_{j+1}, A_j) = \sum_{l=1}^{n}[bond(A_l, A_{l-1})+bond(A_i, A_{l+1})] + \sum_{l=i+1}^{n}[bond(A_l, A_{l-1})+bond(A_i, A_{l+1})] + 2bond(A_i, A_l))$

Bây giờ xét đến việc đặt một thuộc tính mới A_k giữa các thuộc tính A_i và A_j trong ma trận lực hút tụ. Số đo lực hút chung mới có thể được viết tương tự như:

$AM_{new} = AM' + AM"+ bond(A_i, A_k) + bond(A_k, A_i) + bond(A_k, A_j)+ bond(A_j, A_k) = AM' + AM"+ 2bond(A_i, A_k) + 2bond(A_k, A_j)$

Vì thế đóng góp thực (net contribution) cho số đo ái lực chung khi đặt thuộc tính A_k giữa A_i và A_j là:

$Cont(A_i, A_k, A_j) = AM_{new} - AM_{old} = 2Bond(A_i, A_k) + 2Bond(A_k, A_j) - 2Bond(A_i, A_j)$

$Bond(A_0, A_k)=0$. Nếu thuộc tính A_k đặt bên phải thuộc tính tận bên phải vì chưa có thuộc tính nào được đặt ở cột k+1 của ma trận CA nên $bond(A_k, A_{k+1})=0$.

***Thí dụ 13*:** Ta xét ma trận được cho trong Thí dụ 12 và tính toán phần đóng góp khi di chuyển thuộc tính A_4 vào giữa các thuộc tính A_1 và A_2, được cho bằng công thức:

$Cont(A_1, A_4, A_2)= 2bond(A_1, A_4)+ 2bond(A_4, A_2)-2bond(A_1, A_2)$

Tính mỗi số hạng chúng ta được:

$Bond(A_1, A_4) = {}^{4}_{z=1}aff(A_z, A_1)aff(A_z, A_4) = aff(A_1,A_1)\ aff(A_1,A_4) + aff(A_2,A_1)\ aff(A_2,A_4) + aff(A_1,A_3)\ aff(A_3,A_4) + aff(A_1,A_4)\ aff(A_4,A_4)$

$= 45*0 +0*75+ 45*3+0*78 = 135$

$Bond(A_4, A_2)= 11865$

$Bond(A_1,A_2) = 225$

Vì thế $cont(A_1, A_4) = 2*135+2*11865+2*225 = 23550$

Thí dụ 14:

Chúng ta hãy xét quá trình gom tụ các thuộc tính của quan hệ Dự án và dùng ma trận ái lực thuộc tính AA.

bước khởi đầu chúng ta chép các cột 1 và 2 của ma trận AA vào ma trận CA và bắt đầu thực hiện từ cột thứ ba. Có 3 nơi có thể đặt được cột 3 là: (3-1-2), (1, 3, 2) và (1, 2, 3). Chúng ta hãy tính đóng góp số ái lực chung của mỗi khả năng này.

thứ tự (0-3-1):

$cont(A_0, A_3, A_1) = 2bond(A_0, A_3)+ 2bond(A_3, A_1) - 2bond(A_0, A_1)$

$bond(A_0, A_3) = bond(A_0, A_1)=0$

$bond(A_3, A_1) = 45*48+5*0+53*45+3*0=4410$

$cont(A_0,\ A_3, A_1) = 8820$

thứ tự (1-3-2)

$cont\ (A_1,\ A_3, A_2)=10150$

th ứ tự (2-3-4)

cont (A_2, A_3, A_4)=1780

Bởi vì đóng góp của thứ tự (1-2-3) là lớn nhất, chúng ta đặt A_3 vào bên phải của A_1. Tính toán tương tự cho A_4 chỉ ra rằng cần phải đặt nó vào bên phải của A_2. Cuối cùng các hàng được tổ chức với cùng thứ tự như các cột và các hàng được trình bày trong hình sau:

	A_1	A_2
A_1	45	0
A_2	0	80
A_3	45	5
A_4	0	75

(a)

	A_1	A_3	A_2
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

(b)

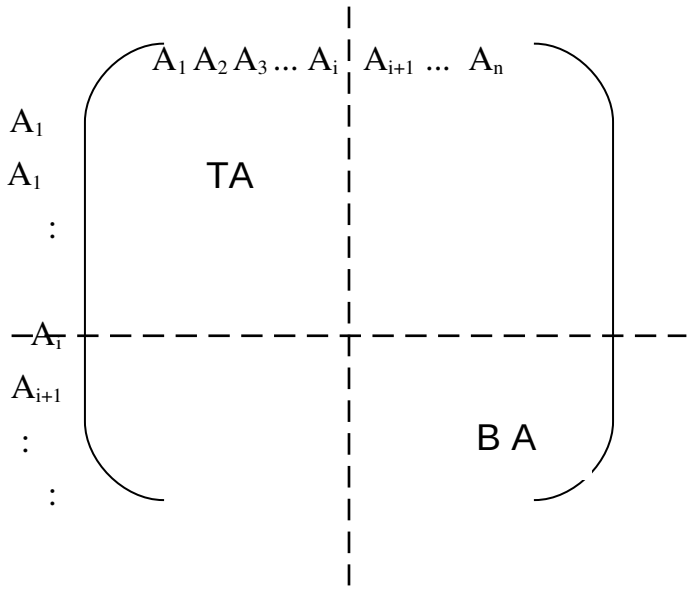
	A_1	A_3	A_2	A_4
A_1	45	45	0	0
A_3	45	53	5	3
A_2	0	5	80	75
A_4	0	3	75	78

(d)

trong hình trên chúng ta thấy quá trình tạo ra hai tụ: một ở góc trên trái chứa các giá trị ái lực nhỏ, còn tụ kia ở dưới góc phải chứa các giá trị ái lực cao. Quá trình phân tụ này chỉ ra cách thức tách các thuộc tính của **Dự án**. Tuy nhiên, nói chung thì ranh rới các phần tách không hoàn toàn rõ ràng. Khi ma trận CA lớn, thường sẽ có nhiều tụ hơn được tạo ra và nhiều phân hoạch được chọn hơn. Do vậy cần phải tiếp cận bài toán một cách có hệ thống hơn.

Thuật toán phân hoạch

Mục đích của hành động tách thuộc tính là tìm ra các tập thuộc tính được truy xuất cùng nhau hoặc hầu như là các tập ứng dụng riêng biệt. Xét ma trân thuộc tính tụ:



A_n

Nếu một điểm nằm trên đường chéo được cố định, hai tập thuộc tính này được xác định. Một tập $\{A_1, A_2,..., A_i\}$ nằm tại góc trên trái và tập thứ hai $\{A_{i+1}, A_{i+2},..., A_n\}$ nằm tại góc bên phải và bên dưới điểm này. Chúng ta gọi 2 tập lần lượt là TA, BA. Tập ứng dụng $Q=\{q_1, q_2,...,q_q\}$ và định nghĩa tập ứng dụng chỉ truy xuất TA, chỉ truy xuất BA hoặc cả hai, những tập này được định nghĩa như sau:

$AQ(q_i) = \{A_j \mid use(q_i, A_j)=1\}$

$TQ = \{q_i \mid AQ(q_i)$ TA$\}$

$BQ = \{q_i \mid AQ(q_i)$ BA$\}$

$OQ = Q - \{TQ$ BQ$\}$

Ở đây nảy sinh bài toán tối ưu hoá. Nếu có n thuộc tính trong quan hệ thì sẽ có n-1 vị trí khả hữu có thể là điểm phân chia trên đường chéo chính của ma trận thuộc tính tụ cho quan hệ đó. Vị trí tốt nhất để phân chia là vị trí sinh ra tập TQ và BQ sao cho tổng các truy xuất chỉ một mảnh là lớn nhất còn tổng truy xuất cả hai mảnh là nhỏ nhất. Vì thế chúng ta định nghĩa các phương trình chi phí như sau:

$$CQ = \sum_{q_i \in Q} \sum_{S_j} ref_j(q_i)acc_j(q_i)$$

$$CTQ = \sum_{q_i \in TQ} \sum_{S_j} ref_j(q_i)acc_j(q_i)$$

$$CBQ = \sum_{q_i \in BQ} \sum_{S_j} ref_j(q_i)acc(q_i)$$

$$COQ = \sum_{q_i \in OQ} \sum_{S_j} ref_j(q_i)acc(q_i)$$

Mỗi phương trình ở trên đếm tổng số truy xuất đến các thuộc tính bởi các ứng dụng trong các lớp tương ứng của chúng. Dựa trên số liệu này, bài toán tối ưu hoá được định nghĩa là bài toán tìm điểm x (1 x n) sao cho biểu thức:

$$Z=CTQ+CBQ-COQ^2$$

lớn nhất. Đặc trưng quan trọng của biểu thức này là nó định nghĩa hai mảnh sao cho giá trị của CTQ và CBQ càng gần bằng nhau càng tốt. Điều này cho phép cân bằng tải trọng xử lý khi các mảnh được phân tán đến các vị trí khác nhau. Thuật toán phân hoạch có độ phức tạp tuyến tính theo số thuộc tính của quan hệ, nghĩa là O(n).

Thuật toán PARTITION

Input: CA: ma trận ái lực tụ; R: quan hệ; ref: ma trận sử dụng thuộc tính;

acc: ma trận tần số truy xuất;

Output: F: tập các mảnh;

Begin

{xác định giá trị z cho cột thứ nhất}

{các chỉ mục trong phương trình chi phí chỉ ra điểm tách}

tính CTQ_{n-1}

tính CBQ_{n-1}

tính COQ_{n-1}

best $CTQ_{n-1}*CBQ_{n-1} - (COQ_{n-1})^2$

do {xác định cách phân hoạch tốt nhất}

begin

for i from n-2 to 1 by -1 do

begin

tính CTQ_i

tính CBQ_i

tính COQ_i

z $CTQ_i*CBQ_i - (COQ_i)^2$

if z > best then

begin

best z

ghi nhận điểm tách bên vào trong hành động xê dịch

end-if

end-for

gọi SHIFT(CA)

end-begin

until không thể thực hiện SHIFT được nữa

Xây dựng lại ma trận theo vị trí xê dịch

R_1 $_{TA}(R)$ K {K là tập thuộc tính khoá chính của R}

R_2 $_{BA}(R)$ K

F $\{R_1, R_2\}$

End. {partition}

Áp dụng cho ma trận CA từ quan hệ **dự án**, kết quả là định nghĩa các mảnh $F_{\text{dự án}}$={Dự án$_1$, Dự án$_2$}

Trong đó: Dự án$_1$=$\{A_1, A_3\}$ và Dự án$_2$= $\{A_1, A_2, A_4\}$. Vì thế

Dự án$_1$={Mã dự án, Ngân sách}

Dự án$_2$={Mã dự án, Tên dự án, Địa điểm}

(ở đây Mã dự án là thuộc tính khoá của **Dự án**)

Kiểm tra tính đúng đắn:

Tính đầy đủ: được bảo đảm bằng thuật toán PARTITION vì mỗi thuộc tính của quan hệ toàn cục được đưa vào một trong các mảnh.

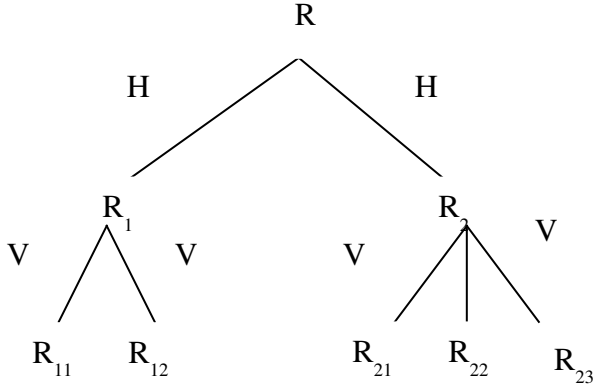
Tính tái thiết được: đối với quan hệ R có phân mảnh dọc $F_R=\{R_1, R_2,...., R_r\}$ và các thuộc tính khoá K

$$\mathbf{R=} \bowtie_{\mathbf{K}} \mathbf{R_i}, \quad \mathbf{R_i} \quad \mathbf{F_R}$$

Do vậy nếu điều kiện mỗi R_i là đầy đủ phép toán nối sẽ tái thiết lại đúng R. Một điểm quan trọng là mỗi mảnh R_i phải chứa các thuộc tính khoá của R.

2.5. Phân mảnh hỗn hợp

Trong đa số các trường hợp, phân mảnh ngang hoặc phân mảnh dọc đơn giản cho một lược đồ CSDL không đủ đáp ứng các yêu cầu từ ứng dụng. Trong trường hợp đó phân mảnh dọc có thể thực hiện sau một số mảnh ngang hoặc ngược lại, sinh ra một lối phân hoạch có cấu trúc cây. Bởi vì hai chiến lược này được áp dụng lần lượt, chọn lựa này được gọi là phân mảnh hỗn hợp.



2.6. Cấp phát

2.6.1 Bài toán cấp phát

Giả sử đã có một tập các mảnh $F=\{F_1, F_2, ...,F_n\}$ và một mạng bao gồm các vị trí $S=\{S_1, S_2, ...,S_m\}$ trên đó có một tập các ứng dụng $Q=\{q_1, q_2, ...,q_q\}$ đang chạy.

Bài toán cấp phát là tìm một phân phối "tối ưu" của F cho S.

Tính tối ưu có thể được định nghĩa ứng với hai số đo:

- Chi phí nhỏ nhất: Hàm chi phí có chi lưu mảnh F_i vào vị trí S_j, chi phí vấn tin mảnh F_i vào vị trí S_j, chi phí cập nhật F_i tại tất cả mọi vị trí có chứa nó và chi phí tryền dữ liệu. Vì thế bài toán cấp phát cố gắng tìm một lược đồ cấp phát với hàm chi phí tổ hợp nhỏ nhất.

- Hiệu năng: Chiến lược cấp phát được thiết kế nhằm duy trì một hiệu quả lớn đó là hạ thấp thời gian đáp ứng và tăng tối đa lưu lượng hệ thống tại mỗi vị trí.

Nói chung bài toán cấp phát tổng quát là một bài toán phức tạp và có độ phức tạp là NP-đầy đủ (NP-complete). Vì thế các nghiên cứu đã được dành cho việc tìm ra các thuật giải heuristec tốt để có lời giải gần tối ưu.

2.6.2 Yêu cầu về thông tin

Ở giai đoạn cấp phát, chúng ta cần các thông tin định lượng về CSDL, về các ứng dụng chạy trên đó, về cấu trúc mạng, khả năng xử lý và giới hạn lưu trữ của mỗi vị trí trên mạng.

Thông tin về CSDL

Độ tuyển của một mảnh F_j ứng với câu vấn tin q_i. Đây là số lượng các bộ của F_j cần được truy xuất để xử lý q_i. Giá trị này ký hiệu là $sel_i(F_j)$

Kích thước của một mảnh F_j được cho bởi

$$Size (F_j) = card (F_j)* length(F_j)$$

Trong đó: $Length(F_j)$ là chiều dài (tính theo byte) của một bộ trong mảnh F_j.

Thông tin về ứng dụng

Hai số liệu quan trọng là số truy xuất đọc do câu vấn tin q_i thực hiện trên mảnh F_j trong mỗi lần chạy của nó (ký hiệu là RR_{ij}), và tương ứng là các truy xuất

cập nhật (UR_{ij}). Thí dụ chúng có thể đếm số truy xuất khối cần phải thực hiện theo yêu cầu vấn tin.

Chúng ta định nghĩa hai ma trận UM và RM với các phần tử tương ứng u_{ij} và r_{ij} được đặc tả tương ứng như sau:

$$u_{ij} = \begin{cases} 1 & \text{nếu vấn tin } q_i \text{ có cập nhật mảnh } F_j \\ 0 & \text{trong trường hợp ngược lại} \end{cases}$$

$$r_{ij} = \begin{cases} 1 & \text{nếu vấn tin } q_i \text{ có cập nhật mảnh } F_j \\ & \text{trong trường hợp ngược lại} \end{cases}$$

Một véctơ O gồm các giá trị o(i) cũng được định nghĩa, với o(i) đặc tả vị trí đưa ra câu vấn tin q_i .

Thông tin về vị trí

Với mỗi vị trí (trạm) chúng ta cần biết về khả năng lưu trữ và xử lý của nó. Hiển nhiên là những giá trị này có thể tính được bằng các hàm thích hợp hoặc bằng phương pháp đánh giá đơn giản.

+ Chi phí đơn vị tính để lưu dữ liệu tại vị trí S_k sẽ được ký hiệu là USC_k.

+ Đặc tả số đo chi phí LPC_k, là chi phí xử lý một đơn vị công việc tại vị trí S_k. Đơn vị công việc cần phải giống với đơn vị của RR và UR.

Thông tin về mạng

Chúng ta giả sử tồn tại một mạng đơn giản, g_{ij} biểu thị cho chi phí truyền mỗi bó giữa hai vị trí S_i và S_j. Để có thể tính được số lượng thông báo, chúng ta dùng fsize làm kích thước (tính theo byte) của một bó dữ liệu.

2.6.3. Mô hình cấp phát

Mô hình cấp phát có mục tiêu làm giảm thiểu tổng chi phí xử lý và lưu trữ dữ liệu trong khi vẫn cố gắng đáp ứng được các đòi hỏi về thời gian đáp ứng. Mô hình của chúng ta có hình thái như sau:

Min (Total Cost)

ứng với ràng buộc thời gian đáp ứng, ràng buộc lưu trữ, ràng buộc xử lý.

Biến quyết định x_{ij} được định nghĩa là

$$x_{ij} = \begin{cases} 1 & \text{nếu mảnh } F_i \text{ được lưu tại vị trí } S_j \\ 0 & \text{trong trường hợp ngược lại} \end{cases}$$

Tổng chi phí

Hàm tổng chi phí có hai thành phần: phần xử lý vấn tin và phần lưu trữ. Vì thế nó có thể được biểu diễn là:

$$\mathbf{TOC} = \sum_{qi \in Q} \mathbf{QPC_i} + \sum_{Sk \in S} \sum_{Fj \in F} \mathbf{STC_{jk}}$$

với QPC_i là chi phí xử lý câu vấn tin ứng dụng q_i, và STC_{jk} là chi phí lưu mảnh F_j tại vị trí S_k.

Chúng ta hãy xét chi phí lưu trữ trước. Nó được cho bởi

$$\mathbf{STC_{jk} = USC_k * size(F_j) * x_{jk}}$$

Chi phí xử lý vấn tin khó xác định hơn. Hầu hết các mô hình cho bài toán cấp phát tập tin FAP tách nó thành hai phần: Chi phí xử lý chỉ đọc và chi phí xử lý chỉ cập nhật. Ở đây chúng tỗi đã chọn một hướng tiếp cận khác trong mô hình cho bài toán DAP và xác định nó như là chi phí xử lý vấn tin bao gồm chi phí xử lý là PC và chi phí truyền là TC. Vì thế chi phí xử lý vấn tin QPC cho ứng dụng q_i là

$$\mathbf{QPC_i = PC_i + TC_i}$$

Thành phần xử lý PC gồm có ba hệ số chi phí, chi phí truy xuất AC, chi phí duy trì toàn vẹn IE và chi phí điều khiển đồng thời CC:

$$\mathbf{PC_i = AC_i + IE_i + CC_i}$$

Mô tả chi tiết cho mỗi hệ số chi phí phụ thuộc vào thuật toán được dùng để hoàn tất các tác vụ đó. Tuy nhiên để minh hoạ chúng tôi sẽ mô tả chi tiết về AC:

$$\mathbf{AC_i} = \sum_{S_k \in S} \sum_{F_j \in F} \mathbf{(u_{ij} * UR_{ij} + r_{ij} * RR_{ij}) * x_{jk} * LPC_k}$$

Hai số hạng đầu trong công thức trên tính số truy xuất của vấn tin q_i đến mảnh F_j. Chú ý rằng $(UR_{ij}+RR_{ij})$ là tổng số các truy xuất đọc và cập nhật. Chúng ta giả thiết rằng các chi phí xử lý chúng là như nhau. Ký hiệu tổng cho biết tổng số các truy xuất cho tất cả mọi mảnh được q_i tham chiếu. Nhân với LPC_k cho ra chi phí của truy xuất này tại vị trí S_k. Chúng ta lại dùng x_{jk} để chỉ chọn các giá trị chi phí cho các vị trí có lưu các mảnh.

Một vấn đề rất quan trọng cần đề cập ở đây. Hàm chi phí truy xuất giả sử rằng việc xử lý một câu vấn tin có bao gồm cả việc phân rã nó thành một tập các vấn tin con hoạt tác trên một mảnh được lưu tại vị trí đó, theo sau là truyền kết quả trở lại về vị trí đã đưa ra vấn tin.

Hệ số chi phí duy trì tính toàn vẹn có thể được mô tả rất giống thành phần xử lý ngoại trừ chi phí xử lý cục bộ một đơn vị cần thay đổi nhằm phản ánh chi phí thực sự để duy trì tính toàn vẹn.

Hàm chi phí truyền có thể được đưa ra giống như cách của hàm chi phí truy xuất. Tuy nhiên tổng chi phí truyền dữ liệu cho cập nhật và cho yêu cầu chỉ đọc sẽ khác nhau hoàn toàn. Trong các vấn tin cập nhật, chúng ta cần cho tất cả mọi vị trí biết nơi có các bản sao còn trong vấn tin chỉ đọc thì chỉ cần truy xuất một trong các bản sao là đủ. Ngoài ra vào lúc kết thúc yêu cầu cập nhật thì không cần phải truyền dữ liệu và ngược lại, cho vị trí đưa ra vấn tin ngoài một thông báo xác nhận, còn trong vấn tin chỉ đọc có thể phải có nhiều thông báo tryền dữ liệu.

Thành phần cập nhật của hàm truyền dữ liệu là:

$$TCU_i = \sum_{S_k} \sum_{F_j} u_{ij} * x_{jk} * g_{o(i),k} + \sum_{S_k} \sum_{F_j} u_{ij} * x_{jk} * g_{k,o(i)}$$

Số hạng thứ nhất để gửi thông báo cập nhật từ vị trí gốc o(i) của q_i đến tất cả bản sao cập nhật. Số hạng thứ hai dành cho thông báo xác nhận.

Thành phần chi phí chỉ đọc có thể đặc tả là:

$$TCR_i = \sum_{F_j} \sum_{S_k} \min (u_{ij} * x_{jk} * g_{o(i),k} + r_{ij} * x_{jk} * (sel_i(F_j) * length(F_j)/fsize) * g_{k,o(i)})$$

Số hạng thứ nhất trong TCR biểu thị chi phí truyền yêu cầu chỉ đọc đến những vị trí có bản sao của mảnh cần truy xuất. Số hạng thứ hai để truyền các kết quả từ những vị trí này đến những vị trí yêu cầu. Phương trình này khẳng định rằng trong số các vị trí có bản sao của cùng một mảnh, chỉ vị trí sinh ra tổng chi phí truyền thấp nhất mới được chọn để thực hiện thao tác này.

Bây giờ hàm chi phí tính cho vấn tin q_i có thể được tính là:

$$TC_i = TCU_i + TCR_i$$

Ràng buộc

Ràng buộc thời gian đáp ứng cần được đặc tả là thời gian thực thi của q_i thời gian đáp ứng lớn nhất của q_i q_i Q

Người ta thích đặc tả số đo chi phí của hàm theo thời gian bởi vì nó đơn giản hoá đặc tả về ràng buộc thời gian thực thi.

Ràng buộc lưu trữ là: STC_{jk} **khả năng lưu trữ tại vị trí** S_k, S_k **S**

Fj F

Trong đó ràng buộc xử lý là:

tải trọng xử lý của q_i tại vị trí S_k khả năng xử lý của S_k, S_k S.

qi Q

CHƯƠNG 3. XỬ LÝ VẤN TIN

Ngữ cảnh được chọn ở đây là phép tính quan hệ và đại số quan hệ. Như chúng ta đã thấy các quan hệ phân tán được cài đặt qua các mảnh. Thiết kế CSDL có vai trò hết sức quan trọng đối với việc xử lý vấn tin vì định nghĩa các mảnh có mục đích làm tăng tính cục bộ tham chiếu, và đôi khi tăng khả năng thực hiện song song đối với những câu vấn tin quan trọng nhất. Vai trò của thể xử lý vấn tin phân tán là ánh xạ câu vấn tin cấp cao trên một CSDL phân tán vào một chuỗi các thao tác của đại số quan hệ trên các mảnh. Một số chức năng quan trọng biểu trưng cho ánh xạ này. Trước tiên câu vấn tin phải được phân rã thành một chuỗi các phép toán quan hệ được gọi là vấn tin đại số. Thứ hai, dữ liệu cần truy xuất phải được cục bộ hóa để các thao tác trên các quan hệ được chuyển thành các thao tác trên dữ liệu cục bộ (các mảnh). Cuối cùng câu vấn tin đại số trên các mảnh phải được mở rộng để bao gồm các thao tác truyền thông và được tối ưu hóa để hàm chi phí là thấp nhất. Hàm chi phí muốn nói đến các tính toán như thao tác xuất nhập đĩa, tài nguyên CPU, và mạng truyền thông.

3.1. Bài toán xử lý vấn tin

Có hai phương pháp tối ưu hóa cơ bản được sử dụng trong các bộ xử lý vấn tin: phương pháp biến đổi đại số và chiến lược ước lượng chi phí.

Phương pháp biến đổi đại số đơn giản hóa các câu vấn tin nhờ các phép biến đổi đại số nhằm hạ thấp chi phí trả lời câu vấn tin, độc lập với dữ liệu thực và cấu trúc vật lý của dữ liệu.

Nhiệm vụ chính của thể xử lý vấn tin quan hệ là biến đổi câu vấn tin cấp cao thành một câu vấn tin tương đương ở cấp thấp hơn được diễn đạt bằng đại số quan hệ. Câu vấn tin cấp thấp thực sự sẽ cài đặt chiến lược thực thi vấn tin. Việc biến đổi này phải đạt được cả tính đúng đắn lẫn tính hiệu quả. Một biến đổi được xem là đúng đắn nếu câu vấn tin cấp thấp có cùng ngữ nghĩa với câu vấn tin gốc, nghĩa là cả hai cùng cho ra một kết quả. Một câu vấn tin có thể có nhiều cách biến đổi tương đương thành đại số quan hệ. Bởi vì mỗi chiến lược thực thi tương đương đều sử dụng tài nguyên máy tính rất khác nhau, khó khăn chính là chọn ra được một chiến lược hạ thấp tối đa việc tiêu dùng tài nguyên.

Thí dụ 3.1:

Chúng ta hãy xét một tập con của lược đồ CSDL đã được cho

NV(MNV, TênNV, Chức vụ)

PC (MNV, MDA, Nhiệm vụ, Thời gian)

Và một câu vấn tin đơn giản sau:

"Cho biết tên của các nhân viên hiện đang quản lý một dự án"

Biểu thức vấn tin bằng phép tính quan hệ theo cú pháp của SQL là:

```
SELECT      TênNV
FROM        NV, PC
WHERE       NV.MNV=PC.MNV
            AND  Nhiệmvụ="Quảnlý"
```

Hai biểu thức tương đương trong đại số quan hệ do biến đổi chính xác từ câu vấn tin trên là:

$$\pi_{TênNV}(\sigma_{Nhiệmvụ="Quảnlý" \wedge NV.MNV=PC.MNV}(NV \times PC))$$

và

$$\pi_{TênNV}(NV \mid\!><\!\mid_{MNV}(\sigma_{Nhiệmvụ="Quảnlý"}(PC)))$$

Hiển nhiên là trong câu vấn tin thứ hai, chúng ta tránh sử dụng tích Descartes, vì thế tiêu dùng ít tài nguyên máy tính hơn câu vấn tin thứ nhất và vì vậy nên được giữ lại.

Trong bối cảnh tập trung, chiến lược thực thi vấn tin có thể được diễn tả chính xác bằng một mở rộng của đại số quan hệ. Nhiệm vụ chính của thể xử lý vấn tin tập trung là đối với một câu vấn tin đã cho, nó phải chọn ra được một câu vấn tin đại số tốt nhất trong số những câu vấn tin tương đương. Bởi vì đây là bài toán phức tạp về mặt tính toán khi số lượng các quan hệ khá lớn, nên nói chung nó thường được rút lại ở yêu cầu là chọn được một lời giải gần tối ưu.

Trong các hệ phân tán, đại số quan hệ không đủ để diễn tả các chiến lược thực thi. Nó phải được cung cấp thêm các phép toán trao đổi dữ liệu giữa các vị trí. Bên cạnh việc chọn thứ tự cho các phép toán đại số quan hệ, thể xử lý vấn tin phân tán cũng phải chọn các vị trí tốt nhất để xử lý dữ liệu, và có thể cả cách biến đổi dữ liệu. Kết quả là không gian lời giải các chiến lược thực thi tăng lên, làm cho việc xử lý vấn tin phân tán tăng lên rất nhiều.

Thí dụ 3.2:

Thí dụ này minh họa tầm quan trọng của việc chọn lựa vị trí và cách truyền dữ liệu của một câu vấn tin đại số. Chúng ta xét câu vấn tin của thí dụ trên:

$$\text{TênNV}(\text{NV} |><|_{\text{MNV}} (\ \text{Nhiệmvụ="Quảnlý"}(\text{PC})))$$

chúng ta giả sử rằng các quan hệ NV và PC được phân mảnh ngang như sau:

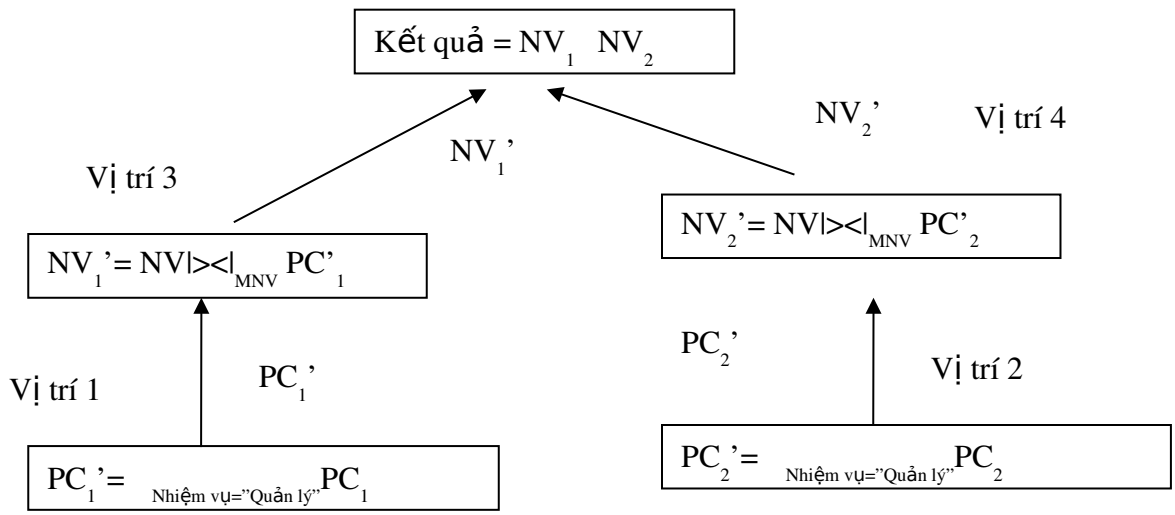
$NV_1 = {}_{MNV \quad \text{"E3"}}(NV)$

$NV_2 = {}_{MNV > \text{"E3"}}(NV)$

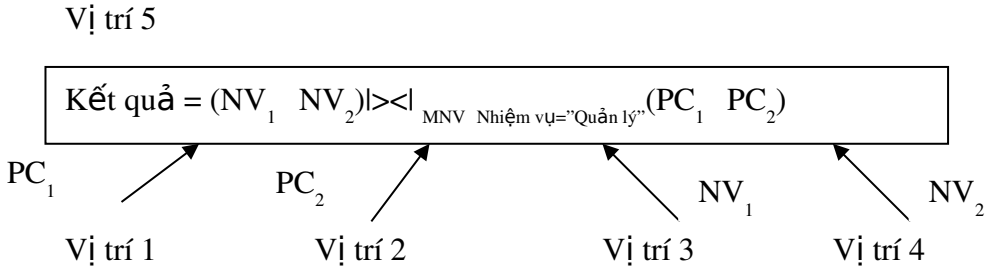
$PC_1 = {}_{MNV \quad \text{"E3"}}(PC)$

$PC_2 = {}_{MNV \quad \text{"E3"}}(PC)$

Các mảnh PC_1, PC_2, NV_1, NV_2 theo thứ tự được lưu tại các vị trí 1, 2, 3 và 4 và kết quả được lưu tại vị trí 5



Hình 4.1a) Chiến lược a



Hình 4.1b) Chiến lược b

Mũi tên từ vị trí i đến vị trí j có nhãn R chỉ ra rằng quan hệ R được chuyển từ vị trí i đến vị trí j. Chiến lược A sử dụng sự kiện là các quan hệ EMP và ASG được phân mảnh theo cùng một cách để thực hiện song song các phép toán chọn và nối. chiến lược B tập trung tất cả các dữ liệu tại vị trí lưu kết quả trước khi xử lý câu vấn tin.

Để đánh giá việc tiêu dùng tài nguyên của hai chiến lược này, chúng ta sử dụng một mô hình chi phí đơn giản sau. Chúng ta giả sử rằng thao tác truy xuất một bộ (tuple access) được ký hiệu là tupacc, là một đơn vị và thao tác truyền một bộ (tuple transfer) tuptrans là 10 đơn vị. Đồng thời chúng ta cũng giả sử là các quan hệ NV và PC tương ứng có 400 và 1000 bộ, và có 20 giám đốc dự án thống nhất cho các vị trí. Cuối cùng chúng ta giả sử rằng các quan hệ PC và NV được gom tụ cục bộ tương ứng theo các thuộc tính Nhiệmvụ và MNV. Vì vậy có thể truy xuất trực tiếp đến các bộ của PC dựa trên giá trị của thuộc tính Nhiệmvụ (tương ứng là MNV cho NV)

* Tổng chi phí của chiến lược A có thể được tính như sau:

1.Tạo ra PC' bằng cách chọn trên PC cần (10+10)* tupacc	= 20
2. Truyền PC' đến vị trí của NV cần (10+10)*tuptrans	= 200
3. Tạo NV' bằng cách nối PC' và NV' cần (10+10)*tupacc*2	= 40
4. Truyền NV' đến vị trí nhận kết quả cần (10+10)*tuptrans	= 200
Tổng chi phí	460

* Tổng chi phí cho chiến lược B có thể được tính như sau:

1. Truyền NV đến vị trí 5 cần 400*tuptrans	= 4.000
2. truyền PC đến vị trí 5 cần 1000*tuptrans	=10.000
3. Tạo ra PC' bằng cách chọn trên PC cần 1000*tupacc	= 1.000
4. Nối NVvà PC cần 400*20*tupacc	= 8.000
Tổng chi phí là	23.000

Trong chiến lược B, chúng ta đã giả sử rằng các phương pháp truy xuất các quan hệ NV và PC dựa trên các thuộc tính Nhiệmvụ và MNV bị mất tác dụng do việc truyền dữ liệu. Đây là một giả thiết hợp lý trong thực tế. Chiến lược A tốt hơn với hệ số 50 "rất có ý nghĩa". Hơn nữa nó đưa ra một cách phân phối công việc giữa các vị trí. Khác biệt còn cao hơn nữa nếu chúng ta giả thiết là tốc độ truyền chậm hơn và/ hoặc mức độ phân mảnh cao hơn.

Chỉ số đánh giá tiêu dùng tài nguyên là tổng chi phí (total cost) phải trả khi xử lý vấn tin. Tổng chi phí là tổng thời gian cần để xử lý các phép toán vấn tin tại các vị trí khác nhau và truyền dữ liệu giữa các vị trí. Một công cụ khác là thời gian đáp ứng của câu vấn tin, là thời gian cần thiết để chạy câu vấn tin. Vì các phép toán có thể được thực hiện song song tại các vị trí khác nhau, thời gian đáp ứng có thể nhỏ hơn nhiều so với tổng chi phí của nó. Trong môi trường CSDL phân tán, tổng chi phí cần phải giảm thiểu là chi phí CPU, chi phí xuất nhập và chi phí truyền. Chi phí CPU là chi phí phải trả khi thực hiện các thao tác trên dữ liệu trong bộ nhớ chính. Chi phí xuất nhập (I/O) là thời gian cần thiết cho các thao tác xuất nhập đĩa. Chi phí truyền tin là thời gian cần để trao đổi dữ liệu giữa các vị rí tham gia vào trong quá trình thực thi câu vấn tin. Chi phí này phải trả khi phải xử lý các thông báo (định dạng/ giải định dạng) và khi truyền dữ liệu trên mạng. Chi phí truyền có lẽ là yếu tố quan trọng nhất được xét đến trong CSDL phân tán.

Độ phức tạp của các phép toán quan hệ: các phép toán chọn, Chiếu (Không loại bỏ trùng lặp) có độ phức tạp l à O(n); Các phép chiếu(C ó loại bỏ trùng lặp), trùng lặp, nối, nối nửa, chia có độ phức tạp là $O(n*\log_n)$; Tích Descartes có độ phức tạp là $O(n^2)$ (N biểu thị lực lượng của quan hệ nếu các bộ thu được độc lập với nhau)

Đánh giá:

+ Các thao tác có tính chọn lựa làm giảm đi lực lượng cần phải thực hiện trước tiên.

+ Các phép toán cần phải được sắp xếp để tránh thực hiện tích Descartes hoặc để lại thực hiện sau.

3.2. Phân rã vấn tin

Phân rã vấn tin là giai đoạn đầu tiên của quá trình xử lý câu vấn tin. Nó biến đổi câu vấn tin ở dạng phép tính quan hệ thành câu vấn tin đại số quan hệ. Các vấn tin nhập xuất đều tham chiếu các quan hệ toàn cục và không dùng đến các thông tin phân bố dữ liệu. Vì thế phân rã vấn tin đều giống nhau trong cả hệ thống tập trung lẫn phân tán, câu vấn tin xuất sẽ đúng về ngữ nghĩa và đạt chất lượng theo nghĩa là đã loại bỏ các hành động không cần thiết. Phân rã vấn tin có thể xem như bốn bước liên tiếp nhau: Chuẩn hoá, phân tích, loại bỏ dư thừa, viết lại câu vấn tin

Chuẩn hoá

Mục đích của chuẩn hoá (normalization) là biến đổi câu vấn tin thành một dạng chuẩn để xử lý tiếp. Chuẩn hoá một vấn tin nói chung gồm có đặt các lượng từ và lượng từ hoá vấn tin bằng cách áp dụng độ ưu tiên của các toán tử logic.

Với các ngôn ngữ quan hệ như SQL, biến đổi quan trọng nhất là lượng từ hoá vấn tin (mênh đề Where), có thể đó là một vị từ phi lượng từ với độ phức tạp nào đó với tất cả các lượng từ cần thiết (hoặc) được đặt phía trước. Có hai dạng chuẩn có thể cho vị từ, một có thứ bậc cao cho AND( ) và loại còn lại cho thứ bậc cao OR ( ). Dạng chuẩn hội là hội (vị từ) của các tuyển vị từ (các vị từ):

$(p_{11}\ \ p_{12}\ \ldots.\ p_{1n})\ \ldots..\ (p_{m1}\ \ p_{m2}\ \ldots.\ p_{mn})$

trong đó p_{ij} là một vị từ đơn giản. Ngược lại, một lượng từ hoá ở dạng chuẩn tuyển như sau:

$(p_{11}\ \ p_{12}\ \ldots.\ p_{1n})\ \ldots.\ (p_{m1}\ \ p_{m2}\ \ldots.\ p_{mn})$

Biến đổi các vị từ phi lượng từ là tầm thường bằng cách sử các quy tắc tương đương cho các phép toán logic (, ,): 9

1. $p_1\ \ p_2 \quad p_2\ \ p_1$
2. $p_1\ \ p_2 \quad p_2\ \ p_1$
3. $p_1\ (\ p_2\ \ p_3) \quad (p_1\ \ p_2\ )\ \ p_3$
4. $p_1\ (\ p_2\ \ p_3) \quad (p_1\ \ p_2\ )\ \ p_3$
5. $p_1\ (\ p_2\ \ p_3) \quad (p_1\ \ p_2\ )\ (p_1\ \ p_3\ )$
6. $p_1\ (\ p_2\ \ p_3) \quad (p_1\ \ p_2\ )\ \ (p_1\ \ p_3\ )$
7. $(p_1\ \ p_2\ ) \quad p_1 \quad p_2$
8. $(p_1 \quad p_2\ ) \quad p_1 \quad p_2$
9. $(\ p) \quad p$

Trong dạng chuẩn tắc tuyển, câu vấn tin có thể được xử lý như các câu vấn tin con hội độc lập, được nối bằng phép hợp (tương ứng với các tuyển mệnh đề).

Nhận xét: Dạng chuẩn tuyển ít được dùng vì dẫn đến các vị từ nối và chọn trùng nhau. Dạng chuẩn hội hay dùng trong thực tế

Thí dụ 3.3:

Tìm tên các nhân viên đang làm việc ở dự án P_1 trong 12 tháng hoặc 24 tháng.

Câu vấn tin được diễn tả bằng SQL như sau:

SELECT TênNV

FROM NV, PC

WHERE NV.MNV=PC.MNV

AND PC.MDA= "P_1"

AND Thời gian=12 OR Thời gian=24

Lượng từ hoá ở dạng chuẩn hội là:

NV.MNV=PC.MNV PC.MDA= "P_1" (Thời gian=12 Thời gian=24)

Còn lượng từ hoá ở dạng chuẩn tuyển là

(NV.MNV=PC.MNV PC.MDA="P_1" Thời gian=12)

(NV.MNV=PC.MNV PC.MDA="P_1" Thời gian=24)

ở dạng sau, xử lý hai hội độc lập có thể là một công việc thừa nếu các biểu thức con chung không được loại bỏ.

Phân tích

Phân tích câu vấn tin cho phép phế bỏ các câu vấn tin đã chuẩn hoá nhưng không thể tiếp tục xử lý được hoặc không cần thiết, những lý do chính là do chúng sai kiểu hoặc sai ngữ nghĩa.

- Một câu vấn tin gọi là sai kiểu nếu nó có một thuộc tính hoặc tên quan hệ chưa được khai báo trong lược đồ toàn cục, hoặc nếu nó áp dụng cho các thuộc tính có kiểu không thích hợp.

Select MaDA

From TenNV >200

- Một câu vấn tin gọi là sai nghĩa nếu các thành phần của nó không tham gia vào việc tạo ra kết quả.

Nếu các các vấn tin không chứa các tuyển và phủ định ta có thể dùng đồ thị vấn tin. Vấn tin chứa phép chọn nối chiếu.

- Biểu diễn bằng đồ thị vấn tin:

+ 1 nút biểu thị quan hệ kết quả

+ Các nút khác biểu thị cho quan hệ toán hạng

+ Một cạnh giữa hai nút không phải là quan hệ kquả biểu diễn cho một nối

+ Cạnh mà nút đích là kết quả sẽ biểu thị cho phép chiếu.

+ Các nút không phải là kết quả sẽ được gán nhãn là một vị từ chọn hoặc 1 vị từ nối (chính nó).

- Đồ thị nối: một đồ thị con quan trọng của đồ thị vấn tin, nó chỉ có các nối.

***Thí dụ 3.4*:**

PC (MãNV, MaDA, NVụ, Tgian)

NV (MaNV, TênNV, CVụ)

DA (MaDA, TênDA, Kphí, Đđiểm)

" Tìm tên, Nvụ các của những người có Cvụ='TP' đã làm việc ở dự án 'CAD/CAM' trong hơn 3 năm"

Select TênNV

From PC, NV,DA

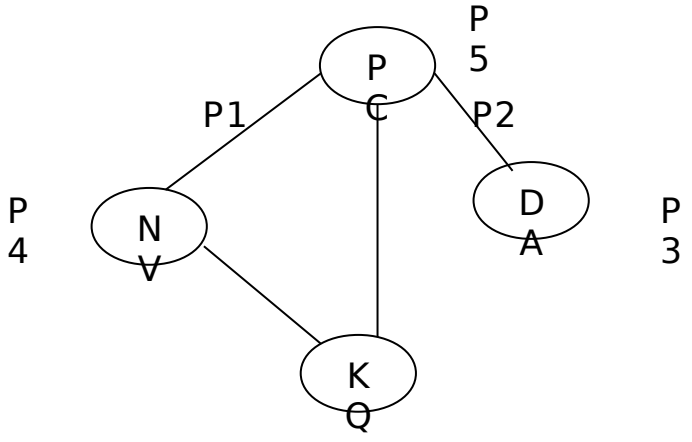
Where PC.MaNV = NV.MaNV and PC.MaDA=DA.MaDA and TênDA='CAD/CAM' and CVụ ='TP' and tgian >36

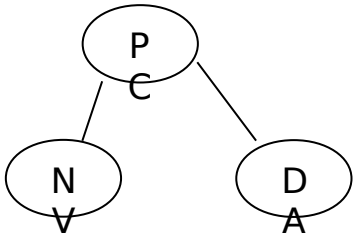
Các vị từ đơn giản:

p1: PC.MaNV = NV.MaNV p2: PC.MaDA=DA.MaDA

p3: TênDA='CAD/CAM' p4: CVụ ='TP' p5: tgian >36

Đồ thị vấn tin *Đồ thị nối*





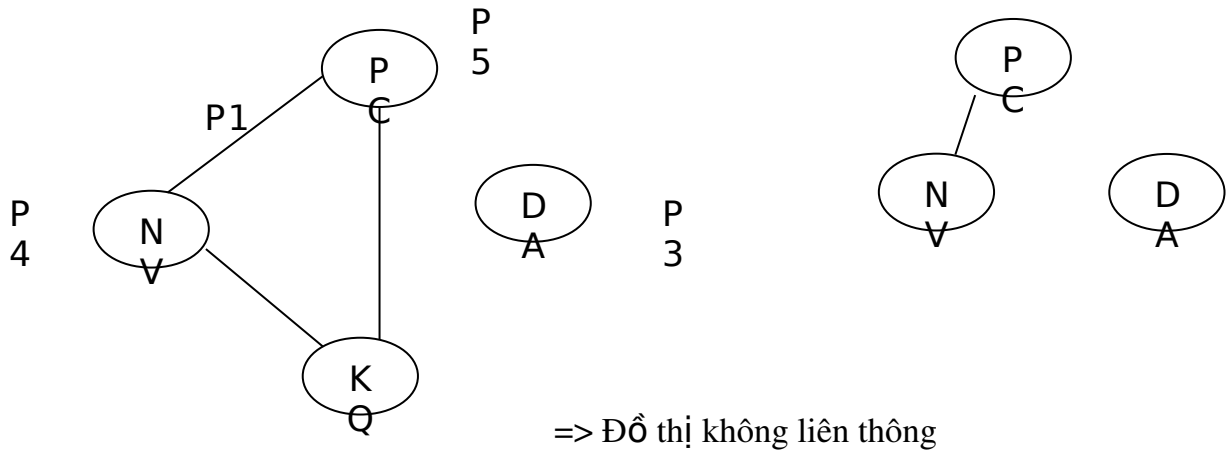
Thí dụ 3.5:

Select TênNV

From PC, NV,DA

Where PC.MaNV = NV.MaNV

and TênDA='CAD/CAM' and CVụ ='TP' and tgian >36



=> Đồ thị không liên thông

Nhận xét: Câu vấn tin sai ngữ nghĩa nếu đồ thị vấn tin của nó không liên thông: 1 hoặc nhiều đồ thị con bị tách rời với đồ thị kết quả.

Loại bỏ dư thừa

Một câu vấn tin của người sử dụng thường được diễn tả trên một khung nhìn có thể được bổ sung thêm nhiều vị từ để có được sự tương ứng khung nhìn - quan hệ, bảo đảm được tính toàn vẹn ngữ nghĩa và bảo mật. Thế nhưng lượng từ hoá vấn tin đã được sửa đổi này có thể chứa các vị từ dư thừa, có thể phải khiến lặp lại một số công việc. Một cách làm đơn giản vấn tin là loại bỏ các vị từ thừa

- Loại bỏ vị từ dư thừa bằng qui tắc luỹ đẳng: 10

1. p p p	6. p v True True
2. p v p p	7. p p False
3. p True p	8. p v p True
4. p v False p	9. p1 (p1v p2) p1
5. p False False	10. p1 v (p1 p2) p1

Thí dụ 3.6 :

Select CVụ

From NV

Where (Not (CVụ ='TP') and (CVụ='TP' or CVụ='PP') and not (CVụ='PP')) or TênNV='Mai'

p1: CVụ =’TP’

p2: CVụ=’PP’

p3: TênNV=’Mai’

Lượng từ hoá:

(p1 (p1 v p2) p2) v p3

áp dụng : (p1 ((p1 p2) v (p2 p2))) v p3

áp dụng 3: (p1 p1 p2) v (p1 p2 p2) v p3

áp dụng 7: (False p2) v (p1 False) v p3

áp dụng 5: False v False v p3 = p3

Viết lại: Select CVụ

From NV

where TênNV=’Mai’

Viết lại câu vấn tin

Bước này được chia thành hai bước nhỏ:

(1) Biến đổi câu vấn tin từ phép tính quan hệ thành đại số quan hệ

(2) Cấu trúc lại câu vấn tin đại số nhằm cải thiện hiệu năng.

Để cho dễ hiểu, chúng ta sẽ trình bày câu vấn tin đại số quan hệ một cách hình ảnh bằng cây toán tử. Một cây toán tử là một cây với mỗi nút lá biểu thị cho một quan hệ được lưu trong CSDL và các nút không phải là nút lá biểu thị cho một quan hệ trung gian được sinh ra bởi các phép toán quan hệ. Chuỗi các phép toán đi theo hướng từ lá đến gốc biểu thị cho kết quả vấn tin.

Biến đổi câu vấn tin phép tính quan hệ bộ thành một cây toán tử có thể thu được dễ dàng bằng cách sau. Trong SQL, các nút lá có sẵn trong mệnh đề FROM. thứ hai nút gốc được tạo ra như một phép chiếu chứa các thuộc tính kết quả. Các thuộc tính này nằm trong mệnh đề SELECT của câu vấn tin SQL. Thứ ba, lượng từ hoá (mệnh đề Where của SQL) được dịch thành chuỗi các phép toán quan hệ thích hợp (phép chọn, nối, hợp, ..) đi từ các nút lá đến nút gốc. Chuỗi này có thể được cho trực tiếp qua thứ tự xuất hiện của các vị từ và toán tử.

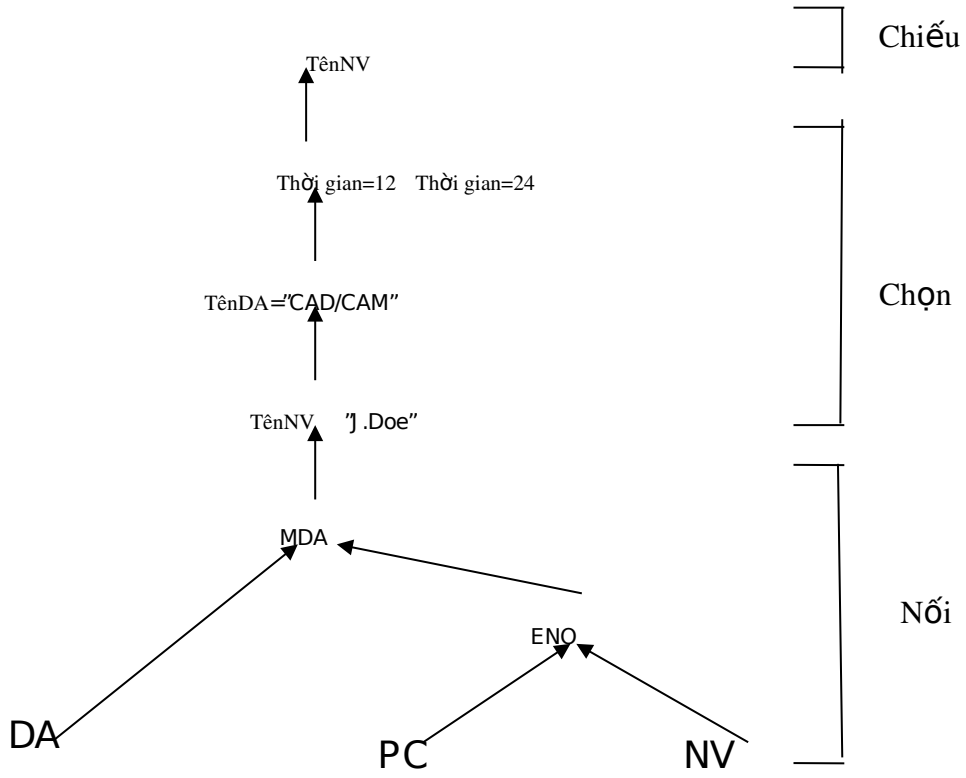
Thí dụ 3.7:

Câu vấn tin: “tìm tên các nhân viên trừ J.Doe đã làm cho dự án CAD/CAM trong một hoặc hai năm”.

Biểu thức SQL là:

```
SELECT      TênNV
FROM        DA, PC, NV
WHERE       PC.MNV=NV.MNV
AND         PC.MDA=DA.MDA
AND         TênNV     “J.Doe”
AND         DA.TênDA=”CAD/CAM”
AND         (Thời gian=12 OR Thời gian=24)
```

Các thể được ánh xạ thành cây trong hình dưới.



Bằng cách áp dụng các quy tắc biến đổi, nhiều cây có thể được thấy rằng tương đương với cây được tạo ra bằng phương pháp được mô tả ở trên. Sáu quy tắc tương đương hữu ích nhất và được xem là các phép toán đại số quan hệ cơ bản :

R, S, T là những quan hệ, trong đó R được định nghĩa trên các thuộc tính $A=\{A_1, A_2,\ldots,A_n\}$ và quan hệ S được định nghĩa trên các thuộc tính $B=\{B_1, B_2, \ldots,B_n\}$.

1. Tính giao hoán của phép toán hai ngôi

R x S S x R

$R \bowtie S$ $S \bowtie R$

Quy tắc này cũng áp dụng được cho hợp nhưng không áp dụng cho hiệu tập hợp hay nối nửa.

2. Tính kết hợp của các phép toán hai ngôi

(R x S)x T R x (Sx T)

$(R \bowtie S) \bowtie T$ $R \bowtie (S \bowtie T)$

3- Tính lũy đẳng của các phép toán đơn ngôi

Nếu R được định nghĩa trên tập thuộc tính A và A' A, A" A và A' A" thì $_{A'}$ $_{A'}(\ _{A"}(R))$ $_{A'}(R)$

$_{p1(A1)}(\ _{p2(A2)}(R))$ $_{p1(A1)\ p2(A2)}(R)$

trong đó p_i là một vị từ được áp dụng cho thuộc tính A_i

4. Giao hoán phép chọn với phép chiếu

$_{A1\ldots An}(\ _{p(Ap)}(R))$ $_{A1\ldots An}(\ _{p(Ap)}(\ _{A1\ldots An,Ap}(R)))$

Chú ý rằng nếu A_p là phần tử của $\{A_1, A_2,\ldots,A_n\}$ thì phép chiếu cuối cùng trên $\{A_1, A_2,\ldots,A_n\}$ ở vế phải của hệ thức không có tác dụng.

5. Giao hoán phép chọn với phép toán hai ngôi

$_{p(Ai)}(R \times S)$ $(\ _{p(Ai)}(R)) \times S$

$_{p(Ai)}(R \bowtie_{p(Aj, Bk)} S)$ $(\ _{p(Ai)}(R)) \bowtie_{p(Aj, Bk)} S$

$_{p(Ai)}(R \cup T)$ $_{p(Ai)}(R) \cup\ _{p(Ai)}(T)$

6-Giao hoán phép chiếu với phép toán hai ngôi

Nếu C=A' B', trong đó A' A, B' B, và A, B là các tập thuộc tính tương ứng của quan hệ R và S, chúng ta có

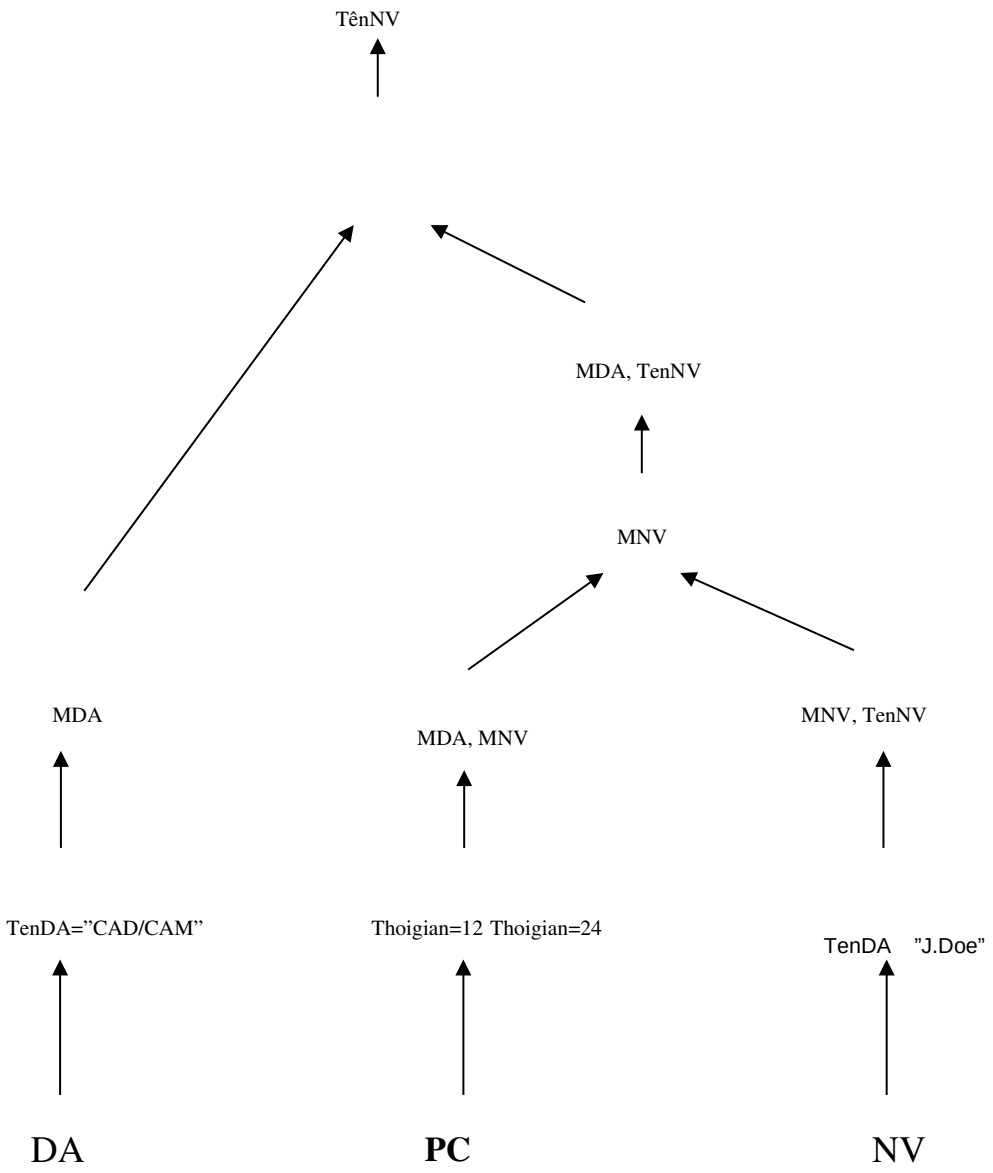
$_{C}(R \times S)$ $_{A}(R) \times\ _{B}(S)$

$_{C}(R \bowtie_{p(Ai, Bj)} S)$ $_{A}(R) \bowtie_{p(Ai, Bj)}\ _{B}(S)$

$_{C}(R \cup S)$ $_{A}(R) \cup\ _{B}(S)$

Các quy tắc trên có thể được sử dụng để cấu trúc lại cây một cách có hệ thống nhằm loại bỏ các cây "xấu". Một thuật toán tái cấu trúc đơn giản sử dụng heuristic trong đó có áp dụng các phép toán đơn ngôi (chọn/ chiếu) càng sớm càng tốt nhằm giảm bớt kích thước của quan hệ trung gian.

Tái cấu trúc cây trong hình trên sinh ra cây trong hình sau. Kết quả được xem là đạt chất lượng theo nghĩa là nó tránh truy xuất nhiều lần đến cùng một quan hệ và các phép toán chọn lựa nhiều nhất được thực hiện trước tiên.



3.3. Cục bộ hóa dữ liệu phân tán

Tầng cục bộ hóa dữ liệu chịu trách nhiệm dịch câu vấn tin đại số trên quan hệ toàn cục sang câu vấn tin đại số trên các mảnh vật lý. Cục bộ hóa có sử dụng các thông tin được lưu trong một lược đồ phân mảnh.

Tầng này xác định xem những mảnh nào cần cho câu vấn tin và biến đổi câu vấn tin phân tán thành câu vấn tin trên các mảnh. Tạo ra câu vấn tin theo mảnh được thực hiện qua hai bước. Trước tiên vấn tin phân tán được ánh xạ thành vấn tin theo mảnh bằng cách thay đổi mỗi quan hệ phân tán bằng chương trình tái thiết của nó. Thứ hai vấn tin theo mảnh được đơn giản hoá và tái cấu trúc để tạo ra một câu vấn tin "có chất lượng". Quá trình đơn giản hoá và tái cấu trúc có thể được

thực hiện theo những quy tắc được sử dụng trong tầng phân rã. Giống như trong tầng đó, câu vấn tin theo mảnh cuối cùng nói chung chưa đạt đến tối ưu bởi vì thông tin liên quan đến các mảnh chưa được sử dụng.

- Cục bộ hoá dữ liệu sẽ xác định các mảnh nào cần cho câu vấn tin. Biến đổi câu vấn tin phân tán thành các câu vấn tin theo mảnh.
- Trong phần này đối với mỗi kiểu phân mảnh ta sẽ trình bày các kỹ thuật rút gọn để tạo các câu vấn tin tối ưu và đơn giản hơn.
- Ta sẽ sử dụng các qui tắn biến đổi và các khám phá, chẳng hạn đẩy các phép toán đơn ngôi xuống thấp như có thể.

Rút gọn phân mảnh ngang nguyên thuỷ

- Việc phân mảnh ngang phân tán 1 quan hệ dựa trên các vị từ chọn

Thí dụ 3.8:

NV (MaNV, TenNV, CVụ)

NV1 = $_{MaNV \quad 'E3'}$(NV)

NV2 = $'_{'E3' < MaNV \quad 'E6'}$(NV)

NV3 = $_{MaNV > 'E6'}$(NV)

} NV = NV1 NV2 NV3

Cách làm:

+ Xác định sau khi tái cấu trúc lại cây con, xem cây nào tạo ra các quan hệ rỗng thì loại bỏ chúng đi.

+ Phân mảnh ngang có thể được dùng để đơn giản hoá phép chọn và phép nối

***Rút gọn với phép chọn*:** Chọn trên các mảnh có lượng từ mâu thuẫn với lượng từ hoá của qui tắc phân mảnh sẽ sinh ra quan hệ rỗng ta loại bỏ chúng.

r_j = nếu t thuộc r : ($t(p_i)$ $t(p_j)$)

Trong đó p_i, p_j là các vị từ chọn, t biểu thị cho 1 bộ, t(p) biểu thị "vị từ p đúng với t".

Ví dụ: NV1 = $_{MaNV \quad 'E3'}$(NV)

NV2 = $'_{'E3' < MaNV \quad 'E6'}$(NV)

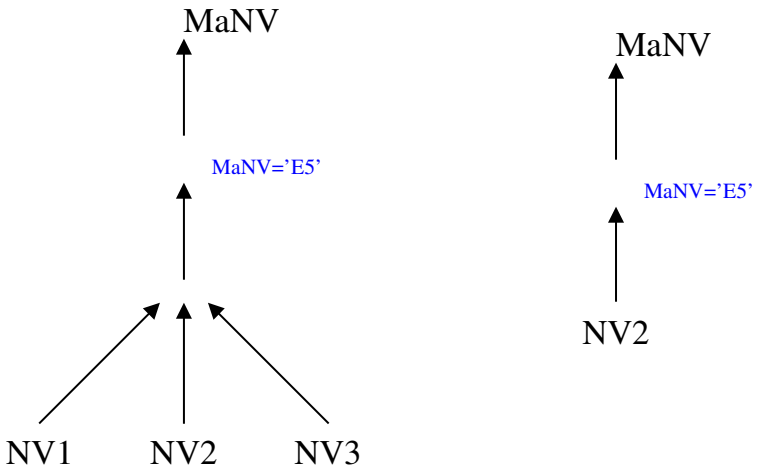
NV3 = $_{MaNV > 'E6'}$(NV)

Select MaNV

} Bằng cách hoán vị phép chọn với phép hợp ta sẽ phát hiện ra vị từ chọn >< vị từ của NV1, NV3. => tạo ra các quan hệ rỗng => loại bỏ

From NV

Where MaNV=’E5’



Rút gọn với phép nối

- Nối trên các quan hệ phân mảnh ngang có thể được đơn giản khi các quan hệ nối được phân mảnh theo thuộc tính nối.

- Đơn giản hoá gồm có phân phối các nối trên các hợp rồi bỏ đi các nối vô dụng.

$$(r1 \quad r2) \ |><| \ s = (r1 \ |><| \ s) \quad (r2 \ |><| \ s)$$

đó r_i là các mảnh còn r, s là các quan hệ

Bằng phép biến đổi này, các hợp có thể được di chuyển lên trên cây toán tử để tất cả các nối có thể có các mảnh đều được lộ ra. Các nối vô dụng được bỏ đi khi các lượng từ hoá của các mảnh có mâu thuẫn.

Thí dụ 3.9:

Cho 2 quan hệ được phân mảnh

NV(MaNV, TênNV, CVụ)

$$NV1 = {}_{MaNV \quad E3}(NV)$$
$$NV2 = {}_{E3 < MaNV \quad E6}(NV)$$
$$NV3 = {}_{MaNV > E6}(NV)$$

PC (MaNV, MaDA, NVụ, Tg)

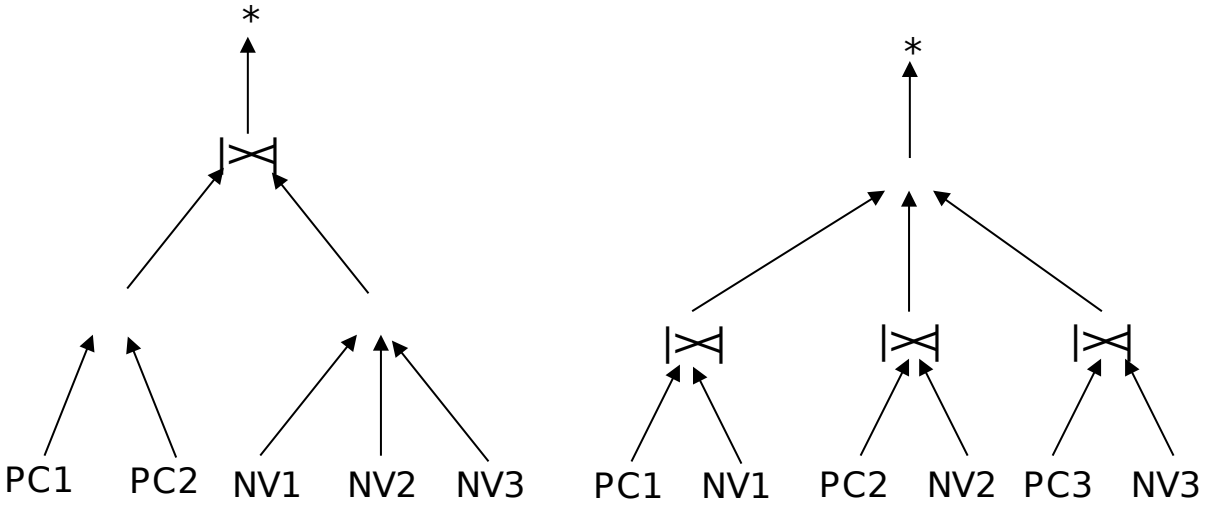
$$PC1 = {}_{MaNV \quad E3}(PC)$$
$$PC2 = {}_{MaNV > E3}(NV)$$

Câu hỏi:

Select *

From NV, PC

Where NV.MaNV = PC.MaNV



Rút gọn cho phân mảnh dọc

Phân mảnh dọc phân tán một quan hệ dựa trên các thuộc tính chiếu. Chương trình cục bộ hoá cho một quan hệ phân mảnh dọc gồm có nối của các mảnh theo thuộc tính chung.

Thí dụ 3.10:

NV(MaNV, TênNV, CVụ)

NV1 = $_{MaNV, TênNV}$(NV)

NV2 = $_{MaNV, CVụ}$(NV)

Chương trình cục bộ hoá: NV = NV1 |><| NV2

Cách làm: Vấn tin trên phân mảnh dọc có thể được rút gọn bằng cách xác định các quan hệ trung gian vô dụng và loại bỏ các cây con đã sinh ra chúng

A= {A1, A2, ...} và được phân mảnh dọc thành r_i (A') trong đó A' A

r_i (D, A') là vô dụng nếu tập thuộc tính chiếu D không thuộc A'

Thí dụ 3.11:

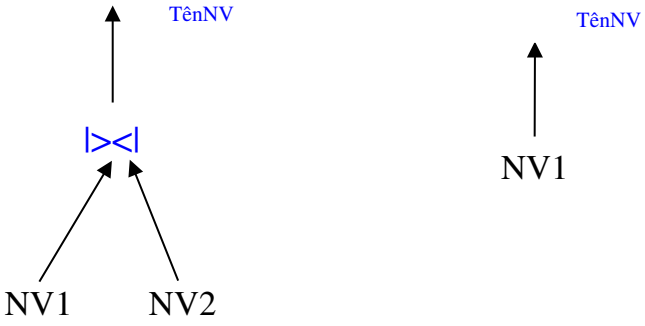
NV(MaNV, TênNV, CVụ)

NV1 = $_{MaNV, TênNV}$(NV)

NV2 = $_{MaNV, CVụ}$(NV)

Select TênNV

From NV



Rút gọn cho phân mảnh ngang dẫn xuất

- Phân mảnh ngang dẫn xuất là một cách để phân phối hai quan hệ mà nhờ đó có thể cải thiện khả năng xử lý các điểm giao nhau giữa phép chọn và phép nối.

- Nếu quan hệ r phải phân mảnh dẫn xuất theo quan hệ s, các mảnh của r và s giống nhau ở thuộc tính nối sẽ nằm cùng vị trí. Ngoài ra s có thể được phân mảnh theo vị từ chọn.

- Phân mảnh dẫn xuất chỉ được sử dụng cho mối liên hệ 1 – N (phân cấp) từ s đến r: trong đó 1 bộ của s có thể khớp với nhiều bộ của r

Thí dụ 3.12:

NV(MaNV, TênNV, CVụ)

$NV1 = \sigma_{Cvụ='TP'}(NV)$

$NV2 = \sigma_{Cvụ\ 'TP'}(NV)$

PC(MaNV, MaDA, NVụ, Tg)

PC1 = PC |>< NV1

PC2 = PC |>< NV2

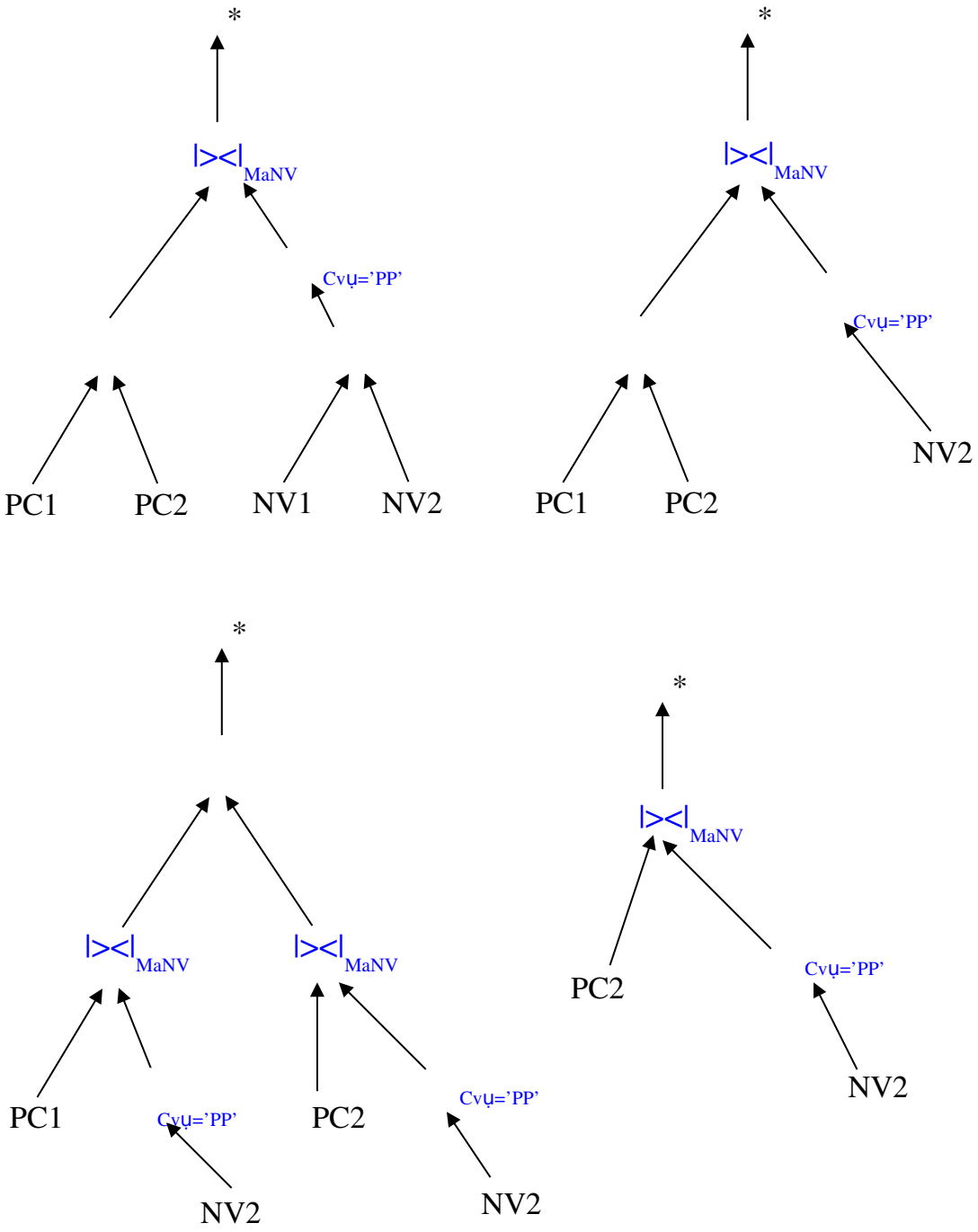
" Đưa ra tất cả các thuộc tính của NV, PC với NVụ ="PP"

Select *

From NV, PC

Where NV.MaNV = PC.MaNV and NV.CVụ="PP"

Câu vấn tin trên các mảnh NV1, NV2, PC1, PC2 đã được định nghĩa. Đẩy phép chọn xuống các mảnh NV1. NV2 câu vấn tin rút gọn lại do mâu thuẫn với vị từ chọn của NV1 = > loại bỏ NV1


*
|><|MaNV
Cvụ='PP'
PC1
PC2
NV1
NV2
*
|><|MaNV
Cvụ='PP'
PC1
PC2
NV2
*
|><|MaNV
|><|MaNV
PC1
Cvụ='PP'
NV2
PC2
Cvụ='PP'
NV2
*
|><|MaNV
PC2
Cvụ='PP'
NV2

Rút gọn cho phân mảnh ngang hỗn hợp

Mục tiêu: Hỗ trợ hiệu quả các câu vấn tin có chứa phép chiếu, chọn và nối

Câu vấn tin trên các mảnh hỗn hợp có thể được rút gọn bằng cách tổ hợp các qui tắc tương ứng đã đuợc dùng trong các phân mảnh ngang nguyên thuỷ, phân mảnh dọc, phân mảnh ngang dẫn xuất.

Qui tắc:

1/ Loại bỏ các quan hệ rỗng được tạo ra bởi các phép toán chọn mâu thuẫn trên các mảnh ngang.

2/ Loại bỏ các quan hệ vô dụng được tạo ra bởi các phép chiếu trên các mảnh dọc

3/ Phân phối các nối cho các hợp nằm cô lập và loại bỏ các nối vô dụng.

Thí dụ 3.13:

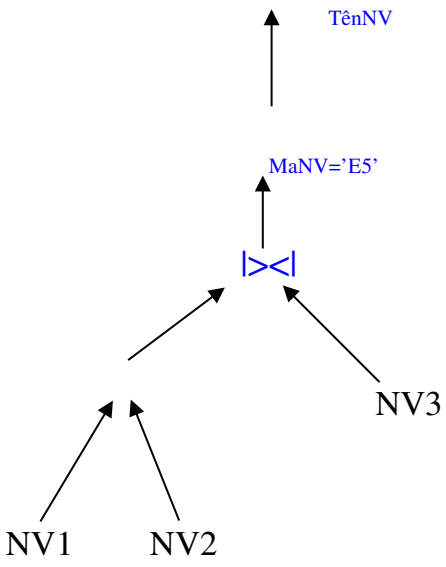
MaNV, TênNV, CVụ) NV2 = { NV1 = $_{MaNV\ E4}$ ($_{MaNV, TênNV}$ (NV))

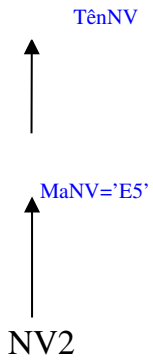
$_{MaNV > E4}$ ($_{MaNV, TênNV}$ (NV))

NV3 = $_{MaNV, CVụ}$ (NV)

Chương trình cục bộ hoá NV = (NV1 NV2) |><| NV3

Câu vấn tin: Tên của nhân viên có mã E5





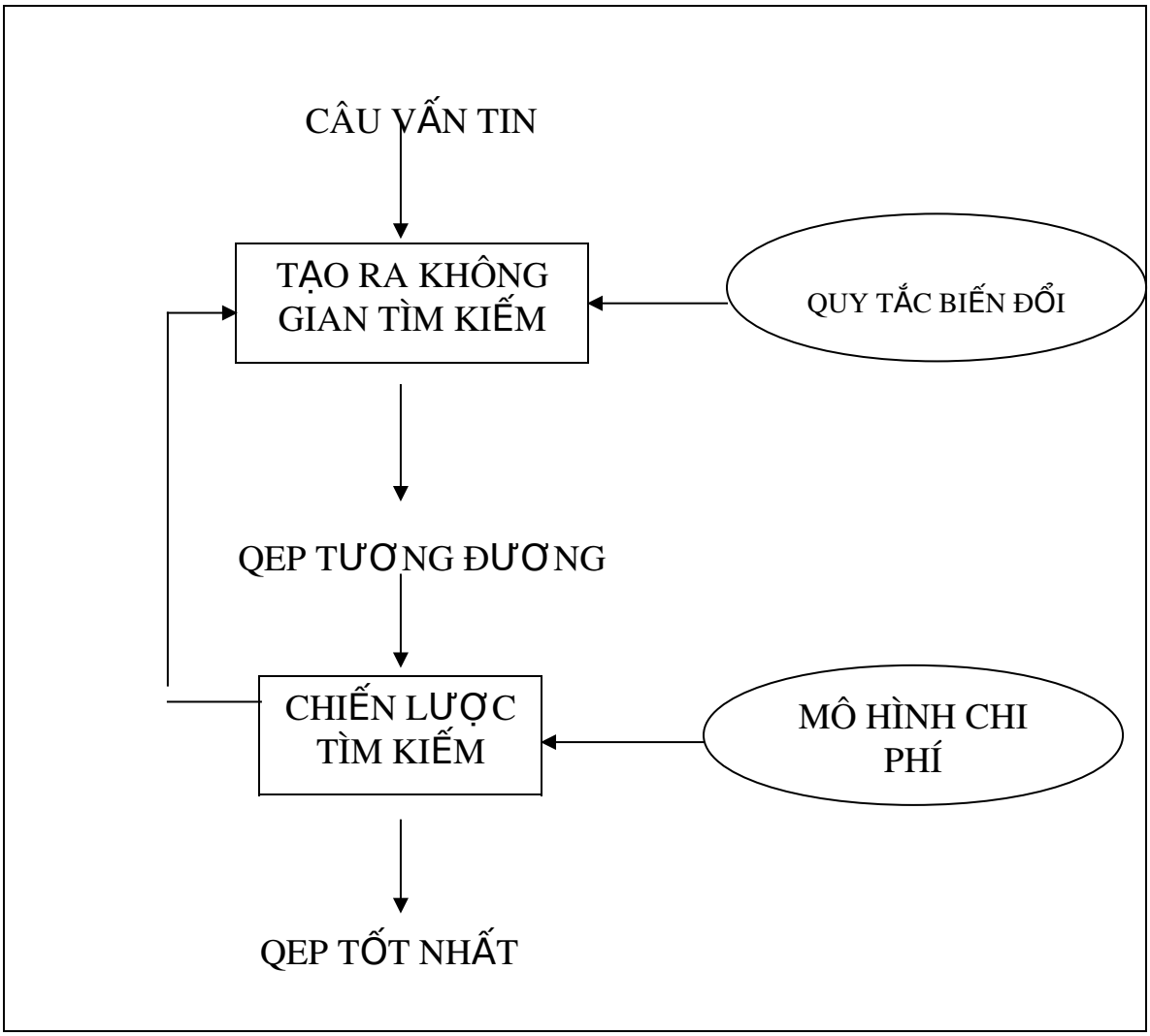
3.4. Tối ưu hoá vấn tin phân tán

Trong phần này chúng ta sẽ giới thiệu về quá trình tối ưu hóa nói chung, bất kể môi trường là phân tán hay tập chung. Vấn tin cần tối ưu giả thiết là được diễn tả bằng đại số quan hệ trên các quan hệ CSDL (có thể là các mảnh) sau khi đã viết lại vấn tin từ biểu thức phép tình quan hệ.

Tối ưu hóa vấn tin muốn nói đến quá trình sinh ra một *hoạch định thực thi vấn tin* (query execution plan, QEP) biểu thị cho chiến lược thực thi vấn tin. Hoạch định được chọn phải hạ thấp tối đa hàm chi phí. Thể tối ưu hóa vấn tin, là một đơn thể phần mềm chịu trách nhiệm thực hiện tối ưu hóa, thường được xem là cấu tạo bởi ba thành phần: một *không gian tìm kiếm* (search space), một *mô hình chi phí* (cost model) và một *chiến lược tìm kiếm* (search strstegy) (xem hình 1.4.4). Không gian tìm kiếm là tập các hoạch định thực thi biểu diễn cho câu vấn tin. Những hoạch định này là tương đương, theo nghĩa là chúng sinh ra cùng một kết quả nhưng khác nhau ở thứ tự thực hiện các thao tác và cách thức cài đặt những thao tác này, vì thế khác nhau về hiệu năng. Không gian tìm kiếm thu được bằng cách áp dụng các quy tắc biến đổi, chẳng hạn những qui tắc cho đại số quan hệ đã mô tả trong phần viết lại câu vấn tin. Mô hình chi phí tiên đoán chi phí của một hoạch định thực thi đã cho. Để cho chính xác, mô hình chi phí phải có đủ thông tin cần thiết về môi trường thực thi phân tán. Chiến lược tìm kiếm sẽ khám phá không gian tìm kiếm và chọn ra hoạch định tốt nhất dựa theo mô hình chi phí. Nó định nghĩa xem các hoạch định nào cần được kiểm tra và theo thứ tự nào. Chi tiết về môi trường (tập trung hay phân tán) được ghi nhận trong không gian và mô hình chi phí.

3.4.1. Không gian tìm kiếm

Các hoạch định thực thi vấn tin thường được trìu tượng hóa qua cây toán tử), trên đó định nghĩa thứ tự thực hiện các phép toán. Chúng ta bổ sung thêm các thông tin như thuật toán tốt nhất được chọn cho mỗi phép toán. Đối với một câu vấn tin đã cho, không gian tìm kiếm có thể được định nghĩa như một tập các cây toán tử tương đương, có được bằng cách áp dụng các qui tắc biến đổi . Để nêu bật các đặc trưng của thể tối ưu hóa vấn tin , chúng ta thường tập trung các *cây nối* (join tree), là cây toán tử với các phép toán nối hoặc tích Descartes. Lý do là các hoán vị thứ tự nối các tác dụng quan trọng nhất đến hiệu năng của các vấn tin quan hệ.

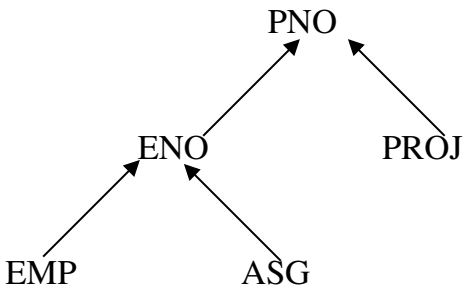


Thí dụ 3.14:

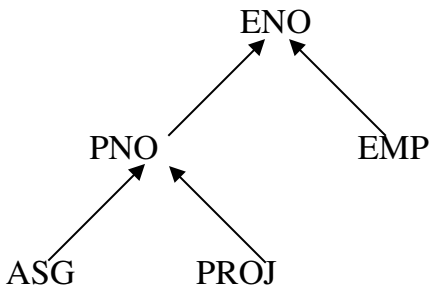
Xét câu vấn tin sau:

```
SELECT      ENAME
FROM        EMP, ASG, PROJ
WHERE       EMP, ENO=ASG.ENO
AND         ASG, PNO=PROJ . PNO
```

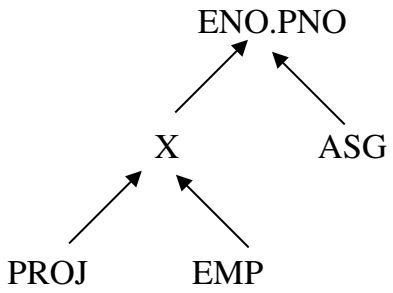
Hình sau minh họa ba cây nối tương đương cho vấn tin đó, thu được bằng cách sử dụng tính chất kết hợp của các toán tử hai ngôi. Mỗi cây này có thể được gán một chi phí dựa trên chi phí của mỗi toán tử. Cây nối (c) bắt đầu với một tích Des-cartes có thể có chi phí cao hơn rất nhiều so với cây còn lại.



(a)

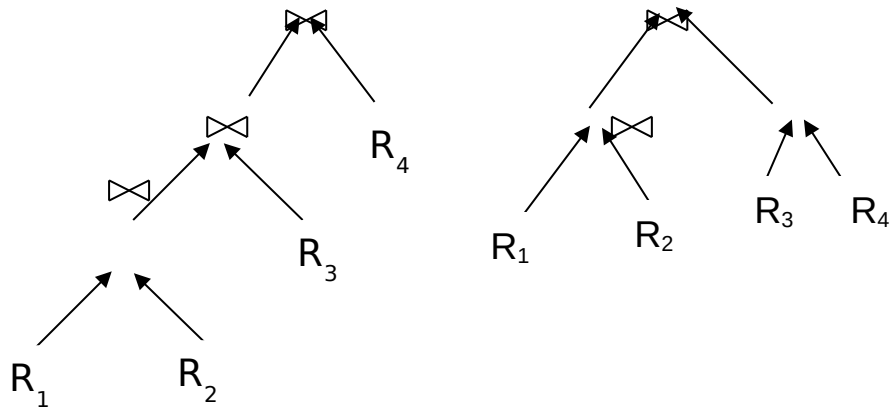


(b)



(c)

Với một câu vấn tin phức tạp (có gồm nhiều quan hệ và nhiều toán tử), số caaytoans tử tương đương có thể rất nhiều. Thí dụ số cây nối có thể thu được từ việc áp dụng tính giao hoán và kết hợp là O(N!) cho N quan hệ. Việc đánh giá một không gian tìm kiếm lớn có thể mất quá nhiều thời gian tối ưu hóa, đôi khi còn tốn hơn cả thời gian thực thi thực sự. Vì thể, thể tối ưu hóa thường hạn chế kích thước cần xem xét của không gian tìm kiếm . Hạn chế thứ nhất là dùng các heuristic. Một heuristic thông dụng nhất là thực hiện phép chọn và chiếu khi truy xuất đến quan hệ cơ sở. Một heuristic thông dụng khác là tránh lấy các tích Descartes không được chính câu vấn tin yêu cầu. Thí dụ trong hình trên cây toán tử (c) không phải là phần được thể tối ưu hóa xem xét trong không gian tìm kiếm.

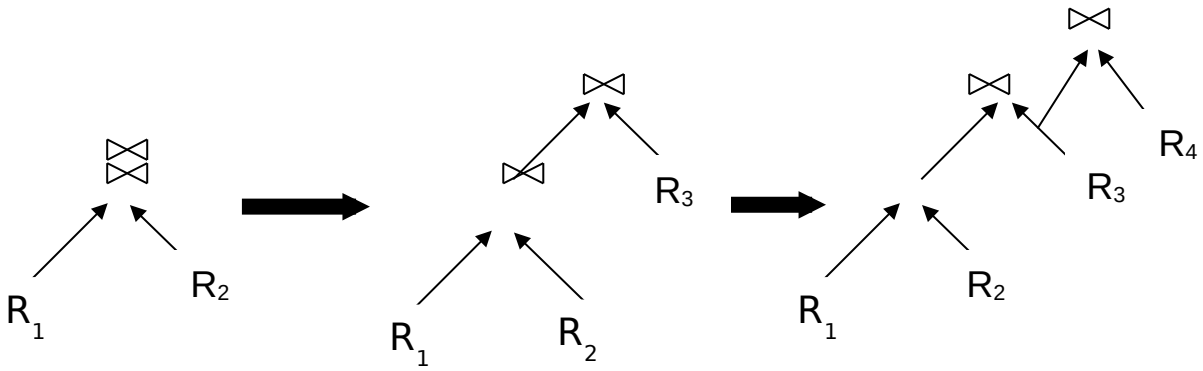


a) Cây nối tuyến tính b) Cây nối xum xuê

Một hạn chế quan trọng khác ứng với hình dạng của cây nối. Hai loại cây nối thường được phân biệt Cây nối tuyến tính và cây nối xum xuê (xem Hình 9.3). Một cây tuyến tính (linear tree) là cây với mỗi nút toán tử có ít nhất một toán hạng là một quan hệ cơ sở. Một cây xum xuê (bushy tree) thì tổng quát hơn và có thể có các toán tử không có quan hệ cơ sở làm toán hạng (nghĩa là cả hai toán hạng đều là các quan hệ trung gian). Nếu chỉ xét các cây tuyến tính, kích thước của không gian tìm kiếm được rút gọn lại thành $O(2^N)$. Tuy nhiên trong môi trường phân tán, cây xum xuê rất có lợi cho việc thực hiện song song.

3.4.2. Chiến lược tìm kiếm

Chiến lược tìm kiếm hay được các thể tối ưu hóa vấn tin sử dụng nhất là *quy hoạch động* (dynamic programming) cới tính chất *đơn định* (deterministic). Các chiến lược đơn định tiến hành bằng cách xây dựng các hoạch định , bắt đầu từ các quan hệ cơ sở, nối thêm nhiều quan hệ tại mỗi bước cho đến khi thu được tất cả mọi hoạch định khả hữu như trong Hình 9.4.. Quy hoạch động xây dựng tất cả mọi hoạch định khả hữu *theo hướng ngang* (breadth-first) trước khi nó chọn ra hoạch định "tốt nhất". Để hạ thấp chi phí tối ưu hóa, các hoạch định từng phần rất có khả năng không dẫn đến một hoạch định tối ưu đều được xén bỏ ngay khi có thể. Ngược lại, một chiến lược đơn định khác là thuật toán thiển cận chỉ xây dựng một hoạch định *theo hướng sâu* (depth-first).

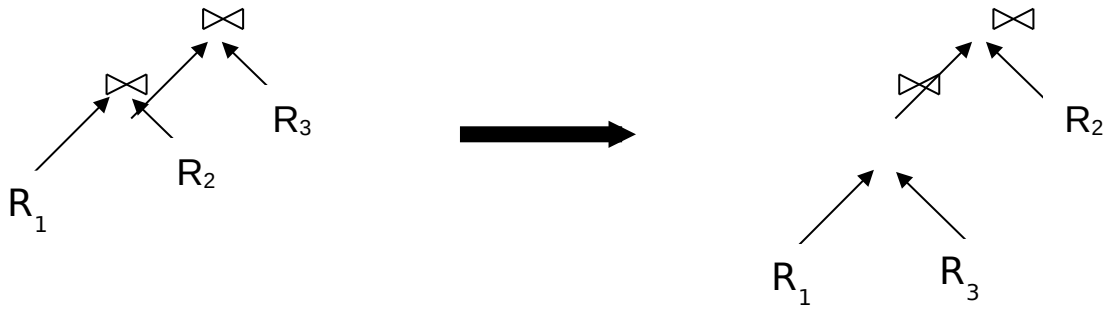


Bước 1 Bước 2 Bước 3

Quy hoạch động hầu như có bản chất vét cạn và bảo đảm tìm ra được các hoạch định. Nó phải trả một chi phí có thể chấp nhận được (theo thời gian và không gian) khi số quan hệ trong câu vấn tin khá nhỏ. Tuy nhiên lối tiếp cận này có

chi phí quá cao khi số quan hệ lớn hơn 5 hoặc 6. Vì lý do này mà các chú ý gần đây đang tập trung vào các *chiến lược ngẫu nhiên hóa* (randomized strategy) để làm giảm độ phức tạp của tối ưu hóa nhưng không bảo đảm tìm được hoạch định tốt nhất. Không giống như các chiến lược đơn định, các chiến lược ngẫu nhiên hóa cho phép thể tối ưu hóa đánh đổi thời gian tối ưu hóa và thời gian thực thi.

Chiến lược ngẫu nhiên hóa chẳng hạn như tập trung vào việc tìm kiếm lời giải tối ưu xung quanh một số điểm đặc biệt nào đó. Chung không đảm bảo sẽ thu được một lời giải tốt nhất nhưng tránh được chi phí quá cao của tối ưu hóa tính theo việc tiêu dùng bộ nhớ và thời gian. Trước tiên một hoặc nhiều hoạch định khởi đầu được xây dựng bằng một chiến lược thiển cận . Sau đó thuật toán tìm cách cải thiện hoạch định này bằng cách thăm các *lân cận* (neighbor) của nó. Một lân cận thu được bằng cách áp dụng một *biến đổi ngẫu nhiên* cho một hoạch định. Thí dụ về một biến đổi điển hình gồm có hoán đổi hai quan hệ toán hạng được chọn ngẫu nhiên của hoạch định như trong đã chứng tỏ bằng thực nghiệm rằng các chiến lược ngẫu nhiên hóa có hiệu năng tốt hơn các chiến lược đơn định khi vấn tin có chứa khá nhiều quan hệ.



3.4.3. Mô hình chi phí phân tán

Mô hình chi phí của thể tối ưu hóa gồm có các hàm chi phí để dự đoán chi phí của các toán tử, số liệu thống kê, dữ liệu cơ sở và các công thức để ước lượng kích thước các kết quả trung gian.

Hàm chi phí

Chi phí của một chiến lược thực thi phân tán có thể được diễn tả ứng với tổng thời gian hoặc với thời gian đáp ứng. *Tổng thời gian* (total time) là tổng tất cả các thành phần thời gian (còn được gọi là chi phí), còn *thời gian đáp ứng* (response time) là thời gian tính từ khi khởi hoạt đến lúc hoàn thành câu vấn tin. Công thức tổng quát để xác định tổng chi phí được mô tả như sau:

$$Total\_time = T_{CPU} * \#insts + T_{I/O} * \#I/Os + T_{MSG} * \#msgs + T_{TR} * \#bytcs$$

Hai thành phần đầu tiên là thời gian xử lý cục bộ, trong đó T_{CPU} là thời gian của một chỉ thị CPU và $T_{I/O}$ là thời gian cho một thao tác xuất nhập đĩa. Thời gian truyền được biểu thị qua hai thành phần cuối cùng. T_{MSG} là thời gian cố định cần để khởi hoạt và nhận một thông báo, còn T_{TR} là thời gian cần để truyền một đơn vị dữ liệu từ vị trí này đến vị trí khác. Đơn vị dữ liệu ở đây tính theo byte (*#byte* là tổng kích thước của tất cả các thông báo), nhưng cũng có thể tính theo những đơn vị khác (thí dụ theo gói). Thông thường chúng ta giả thiết T_{TR} là một giá trị không đổi. Điều này có thể không đúng trong các mạng WAN, trong đó một số vị trí nằm xa hơn so với một số khác. Tuy nhiên giả thiết này làm đơn giản quá trình tối ưu hóa rất nhiều. Vì thế thời gian truyền *#byte* dữ liệu từ vị trí này đến vị trí khác được giả thuyết là một hàm tuyến tính theo #bytes:

$$CT(\#bytes) = T_{MSG} + T_{TR} * \#bytes$$

Các chi phí nói chung được diễn tả theo đơn vị thời gian, và từ đó có thể chuyển thành các đơn vị khác (thí dụ như đô la).

Giá trị tương đối của các hệ số chi phí đặc trưng cho môi trường CSDL phân tán. Topo mạng có ảnh hưởng rất lớn đến tỷ số giữa các thành phần này. Trong mạng WAN như Internet, thời gian truyền thường là hệ số chiếm đa phần. Tuy nhiên trong các mạng LAN thì các hệ số thành phần cân bằng hơn. Những nghiên cứu ban đầu đã chỉ ra rằng tỷ số giữa thời gian truyền và thời gian xuất nhập một trang vào khoảng 20:1 đối với mạng WAN, đối với các mạng Ethernet điển hình (10Mbds) thì vào khoảng 1:1,6. Vì thế phần lớn các hệ DBMS phân tán được thiết kế trên các mạng WAN đều bỏ qua chi phí xử lý cục bộ và tập trung vào vấn đề cực tiểu hóa chi phí truyền. Ngược lại các DBMS phân tán được thiết kế cho mạng LAN đều xét đến cả ba thành phần chi phí này. Các mạng nhanh hơn cả mạng WAN lẫn mạng LAN đã cải thiện các tỷ lệ nêu trên thiên về chi phí truyền khi tất cả mọi thứ khác đều như nhau. Tuy nhiên thời gian truyền vẫn là một yếu tố chiến đa phần trong các mạng WAN như Internet bởi vì dữ liệu cần phải được di chuyển đi đến các vị trí xa hơn.

Khi thời gian đáp ứng vấn tin là hàm mục tiêu của thể tối ưu hóa, chúng ta cần phải xét đến vấn đề xử lý cục bộ song song và truyền song song. Công thức tổng quát của thời gian đáp ứng là:

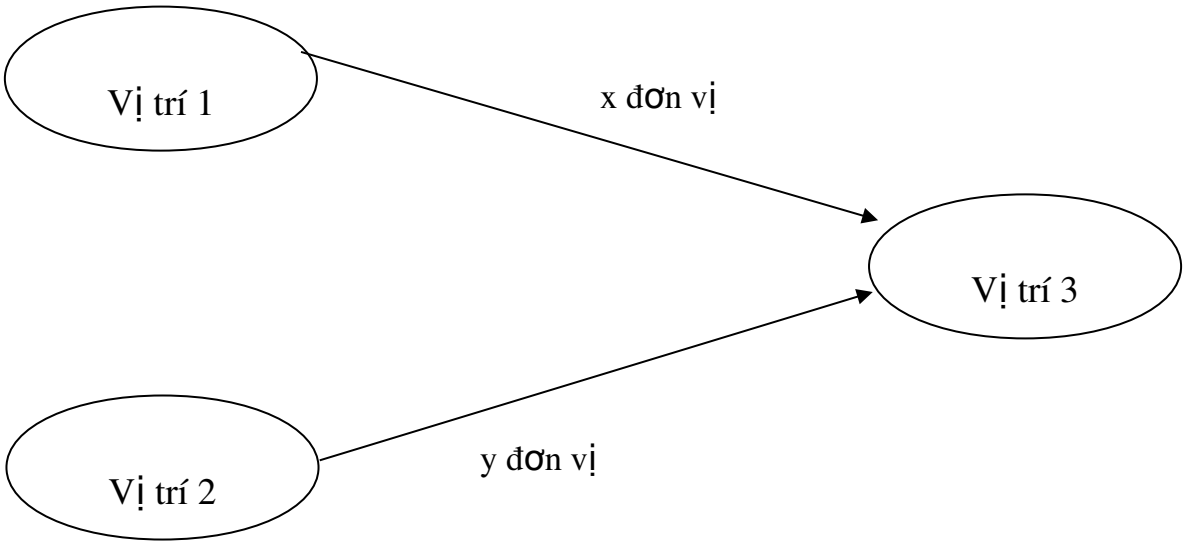
$$Response\_time = T_{CPU} * seq\_\#insts + T_{I/O} * seg\_\#I/Os$$

$$+ T_{MSG} * seg\_\#msgs + T_{TR} * seg\_\#bytes$$

Trong đó *seq_ #x*, với x có thể là các chỉ thị *(insts),* các xuất nhập I/O, các thông báo *(msgs)* hoặc *bytes,* là số lượng x tối đa phải được thực hiện một cách tuần tự khi thực hiện vấn tin. Vì vậy mọi xử lý và truyền dữ liệu thực hiện song song đều được bỏ qua.

Thí dụ 3.15:

Chúng ta minh họa sự khác biệt giữa tổng chi phí và thời gian đáp ứng qua thí dụ trong Hình 6, trong đó kết quả trả lời được tính tại vị trí 3, dữ liệu được lấy từ vị trí 1 và 2. Để đơn giản, chúng ta phải giả sử rằng chỉ xét đến chi phí truyền.



Giả sử rằng T_{MSG} và T_{TR} được diễn tả theo đơn vị thời gian. Tổng chi phí truyền x đơn vị dữ liệu từ vị trí 1 đến vị trí 3 và y đơn vị dữ liệu từ vị trí 2 đến vị trí 3 là

$$Total\_time = 2\ T_{MSG} + T_{TR} * (x + y)$$

Thời gian đáp ứng cho câu vấn tin này có thể tính xấp xỉ là:

$$Response\_time = max\ \{T_{MSG} + T_{TR} * x;\ T_{MSG} + T_{TR} * y)$$

bởi vì các thao tác truyền dữ liệu được thực hiện song song.

Hạ thấp tối đa thời gian đáp ứng được thực hiện bằng cách làm tăng mức độ thực thi song song. Tuy nhiên điều này không có nghĩa là tổng thời gian cũng được hạ thấp. Ngược lại nó có thể làm tăng tổng thời gian, thí dụ như do tăng xử lý song song cục bộ và truyền song song. Hạ thấp tổng thời gian cho thấy là đã cải thiện được việc sử dụng tài nguyên, vì thế làm tăng lưu lượng của hệ thống. Trong thực hành cần cân đối cả hai thời gian này.

Số liệu thống kê CSDL

Tác nhân chính ảnh hưởng đến hiệu quả hoạt động của một chiến lược thực thi là kích thước các quan hệ trung gian đã được tạo ra. Khi một phép toán tiếp theo nằm tại một vị trí khác, quan hệ trung gian phải được di chuyển đến đó. Đó chính là điều khiến chúng ta phải ước lượng kích thước của các kết quả trung gian của các phép toán đại số quan hệ nhằm giảm thiểu lượng dữ liệu phải truyền. Việc ước lượng này dựa trên các thông tin thống kê về các quan hệ cơ sở và các công thức để dự đoán lực lượng của các kết quả. Dĩ nhiên là có những được mất giữa tính chính xác của các số liệu thống kê và chi phí quản lý chúng: số liệu càng chính xác, chi phí càng cao. Đối với một quan hệ R được định nghĩa trên tập thuộc tính $A = \{A_1,$

A_2, ……., A_n} và được phân mảnh là R_1, R_2, ……., R_n dự liệu thống kê điển hình như sau:

1. Đối với mỗi thuộc tính A_i chiều dài (theo số byte) được ký hiệu là *length* (A_i), và đối với mỗi thuộc tinh A_i của mỗi mảnh R_j, số lượng phân biệt các giá trị của A_i, là lực lượng khi chiếu mảnh R_j trên A_i , được ký hiệu là *card* ($\pi A_i(R_j)$).

2. Ứng với miền của mỗi thuộc tính A_i trên một tập giá trị sắp thứ tự được (thí dụ số nguyên hoặc số thực), giá trị lớn nhất và nhỏ nhất được ký hiệu là *max* (A_i) và *min*(A_i).

3. Ứng với miền của mỗi thuộc tính A_i lực lượng của miền được ký hiệu là *card*(*dom*[A_i]). Giá trị này cho biết số lượng các giá trị duy nhất trong *dom*[A_i].

4. Số lượng các bộ trong mỗi mảnh R_j được ký hiệu là *card*(R_j)

Đôi khi dữ liệu thống kê cũng bao gồm *hệ số chọn nối* (join selectivity factor) đối với một số cặp quan hệ, nghĩa là tỷ lệ các bộ có tham gia vào nối. Hệ số được chọn nối được ký hiệu là SF_j của quan hệ R và S là một giá trị thực giữa 0 và 1.:

$$SF_j = \frac{card(R \bowtie S)}{card(R) * card(S)}$$

Chẳng hạn hệ số chọn nối 0.5 tương ứng với một quan hệ nối cực lớn, trong khi đó hệ số 0.001 tương ứng với một quan hệ khá nhỏ. Chúng ta nói rằng nối có *độ chọn kém* trong trường hợp đầu và *độ chọn tốt* trong trường hợp sau:

Dữ liệu thống kê này rất có ích cho việc dự đoán kích thước quan hệ trung gian.

$$Size(R) = card(R) * length(R)$$

Trong đó *length* (*R*) là chiều dài (theo byte) của một bộ của *R*, được tính từ các chiều dài của các thuộc tính của *R*. Việc ước lượng *card* (*R*), số lượng các bộ trong *R*, đòi hỏi phải sử dụng các công thức được cho trong phần tiếp theo.

Lực lượng của các kết quả trung gian

Dữ liệu thống kê rất có ích khi đánh giá lực lượng của các kết quả trung gian. Hai giả thiết đơn giản thường được đưa ra về CSDL. Phân phối của các giá trị thuộc tính trong một quan hệ được giả định là thống nhất, và tất cả mọi thuộc tính đều được độc lập, theo nghĩa là giá trị của một thuộc tính không ảnh hưởng đến giá trị của các thuộc tính khác. Hai giả thiết này thường không đúng trong thực tế, tuy

nhiên chúng làm cho bài toán dễ giải quyết hơn. Trong những đoạn sau, chúng ta trình bày các công thức ước lượng, lực lượng các kết quả của các phép toán đại số cơ bản (phép chọn, phép chiếu, tích Descartes, nối, nối nửa, hợp, và hiệu). Quan hệ toán hạng được ký hiệu là *R* và *S*. *Hệ số chọn* của một phép toán đó được biểu thị là SF_{OP}, *với OP* biểu thị cho phép toán.

Phép chọn. Lực lượng của phép chọn là:

$$Card(\sigma_F(R)) = SF_S(F) * card(R)$$

Trong đó $SF_S(F)$ phụ thuộc vào công thức chọn và có thể được tính như sau, với $p(A_i)$ và $p(A_j)$ biểu thị cho vị trí từ thuộc tính A_i và A_j.

$$SF_S(A = value) = \frac{1}{card(\pi_A(R))}$$

$$SF_S(A > value) = \frac{max(A) - value}{max(A) - min(A)}$$

$$SF_S(A < value) = \frac{value - min(A)}{max(A) - min(A)}$$

$$SF_S(p(A_i) \wedge p(A_j)) = SF_S(p(A_i) * SF_S(p(A_j))$$

$$SF_S(p(A_i) \vee p(A_j)) = SF_S(p(A_i) * SF_S(p(A_j)) - (SF_S(p(A_i)) * SF_S(p(A_j)))$$

$$SF_S(A_i \quad \{value\}) = SF_S(A = value) * card(\{values\})$$

Phép chiếu. Chiếu có thể loại bỏ hoặc không loại bỏ các bộ giống nhau. Ở đây chúng ta xem như chiếu có kèm theo cả việc loại bỏ này. Một phép chiếu bất kỳ rất khó ước lượng chính xác bởi vì mối tương quan giữa các thuộc tính được chiếu thường không được biết. Tuy nhiên có hai trường hợp đặc biệt có ích nhưng việc ước lượng hoàn toàn tầm thường. Nếu chiếu của quan hệ *R* dựa trên thuộc tính *A* duy nhất, lực lượng chỉ là số bộ thu được khi thực hiện phép chiếu. Nếu một trong các thuộc tính chiếu là khóa của *R* thì

$card\ (\pi_A(R)) = card(R)$

Tích Descartes. Lực lượng của tích Descartes của quan hệ *R* và *S* là

$card\ (R \times S) = card(R) * card(S)$

Nối. Không có một phương pháp tổng quát nào để tính lực lượng của nối mà không cần thêm thông tin bổ sung. Cận trên của lực lượng cho nối là lực lượng của tích Descartes. Một số hệ thống, chẳng hạn như hệ INGRES phân tán sử dụng cận trên này, một ước lượng hơi quá đáng. R* sử dụng thương số của trên này với một hằng số, phản ánh sự kiện là kết quả nối luôn nhỏ hơn tích Descartes. Tuy nhiên có một trường hợp xảy ra khá thường xuyên nhưng việc ước lượng lại khá đơn giản. Nếu *R* được thực hiện *nối bằng* với S trên thuộc tính A của R và thuộc tính B của S, trong đó A là khóa của quan hệ R và B là khóa ngoại của quan hệ S thì lực lượng của kết quả đó có thể tính xấp xỉ là:

$Card\ (R \bowtie_{A=B} S) = card(S)$

Bởi vì mỗi bộ của S khớp với tối đa một bộ của R. Hiển nhiên là điều này cũng đúng nếu B là khóa của S và A là khóa ngoại của R. Tuy nhiên ước lượng này là cận trên bởi vì nó giả sử rằng mỗi bộ của S đều tham gia vào trong nối. Đối với những nối quan trọng khác, chúng ta cần duy trì hệ số chọn nối SF_J như thành phần của các thông tin thống kê. Trong trường hợp dó lực lượng của kết quả là:

$Card\ (R \bowtie S) = SF_J * card(R) * card(S)$

Nối nửa. Hệ số chọn của nối nửa giữa R và S cho bởi ty lệ phần trăm các bộ của R có nối với các bộ của S. Một xấp xỉ cho hệ số chọn nối nửa được đưa ra trọng là:

$$SF_{SJ}\ (R \mid>< S) = \frac{Card\ (\pi_A(S))}{Card(dom[A])}$$

Công thức này chỉ phụ thuộc vào thuộc tính A và S. VÌ thế nó thường được gọi là hệ số chọn của thuộc tính A và S, ký hiệu là $SF_{SJ}(S.A)$, và là hệ số chọn của S.A trên bất kỳ một thuộc tính nào có thể nối được với nó. Vì thế lực lượng của nối nửa được cho bởi:

$$Card\ (R \mid>< S) = SF_{SJ}(S.A) * card(R)$$

Xấp xỉ này có thể được xác nhận trên một trường hợp rất thường gặp, đó là khi R.A là khóa ngoại của S (S.A là khóa chính). Trong trường hợp này, hệ số chọn nối nửa là 1 bởi vì $card\ (\pi_A(S)) = Card(dom[A])$ cho thấy rằng lực lượng của nối nửa là $card\ (R)$.

Phép hợp. Rất khó ước lực lượng trong trường hợp của R và S bởi vì các bộ giống nhau bị loại bỏ trong hợp. Chúng ta chỉ trình bày công thức đơn giản cho các cận trên và dưới, tương ứng là:

$$card\ (R) + card\ (S)$$

$$max\{card\ (R), card\ (S)\}$$

Chú ý rằng những công thức này giả thiết R và S không chứa các bộ giống nhau.

Hiệu. Giống như phép hợp, chúng ta chỉ trình bày các cận trên và dưới. Cận trên của $card\ (R - S)$ là $card\ (R)$, còn cận dưới là 0.

3.4.4. Xếp thứ tự nối trong các vấn tin theo mảnh

Như chúng ta đã biết việc sắp xếp các nối là một nội dung quan trọng trong quá trình tối ưu hóa vấn tin tập trung. Xếp thứ tự nối trong ngữ cảnh phân tán dĩ nhiên là quan trọng hơn bởi vì nối các mảnh làm tăng thời gian truyền. Hiện có hai cách tiếp cận cơ bản để sắp thứ tự các nối trong các vấn tin mảnh. Một là tối ưu hóa trực tiếp việc xếp thứ tự nối, còn cách kia thì thay các nối bằng các tổ hợp của nối nửa nhằm giảm thiểu chi phí truyền.

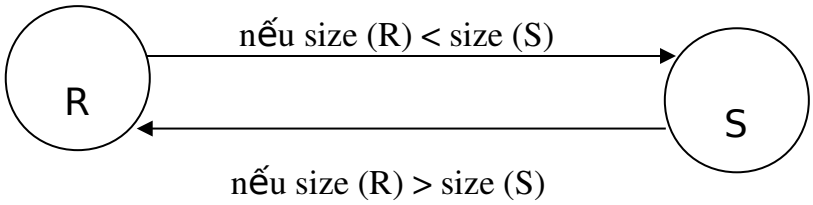
Xếp thứ tự nối.

Một số thuật toán tối ưu hóa việc sắp thứ tự nối một cách trực tiếp mà không dùng các nối nửa. Các thuật toán của hệ INGRES phân tán và System R* là đại diện cho nhóm này. Mục đích của phần này là trình bày các vấn đề phức tạp của việc sắp thứ tự nối và tạo tiền đề cho phần tiếp theo có sử dụng nối nửa để tối ưu hóa các câu vấn tin nối.

Chúng ta cần đưa ra một số giả thiết nhằm tập trung vào các vấn đề chính. Bởi vì câu vấn tin được cục bộ hóa và được diễn tả trên các mảnh, chúng ta không cần phải phân biệt giữa các mảnh của cùng một quan hệ và các mảnh được lưu tại một vị trí cụ thể. Nhằm tập trung vào việc sắp thứ tự nối, chúng ta bỏ qua thời gian xử lý cục bộ, với giả thiết là các thao tác rút gọn (chọn, chiếu) được thực hiện cục bộ hoặc trước khi, hoặc trong khi nối, (cần nhớ rằng thực hiện phé chọn trước

không phải lúc nào cũng hiệu quả). Vì thế chúng ta chỉ xét các câu vấn tin nối mà các quan hệ toán hạng được lưu tại các vị trí khác nhau. Chúng ta giả sử rằng việc di chuyển quan hệ được thực hiện theo chế độ *mỗi lần một tập* chứ không phải *nỗi lần một bộ*. Cuối cùng chúng ta bỏ qua thời gian truyền dữ liệu để có được dữ liệu tại vị trí kết quả.

Trước tiên chúng ta tập trung vào một vấn đề đơn giản hơn là truyền toán hạng trong một nối. Câu vấn tin là $R \bowtie S$, trong đó R và S là các quan hệ được lưu tại những vị trí khác nhau. Chọn lựa quan hệ để truyền, hiển nhiên là gửi quan hệ nhỏ đến vị trí của quan hệ lớn, cho ra hai khả năng như được trình bày trong Hình 7. Để có thể đưa ra một chọn lựa, chúng ta cần ước lượng kích thước của R và S. Bây giờ chúng ta xét trường hợp có nhiều hơn hai quan hệ trong một nối. Giống như trường hợp một nối đơn, mục đích của thuật toán xếp thứ tự nối là truyền những quan hệ nhỏ. Khó khăn nảy sinh từ sự kiện là các nối có thể làm giảm hoặc tăng kích thước của các quan hệ trung gian. Vì thế ước lượng kích thước kết quả nối là điều bắt buộc nhưng cũng rất khó. Một giải pháp là ước lượng chi phí truyền của tất cả các chiến lược rồi chọn ra một chiến lược tốt nhất. Tuy nhiên số lượng của các chiến lược sẽ tăng nhanh theo số quan hệ. Lối tiếp cận này, được dùng rong System*R, có chi phí tối ưu hóa cao, mặc dù nó sẽ được "trả lại" rất nhanh nếu câu vấn tin được thực hiện thường xuyên.

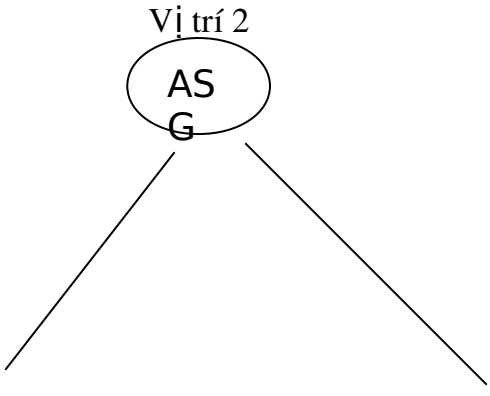


Thí dụ 3.16:

Xét câu vấn tin được biểu diễn dưới dạng đại số quan hệ:

$$\text{PROJ} \bowtie_{\text{PNO}} \text{EMP} \bowtie_{\text{ENO}} \text{ASG}$$

Với đồ thị nối được trình bày trong Hình 8. Chú ý rằng chúng ta đã đưa ra một số giả thiết về vị trí của ba quan hệ. Câu vấn tin này có thể được thực hiện ít nhất là bằng năm cách khác nhau. Chúng ta mô tả những chiến lược này bằng những chương trình sau, trong đó (R →vị trí j) biểu thị "quan hệ R được chuyển đến vị trí j"



1. EMP → vị trí 2. Vị trí 2 tính EMP' = EMP ⋈ ASG.EMP' → vị trí 3. Vị trí 3 tính EMP'⋈ PROJ

2. ASG → vị trí 1. Vị trí 1 tính EMP' = EMP ⋈ ASG.EMP' → vị trí 3. Vị trí 3 tính EMP' ⋈ PROJ

3. ASG → vị trí 3. Vị trí 3 tính ASG' = ASG ⋈ PROJ.ASG' → vị trí 1. Vị trí 1 tính ASG'⋈ EMP

4. PROJ → vị trí 2. Vị trí 2 tính PROJ' = PROJ ⋈ ASG. PROJ →vị trí 1.Vị trí 1 tính PROJ' ⋈ EMP

5. EMP → vị trí 2. PROJ →vị trí 2. Vị trí 2 tính EMP ⋈ PROJ ⋈ ASG

Để chọn ra một chương trình trong số này, chúng ta phải biết hoặc dự đoán được các kích thước: size (EMP), size (ASG), size (PROJ), size (EMP ⋈ ASG) và size (ASG ⋈ PROJ). Hơn nữa nếu xem xét cả thời gian đáp ứng, việc tối ưu hóa phải tính đến vấn đề là truyền dữ liệu có thể được thực hiện song song trong chiến lược 5. Một phương án khác để liệt kê tất cả các giải pháp là dùng các heuristic chỉ xét đến kích thước các quan hệ toán hạng bằng cách giả thiết, chẳng hạn là lực lượng của nối được tạo ra là tích của các lực lượng. Trong trường hợp này, các quan hệ được xếp thứ tự theo kích thước và thứ tự thực hiện được cho bởi cách xếp thứ tự và đồ thị nối. Thí dụ thứ tự (EMP, ASG, PROJ) có thể sử dụng chiến lược 1, còn thứ tự (PROJ, ASG, EMP) có thể dùng chiến lược 4.

Các thuật toán dựa trên nối nửa

Trong phần này chúng ta trình bày xem phải sử dụng nối nửa như thế nào để hạ thấp tổng thời gian của các vấn tin nối. Ở đây chúng ta cũng dùng giả thiết giống như trong phần 1. Thiếu sót chính của phương pháp nối được mô tả trong phần trước đó là toàn bộ quan hệ toán hạng phải được truyền qua lại giữa các vị trí. Đối với một quan hệ, nối nửa hành động như một tác nhân rút gọn kích thước giống như một phép chọn.

Nối của hai quan hệ R và S trên thuộc tính A, được lưu tương ứng tại vị trí 1 và 2, có thể được tính bằng cách thay một hoặc cả hai toán hạng bằng một nối nửa với quan hệ kia nhờ các quy tắc sau đây.$R \bowtie_A S \Leftrightarrow (R \mid\!><_A S) \bowtie_A S$

$$\Leftrightarrow R \bowtie_A (S \mid\!><_A R)$$

$$\Leftrightarrow (R |><|_A S) \bowtie_A (S |><|_A R)$$

Chọn lựa giữa một trong ba chiến lược nối nửa đòi hỏi phải ước lượng các chi phí tương ứng của chúng. Sử dụng nối nửa sẽ có ích nếu chi phí tạo và gửi nó đến vị trí kia nhỏ hơn chi phí gửi toàn bộ quan hệ toán hạng và thực hiện nối thực hiện. Để minh họa ích lợi của nối nửa, chúng ta hãy so sánh các chi phí của hai chọn lựa. $R \bowtie {}_AS$ với $(R |><|_A S) \bowtie_A S$, giả thiết rằng size(R) <size(S)

Chương trình sau đây, sử dụng ký pháp của phần 1, dùng nối nửa:

1. $\pi_A(S) \rightarrow$ vị trí 1

2. Vị trí 1 tính $R' = R |><|_A S$

3. $R' \rightarrow$ vị trí 2

4. Vị trí 2 tính $R' \bowtie {}_AS$

Để cho đơn giản, chúng ta hãy bỏ qua hằng T_{MSG} trong thời gian truyền với giả thiết là toán hạng T_{TR} * size(R) lớn hơn nó rất nhiều. Sau đó chúng ta có thể so sánh hai chọn lựa này theo số lượng dữ liệu được truyền. Chi phí của thuật toán dựa trên nối là chi phí truyền quan hệ R đến vị trí 2. Chi phí của thuật toán dựa trên nối nửa là chi phí của bước 1 và 3 ở trên. Vì thế phương pháp nối nửa sẽ tốt hơn nếu:

Size $(\pi_A(S))$ + size$(R |><|_A S) <$ size(R)

Phương pháp nối nửa tốt hơn nếu nối hành động như một tác nhân rút gọn đầy đủ, nghĩa là nếu chỉ một số ít các bộ của R tham gia vào trong nối. Phương pháp nối tốt hơn nếu hầu như tất cả các bộ của R đều tham gia vào nối bởi vì phương pháp nối nửa đòi hỏi thêm một lần truyền kết quả chiếu trên thuộc tính nối. Chi phí của bước thực hiện chiếu có thể hạ thấp tối đa bằng các mã hóa kết quả chiếu trong các mảng Bit, nhờ đó làm giảm đi chi phí truyền các giá trị thuộc tính được nối. Điều quan trọng cần biết rằng không có cách tiếp cận nào là tốt nhất; chúng cần được xem như là những bổ trợ cho nhau.

Tổng quát hơn nối nửa có thể làm giảm đi kích thước của quan hệ toán hạng có trong các câu vấn tin đa nối. Tuy nhiên việc tối ưu hóa vấn tin sẽ trở lên phức tạp hơn trong trường hợp này. Hãy xét lại đồ thị nối của các quan hệ EMP, ASG, PROJ được cho trong Hình 9. Chúng ta có thể áp dụng thuật toán nối trước đây bằng cách dùng các nối nửa cho từng nối. Vì thế một thí dụ về một chương trình tính $EMP \bowtie ASG \bowtie PROJ$ là $EMP' \bowtie ASG'$ EMP PROJ, trong đó $EMP' = EMP |><| ASG$ và $ASG' = ASG |><| PROJ$

Tuy nhiên chúng ta có thể rút gọn thêm nữa kích thước của một quan hệ toán hạng bằng cách dùng nhiều nối nửa, Thí dụ EMP' trong chương trình ở trên có thể được thay bằng EMP" và được tính bằng:

$EMP'' = EMP |><| (ASG |><| PROJ)$

Bởi vì nếu size(ASG |><PROJ) ≤ size(ASG) chúng ta có size(EMP") ≤ size(EMP'). Do đó theo cách này, EMP được rút bởi một chuỗi nói nửa: EMP |>< (ASG |>< PROJ).

Một dãy nối nửa như thế được gọi là một chương trình nối nửa cho EMP. Tương tự chúng ta có thể có các chương trình nối nửa cho một quan hệ bất kỳ. Thí dụ PROJ có thể được rút gọn bằng một chương trình nối nửa PROJ |>< (ASG |>< EMP) Tuy nhiên không phải tất cả các quan hệ có trong một câu vấn tin đều cần phải rút gọn; đặc biệt chúng ta có thể bỏ qua những quan hệ không có mặt trong các nối cuối cùng.

Đối với một quan hệ đã cho sẽ có nhiều chương trình nối nửa khác nhau. Số lượng các khả năng thực hiện tỷ lệ là mũ theo số quan hệ. Thế nhưng có một chương trình nối nửa tối ưu, được gọi là trình rút gọn hoàn toàn, mà với mỗi quan hệ R nó rút gọn R nhiều hơn các chương trình khác. Vấn đề là tìm ra trình rút gọn hoàn toàn này. Một phương pháp đơn giản là ước lượng kết quả rút gọn kích thước của tất cả mọi chương trình nối nửa khả hữu và chọn ra chương trình tốt nhất, các bài toán liệt kê gặp phải hai vấn đề.

1. Một lớp vấn tin có các chu trình trong đồ thị nối, được gọi là vấn tin có vòng (cyclic queries), và loại vấn tin này không có trình rút gọn hoàn toàn (full reducer).

2. Một loại vấn tin khác, được gọi là vấn tin cây (Tree queries) thì có trình rút gọn hoàn toàn nhưng số lượng các chương trình nối nửa cần kiểm tra tỷ lệ làm hàm mũ theo số lượng quan hệ, khiến cho phương pháp liệt kê trở thành các loại bài toán NP-hard.

Trong những đoạn sau chúng ta thảo luận về các giải pháp cho những bài toán này.

Thí dụ 3.17:

Xét các quan hệ sau, trong đó có thêm thuộc tính CITY cho các quan hệ EMP (có tên mới là ET) và PROJ (có tên mới là PT) của CSDL:

ET (ENO, ENAME, TITLE, CITY)

ASG (ENO, PNO, RESP, DUR)

PT (PNO, PNAME, BUDGET, CITY)

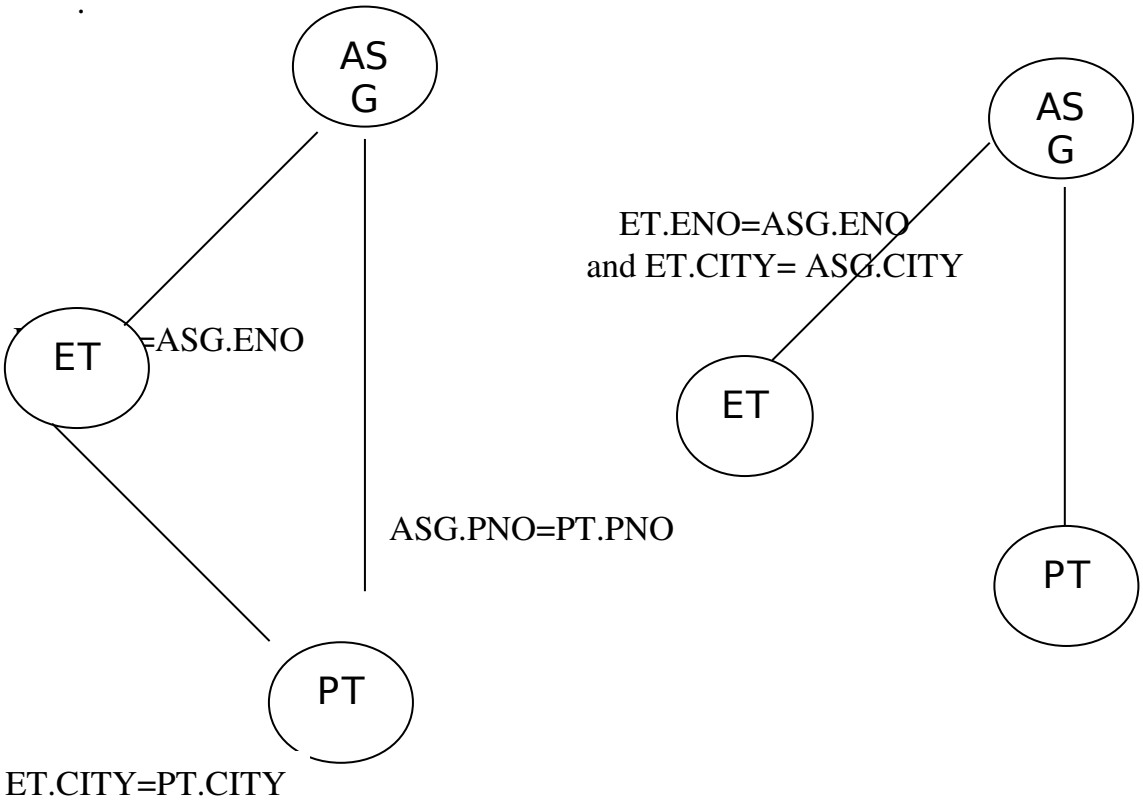
Câu vấn tin sau xuất tên của tất cả nhân viên sống trong thành phố có dự án của họ đang được thực hiện

```
SELECT      ET.ENAME
FROM        ET, ASG, PT
WHERE       ET.ENO = ASG.ENO
```

AND ASG. PNO = PT. PNO

AND ET.CITY = PT. CITY

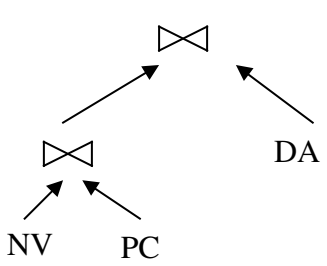
Không có trình rút gọn hoàn toàn nào cho câu vấn tin trong Thí dụ 9.7. Thực sự có thể dẫn xuất được các chương trình nối nửa để rút gọn nó nhưng số lượng các phép toán nhân với số lượng các bộ trong mỗi quan hệ khiến cho lối tiếp cận nayfkhoong hiệu quả. Một giải pháp là biến đổi đồ thị có vòng thành một cây bằng cách loại bỏ một cung của đồ thị và thêm vào các vị từ thích hợp cho các cung khác sao cho vị từ được loại bỏ toàn nhờ tính bắc cầu



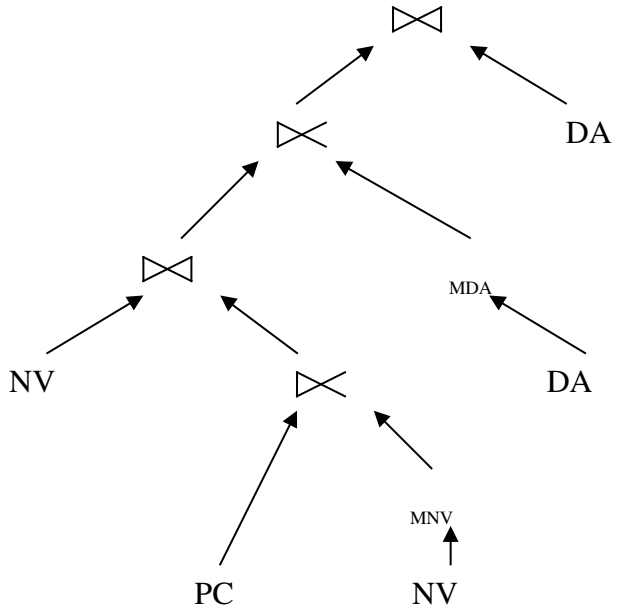
(a) vấn tin có vòng (b) Vấn tin không vòng tương đương

Ở thí dụ hình b, trong đó cung (ET, PT) được loại bỏ, vị từ được thêm vào ET.CITY= ASG.CITY và ASG.CITY= PT.CITY kéo theo ET.CITY=PT.CITY nhờ tính bắc cầu. Vì thế câu vấn tin không vòng tương đương với vấn tin có vòng. Việc thêm những vị từ này dẫn đến việc thêm thuộc tính CITY trong quan hệ ASG. Vì thế các giá trị cho thuộc tính CITY phải được gửi tới ET hoặc ASG.

Mặc dù trình rút gọn hoàn toàn cho các vấn tin cây có tồn tại, bài toán tìm ra chúng thuộc loại NP-hard. Tuy nhiên có một lớp vấn tin quan trọng gọi là vấn tin mắt xích (chained query) có một thuật toán đa thức cho chúng. Một vấn tin mắt xích có một đồ thị nối, trong đó các quan hệ có thể được sắp thứ tự , và mối quan hệ chỉ nối với quan hệ kế tiếp theo thứ tự đó.Thí dụ câu vấn tin trong hình tren là một vấn tin mắt xích. Do rất khó cài đặt một thuật toán với các trình rút gọn hoàn toàn, phần lớn các hệ thống đều dùng các nối nửa đơn lẻ để rút gọn kích thước quan hệ.



Phương pháp nối



Phương pháp nối nửa

So sánh nối và nối nửa

Nếu so sánh và nối nửa phải thực hiện nhiều phép toán hơn nhưng rất có thể trên các toán hạng nhỏ hơn. Hình tren minh họa những khác biệt này qua một cặp chiên lược nối và nối nửa tương đương cho câu vấn tin có đồ thị nối được cho trong Hinh 11. Nối của hai quan hệ EMP⋈ ASG trong hình được thực hiện bằng cách gửi một quan hệ, chẳng hạn ASG đến vị trí của quan hệ kia EMP và hoàn toàn nối tại vị trí đó. Tuy nhiên khi dùng nối nửa thì tránh phải truyền quan hệ ASG. Thay vào đó là truyền các giá trị thuộc tính nối của EMP đến vị trí của ASG, sau đó là truyền các bộ đối so sánh được của quan hệ ASG đến vị trí của quan hệ EMP, rồi hoàn tất nối ở đó. Nếu chiều dài thuộc tính nối nhở hơn chiều dài của một bộ và nối nửa có độ tuyển chọn tốt thì phương pháp nối nửa có thể làm tăng thời gian xử lý cục bộ bởi vì một trong hai quann hệ được nối phải truy xuất hai lần. Thí dụ các quan hệ EMP và PROJ được truy xuất hai lần. Hơn nữa nối của hai quan hệ trung gian sinh ra từ nối nửa không tận dụng được chỉ mục có sẵn trên quan hệ cơ sở. Vì thế sử dụng nối nửa có thể không phải là ý kiến hay nếu thời gian truyền dữ liệu không phải là yếu tố chiếm ưu thế như trường hợp các mạng cục bộ.

Nối nửa vẫn có ích trong các mạng tốc độ cao nếu chúng ta có độ tuyển chọn rất tốt và được cài đặt bằng các mảng bit Một mảng bit BA[1:n] rất có ích trong việc mã hóa các giá trị thuộc tính nối có trong quan hệ. Chúng ta hãy xét nối nửa R |>< S. Thế thì BA[i] được đặt là 1 nếu tồn tại một giá trị thuộc tính nối A al trong quan hệ S sao cho h(val) = i trong đó h là một hàm băm. Bằng không thì BA[i] được đặt bằng 0. Một mảng bit như thế sẽ nhỏ hơn nhiều so với một danh sách các giá trị thuộc tính nối. Vì thế truyền một mảng bit thay vì giá trị thuộc tính nối đến vị trí của quan hệ R sẽ tiết kiệm được thời gian truyền tin. Nối nửa có thể được thực hiện như sau. Mỗi bộ của quan hệ R có giá trị thuộc tính nối là val sẽ thuộc về nối nửa nếu BA[h(val)] = 1.

CHƯƠNG 4. QUẢN LÝ GIAO DỊCH

4.1. Các khái niệm

1.5- Quản lý giao dịch

Giao dịch

Có nhiều định nghĩa khác nhau về *giao dịch* (transaction). Trong đó có 2 định nghĩa được nhiều người sử dụng là của Jeffrey D. Ullman và M.Tamer Ozsu và Patrick Valduriez

Jeffrey D. Ullman [10] cho rằng, *giao dịch* là một thực hiện của một chương trình. Chương trình này có thể là một câu vấn tin hoặc một chương trình trong ngôn ngữ chủ, trong đó có gắn vào một ngôn ngữ vấn tin. Một *giao dịch* sẽ được đọc dữ liệu từ một CSDL vào vùng làm việc riêng (private workpace), thực hiện các tính toán trong vùng làm việc này và ghi dữ liệu từ vùng làm việc này vào cơ sở dữ liệu. Như vậy, các tính toán do giao dịch thực hiện không làm thay đổi cơ sở dữ liệu cho đến khi các giá trị mới được ghi vào cơ sở dữ liệu.

Một định nghĩa khác của M. Tamer Ozsu và Patrick Valduriez cho rằng *giao dịch* là một đơn vị tính toán nhất quán và tin cậy. Điều này có nghĩa là, một giao dịch thực hiện một truy xuất trên cơ sở dữ liệu, gây ra một sự biến đổi trạng thái. Nếu cơ sở dữ liệu đã nhất quán trước khi thực hiện giao dịch thì cũng sẽ nhất quán khi kết thúc giao dịch cho dù giao dịch này có thực hiện đồng thời với các giao dịch khác hoặc xảy ra sự cố trong lúc nó được thực hiện.

Nói chung, một *giao dịch* được xem như được tạo bởi một dãy các thao tác đọc và ghi trên cơ sở dữ liệu cùng với các bước tính toán cần thiết. Theo nghĩa này, một *giao dịch* được xem như một chương trình có các câu vấn tin truy vấn đến cơ sở dữ liệu được gắn vào. *Giao dịch* là một thực hiện của một chương trình. Một câu vấn tin cũng có thể được xem là một chương trình và được đưa ra như một giao dịch.

Quản lý giao dịch: Bộ quản lý giao dịch (transaction manager - TM) chịu trách nhiệm điều phối việc thực hiện các thao tác cơ sở dữ liệu của các ứng dụng. Bộ quản lý giao dịch cài đặt một giao diện cho các ứng dụng, bao gồm các lệnh: *begin-transaction, read, write, commit và abort*. Các lệnh này được xử lý trong một DBMS phân tán.

Mục dữ liệu

Mục dữ liệu (item) là các đơn vị dữ liệu trong cơ sở dữ liệu. Bản chất và kích thước các *mục dữ liệu* do nhà thiết kế chọn. Kích thước của các đơn vị này được lựa chọn sao cho việc truy xuất dữ liệu có hiệu quả. Chẳng hạn trong mô hình dữ liệu quan hệ, chúng ta có thể chọn các mục lớn như các quan hệ, hoặc các mục nhỏ như các bộ hay thành phần của các bộ. Kích thước của các *mục dữ liệu* được hệ thống sử dụng gọi là độ *mịn* (granularity) của hệ thống. Một hệ thống được gọi là *mịn* (fine-grained), nếu nó sử dụng các mục dữ liệu nhỏ và hệ thống là *thô* (coarse-grained), nếu nó sử dụng các mục dữ liệu lớn. Độ càng thô sẽ giảm chi phí quản lý việc truy cập các mục, ngược lại độ càng mịn lại cho phép nhiều hoạt động đồng thời hơn.

Phương pháp thông dụng nhất để điều khiển việc truy cập các mục là sử dụng *khoá chốt (*lock*)*. *Bộ quản lý khoá chốt* (lock manager) là thành phần của DBMS trịu trách nhiệm theo dõi một mục *I* hiện có giao dịch nào đang đọc ghi vào các thành phần của *I* hay không. Nếu có thì bộ quản lý khoá chốt sẽ cản trở ngăn cản không cho giao dịch khác truy cập *I* trong trường hợp truy cập có thể xảy ra xung đột, chẳng hạn việc bán một ghế trên một chuyến máy bay hai lần.

Bộ xếp lịch và các giao thức

Để ngăn ngừa sự bế tắc người ta có thể sử dụng *bộ lập lịch* (scheduler) và các *giao thức* .

Bộ lập lịch là một thành phần của hệ thống cơ sở dữ liệu, có vai trò làm trọng tài phân xử các yêu cầu đang có xung đột, chịu trách nhiệm sắp xếp một lịch biểu cho các thao tác của các giao dịch. Chẳng hạn chúng ta đã biết cách loại bỏ khoá sống của một bộ lập lịch "đến trước phục vụ trước". Một bộ lập lịch cũng có thể xử lý các khoá gài và tính bất khả tuần tự bằng cách:

Nó cũng có thể buộc một giao dịch phải đợi cho đến khi khoá mà giao dịch yêu cầu được giải phóng.

Buộc một giao dịch phải huỷ bỏ và tái thực hiện.

Giao thức theo nghĩa tổng quát nhất, *chỉ là một hạn chế trên chuỗi các bước nguyên tử mà một giao dịch có thể thực hiện*, là các qui tắc mà các giao dịch phải tuân theo. Chẳng hạn, chiến lược tránh khoá gài bằng cách yêu cầu khoá chốt trên các mục theo một thứ tự cố định nào đó chính là một giao thức.

Các tính chất của giao dịch

1) Tính nguyên tử

Quản lý giao dịch là một cố gắng làm cho các thao tác phức tạp xuất hiện dưới dạng nguyên tử (atomic). Nghĩa là một thao tác có thể xảy ra trọn vẹn hoặc không xảy ra. Nếu xảy ra, không có biến cố hay *giao dịch* nào cùng xảy ra trong suốt thời gian tồn tại của nó. Cách thông dụng nhằm đảm bảo được tính nguyên tử của các giao dịch là *phương pháp tuần tự hoá* (serialization). Phương pháp này làm cho các giao dịch được thực hiện tuần tự.

Một giao dịch không có tính nguyên tử nếu:

- Trong một hệ thống phân chia thời gian, lát thời gian cho giao dịch T có thể kết thúc trong khi T đang tính toán và các hoạt động của một giao dịch khác sẽ được thực hiện trước khi T hoàn tất. Hoặc

- Một giao dịch không thể hoàn tất được. Chẳng hạn khi nó phải chấm dứt giữa chừng, có thể vì nó thực hiện một phép tính không hợp lệ, hoặc có thể do đòi hỏi những dữ liệu không được quyền truy xuất. Bản thân hệ thống CSDL có thể buộc giao dịch này ngừng lại vì nhiều lý do. Chẳng hạn giao dịch có thể bị kẹt trong một khoá gài.

Trong thực tế, mỗi giao dịch đều có một chuỗi các bước cơ bản như: *đọc* hay *ghi* một *mục dữ liệu* (item) vào CSDL và *thực hiện các phép tính toán số học đơn giản trong vùng làm việc*, hoặc các bước sơ đẳng khác như các bước *khoá* chốt, *giải phóng khoá, uỷ thác (hoàn tất)* giao dịch và có thể có những bước khác nữa. Chúng ta luôn giả sử rằng những bước sơ đẳng này là nguyên tử. Thậm chí thao tác kết thúc *lát thời gian* xảy ra khi đang tính toán cũng có thể xem là nguyên tử. Bởi vì nó xảy ra trong vùng làm việc cục bộ và không có gì có thể ảnh hưởng đến vùng làm việc đó cho đến khi giao dịch đang thực hiện dở các phép tính số học được tái hoạt động trở lại.

2)Tính nhất quán (consistency)

Tính nhất quán của một giao dịch chỉ đơn giản là tính đúng đắn của nó. Nói cách khác, một giao dịch là một chương trình đúng đắn, ánh xạ cơ sở dữ liệu từ trạng thái nhất quán này sang trạng thái nhất quán khác. Việc xác nhận rằng các giao dịch nhất quán là vấn đề điều khiển dữ liệu ngữ nghĩa. Cơ sở dữ liệu có thể tạm thời không nhất quán khi giao dịch đang thực hiện, nhưng nó phải trở về trạng thái nhất quán khi kết thúc giao dịch.

Tính nhất quán giao dịch muốn nói đến hành động của các giao dịch đồng thời. Chúng ta mong rằng CSDL vẫn nhất quán ngay cả khi có một số yêu cầu của người

sử dụng đồng thời truy nhập đến CSDL. Tính chất phức tạp nảy sinh khi xét đến các CSDL có nhân bản.

Nếu cơ sở dữ liệu được nhân bản, tất cả các bản sao phải có trạng thái giống nhau vào lúc kết thúc giao dịch. Điều này gọi là tính tương đương một bản, và trạng thái nhất quán của các bản sao được gọi là trạng thái nhất quán tương hỗ.

3) Tính biệt lập

Tính biệt lập là tính chất của các giao dịch, đòi hỏi mỗi giao dịch phải luôn nhìn thấy cơ sở dữ liệu nhất quán. Nói cách khác, một giao dịch đang thực thi không thể làm lộ ra các kết quả của nó cho những giao dịch khác đang cùng hoạt động trước khi nó uỷ thác.

Có một số lý do cần phải nhấn mạnh đến tính biệt lập:

Một là phải duy trì tính nhất quán qua lại giữa các giao dịch. Nếu hai giao dịch đồng thời truy xuất đến một mục dữ liệu đang được một trong chúng cập nhật thì không thể bảo đảm rằng giao dịch thứ hai đọc được giá trị đúng.

Hai là, tính biệt lập cho phép khắc phục các hiện tượng huỷ bỏ dây chuyền (cascading abort). Bởi vì nếu mỗi giao dịch cho phép các giao dịch khác đọc các mục mà nó đang thay đổi trước khi có uỷ thác, nếu nó bị huỷ bỏ, mọi thao tác đọc các giá trị những mục này cũng phải huỷ bỏ theo. Điều này gây những chi phí đáng kể cho DBMS.

Vấn đề biệt lập có liên quan trực tiếp đến tính nhất quán CSDL và vì thế là đề tài của điều khiển đồng thời.

4) Tính bền vững

Tính bền vững muốn nói đến tính chất của giao dịch bảo đảm rằng một khi giao dịch uỷ thác, kết quả của nó được duy trì cố định và không bị xoá ra khỏi CSDL. Vì thế DBMS bảo đảm rằng kết quả của giao dịch sẽ vẫn tồn tại dù có xảy ra các sự cố hệ thống. Đây chính là lý do mà chúng ta đã nhấn mạnh rằng giao dịch uỷ thác trước khi nó thông báo cho người sử dụng biết rằng nó đã hoàn tất thành công, tính bền vững đưa ra các vấn đề khôi phục dữ liệu, nghĩa là cách *khôi phục CSDL* về trạng thái nhất quán mà ở đó mọi hành động đã uỷ thác đều được phản ánh.

Tính khả tuần tự của các lịch biểu và việc sử dụng chúng

Mục đích của giao thức điều khiển tương tranh là xếp lịch thực hiện sao cho không xảy ra sự tác động lẫn nhau giữa chúng. Có một giải pháp đơn giản: Chỉ cho phép một giao dịch thực hiện tại một thời điểm. Nhưng mục đích của hệ quản trị cơ sở dữ liệu đa người dùng lại là tối đa hoá sự thực hiện đồng thời trong hệ thống sao cho những giao dịch thực hiện đồng thời không ảnh hưởng lẫn nhau.

Lịch biểu là một dãy (có thứ tự) các thao tác của một tập các giao dịch tương tranh mà trong đó thứ tự của mỗt thao tác trong mỗi giao dịch được bảo toàn.

Đây là vấn đề xử lý hoạt động đồng thời có liên quan đến các nhà thiết kế CSDL chứ không phải các nhà thiết kế các hệ thống đồng thời tổng quát.

Giả sử chúng ta có một tập các giao dịch S={T_1, T_2, T_3, ... }.

Lịch biểu tuần tự: Chúng ta thấy ngay rằng nếu các giao dịch thực hiện *tuần tự* theo một thứ tự nào đó, các thao tác của mỗi giao dịch được thực hiện kế tiếp nhau, không có một thao tác nào của các giao dịch khác xen kẽ vào thì các sự cố tranh chấp chắc chắn không xảy ra và trong CSDL chúng ta có một kết quả nào đó.

Chúng ta định nghĩa một *lịch biểu* cho một tập các giao dịch S là *thứ tự (có thể xen kẽ) các bước cơ bản của của các giao dịch (khoá, đọc, ghi, ...) được thực hiện.*

Các bước của một giao dịch đã cho phải xuất hiện trong lịch biểu theo đúng thứ tự xảy ra trong giao dịch đó.

Lịch biểu không tuần tự: Là lịch mà trong đó các thao tác của một tập các giao dịch tương tranh được xen kẽ vào nhau.

Bởi vì *luôn có lịch biểu tuần tự* cho tập giao dịch S vì vậy chúng ta sẽ giả sử rằng hoạt động của các giao dịch đồng thời là *đúng đắn* nếu và chỉ nếu *tác dụng của nó giống như tác dụng có được của lịch biểu tuần tự.*

Lịch biểu được gọi là *khả tuần tự* (serializable) *nếu tác dụng của nó giống với tác dụng của một lịch biểu tuần tự.*

Lịch biểu được gọi là bất khả tuần tự nếu tác dụng của nó không giống với tác dụng của lịch biểu tuần tự.

Mục tiêu của bộ xếp lịch là với một tập các giao dịch đồng thời, đưa ra được một lịch biểu khả tuần tự.

Trong việc tuần tự hoá, thứ tự của các thao tác đọc và ghi rất quan trọng:

- Nếu hai thao tác chỉ đọc một mục dữ liệu thì chúng sẽ không ảnh hưởng đến nhau và thứ tự giữa chúng không quan trọng

- Nếu hai thao tác đọc hay ghi trên hai mục dữ liệu hoàn toàn khác nhau thì chúng sẽ không ảnh hưởng đến nhau và thứ tự giữa chúng không quan trọng
- Nếu một thao tác ghi một mục dữ liệu và một thao tác khác đọc hay ghi trên chính mục dữ liệu này thì thứ tự giữa chúng rất quan trọng.

Xét các lịch biểu

Lịch biểu tuần tự		Lịch biểu khả tuần tự		Lịch biểu bất khả tuần tự	
T_1	T_2	T_1	T_2	T_1	T_2
Read A		ReadA		ReadA	
A:=A-10			ReadB	A:=A-10	
WriteA		A:=A-10			ReadB
Read B			B:=B –20	WriteA	
B:=B+10		WrieA			B:=B-20
WriteB			WriteB	ReadB	
	ReadB	ReadB			WriteB
	B:=B-20		Read C	B:=B+10	
	WriteB	B:=B+10			ReadC
	Read C		C:=C+20	WriteB	
	C:=C+20	WriteB			C:=C+20
	Write C		WriteC		Write C
(a)		(b)		(c)	

Hình 4.1. Một số lịch biểu

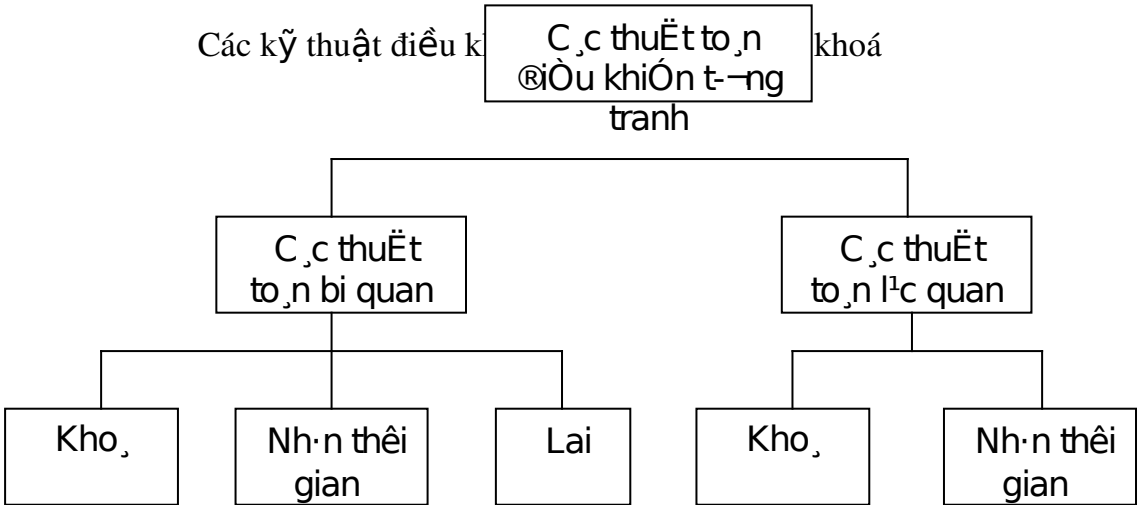
Hình 4.1 (a) là một lịch biểu tuần tự

Hình 4.1 (b) là lịch biểu khả tuần tự

Hình 4.1 (c) là lịch biểu bất khả tuần tự

Trong thực tế, qua các tính chất của đại số đơn thuần chúng ta có thể gặp một lịch biểu là bất khả tuần tự nhưng nó cho cùng kết quả so với lịch biểu tuần tự.

Các kỹ thuật điều k[illegible] khoá



Khoá

Khoá (Lock) là một đặc quyền của một giao dịch được bộ quản lý khoá trao cho để có thể truy cập trên một mục dữ liệu. Hay khoá là một biến gắn với một mục dữ liệu trong cơ sở dữ liệu để biểu diễn trạng thái của một mục dữ liệu này trong mối liên quan đến thao tác thực hiện trên đó. Bộ quản lý khoá cũng có thể thu hồi lại khoá này. Tại một thời điểm, mục dữ liệu X có một trong 3 trạng thái:

- Có *khoá đọc* (read-lock)(còn gọi là khoá chia sẻ – shared lock): chỉ cho phép một giao dịch đọc một mục nhưng không được cập nhật trên mục này.

- Có *khoá ghi* (wrire-lock) (còn gọi là khoá độc quyền – exclusive lock): cho phép thực hiện cả hai thao tác đọc ghi.

- *Không có khoá.*

Các khoá được sử dụng theo cách sau:

+ Bất kỳ một giao dịch nào cần truy cập vào một mục dữ liệu trước hết phải khoá mục dữ liệu đó lại. Giao dịch sẽ yêu cầu một khoá đọc nếu chỉ cần đọc dữ liệu và yêu cầu khoá ghi nếu vừa cần đọc và cần ghi dữ liệu.

+ Nếu mục dữ liệu đó chưa bị khoá bởi một giao dịch nào khác thì khoá sẽ được cấp phát theo đúng yêu cầu.

+ Nếu mục dữ liệu đó đang bị khoá, HQT CSDL sẽ xác định xem khoá được yêu cầu có tgương thích với khoá hiện hành hay không. Khi một giao dịch yêu cầu cấp một khoá đọc cho nó trên một mục dữ liệu mà trên mục đó đang có một khoá đọc (của giao dịch khác) thì khoá yêu cầu này được cấp phát. Trong trường hợp khoá yêu cầu là khoá ghi thì giao dịch yêu cầu khoá sẽ phải chờ cho đến khi khoá hiện hành được giải phóng mới được cấp khoá.

+ Một giao dịch tiếp tục giữ một khoá cho đến thời điểm khoá đó được giải phóng, thời điểm này hoặc nằm trong quá trình thực hiện giao dịch hoặc là thời điểm giao dịch được chuyển giao hay bị huỷ bỏ. Chỉ khi khoá ghi được giải phóng thì kết quả cua thao tác ghi mới thấy được đối với các giao dịch khác.

Một số hệ thống còn cho phép giao dịch đưa các khoá đọc trên một mục dữ liệu và sau đó nâng cấp khoá lên thành khoá ghi. Điều này cho phép một giao dịch kiểm tra dữ liệu trước, sau đó mới quyết định có cập nhật hay không.

Bộ quản lý khoá lưu các khoá trong một *bảng khoá* (lock table).

Khi điều khiển các hoạt động tương tranh bằng khoá, có thể xảy ra các tình huống:

Khoá sống (live-lock) là tình huống mà một giao dịch yêu cầu khoá trên một mục mà chẳng bao giờ nhận được khoá trong khi luôn có một giao dịch khác giữ khoá trên mục này (*khoá sống* trên mục A của giao dịch T là khoá không khoá được A vì A luôn bị khoá bởi một giao dịch khác), mặc dù có một số lần giao dịch này có cơ hội nhận khoá trên mục đó. Rất nhiều giải pháp đã được các nhà thiết kế hệ điều hành đề xuất vấn đề giải quyết *khoá sống*. Có thể sử dụng một chiến lược đơn giản *"đến trước, phục vụ trước"* để loại bỏ được khoá sống.

Bế tắc hay *khoá gài* (deadlock) là tình huống mà trong đó mỗi giao dịch trong một tập hay nhiều giao dịch đang đợi nhận khoá của một mục hiện đang bị khoá bởi một giao dịch khác trong một tập giao dịch đó và ngược lại (một mục trong giao dịch này bị *gài* bởi giao dịch khác và ngược lại).

Ví dụ Giả sử có hai giao dịch đồngthời T_1 và T_2 như sau:

T_1 : Lock A ; Lock B ; Unlock A ; Unlock B;

T_2 : Lock B ; Lock A ; Unlock B ; Unlock A;

T_1 và T_2 cùng có thể thực hiện một số tác vụ nào đó trên A và B. Giả sử T_1 và T_2 được thực hiện cùng lúc. T_1 yêu cầu và được trao khoá trên A, còn T_2 yêu cầu và được trao khoá trên B. Do đó khi T_1 yêu cầu khoá trên B nó sẽ phải đợi vì T_2 đã khoá B. Tương tự T_2 yêu cầu khoá trên A nó sẽ phải đợi vì T_1 đã khoá A. Kết quả là không một giao dịch nào tiếp tục hoạt động được: mỗi giao dịch đều phải đợi giao dịch kia mở khoá, và chúng đều phải đợi nhưng chẳng bao giờ nhận được khoá như yêu cầu.

Để tránh bế tắc có thể sử dụng các giải pháp:

(i) *Buộc các giao dịch phải đưa ra tất cả các yêu cầu khoá* cùng một lúc và bộ *quản lý khoá* trao tất cả các *khoá* cho chúng nếu được, hoặc không trao và cho giao dịch này đợi nếu một hay nhiều khoá được yêu cầu đang bị giữ bởi một giao dịch khác.

(ii) *Gán một thứ tự tuyến tính* cho các mục và yêu cầu tất cả các giao dịch phải xin khoá theo đúng thứ tự này.

(iii) *Một cách khác để xử lý bế tắc* là định kỳ kiểm tra yêu cầu khoá và phát hiện có xảy ra bế tắc không. Bằng cách dùng đồ thị chờ, với các nút biểu diễn các giao dịch và các cung $T_i \rightarrow T_j$ biểu thị T_j đang đợi nhận khoá trên một mục đang được T_i giữ. Nếu trong đồ thị có chu trình, sự bế tắc sẽ xảy ra, và nếu không có chu trình thì kết luận không có khoá gài hay bế tắc. Nếu một khoá gài bị phát hiện, khi đó hệ thống sẽ buộc một trong các giao dịch bị bế tắc phải khởi động lại và tác dụng của giao dịch đó trên cơ sở dữ liệu phải được hoàn toàn trả lại.

4. 2. Mô hình khoá cơ bản

Khoá (Lock) là một đặc quyền truy cập trên một mục dữ liệu mà bộ quản lý khoá (lock manager) có thể trao cho một giao dịch hoặc thu hồi lại. Trong các mô hình giao dịch có sử dụng khoá không chỉ có các thao tác đọc và ghi các mục mà còn có thao tác khoá (lock) và mở khoá (unlock) chúng. Mỗi mục được khoá phải được mở khoá sau đó. Với một mục A, giữa bước lock A và unlock A kế tiếp của một giao dịch, giao dịch này phải được coi là đang giữ một khoá trên A.

Trong mô hình này, chúng ta dựa trên các giả định sau:

- Một khoá phải được đặt trên một mục trước khi đọc hay ghi mục đó.

- Các thao tác khoá hoạt động trên cơ sở đồng bộ hoá, nghĩa là nếu một giao dịch khoá một mục đã bị khoá trước đó bởi một giao dịch khác, nó không thể thao tác trên mục này cho đến khi khoá này được giải phóng bằng lệnh mở khoá do giao dịch đang giữ khoá trước đó thực hiện.

- Mỗi giao dịch đều có thể mở được mọi khoá do chính nó khoá.

- Một giao dịch sẽ không yêu cầu khoá một mục nếu nó đang hiện giữ khoá của mục đó, hoặc mở khoá một mục mà nó hiện không giữ khoá trên mục đó.

Các lịch biểu tuân theo quy tắc này được gọi là *hợp lệ* .

Ví dụ : Xét hai giao dịch đồng thời T_1 và T_2 cùng truy xuất đến mục dữ liệu A theo mô hình này sẽ là:

T_1	Lock A	T_2	Lock A
	Read A		Read A
	A:=A+1		A:=A+1
	Write A		Write A
	Unlock A		Unlock A

Nếu T_1 bắt đầu trước T_2, nó sẽ yêu cầu khoá trên mục A. Giả sử không có giao dịch nào đang khoá A, bộ quản lý khoá sẽ cho nó khoá mục này. Khi đó chỉ có T_1 mới được truy xuất đến mục này. Nếu T_2 bắt đầu trước khi T_1 chấm dứt thì T_2 thực hiện Lock A, hệ thống buộc T_2 phải đợi. Chỉ đến khi T_1 thực hiện lệnh Unlock A, hệ thống mới cho phép T_2 tiến hành. Như vậy, T_1 hoàn thành trước khi T_2 bắt đầu và kết quả là sau hai giao dịch, giá trị của A sẽ là 32

Với mô hình này, để kiểm tra tính khả tuần tự của một lịch biểu, ta sẽ xem xét thứ tự mà các giao dịch khoá một mục đã cho. Thứ tự này phải thống nhất với thứ tự trong lịch biểu tuần tự tương đương. Đây thực chất là việc kiểm tra một đồ thị có chu trình hay không.

Thuật toán 2.1: Kiểm tra tính khả tuần tự của một lịch biểu.

Nhập: Một lịch biểu S cho một tập các giao dịch T_1, T_2 , ... , T_k.

Xuất: Khẳng định S có khả tuần tự hay không? Nếu có thì đưa ra một lịch biểu tuần tự tương đương với S.

Phương pháp:

Bước 1: Tạo ra một đồ thị có hướng G (gọi là đồ thị tuần tự hoá), có các nút là các giao dịch, các cung của đồ thị này được xác định như sau:

Gọi S là a_1, a_2, ... a_n

trong đó mỗi a_i là một thao tác của một giao dịch có dạng

T_j : Lock A_m hoặc T_j : Unlock A_m

với T_j là giao dịch thực hiện thao tác khoá hoặc mở mục A_m.

Nếu a_i là T_j : Unlock A_m

thì hành động a_p kế tiếp a_i có dạng T_s : Lock A_m. Nếu s j thì vẽ một cung từ T_j đến T_s. Cung này có nghĩa là trong lịch biểu tuần tự tương đương, T_j phải đi trước T_s.

Bước 2: Kiểm tra, G có chu trình thì S bất khả tuần tự. Nếu G không có chu trình thì ta tìm một thứ tự tuyến tính cho các giao dịch, trong đó T_i đi trước T_j khi có một cung đi từ T_i T_j. Để tìm thứ tự tuyến tính đó, ta thực hiện quá trình sắp xếp topo như sau. Đầu tiên ta xuất phát từ một nút T_i không có cung vào (ta luôn tìm thấy một nút như thế, nếu không thì G là một đồ thị có chu trình), liệt kê T_i rồi loại bỏ T_i ra khỏi G. Sau đó lặp lại quá trình trên cho đến khi đồ thị không còn nút nào nữa. Khi đó, thứ tự các nút được liệt kê là thứ tự tuần tự của các giao dịch.

Ví dụ 2.3: Giả sử ta có lịch biểu của ba giao dịch T_1, T_2, T_3 như sau

T_1 : Lock A

T_2 : Lock B

T_2 : Lock C

T_2 : Unlock B

T_1 : Lock B

T_1 : Unlock A

T_2 : Lock A

T_2 : Unlock C

T_2 : Unlock A

T_3 : Lock A

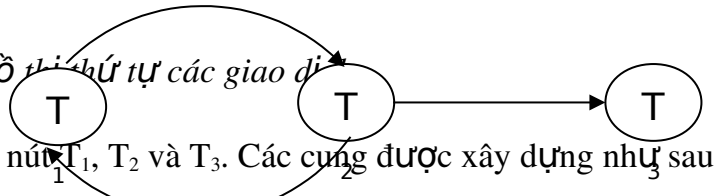
T_3 : Lock C

T_1 : Unlock B

T_3 : Unlock C

T_3 : Unlock A

Hình 2.4 Đồ thị thứ tự các giao dịch



Đồ thị có 3 nút T_1, T_2 và T_3. Các cung được xây dựng như sau:

ở bước (4) ta có T_2 : Unlock B, bước tiếp theo có lệnh Lock B là bước (5) T_1: Lock B. Vậy ta vẽ một cung từ T_2 T_1.

ở bước (6) ta có T_1 : Unlock A, bước tiếp theo có lệnh Lock A là bước (7) T_2: Lock A. Vậy ta vẽ một cung từ T_1 T_2.

ở bước (8) ta có T_2 : Unlock C, bước tiếp theo có lệnh Lock C là bước (11) T_3: Lock C. Vậy ta vẽ một cung từ T_2 T_3.

ở bước (9) ta có T_2 :Unlock A, bước tiếp theo có lệnh Lock A là bước (10) có T_3: Lock A. Vậy ta vẽ một cung từ T_2 T_3.

Đồ thị này có một chu trình nên lịch biểu đã cho bất khả tuần tự.

Ví dụ : Lịch biểu của ba giao dịch T_1, T_2, T_3

(1) T_2 : Lock A

(2) T_2 : Unlock A

(3) T_3 : Lock A

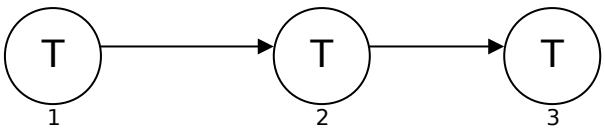
(4) T_3 : Unlock A

(5) T_1 : Lock B

(6) T_1 : Unlock B

(7) T_2 : Lock B

(8) T_2 : Unlock B



Hình 2.5. Đồ thị tuần tự cho ba giao dịch

4.3. Mô hình khoá đọc và khoá ghi

Trong mô hình khoá cơ bản, ta giả sử rằng khi khoá một mục có thể thay đổi mục đó. Trên thực tế, có những trường hợp một giao dịch truy cập một mục theo nghĩa chỉ đọc giá trị của mục đó không thay đổi giá trị của mục đó. Vì vậy nếu ta phân biệt hai loại truy cập: chỉ đọc (read only) và đọc ghi (read write), thì ta có thể tiến hành được một số thao tác đồng thời bị cấm trong mô hình khoá cơ bản. Khi đó, ta phân biệt hai loại khoá như sau:

Khoá đọc (read lock or shared lock) ký hiệu RLock hoạt động như sau: một giao dịch T chỉ muốn đọc một mục A sẽ thực hiện lệnh RLock A, ngăn không cho bất kỳ giao dịch khác ghi giá trị mới vào A trong khi T đã khoá A, nhưng các giao dịch khác vẫn có thể giữ một khoá đọc trên A cùng lúc với T.

Khoá ghi (write lock) ký hiệu WLock hoạt động như trong mô hình khoá cơ bản, nghĩa là một giao dịch muốn thay đổi giá trị của mục A sẽ thực hiện lệnh WLock A. Khi đó không một giao dịch nào được lấy khoá đọc hoặc khoá ghi trên mục đó.

Cả khoá đọc và khoá ghi đều được mở bằng lệnh Unlock. Ngoài các giả định như mô hình khoá cơ bản, ta có thêm giả định rằng một giao dịch có thể yêu cầu một khoá ghi trên mục mà nó đang giữ khoá đọc.

Hai lịch biểu là tương đương nếu:

- Chúng sinh ra cùng một giá trị cho mỗi mục, và

- Mỗi khoá đọc được áp dụng bởi một giao dịch xảy ra trong cả hai lịch biểu vào những lúc mục bị khoá có cùng giá trị.

Thuật toán 2.2: Kiểm tra tính khả tuần tự của các lịch biểu với các khoá đọc / ghi

Nhập: Một lịch biểu S cho một tập các giao dịch $T_1, T_2, ..., T_k$.

Xuất: Khẳng định S có khả tuần tự hay không? Nếu có thì đưa ra một lịch biểu tuần tự tương đương với S.

Phương pháp:

Bước 1: Chúng ta xây dựng một đồ thị có hướng G (gọi là đồ thị tuần tự hoá), có các nút là các giao dịch. Các cung của đồ thị được xác định bằng quy tắc sau:

Giả sử trong S, T_i nhận khoá đọc hoặc khoá ghi mục A, T_j là giao dịch kế tiếp khoá ghi A, và i j, thì ta sẽ đặt một cung từ T_i T_j.

Giả sử trong S, giao dịch T_i khoá ghi A, T_m là khoá đọc A sau khi T_i mở khoá A nhưng trước các giao dịch khác khoá ghi A, và i m, thì ta sẽ đặt một cung từ T_i T_m .

Bước 2: Kiểm tra, nếu G có chu trình thì S bất khả tuần tự. Nếu G không có chu trình thì một sắp xếp topo của G là thứ tự tuần tự của các giao dịch này.

Ví dụ 2.5: Một lịch biểu của bốn giao dịch :

(1) T_2 : RLock A
(2) T_3 : RLock A
(3) T_2 : WLock B
(4) T_2 : Unlock A
(5) T_3 : WLock A
(6) T_2 : Unlock B
(7) T_1 : RLock B
(8) T_3 : Unlock A
(9) T_4 : RLock B

(10) T_1 : RLock A

(11) T_4 : Unlock B

(12) T_1 : WLock C

(13) T_1 : Unlock A

(14) T_4 : WLock A

(15) T_4 : Unlock A

(16) T_1 : Unlock B

(17) T_1 : Unlock C

Đồ thị tuần tự hoá của lịch biểu này được trình bày trong hình 2.6.

Các nút là bốn giao dịch T_1 , T_2 , T_3 , T_4. Các cung được xác định như sau:

ở bước (4) T_2 mở khoá mục A

Bước (5) T_3 khoá ghi mục A, T_3 phải đi sau T_2, có một cung từ T_2 đến T_3.

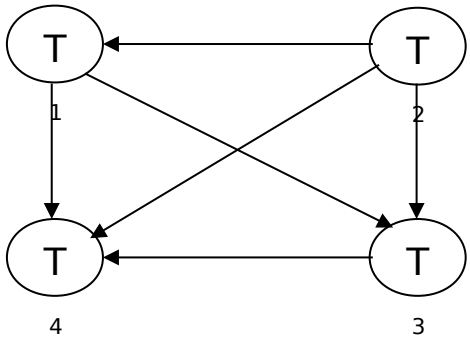
ở bước (6) T_2 mở khoá mục B

Bước (7) T_1 các khoá đọc mục B và T_4 ở bước (9). Như vậy T_1 và T_4 phải đi sau T_3, có một cung từ T_2 đến các nút này.

ở bước (8) T_3 mở khoá mục A,

Bước (10) T_1 là khoá đọc mục A và khoá ghi mục A của T_4 ở bước (14). Như vậy T_1 và T_4 phải đi sau T_3, có một cung từ T_3 đến các nút này.

ở bước (13) T_1 mở khoá mục A, bước (14) T_4 khoá ghi mục A, T_4 phải đi sau T_1, có một cung từ T_1 đến T_4.



Sắp xếp topo cho đồ thị ta được thứ tự các giao dịch là: T_1 T_2 T_3 T_4

Giao thức hai pha của mô hình trước cũng có thể áp dụng cho mô hình này. Các khoá đọc và khoá ghi đều đi trước bước mở khoá, và điều đó sẽ đảm bảo tính khả tuần tự của lịch biểu.

Trong mô hình này ta cũng rút ra được qui tắc liên quan đến việc trao khoá như sau:

Một khoá đọc trên một mục có thể được trao cho một giao dịch nếu không có khoá ghi nào đang được một giao dịch khác giữ trên nó.

Một khoá ghi trên một mục chỉ có thể được trao cho một giao dịch nếu không có khoá đọc hoặc khoá ghi nào đang được một giao dịch khác giữ trên mục đó.

4.4. Thuật toán điều khiển tương tranh bằng nhãn thời gian

Để đảm bảo tính khả tuần tự của các lịch biểu, ngoài các mô hình sử dụng khoá như đã trình bày ở trên. Ta còn sử dụng nhãn thời gian (timestamp). ý tưởng chính là gán cho mỗi giao dịch một nhãn thời gian, đó chính là điểm bắt đầu của giao dịch.

Thiết lập nhãn thời gian

Nếu tất cả các giao dịch đều được bộ lập lịch *gán nhãn thời gian* thì bộ lập lịch sẽ duy trì bộ đếm số lượng các giao dịch đã được lập lịch. Khi có một giao dịch mới yêu cầu được lập lịch, bộ lập lịch sẽ tăng bộ đếm số lượng này lên một đơn vị và gán trị số đó cho giao dịch có yêu cầu. Như vậy, không thể xảy ra trường hợp hai giao dịch có cùng nhãn thời gian, và thứ tự tương đối giữa các *nhãn thời gian* của các giao dịch cũng chính là thứ tự mà các giao dịch được thực hiện.

Một cách *gán nhãn thời gian* khác cho các giao dịch là dùng giá trị của đồng hồ hệ thống tại thời điểm bắt đầu giao dịch.

Trong trường hợp tồn tại nhiều bộ xếp lịch do hệ thống CSDL chạy trên một máy đa bộ xử lý hoặc trong hệ CSDL phân tán, thì ta phải gán thêm một hậu tố duy nhất cho mỗi nhãn thời gian. Hậu tố này chính là định danh của bộ xử lý tương ứng. Khi đó, việc đồng bộ hoá các bộ đếm hoặc đồng hồ được dùng ở mỗi bộ xử lý là một yêu cầu quan trọng để đảm bảo tính khả tuần tự của các lịch biểu.

Đảm bảo tính khả tuần tự bằng nhãn thời gian

Qui tắc duy trì thứ tự tuần tự của nhãn thời gian như sau. Giả sử ta có một giao dịch có nhãn thời gian t đang muốn thực hiện một thao tác X trên một mục có thời điểm đọc t_r và thời điểm ghi t_w thì:

a/ Cho thực hiện thao tác này nếu:

X = Read và t t_w hoặc

X = Write và t t_r và t t_w

Trong trường hợp trước, đặt thời điểm đọc là t nếu $t > t_r$ và trong trường hợp sau, đặt thời điểm ghi là t nếu $t > t_w$.

b/ Không thực hiện gì nếu X = Write và t_r t < t_w.

c/ Huỷ bỏ giao dịch này nếu: X = Read và t < t_w hoặc

X = Write và t < t_r

Ví dụ : Trong đó RLock được xem là Read, WLock được xem là Write, các bước Unlock không tồn tại. Các giao dịch T_1, T_2 , T_3, T_4 có các nhãn thời gian lần lượt là 100, 200, 300, 400.

ở bước (1) , T_2 (có nhãn thời gian là t = 200) đọc A (có WT = 0), tức t t_w. Thao tác này là được phép, và vì t > t_r (bằng 0) nên RT của A được đặt lại là 200.

Tương tự ở bước (2) , T_3 (có nhãn thời gian là t = 300) đọc A (có WT = 0), tức t t_w. Thao tác này là được phép, và vì t > t_r (bằng 200) nên RT của A được đặt lại là 300.

ở bước (3) , T_2 (có nhãn thời gian là t = 200) ghi B (có RT = 0 và WT = 0), tức t T_r và t t_w. Thao tác này là được phép, vì t > t_W (bằng 0) nên WT của B được đặt lại là 200.

ở bước (4) , T_3 (có nhãn thời gian là t = 300) ghi A (có RT = 300 và WT=0), tức t T_r và t t_w. Thao tác này là được phép, vì t > t_W (bằng 0) nên WT của A được đặt lại là 300.

ở bước (5) , T_1 (có nhãn thời gian là t = 100) đọc B (có WT = 200), tức t < t_w. Vì vậy thao tác này bị huỷ bỏ.

TT	T_1	T_2	T_3	T_4	A	B	C
	100	200	300	400	RT=0 WT=0	RT=0 WT=0	RT=0 WT=0
(1)		Read A			RT=200		
(2)			Read A		RT=300		
(3)		Write B				WT=200	
(4)			Write A		WT=300		
(5)	Read B T_1 bị huỷ bỏ						

Trong quá trình thực hiện giao dịch CSDL có thể tạm thời không nhất quán nhưng CSDL phải nhất quán khi giao dịch kết thúc. Tính tin cậy dựa vào hai khả năng sau:

+ Khả năng phục hồi nhanh của hệ thống khi nhiều kiểu lỗi xảy ra (Khi các lỗi xảy ra hệ thống có thể chịu đựng được và có thể tiếp tục cung cấp các dịch vụ).

+ Khôi phục: đạt được trạng thái nhất quán. Trở về trạng thái nhất quán trước đó hoặc tiếp tới trạng thái nhất quán mới sau khi xảy ra lỗi. Nhất quán về giao tác liên quan tới sự thực hiện các truy nhập trùng nhau.

Việc quản lý giao tác tiếp xúc với các vấn đề luôn giữ CSDL trong trạng thái nhất quán khi xảy ra các truy nhập trùng nhau và các lỗi.

PHẦN 1
CƠ SỞ DỮ LIỆU SUY DIỄN

2.1. Giới thiệu chung

- Khi niệm về CSDL suy diễn được nhiều nhà nghiờn cứu đề cập đến theo hướng phát triển cỏc kết quả mà Green đó đạt được vào năm 1969 về cỏc hệ thống cõu hỏi – trả lời.

- Xuất phát từ quan điểm lý thuyết, **các CSDL suy diễn có thể được coi như các chương trình logic với sự khái quát hoá khái niệm về CSDL quan hệ.** Đó là cách tiếp cận của Brodie và Manola vào năm 1989, của Codd vào năm 1970, của Date vào năm 1986, của Gardarin và Valdurier vào năm 1989 và của Ullman vào năm 1984.

- Lập trình logic là mảng công việc trước tiên khi chứng minh định lý cơ học. Sự thật thì việc chứng minh định lý đã tạo nên cơ sở cho hầu hết hệ thống lập trình logic hiện nay. Tư tưởng cơ bản của lập trình logic là sử dụng logic toán học như ngôn ngữ lập trình. Điều này được đề cập trong tài liệu của Kowaski năm 1970, và được Colmerauer đưa vào thực hành năm 1975 trong các cài đặt ngôn ngữ lập trình logic đầu tiên, tức là ngôn ngữ PROLOG (PROgramming LOGic). Nhờ sự hình thức hoá, Kowalski đã xem xét tập con của các logic bậc một, gọi là logic mệnh đề Horn. Một câu hay một mệnh đề theo logic có thể có nhiều điều kiện đúng nhưng chỉ có một hay không có kết luận đúng.

- Đối với nhu cầu thực hành CSDL suy diễn xử lý cỏc cừu khụng phức tạp như cỏc cừu trong hệ thống lập trỡnh logic. Số cỏc luật, tức là số cỏc cừu với cỏc điều kiện khụng trống trong CSDL suy diễn nhỏ hơn số cỏc sự kiện, tức cỏc cừu với điều kiện rỗng.

-Một khía cạnh khác nhau nữa giữa CSDL suy diễn và lập trình logic là các hệ thống lập trình logic nhấn mạnh các chức năng, trong khi CSDL suy diễn nhấn mạnh tính hiệu quả. Cơ chế suy diễn dùng trong CSDL suy diễn để tính toán trả lời không được tổng quát như trong lập trình logic.

- Ngoài việc dựng logic để diễn tả cỏc cõu CSDL, người ta cũn dựng logic để diễn tả những cõu hỏi và cỏc đIũu kiện toàn vẹn.

2.2- CSDL suy diễn

2.2.1. Mô hình CSDL suy diễn

Mô hình dữ liệu gồm:

+ Kí pháp toán học để mô tả hình thức dữ liệu và các quan hệ, và

+ Kỹ thuật để xử lý dữ liệu như trả lời các câu hỏi, kiểm tra điều kiện toàn vẹn.

Ngôn ngữ bậc một được dùng như kí pháp toán học để mô tả dữ liệu trong mô hình CSDL suy diễn và dữ liệu được xử lý trong các mô hình như vậy nhờ việc đánh giá công thức logic. Tiếp cận của logic bậc một như nền tảng lý thuyết của các hệ thống CSDL suy diễn.

Tuy nhiên để dễ biểu diễn hình thức các khái niệm về CSDL suy diễn, người ta thường dùng phép toán vị từ, tức logic vị từ bậc nhất. Logic vị từ bậc nhất là ngôn ngữ hình thức dùng để thể hiện các quan hệ giữa các đối tượng và để suy diễn ra quan hệ mới.

Định nghĩa 1: *Mỗi một hằng số, một biến số hay một hàm số áp lên các term là một hạng thức (term)*

Hàm n ngôi f(x1,x2,..,xn); xi | i = 1,2,..,n là một hạng thức thì f(x1,x2, …,xn) là một term.

Định nghĩa 2: *Công thức nguyên tố(công thức nhỏ nhất) là kết quả của việc ứng dụng một vị từ trên các tham số của term dưới dạng P(t1, t2,…, tn).*

Nếu P là vị từ có n ngôi và ti | i=1,2, ..,n là một hạng thức(term).

Định nghĩa 3: *(Literal) Dãy các công thức nguyên tố hay phủ định của công thức nguyên tố đã được phân tách qua các liên kết logic (, , , , , ,) thì công thức đó được thiết lập đúng đắn.*

(i): Một công thức nguyên tố là công thức thiết lập đúng đắn.

(ii): F, G là Công thức thiết lập đúng đắn => F G, F G, F G, F G, F , G cũng là các công thức thiết lập đúng đắn.

(iii): Nếu F là Công thức thiết lập đúng đắn, mà x là một biến tự do trong F => (x)F và (x)F cũng là các công thức thiết lập đúng đắn (x, x trong F).

Ví dụ 1: Cho quan hệ R(A1, A2,…, An) với n bậc (tức n thuộc tính) => là một vị từ n ngôi. Nếu r R (r bộ của R) => (r.A1, r.A2,…, r.An) => R(A1, A2,.., An) nhận giá trị đúng.

Nếu r R (r bộ của R) => gán (r.A1, r.A2,…, r.An) => R(A1, A2,.., An) nhận giá trị sai.

Định nghĩa 4: Câu(Clause)

Công thức có dạng P1 P2 …. Pn Q1 Q2 …. Qn

Trong đó: Pi và Qj (i,j=1,2,…,n) là các Literal dương

Trong hệ thống logic, Literal dương có dạng nguyên tố, nhỏ nhất, trái với Literal âm là phủ định của nguyên tố.

Định nghĩa 5: Câu Horn (Horn clause)

là câu có dạng P1 P2 …. Pn Q1

Định nghĩa 6: *CSDL suy diễn tổng quát* (General deductive database)

CSDL suy diễn tổng quát, hay CSDL tổng quát, hay CSDL suy diễn được xác định như cặp (D,L), trong đó D là tập hữu hạn của các câu CSDL và L là ngôn ngữ bậc một.

Giả sử L có ít nhất hai ký hiệu, một là ký hiệu hằng số và một ký kiệu vị từ.

+ Một CSDL xác định (hay CSDL chuẩn) là CSDL suy diễn(D,L) mà D chỉ chứa các câu xác định(câu chuẩn).

+ Một CSDL quan hệ là CSDL suy diễn (D,L) mà D chỉ chứa các sự kiện xác định .

Vậy CSDL quan hệ là một dạng đặc biệt của CSDL tổng quát, hay chuẩn, hay xác định. Còn một CSDL xác định là dạng đặc biệt của CSDL chuẩn hay tổng quát.

2.2.2. Lý thuyết mô hình đối với CSDL quan hệ

2.2.2.1. Nhìn nhận CSDL theo quan điểm logic

Một CSDL có thể được nhìn nhận dưới quan điểm của logic như sau:

Lý thuyết bậc một, hay

Diễn giải của lý thuyết bậc một

Theo quan điểm diễn giải, các câu hỏi và các điều kiện toàn vẹn là công thức dùng để đánh giá việc sử dụng định nghĩa ngữ nghĩa. Còn theo quan điểm lý thuyết, các câu hỏi được coi như các định lý có thể chứng minh được hay công thức hiển nhiên theo lý thuyết này.

Hai tiếp cận này được tham chiếu đến như quan điểm lý thuuyết mô hình, hay quan điểm cấu trúc quan hệ, và quan điểm lý thuyết chứng minh. Hai quan

điểm trên đã được hình thức hoá thành khái niệm tương ứng của cơ sở dữ liệu thông thường và CSDL suy diễn.

Tư tưởng đằng sau quan điểm lý thuyết chứng minh của CSDL(D,L) là

(i) Xây dựng một lý thuyết T, gọi là lý thuyết chứng minh của (D,L), bằng cách dùng các câu D và ngôn ngữ L, và

(ii) Trả lời các câu hỏi trong CSDL.

2.2.2.2. Nhìn lại CSDL quan hệ

ở đây ta xét lớp các CSDL quan hệ, tức là các sự kiện làm nền dựa trên nền của các sự kiện, với các ngôn ngữ không chứa bất kỳ kí hiệu hàm nào. Các giả thiết được đặt ra trên lớp của các CSDL quan hệ để đánh giá các câu hỏi:

1) **Giả thiết về thế giới đón***g*(CWA Close World Assumption): Khẳng định rằng các thông tin không đúng trong CSDL được coi là sai, tức là R(a1,a2,..,an) coi là đúng chỉ khi sự kiện R(a1,a2,..,an) không xuất hiện trong CSDL.

Ví dụ: Có CSDL sau: Hoc_sinh(Xuân)

Sinh_vien(Đông)

Nghiên_cưu(Đông)

Thich(Xuân, Toán)

Như vậy theo CWA thì bộ Thich (Đông, Toán) được giả sử là đúng, tức Đông không thích Toán.

2) **Giả thiết về tên duy nhất** (UNA Unique Name Assumption): Khẳng định các hằng số của các tên khác nhau được coi là khác nhau.

Theo ví dụ trên có thể nói rằng hai hằng số Xuân và Đông gán tên duy nhất cho hai sinh viên khác nhau.

3) **Giả thiết về bao đóng của miền** (DCA Domain Closure Assumption): Cho rằng không có các hằng số ngoài các hằng số trong ngôn ngữ của CSDL.

Theo ví dụ trên có thể nói rằng Triết không phải là hằng đúng.

Cho CSDL quan hệ (D,L), D có một vài hạn chế L không chứa kí hiệu hàm nào. Vậy CSDL này có thể được coi là diễn giải của lý thuyết bậc môt gồm có

ngôn ngữ L và các biến của L , như đã được sắp đặt trên miền trong diễn giải này. Việc đánh giá công thức Logic trong diễn giải này dựa trên :

R(a1,a2,…,an) đúng chỉ khi R(a1,a2,…,an) D

Các tiên đề của ngôn ngữ T: Theo quan điểm lý thuyết chứng minh của CSDL quan hệ thu được bằng cách xây dựng lý thuyết T trong ngôn ngữ L.

T1. *Xác nhận*: Đối với mỗi sự kiện R(a1,a2,…,an) D => R(a1,a2,…,an) được xác định.

T2. *Các tiên đề đầy đủ*: Với mỗi kí hiệu quan hệ R,

nếu $R(a_1^1, a_2^1, \ldots, a_n^1)$, $R(a_1^2, a_2^2, \ldots, a_n^2)$,…, $R(a_1^m, a_2^m, \ldots, a_n^m)$ kí hiệu cho các sự kiện của R thì tiên đề đầy đủ đối với R là:

x1, x2,…, xn R(a1, a2,..,an) $(x1 = a_1^1 \quad x2 = a_2^1 \quad \ldots \quad xn = a_n^1)$

$(x1 = a_1^2 \quad x2 = a_2^2 \quad \ldots \quad xn = a_n^2) \quad \ldots \quad (x1 = a_1^m \quad x2 = a_2^m \quad \ldots \quad xn = a_n^m)$

T3. *Các tiên đề về tên duy nhất*: Nếu a1, a2,.., ap là tất cả những kí hiệu hằng số của L thì

(a1 a2), (a1 a3), …., (a1 a_p), (a2 a3), (a2 a4),…, (a_{p-1} a_p)

T4. *Các tiên đề về bao đóng của miền*: Nếu a1, a2,.., ap là các kí hiệu hằng số của L thì: x((x=a1) (x=a2) …. (x=ap))

T5. *Các tiên đề tương đương*:

1. x(x=x)
2. x y((x=y) (y=x))
3. x y z ((x=y) (y=z) (x=z))
4. x1, x1,…, xn(P(x1, x2,.., xn) (x1=y1) (x2=y2) …. (xn=yn) (y1, y2,.., yn))

2.2.3. Nhìn nhận CSDL suy diễn

ở đây chỉ nhìn nhận lý thuyết chứng minh áp dụng cho CSDL suy diễn.

Ngôn ngữ L của CSDL (D, L) được xây dựng chỉ bằng các kí hiệu xuất hiện trong D, và người ta có thể dùng bất kì ngữ nghĩa thủ tục nào trong ngữ cảnh của chương trình logic như công cụ để tìm các câu trả lời bằng cách suy diễn từ lý thuyết chứng minh T, lý thuyết T đảm bảo ngữ nghĩa của D nhất trí với ngữ nghĩa của T.

Liên quan đến CSDL suy diễn, người ta đưa ra **Comp**(D) như là lý thuyết chứng minh của CSDL (D, L) và dùng cách giải SLDNF để tìm câu trả lời cho câu hỏi.

Giả sử (D, L) là CSDL chuẩn. Như trong trường hợp của CSDL quan hệ, quan điểm lý thuyết chứng minh của D đạt được bằng cách xây dựng một lý thuyết T trong ngôn ngữ L.

Các tiên đề lý thuyết của T như sau:

1) **Các tiên đề về đầy đủ**: Tiên đề có được do hoàn thiện mỗi kí hiệu vị từ của L, tương ứng với các câu trong D.

2) **Tiên đề về duy nhất của tên và về tính tương đương***:* các tiên đề về lý thuyết tương đương là tuỳ theo các kí hiệu hằng số, hàm số và vị từ của L.

3) **Tiên đề về bao đóng của miền:** Nếu a1, a2,…, ap là tất cả những phần tử của L và f1, f2,..,fq là các kí hiệu hàm số của L, thì tiên đề về bao đóng của miền, theo Lloyd năm 1987, Mancarella năm 1988 như sau:

x((x=a1) (x=ap) (x1, x2,.., xm(x = f1(x1, x2,.., xm))) … (y1, y2, …., yn(x = fq(y1, Y2,…, yn))))

2.2.4. Các giao tác trên CSDL suy diễn

Định nghĩa 1: **Giao tác (Transaction)**

Một giao tác trong CSDL suy diễn là một một xâu hữu hạn của các phép toán, hay các hành động bổ sung, loại bỏ hay cập nhật các câu.

Vì một CSDL suy diễn được xem như tập các câu, tức là theo quan điểm lý thuyết mô hình, không một phép loại bỏ hay cập nhật nào được phép thực hiện trên sự kiện. Các sự kiện là ngầm có trong CSDL.

Định nghĩa 2**: Khẳng định (Commit)**

Một giao tác được gọi là được khẳng định tốt nếu toàn bộ xâu các phép toán tạo nên kết quả tốt của giao tác.

Lý do chính của việc không đảm bảo hoàn thành tốt một giao tác là sự vi phạm điều kiện toàn vẹn khi thực hiện các phép toán trong giao tác, hay hư hỏng hệ thống, tính toán vô hạn.

2.3. CSDL dựa trên Logic

Trong phần này ta đi nghiên cứu CSDL dựa trên Logic mà cụ thể là chương trình DATALOG.

DATALOG là một ngôn ngữ phi thủ tục dựa trên logic vị từ bậc nhất.

Người ta sử dụng để mô tả thông tin cần thiết không theo cách lấy thông tin trong các thủ tục bình thường mà dựa trên logic (ngôn ngữ DATALOG).

2.3.1. Cú pháp

+ Ký hiệu :

+ vị từ so sánh : <tên thuộc tính> <giá trị>

(Biến) so sánh với (giá trị)

= {<, >, <=, >=, =, <>}

+ Cách biểu diễn các luật(Clause – Rule)

Q P1, P2, ..,Pn

Dấu "," ⇔ AND ( )

Dấu ";" ⇔ OR ( )

Dấu " " : Kéo theo

Pi : là các tiên đề, giả thiết, đích con, vị từ

Q : là kết luận hay là sự kiện

+ Nếu n = 0 : Q Các sự kiện của CSDL cài đặt.

+ Nếu P P1, P2,…,Pn thì P là luật đệ quy (hay vị từ ở trong thân và đầu luật)

2.3.2. Ngữ nghĩa

là tập tất cả các sự kiện được suy diễn ra từ chương trình DATALOG.

Ví dụ:

(r1) Chamẹ(x,y) Bố(x,y)

(r2) Chamẹ(x,y) mẹ(x,y)

(r3) Ôngbà(x,y) Chamẹ(x,z) , Chamẹ(z,y)

(r4) Bố(x,y) (r7) : TổTiên(x,y) Chamẹ(x,z) , TổTiên(z,y)

(r5) Mẹ(x,y) (r6) : TổTiên(x,y) , Chamẹ(x,y)

2.3.3. Cấu trúc cơ bản

CSDL DATALOG gồm hai loại quan hệ:

Các quan hệ cơ sở được lưu trữ trong CSDL, có dạng như người ta thấy. Người ta còn gọi cơ sở này là CSDL mở rộng EDB (Extended Database).

Các quan hệ suy diễn không cần lưu trong CSDL. Chúng được dùng như quan hệ tạm thời, chứa các kết quả trung gian khi trả lời câu hỏi. Các quan hệ này được gọi là CSDL theo mục đích IDB (Intentional Database).

Mỗi quan hệ có tên và số cột.

Khác với đại số quan hệ, các thuộc tính của mỗi quan hệ trong DATALOG không mạng tên hiện rõ. Thay vì có tên, mỗi thuốc tính căn cứ vào giá trị của nó.

Các chương trình DATALOG có một tập hữu hạn các luật tác động đến các quan hệ cơ bản và quan hệ suy diễn.

Trước khi đưa ra định nghĩa hình thức ta xét **ví dụ** sau:

+ Có luật về ngân hàng như sau:

Ca(Y,X) Gửitiền(“Hà Nội”, X, Y, Z), Z>1200

Luật này gồm quan hệ cơ sở là “Gửitiền”, quan hệ suy diễn là “Ca”. Luật này rút ra các cặp <Tên khách hàng, Tài khoản> của tất cả các khách hàng có tài khoản tại chi nhánh “Hà Nội” và có số dư lớn hơn 1200.

+ Luật trên có thể viết được dưới dạng biểu thức tính toán tương đương trên miền xác định và kết quả được bổ sung vào quan hệ suy diễn mới “Ca”

{ <X, Y> | W, Z (W, X, Y, Z) Gửitiền W= “Hà Nội” Z>1200}

Từ đó ta đi đến một số công thức sau:

1) Các luật được xây dựng trên các Literal có dạng sau:

P(A1, A2,…, An), trong đó: P là tên của quan hệ cơ sở hay quan hệ suy diễn. Mỗi Ai (i=1,2,…,n) là hằng số hay tên biến.

2) Một luật trong DATALOG có dạng:

P(X1, X2,…, Xn) Q1(X11, X12,…,X1,m1),

Q2(X21, X22,…,X2,m2),…, Qr(Xr1, Xr2,…,Xr,mr), e

Trong đó: + P là tên của quan hệ suy diễn

+ Mỗi Qi là tên của quan hệ cơ sở hay quan hệ suy diễn

+ e là biểu thức vị từ số học đối với các biến xuất hiện trong P và tất cả các Qi (mỗi biến xuất hiện trong P cũng xuất hiện trong Qi nào đó).

Literal P(X1, X2,…, Xn) được gọi là đầu của luật, phần còn lại gọi là thân của luật.

Để hiểu chính xác cách thức diễn giải một luật trong Datalog, người ta xác định khái niệm thay thế luật và hiện trạng của luật.

Định nghĩa 7: **Thay thế luật** (Rule Substitution)

Việc thay thế luật được áp dụng cho một luật là việc thay mỗi biến trong luật bằng một biến hay một hằng.

Tức là, nếu một biến xuất hiện nhiều lần trong một luật thì phải thay nó bằng cùng một biến hay cùng một hằng số.

Ví dụ: Thay thế đối với luật nêu trong ví dụ trên, biến Z được thay bằng W và các biến kia được thay bằng hằng số.

Ca("Mỗ", 123) Gươitiên("Hà Nội", 123, "Mỗ", W), W>1200

Tuy nhiên, nếu thay X bằng hằng số 123 và 333 thì không được

Ca("Mỗ", 123) Gươitiên("Hà Nội", 333, "Mỗ", W), W>1200 => sai

Định nghĩa 8: **Hiện trạng của luật** (Rule instantiation)

Hiện trạng của luật là việc thay thế hợp lệ các biến bằng các hằng số.

Một thay thế đúng cho người ta một hiện trạng của luật.

Ví dụ: Ca("Mỗ", 123) Gươitiên("Hà Nội", 123, "Mỗ", 1500), 1500>1200

Đối với luật cụ thể, có thể có nhiều hiện trạng hợp lệ.

Để xem Datalog diễn giải luật ra sao, người ta xét một hiện trạng của luật:

P(X1, X2,…, Xn) Q1(X11, X12,…,X1,m1), Q2(X21, X22,…,X2,m2),…, Qr(Xr1, Xr2,…,Xr,mr), e

P đúng nếu các biểu thức:

Q1(**C**11,C12,…,C1,m1) Q2(**C**21,C22,…,C2,m2) … Qr(**C**r1, Cr2,…,Cr,mr) e

Có giá trị đúng, Literal Qi(Ci1, Ci2,…,Ci,mi) là đúng nếu n_bộ (Ci1, Ci2, …,Ci,mi) có mặt trong quan hệ Qi

Ví dụ: Đối với luật: Ca(Y,X) Gửitiền("Hà Nội", X, Y, Z), Z>1200

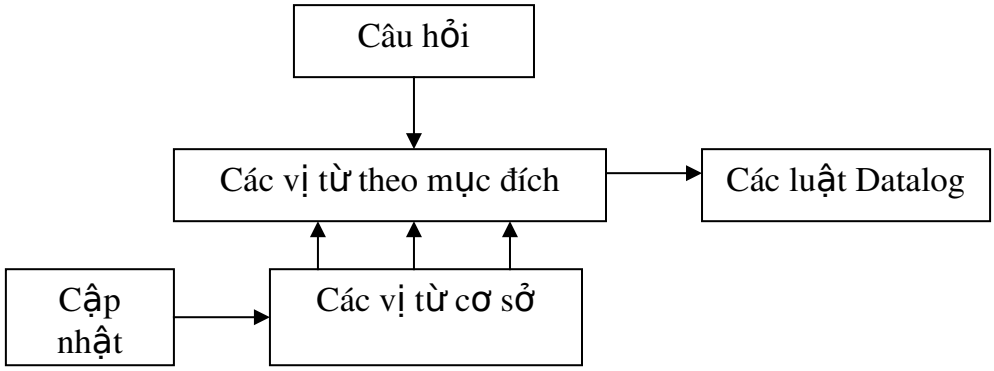
Ca(Y, X) là đúng khi có hằng số C1 thoả mãn điều kiện sau: C1>1200

n_bộ("Hà Nội", 123, "Mỗ", C1) có trong quan hệ "Gưitiên".

Định nghĩa 9: Hệ quản trị CSDL suy diễn (Deductive DBMS)

Hệ quản trị CSDL cho phép suy diễn các n_bộ của vị từ theo mục đích bằng bằng cách sử dụng các luật logic.

Các chức năng của hệ quản trị CSDL suy diễn được mô tả như sau:



CSDL suy diễn được xây dựng dựa trên các quan hệ cơ sở và quan hệ suy diễn. Hệ quản trị CSDL này được gọi là suy diễn bởi lẽ nó cho phép suy ra các thông tin từ các dữ liệu đã lưu trữ theo cơ chế suy diễn logic. Các thông tin là các vị từ theo mục đích, các thông tin này có được khi người ta tương tác với vị từ theo mục đích hoặc cập nhật vị từ cơ sở.

Định nghĩa 10: **Câu hỏi Datalog** (Datalog Query)

Một câu hỏi trong CSDL suy diễn gồm có:

Một chương trình Datalog, tức là một tập hữu hạn, có thể rỗng của các luật.

Một Literal đơn có dạng P(x1,x2,..,xn)? Trong đó xi (i=1,2,..,n) là hằng số hoặc tên biến.

Việc khai thác câu hỏi trước tiên là tính chương trình Datalog, nếu có. Tiếp theo P(x1, x2,.., xn) được đánh giá. Thủ tục này tương tự như lựa chọn trong quan hệ P theo ràng buộc phù hợp.

Ví dụ 1: Tìm tất cả các n_bộ của quan hệ vay tại chi nhánh Hà Nội.

Khi đó ta có: Vay("Hà Nội",X, Y, Z)

Câu hỏi này không có chương trình Datalog.

Ví dụ 2: Tính tập các khách hàng của chi nhánh "Hà Nội" có tài khoản mà số dư trên 1200. Chương trình Datalog chỉ có một luật đơn.

C(Y) Guitien("Hà Nội", X, Y, Z), Z>1200

C(Y)?

Câu C(Y)? là thừa; vì nó chỉ nhằm xác định quan hệ cần thể hiện. Để loại trừ hiện tượng thừa, người ta có thể dùng kí pháp ngắn gọn, nếu không sợ bị lẫn lộn, nhầm lẫn.

Nếu bấy giờ cho câu P(x1, x2,.., xn) và yêu cầu chương trình Datalog bao hàm một luật đơn phân biệt là: Hỏi(x1, x2,…,xn) …..

Trong đó xi (i=1, 2,…,n) là tên biến. Điều này hiểu rằng người ta có câu:

Hỏi(x1, x2,…,xn)?

Câu này là một phần của câu hỏi. Do vậy, câu hỏi sau là tương đương với câu hỏi trên là:

Hỏi(Y) Guitien("Hà Nội", X, Y, Z), Z>1200

2.3.4. Cấu trúc của câu hỏi

Để trình bày cấu trúc của câu hỏi người ta sử dụng đồ thị luật

Định nghĩa 11: Đồ thị luật (Rule Graph)

Một đồ thị luật đối với câu hỏi q là đồ thị có hướng mà:

Các nút của đồ thị ứng với tập các kí hiệu Literal có mặt trong các luật của q.

Cung của đồ thị ứng với quan hệ trước giữa Literal trong thân của luật và Literal có mặt trong đầu của luật đó. Do vậy đồ thị sẽ có cung

a_i a_j

Nếu luật này có mặt trong câu hỏi: a_i …a_j …

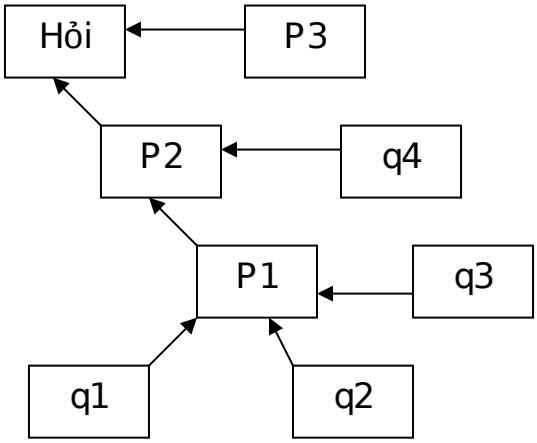
Chú ý: Việc xây dựng này không tính đến tập các biến và các hằng số có mặt trong các luật đa dạng của câu hỏi này. Thông tin duy nhất người ta dùng là tập các kí hiệu Literal và quan hệ của chúng theo các luật đa dạng.

Ví dụ 1: Xét câu hỏi:

p1(X, Y, Z) q1(X, Y), q2(X, Z), q3(Y, Z)

p2(A, B)	p1(A, B), q4(B, A)
Hỏi(B)	p2(A, B), p3(B, A)

Đồ thị ứng với câu hỏi này là:



Đồ thị trên là đồ thị không có chu trình thường được gọi là câu hỏi **không đệ quy.**

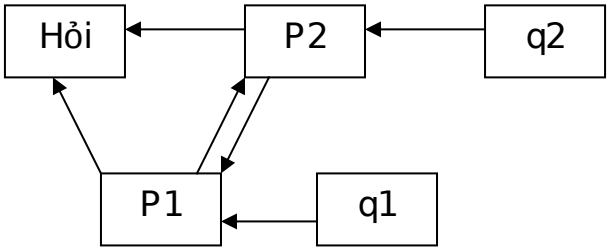
Ví dụ 2: Xét câu hỏi:

p1(A, B, C) q1(A, B), P2(B, C)

p2(X, Y) q2(X), p1(X, Y, Z)

Hỏi(A, B) p1(A, B, C), p2(B, C)

Đồ thị ứng với câu hỏi này là:



Đồ thị này là đồ thị có chu trình thường được gọi là câu hỏi **đệ quy.**

Kết luận: + Việc xây dựng cấu trúc của câu hỏi cho phép chúng ta dễ dàng trong việc đánh giá câu hỏi.

+ Giữa câu hỏi đệ quy và câu hỏi không đệ quy cũng có nhiều khác nhau ở khía cạnh loại hình câu hỏi trên CSDL. Thực tế cho thấy việc đánh giá câu hỏi đệ quy phức tạp hơn đánh giá câu hỏi thường.

2.3.5. So sánh DATALOG với đại số quan hệ

Về mặt cơ bản ngôn ngữ Datalog với các câu hỏi không đệ quy được xem như tương đương với đại số quan hệ về khả năng thể hiện.

Với các câu hỏi đệ quy cho phép người ta một công cụ mạnh hơn các ngôn ngữ quan quan hệ. Điều này ngôn ngữ Datalog cho phép hỏi các câu hỏi không được phép trong đại số quan hệ.

(1) ***Phép hợp*** : là tập các luật có cùng đầu luật

Hỏi(X1, X2,…,Xn) r1(X1, X2,…,Xn)

Hỏi(Y1, Y2,…,Yn) r2(Y1, Y2,…,Yn)

Ví dụ 1: (r1) Chamẹ(x,y) Bố(x,y)

(r2) Chamẹ(x,y) mẹ(x,y)

Ví dụ 2:Tìm tên của các khách hàng tại chi nhánh "Hà Nội", làm như sau:

Hỏi(Y) Vay("Hà Nội", X, Y, Z)

Hỏi(B) Gưi tiền("Hà Nội", A, B, C)

Chú ý: hai luật thể hiện phép hợp là tách biệt

(2) ***Phép chọn*** : ứng với một luật mà thân luật có một vị từ so sánh -> biểu thức chọn.

Phép chọn chọn các n_bộ trong quan hệ r được viết dưới dạng câu hỏi:

r(x1, x2,…, xn)?

Trong đó: xi (i=1, 2,..,n) là tên biến hay một hằng số.

Ví dụ 1: Chamẹ(x,y) Chamẹ(x,y) , y= Dũng

điều này $_{y = \text{'Dũng'}}$(Chamẹ(x,y)) (phép chọn với điều kiện là y= 'Dũng')

Ví dụ 2: Chọn (tìm kiếm) tên của những khách hàng vay quá 1000?

Hỏi(Y) Vay("Hà Nội", X, Y, Z), Z >1000

(3) ***Phép chiếu*** : là phép toán ứng với một số luật mà có một số biến ở thân luật mà không xuất hiện trong đầu luật.

Cha(x) = KQ(x) Chamẹ(x,y) , y = Dũng

(4) ***Phép kết nối*** : là phép ứng với luật mà có biến chung ở các vị từ của thân luật.

Phép kết nối hai quan hệ r1 và r2 được viết dưới dạng Datalog như sau:

Hỏi(X1, X2,…,Xn, Y1, Y2,.., Ym) r1(X1, X2,…,Xn), r2(Y1, Y2,.., Ym)

Trong đó: Xi, Yj | i=1,2,..,n và j=1,2,..,m là các tên biến phân biệt nhau.

Ví dụ 1: (r3) Ôngbà(x,y) Chamẹ(x,z) , Chamẹ(z,y)

(5) ***Khả năng đệ quy***:

Ví dụ 1: như (r7)

(r4) Bố(x,y) (r7) : TổTiên(x,y) Chamẹ(x,z) , TổTiên(z,y)

Ví dụ 2: Giả sử có lược đồ quan hệ:

Quản lý(Tên nhân công, tên người quản lý)

Lược đồ thể hiện mối quan hệ người quản lý và nhân công.

Giả sử "Quản lý" là một quan hệ theo mô hình trên.

Tên nhân công	Tên người quản lý
Mỗ	Mễ
Hoa	Mỗ
Mai	Mỗ
Lan	Mỗ
Chén	Hoa
Tích	Hoa

Yêu cầu:1) Tìm tên của những người làm việc trực tiếp dưới quyền của ông

Mỗ, tức phụ thuộc mức 1, viết như sau:

Hỏi(X) Quản lý(X, "Mỗ")

2) Để tìm tên của những người làm việc trực tiếp dưới quyền của

người do ông Mỗ quản lý, tức phục thuộc mức 2 vào ông Mỗ, Viết như sau:

Hỏi(X) Quản lý(X, Y), Quản lý(Y, "Mỗ")

Như vậy, người ta không thể thể hiện yêu cầu tìm người phụ thuộc bậc n vào

ông Mỗ trong đại số quan hệ được. Dĩ nhiên câu hỏi tìm tên của nhân công làm việc dưới quyền của ông Mỗ, trực tiếp hay gián tiếp, không thể tạo được bằng đại số quan hệ hay bằng Datalog với các câu hỏi không đệ quy. Nguyên nhân là do người ta không biết ông Mỗ quản lý đến mức nào. Tuy nhiên người có thể tạo câu hỏi này trong Datalog dưới dạng **câu hỏi đệ quy** như sau:

e(X) Quản lý(X, "Mỗ")

e(X) Quản lý(X, Y), e(Y)

Hỏi(X) e(X)

Chú ý: a)

Cách 1: Đối với những câu hỏi đệ quy người ta cũng có thể chuyển về câu hỏi không đệ quy bằng cách sử dụng ngôn ngữ tựa Pascal với một số lần hữu hạn các bước lặp. Việc lặp được thể hiện qua câu lệnh **Repeat.** Điều kiện trong câu **Until** sẽ kiểm tra về tập hợp, như tính bằng nhau, bao nhau hay rỗng. Trong câu **Until** các quan hệ suy diễn được coi như các tập. Do vậy câu hỏi đệ quy trên có thể được viết lại như sau:

e'(X) Quản lý(X, "Mỗ")

Repeat

e(X) e'(X)

e'(X) Quản lý(X, Y), e(Y)

Until e = e'

Mô tả:

- Luật đầu tiên tìm nhân công mà ông Mỗ trực tiếp quản lý. Khi hoàn thành các luật trong vòng **Repeat** được đánh giá.

- Tại mỗi lần lặp, mức tiếp theo của nhân công được tìm và được bổ sung vào tập **e.**

- Thủ tục này kết thúc khi tập e = e' (Khi không còn nhân công mới có thể được bổ sung vào **e**.). Mặt khác, do tập những người quản lý là hữu hạn.

Cách thực hiện: Theo dõi chu trình với các dữ liệu trong bảng khi chạy.

e' = {Hoa, Lan, Mai}

e = {Hoa, Lan, Mai}

e' = {Hoa, Lan, Mai, Chén, Tích}

e = {Hoa, Lan, Mai, Chén, Tích}

Cách 2: Ngoài cách làm như trên người ta có thể có cách làm khác mà vẫn đạt được kết quả như trên:

m(X, Y) Quản lý(X, Y)

m(X, Y) Quản lý(X, Z), m(Z, Y)

Hỏi(X) m(X, "Mỗ")

So sánh giữa **cách 1** và **cách 2:**

Cách 1: Tìm ra các nhân công của ông Mỗ. Cách này cho phép tìm nhanh hơn.

Cách 2: Tìm tất cả quan hệ nhân công – người quản lý rồi chọn ra các cặp có tên người quản lý là Mỗ

b) Khác với câu hỏi không đệ quy, người ta có nhiều chiến lược đánh giá câu hỏi đệ quy như chiến lược đánh giá từ dưới – lên.

Để đánh giá câu hỏi đệ quy **e** được gọi là **đánh giá thô.** Tuy nó đơn giản những không mấy hiệu qủa trong số các chiến lược dưới – lên. Sự không hiệu quả là do khi người ta sử dụng luật đệ quy, tập **e** trước đó đã được sử dụng trong tính toán. Để hiệu quả hơn, người ta dùng **đánh giá nửa thô**. Dưới đây chỉ các nhân công vừa được bổ sung trong lần lặp trước mới được luật xét đến.

Cách 11:

i:=0

e_i (X) Quản lý(X, "Mỗ")

Repeat

e (X) e_i (X)

e_{i+1}(X) Quản lý(X, Y), e_i (Y)

i: = i +1

Until e_i e

Cách 21:

i:=0

m_i (X, Y) Quản lý(X, Y)

Repeat

m (X, Y) m_i (X, Y)

m_{i+1}(X, Y) Quản lý(X, Z), m_i (Z, Y)

i: = i +1

Until m_i m

Hỏi(X) Quản lý(X, "Mỗ")

Lưu ý: Dù đã có phương pháp đánh giá tốt hơn đánh giá thô, người ta vẫn không đạt được hiệu quả như trong câu hỏi cho cùng kết quả trước đó. Cũng có nhiều kĩ thuật đảm bảo làm tinh kĩ thuật nửa thô.

2.3.6. Các hệ CSDL chuyên gia

Qua phần trên, người ta thấy rằng các luật dựa trên logic có thể tích hợp được vào CSDL quan hệ. Các luật như vậy bắt đầu từ các sự kiện trong các n_bộ của các bảng quan hệ. Các hệ chuyên gia dùng ý này để thực hiện hơn nữa các hoạt động có điều khiển.

Định nghĩa: **Hệ thống CSDL chuyên gia**(Expert Database System)

Một hệ thống CSDL chuyên gia bao gồm các luật có dạng "nếu có tập các n_bộ nào đó trong CSDL, thì một thủ tục đặc biệt được khai thác".

Thủ tục này có thể cập nhật CSDL; và câu lệnh **IF** của các luật khác có thể đúng và thủ tục khác được thực hiện… Như vậy CSDL loại này gọi là **CSDL năng động**.

Cấu trúc của hệ thống CSDL chuyên gia tương tự như cấu trúc của hệ chuyên gia trong trí tuệ nhân tạo. Khác nhau chính giữa hai loại hình này là việc sử dụng CSDL hoặc sử dụng bộ nhớ trong, hay bộ nhớ ảo.

Theo dạng chuẩn, một hệ thống CSDL chuyên gia gồm CSDL chuẩn và hệ chuyên gia chuẩn. Hệ chuyên gia hỏi bằng ngôn ngữ của CSDL, chẳng hạn như ngôn ngữ SQL và đợi trả lời từ phía CSDL.

2.4. Một số vấn đề khác

Ngoài cách tiếp cận về CSDL suy diễn như trên, người ta còn quan tâm đến một số vấn đề về CSDL suy diễn sau:

- **Thứ nhất** là: những đặc trưng của quá trình xử lý câu hỏi. Cần thiết mô tả chi tiết hơn về lựa chọn các chiến lược đánh giá câu hỏi đối với CSDL xác định và các đích xác định. Mặt khác việc xử lý câu hỏi trong môi trường song song cũng được quan tâm.

- **Thứ hai** là: các nghiên cứu hệ thống về các khía cạnh của điều kiện toàn vẹn. Cần có sự phân loại chi tiết tuỳ theo bản chất của ràng buộc, cách thể hiện của ràng buộc trong công thức logic, và các quan điểm khác nhau về thoả mãn và về kiểm tra toàn vẹn trong CSDL suy diễn. Bên cạnh đó cần có các phương pháp quản lý điều kiện toàn vẹn trong CSDL suy diễn.

- **Thứ ba** là: mẫu hình của hệ thống CSDL suy diễn. Đó là một số kiến trúc có thể chấp nhận được đối với hệ thống CSDL suy diễn. Khi đã chấp nhận một số kiến trúc nào đó, CSDL suy diễn mẫu sẽ được phát triển trước khi dùng bộ diễn giải Prolog.

- **Thứ tư** là: các CSDL suy diễn song song. Việc giới thiệu một vài kiến trúc song song của CSDL suy diễn gồm các thuật toán mô tả chi tiết quá trình xử lý câu hỏi. Các câu hỏi được coi là xác định và CSDL suy diễn được xác định tách biệt, tự do về chức năng. Việc đánh giá song song đối với các điều kiện toàn vẹn cũng là quan trọng.

- **Thứ năm** là: việc hình thức hoá các chức năng gộp lớn và các dữ liệu toàn vẹn. Trong các phần trước điều kiện toàn vẹn chỉ là tĩnh và không gộp lớn, dùng cho CSDL chuẩn. Khi phát tiển CSDL, các điều kiện toàn vẹn cũng được làm phù hợp. Người ta hình thức hoá các chức năng gộp lớn, các điều kiện toàn vẹn và các ràng buộc trên giao tác.