**TRƯỜNG ĐẠI HỌC SÀI GÒN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**Tiểu luận học phần: Cơ sở dữ liệu nâng cao**

NGÔN NGỮ ĐẠI SỐ QUAN HỆ NÂNG CAO

**Giảng viên hướng dẫn: Nguyễn Hòa**

**Sinh viên thực hiện:**

**NguyễnVăn Cường\_3123411045**

**Đặng Thành Sơn\_ 3123411258**

**Nguyễn Sĩ Huy\_ 3123411122**

**Trương Văn Tuấn\_3123411328**

Thành phố Hồ Chí Minh, tháng 9 năm 2025

# LỜI CAM ĐOAN

Chúng tôi là nhóm 2 , sinh viên lớp DCT123C6, xin cam đoan: Tiểu luận “Ngôn ngữ đại số quan hệ nâng cao” là quá trình nghiên cứu của riêng nhóm tôi dưới sự hướng dẫn của thầy Nguyễn Hòa. Các tài liệu, dữ liệu và thông tin tham khảo trong tiểu luận đều có nguồn gốc rõ ràng và được trích dẫn đầy đủ theo quy định. Chúng tôi không sao chép sử dụng bài làm của người khác một cách trái phép. Nếu vi phạm tôi xin chấp nhận mọi hình thức kỷ luật.

Thành phố Hồ Chí Minh, ngày 29 tháng 9 năm 2025

Nhóm thực hiện

Nguyễn Văn Cường\_3123411045

Đặng Thành Sơn\_ 3123411258

Nguyễn Sĩ Huy\_ 3123411122

Trương Văn Tuấn\_3123411328

# MỤC LỤC

[LỜI CAM ĐOAN i](#_bookmark0)

[MỤC LỤC](#_bookmark1) ii

[DANH MỤC CÁC HÌNH VẼ iii](#_bookmark2)

[DANH MỤC CÁC BẢNG… iv](#_TOC_250001)

[LỜI MỞ ĐẦU 1](#_bookmark3)

[CHƯƠNG 1. TỔNG QUAN VỀ HỆ ĐIỀU HÀNH DI ĐỘNG 3](#_bookmark4)

* 1. [Khái niệm về hệ điều hành di động 3](#_bookmark5)
     1. [Vai trò của hệ điều hành trong điện thoại di động 4](#_bookmark6)
     2. [Lịch sử phát triển của hệ điều hành di động 5](#_bookmark7)
  2. [Thị trường và sự phát triển của các hệ điều hành hiện nay 8](#_bookmark8)
  3. [Các yếu tố ảnh hưởng đến việc lựa chọn hệ điều hành của người dùng: 10](#_bookmark9)
  4. [Tóm tắt chương 1 13](#_bookmark10)

[CHƯƠNG 2. PHÂN TÍCH CÁC HỆ ĐIỀU HÀNH DI ĐỘNG PHỔ BIẾN 14](#_bookmark11)

* 1. [Android 14](#_bookmark12)
     1. [Tổng quan về Android 14](#_bookmark13)
     2. [Ưu điểm và nhược điểm của Android: 16](#_bookmark14)
  2. [IOS 17](#_bookmark15)
     1. [Tổng quan về IOS 17](#_bookmark16)
     2. [Ưu điểm và nhược điểm của iOS 19](#_bookmark17)
  3. [BlackBerryOS 22](#_bookmark18)
     1. [Tổng quan về BlackBerryOS 22](#_bookmark19)
     2. [Ưu điểm và nhược điểm của BlackBerryOS. 24](#_bookmark20)
  4. [Windows Phone 25](#_bookmark21)
     1. [Tổng quan về Windows Phone 25](#_bookmark22)
     2. Ưu và nhược điểm của Windows Phone… 26
  5. [Tóm tắt chương 2 27](#_bookmark23)

[CHƯƠNG 3. SO SÁNH VÀ XU HƯƠNG PHÁT TRIỂN CỦA CÁC HỆ ĐIỀU](#_bookmark24) [HÀNH DI ĐỘNG 29](#_bookmark24)

* 1. [So sánh về khả năng tùy chỉnh, bảo mật và hiệu năng giữa các hệ điều hành 29](#_bookmark25)
     1. [Khả năng tùy chỉnh 29](#_bookmark26)
     2. [Bảo mật 30](#_bookmark27)
     3. [Hiệu năng 31](#_bookmark28)
  2. [Thị phần của các hệ điều hành trong tương lai 33](#_bookmark29)
     1. [Android : thống trị thị trường 33](#_bookmark30)
     2. [iOS: Sự bền bỉ trong phân khúc cao cấp 34](#_TOC_250000)
     3. BlackBerry: Từ Đỉnh Cao đến Sự Chuyển Hướng 35
     4. Window Phone : sự sụp đổ 36
  3. [Xu hướng phát triển của hệ điều hành di động và tác động của chúng đến thị](#_bookmark32)

[trường. 37](#_bookmark32)

* 1. [Tóm tắt chương 3 38](#_bookmark31)

[KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 39](#_bookmark33)

[TÀI LIỆU THAM KHẢO 41](#_bookmark34)

**DANH MỤC CÁC HÌNH VẼ**

# DANH MỤC CÁC BẢNG

# LỜI MỞ ĐẦU

# CHƯƠNG 1. GIỚI THIỆU VÀ CƠ SỞ LÝ THUYẾT

## Bối cảnh và tầm quan trọng của Hệ Quản trị cơ Sở Dữ liệu

Dữ liệu hiện nay được xem là một trong những tài sản quý giá nhất của các tổ chức và doanh nghiệp Tốc độ phát triển nhanh của công nghệ đã tạo ra lượng dữ liệu khổng lồ cần được quản lý một cách hiệu quả để đáp ứng kịp thời các nhu cầu kinh doanh và ra quyết định. Các phương pháp lưu trữ dữ liệu truyền thống, như lưu trữ file riêng biệt, đã không còn phù hợp do gặp nhiều nhược điểm như dư thừa dữ liệu và thiếu tính nhất quán [1]. Fundamentals…

Hệ quản trị cơ sở dữ liệu (DBMS) xuất hiện nhằm khắc phục những hạn chế này bằng cách cung cấp một môi trường lưu trữ dữ liệu tập trung với các chức năng quản lý dữ liệu toàn diện. DBMS cho phép người dùng truy cập dữ liệu một cách linh hoạt thông qua các ngôn ngữ truy vấn, đồng thời đảm bảo tính bảo mật, tính toàn vẹn và khả năng xử lý đồng thời nhiều giao dịch. [3] Database System

Một ưu điểm quan trọng của DBMS là khả năng giảm thiểu sự trùng lặp dữ liệu và cung cấp môi trường đảm bảo an toàn, nhất quán cho dữ liệu trong môi trường đa người dùng. Điều này giúp tăng khả năng chia sẻ thông tin giữa các phòng ban, nâng cao hiệu quả và tính chính xác trong các hoạt động kinh doanh [2] Database System Concepts

Ngoài ra, DBMS còn là nền tảng thiết yếu cho sự phát triển của các ứng dụng công nghệ hiện đại như điện toán đám mây, trí tuệ nhân tạo và phân tích dữ liệu lớn. Nhờ vào khả năng quản lý dữ liệu hiệu quả, các tổ chức được trang bị để đáp ứng những thách thức của kỷ nguyên số và thúc đẩy quá trình chuyển đổi số một cách bền vững.[3]

Tổng kết lại, DBMS không chỉ là một phần mềm lưu trữ mà còn là công cụ chiến lược giúp các tổ chức quản lý tốt hơn tài nguyên dữ liệu của mình, từ đó hỗ trợ mạnh mẽ cho sự phát triển và đổi mới trong bối cảnh công nghệ ngày càng phức tạp. [1]

## Giới thiệu về Mô hình quan hệ và Ngôn ngữ Truy vấn

Mô hình quan hệ, do E.F. Codd đề xuất năm 1970, là mô hình dữ liệu được sử dụng rộng rãi nhất trong các hệ quản trị cơ sở dữ liệu hiện đại. Theo mô hình này, dữ liệu được tổ chức dưới dạng các quan hệ (Relation) đại diện cho bảng dữ liệu, với các bộ (Tuple) tương ứng hàng dữ liệu và các thuộc tính (Attribute) tương tự như cột trong bảng.[1]

Một quan hệ được mô tả qua một schema bao gồm tên quan hệ và tập các thuộc tính, cùng với miền giá trị (domain) của từng thuộc tính. Một tuple trong quan hệ là một bộ giá trị thuộc về các miền tương ứng, đại diện cho một bản ghi duy nhất trong bảng. Các thuộc tính phải mang giá trị nguyên tử, không được chứ các tập con hoặc các tập giá trị phức tạp, đảm bảo tuân thủ nguyên tắc dạng bình thường thứ nhất (1NF).[3]

Trong hệ quản trị cơ sở dữ liệu, người dùng thường sử dụng các ngôn ngữ truy vấn để tương tác với dữ liệu. Các ngôn ngữ này được chia làm hai nhóm chính: ngôn ngữ hình thức và ngôn ngữ thương mại.

*Ngôn ngữ hình thức*

Ngôn ngữ hình thức gồm hai loại chính là Đại số Quan hệ (Relational Algebra) và Giải tích Quan hệ (Relational Calculus). Đại số quan hệ là ngôn ngữ thủ tục, định nghĩa các phép toán trên tập quan hệ như chọn (select), chiếu (project), hợp (union), hiệu (difference), và phép kết nối (join) để tạo ra kết quả. Giải tích quan hệ là ngôn ngữ phi thủ tục, biểu diễn các truy vấn dưới dạng các biểu thức logic, tập trung vào mô tả "liệu một tuple có phù hợp hay không" mà không cần chỉ định cách thực thi.

*Ngôn ngữ thương mại*

SQL (Structured Query Language) là ngôn ngữ truy vấn tiêu chuẩn và phổ biến nhất được sử dụng rộng rãi trong các hệ quản trị cơ sở dữ liệu thương mại dựa trên mô hình quan hệ. SQL kết hợp các đặc điểm của ngôn ngữ hình thức với cú pháp thân thiện người dùng, cho phép thực hiện các truy vấn, cập nhật dữ liệu, quản lý ràng buộc và nhiều thao tác phức tạp khác trên cơ sở dữ liệu một cách hiệu quả. [2]

SQL là ngôn ngữ khai báo, người dùng chỉ cần xác định kết quả mong muốn mà không cần mô tả chi tiết cách lấy dữ liệu ra sao. Điều này giúp cho việc viết và tối ưu truy vấn trở nên thuận tiện và thực tế hơn so với việc dùng trực tiếp các phép toán của đại số hay giải tích quan hệ.

Như vậy, mô hình quan hệ kết hợp với các ngôn ngữ truy vấn hình thức và thương mại tạo thành nền tảng lý thuyết và công cụ thực tiễn cho việc khai thác và quản lý dữ liệu trong hệ quản trị cơ sở dữ liệu ngày nay.

## Đại số quan hệ : Vai trò và đặc điểm

Đại số quan hệ (Relational Algebra - RA) là một ngôn ngữ truy vấn thủ tục được phát triển dựa trên mô hình quan hệ, mà trong đó các truy vấn được biểu diễn thông qua các phép toán cơ bản như chọn (select), chiếu (project), hợp (union), giao (intersection), kết nối (join) và khác. Đây là một ngôn ngữ trung gian có tính toán học cao, cho phép định nghĩa rõ ràng các phép biến đổi trên dữ liệu quan hệ[1]

Vai trò của đại số quan hệ được xem như nền tảng lý thuyết vững chắc cho sự phát triển của các ngôn ngữ truy vấn thương mại, trong đó SQL là điển hình nhất. SQL mặc dù là ngôn ngữ khai báo phi thủ tục, nhưng trong bên trong, các truy vấn SQL được dịch sang các biểu thức đại số quan hệ để thực thi. Nhờ đó, đại số quan hệ cung cấp cơ sở để nghiên cứu, phân tích và tối ưu hóa truy vấn hiệu quả trong các hệ quản trị cơ sở dữ liệu.[3]

Tính quan trọng của đại số quan hệ còn thể hiện qua khả năng hỗ trợ tối ưu hóa truy vấn tự động, giúp hệ thống lựa chọn các chiến lược truy vấn tốt nhất, giảm thiểu chi phí truy cập và xử lý dữ liệu. Việc hiểu và vận dụng thành thạo đại số quan hệ giúp các nhà nghiên cứu và phát triển hệ thống xây dựng các trình tối ưu truy vấn mạnh mẽ, từ đó nâng cao hiệu suất vận hành của hệ thống cơ sở dữ liệu.[2]

Tóm lại, đại số quan hệ không chỉ đơn thuần là một công cụ để mô tả truy vấn mà còn là cấu nối quan trọng giữa lý thuyết mô hình quan hệ và thực tiễn ứng dụng ngôn ngữ truy vấn trong các hệ thống cơ sở dữ liệu hiện đại

## Tóm tắt chương 1

Chương 1 đã giới thiệu tổng quan về cơ sở dữ liệu và hệ quản trị cơ sở dữ liệu (DBMS), nhấn mạnh tầm quan trọng của DBMS trong việc quản lý, lưu trữ và truy xuất dữ liệu một cách hiệu quả trong kỷ nguyên số hiện nay. DBMS giúp khắc phục các hạn chế của phương pháp lưu trữ file truyền thống như dư thừa dữ liệu, thiếu nhất quán và bảo mật kém, đồng thời tạo nền tảng vững chắc cho các hệ thống công nghệ thông tin hiện đại.

Tiếp theo, chương trình bày mô hình quan hệ như một mô hình dữ liệu chủ đạo, với các thành phần cơ bản như relation (quan hệ), tuple (bộ), attribute (thuộc tính) và schema (sơ đồ quan hệ). Mô hình này dựa trên lý thuyết tập hợp và logic toán học để tổ chức dữ liệu thành các bảng có cấu trúc rõ ràng, giúp dễ dàng khai thác và bảo trì dữ liệu.

Cuối cùng, đại số quan hệ được giới thiệu như một ngôn ngữ truy vấn thủ tục với các phép toán cơ bản, đóng vai trò là nền tảng lý thuyết cho việc xây dựng và tối ưu hóa các truy vấn trong SQL và các hệ quản trị cơ sở dữ liệu quan hệ. Việc hiểu và ứng dụng đại số quan hệ giúp tăng cường hiệu năng truy vấn và cải thiện khả năng xử lý dữ liệu tổng thể của hệ thống.

# CHƯƠNG 2. CÁC TOÁN TỬ CƠ BẢN CỦA ĐẠI SỐ QUAN HỆ

## 2.1. Nhóm toán tử tập hợp

### 2.1.1. Phép hợp

#### a. Định nghĩa và khái niệm

Trong lĩnh vực cơ sở dữ liệu, toán tử tập hợp giữ vai trò trung tâm trong việc kết nối và tổng hợp dữ liệu từ nhiều nguồn khách nhau. Trong số ấy, phép hợp (union) không chỉ là một thao tác toán học đơn thuần mà còn là công cụ quan trọng giúp ta tháo gỡ những thách thức về xử lý thông tin đa dạng và phức tạp ngày nay. Phép hợp, nếu được vận dụng đúng mực, không những tối ưu hóa quy trình truy vấn mà còn đảm bảo tính nhất quán và toàn vẹn cho tập dữ liệu cuối cùng.

**Nguồn:** [1] Ramez Elmasri, Shamkant B. Navathe, Fundamentals of Database Systems, 6th Edition, Addison-Wesley, 2011.

Phép hợp giữa hai quan hệ R và S , ký hiệu là R∪S , được định nghĩa là quan hệ chứa tất cả các bộ (record) thuộc về R, hoặc thuộc về SS, hoặc thuộc cả hai. Một điểm mấu chốt cần nhấn mạnh là phép hợp tự động loại bỏ các bộ trùng lặp, nghĩa là mỗi bộ dữ liệu chỉ xuất hiện một lần duy nhất trong kết quả.

Tuy nhiên, phép hợp không phải phép toán cho phép áp dụng lỏng lẻo; nó đòi hỏi tính khả hợp của hai quan hệ. Cụ thể, hai quan hệ phải có cùng số lượng thuộc tính và các thuộc tính tương ứng cần có kiểu dữ liệu tương thích. Đây là điều kiện tiên quyết để đảm bảo truy vấn được thực thi chính xác và kết quả có ý nghĩa thực tiễn.

#### b. Biểu diễn bằng các toán tử cơ bản

Phép hợp (union) là một trong những phép toán tập hợp cơ bản, thường xuyên được sử dụng trong đại số quan hệ để phục vụ cho các bài toán cần tổng hợp dữ liệu từ nhiều nguồn khác nhau về một lược đồ chung. Khi xét hai quan hệ  và  trên cùng một lược đồ thuộc tính , phép hợp – ký hiệu  – tạo ra một quan hệ mới bao gồm tất cả các bộ (tuple) xuất hiện trong , trong , hoặc trong cả hai.

Công thức tổng quát của phép hợp được trình bày như sau:

Về bản chất, phép hợp thực hiện việc tổng hợp toàn bộ dữ liệu có thể, đồng thời đảm bảo **loại bỏ các bộ trùng lặp** – nghĩa là mỗi bộ chỉ xuất hiện một lần trong kết quả, bất kể bộ đó xuất hiện ở đâu trong hai tập nguồn dữ liệu. Đây là đặc điểm giúp đảm bảo tính thống nhất và không dư thừa dữ liệu của phép toán này.

Để phép hợp được thực hiện, hai quan hệ phải khả hợp (union-compatible). Điều kiện khả hợp yêu cầu hai quan hệ phải có cùng số lượng thuộc tính và mỗi cặp thuộc tính tương ứng phải có cùng miền giá trị và cùng ý nghĩa. Nếu điều kiện này bị vi phạm, kết quả phép hợp sẽ không có ý nghĩa trong ngữ cảnh dữ liệu quan hệ.​

**Vai trò thực tiễn**

Phép hợp thường được vận dụng trong thực tế để tổng hợp dữ liệu thống kê, lập báo cáo kết quả từ nhiều giai đoạn, hoặc xây dựng danh sách tổng hợp từ nhiều nguồn có cấu trúc dữ liệu giống nhau. Đặc biệt, phép hợp là giải pháp lý tưởng khi cần thiết lập bộ dữ liệu đầy đủ mà không bị giới hạn trong một quan hệ duy nhất, nhưng vẫn giữ được tính nhất quán thông tin.

#### c. Ví dụ minh họa

**Hợp danh sách nhân viên từ hai phòng ban**

Giả sử quan hệ employee(emp\_id, dept\_id, name) lưu thông tin nhân viên các phòng ban.

|  |  |  |
| --- | --- | --- |
| emp\_id | dept\_id | Name |
| 1 | D01 | Tuan |
| 2 | D02 | Son |
| 3 | D03 | Cuong |
| 4 | D04 | Huy |

**Yêu cầu:** Tìm tất cả nhân viên thuộc phòng ban D01 hoặc D02.

**Giải:**

Danh sách nhân viên phòng D01:

Danh sách nhân viên phòng D02:

Hợp hai danh sách này:

Kết quả: Bao gồm nhân viên có dept\_id là D01 và D02, từng nhân viên xuất hiện một lần.

### 2.1.2. Phép hiệu

#### a. Định nghĩa và khái niệm

Trong hệ quản trị cơ sở dữ liệu, toán tử tập hợp không chỉ là công cụ kỹ thuật mà còn phản ánh tư duy phân tích của người vận hành. Trong số các toán tử ấy, phép hiệu (set difference) – ký hiệu chuẩn là dấu gạch ngang – nổi bật như một phương thức cực kỳ hữu dụng để lọc bỏ hoặc loại trừ thông tin không cần thiết.

Phép hiệu của hai quan hệ R và S, ký hiệu R - S, tạo ra một quan hệ mới chỉ chứa các bộ xuất hiện trong R nhưng hoàn toàn vắng mặt trong S. Chú ý: thứ tự toán hạng có tính quyết định. Nghĩa là R - S sẽ khác hoàn toàn S - R. Nếu đảo ngược, kết quả có thể thay đổi hoàn toàn về ý nghĩa lẫn nội dung.

Hơn nữa, toán tử - yêu cầu hai quan hệ phải khả hợp, tức có cùng số lượng thuộc tính và kiểu dữ liệu tương ứng cho mỗi cột – đảm bảo thao tác kiểm tra loại trừ được thực thi một cách chính xác và không phát sinh lỗi logic hoặc kỹ thuật.

#### b. Biểu diễn bằng các toán tử cơ bản

Bên cạnh phép giao, phép trừ (set difference) là một toán tử không thể thiếu, biểu hiện năng lực loại bỏ, sàng lọc hoặc xác định phần không giao nhau giữa các quan hệ. Được ký hiệu là , phép trừ hai quan hệ khả hợp  và  trên cùng một lược đồ U trả về một quan hệ gồm tất cả các bộ của  mà không xuất hiện trong :

Việc sử dụng phép trừ thường xuất hiện trong các bài toán kiểm tra số liệu không trùng lặp, như xác định danh sách sinh viên đã đăng ký môn học nhưng chưa nộp học phí, hay lọc các hóa đơn chưa được thanh toán trong hệ thống tài chính. Cũng như phép giao, phép trừ đòi hỏi hai quan hệ phải khả hợp để thao tác diễn ra hợp lệ về mặt cú pháp và ý nghĩa.​

Hai phép toán này, khi kết hợp với các phép tập hợp khác trong đại số quan hệ, chính là nền tảng cho các thao tác truy vấn và phân tích dữ liệu hiện đại. Việc vận dụng đúng vai trò của phép giao và phép trừ sẽ tối ưu hóa cả hoạt động quản trị lẫn lập trình trong lĩnh vực quản lý dữ liệu chuyên sâu.

#### c. Ví dụ minh họa

Xác định sinh viên chưa đăng ký môn học bắt buộc

Cho hai quan hệ:

all\_students(student\_id, name) chứa danh sách tất cả sinh viên trong trường.

registered(student\_id, course\_id) chứa danh sách sinh viên đã đăng ký môn học bắt buộc với mã môn C101.

Yêu cầu: Tìm sinh viên chưa đăng ký môn học C101.

Giải:

Lấy danh sách sinh viên đã đăng ký môn C101:

Dùng phép hiệu để tìm sinh viên chưa đăng ký:

Kết quả là danh sách sinh viên từ all\_students không nằm trong tập sinh viên đã đăng ký môn bắt buộc.

### 2.1.3. Phép tích đề-các

#### a. Định nghĩa và khái niệm

Trong quá trình thao tác với dữ liệu, phép Tích Đề-các (hay còn gọi là Tích Descartes, ký hiệu ) giữa hai quan hệ  và  là một khái niệm nền tảng, đóng vai trò trung gian cho các phép toán phức tạp hơn, đặc biệt là phép kết (join) trong truy vấn cơ sở dữ liệu.

Phép Tích Đề-các  tạo ra một quan hệ mới bằng cách kết hợp mỗi bộ của quan hệ  với **mỗi bộ** của quan hệ . Cụ thể, nếu  có các thuộc tính  và  có các thuộc tính , thì quan hệ kết quả sẽ mang lược đồ ; mỗi bộ là sự ghép nối của một bộ từ  với một bộ từ .​

Cú pháp chuẩn:

**Ảnh hưởng và hệ quả thực tiễn**

Bùng nổ dữ liệu: Nếu R có  bộ và  có  bộ, thì  sẽ có  bộ dữ liệu. Số lượng thuộc tính là tổng số thuộc tính của hai quan hệ, tức . Trong các hệ thống lớn, một phép Tích Đề-các không kiểm soát dễ tạo ra khối lượng dữ liệu "bùng nổ" vượt quá năng lực xử lý hoặc ý nghĩa nghiệp vụ thực tế.​

Kết hợp vô nghĩa: Quá trình này diễn ra đơn thuần về mặt kỹ thuật, không ràng buộc bất kỳ điều kiện logic nào. Mọi bộ từ  sẽ được kết với mọi bộ từ , bất chấp chúng có thực sự liên hệ với nhau hay không. Ví dụ, mọi thông tin sinh viên sẽ kết hợp với mọi khoá học, dù sinh viên chưa từng đăng ký khoá học đó.

**Cơ sở xây dựng phép kết**:

Tích Đề-các hầu như không được sử dụng độc lập trong thực tế, mà thường là bước trung gian kỹ thuật để định nghĩa các phép toán kết, điển hình là:

Trong đó, phép chọn () được dùng để lọc kết quả của Tích Đề-các dựa trên điều kiện , tạo ra các liên hệ có ý nghĩa nghiệp vụ thực sự.

Lưu ý về lược đồ quan hệ: Khi thực hiện Tích Đề-các, cần chú ý các thuộc tính trùng tên giữa hai quan hệ; điều này đòi hỏi việc định danh (naming) rõ ràng hoặc tiền tố (prefix) để tránh xung đột thuộc tính trong lược đồ kết quả.

b. Biểu diễn bằng các toán tử cơ bảnGiả sử ta có hai quan hệ:

Quan hệ  có lược đồ

Quan hệ  có lược đồ

Phép Tích Descartes  cho ra một quan hệ mới trên lược đồ , trong đó:

Mỗi bộ kết quả là sự ghép nối một bộ bất kỳ của  với một bộ bất kỳ của

Cụ thể hơn:

Phép toán này tạo ra số lượng bộ kết quả bằng tích số bộ của hai quan hệ nguồn, tức là nếu  có  bộ và  có  bộ thì  sẽ có  bộ dữ liệu.

Đây là phép toán mang tính "cơ học thuần túy", kết nối mọi bộ từ hai quan hệ mà không cần điều kiện logic nào. Trong thực tế, Tích Descartes thường chỉ là bước trung gian để xây dựng các phép toán phức tạp hơn như phép kết (join), nơi người ta bổ sung thêm điều kiện lọc để tạo ra các mối quan hệ thực sự ý nghĩa.

Phép Tích Descartes chủ yếu hỗ trợ xây dựng truy vấn tự tham chiếu, hoặc dùng làm nền tảng cho các phép hợp nhất dữ liệu có điều kiện (ví dụ: nối dữ liệu sinh viên với môn học mà sinh viên đó đăng ký thông qua điều kiện khoá ngoại).

#### c. Ví dụ minh họa

Đề bài:

Một nhà hàng phục vụ bữa trưa theo thực đơn cố định (combo), bao gồm một món chính và một món tráng miệng.

Tập hợp các món chính là M = {Cơm gà, Phở bò, Bún chả}.

Tập hợp các món tráng miệng là T = {Chè,Hoa quả}.

Hỏi nhà hàng có thể tạo ra bao nhiêu suất ăn trưa (combo) khác nhau? Hãy dùng Tích Đề-các để biểu diễn tất cả các suất ăn này.

1. Mô hình hóa bài toán:

Mỗi suất ăn trưa là một sự kết hợp của một món chính và một món tráng miệng. Chúng ta có thể biểu diễn mỗi suất ăn này dưới dạng một cặp có thứ tự (m, t), trong đó m ∈ M và t ∈ T.

Tập hợp tất cả các suất ăn có thể có chính là Tích Đề-các M × T .

2. Giải quyết bài toán:

Số lượng suất ăn khác nhau chính là lực lượng của Tích Đề-các:

∣M×T∣=∣M∣×∣T∣

Ta có ∣M∣=3 (Cơm gà, Phở bò, Bún chả).

Ta có ∣T∣=2 (Chè, Hoa quả).

Vậy, số suất ăn khác nhau là ∣M×T∣=3×2=6.

3. Liệt kê các suất ăn :

M×T = {(cơm gà, chè),(cơm gà, hoa quả),

(Phở bò , chè).(Phở bò , hoa quả),

(Bún chả, chè),(Bún chả, hoa quả)}

**Kết quả:** Bài toán này minh họa khả năng ứng dụng của Tích Đề-các trong việc giải các bài toán tổ hợp cơ bản. Bằng cách mô hình hóa các lựa chọn thành các tập hợp, chúng ta có thể dùng Tích Đề-các để tìm ra (và liệt kê) tất cả các tổ hợp có thể có một cách hệ thống.

## 2.2. Nhóm các toán tử quan hệ cơ bản

### 2.2.1. Phép chọn

#### a. Định nghĩa và khái niệm

Phép chọn là một toán tử một ngôi trong đại số quan hệ, được sử dụng để chọn ra các bộ (tuple) từ một quan hệ sao cho các bộ này thỏa mãn một điều kiện cụ thể (vị từ). Toán tử chọn tạo thành một quan hệ mới chứa các bộ dữ liệu được lọc theo điều kiện đó, nhằm khai thác những thông tin cần thiết từ tập dữ liệu lớn hơn.

#### b. Biểu diễn bằng các toán tử cơ bản

Phép Chọn trên một quan hệ  được thực hiện dựa trên một điều kiện chọn . Toán tử này, được ký hiệu là , tạo ra một quan hệ mới  chứa tất cả các bộ  trong quan hệ  sao cho bộ  đó thoả mãn điều kiện .

Công thức biểu diễn cụ thể như sau:

Điều đó có nghĩa, phép chọn lọc ra những bộ  mà tại đó vị từ  đánh giá đúng. Như vậy, phép chọn có vai trò làm bộ lọc dữ liệu, chỉ giữ lại các bộ dữ liệu thỏa mãn một hoặc một tập hợp các điều kiện logic nhất định.

**Điều kiện chọn và biểu thức logic**

Điều kiện chọn , thường được gọi là vị từ (predicate), thực chất là một biểu thức logic phức tạp được kết hợp từ các toán tử logic cơ bản và các phép so sánh. Cụ thể gồm:

Các phép toán logic:

* + AND (và), ký hiệu
  + OR (hoặc), ký hiệu
  + NOT (phủ định), ký hiệu

Các phép so sánh: < , > , ≤ , ≥ , ≠ , =

Những toán tử này vận hành trên các giá trị thuộc tính của các bộ trong quan hệ, giúp định nghĩa những tiêu chí phức tạp để lựa chọn dữ liệu.

#### c. Ví dụ minh họa

Hãy bắt đầu với một bảng (quan hệ) tên là NHANVIEN. Bảng này có thể được xem là một tập con của Tích Đề-các của các miền giá trị (ví dụ: Tên x Tuổi x Phòng ban).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MaNV | Ten | Tuoi | Phong Ban | Luong |
| NV01 | AN | 28 | Kế toán | 12000 |
| NV02 | Binh | 20 | Kinh doanh | 10000 |
| NV03 | Tuan | 25 | Kỹ thuật | 15000 |
| NV04 | Son | 30 | Kế toán | 12000 |
| NV05 | Cuong | 24 | Kinh doanh | 10000 |

Yêu cầu : Tìm tất cả nhân viên làm việc ở phòng "Kinh doanh".

Giải thích: Phép toán này sẽ duyệt qua từng hàng của bảng NHANVIEN. Với mỗi hàng, nó kiểm tra giá trị của cột PhongBan.

Giải :

σ PhongBan=’Kinh doanh’ ​ (NHANVIEN)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MaNV | Ten | Tuoi | Phong Ban | Luong |
| NV02 | Binh | 20 | Kinh doanh | 10000 |
| NV05 | Cuong | 24 | Khinh doanh | 10000 |

Yêu cầu : Tìm tất cả nhân viên ở phòng "Kỹ thuật" VÀ có Tuoi dưới 30.

Giải thích: Phép toán này kiểm tra đồng thời hai điều kiện

Giải :

σ PhongBan=’Kỹ thuật’∧Tuoi<30 ​ (NHANVIEN)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MaNV | Ten | Tuoi | Phong Ban | Luong |
| NV03 | Tuan | 25 | Kỹ thuật | 15000 |

### 2.2.2.Phép chiếu

#### a. Định nghĩa và khái niệm

Phép Chiếu là một toán tử một ngôi, được sử dụng để chọn ra những thuộc tính (cột) cụ thể từ một quan hệ. Kết quả là một quan hệ mới chỉ chứa các cột được chỉ định, loại bỏ hoàn toàn các cột không cần thiết. Như vậy, phép chiếu thực hiện việc lọc dữ liệu theo chiều dọc, giúp tập trung vào các thuộc tính quan trọng trong quá trình phân tích hay truy vấn.

Loại bỏ trùng lặp:Một điểm rất quan trọng cần nhấn mạnh là kết quả của phép chiếu là một tập hợp, do đó khi việc chọn cột dẫn đến các bộ (hàng) giống hệt nhau thì chỉ giữ lại một bản duy nhất, các bản trùng sẽ bị loại bỏ hoàn toàn, đảm bảo sự duy nhất và không trùng lặp trong kết quả.

#### b. Biểu diễn bằng các toán tử cơ bản

Mở rộng phép chiếu trong đại số quan hệ nhằm tăng tính linh hoạt và khả năng xử lý, cho phép sử dụng các phép toán số học trong danh sách thuộc tính được chọn. Điều này giúp không chỉ chọn ra các thuộc tính có sẵn mà còn cho phép thực hiện tính toán trực tiếp trên các thuộc tính đó trong quá trình chiếu.

Cụ thể, ký hiệu cho phép mở rộng này là:

trong đó:

 là biểu thức đại số quan hệ (có thể là một quan hệ hoặc một biểu thức phức tạp hơn),  là các biểu thức số học liên quan đến hằng số hoặc các thuộc tính từ quan hệ .

Như vậy, mỗi  có thể là: Một hằng số cụ thể, thuộc tính đơn giản,hoặc một phép toán số học gồm các phép cộng, trừ, nhân, chia,... áp dụng trên các thuộc tính.

Phép chiếu mở rộng này rất hữu ích trong việc tạo ra các báo cáo tổng hợp, tính toán trực tiếp trên dữ liệu mà không cần phải viết thêm các truy vấn phức tạp hoặc xử lý sau đó.

#### c. Ví dụ minh họa

Lấy đề bài từ ví dụ ........

Yêu cầu : Chỉ hiển thị Tên và Lương của tất cả nhân viên.

Giải thích: Phép toán này sẽ lấy tất cả các hàng, nhưng chỉ giữ lại 2 cột là Ten và Luong. Trong trường hợp này, không có hai hàng nào có (Tên, Lương) giống hệt nhau, nên không có hàng nào bị loại bỏ.

Giải :

π Ten,Luong ​ (NHANVIEN)

|  |  |
| --- | --- |
| MaNV | Luong |
| NV01 | 12000 |
| NV02 | 10000 |
| NV03 | 15000 |
| NV04 | 12000 |
| NV05 | 10000 |

Yêu cầu : Liệt kê tất cả các phòng ban mà công ty đang có.

Giải thích: Phép toán này chỉ giữ lại cột PhongBan.

Hàng NV01: PhongBan = 'Kế toán'. (Giữ lại)

Hàng NV02: PhongBan = 'Kinh doanh'. (Giữ lại)

Hàng NV03: PhongBan = 'Kỹ thuật'. (Giữ lại)

Hàng NV04: PhongBan = 'Kinh doanh'. (Hàng này trùng với hàng 2 => Loại bỏ)

Hàng NV05: PhongBan = 'Kỹ thuật'. (Hàng này trùng với hàng 3 => Loại bỏ)

Giải:

π PhongBan ​ (NHANVIEN)

|  |
| --- |
| Phong ban |
| Kế toán |
| Kinh doanh |
| Kỹ thuật |

Yêu cầu : Tìm Tên và Tuổi của các nhân viên làm việc ở phòng "Kỹ thuật".

Giải thích : Chúng ta thực hiện từ trong ra ngoài

Phép Chọn (Bên trong): σ PhongBan=’Kỹ thuật’ ​ (NHANVIEN)

Thao tác này sẽ lọc và chỉ giữ lại các hàng NV03.

Phép Chiếu (Bên ngoài) : π Ten,Tuoi(σ PhongBan=’Kỹ thuật’ (NHANVIEN))

Áp dụng phép chiếu lên bảng tạm thời ở trên, ta chỉ giữ lại cột Ten và Tuoi.

Giải:

π Ten,Tuoi(σ PhongBan=’Kỹ thuật’ (NHANVIEN))

|  |  |
| --- | --- |
| Ten | Tuoi |
| Tuan | 25 |

### 2.2.3.Phép đổi tên

#### a. Định nghĩa và khái niệm

Phép Đổi Tên, ký hiệu là , là một toán tử một ngôi không làm thay đổi nội dung dữ liệu mà chỉ thay đổi tên của lược đồ quan hệ. Nó rất cần thiết trong các tình huống sau:

Truy vấn tự tham chiếu (self-join): Khi cần thực hiện Tích Đề-các (hay Kết) một quan hệ với chính nó. Chẳng hạn, để so sánh mức lương của các giảng viên với nhau, ta phải lấy . Tuy nhiên, không thể phân biệt được các thuộc tính như salary ở quan hệ bên trái hay bên phải nếu không đổi tên. Phép Đổi Tên giúp giải quyết vấn đề này bằng cách tạo ra các tên quan hệ và thuộc tính riêng biệt.

Đặt tên cho kết quả trung gian: Khi một biểu thức con trở nên quá phức tạp,  cho phép đặt một tên tạm thời, ngắn gọn, dễ hiểu cho quan hệ kết quả để tiện sử dụng trong các biểu thức truy vấn tiếp theo.

#### b. Biểu diễn bằng các toán tử cơ bản

Phép đổi tên là một toán tử một ngôi quan trọng trong đại số quan hệ, cho phép thay đổi tên quan hệ hoặc tên các thuộc tính mà không làm thay đổi dữ liệu bên trong. Đây là công cụ hữu ích giúp phân biệt các quan hệ giống nhau trong các truy vấn phức tạp hoặc khi cần đánh dấu các kết quả trung gian.

**Đổi tên quan hệ:**

Trong đó:

 là biểu thức đại số quan hệ (hoặc quan hệ nguồn),

 là tên mới sẽ được đặt cho quan hệ kết quả.

Đổi tên quan hệ và các thuộc tính:

Trong đó:  là tên quan hệ mới,  là danh sách các tên thuộc tính mới theo thứ tự tương ứng với các thuộc tính trong quan hệ .

Phép đổi tên giúp làm rõ và thuận tiện hơn trong việc xây dựng các truy vấn phức tạp, đặc biệt cần phân biệt các cá thể cùng quan hệ hoặc quản lý các kết quả trung gian một cách hiệu quả.

#### c. Ví dụ minh họa

Yêu cầu : Tạo một bảng mới từ NHANVIEN nhưng đổi tên cột Ten thành HoVaTen và PhongBan thành BoPhan.

Giải thích : Dữ liệu y hệt, chỉ có tiêu đề cột thay đổi

Giải :

ρ MaNV,HoVaTen,Tuoi,BoPhan,Luong ​ (NHANVIEN)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MaNV | Ho Va Ten | Tuoi | Bo Phan | Luong |
| NV01 | AN | 28 | Kế toán | 12000 |
| NV02 | Binh | 20 | Kinh doanh | 10000 |
| NV03 | Tuan | 25 | Kỹ thuật | 15000 |
| NV04 | Son | 30 | Kế toán | 12000 |
| NV05 | Cuong | 24 | Kinh doanh | 10000 |

# CHƯƠNG 3. CÁC TOÁN TỬ DẪN XUẤT NÂNG CAO

## Phép giao (intersection)

### 3.1.1. Định nghĩa và khái niệm

Trong Đại số Quan hệ, phép Giao (INTERSECTION) là một toán tử nhị phân (binary operator) được sử dụng để tìm ra các bộ (tuple) xuất hiện trong cả hai quan hệ. Phép giao thể hiện phần dữ liệu chung giữa hai tập quan hệ, nghĩa là chỉ những bộ có mặt đồng thời trong cả hai mới được đưa vào kết quả.

**Nguồn:** [1] Ramez Elmasri, Shamkant B. Navathe, Fundamentals of Database Systems, 6th Edition, Addison-Wesley, 2011.

Phép giao là một trong các phép toán tập hợp cơ bản, cùng với phép hợp (UNION) và phép hiệu (SET DIFFERENCE). Các phép này có nguồn gốc từ toán học và được mở rộng để áp dụng trong mô hình dữ liệu quan hệ.

Khi áp dụng phép giao, hai quan hệ R và S phải tương thích hợp (union-compatible). Điều này có nghĩa là:

* + - * Hai quan hệ phải có cùng số lượng thuộc tính.
      * Mỗi cặp thuộc tính tương ứng giữa hai quan hệ phải có cùng miền giá trị (domain).

Nếu điều kiện này được thỏa mãn, phép giao giữa hai quan hệ được định nghĩa như sau:

R ∩ S ={t ∣ t ∈ R và t ∈ S}

Kết quả: là một quan hệ mới bao gồm các bộ có mặt trong cả R và S. Theo quy ước, kết quả sẽ giữ nguyên tên thuộc tính của quan hệ thứ nhất R. Nếu cần, người thiết kế có thể dùng phép đổi tên (Rename – ρ) để điều chỉnh lại tên thuộc tính cho phù hợp với ngữ cảnh truy vấn.

### 3.1.2. Biểu diễn bằng các toán tử cơ bản

Phép giao không phải là một toán tử cơ bản trong Đại số Quan hệ vì nó có thể được biểu diễn lại bằng các phép toán khác như phép hợp (UNION) và phép hiệu (SET DIFFERENCE).

Cụ thể, phép giao có thể được viết lại dưới dạng:

R ∩ S = R − ( R – S )

Điều này có nghĩa là phần giao của hai quan hệ chính là tập hợp các bộ thuộc R sau khi đã loại bỏ những bộ không có trong S. Ngoài ra, trong một số cách biểu diễn khác, phép giao cũng có thể được mô tả dưới dạng kết hợp của phép hợp và phép hiệu như sau:

R ∩ S = (( R ∪ S ) − ( R – S )) − ( S – R )

Như vậy, phép giao là một toán tử dẫn xuất (derived operator), vì có thể được định nghĩa dựa trên các phép toán cơ bản. Tuy nhiên, trong thực tế, việc dùng trực tiếp ký hiệu ∩ giúp các biểu thức truy vấn trở nên ngắn gọn và trực quan hơn nhiều.

**Nguồn:** [1] Ramez Elmasri, Shamkant B. Navathe, Fundamentals of Database Systems, 6th Edition, Addison-Wesley, 2011.

### 3.1.3. Ví dụ minh họa

Xét hai quan hệ có cùng cấu trúc thuộc tính sau:

**STUDENT**

|  |  |
| --- | --- |
| **Fname** | **Fname** |
| Susan | Yao |
| Ramesh | Shah |
| Johnny | Kohler |
| Barbara | Jones |
| Amy | Ford |
| Jimmy | Wang |
| Ernest | Gilbert |

**INSTRUCTOR**

|  |  |
| --- | --- |
| Fname | Lname |
| John | Smith |
| Ricardo | Browne |
| Susan | Yao |
| Francis | Johnson |
| Ramesh | Shah |

Hai quan hệ này có cùng cấu trúc thuộc tính (Fname, Lname), do đó chúng tương thích hợp và có thể áp dụng các phép toán tập hợp như UNION, INTERSECTION và MINUS.

Khi thực hiện phép giao:

RESULT ← STUDENT ∩ INSTRUCTOR

Kết quả thu được là:

|  |  |
| --- | --- |
| **Fname** | **Fname** |
| Susan | Yao |
| Ramesh | Shah |

Kết quả này thể hiện rằng hai cá nhân **Susan Yao** và **Ramesh Shah** đồng thời có trong cả hai quan hệ, nghĩa là họ vừa là sinh viên vừa là giảng viên.

Phép giao cho phép loại bỏ các bản ghi trùng lặp và chỉ giữ lại các bộ thực sự xuất hiện trong cả hai quan hệ, qua đó giúp nhận diện và xử lý các phần dữ liệu chung giữa các bảng trong cơ sở dữ liệu.

### 3.1.4. Tính chất đại số của phép giao

Phép giao có một số tính chất quan trọng trong Đại số Quan hệ, phản ánh trực tiếp các quy tắc trong lý thuyết tập hợp:

**Tính giao hoán :**

Phép Giao là một phép toán giao hoán, nghĩa là thứ tự của hai quan hệ không ảnh hưởng đến kết quả cuối cùng.

R ∩ S = S ∩ R

Ví dụ: Nếu R = {1, 2, 3} và S={2,3,4} thì cả R ∩ S và S ∩ R đều cho kết quả {2,3}.

Điều này thể hiện rằng khi hai quan hệ R và S có cùng cấu trúc (union-compatible), việc hoán đổi vị trí của chúng trong biểu thức không làm thay đổi tập kết quả.

**Tính kết hợp :**

Tương tự, phép Giao còn có tính kết hợp, nghĩa là khi áp dụng phép toán này cho ba hay nhiều quan hệ khác nhau, ta có thể nhóm các phép toán theo bất kỳ thứ tự nào mà không ảnh hưởng đến kết quả.

( R ∩ S ) ∩ T = R ∩ ( S ∩ T)

Ví dụ: Với 𝑅 = {1, 2, 3} , 𝑆 = {2, 3, 4}, và 𝑇 = {3, 4, 5}, kết quả ở cả hai vế đều là {3}.

Tính chất này giúp cho việc biểu diễn và xử lý các truy vấn phức hợp trong hệ cơ sở dữ liệu trở nên linh hoạt hơn.

**Nguồn:** [1] Ramez Elmasri, Shamkant B. Navathe, Fundamentals of Database Systems, 6th Edition, Addison-Wesley, 2011.

## Phép kết (join)

### 3.2.1. Kết theta (theta JOIN)

Phép kết nối là một trong những phép toán quan trọng và phổ biến trong đại số quan hệ. Mục đích của phép kết nối là kết hợp dữ liệu từ hai hay nhiều quan hệ khác nhau dựa trên một điều kiện xác định, nhằm tạo ra một quan hệ mới chứa thông tin có ý nghĩa hơn. Trong đó, phép kết nối Theta (Theta Join) là dạng tổng quát nhất của phép kết nối, cho phép sử dụng bất kỳ toán tử so sánh nào trong điều kiện nối, thay vì chỉ giới hạn ở phép bằng.

Phép kết nối Theta được ký hiệu là ⋈θ​, trong đó “θ” là toán tử so sánh được sử dụng trong điều kiện nối. Giả sử có hai quan hệ R(A1,A2,…,An) và S(B1, B2,…,Bm) , khi đó điều kiện nối có thể được biểu diễn dưới dạng:

C = (Ai​θBj​)

với θ ∈ {=,≠,<,>,≤,≥}. Khi thực hiện phép kết nối Theta, chỉ những cặp bộ (tr, ts) trong đó tr ∈ R và tr ∈ S thỏa mãn điều kiện C mới được đưa vào kết quả. Kết quả của phép toán là một quan hệ mới có lược đồ bao gồm toàn bộ thuộc tính của hai quan hệ ban đầu.

Phép kết nối Theta không phải là một phép toán cơ bản, mà là phép toán dẫn xuất, được định nghĩa dựa trên sự kết hợp giữa phép chọn (σ) và phép tích Descartes (×). Cụ thể, phép kết nối Theta có thể được viết lại dưới dạng:

R ⋈C​ S = σC​(R × S)

Trong đó, phép tích Descartes R × S tạo ra tất cả các cặp bộ có thể giữa hai quan hệ, còn phép chọn σC​ được sử dụng để lọc các bộ thỏa mãn điều kiện C. Điều này cho thấy phép kết nối Theta có thể được hiểu như một bước rút gọn có điều kiện trên kết quả của phép tích Descartes, giúp giảm đáng kể số lượng bộ được xét so với việc kết hợp toàn bộ các giá trị.

Phép kết nối Theta có tính tổng quát cao, vì nó cho phép sử dụng nhiều loại toán tử khác nhau để mô tả mối quan hệ giữa các thuộc tính của hai quan hệ. Nhờ đó, người dùng có thể biểu diễn linh hoạt các truy vấn mang tính điều kiện phức tạp, chẳng hạn như so sánh giá trị thuộc tính, giới hạn miền dữ liệu, hay kiểm tra chênh lệch giữa các cột trong hai bảng. Trong đại số quan hệ, đây được xem là phép toán trung gian quan trọng để hình thành các phép kết nối cụ thể khác như Equi-Join hoặc Natural Join.

**Xét ví dụ hai quan hệ sau:**

SINHVIEN(MSSV, HoTen, MaLop, DiemTB)

LOP(MaLop, TenLop, Khoa)

Mục tiêu là tìm tất cả các cặp sinh viên và lớp học sao cho mã lớp trong quan hệ SINHVIEN trùng với mã lớp trong quan hệ LOP. Khi đó điều kiện nối là:

C = (SINHVIEN.MaLop = LOP.MaLop)

Phép kết nối Theta tương ứng được viết:

SINHVIEN⋈SINHVIEN.MaLop=LOP.MaLop​LOP

và có thể biểu diễn bằng các phép cơ bản như sau:

σSINHVIEN.MaLop=LOP.MaLop​(SINHVIEN × LOP)

Kết quả thu được là một quan hệ mới chứa các thuộc tính của cả hai bảng:

(MSSV, HoTen, SINHVIEN.MaLop, DiemTB, LOP.MaLop, TenLop, Khoa)

Mỗi bộ trong quan hệ kết quả thể hiện thông tin đầy đủ về sinh viên và lớp mà họ đang theo học.

Ngoài việc dùng toán tử bằng “=”, phép kết nối Theta còn có thể sử dụng các toán tử khác để mở rộng khả năng biểu diễn truy vấn. Chẳng hạn, nếu cần tìm các sinh viên có điểm trung bình cao hơn điểm trung bình chung của lớp mà họ học, điều kiện kết nối có thể viết:

C = (SINHVIEN.DiemTB > LOP.DiemTBtb)

Khi đó, phép kết nối Theta trở thành:

SINHVIEN⋈SINHVIEN.DiemTB>LOP.DiemTBtb​LOP

Phép toán này giúp xác định những sinh viên nổi bật hơn mức trung bình của lớp, minh họa rõ khả năng linh hoạt của phép kết nối Theta trong việc mô tả các mối quan hệ logic giữa các thuộc tính dữ liệu.

Phép kết nối Theta cũng có các tính chất đại số quan trọng tương tự như các phép nối khác trong đại số quan hệ, bao gồm tính giao hoán và tính kết hợp. Nghĩa là, với ba quan hệ R, S và T, ta có thể thực hiện các phép nối theo bất kỳ thứ tự nào mà không làm thay đổi kết quả cuối cùng:

R ⋈ S=S ⋈ R và (R ⋈ S) ⋈ T = R⋈(S ⋈ T)

Những tính chất này giữ vai trò quan trọng trong tối ưu hóa truy vấn, giúp hệ quản trị cơ sở dữ liệu có thể lựa chọn thứ tự thực hiện phép nối hiệu quả nhất để giảm chi phí xử lý.

Phép kết nối Theta cũng đóng vai trò nền tảng cho nhiều phép toán nâng cao trong đại số quan hệ. Các phép như Equi-Join (khi θ ==), Natural Join (loại bỏ các cột trùng lặp sau khi Equi-Join), hay Outer Join (bảo toàn các bộ không khớp) đều có thể xem như các trường hợp đặc biệt hoặc mở rộng của phép kết nối Theta. Nhờ vậy, phép nối này được xem là cầu nối giữa lý thuyết đại số quan hệ thuần túy và các thao tác truy vấn phức tạp trong hệ quản trị cơ sở dữ liệu.

**3.2.2. Kết bằng (equi JOIN)**

Phép kết nối bằng là một trường hợp đặc biệt của phép kết nối Theta, trong đó toán tử so sánh được sử dụng là dấu bằng (=). Mục tiêu của phép kết nối bằng là kết hợp các bộ dữ liệu từ hai quan hệ khác nhau khi các giá trị của một hay nhiều thuộc tính trong hai quan hệ đó bằng nhau. Đây là một trong những phép toán phổ biến nhất trong đại số quan hệ, vì trong hầu hết các cơ sở dữ liệu, các quan hệ thường được liên kết thông qua các khóa có cùng giá trị định danh.

Giả sử có hai quan hệ R(A1, A2,…, An) và S(B1​, B2​,…, Bm​), trong đó tồn tại cặp thuộc tính Ai​ và Bj ​ có cùng miền giá trị. Khi đó, phép kết nối bằng được ký hiệu là:

R ⋈ Ai​=Bj​​ S

và được định nghĩa là tập hợp tất cả các bộ kết hợp (tr, ts) , trong đó tr ∈ R và ts ∈ S thỏa mãn điều kiện tr[Ai] = ts[Bj]. Kết quả của phép toán là một quan hệ mới có lược đồ bao gồm tất cả các thuộc tính của hai quan hệ đầu vào, trong đó các cặp bộ có giá trị thuộc tính bằng nhau được kết hợp lại.

Phép kết nối bằng, tương tự như các phép nối khác, có thể được biểu diễn bằng các phép toán cơ bản trong đại số quan hệ. Cụ thể, phép này được xác định bởi công thức:

R⋈Ai​=Bj​​ S = σAi​=Bj​​(R × S)

Điều này có nghĩa rằng phép kết nối bằng được hình thành bằng cách thực hiện phép tích Descartes giữa hai quan hệ R và S, sau đó áp dụng phép chọn để giữ lại các bộ có giá trị thuộc tính bằng nhau. Phương pháp này giúp làm rõ bản chất của Equi-Join như là một phép chọn có điều kiện bằng trên tích Descartes.

Để minh họa, xét hai quan hệ sau:

NHANVIEN(MaNV, TenNV, MaPB, Luong)

PHONGBAN(MaPB, TenPB, DiaChi)

Điều kiện nối là :

C = (NHANVIEN.MaPB = PHONGBAN.MaPB)

Trong phép kết nối bằng, điều kiện nối được chỉ định tường minh thông qua dấu bằng (=) giữa các thuộc tính tương ứng của hai quan hệ, nhằm xác định các cặp bộ có giá trị thuộc tính bằng nhau.

Khi thực hiện phép kết nối bằng, ta thu được:

NHANVIEN ⋈NHANVIEN.MaPB=PHONGBAN.MaPB​ PHONGBAN

Kết quả của phép toán là một quan hệ mới chứa các thuộc tính:

(MaNV, TenNV, MaPB(NHANVIEN)​, Luong, MaPB(PHONGBAN)​, TenPB, DiaChi)

Trong kết quả, ta nhận thấy rằng thuộc tính MaPB xuất hiện hai lần, một lần từ quan hệ NHANVIEN và một lần từ PHONGBAN. Đây là đặc điểm tiêu biểu của phép Equi-Join: phép nối này không loại bỏ các thuộc tính trùng lặp, mà giữ nguyên toàn bộ cấu trúc của hai quan hệ ban đầu. Việc trùng thuộc tính phản ánh đúng bản chất của phép toán này, khi nó chỉ đảm bảo điều kiện “bằng nhau” mà không thực hiện thao tác gộp hay loại bỏ.

Đặc điểm này cũng là cơ sở để hình thành phép kết nối tự nhiên (Natural Join), một biến thể của Equi-Join, trong đó các thuộc tính trùng tên sẽ được tự động loại bỏ khỏi kết quả để tạo ra một quan hệ ngắn gọn, dễ đọc và trực quan hơn. Điều đó giúp Natural Join thể hiện cùng một mối quan hệ dữ liệu như Equi-Join, nhưng dưới dạng cô đọng và tối ưu hơn.

Phép kết nối bằng cũng có các tính chất đại số quan trọng tương tự như các phép nối khác trong đại số quan hệ, bao gồm tính giao hoán và tính kết hợp. Nghĩa là, với ba quan hệ R, S và T, ta có thể thực hiện các phép nối theo bất kỳ thứ tự nào mà không làm thay đổi kết quả cuối cùng:

R ⋈ S= S ⋈ R và (R ⋈ S) ⋈ T = R ⋈ (S ⋈ T)

Những tính chất này giữ vai trò quan trọng trong tối ưu hóa truy vấn, giúp hệ quản trị cơ sở dữ liệu có thể lựa chọn thứ tự thực hiện phép nối hiệu quả nhất để giảm chi phí xử lý.

Tóm lại, phép kết nối bằng là một phép nối nền tảng trong đại số quan hệ, thường được sử dụng để liên kết dữ liệu giữa các bảng có khóa chung. Nó thể hiện mối quan hệ logic giữa các thực thể có cùng giá trị thuộc tính, đồng thời là nền tảng cho các phép nối mở rộng như Natural Join hay Outer Join. Hiểu rõ đặc điểm của phép Equi Join đặc biệt là hiện tượng trùng thuộc tính giúp nắm bắt sâu hơn bản chất của các phép nối trong cơ sở dữ liệu quan hệ.

**3.2.3. Phép kết tự nhiên ( natural join )**

Phép kết nối tự nhiên là một biến thể đặc biệt của phép kết nối bằng (Equi-Join), được sử dụng phổ biến trong đại số quan hệ. Điểm khác biệt của phép kết nối tự nhiên là nó tự động loại bỏ các thuộc tính trùng tên trong kết quả, nhờ đó giúp tạo ra một quan hệ mới ngắn gọn, trực quan và dễ hiểu hơn. Phép toán này được gọi là “tự nhiên” vì nó dựa trên sự tương đồng tên thuộc tính giữa các quan hệ, chứ không cần chỉ định điều kiện nối một cách tường minh.

Giả sử có hai quan hệ R(A1, A2,…, An) và S(B1, B2,…, Bm). Nếu hai quan hệ này có một hoặc nhiều thuộc tính trùng tên, khi đó phép kết nối tự nhiên giữa chúng được ký hiệu là:

R ⋈ S

và được định nghĩa là phép nối giữa R và S trên tất cả các cặp thuộc tính có cùng tên và cùng miền giá trị. Kết quả của phép nối là một quan hệ mới, trong đó:

* + - * Chỉ giữ lại một bản duy nhất của mỗi thuộc tính trùng tên.
      * Chứa tất cả các thuộc tính không trùng tên từ cả hai quan hệ.

Phép kết nối tự nhiên có thể được xem như một phép Equi-Join kèm theo bước loại bỏ các thuộc tính trùng lặp. Vì vậy, nó có thể biểu diễn bằng công thức sau:

R ⋈ S = ΠA,B​(σAi​=Bi​​(R × S))

trong đó phép chọn σAi=Biσ\_{A\_i = B\_i}σAi​=Bi​​ thực hiện điều kiện bằng nhau giữa các thuộc tính trùng tên, và phép chiếu ΠA,B loại bỏ các cột trùng lặp trong kết quả.

Để minh họa, xét lại ví dụ hai quan hệ:

NHANVIEN(MaNV, TenNV, MaPB, Luong)

PHONGBAN(MaPB, TenPB, DiaChi)

Điều kiện nối là:

Điều khác biệt là ở phép kết nối tự nhiên, điều kiện này không cần chỉ định tường minh như trong Equi-Join, mà được tự động xác định dựa trên các thuộc tính trùng tên và cùng miền giá trị giữa hai quan hệ ở đây là MaPB.

Khi thực hiện phép kết nối tự nhiên, ta có:

NHANVIEN ⋈ PHONGBAN

Kết quả của phép toán là một quan hệ mới chứa các thuộc tính:

(MaNV, TenNV, MaPB, Luong, TenPB, DiaChi)

Kết quả thu được là một quan hệ thể hiện mối liên hệ giữa mỗi nhân viên và phòng ban mà họ thuộc về, trong đó thuộc tính MaPB chỉ xuất hiện một lần nhờ đặc tính tự động loại bỏ trùng lặp của phép Natural Join. Kết quả này vì vậy ngắn gọn hơn và dễ quan sát hơn so với kết quả của phép Equi Join.

Phép kết nối tự nhiên cũng kế thừa các tính chất đại số của phép nối tổng quát, bao gồm tính giao hoán và tính kết hợp. Nghĩa là, với ba quan hệ R, S và T, ta có thể thực hiện các phép nối tự nhiên theo bất kỳ thứ tự nào mà không làm thay đổi kết quả cuối cùng:

R ⋈ S = S ⋈ R và (R ⋈ S) ⋈ T = R ⋈ (S ⋈ T)

Những tính chất này rất quan trọng trong tối ưu hóa truy vấn, giúp hệ quản trị cơ sở dữ liệu lựa chọn thứ tự thực hiện phép nối sao cho hiệu quả nhất.

Tóm lại, phép kết nối tự nhiên là một phép nối mở rộng từ Equi Join, vừa giữ được bản chất logic của phép nối bằng, vừa loại bỏ sự dư thừa thuộc tính. Phép Natural Join giúp biểu diễn các mối quan hệ giữa các bảng dữ liệu một cách ngắn gọn, rõ ràng và dễ xử lý hơn. Nhờ ưu điểm này, nó trở thành một trong những phép toán được sử dụng rộng rãi nhất trong thực tế khi mô hình hóa và truy vấn cơ sở dữ liệu quan hệ.

**3.2.4. Phép kết ngoài ( outer join )**

Phép kết nối ngoài (Outer Join) là một mở rộng của các phép kết nối thông thường trong đại số quan hệ, chẳng hạn như Equi Join hoặc Natural Join. Điểm khác biệt của phép nối này là nó không chỉ giữ lại các bộ thỏa mãn điều kiện nối giữa hai quan hệ, mà còn bảo toàn những bộ không có giá trị khớp ở một trong hai quan hệ. Nhờ vậy, kết quả của phép kết nối ngoài luôn bao gồm toàn bộ dữ liệu từ một hoặc cả hai quan hệ, với các giá trị thiếu được điền bằng NULL.

Mục đích của phép kết nối ngoài là giúp tránh mất dữ liệu trong các tình huống mà một trong hai bảng có các bộ không có giá trị khớp, chẳng hạn như khi muốn liệt kê toàn bộ nhân viên kể cả những người chưa được phân công phòng ban, hoặc toàn bộ phòng ban kể cả những phòng chưa có nhân viên.

#### 3.2.4.1. Phân loại phép kết nối ngoài

Phép kết nối ngoài bao gồm ba biến thể chính:

* + - * **Left Outer Join (⋈⟕)** – Giữ lại tất cả các bộ của quan hệ bên trái
      * **Right Outer Join (⋈⟖)** – Giữ lại tất cả các bộ của quan hệ bên phải.
      * **Full Outer Join (⋈⟗)** – Giữ lại tất cả các bộ của cả hai quan hệ.

Các phép này chỉ khác nhau ở phạm vi giữ lại các bộ không khớp, còn bản chất xử lý phần “nối” vẫn tương tự như Equi-Join hoặc Natural Join.

#### 3.2.4.2. Ký hiệu và công thức

Giả sử có hai quan hệ R và S cùng điều kiện nối C:

**Left Outer Join:**

R⟕C​S

Bao gồm tất cả các bộ từ R⋈CS và các bộ còn lại của R không có bộ tương ứng trong S, với các giá trị thiếu được điền bằng NULL.

**Right Outer Join:**

R⟖C​S

Giữ lại toàn bộ các bộ từ S, kể cả những bộ không có đối tượng tương ứng trong R.

**Full Outer Join:**

R⟗C​S

Kết hợp kết quả của cả Left và Right Outer Join, đảm bảo rằng tất cả các bộ từ cả hai quan hệ đều được đưa vào kết quả cuối cùng.

#### 3.2.4.3. Ví dụ minh họa

Xét hai quan hệ:

NHANVIEN(MaNV, TenNV, MaPB)

PHONGBAN(MaPB, TenPB)

Giả sử điều kiện nối là:

C=(NHANVIEN.MaPB=PHONGBAN.MaPB)

* **Left Outer Join:**

NHANVIEN⟕NHANVIEN.MaPB=PHONGBAN.MaPB​PHONGBAN

Kết quả bao gồm tất cả nhân viên, kể cả những người chưa được phân vào phòng ban nào.

Các giá trị thuộc tính của PHONGBAN sẽ được điền NULL nếu không có giá trị khớp.

|  |  |  |  |
| --- | --- | --- | --- |
| **MaNV** | **TenNV** | **MaPB** | **TenPB** |
| NV01 | An | P01 | Kinh doanh |
| NV02 | Bình | P02 | Kế toán |
| NV03 | Cường | NULL | NULL |

* **Right Outer Join:**

NHANVIEN⟖NHANVIEN.MaPB=PHONGBAN.MaPB​PHONGBAN

Kết quả bao gồm tất cả phòng ban, kể cả những phòng chưa có nhân viên nào.

Các giá trị của NHANVIEN sẽ là NULL nếu không có bộ tương ứng.

|  |  |  |  |
| --- | --- | --- | --- |
| **MaNV** | **MaNV** | **MaPB** | **TenPB** |
| NV01 | An | P01 | Kinh doanh |
| NV02 | Bình | P02 | Kế toán |
| NULL | NULL | P03 | Nhân sự |

* **Full Outer Join:**

NHANVIEN⟗NHANVIEN.MaPB=PHONGBAN.MaPB​PHONGBAN

Kết quả bao gồm tất cả các bộ từ cả hai quan hệ, dù có hay không có giá trị khớp.

|  |  |  |  |
| --- | --- | --- | --- |
| **MaNV** | **MaNV** | **MaPB** | **TenPB** |
| NV01 | An | P01 | Kinh doanh |
| NV02 | Bình | P02 | Kế toán |
| NV03 | Cường | NULL | NULL |
| NULL | NULL | P03 | Nhân sự |

#### 3.2.4.4. Tính chất của phép kết nối ngoài

Phép kết nối ngoài mở rộng tính chất giao hoán và kết hợp của phép nối thông thường, tuy nhiên có một số điểm cần lưu ý:

* Left Outer Join và Right Outer Join không hoàn toàn giao hoán, do vai trò “bên trái – bên phải” khác nhau. Tuy nhiên, chúng có mối quan hệ đối xứng:

R ⟕ S = S ⟖ R

* **Full Outer Join** có tính giao hoán và kết hợp tương tự phép nối trong:

R ⟗ S =S ⟗ R, (R ⟗ S) ⟗ T = R⟗ (S ⟗ T)

#### 3.2.4.5. Ý nghĩa và ứng dụng

Phép kết nối ngoài đặc biệt hữu ích trong các tình huống dữ liệu không đầy đủ hoặc thiếu quan hệ ràng buộc. Nó cho phép người dùng:

* Giữ lại toàn bộ thông tin của một bảng chính mà không bị mất dữ liệu.
* Thực hiện các truy vấn tổng hợp, thống kê, và báo cáo đầy đủ hơn.
* Hỗ trợ các bài toán thực tế như “liệt kê tất cả nhân viên và phòng ban, kể cả khi chưa có sự phân công”.

Phép Outer Join do đó được xem là sự mở rộng linh hoạt và thực tiễn của các phép nối cơ bản trong đại số quan hệ, đồng thời là nền tảng quan trọng trong các hệ quản trị cơ sở dữ liệu hiện đại.

## 3.3.Phép chia ( Division )

**3.3.1. Khái niệm và ý nghĩa**

Phép Chia (Division – ÷) là một trong những phép toán nhị phân quan trọng trong đại số quan hệ, được sử dụng cho những truy vấn có chứa cụm từ “for all” (cho tất cả).

Phép chia giúp ta tìm ra các giá trị của một thuộc tính trong quan hệ R(X, Y) sao cho giá trị đó liên kết với tất cả các giá trị của thuộc tính trong một quan hệ khác S(Y).

Nói một cách cụ thể hơn, nếu ta có hai quan hệ R(X, Y) và S(Y), thì kết quả của phép chia R ÷ S là một quan hệ T(X) gồm các giá trị X mà đối với mọi giá trị Y trong S, cặp (X, Y) đều tồn tại trong R. Điều này có nghĩa là phép chia dùng để biểu diễn mối quan hệ dạng “X có liên quan đến tất cả các Y trong S”.

Phép chia rất hữu ích trong các tình huống thực tế, khi ta cần tìm các thực thể thỏa mãn tất cả các điều kiện trong một tập xác định.

Ví dụ:

* + - * Tìm các **sinh viên đã đăng ký tất cả các khóa học** trong chương trình đào tạo
      * Tìm các **nhân viên làm việc trong mọi dự án** do một phòng ban quản lý.

Những loại truy vấn này thường khó biểu diễn bằng các phép toán đơn giản, nhưng phép chia giúp mô tả chúng một cách tự nhiên và chính xác hơn.

**3.3.2. Biểu diễn bằng các phép toán cơ bản**

Phép chia có thể được biểu diễn lại bằng các phép toán cơ bản của đại số quan hệ như phép chiếu (π), phép hiệu (−) và phép tích Descartes (×).

Ta có công thức tổng quát như sau:

T(X) = πX​(R) − πX​((πX​(R) × S) − R)

Trong đó:

* + - * R(X, Y): quan hệ ban đầu chứa cặp giá trị (X, Y).
      * S(Y): quan hệ chứa tập giá trị Y mà ta muốn kiểm tra điều kiện “cho tất cả”.
      * T(X): kết quả phép chia, bao gồm các giá trị X thỏa mãn rằng với mọi Y trong S, cặp (X, Y) đều có trong R.
      * X là tập thuộc tính còn lại của R không có trong S

Ý nghĩa của từng phần trong công thức được giải thích như sau:

* + - * πX​(R) × S: tạo ra tất cả các cặp khả dĩ giữa X trong R và Y trong S
      * (πX(R) × S) − R: loại bỏ những cặp (X, Y) không thực sự tồn tại trong R
      * πX((πX(R) × S) − R): xác định các X thiếu ít nhất một Y trong S
      * Cuối cùng, πX− phần trên sẽ cho ra các X không bị thiếu bất kỳ Y nào trong S, tức là các X liên kết với toàn bộ các Y — đó chính là kết quả của phép chia.

Như vậy, phép chia có thể được xem là sự lọc các phần tử X đầy đủ, loại bỏ những phần tử không có quan hệ với toàn bộ các giá trị Y trong S.

**3.3.3. Ví dụ minh họa**

Xét ví dụ trong cơ sở dữ liệu quản lý nhân sự: “Tìm tên các nhân viên làm việc trong tất cả các dự án được quản lý bởi phòng ban số 5.”

Giả sử ta có các quan hệ sau:

* + - * WORKS\_ON(Essn, Pno): chứa thông tin về các nhân viên (Essn) và các dự án (Pno) mà họ đang tham gia
      * PROJECT(Pnumber, Dnum): chứa thông tin về mã dự án và phòng ban quản lý dự án đó.
* **EMPLOYEE(Ssn, Fname, Lname, Dno, …)**: chứa thông tin nhân viên.

**Bước 1:** Chọn tất cả các dự án do **phòng ban số 5** quản lý:

S←πPnumber​(σDnum=5​(PROJECT))

**Bước 2:** Xác định quan hệ **R**, thể hiện mối quan hệ giữa nhân viên và dự án mà họ tham gia:

R←WORKS\_ON(Essn,Pno)

**Bước 3:** Áp dụng **phép chia (÷)** để tìm ra các nhân viên làm việc trong **tất cả các dự án** thuộc **S**:

T(Essn)← R ÷ S

**Bước 4:** Liên kết kết quả với quan hệ **EMPLOYEE** để lấy tên nhân viên:

RESULT←πFname,Lname​(T ⋈ EMPLOYEE)

Kết quả RESULT sẽ chứa tên các nhân viên làm việc trong mọi dự án do phòng ban số 5 quản lý.

Qua ví dụ này, ta thấy phép chia (÷) đặc biệt hữu ích trong các truy vấn có chứa điều kiện “cho tất cả”, chẳng hạn như “làm việc trong mọi dự án”, “đăng ký mọi môn học”, v.v.  
Những dạng truy vấn này khó thể hiện bằng các phép toán đơn giản như chọn, chiếu, hoặc kết nối, nhưng có thể được biểu diễn ngắn gọn và rõ ràng bằng phép chia.

**Chương 4. ÁNH XẠ ĐẠI SỐ QUAN HỆ VÀO NGÔN NGỮ SQL**

## 4.1. Ánh xạ các toán tử cơ bản

Đại số Quan hệ (Relational Algebra – RA) cung cấp một tập hợp các phép toán được thiết kế để thao tác trên các quan hệ . Tính chất quan trọng nhất của Đại số Quan hệ là **tính đóng (closure)**: đầu vào của một phép toán là một hoặc hai quan hệ, và **kết quả đầu ra luôn là một quan hệ mới**. Nhờ tính chất này, các phép toán đại số quan hệ có thể được **kết hợp với nhau** **(composed)** để tạo thành các biểu thức, tương tự như việc kết hợp các phép toán số học như cộng, trừ, nhân, chia. Các biểu thức này cho phép định nghĩa một số lượng lớn các truy vấn cho cơ sở dữ liệu quan hệ.

Tập hợp các phép toán cơ bản (fundamental operations) trong Đại số Quan hệ bao gồm:

* Phép Chọn (Selection, *σ*)
* Phép Chiếu (Projection, *π*)
* Phép Tích Descartes (Cartesian product, ×)
* Phép Hợp (Union, ∪)
* Phép Trừ (Difference, − hoặc \)
* Phép Giao (Intersection, ∩)

Các phép toán này được phân loại dựa trên số lượng quan hệ đầu vào:

* **Các Toán tử Đơn nguyên (Unary Operations)**
* **Các Toán tử Nhị nguyên (Binary Operations)**

### 4.1.1. Ánh xạ của các phép chọn (σ) và phép chiếu (π)

Các phép toán chọn (*σ*) và chiếu (*π*) là các phép toán đơn nguyên (chỉ thao tác trên một quan hệ) và là những phép toán cơ bản nhất để thao tác dữ liệu quan hệ

**Phép chọn (*σ*)**

**Khái niệm**

* + **Mục đích phép chọn:** **Chọn các bộ (tuples/hàng)** thỏa mãn một điều kiện (vị từ - predicate) đã cho.
  + Phép Chọn hoạt động như một bộ lọc (filter), chỉ giữ lại các hàng thỏa mãn điều kiện. Nó có thể được hình dung là sự phân chia ngang (horizontal partition) của quan hệ.
  + **Kết quả:** Quan hệ mới chứa các bộ từ *R* thỏa mãn *C*. Quan hệ kết quả có lược đồ (số lượng thuộc tính) giống như quan hệ đầu vào *R*.
  + **Điều kiện C:** Là một biểu thức Boolean (logic) kết hợp các phép toán ∧ (AND), ∨ (OR), và ¬ (NOT) trên các điều kiện nguyên tử. Điều kiện nguyên tử có thể là so sánh giữa hai tên cột hoặc so sánh giữa tên cột với một giá trị hằng.
  + **Tính chất:** Phép Chọn có tính giao hoán: *σC*1​(*σC*2​(*R*)) = *σC*2​(*σC*1​(*R*)).

**Ánh xạ sang SQL**

Phép toán Chọn (*σ*) trong Đại số Quan hệ ánh xạ trực tiếp sang **mệnh đề WHERE** trong câu lệnh SELECT của SQL.

*σC*​(*R*) → **WHERE** *C* (trong câu lệnh SELECT)

* + Điều kiện *C* trong *σC*​(*R*) được đặt trong mệnh đề WHERE của SQL.
  + Trong SQL, các phép toán logic tương ứng là AND, OR, và NOT

**Phép chiếu (π)**

**Khái niệm:**

* + **Mục đích phép chiếu:** **Chọn các thuộc tính (attributes/cột)** cụ thể từ một quan hệ, loại bỏ các cột không cần thiết.
  + Phép Chiếu hoạt động như một sự phân chia dọc (vertical partition) của quan hệ.
  + **Ký hiệu:** *π*A​(*R*), trong đó *R* là quan hệ đầu vào và *A* là tập hợp các thuộc tính cần chiếu (các cột cần hiển thị).
  + **Tính chất về trùng lặp:** Vì trong mô hình quan hệ toán học, một quan hệ là một **tập hợp** các bộ (set of tuples), phép Chiếu **tự động loại bỏ các bộ trùng lặp**.
  + **Mở rộng:** Phép Chiếu có thể được mở rộng thành **Phép Chiếu Tổng quát (Generalized Projection)** để cho phép sử dụng các biểu thức số học hoặc hàm trên các thuộc tính trong danh sách chiếu (ví dụ: *πTENNV*,(*HSL*+*HSCV*)×*LUONG*\_*CB*​ (*BANGLUONG*))

**Ánh xạ sang SQL**

Phép Chiếu (*π*) trong Đại số Quan hệ ánh xạ trực tiếp sang **mệnh đề SELECT** trong SQL:

*πA*​(*R*) → **SELECT DISTINCT** *A* (hoặc **SELECT** *A*)

* + Khi ánh xạ *π* (của Đại số Quan hệ thuần túy, vốn tự động loại bỏ trùng lặp) sang SQL, cần sử dụng từ khóa **DISTINCT** trong mệnh đề SELECT để đảm bảo kết quả phù hợp với định nghĩa tập hợp (set version) của Đại số Quan hệ.
  + Nếu từ khóa **DISTINCT** bị bỏ qua, SQL sẽ coi các bảng là đa tập hợp (multiset), cho phép các bộ trùng lặp xuất hiện trong kết quả

**Một vài ví dụ chuyển đổi từ ĐSQH sang SQL**

VD: Cho quan hệ *INSTRUCTOR* có lược đồ:

***INSTRUCTOR***(***ID***, *name*, *dept*\_*name*, *salary*).

* **Truy vấn 1: Tìm ID, tên và lương của tất cả các giảng viên.**

**ĐSQH | SQL**

Hình 4. 1: Tìm thông tin của gv

* **Truy vấn 2: Tìm tất cả các giảng viên thuộc khoa "Physics".**

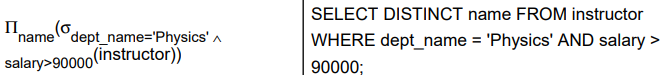
**ĐSQH | SQL**

Hình 4. 2: Tìm gv khoa vật lí

* **Truy vấn 3: Tìm tên của các giảng viên thuộc khoa "Physics" có mức lương lớn hơn $90000.**

Bài toán này đòi hỏi sự kết hợp của Phép Chọn và Phép Chiếu.

Phép Chọn được áp dụng trước để giới hạn số lượng bộ (hàng), sau đó Phép Chiếu được áp dụng để chọn các thuộc tính (cột) mong muốn:

**ĐSQH | SQL**

Hình 4. 3: Tìm gv khoa vật lí có lương > 90000$

### 4.1.2. Ánh xạ toán tử tập hợp ( ∪, \ , ∩ )

Trước khi thực hiện bất kỳ phép toán tập hợp cơ bản nào (Hợp, Giao, Trừ), các quan hệ tham gia phải thỏa mãn điều kiện **Tương thích Tập hợp** (Union Compatibility).

Các điều kiện này bao gồm:

* **Cùng số lượng thuộc tính (Arity):** Hai quan hệ phải có cùng số lượng cột.
* **Tương thích miền giá trị (Domain Compatibility):** Các thuộc tính tương ứng (thứ *i*) trong hai quan hệ phải được lấy từ cùng một miền giá trị (domain).

Sự tương thích này đảm bảo rằng kết quả của phép toán cũng là một quan hệ hợp lệ

**Phép hợp (∪)**

**Khái niệm**

Phép Hợp (∪, ký hiệu ĐSQH: *R* ∪ *S*) tạo ra một quan hệ mới bao gồm tất cả các bộ (hàng) có trong quan hệ *R*, hoặc quan hệ *S*, hoặc cả hai.

**Trong SQL, phép Hợp được ánh xạ trực tiếp sang từ khóa UNION**.

**Đặc điểm quan trọng:**

* + Phép toán UNION trong SQL mặc định **tự động loại bỏ các bộ trùng lặp**, phù hợp với định nghĩa toán học rằng một quan hệ là một tập hợp các bộ.
  + Nếu muốn giữ lại tất cả các bộ trùng lặp (tức là làm việc với multisets), SQL có từ khóa UNION ALL. Trong trường hợp sử dụng UNION ALL, số lượng bản sao của một bộ trong kết quả bằng tổng số bản sao của bộ đó trong hai quan hệ đầu vào.
  + Về mặt Đại số, phép toán Hợp có tính chất **giao hoán** (*R* ∪ *S* ≡ *S* ∪ *R*) và **kết hợp**

VD: Tính giao hoán và kết hợp của phép hợp

* *(E1 ∪ E2) ∪ E3 ≡ E1 ∪ (E2 ∪ E3)*
* *(E1 ∩ E2) ∩ E3 ≡ E1 ∩ (E2 ∩ E3)*

**Ánh xạ sang SQL**

*R* ∪ *S* → **SELECT \* FROM** R **UNION SELECT \* FROM** S**;**.

**Lưu ý:** **Yêu cầu** thỏa mãn điều kiện **Tương thích Tập hợp:** Cả hai truy vấn con (hai vế của UNION) phải tương thích tập hợp (cùng số lượng và loại thuộc tính).

**Phép trừ (− hay ∖)**

**Khái niệm**

Phép Trừ (Set Difference, ký hiệu ĐSQH: *R* − *S* hoặc *R* ∖ *S*) tạo ra một quan hệ mới chứa các bộ có mặt trong *R* nhưng **không** có mặt trong *S*.

**Trong SQL, phép Trừ được ánh xạ sang từ khóa EXCEPT**.

* + Lưu ý: Một số hệ thống quản trị cơ sở dữ liệu (DBMS) khác như Oracle hoặc MS SQLServer sử dụng từ khóa ***MINUS*** thay cho EXCEPT.
  + **Phép trừ không có tính giao hoán (*R* − *S* ≠ *S* − *R*).**

**Các cách biểu diễn thay thế phổ biến trong thực tế:**

Phép Trừ thường được mô phỏng bằng cách sử dụng các phép toán truy vấn con (subqueries) hoặc phép kết ngoài (Outer Join), đặc biệt là trong các hệ thống không hỗ trợ trực tiếp EXCEPT.

* **Sử dụng NOT IN:** Dùng mệnh đề WHERE và toán tử NOT IN để loại trừ các bộ trong *R* mà tồn tại trong *S*.
* **Sử dụng NOT EXISTS:** Dùng mệnh đề WHERE và toán tử NOT EXISTS với một truy vấn con trên *S* để tìm các bộ trong *R* không có bộ nào khớp trong *S*. Phương pháp này được sử dụng để mô phỏng sự khác biệt tập hợp.
* **Sử dụng Phép kết ngoài trái (Left Outer Join) loại trừ:** *R* − *S* có thể được ánh xạ thành phép kết ngoài trái *R* LEFT JOIN *S* nhưng **chỉ giữ lại** các bộ trong *R* mà không có đối sánh nào trong *S*. Điều này được thực hiện bằng cách thêm điều kiện WHERE để kiểm tra rằng một thuộc tính khóa trong *S* là NULL (ví dụ: **WHERE** B.a1 **IS NULL**). Trong tài liệu, đây được gọi là Left Join Excluding (LJ(E)).

**Ánh xạ sang SQL**

*R* – *S hay (R \ S)* → **SELECT \* FROM** R **EXCEPT SELECT \* FROM** S**;**.

**Lưu ý chi tiết về phương pháp trên :**

* **Multiset Difference:** Để giữ lại các bản sao (multiset), SQL sử dụng toán tử **EXCEPT ALL**. Số lượng bản sao của một bộ trong kết quả bằng số bản sao của bộ đó trong *R* trừ đi số bản sao của bộ đó trong *S*, miễn là hiệu số đó dương.
* **Hạn chế của DBMS:** MySQL không hỗ trợ toán tử EXCEPT.

**Các cách ánh xạ khác:**

Vì EXCEPT không được hỗ trợ phổ biến, phép trừ thường được mô phỏng bằng

cách sử dụng các phép toán Join hoặc truy vấn con như những phương pháp sau:

* **Sử dụng NOT IN (Truy vấn con):**

*R* − *S* → **SELECT \* FROM** R

**WHERE** (A1, A2, ...)

**NOT IN (SELECT** A1, A2, ... **FROM** S**);**

Áp dụng khi hai quan hệ chỉ có một hoặc một vài thuộc tính tương ứng cần kiểm tra.

* **Sử dụng NOT EXISTS (Truy vấn con tương quan):**

*R* − *S* → **SELECT** R.\* **FROM** R

**WHERE NOT EXISTS (SELECT \* FROM** S

**WHERE** R.Key = S.Key

**AND** R.A1 = S.A1 **...);**.

Phương pháp này mô phỏng phép trừ bằng cách tìm các bộ trong *R* không có bộ nào khớp trong *S*.

* **Sử dụng Left Join Loại trừ (LJ(E)):**

*R* − *S* → **SELECT** R.\* **FROM** R

**LEFT JOIN** S **ON** R.Key = S.Key

**WHERE** S.Key **IS NULL;**.

Phương pháp này được gọi là Left Join Excluding (LJ(E)). Truy vấn này giữ lại tất cả các bộ từ *R* không tìm thấy đối sánh nào trong *S*, bằng cách kiểm tra giá trị NULL trong một thuộc tính khóa của *S*. Phương pháp này cũng được sử dụng để ánh xạ phép toán Trừ D (A - B).

**Phép Giao (∩)**

**Khái niệm**

Phép Giao (∩, ký hiệu ĐSQH: *R* ∩ *S*) tạo ra một quan hệ mới bao gồm các bộ có mặt trong **cả** *R* và *S*.

**Trong SQL, phép Giao được ánh xạ trực tiếp sang từ khóa INTERSECT**.

* + Phép **INTERSECT** cũng tự động loại bỏ các bản sao. Tương tự như UNION, phiên bản **INTERSECT** **ALL** tồn tại trong một số hệ thống để giữ lại các bản sao (multisets).
  + Phép Giao có thể được tính toán chỉ bằng các phép Trừ: *R* ∩ *S* = *R* − (*R* − *S*).

**Các cách biểu diễn thay thế hiệu quả và phổ biến:**

Phép Giao thường được biểu diễn bằng các phép toán kết (Join) hoặc truy vấn con vì không phải tất cả các hệ thống đều hỗ trợ INTERSECT (chẳng hạn như MySQL không hỗ trợ INTERSECT và EXCEPT).

* **Sử dụng Phép kết trong (Inner Join):** Đây là phương pháp hiệu quả nhất để tìm giao của hai quan hệ dựa trên tất cả các thuộc tính. Cụ thể:

*R* ∩ *S* ≡ *R* **INNER JOIN** *S*

* **Sử dụng IN:** Dùng mệnh đề WHERE và toán tử IN để kiểm tra các bộ trong *R* có tồn tại trong *S*.
* **Sử dụng EXISTS:** Dùng mệnh đề WHERE EXISTS với một truy vấn con trên *S* để tìm các bộ trong *R* có đối sánh trong *S*

**Ánh xạ sang SQL**

*R* ∩ *S* → **SELECT \* FROM** R **INTERSECT SELECT \* FROM** S**;**

**Lưu ý chi tiết về cách trên:**

* **Giữ lại trùng lặp (Multiset semantics):** Để giữ lại các bản sao, SQL sử dụng toán tử **INTERSECT ALL**. Số lượng bản sao của một bộ trong kết quả bằng số lượng bản sao tối thiểu của bộ đó trong *R* và *S*.
* **Hạn chế :** MySQL không triển khai toán tử INTERSECT.

**Các phương pháp ánh xạ thay thế:**

Phép Giao là phép toán không cơ bản và có thể được biểu diễn bằng các phép toán khác trong ĐSQH, ví dụ: *R* ∩ *S* = *R* − (*R* − *S*). Trong SQL, nó thường được mô phỏng bằng các phép kết hoặc truy vấn con vì tính hiệu quả cao hơn.

* **Sử dụng Inner Join (IJ) (Phương pháp hiệu quả):**

*R* ∩ *S* → **SELECT** R.\* **FROM** R

**INNER JOIN** S **ON** R.A = S.A **AND** R.B = S.B **...;**.

Phép Join trong (Inner Join) dựa trên tất cả các thuộc tính chung (khi các bảng tương thích tập hợp) tạo ra kết quả giống hệt với phép Giao. Trong thực tế, người ta hay sử dụng Inner Join để tìm Intersection vì nó **nhanh hơn** so với Outer Join có điều kiện.

* **Sử dụng EXISTS (Truy vấn con tương quan):**

*R* ∩ *S* → **SELECT** R.\* **FROM** R

**WHERE EXISTS (SELECT \* FROM** S

**WHERE** R.A = S.A **AND** R.B = S.B **...);**.

Phương pháp này kiểm tra sự tồn tại của một bộ khớp hoàn toàn trong *S* đối với mỗi bộ trong *R*.

* **Sử dụng IN (Truy vấn con):**

*R* ∩ *S* → **SELECT DISTINCT** A **FROM** R

**WHERE** A **IN (SELECT** A **FROM** S**);**.

Áp dụng khi chỉ kiểm tra một thuộc tính hoặc một tập hợp các thuộc tính cụ thể. Toán tử = SOME là tương đương với IN.

* **Sử dụng Left/Right Outer Join Bao gồm (LJ(I)/RJ(I)):**

*R* ∩ *S* → **SELECT** R.\* **FROM** R

**LEFT JOIN** S **ON** R.Key = S.Key

**WHERE** S.Key **IS NOT NULL;**.

Phương pháp này còn được gọi là Left Join Including (LJ(I)). Cách này ánh xạ phép toán Giao (*I*) bằng cách sử dụng Left Join và chỉ giữ lại các bộ trong *R* mà tìm thấy đối sánh trong *S*, kiểm tra bằng điều kiện một thuộc tính khóa của *S* **IS NOT NULL**. Tuy nhiên, các nguồn tin nhấn mạnh rằng Inner Join (IJ) là phương pháp hiệu quả hơn để tìm Intersection

**Ánh xạ Toán tử Tập hợp trong Đại số Quan hệ vào SQL**

Các toán tử tập hợp cơ bản (∪, ∩, ∖) yêu cầu các quan hệ đầu vào phải thỏa mãn Tính Tương thích Tập hợp (*Union Compatibility*), các quan hệ phải có cùng số lượng thuộc tính (*arity*) và các thuộc tính tương ứng phải có cùng miền giá trị.

Chúng ta sẽ sử dụng hai quan hệ giả định C (Current Records) và N (New Records), với lược đồ như sau:

Lược đồ Quan hệ:

• **C** (**PID, SID**, Cost): Bản ghi sản phẩm hiện tại (ID Sản phẩm, ID Nhà cung cấp, Chi phí).

• **N** (**PID, SID**, Cost): Bản ghi sản phẩm mới.

Các phép toán tập hợp (∪,∩,∖) đòi hỏi các quan hệ tham gia phải **tương thích tập hợp** (*Union Compatibility*). Sự tương thích này yêu cầu chúng phải có cùng số lượng thuộc tính (arity) và các thuộc tính tương ứng phải có cùng miền giá trị (domain compatibility).

**Ánh xạ Phép Hợp (∪) → UNION**

Phép toán Hợp (*C* ∪ *N*) tìm ra tất cả các bản ghi tồn tại trong các bảng được truy vấn.

**Yêu cầu**: Tìm danh sách tất cả các sản phẩm hiện có và sản phẩm mới (Discontinued + New + Common).

**Phương pháp Chuẩn (Loại bỏ Trùng lặp):**

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *C* ∪ *N* | **SELECT \* FROM** C **UNION SELECT \* FROM** N; |

Bảng 4. 1: Bảng VD phương pháp chuẩn phép hợp

**Phương pháp Multiset (Giữ lại Trùng lặp):**

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *C* ∪all ​*N* | **SELECT \* FROM** C **UNION ALL SELECT \* FROM** N; |

Bảng 4. 2: Bảng VD phương pháp lấy phần trùng lặp của phép hợp

**Lưu ý:** Số lượng bản sao của một bộ trong kết quả bằng tổng số lần xuất hiện của bộ đó trong C và N

**Ánh xạ Phép Giao (∩) → INTERSECT**

Phép toán Giao (*C* ∩ *N*) tìm các bản ghi sản phẩm/nhà cung cấp/chi phí **chung** (common records) có mặt trong tất cả các bảng được truy vấn.

**Yêu cầu:** Tạo ra một quan hệ mới chứa các bộ có mặt trong cả C và N

**Phương pháp Chuẩn (INTERSECT):**

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *C* ∩ *N* | **SELECT \* FROM** C **INTERSECT SELECT \* FROM** N; |

Bảng 4. 3: VD phương pháp chuẩn của phép giao

*Lưu ý:* Toán tử INTERSECT **tự động loại bỏ trùng lặp**. Tuy nhiên, **MySQL không hỗ trợ** toán tử này.

**Phương pháp thay thế hiệu quả: Sử dụng INNER JOIN (IJ)**

Phép Giao có thể được ánh xạ bằng phép **Kết Trong (Inner Join)** trên *tất cả các thuộc tính* chung. Phương pháp này **thường hiệu quả hơn** so với việc sử dụng Outer Join có điều kiện.

ĐSQH: *C* ∩ *N* ≡ *πC*.∗​(*C*⋈all attributes​*N*)

SQL:

**SELECT** C.\* **FROM** C

**INNER JOIN** N **ON** (C.PID = N.PID)

**AND** (C.SID = N.SID)

**AND** (C.Cost = N.Cost);

**Lưu ý:** Truy vấn này tìm ra các bộ dữ liệu khớp nhau hoàn toàn trên tất cả các thuộc tính định danh (**PID, SID**, Cost).

**Phương pháp truy vấn con tương quan : Sử dụng EXISTS**

Phương pháp này kiểm tra liệu có tồn tại bộ khớp hoàn toàn trong *N* cho mỗi bộ trong *C* hay không.

ĐSQH: *C* ∩ *N* ≡ *σ*∃ *n*∈*N* (*C* = *n*)​(*C*)

SQL:

**SELECT** \* **FROM C**

**WHERE EXISTS** (**SELECT** \* **FROM** N

**WHERE** (C.PID = N.PID)

**AND** (C.SID = N.SID)

**AND** (C.Cost = N.Cost););

Phương pháp này được coi là một truy vấn chuyên biệt (specialized query) và **không hiệu quả** bằng phương pháp INNER JOIN .

**Phương pháp sử dụng LEFT JOIN INCLUDING (LJ(I))**

Phép Giao (*I*) có thể được ánh xạ bằng Left Join Including (LJ(I)). Điều này đòi hỏi phải sử dụng điều kiện WHERE ... IS NOT NULL (Including Predicate) trên một thuộc tính khóa của bảng bên phải (*N*) để giữ lại các bản ghi khớp.

ĐSQH: *I* = *C* ∩ *N =* πC.∗​(C ⋈ C.PID = N.PID ∧ C.SID = N.SID ∧ C.Cost = N.Cost ​N)

SQL:

**SELECT** C.\* **FROM** C

**LEFT JOIN** N **ON** (C.PID = N.PID)

**AND** (C.SID = N.SID) **AND** (C.Cost = N.Cost)

**WHERE** (N.PID) **IS NOT NULL**;

**Lưu ý**: Điều kiện **WHERE** (N.PID) **IS NOT NULL** là điều kiện "Bao gồm" (Including, ký hiệu LJ(I)) nhằm đảm bảo chỉ các bộ từ C có đối sánh trong N mới được giữ lại

**Ánh xạ Phép Hiệu (∖) → EXCEPT**

Phép toán Hiệu (*C* ∖ *N*) tìm các bản ghi **chỉ có trong** *C* mà **không có** trong *N*. **Yêu cầu:** Tìm bản ghi đã ngừng sản xuất (Discontinued Records).

**Phương pháp chuẩn: Sử dụng EXCEPT/MINUS**

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *C* ∖ *N* | **SELECT** \* **FROM** C **EXCEPT SELECT** \* **FROM** N; |

Bảng 4. 4: VD phương pháp chuẩn của phép hiệu

*Lưu ý:* Oracle sử dụng từ khóa **MINUS** thay cho EXCEPT. Cả hai toán tử này đều yêu cầu **tính tương thích tập hợp**. Tuy nhiên, **MySQL không hỗ trợ** toán tử này.

**Phương pháp ưu tiên phổ biến: Sử dụng LEFT JOIN EXCLUDING (LJ(E))**

Phép Hiệu *C* \ *N* được ánh xạ trực tiếp bằng cách sử dụng **Left Join Excluding (LJ(E))**. Phương pháp này giữ lại tất cả các bộ từ *C* không tìm thấy đối sánh trong *N*.

ĐSQH: *C* \ *N* = πC.\*(σN.PID**IS NULL**(C ⋉ N))

SQL:

**SELECT** C.\* **FROM** C

**LEFT JOIN** N **ON** (C.PID = N.PID)

**AND** (C.SID = N.SID) **AND** (C.Cost = N.Cost)

**WHERE** (N.PID) **IS NULL**;

*Lưu ý quan trọng:* Điều kiện **WHERE** (N.PID) **Is Null** là predicate "Excluding" (E). Ta **chỉ cần kiểm tra một thuộc tính khóa** (ví dụ: PID) của bảng bên phải (*N*) là đủ, vì nếu thuộc tính khóa này là NULL, thì toàn bộ bộ đó không có đối sánh.

**Yêu cầu:** Tìm các bản ghi **chỉ có trong N mà không có trong C** (*N* ∖ *C*)

**Phương pháp Right Join : Tìm các sản phẩm mới (New Records)**

Phép toán Hiệu *N* \ *C*(các sản phẩm có trong *N* nhưng không có trong *C*, tức là Newly Available) là đối xứng của *C* \ *N* và được ánh xạ bằng **Right Join Excluding (RJ(E))**.

ĐSQH: *N \ C* = πN.\*(σC.PID**IS NULL**(N ⋉ C))

**SELECT** N.\* **FROM** C

**RIGHT JOIN** N **ON** (C.PID = N.PID)

**AND** (C.SID = N.SID) **AND** (C.Cost = N.Cost)

**WHERE** (C.PID) **IS NULL**;

Lưu ý:

Sử dụng RIGHT JOIN để giữ lại tất cả các bộ từ bảng bên phải (N).

Điều kiện WHERE (C.PID) IS NULL loại trừ những bộ từ N có đối sánh trong C, chỉ giữ lại các bộ độc nhất trong N.

**Phương pháp thay thế dùng NOT EXISTS:**

ĐSQH:

SQL:

**SELECT** C.\* **FROM** C

**WHERE NOT EXISTS** (**SELECT** \* **FROM** N

**WHERE** C.PID = N.PID

**AND** C.SID = N.SID

**AND** C.Cost = N.Cost);

### 4.1.3. Ánh xạ tích Descartes ( × )

**Khái niệm:**

Phép Tích Đề-các (*R* × *S*) là một phép toán nhị nguyên tạo ra một quan hệ mới bằng cách **ghép nối mọi bộ (tuple) của quan hệ** *R* **với mọi bộ của quan hệ** *S*.

• **Lược đồ kết quả:** Lược đồ của quan hệ kết quả là sự ghép nối của các lược đồ *R* và *S*.

• **Số lượng bộ:** Nếu *R* có *nR*​ bộ và *S* có *nS*​ bộ, quan hệ kết quả *R*×*S* sẽ trả về *nR*​× *nS*​ bộ kết quả.

• **Thuộc tính trùng lặp:** Nếu *R* và *S* có các thuộc tính chung tên (ví dụ: thuộc tính *B*), các thuộc tính này trong kết quả Tích Đề-các sẽ được phân biệt bằng cách sử dụng tiền tố tên quan hệ (ví dụ: *R*.*B* và *S*.*B*).

**Ánh xạ sang SQL:**

Phép Tích Đề-các được ánh xạ sang SQL bằng cách **liệt kê nhiều bảng trong mệnh đề FROM mà không có mệnh đề WHERE để giới hạn các bộ kết hợp**.

Nói cách khác, **mệnh đề FROM tự thân nó đã định nghĩa một Tích Đề-các** của các quan hệ được liệt kê.

**Liệt kê trong FROM (dùng dấu phẩy):**

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| R × S | **SELECT \* FROM** R, S**;** |

Bảng 4. 5: Ánh xạ SQL dùng dấu phẩy

**Liệt kê trong FROM (dùng CROSS JOIN):**

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| R × S | **SELECT \* FROM** R  **CROSS JOIN** S; |

Bảng 4. 6: Ánh xạ SQL dùng CROSS JOIN

**Ví dụ minh họa Tích Đề-các**Chúng ta sử dụng hai quan hệ giả định: **NHANVIEN** và **DUAN**.

**Lược đồ Quan hệ:**

• **NHANVIEN (MSNV, TenNV, Luong)**

• **DUAN (MSDA, TenDA, DiaDiem)**

**Yêu cầu:** Tạo ra tất cả các kết hợp có thể có giữa mọi nhân viên và mọi dự án (ví dụ, nếu có 10 nhân viên và 5 dự án, kết quả có 50 bộ).

**Phương pháp chuẩn của Tích Descartes**

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| NHANVIEN × DUAN | **SELECT** \* **FROM** NHANVIEN, DUAN; |
|  | **SELECT** \* **FROM** NHANVIEN **CROSS** **JOIN** DUAN; |

Bảng 4. 7: Phương pháp ánh xạ chuẩn của tích Descartes

**Giải thích & Lưu ý:**

• Kết quả chuẩn của Tích Đề-các thường là **vô nghĩa** trong ngữ cảnh truy vấn thông thường vì nó ghép nối cả các bộ không liên quan.

• Nếu không có mệnh đề WHERE để giới hạn, kết quả có thể là một quan hệ rất lớn.

• **Tên thuộc tính:** Kết quả chứa tất cả các thuộc tính của NHANVIEN (MSNV, TenNV, Luong) và DUAN (MSDA, TenDA, DiaDiem). Vì không có thuộc tính trùng tên trong ví dụ này, không cần dùng tiền tố.

**Lược đồ Quan hệ:**

• **NHANVIEN (ID, TenNV, MaDV)**

• **DONVI (MaDV, TenDV, DiaDiem)** (Lưu ý: MaDV là thuộc tính chung)

**Yêu cầu:** Tìm Tên nhân viên và Tên đơn vị mà họ làm việc.

**Phương pháp kết hợp với Phép Chọn (× và *σ*):**

Tích Đề-các thường được sử dụng ngay lập tức sau đó là một phép Chọn (*σ*) để giới hạn các bộ khớp theo một điều kiện nào đó. Sự kết hợp này chính là định nghĩa cơ bản của phép **Theta Join**: **R** ⋈**C** ​**S** ≡ *σ***C**​(**R** × **S**).

Chúng ta sử dụng hai quan hệ có thuộc tính chung: **NHANVIEN** và **DONVI**.

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *π*TenNV,TenDV​(*σ*NHANVIEN.MaDV = DONVI.MaDV​(NHANVIEN × DONVI)) | **SELECT** T.TenNV, D.TenDV  **FROM** NHANVIEN  **AS** T, DONVI **AS** D  **WHERE** T.MaDV = D.MaDV; |

Bảng 4. 8: VD cho việc kết hợp Tích Descartes với Phép Chọn

**Giải thích & Ứng dụng:**

1. **Mệnh đề FROM**: Tạo ra Tích Đề-các giữa NHANVIEN và DONVI (ghép nối mọi bộ nhân viên với mọi bộ đơn vị).

2. **Mệnh đề WHERE**: Thực hiện phép Chọn (*σ*) trên kết quả Tích Đề-các, chỉ giữ lại những bộ thỏa mãn điều kiện kết nối (T.MaDV = D.MaDV).

3. **Mệnh đề SELECT**: Thực hiện phép Chiếu (*π*) để chọn các cột mong muốn (TenNV và TenDV).

**Lưu ý quan trọng (Khi nào nên dùng):**

• **Mô phỏng Join:** Đây là cách nguyên thủy để thực hiện phép Kết Nội (*Inner Join*) hoặc *Theta Join* trong SQL, đặc biệt khi JOIN không được hỗ trợ hoặc để hiểu rõ cơ chế hoạt động bên dưới.

• **Ưu tiên thực tế:** Trong SQL hiện đại, **nên sử dụng cú pháp INNER JOIN ... ON** (hoặc JOIN ... ON) thay vì Tích Đề-các kết hợp với WHERE. Cú pháp JOIN rõ ràng hơn, giúp phân biệt giữa điều kiện kết nối và điều kiện lọc thông thường, và thường được trình tối ưu hóa truy vấn (Query Optimizer) xử lý hiệu quả hơn.

## 4.2. Ánh xạ các phép kết nối

Phép Kết nối là một phép toán không cơ bản, vì nó có thể được biểu diễn thông qua sự kết hợp của phép Tích Descartes (×) và phép Chọn (*σ*).

Các phép Kết nối được phân loại thành hai nhóm chính trong SQL: Kết Nội (Inner Join) và Kết Ngoài (Outer Join)

### 4.2.1. Ánh xạ Natural Join ( ⋈ )

**Khái niệm và Đặc điểm**

Phép Kết Tự nhiên (Natural Join, ký hiệu ⋈) là một dạng đặc biệt của phép Kết Bằng (Equijoin).

* **Cơ chế hoạt động:** Natural Join hoạt động trên hai quan hệ *R* và *S* bằng cách tìm tất cả các cặp bộ, một từ *R* và một từ *S*, có giá trị bằng nhau trên **tất cả các thuộc tính có tên chung** xuất hiện trong lược đồ của cả hai quan hệ.
* **Lược đồ kết quả:** Phép toán này loại bỏ các thuộc tính bị trùng lặp trong kết quả. Lược đồ kết quả là tập hợp các thuộc tính của *R* hợp với tập hợp các thuộc tính của *S*, nhưng các thuộc tính chung chỉ xuất hiện một lần.
* **Trường hợp đặc biệt:** Nếu hai quan hệ không có thuộc tính chung, Natural Join sẽ thoái hóa thành phép Tích Descartes (*R*⋈*S* = *R*×*S*).

**Ánh xạ sang SQL**

* **Sử dụng từ khóa NATURAL JOIN:**

Phép Natural Join (⋈) được ánh xạ trực tiếp sang SQL bằng cú pháp NATURAL JOIN.

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *R* ⋈ *S* | **SELECT** \* **FROM** R  **NATURAL** **JOIN** S; |

Bảng 4. 9: Phương pháp ánh xạ NATURAL JOIN

* **Sử dụng INNER JOIN với USING:**

Để cung cấp lợi ích của Natural Join nhưng tránh rủi ro tự động khớp nối các thuộc tính có tên trùng lặp một cách không mong muốn (ví dụ: khi thêm thuộc tính mới vào lược đồ, ngữ nghĩa truy vấn có thể thay đổi), SQL cung cấp cú pháp JOIN ... USING.

JOIN ... USING (A1, A2, ...) cho phép chỉ định chính xác các cột cần được khớp nối. Hai quan hệ được nối phải có các thuộc tính với tên được chỉ định

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *R*⋈{*A*1​,*A*2​}​*S* | **SELECT** \* **FROM** R **JOIN** S **USING** (A1, A2); |

Bảng 4. 10: : Phương pháp chỉ định cột cần ánh xạ

* **Sử dụng INNER JOIN với ON (Theta Join/Equijoin)**

Chúng ta có thể ánh xạ một Natural Join thông qua một Equijoin (Phép Kết Bằng, tức là Theta Join với điều kiện chỉ dùng phép “ = ”) bằng cách chỉ định điều kiện khớp nối trên tất cả các thuộc tính chung và sau đó chỉ chiếu các thuộc tính mong muốn. Cú pháp **JOIN ... ON** hoặc **INNER JOIN ... ON** cho phép chỉ định một vị từ tùy ý (C) trên các quan hệ được nối

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *R* ⋈C *S* | **SELECT \* FROM** R  **JOIN** S **ON** C; |

Bảng 4. 11 : Phương pháp ánh xạ Equijoin

*Lưu ý:* Khi sử dụng JOIN ... ON, kết quả có thể chứa hai thuộc tính khớp nối (ví dụ: student.ID và takes.ID), khác với NATURAL JOIN chỉ giữ lại một. Tuy nhiên, một truy vấn sử dụng JOIN ... ON có thể được thay thế bằng một biểu thức tương đương không có ON, trong đó vị từ được chuyển sang mệnh đề WHERE

### 4.2.2. Ánh xạ Outer Join ( ⋉, ⋊,⪥ )

**Khái niệm và Mục đích**

Phép Kết Ngoài (Outer Join) được phát triển để giải quyết vấn đề **mất thông tin** xảy ra trong các phép Kết Nội (Inner Join). Trong một Inner Join, chỉ những bộ có đối sánh (matching tuples) mới được giữ lại trong kết quả. Nếu một bộ ở một trong hai quan hệ không có bộ đối sánh nào ở quan hệ kia, bộ đó sẽ bị "mất" khỏi kết quả.

Outer Join hoạt động tương tự như các phép Kết Nội, nhưng nó **bảo toàn các bộ bị mất** bằng cách tạo các bộ trong kết quả chứa **giá trị NULL** cho các thuộc tính bị thiếu.

**Có ba dạng Outer Join chính:**

**1. Left Outer Join (**R ⋉C S**):**

• **Mục đích:** Bảo toàn các bộ chỉ trong quan hệ được đặt tên trước (bên trái) phép toán.

• **Cơ chế hoạt động:** Đầu tiên, tính kết quả của Inner Join. Sau đó, với mỗi bộ *t* trong quan hệ bên trái không khớp với bất kỳ bộ nào trong quan hệ bên phải, một bộ *r* mới được thêm vào kết quả. Các thuộc tính của *r* có nguồn gốc từ quan hệ bên trái sẽ được điền bằng giá trị của *t*, và **các thuộc tính còn lại (từ quan hệ bên phải) được điền bằng giá trị NULL**.

• **Ánh xạ sang SQL:** Được biểu diễn bằng LEFT JOIN hoặc LEFT OUTER JOIN.

**2. Right Outer Join (**R ⋊C S**):**

• **Mục đích:** Đối xứng với Left Outer Join. Bảo toàn các bộ chỉ trong quan hệ được đặt tên sau (bên phải) phép toán.

• **Cơ chế hoạt động:** Các bộ từ quan hệ bên phải không khớp với bất kỳ bộ nào từ quan hệ bên trái sẽ được đệm bằng NULL và thêm vào kết quả của Right Outer Join.

• **Ánh xạ sang SQL:** Được biểu diễn bằng RIGHT JOIN hoặc RIGHT OUTER JOIN.

**3. Full Outer Join (**R ⪥ C S hoặc (R ⋉C S)∪( R ⋊ C S)**):**

• **Mục đích:** Bảo toàn các bộ trong **cả hai quan hệ**.

• **Cơ chế hoạt động:** Full Outer Join là sự kết hợp (hợp) của Left Outer Join và Right Outer Join tương ứng. Các bộ từ quan hệ bên trái không khớp với bên phải được đệm NULL ở phía bên phải, và các bộ từ quan hệ bên phải không khớp với bên trái được đệm NULL ở phía bên trái, sau đó tất cả được hợp lại

• **Ánh xạ sang SQL:** Được biểu diễn bằng FULL JOIN hoặc FULL OUTER JOIN.

Mệnh đề ON và WHERE trong Outer Join

Trong Outer Join, điều kiện được chỉ định trong mệnh đề ON và WHERE có hành vi khác nhau.

• **Mệnh đề ON:** Là một phần của đặc tả Outer Join. Nó xác định các bộ nào khớp để tạo ra Inner Join, và do đó, xác định các bộ nào sẽ được đệm NULL và thêm vào kết quả.

• **Mệnh đề WHERE:** Được áp dụng sau khi Outer Join đã được tính toán và các bộ đệm NULL đã được thêm vào. Mệnh đề WHERE lọc kết quả cuối cùng, bao gồm cả các bộ có giá trị NULL.

Ví dụ, nếu sử dụng LEFT OUTER JOIN ON C và sau đó thêm điều kiện WHERE C, kết quả có thể khác hoàn toàn so với việc sử dụng INNER JOIN WHERE C, vì các bộ được đệm NULL do Outer Join có thể bị loại bỏ bởi mệnh đề WHERE sau đó.

Các dạng kết nối và điều kiện kết nối khác nhau trong SQL có thể được tóm tắt như sau: bất kỳ hình thức kết nối nào (Inner, Left Outer, Right Outer, hoặc Full Outer) đều có thể được kết hợp với bất kỳ điều kiện kết nối nào (NATURAL, USING, hoặc ON).

**Ví dụ tổng hợp: Chuyển đổi các dạng Join khác nhau sang SQL**

Chúng ta sử dụng hai quan hệ giả định:

**STUDENT** (**ID**, name, dept\_name, tot\_cred) và

**TAKES** (**ID**, course\_id, sec\_id, semester, year, grade). Thuộc tính chung là ID.

**Yêu cầu:** Tìm tên sinh viên và mã khóa học của tất cả các khóa học mà SV đã học. (Chỉ bao gồm sinh viên đã học khóa học).

**Dùng Natural Join (Kết Tự nhiên)**

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *πname, course\_id​ (STUDENT ⋈ TAKES)* | **SELECT** name, course\_id  **FROM** STUDENT  **NATURAL JOIN** TAKES**;** |

Bảng 4. 12: Ánh xạ Natural Join

**Giải thích:** Phép toán này hoạt động bằng cách tự động tìm các bộ có giá trị bằng nhau trên **tất cả các thuộc tính có tên chung** (trong ví dụ này là ID).

**Lưu ý về cú pháp thay thế:** Trong thực tế, ta nên sử dụng cú pháp **JOIN ... USING (ID)** để chỉ định rõ ràng các cột cần khớp nối. Phương pháp này giúp **tránh rủi ro tự động khớp nối** các thuộc tính có tên trùng lặp một cách không mong muốn nếu lược đồ thay đổi, giữ cho ngữ nghĩa truy vấn ổn định

**Phương pháp mô phỏng bằng Tích Descartes và Theta Join**

**Dùng Where**

ĐSQH: *π*name, course\_id​(*σ*STUDENT.ID = TAKES.ID​ (STUDENT × TAKES))

SQL:

**SELECT** name, course\_id

**FROM** student, takes

**WHERE** student.ID = takes.ID;

**Phương pháp:** Mệnh đề FROM student, takes thực hiện phép **Tích Descartes** (×), tạo ra tất cả các bộ kết hợp có thể. Mệnh đề WHERE student.ID = takes.ID sau đó thực hiện phép **Chọn (***σ***)** để chỉ giữ lại những bộ thỏa mãn điều kiện kết nối.

**Lưu ý :** Trong SQL hiện đại, **nên sử dụng cú pháp INNER JOIN ... ON** thay thế cho Tích Descartes kết hợp với WHERE. Cú pháp JOIN rõ ràng hơn, giúp phân biệt điều kiện kết nối với điều kiện lọc, và thường được Trình tối ưu hóa Truy vấn (Query Optimizer) xử lý **hiệu quả hơn**

**Dùng phương pháp ánh xạ phổ biến Inner Join … On**

ĐSQH: *π*name, course\_id​(STUDENT ⋈ID ​TAKES)

SQL:

**SELECT** T1.name, T2.course\_id

**FROM** STUDENT T1

**INNER** **JOIN** TAKES T2 **ON** T1.ID = T2.ID;

**Lưu ý về thuộc tính trùng lặp:** Khi sử dụng cú pháp JOIN ... ON, kết quả **có thể chứa hai thuộc tính khớp nối** (ví dụ: T1.ID và T2.ID), khác với NATURAL JOIN. Việc sử dụng INNER JOIN ON cho phép chỉ định một vị từ tùy ý (ví dụ: điều kiện không chỉ là phép bằng) trên các quan hệ được nối

**Yêu cầu:** Liệt kê tất cả sinh viên cùng với các khóa học SV đã học. Nếu sinh viên chưa học khóa nào, thông tin khóa học sẽ là NULL.

**Phương pháp  Left Outer Join**

ĐSQH**:** STUDENT ⋉STUDENT.ID=TAKES.ID​ TAKES

SQL:

* Cách 1: Dùng Natural Left Outer Join:

**SELECT** \* **FROM** student

**NATURAL** **LEFT** **OUTER** **JOIN** takes;

* Cách 2: Dùng Left Outer Join lên 1 cột cụ thể:

**SELECT** \* **FROM** STUDENT

**LEFT** **OUTER** JOIN TAKES

**ON** STUDENT.ID = TAKES.ID;

**Ứng dụng :**Phép trừ tập hợp *R* − *S* có thể được ánh xạ bằng Left Outer Join loại trừ.

**Yêu cầu:** Tìm tất cả sinh viên chưa học khóa học nào.

**Left Join Excluding (LJ(E))**

ĐSQH: πID​ (σCOURSE\_IDIS NULL​(STUDENT ⋉ TAKES))

SQL:

**SELECT** ID **FROM** STUDENT

**NATURAL LEFT OUTER JOIN** TAKES

**WHERE** COURSE\_ID **IS NULL**;

**Phương pháp:** Cách này thực hiện LEFT JOIN (để không bỏ sót sinh viên nào) và sau đó sử dụng điều kiện **loại trừ** WHERE [thuộc tính của bảng bên phải] IS NULL.

**Lưu ý quan trọng:** Chỉ cần kiểm tra **một thuộc tính khóa** của bảng bên phải (course id hoặc N.PID) là đủ. Nếu thuộc tính khóa này là NULL, điều đó xác nhận bộ từ bảng bên trái không tìm thấy đối sánh nào trong bảng bên phải, qua đó mô phỏng chính xác phép Trừ tập hợp

**Yêu cầu**:  Liệt kê tất cả các khóa học trong bảng TAKES và thông tin sinh viên tương ứng. (Nếu có khóa học nào không có sinh viên nào trong STUDENT khớp, bộ đó sẽ được giữ lại, đệm NULL cho các thuộc tính STUDENT).

**Phương pháp Right Outer Join**

ĐSQH: TAKES ⋊TAKES.ID = STUDENT.ID​ STUDENT

SQL:

**SELECT** \* **FROM** TAKES

**NATURAL RIGHT OUTER JOIN** STUDENT;

**Mục đích:** Bảo toàn **tất cả các bộ từ quan hệ bên phải** (STUDENT).

**Cách hoạt động:** Tương tự như Left Outer Join, nhưng bộ từ TAKES (bên trái) sẽ được đệm NULL nếu không khớp với bất kỳ bộ nào trong STUDENT (bên phải)

**Yêu cầu:** Hiển thị danh sách tất cả sinh viên khoa “Comp”.“Sci”. và tất cả các khóa học được tổ chức vào Spring 2017, đảm bảo không mất thông tin nào từ cả hai phía.

**Phương pháp Full Outer Join (Kết Ngoài Toàn phần)**

**Cách 1: Sử dụng Natural Full Outer Join**

ĐSQH: (σDEPT\_NAME =′ Comp’.’Sci.′​ (STUDENT)) ⪥STUDENT.ID = TAKES.ID​(σSEMESTER=′Spring′ ∧ YEAR=2017 ​(TAKES))

SQL:

**SELECT** \* **FROM** (

**SELECT** \* **FROM** student

**WHERE** dept name = 'Comp.’. ‘Sci.')

**NATURAL FULL OUTER JOIN**

(**SELECT** \* **FROM** takes

**WHERE** semester = 'Spring'

**AND** year = 2017);

**Cách 2: Sử dụng Join On**

ĐSQH: R ⪥R.Key=S.Key​ S

SQL:

**SELECT** \* **FROM** R

**FULL JOIN** S **ON** R.Key = S.Key;

**Phương pháp giải:** Kết hợp cả Left và Right Outer Join. Bảo toàn tất cả các bộ từ cả hai bảng. Nếu một bộ từ *R* không khớp, nó được đệm NULL ở phía *S*. Nếu một bộ từ *S* không khớp, nó được đệm NULL ở phía *R*

### 4.2.3. Ánh xạ Theta Join ( θ )

**Khái niệm và Đặc điểm**

Phép Kết Theta (Theta Join, ký hiệu *R* ⋈*θ*​ *S*) là một phép toán **không cơ bản** trong Đại số Quan hệ.

Nó được định nghĩa một cách hình thức thông qua sự kết hợp của Phép Tích Descartes (×) và Phép Chọn (*σ*).

*R*⋈*θ* ​*S* ≡ *σθ* ​(*R* × *S*)

• **Mục đích:** Tạo ra một quan hệ mới bằng cách ghép nối các bộ từ hai quan hệ *R* và *S* mà thỏa mãn một điều kiện kết nối *θ* tùy ý.

• **Điều kiện phép** *θ***:** Là một vị từ (predicate) tùy ý, có thể sử dụng bất kỳ toán tử so sánh nào. Các toán tử so sánh bao gồm { <, ≤, >, ≥, ≠} không có kết bằng .

• **Lược đồ kết quả:** Lược đồ của *R*⋈*θ*​ *S* là sự kết hợp của tất cả các thuộc tính của *R* và *S*.

• **Tính chất:** Kết Theta là một biến thể của phép toán Natural Join, cho phép gộp phép Chọn và Tích Descartes thành một phép toán duy nhất.

**Ánh xạ sang SQL**

Theta Join có thể được ánh xạ sang SQL bằng hai cách chính:

**Phương pháp Cổ điển (Dựa trên định nghĩa Đại số Quan hệ): Sử dụng Tích Descartes và Mệnh đề WHERE**

Đây là phương pháp chuẩn, mô phỏng dựa trên định nghĩa

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *R*⋈*R*.*A θ S*.*B*​ *S* | **SELECT \* FROM** R, S  **WHERE** R.A *θ* S.B; |

Bảng 4. 12: Ánh xạ dùng mệnh đề Where

**Lưu ý**: θ ở đây là 1 trong những tập hơp toán tử so sánh như ở điều kiện trên không bao gồm toán tử so sánh “=”

**Phương pháp Hiện đại (Ưu tiên sử dụng): Sử dụng cú pháp JOIN... ON**

SQL cung cấp cú pháp rõ ràng JOIN ... ON để thực hiện Theta Join, cho phép chỉ định một vị từ chung trên các quan hệ được nối.

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *R*⋈*θ* ​*S* | **SELECT \* FROM** R  **JOIN** S **ON *θ* ;** |

Bảng 4. 13: Ánh xạ dùng Join On

Trong đó, *θ* là một vị từ SQL hợp lệ (ví dụ: R.A <= S.B, R.Salary > S.Budget, v.v.).

**Giải thích**

• **Về tính hiệu quả và độ phổ biến:** Phương pháp sử dụng cú pháp **INNER JOIN... ON** là phương pháp **ưu tiên và phổ biến nhất** trong thực tế.

**Tính rõ ràng:** Cú pháp này rõ ràng hơn, tách điều kiện kết nối (ON clause) khỏi điều kiện lọc thông thường (WHERE clause).

**Tối ưu hóa truy vấn:** Mặc dù về mặt lý thuyết, truy vấn sử dụng Tích Descartes kết hợp với WHERE có thể tương đương, các hệ thống cơ sở dữ liệu (DBMS) thường dịch các biểu thức SQL sang một biểu diễn cấp thấp dựa trên Đại số Quan hệ mở rộng để thực hiện Tối ưu hóa Truy vấn. Cú pháp JOIN... ON giúp trình tối ưu hóa truy vấn (Query Optimizer) dễ dàng xử lý và tìm ra kế hoạch thực thi hiệu quả hơn.

**Lưu ý:** Việc sử dụng JOIN... ON cho phép thể hiện một lớp điều kiện kết nối phong phú hơn so với NATURAL JOIN. Nếu sử dụng JOIN... ON, kết quả có thể chứa hai thuộc tính khớp nối (ví dụ: T1.ID và T2.ID), khác với NATURAL JOIN chỉ giữ lại một.

### 4.2.4. Ánh xạ EquiJoin ( R⋈C = s.C S)

**Khái niệm và Đặc điểm**

Phép Kết Bằng (Equijoin) là một **trường hợp đặc biệt** của Theta Join.

• **Định nghĩa:** Phép Kết Bằng là một Theta Join mà điều kiện kết nối *θ* **chỉ bao gồm các phép so sánh bằng (=)**.

• **Sự khác biệt với Natural Join:** Trong Equijoin, tất cả các thuộc tính của cả hai quan hệ (kể cả các thuộc tính được sử dụng để khớp nối) đều được giữ lại trong quan hệ kết quả. Trong khi đó, Natural Join chỉ giữ lại một bản sao của các thuộc tính chung.

• Equijoin là cơ chế cốt lõi để "vật chất hóa" (materialize) các mối quan hệ giữa các bảng trong mô hình quan hệ.

**Ánh xạ sang SQL**

Equijoin có thể được ánh xạ bằng cả phương pháp ngầm định và phương pháp tường minh:

**Phương pháp Ngầm định phổ biến: Dùng Mệnh đề WHERE**

Trong nhiều hệ thống SQL, Equijoin được thực hiện bằng cách liệt kê các bảng trong mệnh đề FROM và sử dụng điều kiện bằng trong mệnh đề WHERE để chỉ định cách các bộ được ghép nối.

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *R*⋈*R*.*A = S*.*B*​ *S* | **SELECT \* FROM** R, S  **WHERE** R.A *=* S.B; |

Bảng 4. 14: Ánh xạ bằng phương pháp ngầm định

**Phương pháp Tường minh (Khuyến nghị): Sử dụng INNER JOIN... ON**

Cú pháp này rõ ràng và thường được khuyến nghị hơn trong SQL hiện đại:

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *R*⋈*R*.*A = S*.*B*​ *S* | **SELECT** \* **FROM** R **INNER** **JOIN** S **ON** R.A = S.B; |

Bảng 4. 15: Ánh xạ bằng phương pháp tường minh

Giải thích Phương pháp

• **Tính chất Equijoin và PK-FK:** Điều kiện kết nối phổ biến nhất là Equijoin, thường xảy ra khi các thuộc tính khớp nối là Khóa chính (Primary Key) và Khóa ngoại (Foreign Key).

• **Phương pháp ưu tiên và độ phổ biến:**

Trong thực tế, cả hai cú pháp (Ngầm định và Tường minh) đều rất phổ biến. Tuy nhiên, việc sử dụng cú pháp **INNER JOIN... ON** được **khuyến nghị hơn** vì nó tường minh, giúp phân biệt rõ ràng giữa logic kết nối và các điều kiện lọc khác.

Các chuyên gia và tài liệu hiện đại thường ưu tiên cú pháp tường minh (INNER JOIN) vì nó giúp mã nguồn dễ đọc, dễ bảo trì và tránh nhầm lẫn giữa Equijoin và Tích Descartes nếu người dùng quên điều kiện WHERE.

• **Lưu ý:** Mặc dù cách viết có vẻ khác nhau, các trình tối ưu hóa truy vấn trong các hệ thống DBMS hiện đại thường sẽ dịch cả hai cú pháp Equijoin (ngầm định và tường minh) thành cùng một kế hoạch thực thi tối ưu, đặc biệt là khi các điều kiện khớp nối dựa trên chỉ mục (index).

**Ví dụ tổng hợp: Chuyển đổi các dạng Join khác nhau sang SQL**

Chúng ta sử dụng hai quan hệ giả định:

**STUDENT** (**ID**, name, dept\_name, tot\_cred) và

**TAKES** (**ID**, course\_id, sec\_id, semester, year, grade). Thuộc tính chung là ID.

**Yêu cầu:** Tìm tên sinh viên và mã khóa học của tất cả các khóa học mà SV đã học. (Chỉ bao gồm sinh viên đã học khóa học).

**Dùng Natural Join (Kết Tự nhiên)**

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| *πname, course\_id​ (STUDENT ⋈ TAKES)* | **SELECT** name, course\_id  **FROM** STUDENT  **NATURAL JOIN** TAKES**;** |

Bảng 4. 16: Ánh xạ Natural Join

**Giải thích:** Phép toán này hoạt động bằng cách tự động tìm các bộ có giá trị bằng nhau trên **tất cả các thuộc tính có tên chung** (trong ví dụ này là ID).

**Lưu ý về cú pháp thay thế:** Trong thực tế, ta nên sử dụng cú pháp **JOIN ... USING (ID)** để chỉ định rõ ràng các cột cần khớp nối. Phương pháp này giúp **tránh rủi ro tự động khớp nối** các thuộc tính có tên trùng lặp một cách không mong muốn nếu lược đồ thay đổi, giữ cho ngữ nghĩa truy vấn ổn định

**Phương pháp mô phỏng bằng Tích Descartes và Theta Join**

**Dùng Where**

ĐSQH: *π*name, course\_id​(*σ*STUDENT.ID = TAKES.ID​ (STUDENT × TAKES))

SQL:

**SELECT** name, course\_id

**FROM** student, takes

**WHERE** student.ID = takes.ID;

**Phương pháp:** Mệnh đề FROM student, takes thực hiện phép **Tích Descartes** (×), tạo ra tất cả các bộ kết hợp có thể. Mệnh đề WHERE student.ID = takes.ID sau đó thực hiện phép **Chọn (***σ***)** để chỉ giữ lại những bộ thỏa mãn điều kiện kết nối.

**Lưu ý :** Trong SQL hiện đại, **nên sử dụng cú pháp INNER JOIN ... ON** thay thế cho Tích Descartes kết hợp với WHERE. Cú pháp JOIN rõ ràng hơn, giúp phân biệt điều kiện kết nối với điều kiện lọc, và thường được Trình tối ưu hóa Truy vấn (Query Optimizer) xử lý **hiệu quả hơn**

**Dùng phương pháp ánh xạ phổ biến Inner Join … On**

ĐSQH: *π*name, course\_id​(STUDENT ⋈ID ​TAKES)

SQL:

**SELECT** T1.name, T2.course\_id

**FROM** STUDENT T1

**INNER** **JOIN** TAKES T2 **ON** T1.ID = T2.ID;

**Lưu ý về thuộc tính trùng lặp:** Khi sử dụng cú pháp JOIN ... ON, kết quả **có thể chứa hai thuộc tính khớp nối** (ví dụ: T1.ID và T2.ID), khác với NATURAL JOIN. Việc sử dụng INNER JOIN ON cho phép chỉ định một vị từ tùy ý (ví dụ: điều kiện không chỉ là phép bằng) trên các quan hệ được nối

**Yêu cầu:** Liệt kê tất cả sinh viên cùng với các khóa học SV đã học. Nếu sinh viên chưa học khóa nào, thông tin khóa học sẽ là NULL.

**Phương pháp  Left Outer Join**

ĐSQH**:** STUDENT ⋉STUDENT.ID=TAKES.ID​ TAKES

SQL:

* Cách 1: Dùng Natural Left Outer Join:

**SELECT** \* **FROM** student

**NATURAL** **LEFT** **OUTER** **JOIN** takes;

* Cách 2: Dùng Left Outer Join lên 1 cột cụ thể:

**SELECT** \* **FROM** STUDENT

**LEFT** **OUTER** JOIN TAKES

**ON** STUDENT.ID = TAKES.ID;

**Ứng dụng :**Phép trừ tập hợp *R* − *S* có thể được ánh xạ bằng Left Outer Join loại trừ.

**Yêu cầu:** Tìm tất cả sinh viên chưa học khóa học nào.

**Left Join Excluding (LJ(E))**

ĐSQH: πID​ (σCOURSE\_IDIS NULL​(STUDENT ⋉ TAKES))

SQL:

**SELECT** ID **FROM** STUDENT

**NATURAL LEFT OUTER JOIN** TAKES

**WHERE** COURSE\_ID **IS NULL**;

**Phương pháp:** Cách này thực hiện LEFT JOIN (để không bỏ sót sinh viên nào) và sau đó sử dụng điều kiện **loại trừ** WHERE [thuộc tính của bảng bên phải] IS NULL.

**Lưu ý quan trọng:** Chỉ cần kiểm tra **một thuộc tính khóa** của bảng bên phải (course id hoặc N.PID) là đủ. Nếu thuộc tính khóa này là NULL, điều đó xác nhận bộ từ bảng bên trái không tìm thấy đối sánh nào trong bảng bên phải, qua đó mô phỏng chính xác phép Trừ tập hợp

**Yêu cầu**:  Liệt kê tất cả các khóa học trong bảng TAKES và thông tin sinh viên tương ứng. (Nếu có khóa học nào không có sinh viên nào trong STUDENT khớp, bộ đó sẽ được giữ lại, đệm NULL cho các thuộc tính STUDENT).

**Phương pháp Right Outer Join**

ĐSQH: TAKES ⋊TAKES.ID = STUDENT.ID​ STUDENT

SQL:

**SELECT** \* **FROM** TAKES

**NATURAL RIGHT OUTER JOIN** STUDENT;

**Mục đích:** Bảo toàn **tất cả các bộ từ quan hệ bên phải** (STUDENT).

**Cách hoạt động:** Tương tự như Left Outer Join, nhưng bộ từ TAKES (bên trái) sẽ được đệm NULL nếu không khớp với bất kỳ bộ nào trong STUDENT (bên phải)

**Yêu cầu:** Hiển thị danh sách tất cả sinh viên khoa “Comp”.“Sci”. và tất cả các khóa học được tổ chức vào Spring 2017, đảm bảo không mất thông tin nào từ cả hai phía.

**Phương pháp Full Outer Join (Kết Ngoài Toàn phần)**

**Cách 1: Sử dụng Natural Full Outer Join**

ĐSQH: (σDEPT\_NAME =′ Comp’.’Sci.′​ (STUDENT)) ⪥STUDENT.ID = TAKES.ID​(σSEMESTER=′Spring′ ∧ YEAR=2017 ​(TAKES))

SQL:

**SELECT** \* **FROM** (

**SELECT** \* **FROM** student

**WHERE** dept name = 'Comp.’. ‘Sci.')

**NATURAL FULL OUTER JOIN**

(**SELECT** \* **FROM** takes

**WHERE** semester = 'Spring'

**AND** year = 2017);

**Cách 2: Sử dụng Join On**

ĐSQH: R ⪥R.Key=S.Key​ S

SQL:

**SELECT** \* **FROM** R

**FULL JOIN** S **ON** R.Key = S.Key;

**Phương pháp giải:** Kết hợp cả Left và Right Outer Join. Bảo toàn tất cả các bộ từ cả hai bảng. Nếu một bộ từ *R* không khớp, nó được đệm NULL ở phía *S*. Nếu một bộ từ *S* không khớp, nó được đệm NULL ở phía *R*

**Yêu cầu**

Tìm tên sinh viên (name) và mã khóa học (course\_id) của các khóa học mà sinh viên đã tham gia, trong đó tổng tín chỉ tích lũy của sinh viên (tot\_cred) phải lớn hơn năm mà họ tham gia khóa học đó (year).

**Lược đồ kết quả:** (name, course\_id, tot\_cred, year, ...)

**Điều kiện kết nối** *θ***:**

*STUDENT*.*ID* = *TAKES*.*ID* ∧ *STUDENT*.*tot*\_*cred* > *TAKES*.*year*

**Dùng Theta Join**

**Phương pháp sử dụng Tích Descartes và Mệnh đề WHERE**

**ĐSQH:**

*πname*,*course*\_*id*​ (*σSTUDENT*.*ID* = *TAKES*.*ID* ∧ *STUDENT*.*tot*\_*cred* > *TAKES*.*year*​ (*STUDENT* × *TAKES*))

**SQL:**

**SELECT** S.name, T.course\_id, S.tot\_cred, T.year

**FROM** STUDENT **AS** S, **TAKES** **AS** T

**WHERE** S.ID = T.ID **AND** S.tot\_cred > T.year;

*Giải thích:* Mệnh đề FROM S, T thực hiện Tích Descartes. Mệnh đề WHERE áp dụng phép Chọn (*σ*) với điều kiện *θ* bao gồm cả điều kiện bằng (S.ID = T.ID) và điều kiện Theta (S.tot\_cred > T.year)

**Phương pháp Khuyến nghị nên dùng: Sử dụng cú pháp JOIN... ON**

ĐSQH:

*STUDENT* ⋈*STUDENT*.*ID* = *TAKES*.*ID* ∧ *STUDENT*.*tot*\_*cred* > *TAKES*.*year* ​(*TAKES)*

SQL:

**SELECT** S.name, T.course\_id, S.tot\_cred, T.year

**FROM** STUDENT **AS** S

**INNER** **JOIN** TAKES **AS** T

**ON** S.ID = T.ID **AND** S.tot\_cred > T.year;

***Lưu ý****:* Khi sử dụng JOIN... ON, phép toán mặc định là Kết Nội (Inner Join).

**Giải thích** :

• **Độ phổ biến:** Cú pháp **INNER JOIN... ON** là phương pháp **ưu tiên** và được sử dụng rộng rãi trong SQL hiện đại.

• **Ưu điểm:** Cú pháp này rõ ràng, tách bạch điều kiện kết nối (ON clause) khỏi các điều kiện lọc thông thường (WHERE clause) (nếu có), giúp truy vấn dễ đọc và bảo trì hơn. Mệnh đề ON cho phép chỉ định một vị từ chung tùy ý (general predicate) lên các quan hệ được nối.

• **Về hiệu suất:** Về mặt logic, truy vấn sử dụng FROM/WHERE và JOIN... ON có thể tương đương. Tuy nhiên, các hệ thống DBMS thường chuyển đổi các biểu thức SQL sang một biểu diễn Đại số Quan hệ mở rộng để tối ưu hóa truy vấn. Cú pháp JOIN... ON có thể giúp trình tối ưu hóa truy vấn dễ dàng áp dụng các chiến lược kết nối hiệu quả hơn.

**Yêu cầu:**

Tìm ID và tên của tất cả sinh viên (name) cùng với mã khóa học (course\_id) mà họ đã đăng ký (ID khớp giữa hai bảng).

**Dùng Equijoin**

**Phương pháp sử dụng Tích Descartes và Mệnh đề WHERE:**

ĐSQH: *πS*.*ID*,*name*,*course*\_*id*​ (*σSTUDENT*.*ID* = *TAKES*.*ID* ​(*STUDENT* × *TAKES*))

SQL:

**SELECT** S.ID, S.name, T.course\_id

**FROM** STUDENT **AS** S, TAKES **AS** T

**WHERE** S.ID = T.ID;

***Giải thích:*** Phương pháp này mô phỏng định nghĩa cơ bản của phép Equijoin thông qua Tích Descartes và Chọn. Đây là cách phổ biến để thực hiện ánh xạ Equijoin

**Phương pháp Tường minh (Khuyến nghị): Sử dụng INNER JOIN... ON**

ĐSQH: *STUDENT*⋈*STUDENT*.*ID* = *TAKES*.*ID*​ (*TAKES)*

SQL:

**SELECT** S.ID, S.name, T.course\_id

**FROM** STUDENT **AS** S

**INNER** **JOIN** TAKES **AS** T **ON** S.ID = T.ID;

***Lưu ý****:* Phép Equijoin bằng cách sử dụng cú pháp JOIN... ON với điều kiện bằng S.ID = T.ID sẽ tạo ra kết quả giống hệt như phương pháp FROM/WHERE.

**Giải thích**

• **Vì sao nên sài phương pháp này:** Phương pháp sử dụng cú pháp **INNER JOIN... ON** (hoặc JOIN... ON) là cách **khuyến nghị** và được sử dụng rộng rãi nhất trong SQL hiện đại.

• **Sự tương đương:** Truy vấn Equijoin sử dụng INNER JOIN T ON S.ID = T.ID cho kết quả tương đương với truy vấn FROM S, T WHERE S.ID = T.ID.

• **Khuyến nghị:** Cú pháp tường minh (INNER JOIN ON) được ưu tiên vì nó giúp người đọc dễ dàng nhận biết mục đích của câu lệnh là thực hiện phép kết nối, làm cho mã nguồn dễ bảo trì hơn so với việc sử dụng cú pháp ngầm định FROM/WHERE.

• **Về hiệu suất:** Các hệ thống quản trị cơ sở dữ liệu (DBMS) thường có các thuật toán tối ưu hóa cao cho Equijoin (như *Hash Join* hoặc *Sort-Merge Join*). Dù bạn viết theo cú pháp nào, trình tối ưu hóa truy vấn thường sẽ chuyển đổi nó thành một kế hoạch thực thi hiệu quả nhất. Tuy nhiên, việc sử dụng cú pháp JOIN... ON cung cấp sự rõ ràng về cấu trúc (tách biệt logic kết nối khỏi logic lọc).

## 4.3. Ánh xạ các toán tử nâng cao khác

Các phép toán Đại số Quan hệ (ĐSQH) cơ bản (Chọn, Chiếu, Tập hợp, Tích Descartes, Kết nối) tạo nên nền tảng cho truy vấn dữ liệu quan hệ. Tuy nhiên, ĐSQH ban đầu không đủ mạnh để xử lý các loại truy vấn phức tạp yêu cầu tính toán thống kê (hàm tổng hợp) hoặc truy vấn yêu cầu định lượng phổ quát ("tất cả").

Do đó, các toán tử nâng cao như **Phép Chia (**÷**)** và **Hàm Tổng hợp (Aggregation)** được giới thiệu để mở rộng khả năng biểu đạt của ĐSQH, và chúng được ánh xạ vào SQL thông qua các cấu trúc phức tạp hơn như GROUP BY và HAVING.

### 4.3.1. Ánh xạ Phép Chia (÷) sang SQL

**Khái niệm**

Phép Chia (÷) là một toán tử nhị nguyên (binary operation) không cơ bản trong ĐSQH.

Cho hai quan hệ *R*(*Z*) và *S*(*X*), trong đó *X* ⊆ *Z* (tập thuộc tính *X* là tập con của tập thuộc tính *Z*), phép chia *R* ÷ *S* tạo ra một quan hệ mới trên lược đồ *Z* − *X*.

Một bộ *t* thuộc *R* ÷ *S* nếu và chỉ nếu:

* *t* thuộc π *Z*−*X* ​(*R*).
* **Với mọi** bộ *ts* trong *S*, tồn tại một bộ *tr*​ trong *R* sao cho *tr*[*X*] = *ts*[*X*] và *tr*​[*Z*−*X*]=*t*.

Phép Chia được sử dụng để giải quyết các truy vấn liên quan đến **định lượng phổ quát** (universal quantification), hay các truy vấn có dạng "Tìm tất cả các thực thể *A* liên quan đến **tất cả** các thực thể *B*".

**Ánh xạ sang SQL (Dùng phép đếm, GROUP BY, và HAVING)**

SQL không có toán tử tương đương trực tiếp cho Phép Chia. Phép Chia thường được mô phỏng bằng cách sử dụng phép đếm kết hợp với các mệnh đề GROUP BY và HAVING trong các truy vấn con lồng nhau.

**Nguyên tắc mô phỏng:**

Để tìm *R* ÷ *S*, chúng ta tìm tất cả các giá trị trong thuộc tính *Z* − *X* (từ *R*) sao cho số lượng các bộ giá trị *X* (từ *S*) mà chúng liên kết đến trong *R* bằng với tổng số lượng bộ trong *S*.

**VD ánh xạ SQL của phép chia**

**Yêu cầu:** Tìm mã NV của tất cả các nhân viên (*E*) làm việc trên **tất cả** các dự án do đơn vị có mã số là 4 phụ trách (*P*4​).

Cho lược đồ quan hệ như sau:

***THAMGIA*\_*DA***(***MS*\_*NV*, *MS*\_*DA***, *SOGIO*).

**DUAN(MS\_DA , MS\_DV,** TEN**)**

**NHANVIEN(MS\_NV,** HOTEN, LUONG**, MS\_DV)**

Giả sử chúng ta có :

• *R* là quan hệ tham gia dự án (Tương ứng với *Z*)

• *S* là tập hợp các dự án do Đơn vị 4 phụ trách:

*Đặt P*4​ = *πMS*\_*DA*​ (*σMS*\_*DV* = 4​(*DUAN*)) (Tương ứng với *X*).

ĐSQH: *πMS*\_*NV*​ (*THAMGIA*\_*DA*) ÷ *P*4​.

SQL:

**SELECT** MS\_NV **FROM** THAMGIA\_DA

**WHERE** MS\_DA **IN** (**SELECT** MS\_DA **FROM** DUAN

**WHERE** MS\_DV = 4)

**GROUP BY** MS\_NV

**HAVING COUNT** (**DISTINCT** MS\_DA) = (**SELECT** **COUNT** (MS\_DA) **FROM** DUAN **WHERE** MS\_DV = 4);

**Giải thích phương pháp:**

**Xác định tập hợp** *S* **(Divisor):** Truy vấn con (SELECT COUNT(MS\_DA) FROM DUAN WHERE MS\_DV = 4) tính tổng số dự án mà Đơn vị 4 phụ trách. Đây là chuẩn mực để so sánh.

**Đếm cho mỗi nhóm:** Mệnh đề GROUP BY MS\_NV nhóm các bộ của *THAMGIA*\_*DA* theo từng nhân viên.

**Áp dụng điều kiện phổ quát:** Mệnh đề HAVING COUNT(DISTINCT MS\_DA) đảm bảo rằng số lượng dự án riêng biệt mà mỗi nhân viên tham gia phải bằng với tổng số dự án mà Đơn vị 4 phụ trách (được tính ở bước 1).

**Lưu ý:**

• **Độ phức tạp:** Phép Chia là một toán tử phức tạp về mặt ngữ nghĩa và thường dẫn đến các truy vấn SQL lồng nhau hoặc sử dụng GROUP BY/HAVING.

• **Phương pháp thay thế:** Ngoài phương pháp đếm, người ta còn có thể mô phỏng Phép Chia bằng cách sử dụng hai lần Phép Trừ Tập hợp (Set Difference) hoặc truy vấn con NOT EXISTS lồng nhau để diễn tả định lượng phổ quát (∀*x*,*P*(*x*) tương đương với ¬∃*x*,¬*P*(*x*)). Tuy nhiên, **phương pháp dùng GROUP BY và HAVING thường được ưa chuộng hơn trong thực tế** vì nó trực quan và dễ tối ưu hóa hơn đối với một số DBMS

### 4.3.2. Ánh xạ các hàm tổng hợp và GROUP BY trong SQL

**Hàm Tổng hợp (Aggregate Functions)**

Các hàm này được thêm vào **ĐSQH Mở rộng (Extended Relational Algebra)**, thường được ký hiệu bằng ℑ hoặc *γ*.

**SQL cung cấp năm hàm tổng hợp tích hợp:**

• **SUM:** Tính tổng các giá trị trong một cột số.

• **AVG:** Tính giá trị trung bình của các giá trị trong một cột số.

• **COUNT:** Đếm số lượng giá trị trong một cột. COUNT(\*) đếm tất cả các hàng, bao gồm cả trùng lặp và NULL (trừ khi áp dụng cho một cột cụ thể, các giá trị NULL thường bị bỏ qua).

• **MIN:** Tìm giá trị nhỏ nhất trong một cột.

• **MAX:** Tìm giá trị lớn nhất trong một cột.

**Lưu ý về Dữ liệu:** Các hàm SUM và AVG chỉ áp dụng cho dữ liệu số, trong khi COUNT, MIN, MAX có thể áp dụng cho các kiểu dữ liệu khác (như chuỗi) nếu có thể sắp xếp được.

**Ánh xạ cơ bản (Không nhóm):** Nếu không có mệnh đề GROUP BY, hàm tổng hợp được áp dụng cho toàn bộ tập hợp các bộ được chọn (sau khi áp dụng FROM và WHERE), trả về một hàng kết quả duy nhất.

**VD ánh xạ SQL**

Cho lược đồ quan hệ sau:

**INSTRUCTOR(ID,** name, dept\_name, salary**)**

**TEACHES(ID, course id, sec id,** semester, year**)**

**Yêu cầu:** Tìm mức lương trung bình của tất cả giảng viên

|  |  |
| --- | --- |
| **ĐSQH** | **SQL** |
| ℑAVG(salary) ​(INSTRUCTOR) | **SELECT** AVG(salary)  **FROM** INSTRUCTOR; |

Bảng 4. 13: VD hàm Average

**Giải thích phương pháp:**

• Trong trường hợp này, AVG(salary) tính giá trị trung bình của cột salary trên toàn bộ quan hệ instructor. Việc giữ lại các bản sao (duplicate retention) là quan trọng khi tính toán giá trị trung bình.

**Mệnh đề GROUP BY và HAVING**

Hàm tổng hợp trở nên mạnh mẽ hơn khi kết hợp với khả năng phân nhóm:

• **GROUP BY:** Được sử dụng để phân chia các bộ của một quan hệ thành các nhóm con (partitions), trong đó các bộ có cùng giá trị trên các thuộc tính được chỉ định sẽ thuộc về cùng một nhóm.

Các hàm tổng hợp sau đó được tính toán độc lập trên mỗi nhóm, tạo ra một hàng tóm tắt cho mỗi nhóm.

Trong mệnh đề SELECT của truy vấn có GROUP BY, **chỉ** được phép sử dụng các thuộc tính nhóm hoặc các hàm tổng hợp.

• **HAVING:** Mệnh đề này áp dụng một điều kiện lọc lên **các nhóm** (groups) được hình thành bởi GROUP BY, chứ không phải lên từng bộ dữ liệu riêng lẻ như WHERE.

HAVING được thực hiện **sau** khi GROUP BY đã tạo ra các nhóm, cho phép nó sử dụng các hàm tổng hợp trong điều kiện lọc.

**Thứ tự thực hiện truy vấn SQL (có GROUP BY và HAVING)**:

* **FROM:** Xác định các bảng đầu vào.
* **WHERE:** Lọc các bộ riêng lẻ thỏa mãn điều kiện.
* **GROUP BY:** Chia các bộ còn lại thành các nhóm.
* **HAVING:** Lọc các nhóm dựa trên điều kiện tổng hợp.
* **SELECT:** Tính toán các hàm tổng hợp cho các nhóm còn lại và chiếu các thuộc tính.
* **ORDER BY:** Sắp xếp kết quả cuối cùng theo thứ tự.
* **VD ánh xạ sang SQL**
* **Yêu cầu:** Tìm mức lương trung bình trong mỗi khoa (dept\_name)
* **Phương pháp Group By**

ĐSQH: dept\_name ℑAVG(salary) ​(INSTRUCTOR*)*

SQL:

**SELECT** dept\_name**, AVG(**salary**)**

**FROM** INSTRUCTOR

**GROUP BY** dept\_name;

**Giải thích phương pháp:**

* **Gom nhóm:** Mệnh đề GROUP BY dept\_name chia quan hệ instructor thành các nhóm, mỗi nhóm chứa các bộ có cùng giá trị dept\_name.
* **Tính toán:** Hàm AVG(salary) được tính toán riêng rẽ cho từng nhóm.
* **Lưu ý:** Bất kỳ thuộc tính nào xuất hiện trong SELECT (như dept\_name) mà không phải là đối số của hàm tổng hợp đều phải xuất hiện trong GROUP BY.

**Yêu cầu:** Tìm tên khoa và lương trung bình, chỉ hiển thị những khoa có mức lương trung bình lớn hơn $42,000.

**Phương pháp Group By + Having**

ĐSQH: ρ(avg\_salary/AVG(salary)) ​(σavg\_salary​>42000 ​(γdept\_name ;AVG(salary)​ (INSTRUCTOR)))

SQL:

**SELECT** dept name, **AVG** (salary) **AS** avg\_salary

**FROM** instructor

**GROUP BY** dept name

**HAVING AVG (salary) > 42000**

**Giải thích phương pháp và Khuyến nghị sử dụng phương pháp này:**

* **HAVING so với WHERE:** Mệnh đề WHERE được áp dụng cho **các bộ riêng lẻ** (trước khi nhóm). Mệnh đề **HAVING** được áp dụng cho **các nhóm** (sau khi tính toán tổng hợp). Do đó, HAVING là nơi duy nhất có thể sử dụng hàm tổng hợp để lọc kết quả.
* **Thứ tự ưu tiên:** Trong thực tế, trình tối ưu hóa truy vấn xử lý FROM, sau đó WHERE, sau đó GROUP BY, và cuối cùng là HAVING.
* **Khuyến nghị thực tế:** Luôn sử dụng WHERE để lọc các điều kiện không tổng hợp nhằm giảm kích thước dữ liệu **trước** khi việc gom nhóm GROUP BY được thực hiện, giúp tăng hiệu suất

# Chương 5. VAI TRÒ CỦA ĐẠI SỐ QUAN HỆ TRONG TÓI ƯU HÓA TRUY VẤN

**5.1 Khái niệm tối ưu hóa truy vấn**

**5.1.1 Định nghĩa tối ưu hóa truy vấn**

Tối ưu hóa truy vấn là một bước quan trọng trong quá trình xử lý truy vấn trong hệ quản trị cơ sở dữ liệu (DBMS). Khi một truy vấn được gửi đến hệ thống, nó thường được biểu diễn dưới dạng câu lệnh SQL - một ngôn ngữ khai báo, trong đó người dùng chỉ định mục tiêu cần đạt được chứ không mô tả cách thực hiện. DBMS phải tự mình lập kế hoạch thực thi sao cho hiệu quả nhất. Tối ưu hóa truy vấn chính là quá trình tìm kiếm và việc lựa chọn kế hoạch thực thi truy vấn với chi phí thấp nhất dựa trên các biểu thức truy vấn dạng đại số quan hệ tương đương.[1,3]

Trình tối ưu truy vấn sẽ biến đổi biểu thức đại số quan hệ thành các biểu thức tương đương khác dựa trên các quy tắc và luật toán để giảm thiểu số lượng tổ chức dữ liệu cần quét, tái cấu trúc thứ tự các phép toán và chọn chiến lược thực thi tối ưu như loại join, chỉ số sử dụng, v.v. Quá trình này giúp giảm đáng kể chi phí thực thi, giảm tài nguyên hệ thống tiêu thụ và tăng hiệu suất tổng thể của hệ thống.[3]

**5.1.2 Mục tiêu của tối ưu hóa truy vấn**

Mục tiêu cuối cùng của tối ưu hóa truy vấn tập trung vào việc giảm chi phí thực thi, cụ thể bao gồm:

- Giảm I/O Cost: Là lượng thời gian và số lần truy cập bộ nhớ ngoài (đĩa cứng), vì việc đọc ghi đĩa chiếm phần lớn thời gian thực thi. Việc tối ưu nhằm hạn chế các lượt đọc đĩa không cần thiết, đặc biệt trong môi trường dữ liệu lớn.

- Tiết kiệm CPU Cost: Tối ưu tránh các phép toán chi phí cao về thực thi, lựa chọn thuật toán xử lý dữ liệu nhanh hơn và hiệu quả hơn.

- Sử dụng bộ nhớ hiệu quả: Tận dụng cache, bộ nhớ đệm một cách tối ưu giúp giảm tần suất truy cập I/O và CPU.

- Cải thiện hiệu năng tổng quát: Giúp hệ thống phản hồi truy vấn nhanh hơn, phục vụ nhiều truy vấn đồng thời mà vẫn đảm bảo độ ổn định hệ thống.

Nhờ đạt được các mục tiêu trên, quá trình tối ưu hóa giúp giảm thời gian phản hồi của hệ thống, tiết kiệm tài nguyên phần cứng và nâng cao trải nghiệm người dùng cuối.[1,2]

**5.1.3 Vị trí của đại số quan hệ trong tối ưu hóa truy vấn**

Đại số quan hệ đóng vai trò đặc biệt quan trọng trong quá trình tối ưu hóa truy vấn. Câu lệnh SQL ban đầu được chuyển dịch sang một biểu thức đại số quan hệ tương đương, đây là ngôn ngữ truy vấn thủ tục nhằm biểu diễn cụ thể các phép toán trên dữ liệu quan hệ. Bộ tối ưu truy vấn dựa vào biểu thức này để thực hiện biến đổi theo các quy tắc tương đương nhằm tìm ra kế hoạch thực thi tối ưu nhất.

Quá trình này không chỉ đơn thuần là biến đổi biểu thức mà còn bao gồm việc đánh giá chi phí thực thi của từng kế hoạch, lựa chọn chiến thuật như sắp xếp thứ tự các phép toán, chọn thuật toán join giữa các bảng (nested loop join, hash join, sort-merge join...), lựa chọn các index, và những kỹ thuật thực thi khác.

Đại số quan hệ giúp tạo ra một ngôn ngữ chung để trình bày các biểu thức truy vấn một cách chính xác và có thể khảo sát, đánh giá hiệu quả các chiến lược thực thi trên nền tảng toán học chắc chắn, từ đó cải tiến các thuật toán tối ưu nhằm đạt hiệu quả thực thi tối đa.[2,3]

Đại số quan hệ cung cấp công cụ mạnh mẽ để định nghĩa và thực hiện các biến đổi biến đổi này. Các thuật toán tối ưu sẽ dựa vào quy tắc tương đương đại số quan hệ để biến đổi và chọn lựa biểu thức phù hợp với chi phí thấp nhất. Qua đó, đại số quan hệ là nền tảng không thể thiếu để xây dựng các trình tối ưu truy vấn hiện đại.[1,3]

**5.1.4 Một số kỹ thuật tối ưu truy vấn nâng cao**

- Tối ưu hóa dựa trên chi phí (Cost-based optimization)

Đây là kỹ thuật phổ biến, bộ tối ưu truy vấn sẽ ước lượng chi phí thực thi các kế hoạch truy vấn khác nhau dựa trên các thống kê và đặc tính dữ liệu (ví dụ: số bản ghi, độ phân tán dữ liệu, kích thước bảng). Kế hoạch có chi phí dự đoán thấp nhất sẽ được lựa chọn để thực thi.[1, 3]

- Tối ưu hóa heuristic (Heuristic optimization):

Sử dụng các quy tắc thực nghiệm, ví dụ như thực hiện chọn (selection) trước chiếu (projection), loại bỏ các phép toán không cần thiết, thay đổi thứ tự join để giảm số lượng bộ trung gian mà không phân tích toàn bộ chi phí.[1]

- Tối ưu hóa JOIN

Join là một trong những phép toán tốn kém nhất, nên việc lựa chọn thuật toán join (nested loop join, sort-merge join, hash join) và thứ tự thực hiện join nhiều bảng là cực kỳ quan trọng. Các kỹ thuật tối ưu join giúp giảm mạnh chi phí truy vấn.[3]

- Sử dụng chỉ mục index (index utilization)

Kết hợp lựa chọn và sử dụng chỉ mục để tăng tốc độ truy cập, tránh scan bảng toàn phần. Kỹ thuật này quan trọng trong tối ưu truy vấn thực tế.[2]

- Biến đổi truy vấn dựa trên các quy tắc tương đương (Query rewrite)

Biến đổi biểu thức đại số quan hệ theo các quy tắc tương đương, ví dụ chuyển đổi biểu thức join, đẩy điều kiện lọc xuống sớm hơn, loại bỏ trùng, phân chia truy vấn phức tạp thành truy vấn đơn giản,... giúp giảm thiểu khối lượng dữ liệu xử lý.[1, 3]

- Tối ưu hóa truy vấn phân tán

Trong môi trường nhiều máy chủ và dữ liệu phân tán, tối ưu hóa cần xem xét vị trí dữ liệu, chi phí truyền thông và thực thi song song, phức tạp hơn so với tối ưu hóa truy vấn trên hệ thống đơn lẻ.[2]

**5.2 Các quy tắc tương đương dại số**

**5.2.1 Tổng quan về các quy tắc tương đương**

Trong tối ưu hóa truy vấn, các biểu thức đại số quan hệ có thể được biến đổi thành các biểu thức tương đương nhằm giảm chi phí thực thi. Hai biểu thức tương đương có nghĩa là luôn trả về cùng một kết quả trên mọi cơ sở dữ liệu. Việc vận dụng các quy tắc tương đương giúp bộ tối ưu truy vấn sinh ra các kế hoạch truy vấn đa dạng và chọn kế hoạch hiệu quả nhất.

Các quy tắc tương đương này dựa trên các phép toán cơ bản của đại số quan hệ như chọn (σ), chiếu (π), hợp (∪), hiệu (−), kết nối (⨝), giao (∩), v.v., và các tính chất toán học như giao hoán, kết hợp, phân phối.

**5.2.2 Các quy tắc tương đương đại số**

Hai biểu thức đại số quan hệ được gọi là tương đương nếu cả hai sinh ra cùng một tập các bộ trên mọi instance hợp lệ của cơ sở dữ liệu

Lưu ý: thứ tự của các bộ không quan trọng

Ta không quan tâm nếu các biểu thức tạo ra kết quả khác nhau trên các cơ sở dữ liệu vi phạm ràng buộc toàn vẹn (integrity constraints)

Trong SQL, đầu vào và đầu ra thường là các tập nhiều bộ (multiset)

Hai biểu thức trong phiên bản multiset của đại số quan hệ được gọi là tương đương nếu cả hai sinh ra cùng một tập nhiều bộ trên mọi instance hợp lệ của cơ sở dữ liệu

Quy tắc tương đương nói rằng hai dạng biểu thức được coi là tương đương

Có thể thay thế biểu thức dạng đầu tiên bằng dạng thứ hai, hoặc ngược lại

**Các quy tắc tương đương:**

1. Các phép chọn liên hợp có thể phân tách thành một dãy các phép chọn đơn lẻ

σθ1∧θ2(E) ≡ σθ1(σθ2(E))

2. Các phép chọn có thể hoán đổi vị trí với nhau

σθ1(σθ2(E)) ≡ σθ2(σθ1(E))

3. Chỉ cần phếp chiếu cuối cùng trong chuỗi các phép chiếu, còn lại có thể bỏ qua

ΠL1(ΠL2(...(Π Ln (E))...)) ≡ ΠL1(E), với L1 ⊆ L2 ... ⊆ Ln

4. Có thể kết hợp phép chọn với phép nhân Descartes và phép nối theta

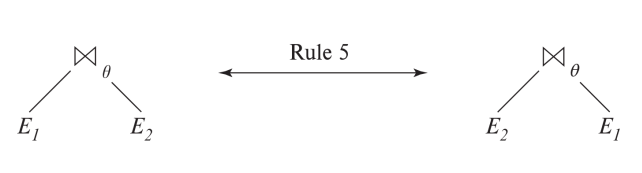
a) σθ(E1×E2) ≡ E1 ⋈θ E2

b) σθ1(E1 ⋈θ2 E2) ≡ E1 ⋈θ1∧θ2 E2

5. Các phép nối theta (và nối tự nhiên) có tính chất giao hoán:

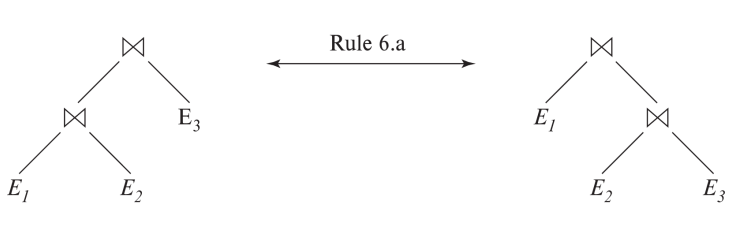
E1 ⋈ E2 ≡ E2 ⋈ E1

*Minh họa quy tắc tương đương trên cây*



6.a) Các phép nối tự nhiên có tính chất kết hợp

(E1 ⋈ E2) ⋈ E3 ≡ E1 ⋈ (E2 ⋈ E3)



6.b) Phép nối theta cũng có tính chất kết hợp như sau:

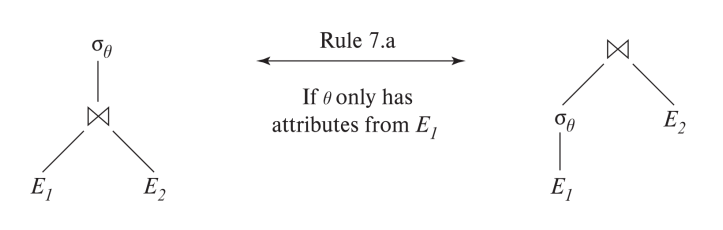
(E1 ⋈θ1 E2) ⋈θ2∧θ3 E3 ≡ E1 ⋈θ1∧θ3 (E2 ⋈θ2 E3)

(với θ2 chỉ chứa các thuộc tính của E2 và E3)

7. Phép chọn phân phối lên phép nói theta trong hai trường hợp:

a) Khi tất cả các thuộc tính trong θ0 chỉ liên quan đến các thuộc tính của một trong hai biểu thức (ví dụ E1 ) đang nối:

σθ0(E1 ⋈θ E2) ≡ (σθ0(E1)) ⋈θ E2



b) Khi θ1 chỉ liên quan đến thuộc tính của E1 và θ2 chỉ liên quan đến thuộc tính của E2:

σθ1∧θ2(E1 ⋈θ E2) ≡ (σθ1(E1)) ⋈θ (σθ2(E2))

8. Phép chiếu phân phối lên phép nối theta như sau:

a) Nếu θ chỉ bao gồm các thuộc tính của L1 ∪ L2

ΠL1∪L2(E1 ⋈θ E2) ≡ ΠL1(E1) ⋈θ ΠL2(E2)

b) Tổng quát, xét phép nối E1 ⋈θ E2:  
- Gọi L1 và L2 là tập thuộc tính lấy từ E1 và E2  
- L3 là các thuộc tính của E1 xuất hiện trong điều kiện nối θ nhưng không thuộc L1∪L2  
- L4 là các thuộc tính của E2 xuất hiện trong điều kiện nối θ nhưng không thuộc  L1∪ L2

ΠL1∪L2(E1⋈θE2)≡ΠL1∪L2(ΠL1∪L3(E1)⋈θΠL2∪L4(E2))ΠL1∪L2(E1⋈θE2)≡Π=(ΠL1∪L3(E1)⋈θΠL2∪L4(E2))  
Các quy tắc tương đương tương tự cũng áp dụng cho các phép nối ngoài (left outer join, right outer join, full outer join).

9) Các phép toán tập hợp (hợp và giao) có tính chất giao hoán:

E1 ∪ E2 ≡ E2 ∪ E1  
E1 ∩ E2 ≡ E2 ∩ E1  
(Phép E1 − E2 thì không giao hoán.)

10) Phép hợp và phép giao có tính chất kết hợp

(E1 ∪ E2) ∪ E3 ≡ E1 ∪ (E2 ∪ E3)  
(E1 ∩ E2) ∩ E3 ≡ E1 ∩ (E2 ∩ E3)

11) Phép chọn phân phối lên hợp, giao và hiệu:

a. σθ(E1 ∪ E2) ≡ σθ(E1) ∪ σθ(E2)   
b. σθ(E1 ∩ E2) ≡ σθ(E1) ∩ σθ(E2)   
c. σθ(E1 − E2) ≡ σθ(E1) − σθ(E2)   
d. σθ(E1 − E2) ≡ σθ(E1) − E2  
e. σθ(E1 − E2) ≡ σθ(E1) − E2  
(Tính tương đương này không hoàn toàn đúng cho phép hợp.)

12. Phép chiếu phân phối lên phép hợp:

ΠL(E1 ∪ E2) ≡ (ΠL(E1)) ∪ (ΠL(E2))

13. Phép chọn phân phối lên phép gom nhóm như sau:

σθ(GY,A(E)) ≡ GY,A(σθ(E))  
(với điều kiện θ chỉ liên quan đến các thuộc tính trong G)

14.a) Phép nối ngoài đầy đủ (full outer join) có tính chất giao hoán:

E1 ⟗ E2 ≡ E2 ⟗ E1

14.b) Phép nối ngoài trái và phải không giao hoán:

E1 ⟕ E2 ≡ E2 ⟖ E1

15. Phép chọn phân phối lên nói ngoài trái và phải như sau, với điều kiện θ1 chỉ liên quan thuộc tính của E1

a) σθ1(E1 ⟕θ E2) ≡ (σθ1(E1)) ⟕θ E2  
b) σθ1(E1 ⟖θ E2) ≡ E2 ⟖θ (σθ1(E1))

16. Có thể thay thế nối ngoài bằng nối trong trong một số điều kiện:

a) σθ1(E1 ⟕θ E2) ≡ σθ1(E1 ⋈θ E2)   
b) σθ1(E1 ⟖θ E2) ≡ σθ1(E1 ⋈θ E2)   
với điều kiện θ1loại bỏ giá trị null ở E2

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

# TÀI LIỆU THAM KHẢO

1. Fundamentals of Database Systems - Elmasri & Navathe (6th edition) Chương 6
2. Tài liệu bạn cung cấp trên GeeksforGeeks có tiêu đề "Introduction of Relational Algebra in DBMS"
3. "Mapping relation algebra operators into SQL queries: A database case study"
4. Unit 4 Relational Algebra (Using SQL DML Syntax): Data Manipulation Language For Relations
5. Abraham-Silberschatz-Henry-F.-Korth-S.-Sudarshan-Database-System-Concepts-McGraw-Hill-Education-2019
6. Chapter4\_Relational\_AlgebraAdvances – Nguyễn Hòa
7. Advances in Probabilistic Databases for Uncertain Information Management-Springer Ber
8. Databases – ramaz
9. Pearson\_Database\_Systems\_A\_Practical\_Approach\_to\_Design\_Implementation\_and\_Management\_6th\_Global\_Edition
10. The Object Data Standard\_3.0
11. Introduction of Relational Algebra in DBMS - GeeksforGeeks