**KHÁC NHAU GIỮA WHERE VÀ HAVING**

**BÀI THAM KHẢO SỐ 1**

WHERE - filter kết quả theo dòng

HAVING - filter kết quả theo GROUP

Ví dụ: bạn có 1 bảng như sau

|  |  |  |
| --- | --- | --- |
| **Student** | **Subject** | **Point** |
| A | Math | 8 |
| A | Physics | 5 |
| B | Math | 4 |
| C | Physics | 9 |

Câu lệnh:

**SELECT Student, sum(Point)**

**FROM Point**

**GROUP BY Student**

*Kết quả trả về:*

|  |  |  |
| --- | --- | --- |
| **Student** | **Point** | **Chú thích** |
| A | 13 | Nó kiếm Student giống nhau ở đây 2 dòng đều có A nên lấy SUM |
| B | 4 |  |
| C | 9 |  |

Câu lệnh:

**SELECT Sudent, sum(Point)**

**FROM Point**

**WHERE Point > 5**

**GROUP BY Student**

*Kết quả trả về:*

|  |  |
| --- | --- |
| **Student** | **Point** |
| A | 8 |
| C | 9 |

*=> Nó sẽ check WHERE từng dòng trước, sau đó mới GROUP BY*

Câu lệnh:

**SELECT Student, sum(Point) as S**

**FROM Point**

**WHERE Point > 5**

**GROUP BY Student**

**HAVING S > 8**

*Kết quả trả về:*

|  |  |
| --- | --- |
| **Student** | **Point** |
| C | 9 |

*=> HAVING thì mới dùng với các điều kiện của cả nhóm (GROUP) như max, sum, count, …được (gọi là các hàm tổng hợp (AGGREGATE function) được define trên 1 tập hợp)*

**Định nghĩa:**

- Cách 1:

+ Where thường dùng để tìm điều kiện cho thuộc tính của Bảng.

+ Having thường dùng để tìm điều kiện trên các Function (SUM,AVG,COUNT…) trên các thuộc tính của Bảng.

- Cách 2:

+ Where : Là câu lệnh điều kiện trả kết quả đối chiếu với từng dòng .

+ Having : Là câu lệnh điều kiện trả kết quả đối chiếu cho nhóm (Sum,AVG,COUNT,…)

***=> Vì vậy nên sau “GROUP BY” thì sẽ chỉ dùng được “HAVING” . Còn “WHERE” thì KHÔNG dùng được sau “GROUP BY”.***

***=> “HAVING” có thể thay thế vị trí dùng cho “WHERE” NHƯNG ngược lại “WHERE” thì KHÔNG thể thay thế vị trí cho “HAVING”.***

***=> “WHERE” phải trước “GROUP BY”.***

**BÀI THAM KHẢO SỐ 2**

**MỆNH ĐỀ WHERE**

*Chức năng chính:* Mệnh đề WHERE ***dùng để kiểm tra các điều kiện cho các thuộc tính của bảng, trả kết quả đối chiếu với từng dòng.***

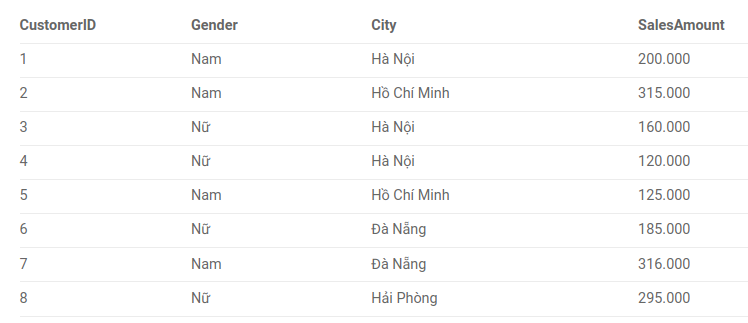
Syntax:

**SELECT** column\_lists

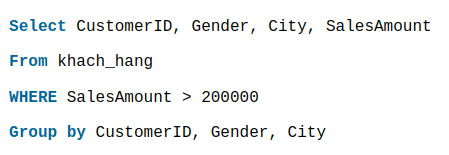
**FROM** table\_name

**WHERE** conditions

**GROUP BY** column\_lists

Ví dụ: Bảng khach\_hang

*Ví dụ khi thực hiện câu truy vấn:*

*Kết quả trả về:*



**MỆNH ĐỀ HAVING**

*Chức năng chính:*

- Mệnh đề HAVING ***được dùng để giới hạn nhóm các hàng trả về trong bảng***, ***trả kết quả đối chiếu cho các nhóm (Sum, Count, Average,…) được tạo bởi Group by.***

- Mệnh đề HAVING ***được thêm vào SQL vì mệnh đề WHERE không thể sử dụng được với các aggregate functions.***

=> ***Có thể sử dụng mệnh đề WHERE và HAVING cùng nhau trong quá trình lựa chọn.*** Trong trường hợp này, ***đầu tiên mệnh đề WHERE lọc các hàng riêng lẻ,*** ***sau đó các hàng được nhóm lại, thực hiện các phép tính tổng hợp*** và ***cuối cùng mệnh đề HAVING lọc các nhóm.***

Syntax:

**SELECT** column\_lists, aggregate\_function (expression)

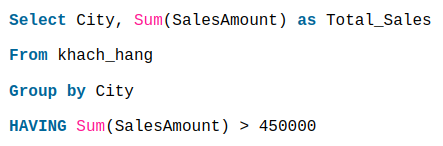
**FROM** table\_name

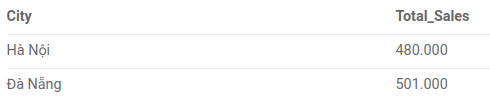
**WHERE** conditions (điều kiện)

**GROUP BY** column\_lists (danh sách các cột gom nhóm)

**HAVING** conditions (điều kiện trên nhóm)

*Ví dụ khi thực hiện câu truy vấn:*

*Kết quả trả về:*



**PHÂN BIỆT MỆNH ĐỀ WHERE VÀ HAVING TRONG SQL**

|  |  |  |
| --- | --- | --- |
| **Nội dung** | **WHERE** | **HAVING** |
| Cách thức Filter | Từng dòng riêng lẻ | Từng nhóm |
| Dùng với aggregate functions | Không | Có |
| Dùng với các câu lệnh | Select, Update, Delete | Chỉ dùng với Select |
| Dùng với Group by | Dùng trước Group by | Dùng sau Group by |

=> **Mệnh đề WHERE** khi được **sử dụng với GROUP BY**, **xuất hiện trước GROUP BY**. Điều này biểu thị rằng **các hàng bộ lọc WHERE trước khi tính toán tổng hợp được thực hiện**. Mặt khác, **HAVING** **xuất hiện sau GROUP BY**, có nghĩa là **nó lọc sau khi tính toán tổng hợp được thực hiện.**

=> Kết luận: **Cả 2 mệnh đề này đều có chức năng lọc dữ liệu**, tuy nhiên vẫn có một số điểm khác biệt ở cách thức lọc, kết hợp với các câu lệnh khác. Nhìn chung **mệnh đề HAVING được trang bị thêm một số tính năng mà mệnh đề WHERE không có**.

**KHÁC NHAU GIỮA DISTINCT VÀ GROUP BY**

**DISTINCT**

Chức năng:

Lệnh này cho phép bạn loại bỏ những bản ghi trùng lặp khi select. Điều này đặc biệt có ý nghĩa khi bảng được select bị trùng lặp dữ liệu trên một vài đầu thuộc tính hoặc khi join các bảng lại với nhau, làm tăng thêm những kết quả không thực sự cần đến.

Syntax:

SELECT DISTINCT Columns

FROM Table\_name

WHERE Where\_conditions

***Chú ý: Cả hai mệnh đề DISTINCT và GROUP BY đều làm giảm số lượng bản ghi trả về trong tậ kết quả bằng cách loại bỏ các bản ghi trùng lặp. Tuy nhiên, nên sử dụng mệnh đề GROUP BY khi muốn áp dụng các hàm tập hợp trên một hoặc nhiều cột.***

***Sử dụng DISTINCTnếu bạn chỉ muốn loại bỏ các bản sao. Sử dụng GROUPY BY nếu bạn muốn áp dụng các toán tử tổng hợp ( MAX, SUM, GROUP\_CONCAT, ... hoặc một mệnh đề HAVING).***

***Nếu bạn muốn nhóm các kết quả của mình, hãy sử dụng GROUP BY, nếu bạn chỉ muốn một danh sách duy nhất của một cột cụ thể, hãy sử dụng DISTINCT. Điều này sẽ giúp cơ sở dữ liệu của bạn có cơ hội tối ưu hóa truy vấn cho nhu cầu của bạn.***

***GROUP BY được sử dụng để nhóm các hàng mà bạn muốn tính toán. DISTINCT sẽ không thực hiện bất kỳ phép tính nào.***

**CÁC LOẠI RÀNG BUỘC (CONSTRAINT) PHỔ BIẾN**

|  |  |
| --- | --- |
| **Loại ràng buộc** | **Ý nghĩa** |
| NOT NULL | Sử dụng để đảm bảo dữ liệu của cột không được nhận giá trị NULL |
| DEFAULT | Gán giá trị mặc định trong trường hợp dữ liệu của cột không được nhập vào hay không được xác định. |
| UNIQUE | Sử dụng để đảm bảo dữ liệu của cột là duy nhất, không trùng lặp giá trị trên cùng 1 cột. |
| PRIMARY KEY (Khóa chính) | Dùng để thiết lập khóa chính trên bảng, xác định giá trị trên tập các cột làm khóa chính phải là duy nhất, không được trùng lặp. Việc khai báo ràng buộc khóa chính yêu cầu các cột phải NOT NULL. |
| FOREIGN KEY (Khóa ngoại) | Dùng để thiết lập khóa ngoại trên bảng, tham chiếu đến bảng khác thông qua giá trị của cột được liên kết. Giá trị của cột được liên kết phải là duy nhất trong bảng kia. |
| CHECK | Bảo đảm tất cả giá trị trong cột thỏa mãn điều kiện nào đó. Đây là hình thức sử dụng phổ biến để kiểm tra tính hợp lệ của dữ liệu (validate data) |

RÀNG BUỘC UNIQUE VÀ RÀNG BUỘC PRIMARY KEY

KHÁI NIỆM PRIMARY KEY

- Một cột có thể được gọi là primary key của bảng nếu nó xác định duy nhất từng bộ (hàng) trong bảng đó. Nó thực thi các ràng buộc toàn vẹn đối với bảng. Chỉ có một primary key trong một bảng. Primary key không chấp nhận các giá trị trùng lặp và NULL. Primary key được chọn cẩn thận khi các thay đổi có thể xảy ra theo cách hiếm khi xảy ra, có nghĩa là khóa chính trong bảng rất hiếm khi thay đổi.

- Primary key có thể được tham chiếu bằng foreign key. Nó tạo ra một chỉ mục nhóm duy nhất trên bảng. Trong một chỉ mục được phân nhóm, các hàng dữ liệu được sắp xếp và lưu trữ trong một bảng hoặc các views trên cơ sở các giá trị chính của nó. Chỉ có thể có một chỉ mục nhóm trong một bảng, lý do đằng sau điều này là một hàng dữ liệu trong bảng chỉ có thể được sắp xếp theo một thứ tự.

KHÁI NIỆM UNIQUE

- Tương tự như primary key, ràng buộc unique key cũng xác định duy nhất một bộ giá trị riêng lẻ trong một mối quan hệ. Nhưng, có những khác biệt nhất định giữa chúng. Một bảng có thể có nhiều hơn một khóa duy nhất. Ràng buộc unique key chỉ có thể chấp nhận một giá trị NULL cho một cột.

- Các ràng buộc duy nhất cũng được tham chiếu bởi foreign key. Nó có thể được sử dụng khi ai đó muốn thực thi các ràng buộc trên một cột và một nhóm các cột không phải là primary key. Không giống như primary key, nó tạo chỉ mục không phân cụm. Các chỉ mục không phân cụm có cấu trúc khác biệt với các hàng dữ liệu. Mỗi mục nhập khóa-giá trị trong đó trỏ đến hàng dữ liệu chứa giá trị khóa do đó nó sử dụng con trỏ.

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Primary Key** | **Unique Key** |
| Công dụng | Nó được sử dụng để làm định danh duy nhất cho mỗi hàng trong bảng. | Nó cũng xác định duy nhất một hàng, nhưng không là khóa chính. |
| NULL | không chấp nhận giá trị NULL | chấp nhận giá trị NULL |
| Số lượng khóa có thể được xác định trong bảng | Chỉ có duy nhất 1 primary key trong 1 bảng | Có thể nhiều hơn 1 |
| Index | Tạo chỉ mục theo nhóm | Tạo chỉ mục không phân cụm |
| Tự động tăng | Hỗ trợ giá trị gia tăng tự động. | Không hỗ trợ giá trị gia tăng tự động. |
| Sửa đổi | Không thể thay đổi hoặc xóa các giá trị được lưu trữ trong các khóa chính. | Có thể thay đổi các giá trị khóa duy nhất. |

- Giống: PRIMARY KEY, IDENTITY, và Unique đều có thể được sử dụng để đảm bảo tính toàn vẹn của dữ liệu.

- Khác:

+ PRIMARY KEY(PK): ràng buộc PK được dùng để định nghĩa khóa chính của bảng. Nó đảm bảo rằng không có giá trị của hàng nào trong cột là giống nhau. Hay là nói 1 cách dễ hiểu ra là nó xác định được tính duy nhất của bản ghi. Ràng buộc PK không cho phép giá trị NULL

+ Unique: Để thay cho việc không sử dụng khóa chính, bạn có thể dùng ràng buộc Unique. Nó cũng xác định được tính duy nhất của bản ghi. Nhưng bạn lưu ý rằng, nó có thể xác định tính duy nhất của bản ghi chứ nó không phải là khóa chính. Ràng buộc Unique cho phép giá trị NULL

+ Identity: Theo lý thuyết thì Identity được sử dụng để tự động sinh ra các giá trị của 1 cột 1 cách tuần tự.

Thực hành thì dễ hơn nhiều. )

IDENTITY(giá trị đầu,giâ trị tăng)

Ví dụ: IDENTITY(1,1) --> Nghĩa là: bắt đầu từ một và sau mỗi bản ghi được chèn vào giá trị nó tăng lên 1

Cũng như Unique, Identity nó cũng đc sử dụng để xác định tính duy nhất của bản ghi và nó không phải là khóa chính.

**SQL SEVER VÀ POSTGRE SQL**

SQL SEVER

Ngôn ngữ truy vấn có cấu trúc hoặc SQL, thường được biết đến nhiều hơn, là ngôn ngữ lập trình được sử dụng để quản lý, truy vấn và truy xuất dữ liệu trong cơ sở dữ liệu quan hệ . Đây là ngôn ngữ tiêu chuẩn được sử dụng bởi các hệ thống quản lý cơ sở dữ liệu quan hệ (RDBMS), bao gồm PostgreSQL, SQL Server, MySQL và Cơ sở dữ liệu Oracle.

SQL thường sử dụng các lệnh được viết ở định dạng câu lệnh cho các truy vấn và các hoạt động cơ sở dữ liệu khác, cho phép người dùng thao tác dữ liệu trong các bảng cơ sở dữ liệu quan hệ. Mặc dù ban đầu được tạo cho cơ sở dữ liệu quan hệ, nhưng SQL đóng vai trò là nền tảng cho nhiều khả năng công nghệ ngày nay, khiến kiến ​​thức SQL trở thành một kỹ năng thiết yếu cho nhiều vai trò công nghệ ngày nay, bao gồm nhà phân tích dữ liệu, kỹ sư cơ sở dữ liệu và thậm chí cả lập trình phụ trợ.

POSTGRE SQL

PostgreSQL là một hệ thống quản lý cơ sở dữ liệu quan hệ đối tượng nguồn mở, được phát hành theo Giấy phép PostgreSQL. Nó hỗ trợ cả quan hệ (SQL) và không quan hệ (JSON) và cung cấp các hàm SQL nâng cao, bao gồm khóa ngoại, truy vấn phụ và trình kích hoạt. PostgreSQL cũng có khả năng mở rộng cao, cho phép bạn xác định các kiểu dữ liệu và tạo các hàm tùy chỉnh.

SỰ TƯƠNG ĐỒNG GIỮA SQL SEVER VÀ POSTGRE SQL

Nói chung, SQL Server và PostgreSQL là một trong những hệ thống quản lý cơ sở dữ liệu quan hệ phổ biến nhất hiện có dựa trên hiệu suất, bảo mật, khả năng mở rộng và khả năng sử dụng của chúng. Cả hai đều bao gồm các tính năng cơ sở dữ liệu quan hệ và cung cấp khả năng tương thích với nhiều ứng dụng doanh nghiệp lớn và nhỏ.

Thông thường, SQL Server là lựa chọn cho các tổ chức lớn hơn phụ thuộc vào các sản phẩm của Microsoft. Tuy nhiên, PostgreSQL đã tạo ra không gian cho chính nó như một hệ thống quản lý cơ sở dữ liệu miễn phí, dễ triển khai, mang lại tính linh hoạt và chức năng tối đa.

SỰ KHÁC BIỆT GIỮA SQL SEVER VÀ POSTGRE SQL

|  |  |
| --- | --- |
| **SQL Sever** | **PostgreSQL** |
| Hệ thống quản lý cơ sở dữ liệu quan hệ | Hệ quản trị cơ sở dữ liệu quan hệ đối tượng |
| Sản phẩm thương mại của Microsoft | Mã nguồn mở (Hoàn toàn miễn phí) |
| Chỉ chạy trên Microsoft hoặc Linux | Chạy trên hầu hết các máy và hệ điều hành |
| Sử dụng Transact-SQL hoặc T-SQL (SQL tiêu chuẩn + chức năng bổ sung) | Sử dụng SQL chuẩn |

Bảng so sánh 2:

|  |  |  |
| --- | --- | --- |
|  | **SQL Sever** | **PostgreSQL** |
| **SELECT** | Select [col1], [col2] | SELECT col1, col2 |
| **Dữ liệu từ các bảng có phân biệt chữ hoa chữ thường không?** | WHERE name = ‘John’ WHERE name = ‘john’  => Không giống nhau | WHERE name = ‘John’ WHERE name = ‘john’  => Không giống nhau |
| **Sử dụng dấu ngoặc kép** | Name = ‘John’ | Name = ‘John’ |
| **Bí danh cho cột và bảng** | SELECT AVG(col1)=avg1 | SELECT AVG(col1) AS avg1 |
| **Làm việc với ngày tháng** | GETDATE()  DATEPART() | CURRENT\_DATE() CURRENT\_TIME() EXTRACT() |

Bảng so sánh 3:

|  |  |
| --- | --- |
| **TẠO DATABASE** | |
| **SQL Sever** | **PostgreSQL** |
| CREATE DATABASE mydb | - Cách thứ nhất sử dụng câu lệnh createdb:  $ createdb mydb  => Chú ý: Câu này thực hiện khi bạn login vào user postgres của hệ điều hành, chứ không cần phải vào trong psql để chạy  => Kết nối đến database PostgreSQL bằng psql và gõ lệnh \l để hiển thị các database trong database cluster, bao gồm cả database mydb mới tạo  - Cách thứ hai sử dụng câu lệnh CREATE DATABASE khi đã kết nối bằng psql:  CREATE DATABASE mydb |
| → Khác nhau | |

|  |  |
| --- | --- |
| **XÓA DATABASE** | |
| **SQL Sever** | **PostgreSQL** |
| DROP DATABASE mydb | - Cách thứ nhất sử dụng câu lệnh dropdb  $ dropdb mydb  - Cách thứ 2 sử dụng câu lệnh DROP DATABASE  DROP DATABASE mydb |
| → Khác nhau | |

|  |  |
| --- | --- |
| **TẠO TABLE** | |
| **SQL Sever** | **PostgreSQL** |
| CREATE TABLE SinhVien  (  MASV nchar(10),  TENSV nvarchar(100),  KHOA nvarchar(100)  ) | CREATE TABLE SinhVien  (  MASV nchar(10),  TENSV nvarchar(100),  KHOA nvarchar(100)  ); |
| → Giống nhau | |

|  |  |
| --- | --- |
| **ĐỔI TÊN TABLE** | |
| **SQL Sever** | **PostgreSQL** |
| EXEC sp\_rename 'SinhVien', 'HocSinh'; | ALTER TABLE SinhVien  RENAME TO HocSinh |
| → Khác nhau | |

|  |  |
| --- | --- |
| **XÓA TABLE** | |
| **SQL Sever** | **PostgreSQL** |
| DROP TABLE mydb | DROP TABLE mydb; |
| → Giống nhau | |

|  |  |
| --- | --- |
| **ĐỔI TÊN COLUMN TRONG TABLE** | |
| **SQL Sever** | **PostgreSQL** |
| EXEC sp\_rename 'HocSinh.MaSV', 'MaHS', 'COLUMN'; | ALTER TABLE HocSinh RENAME COLUMN MaSV TO MaHS; |
| → Khác nhau | |

|  |  |
| --- | --- |
| **THAY ĐỔI KIỂU DỮ LIỆU** | |
| **SQL Sever** | **PostgreSQL** |
| ALTER TABLE SinhVien  ALTER COLUMN MaHS char(10) | ALTER TABLE SinhVien  ALTER COLUMN MaHS TYPE char(10); |
| → Khác nhau | |

|  |  |
| --- | --- |
| **THÊM MỘT COLUMN VÀO TABLE** | |
| **SQL Sever** | **PostgreSQL** |
| ALTER TABLE HocSinh  ADD DiaChi nvarchar(50) | ALTER TABLE HocSinh  ADD COLUMN DiaChi nvarchar(50); |
| → Khác nhau | |

|  |  |
| --- | --- |
| **XÓA MỘT COLUMN KHỎI TABLE** | |
| **SQL Sever** | **PostgreSQL** |
| ALTER TABLE HocSinh  DROP DiaChi; | ALTER TABLE HocSinh  DROP COLUMN DiaChi; |
| → Khác nhau | |

|  |  |
| --- | --- |
| **THÊM DỮ LIỆU VÀO BẢNG** | |
| **SQL Sever** | **PostgreSQL** |
| INSERT INTO HocSinh  VALUES(‘CK101’, N’Nguyễn Nam’, N’Cơ Khí’) | INSERT INTO HocSinh  VALUES (‘CK101’, N’Nguyễn Nam’, N’Cơ Khí’); |
| → Giống nhau | |

|  |  |
| --- | --- |
| **CẬP NHẬT DỮ LIỆU BẢNG** | |
| **SQL Sever** | **PostgreSQL** |
| UPDATE HocSinh  SET KHOA = N'Công nghệ Kỹ Thuật Cơ khí'  WHERE MaHS = ‘CK101’ | UPDATE HocSinh  SET KHOA = N'Công nghệ Kỹ Thuật Cơ khí'  WHERE MaHS = ‘CK101’; |
| → Giống nhau | |

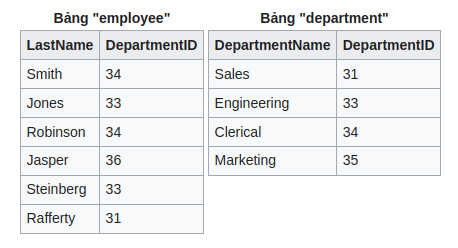
|  |  |
| --- | --- |
| **XÓA DỮ LIỆU BẢNG** | |
| **SQL Sever** | **PostgreSQL** |
| DELETE HocSinh  WHERE MaHS = ‘CK101’ | DELETE FROM HocSinh  WHERE MaHS = ‘CK101’; |
| → Khác nhau | |

|  |  |
| --- | --- |
| **THÊM RÀNG BUỘC PRIMARY KEY CHO COLUMN CỦA TABLE ĐÃ TẠO** | |
| **SQL Sever** | **PostgreSQL** |
| ALTER TABLE HocSinh  ADD CONSTRAINT pk\_hs PRIMARY KEY (MaHS); | ALTER TABLE HocSinh  ADD CONSTRAINT pk\_hs PRIMARY KEY (MaHS); |
| → Giống nhau | |

**TÌM HIỂU VỀ PHÉP KẾT NỐI JOIN**

**ĐỊNH NGHĨA: Join dùng để kết nối nhiều bản ghi từ 2 bảng dữ liệu trong 1 cơ sở dữ liệu quan hệ và kết quả được đưa vào một bảng (tạm).** Trong ngôn ngữ truy vấn theo cấu trúc (SQL) **có ba loại kết hợp sau: nội, ngoại và chéo.** **Kết ngoại được chia ra thêm thành kết ngoại bên trái (left outer join), kết ngoại bên phải (right outer join), và kết ngoại đủ (full outer join)**.

*Ví dụ trên bảng*

**Cần chú ý:** DepartmentID của nhân viên Jasper không có dòng tương ứng nào trong bảng department, và phòng Marketing thì không có dòng tương ứng trong bảng employee.

**CÁC HÌNH THỨC KẾT HỢP:**

**KẾT NỘI**

**Phép kết nội (inner join)** thực chất là **tìm giao của hai bảng dữ liệu**. Đây là loại kết hợp thường được dùng nhất và được xem như là phép kết hợp mặc định.

Ví dụ dưới đây về phép kết hợp *trích tất cả các bản ghi trong bảng dữ liệu A (employee)* và *tìm những bản ghi tương ứng trong bảng B (department)*. *Nếu không tìm được, những bản ghi trong A sẽ không được đưa vào kết quả.* *Nếu có nhiều kết quả tìm được trong B tương ứng với vị từ, thì 1 dòng sẽ được trả về ứng với mỗi vị từ (giá trị ở bảng A sẽ được lặp)*.

**Cần đặc biệt chú ý khi kết hợp những bảng trên những cột mà có thể là NULL vì giá trị NULL sẽ không bao giờ có tương ứng.**

Ví dụ về phép kết nội (cú pháp chuẩn ANSI 92):

**SELECT** \*

**FROM** employee

**INNER JOIN** department

**ON** employee.DepartmentID = department.DepartmentID

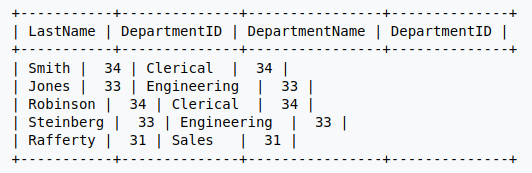
Ví dụ về phép kết nội (cú pháp không chuẩn):

**SELECT** \*

**FROM** employee, department

**WHERE** employee.DepartmentID = department.DepartmentID

Kết quả kết hợp:

*Chú ý rằng nhân viên Jasper và phòng Marketing không xuất hiện. Không một bản nào trong những bản này có dòng tương ứng trong bảng kết hợp của chúng, nên đã bị loại ra khỏi kết quả của phép kết nội này.*

**KẾT NGOẠI BÊN TRÁI**

**Phép kết ngoại bên trái (left outer join)** khác rất nhiều với 1 phép kết nội. Thay vì giới hạn những kết quả thu được trong cả hai bảng, nó **chỉ giới hạn đối với những kết quả ở bảng bên trái (A)**. **Nghĩa là nếu mệnh đề ON không có bản ghi tương ứng bên bảng B, 1 dòng trong kết quả vẫn được trả về nhưng với giá trị NULL cho mỗi cột trong bảng B.**

**Nó trả về tất cả những giá trị từ bản bên trái + những giá trị tương ứng với bảng bên phải hoặc là null (khi những giá trị ở bảng bên phải không tương ứng)**

*Chẳng hạn*, nó cho phép ta tìm phòng ban của nhân viên, *nhưng vẫn trả về nhân viên ngay cả khi phòng ban của người ấy là NULL hoặc không tồn tại. Ví dụ ở trên sẽ bỏ qua những nhân viên trong phòng ban không tồn tại.*

Ví dụ về phép kết ngoại bên trái (cú pháp chuẩn ANSI 92):

**SELECT** **distinct** \*

**FROM** employee

**LEFT OUTER JOIN** department

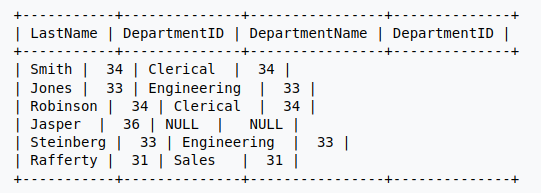
**ON** employee.DepartmentID = department.DepartmentID

Ví dụ về phép kết ngoại bên trái (cú pháp không chuẩn):

**SELECT** \*

**FROM** employee, department

**WHERE** employee.DepartmentID = department.DepartmentID



**KẾT NGOẠI BÊN PHẢI**

**Phép kết ngoại bên phải (right outer join)** hầu như **tương tự với phép kết ngoại bên trái,** **trừ 1 điều là thứ tự các bảng đổi lại**. **Mỗi bản ghi từ bảng bên phải, B hoặc department sẽ được trả về và giá trị NULL sẽ được trả về cho những dòng mà không có bản ghi tương ứng bên bảng A.**

**Nó trả về tất cả những giá trị từ bảng bên phải + giá trị tương ứng từ bảng bên trái (hoặc null)**

Ví dụ về phép kết ngoại bên phải (cú pháp chuẩn ANSI 92):

**SELECT** \*

**FROM** employee

**RIGHT OUTER JOIN** department

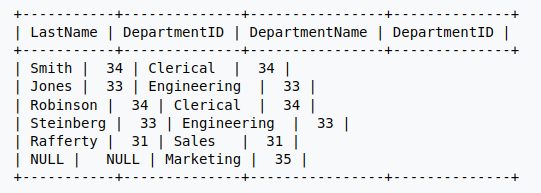
**ON** employee.DepartmentID = department.DepartmentID

Ví dụ về phép kết ngoại bên phải (cú pháp không chuẩn):

**SELECT** \*

**FROM** employee, department

**WHERE** employee.DepartmentID = department.DepartmentID

**KẾT NGOẠI ĐỦ**

**Phép kết ngoại đủ (full outer join) kết hợp cả kết quả của cả phép kết ngoại bên trái và phép kết ngoại bên phải. Những phép kết này đưa ra bản ghi của cả hai bảng dữ liệu, và lấp đầy những dòng tương ứng bị thiếu của cả hai phía bằng NULLs.**

*Một vài hệ cơ sở dữ liệu không hỗ trợ chức năng này, nhưng nó có thể được thay thế bằng việc dùng phép kết ngoại bên trái và bên phải và phép hợp (union). (Xem dưới đây)*

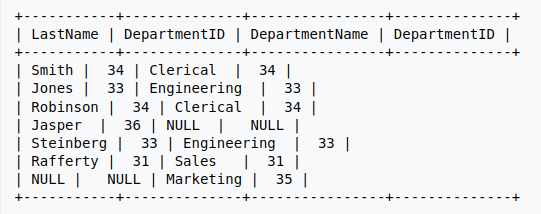
Ví dụ về phép kết ngoại đủ (cú pháp chuẩn ANSI 92):

**SELECT** \*

**FROM** employee

**FULL OUTER JOIN** department

**ON** employee.DepartmentID = department.DepartmentID

*Cùng 1 ví dụ, cho những cơ sở dữ liệu không hỗ trợ phép kết ngoại đủ:*

**SELECT** employee.LastName, employee.DepartmentID, department.DepartmentName, department.DepartmentID

**FROM** employee

**LEFT JOIN** department

**ON** employee.DepartmentID = department.DepartmentID

**UNION**

**SELECT** employee.LastName, employee.DepartmentID, department.DepartmentName, department.DepartmentID

**FROM** employee

**RIGHT JOIN** department

**ON** employee.DepartmentID = department.DepartmentID

**WHERE** employee.DepartmentID IS NULL

**KẾT CHÉO**

Mặc dù không được sử dụng thường xuyên, **phép kết chéo (cross join)** **là cơ sở mà dựa trên đó phép kết nội được tạo nên.** Một phép kết chéo **trả về tích Descartes của những tập hợp các dòng từ những bảng được kết.**

*Câu SQL dùng cho phép kết chéo liệt kê những bảng dữ liệu sẽ được kết (FROM), nhưng không bao gồm bất cứ vị từ lọc nào (WHERE).*

Ví dụ về phép kết chéo (cú pháp chuẩn ANSI 92):

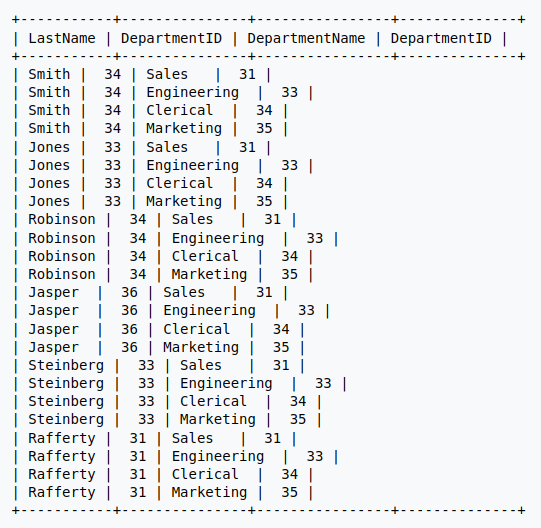
**SELECT** \*

**FROM** employee **CROSS JOIN** department;

Ví dụ về phép kết chéo (cú pháp thay thế):

**SELECT** \*

**FROM** employee, department;

Ta có thể thấy ***phép kết chéo không tìm những bản ghi tương ứng***. Phép kết này ***ít khi được dùng***, ***ngoại trừ để phát sinh tất cả những kết hợp có khả năng của các bản ghi từ các bảng dữ liệu mà không chia sẻ phần tử chung.***

Link Unique:

<https://freetuts.net/rang-buoc-unique-trong-sql-server-1603.html>

Link Join:

<https://vimentor.com/vi/lesson/03-left-join-va-right-join>

Link truy vấn nhiều bảng:

<https://xuanthulab.net/truy-van-nhieu-bang-voi-join-trong-sql.html>

Link các mệnh đề:

<https://viblo.asia/p/co-so-du-lieu-la-gimysql-mot-so-cau-lenh-truy-van-co-so-du-lieu-thuong-dung-gDVK26AAKLj>