**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

Trong các tổ chức hiện đại, dữ liệu đóng vai trò trung tâm trong mọi hoạt động — từ quản lý khách hàng, giao dịch tài chính, quản trị nhân sự cho đến sản xuất và nghiên cứu. Sự gia tăng nhanh chóng về khối lượng và độ phức tạp của dữ liệu khiến các phương pháp lưu trữ truyền thống bằng tệp (file-processing systems) trở nên lỗi thời và khó kiểm soát.

Các hệ thống tệp độc lập thường gặp nhiều hạn chế: dữ liệu bị **trùng lặp và không nhất quán**, khó chia sẻ giữa các ứng dụng, cấu trúc dữ liệu phụ thuộc chặt chẽ vào chương trình, thiếu **cơ chế kiểm soát đồng thời và phục hồi** khi xảy ra lỗi. Ngoài ra, việc duy trì **tính toàn vẹn và bảo mật dữ liệu** cũng rất khó khăn khi không có cơ chế quản lý tập trung.

Để khắc phục những vấn đề này, hệ quản trị cơ sở dữ liệu (DBMS) được phát triển như một lớp phần mềm trung gian giữa người dùng, ứng dụng và dữ liệu. DBMS cung cấp môi trường có cấu trúc giúp lưu trữ, truy xuất và cập nhật dữ liệu một cách hiệu quả, đồng thời đảm bảo các yêu cầu về an toàn, nhất quán và khả năng mở rộng.

Một trong những nguyên lý cốt lõi của DBMS là **sự trừu tượng hóa dữ liệu (data abstraction)**. Hệ thống phân tách dữ liệu thành ba mức mô tả chính:

* **Mức vật lý (physical level)**: mô tả cách dữ liệu được lưu trữ thực tế trong bộ nhớ hoặc trên đĩa.
* **Mức logic (logical level)**: biểu diễn cấu trúc của toàn bộ cơ sở dữ liệu dưới dạng các bảng, thuộc tính và mối quan hệ.
* **Mức nhìn (view level)**: thể hiện dữ liệu dưới góc nhìn của người dùng hoặc ứng dụng cụ thể.

Cấu trúc ba lớp này cho phép **độc lập dữ liệu** – thay đổi ở một mức không ảnh hưởng tới mức khác. Nhờ đó, các ứng dụng có thể phát triển hoặc bảo trì mà không cần chỉnh sửa cách dữ liệu được lưu trữ.

Ngoài ra, DBMS đảm nhận nhiều chức năng quan trọng khác:

* **Quản lý tính toàn vẹn dữ liệu (data integrity)** thông qua các ràng buộc như khóa chính, khóa ngoại và miền giá trị.
* **Quản lý truy cập đồng thời (concurrency control)** để nhiều người dùng có thể thao tác cùng lúc mà không gây xung đột.
* **Đảm bảo tính bền vững (durability)** và **khả năng phục hồi (recovery)** thông qua cơ chế ghi log và khôi phục sau lỗi.
* **Cung cấp bảo mật (security)** bằng các cơ chế phân quyền và xác thực người dùng.
* **Hỗ trợ ngôn ngữ truy vấn cấp cao (SQL)** giúp người dùng tương tác với dữ liệu một cách linh hoạt mà không cần thao tác trực tiếp ở tầng vật lý.

Nhờ các chức năng này, DBMS trở thành nền tảng không thể thiếu trong mọi hệ thống thông tin hiện đại, từ quy mô nhỏ đến các tập đoàn toàn cầu. Nó không chỉ giúp tăng hiệu quả xử lý và giảm lỗi dữ liệu, mà còn cho phép tổ chức khai thác giá trị từ dữ liệu để ra quyết định chiến lược.

**Nguồn :** [1] Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts, 7th Edition, McGraw-Hill Education, 2019.

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

Thị trường hệ điều hành di động hiện nay chủ yếu bao gồm Android, iOS, BlackBerry OS, và Windows Phon**e,** mỗi hệ điều hành mang đến những đặc điểm và mục tiêu phục vụ khác nhau, tạo nên sự đa dạng cho thị trường di động toàn cầu. Sự cạnh tranh giữa các hệ điều hành này ảnh hưởng mạnh mẽ đến sự phát triển của các thiết bị di động và dịch vụ đi kèm.

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

Việc lựa chọn hệ điều hành (HĐH) của người dùng bị ảnh hưởng bởi nhiều yếu tố khác nhau, bao gồm tính dễ sử dụng, bảo mật, tính năng hỗ trợ, và khả năng tương thích với các thiết bị khác. Mỗi yếu tố có tác động riêng, tạo ra những khác biệt trong trải nghiệm người dùng và góp phần định hình thị trường hệ điều hành di động.

## Tóm tắt chương 1

điều hành di động hiện nay.

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

## Nhóm toán tử tập hợp

Hệ điều hành android

## Phép hợp

Hệ điều hành Android được phát triển bởi Android Inc., một công ty khởi nghiệp

## Phép hiệu

Hệ điều hành Android đã trở thành một trong những nền tảng di động phổ biến nhất trên thế giới, nhờ vào những ưu điểm vượt trội của nó.

**2.1.3 Phép tích đề-các**

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

## Nhóm các toán tử quan hệ cơ bản

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

## Phép chọn

iCloud, tạo ra một hệ sinh thái tích hợp mà người dùng khó có thể từ chối.

## Phép chiếu

aaaaaaaaaaaaaaa

## Phép đổi tên

aaaaaaaaaaaaa

## Tóm tắt chương 2

Chương này chủ yếu tập trung phân tích các hệ điều hành di động phổ biến hiện nay, bao gồm Android, iOS, BlackBerryOS và Windows Phone, qua đó làm nổi bật đặc trưng và ưu nhược điểm của từng nền tảng. Android nổi bật nhờ vào mã nguồn mở, khả năng tùy chỉnh cao, nhưng gặp khó khăn về vấn đề bảo mật. Trong

khi đó iOS ghi điểm nhờ trải nghiệm mượt mà, bảo mật mạnh mẽ, song hạn chế về tùy biến và giá thành. BlackBerryOS giữ vị thế trong bảo mật doanh nghiệp, nhưng dần tụt lại bởi giao diện lỗi thời và hệ sinh thái kém phong phú. Ngược lại Windows Phone gây ấn tượng với giao diện hiện đại nhưng không đủ ứng dụng để cạnh tranh. Chương này không chỉ giúp người đọc hiểu sâu hơn về thị trường hệ điều hành mà còn gợi mở cách lựa chọn tối ưu cho từng nhu cầu.

# 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.

**Tính phân phối :**

Phép Giao cũng có thể phân phối qua phép Hợp (UNION) hoặc Hiệu (SET DIFFERENCE).

R ∩ ( S ∪ T ) = ( R ∩ S ) ∪ ( R ∩ T )

Ví dụ: Với R={1,2,3,4}, S={3,4,5}, và T={4,5,6}, cả hai vế cho ra tập {3,4}.

Tính chất này cho phép hệ thống tối ưu hóa truy vấn bằng cách chuyển đổi các phép toán phức hợp thành những phép nhỏ hơn, giúp giảm lượng dữ liệu trung gian cần xử lý.

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

## Phép kết (join)

Aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

### 3.2.1. Kết theta (theta JOIN)

**Android** hiện đang chiếm khoảng 70% thị phần toàn cầu trong lĩnh vực di động, với IOS : sự bền bỉ trong phân khúc cao cấp

**3.2.2. Kết bằng (equi JOIN)**

điện thoại di động vì Microsoft đã chuyển hướng chiến lược, tập trung vào các lĩnh

**3.2.3. Kết tự nhiên (natural JOIN)**

Aaaaaaaaaa

### 3.2.4. Phép kết ngoài (outer JOIN)

aaaaaaaaaaaaaaaaaa

## 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:

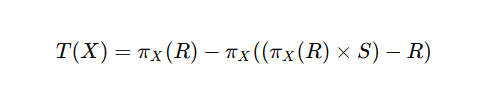
* + - * Tìm các **sinh viên đã đăng ký tất cả các khóa học** trong chương trình đào tạo
      * Tìm các **nhân viên làm việc trong mọi dự án** do một phòng ban quản lý.

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:



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.

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.

Đầu tiên, ta chọn ra tất cả các dự án được phòng ban số 5 quản lý:



Tập kết quả S chính là danh sách các dự án cần xem xét.

Sau đó, ta xác định quan hệ R gồm các cặp (Essn, Pno) thể hiện nhân viên làm việc trong từng dự án.  
Khi đó, ta áp dụng phép chia:

T ← R ÷ S

Kết quả T bao gồm các Essn (mã nhân viên) có trong WORKS\_ON, thỏa mãn điều kiện là tham gia tất cả các dự án trong S.

Từ T, ta có thể chiếu sang quan hệ EMPLOYEE để lấy ra tên nhân viên tương ứng.

Qua ví dụ này, có thể thấy phép chia đặc biệt hữu ích khi xử lý các truy vấn có dạng “cho tất cả”, vốn không thể biểu diễn gọn gàng bằng các phép toán khác trong đại số quan hệ.

**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".**

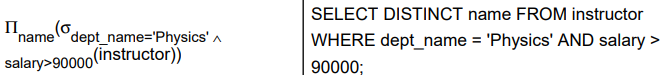
**ĐSQH | SQL**

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*

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

# 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**