PostgreSQL 15.2 Documentation

# Part 0. Preface

## **Chapter** 1. PostgreSQL là gì ?

PostgreSQL là một hệ thống quản lý cơ sở dữ liệu quan hệ đối tượng ( ORDBMS ) dựa trên POSTGRES, Phiên bản 4.2 , được phát triển tại Khoa Khoa học Máy tính của Đại học California tại Berkeley.

PostgreSQL là hậu duệ nguồn mở của mã Berkeley ban đầu này. Nó hỗ trợ một phần lớn tiêu chuẩn SQL và cung cấp nhiều tính năng hiện đại:

* Complex queries
* Foreign keys
* Triggers
* Updatable views
* Transactional integrity
* Multiversion concurrency control

Ngoài ra, người dùng có thể mở rộng PostgreSQL theo nhiều cách, ví dụ bằng cách thêm mới

* Data types
* Functions
* Operators
* Aggregate functions
* Index methods
* Procedural languages

## **Chapter** 2. Quy ước

Các quy ước sau đây được sử dụng trong phần tóm tắt của lệnh: dấu ngoặc vuông ( [and]) biểu thị các phần tùy chọn. Dấu ngoặc nhọn ( {and}) và đường thẳng đứng (|) cho biết bạn phải chọn một phương án. Dấu chấm (...) có nghĩa là phần tử đứng trước có thể được lặp lại. Tất cả các ký hiệu khác, kể cả dấu ngoặc đơn, nên được hiểu theo nghĩa đen.

# Part 1. Tutorial

## Chapter 1. Getting Started

### 1.1. Creating a Database

Một máy chủ PostgreSQL đang chạy có thể quản lý nhiều cơ sở dữ liệu. Thông thường, một cơ sở dữ liệu riêng biệt được sử dụng cho từng dự án hoặc cho từng người dùng.

Để tạo một cơ sở dữ liệu mới, trong ví dụ này có tên là mydb, bạn sử dụng lệnh sau:

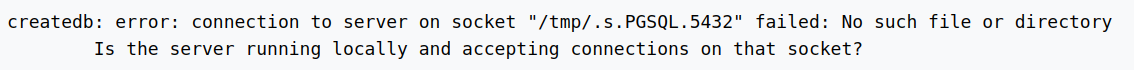
Nếu điều này không tạo ra phản hồi thì bước này đã thành công và bạn có thể bỏ qua phần còn lại của phần này.

Nếu bạn thấy một thông báo tương tự như:

Thì PostgreSQL không được cài đặt đúng cách. Nó hoàn toàn không được cài đặt hoặc đường dẫn tìm kiếm của trình bao của bạn không được đặt để bao gồm nó. Thay vào đó, hãy thử gọi lệnh bằng một đường dẫn tuyệt đối:

Đường dẫn tại trang web của bạn có thể khác. Liên hệ với quản trị viên trang web của bạn hoặc kiểm tra hướng dẫn cài đặt để khắc phục tình trạng này.

Một thông báo khác có thể là thế này:

Điều này có nghĩa là máy chủ chưa được khởi động hoặc nó không lắng nghe nơi createdbcần liên hệ với nó. Một lần nữa, hãy kiểm tra hướng dẫn cài đặt hoặc tham khảo ý kiến của quản trị viên.

Một thông báo khác có thể là thế này:

Điều này sẽ xảy ra nếu quản trị viên chưa tạo tài khoản người dùng PostgreSQL cho bạn. ( Tài khoản người dùng PostgreSQL khác với tài khoản người dùng hệ điều hành). Bạn sẽ cần trở thành người dùng hệ điều hành mà PostgreSQL đã được cài đặt (thường là postgres) để tạo tài khoản người dùng đầu tiên. Cũng có thể là bạn đã được chỉ định một tên người dùng PostgreSQL khác với tên người dùng hệ điều hành của bạn; trong trường hợp đó, bạn cần sử dụng khóa chuyển **-U** hoặc đặt biến môi trường **PGUSER** để chỉ định tên người dùng PostgreSQL của mình .

Nếu bạn có tài khoản người dùng nhưng nó không có các đặc quyền cần thiết để tạo cơ sở dữ liệu, bạn sẽ thấy như sau:

Không phải mọi người dùng đều có quyền tạo cơ sở dữ liệu mới. Nếu PostgreSQL từ chối tạo cơ sở dữ liệu cho bạn thì quản trị viên trang web cần cấp cho bạn quyền tạo cơ sở dữ liệu. Tham khảo ý kiến quản trị viên trang web của bạn nếu điều này xảy ra. Nếu bạn đã tự cài đặt PostgreSQL thì bạn nên đăng nhập cho mục đích của hướng dẫn này bằng tài khoản người dùng mà bạn đã khởi động máy chủ.

|  |
| --- |
| ***NOTE***  *“Tên người dùng PostgreSQL tách biệt với tài khoản người dùng hệ điều hành. Khi bạn kết nối với cơ sở dữ liệu, bạn có thể chọn tên người dùng PostgreSQL để kết nối; nếu không, nó sẽ mặc định có cùng tên với tài khoản hệ điều hành hiện tại của bạn. Khi điều đó xảy ra, sẽ luôn có một tài khoản người dùng PostgreSQL có cùng tên với người dùng hệ điều hành đã khởi động máy chủ và điều đó cũng xảy ra là người dùng đó luôn có quyền tạo cơ sở dữ liệu. Thay vì đăng nhập với tư cách người dùng đó, bạn cũng có thể chỉ định-Utùy chọn ở mọi nơi để chọn tên người dùng PostgreSQL để kết nối.”* |

Bạn cũng có thể tạo cơ sở dữ liệu với các tên khác. PostgreSQL cho phép bạn tạo bất kỳ số lượng cơ sở dữ liệu nào tại một trang nhất định. Tên cơ sở dữ liệu phải có ký tự chữ cái đầu tiên và được giới hạn ở độ dài 63 byte. Một lựa chọn thuận tiện là tạo một cơ sở dữ liệu có cùng tên với tên người dùng hiện tại của bạn. Nhiều công cụ coi tên cơ sở dữ liệu đó là mặc định, vì vậy nó có thể giúp bạn tiết kiệm một số thao tác nhập. Để tạo cơ sở dữ liệu đó, chỉ cần gõ:

Nếu bạn không muốn sử dụng cơ sở dữ liệu của mình nữa, bạn có thể xóa nó. Ví dụ: nếu bạn là chủ sở hữu (người tạo) của cơ sở dữ liệu mydb, bạn có thể hủy nó bằng lệnh sau:

(Đối với lệnh này, tên cơ sở dữ liệu không mặc định là tên tài khoản người dùng. Bạn luôn cần chỉ định nó.) Hành động này sẽ xóa tất cả các tệp được liên kết với cơ sở dữ liệu theo cách vật lý và không thể hoàn tác, vì vậy, điều này chỉ nên được thực hiện với mức độ nghiêm trọng. của sự suy nghĩ trước.

|  |
| --- |
| *Thông tin thêm về createdbvà dropdbcó thể được tìm thấy trong* [*createdb*](https://www-postgresql-org.translate.goog/docs/current/app-createdb.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) *và* [*dropdb*](https://www-postgresql-org.translate.goog/docs/current/app-dropdb.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) *tương ứng.* |

### 1.2. Accessing a Database

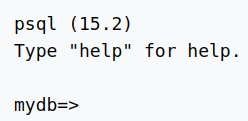
Khi bạn đã tạo cơ sở dữ liệu, bạn có thể truy cập nó bằng cách:

* Chạy chương trình PostgreSQL interactive terminal program, được gọi là psql , cho phép bạn nhập, chỉnh sửa và thực thi các lệnh SQL một cách tương tác .
* Sử dụng pgAdmin hoặc sử dụng các bộ office có hỗ trợ ODBC hoặc JDBC để tạo và thao tác cơ sở dữ liệu.

Bạn có thể muốn bắt đầu psql để thử các ví dụ trong hướng dẫn này. Nó có thể được kích hoạt cho cơ sở dữ liệu mydb bằng cách gõ lệnh:

Nếu bạn không cung cấp tên cơ sở dữ liệu thì nó sẽ mặc định là tên tài khoản người dùng của bạn.

Trong psql, bạn sẽ được chào đón với thông báo:

Dòng cuối cùng cũng có thể là:

Điều đó có nghĩa là bạn là một siêu người dùng cơ sở dữ liệu, điều này rất có thể xảy ra nếu bạn tự cài đặt phiên bản PostgreSQL . Trở thành siêu người dùng có nghĩa là bạn không phải chịu sự kiểm soát truy cập.

Nếu bạn gặp sự cố khi bắt đầu, psqlhãy quay lại phần trước. Các chẩn đoán của createdbvà psqltương tự nhau, và nếu cái trước hoạt động thì cái sau cũng hoạt động.

Dòng cuối cùng do psql in ra là lời nhắc và nó cho biết rằng psql đang lắng nghe bạn và bạn có thể nhập các truy vấn SQL vào một không gian làm việc do psql duy trì. Hãy thử các lệnh sau:

Chương trình psql có một số lệnh bên trong không phải là lệnh SQL. Chúng bắt đầu bằng ký tự gạch chéo ngược, “\”. Ví dụ: bạn có thể nhận trợ giúp về cú pháp của các lệnh SQL PostgreSQL khác nhau bằng cách nhập:

Để thoát khỏi psql, gõ:

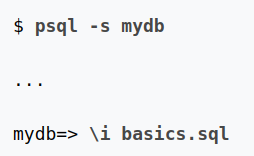


## Chapter 2. The SQL Language

### 2.1. Introduction

Giả định rằng bạn đã tạo một cơ sở dữ liệu có tên mydb, như được mô tả trong chương trước và đã có thể khởi động psql.

Thao tác này tạo các tập lệnh và biên dịch các tệp C chứa các hàm và kiểu do người dùng định nghĩa. Sau đó, để bắt đầu hướng dẫn, hãy làm như sau:

Lệnh \i đọc các lệnh từ tệp được chỉ định. Tùy chọn -s của psql đưa bạn vào chế độ một bước sẽ tạm dừng trước khi gửi từng câu lệnh đến máy chủ. Các lệnh dùng trong phần này nằm trong file basics.sql.

### 2.2. Giới thiệu

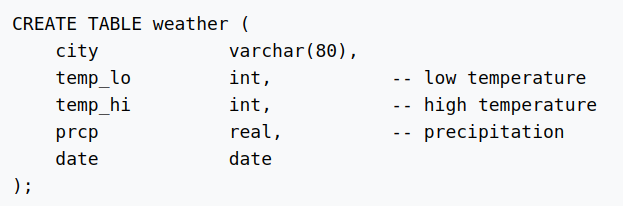
PostgreSQL là một hệ thống quản lý cơ sở dữ liệu quan hệ (RDBMS). Điều đó có nghĩa là nó là một hệ thống quản lý dữ liệu được lưu trữ trong các quan hệ. Quan hệ thực chất là một thuật ngữ toán học cho bảng.

Mỗi bảng là một tập hợp các hàng được đặt tên. Mỗi hàng của một bảng nhất định có cùng một tập hợp các cột được đặt tên và mỗi cột thuộc một kiểu dữ liệu cụ thể. Trong khi các cột có thứ tự cố định trong mỗi hàng, điều quan trọng cần nhớ là SQL không đảm bảo thứ tự của các hàng trong bảng theo bất kỳ cách nào (mặc dù chúng có thể được sắp xếp rõ ràng để hiển thị).

Các bảng được nhóm thành các cơ sở dữ liệu và một tập hợp các cơ sở dữ liệu được quản lý bởi một phiên bản máy chủ PostgreSQL duy nhất tạo thành một cụm cơ sở dữ liệu.

### 2.3. Creating a New Table

Bạn có thể tạo một bảng mới bằng cách chỉ định tên bảng, cùng với tất cả các tên cột và loại của chúng:

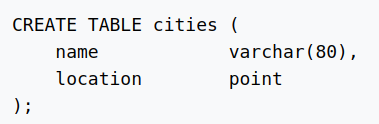
Bạn có thể nhập cái này vào psql bằng cách ngắt dòng. psql sẽ nhận ra rằng lệnh không bị chấm dứt cho đến khi có dấu chấm phẩy.

Khoảng trắng (tức là khoảng trắng, tab và dòng mới) có thể được sử dụng tự do trong các lệnh SQL. Điều đó có nghĩa là bạn có thể gõ lệnh được căn chỉnh khác với ở trên hoặc thậm chí tất cả trên một dòng. Hai dấu gạch ngang (“--”) giới thiệu nhận xét. Bất cứ thứ gì theo sau chúng đều bị bỏ qua cho đến cuối dòng. SQL không phân biệt chữ hoa chữ thường đối với các từ khóa và mã định danh, ngoại trừ khi mã định danh được trích dẫn kép để giữ nguyên chữ hoa/thường (không được thực hiện ở trên).

varchar(80) chỉ định một kiểu dữ liệu có thể lưu trữ các chuỗi ký tự tùy ý có độ dài tối đa 80 ký tự. int là kiểu số nguyên bình thường. real là một loại để lưu trữ các số dấu phẩy động có độ chính xác đơn. date nên được tự giải thích. cột của loại ngày cũng được đặt tên là ngày. Điều này có thể thuận tiện hoặc gây nhầm lẫn.

PostgreSQL hỗ trợ các kiểu SQL tiêu chuẩn int, smallint, real, double precision, char(N), varchar(N), date, time, timestamp và interval, cũng như các kiểu tiện ích chung khác và một tập hợp phong phú các kiểu hình học. PostgreSQL có thể được tùy chỉnh với số lượng tùy ý các loại dữ liệu do người dùng xác định. Do đó, tên loại không phải là từ khóa trong cú pháp, trừ khi được yêu cầu hỗ trợ các trường hợp đặc biệt trong tiêu chuẩn SQL.

Ví dụ thứ hai sẽ lưu trữ các thành phố và vị trí địa lý liên quan của chúng:

Loại point là một ví dụ về loại dữ liệu dành riêng cho PostgreSQL..

Cuối cùng, cần lưu ý rằng nếu bạn không cần một bảng nữa hoặc muốn tạo lại nó theo cách khác, bạn có thể xóa nó bằng lệnh sau:

### 2.4. Populating a Table With Rows

Câu lệnh INSERT được sử dụng để điền các giá trị vào một bảng

Lưu ý rằng tất cả các loại dữ liệu đều sử dụng các định dạng đầu vào khá rõ ràng. Các hằng số không phải là giá trị số đơn giản thường phải được bao quanh bởi dấu nháy đơn ('), như trong ví dụ. Kiểu **date** thực sự khá linh hoạt trong những gì nó chấp nhận, nhưng đối với hướng dẫn này, chúng tôi sẽ sử dụng định dạng rõ ràng được hiển thị ở đây.

Loại **point** yêu cầu một cặp tọa độ làm đầu vào, như được hiển thị ở đây:

Một cú pháp thay thế cho phép bạn liệt kê các cột một cách rõ ràng:

Bạn có thể liệt kê các cột theo thứ tự khác nếu muốn hoặc thậm chí bỏ qua một số cột, ví dụ: nếu prcp không xác định:

Bạn cũng có thể đã sử dụng COPY để tải một lượng lớn dữ liệu từ các tệp văn bản phẳng. Điều này thường nhanh hơn vì lệnh COPY được tối ưu hóa cho ứng dụng này trong khi cho phép ít linh hoạt hơn INSERT. Một ví dụ sẽ là:

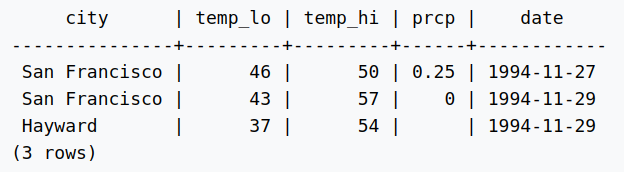
Bạn có thể đọc thêm về lệnh COPY trong [**COPY**](https://www-postgresql-org.translate.goog/docs/current/sql-copy.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp)

### 2.5. Querying a Table

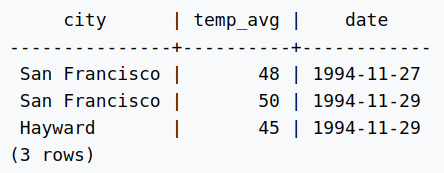
SELECT được dùng để truy vấn dữ liệu từ bảng. Câu lệnh được chia thành một danh sách chọn (phần liệt kê các cột sẽ được trả về), danh sách bảng (phần liệt kê các bảng để lấy dữ liệu từ đó).

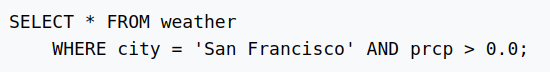
Dấu \* ở đây là viết tắt của chọn tất cả. Kết quả trả về sẽ tương tự với câu lệnh:

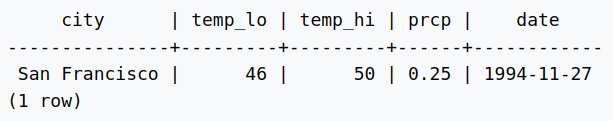
Kết quả trả về:

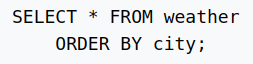
Bạn có thể viết các biểu thức, không chỉ các tham chiếu cột đơn giản, trong danh sách chọn. Mệnh đề AS dùng để dán nhãn tên cột đầu ra cho kết quả. Ví dụ, bạn có thể làm:

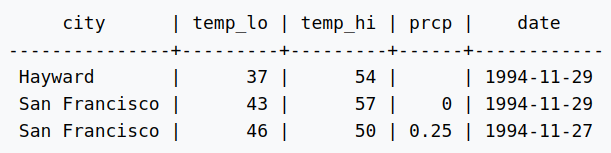
Kết quả trả về:

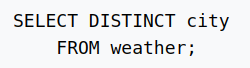
Một truy vấn có thể được “ đủ tiêu chuẩn ” bằng cách thêm một mệnh đề WHERE chỉ định những hàng nào được mong muốn. Mệnh đề WHERE chứa một biểu thức Boolean (giá trị thực) và chỉ những hàng mà biểu thức Boolean là true mới được trả về. Các toán tử Boolean thông thường ( AND, OR, và NOT) được cho phép trong tiêu chuẩn. Ví dụ: đoạn mã sau truy xuất thời tiết của San Francisco vào những ngày mưa:

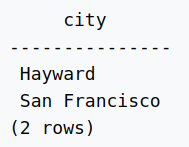
Kết quả trả về:

Bạn có thể yêu cầu trả về kết quả của truy vấn theo thứ tự đã sắp xếp:

Kết quả trả về:

Bạn có thể yêu cầu xóa các hàng trùng lặp khỏi kết quả:

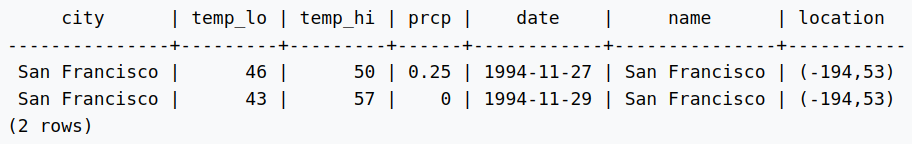
Kết quả trả về:

Bạm có thể kết hợp DISTINCT và ORDER BY:

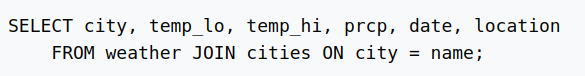
### 2.6. Joins Between Tables

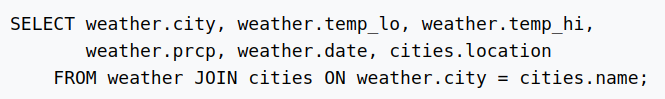
Truy vấn truy cập nhiều bảng (hoặc nhiều phiên bản của cùng một bảng) cùng một lúc được gọi là truy vấn nối . Chúng kết hợp các hàng từ một bảng với các hàng từ bảng thứ hai, với một biểu thức chỉ định những hàng nào sẽ được ghép nối. Ví dụ: để trả về tất cả các bản ghi thời tiết cùng với vị trí của **city** được liên kết, cơ sở dữ liệu cần so sánh cột **city** của mỗi hàng trong bảng **weather** với cột **name** của tất cả các hàng trong bảng **cities** và chọn các cặp hàng mà các giá trị này khớp với nhau. Điều này sẽ được thực hiện bằng truy vấn sau:

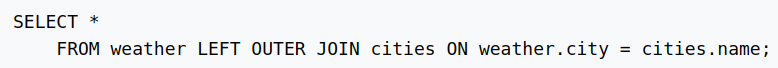
Kết quả trả về:

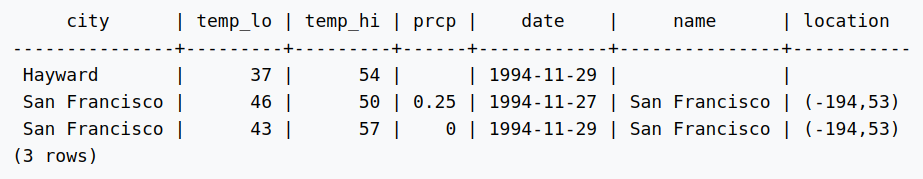
Không có hàng kết quả cho thành phố Hayward. Điều này là do không có mục phù hợp trong bảng thành phố cho Hayward, do đó phép nối sẽ bỏ qua các hàng chưa khớp trong bảng thời tiết.

Có hai cột chứa tên thành phố. Điều này đúng vì danh sách các cột từ bảng thời tiết và thành phố được nối với nhau. Tuy nhiên, trong thực tế, điều này là không mong muốn, vì vậy bạn có thể muốn liệt kê các cột đầu ra một cách rõ ràng hơn là sử dụng **\***.

Vì tất cả các cột đều có tên khác nhau nên trình phân tích cú pháp sẽ tự động tìm thấy chúng thuộc về bảng nào. Nếu có tên cột trùng lặp trong hai bảng, bạn cần xác định tên cột để hiển thị tên cột mà bạn muốn nói.

Bây giờ chúng ta sẽ tìm hiểu làm thế nào chúng ta có thể lấy lại các bản ghi Hayward. Điều chúng ta muốn truy vấn thực hiện là quét bảng **weather** và đối với mỗi hàng để tìm các hàng **cities** phù hợp. Nếu không tìm thấy hàng phù hợp, chúng tôi muốn một số "**giá trị trống**" được thay thế cho các cột của bảng **cities**.

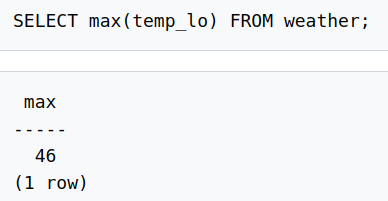
Kết quả trả về:

Truy vấn này được gọi là nối ngoài bên trái vì bảng được đề cập ở bên trái của toán tử nối sẽ có mỗi hàng của nó ở đầu ra ít nhất một lần, trong khi bảng bên phải sẽ chỉ có những hàng đầu ra khớp với một số hàng của bảng bên trái. Khi xuất ra một hàng ở bảng bên trái không khớp với hàng ở bảng bên phải, các giá trị trống (**null**) sẽ được thay thế cho các cột ở bảng bên phải.

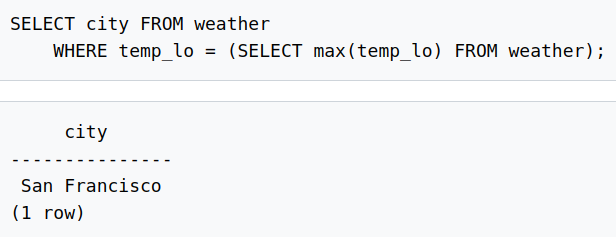
### 2.7. Aggregate Functions

Ví dụ: có các tập hợp để tính count, sum, avg(trung bình), max(tối đa) và min(tối thiểu) trên một tập hợp các hàng.

Ví dụ: chúng ta có thể tìm thấy chỉ số nhiệt độ thấp cao nhất ở bất kỳ đâu với:

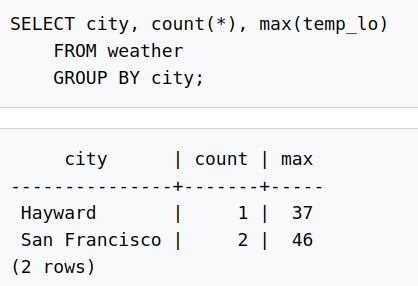


Truy vấn phụ:

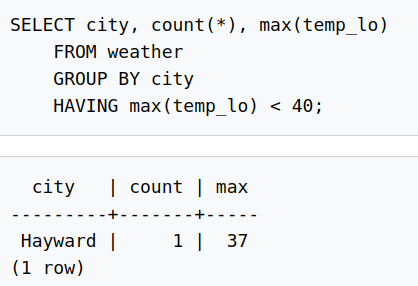


Điều này không sao vì truy vấn con là một tính toán độc lập tính toán tổng hợp của chính nó một cách riêng biệt với những gì đang xảy ra trong truy vấn bên ngoài.

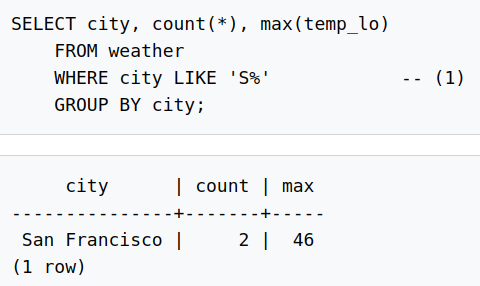
Tập hợp cũng rất hữu ích khi kết hợp với mệnh đề GROUP BY. Ví dụ: chúng ta có thể nhận được số lần đọc và nhiệt độ thấp tối đa quan sát được ở mỗi thành phố



Lọc các hàng đã được nhóm bằng **HAVING**

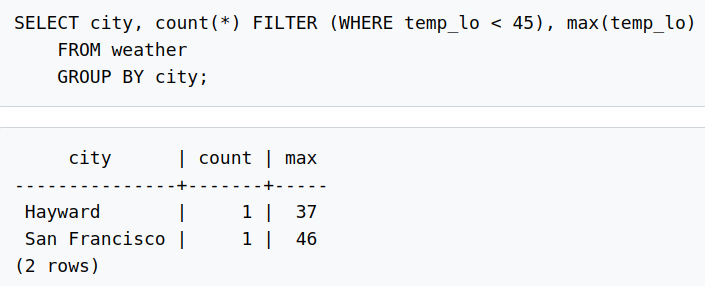


Kết hơp thêm điều kiện tên các điều kiện bắt đầu bằng chứ **S**



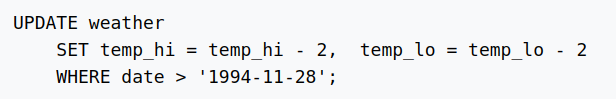
|  |
| --- |
| Điều quan trọng là phải hiểu sự tương tác giữa các tập hợp và các mệnh đề **WHERE** và **HAVING** của SQL. Sự khác biệt cơ bản giữa **WHERE** và **HAVING** là: **WHERE** chọn các hàng đầu vào trước khi các nhóm và tổng hợp được tính toán (do đó, nó kiểm soát các hàng nào sẽ được tính toán tổng hợp), trong khi **HAVING** chọn các hàng của nhóm sau khi các nhóm và tổng hợp được tính toán. Do đó, mệnh đề **WHERE** không được chứa các hàm tổng hợp; Mặt khác, mệnh đề **HAVING** luôn chứa các hàm tổng hợp. |

Một cách khác để chọn các hàng đi vào tính toán tổng hợp là sử dụng **FILTER**, đây là một tùy chọn cho mỗi tổng hợp:

**FILTER** rất giống với **WHERE**, ngoại trừ việc nó chỉ xóa các hàng khỏi đầu vào của hàm tổng hợp cụ thể mà nó được gắn vào.

### 2.8. Updates

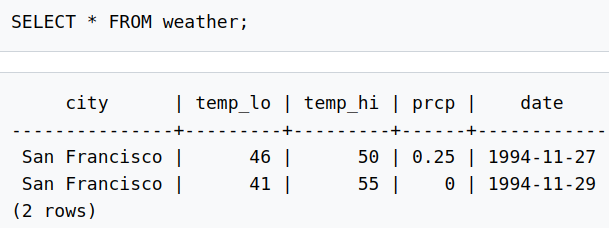
Bạn có thể cập nhật các hàng hiện có bằng lệnh UPDATE. Giả sử bạn phát hiện ra rằng tất cả các chỉ số nhiệt độ đều giảm 2 độ sau ngày 28 tháng 11. Bạn có thể sửa dữ liệu như sau:

Kết quả sau khi được update

### 2.9. Deletions

Các hàng có thể được xóa khỏi bảng bằng lệnh DELETE. Giả sử bạn không còn quan tâm đến thời tiết của Hayward. Sau đó, bạn có thể thực hiện các thao tác sau để xóa các hàng đó khỏi bảng:

Tất cả các bản ghi thời tiết thuộc về Hayward đều bị xóa.

Hãy cẩn thận với câu lệnh:

=> Câu lệnh này sẽ xóa toàn bộ dữ liệu trong table được chọn.

## Chapter 3. Advanced Features

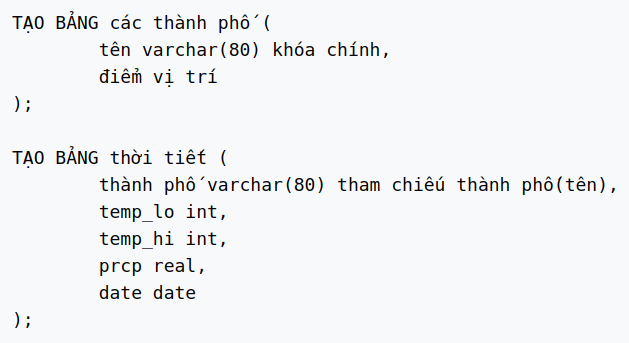
### 3.1. Views

Giả sử danh sách kết hợp các bản ghi thời tiết và vị trí thành phố được ứng dụng của bạn quan tâm đặc biệt, nhưng bạn không muốn nhập truy vấn mỗi khi cần. Bạn có thể tạo **view** trên truy vấn, cái **view** này sẽ đặt tên cho truy vấn mà bạn có thể tham chiếu giống như một bảng thông thường

### 3.2. Foreign Keys

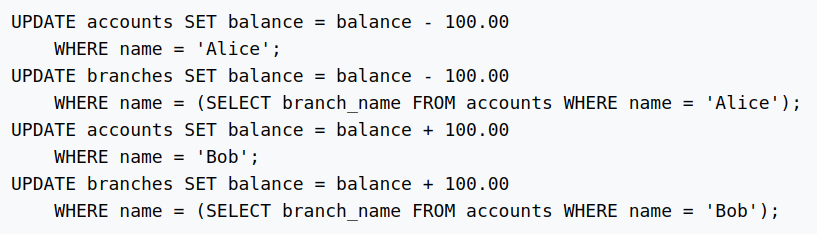
Bạn muốn đảm bảo rằng không ai có thể chèn hàng vào bảng **weather** không có mục phù hợp trong bảng cities. Điều này được gọi là duy trì tính toàn vẹn tham chiếu của dữ liệu của bạn.

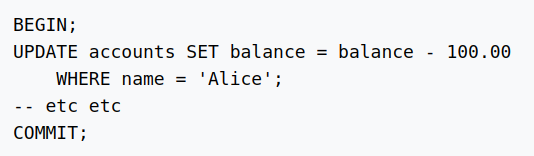
Khai báo bảng có khóa:

Bây giờ hãy thử chèn một bản ghi không hợp lệ:

### 3.3. Transactions

Điểm cốt yếu của một Transactions là nó kết hợp nhiều bước vào một hoạt động duy nhất, tất cả hoặc không có gì. Các trạng thái trung gian giữa các bước không hiển thị đối với các Transactions đồng thời khác và nếu một số lỗi xảy ra khiến Transactions không thể hoàn thành, thì không có bước nào ảnh hưởng đến cơ sở dữ liệu cả.

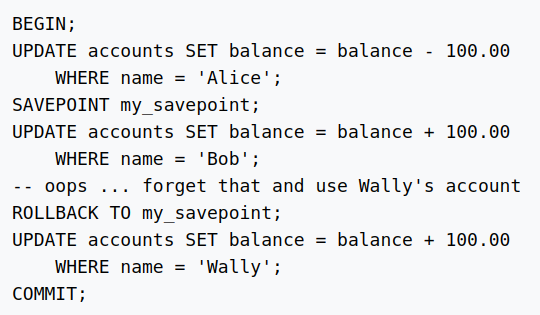
Trong PostgreSQL, một Transaction được thiết lập bằng cách bao quanh các lệnh SQL của Transaction bằng các lệnh BEGIN và COMMIT.

Nếu trong quá trình thực hiện Transactions, quyết định không muốn cam kết (có lẽ nhận thấy rằng số dư của Alice bị âm), có thể đưa ra lệnh **ROLLBACK** thay vì **COMMIT** và tất cả các cập nhật của cho đến nay sẽ bị hủy.

PostgreSQL thực sự coi mọi câu lệnh SQL như đang được thực thi trong một Transactions. Nếu bạn không đưa ra lệnh **BEGIN**, thì mỗi câu lệnh riêng lẻ có một **BEGIN** ngầm định và (nếu thành công) **COMMIT** bao quanh nó. Một nhóm các câu lệnh được bao quanh bởi **BEGIN** và **COMMIT** đôi khi được gọi là **Transaction block.**

Có thể kiểm soát các báo cáo trong một Transactions theo cách chi tiết hơn thông qua việc sử dụng **savepoints** . Điểm lưu trữ cho phép bạn loại bỏ có chọn lọc các phần của Transactions, trong khi cam kết phần còn lại. Sau khi xác định điểm lưu trữ với **SAVEPOINT**, nếu cần, bạn có thể quay lại điểm lưu trữ với **ROLLBACK** **TO**. Tất cả các thay đổi cơ sở dữ liệu của Transactions giữa việc xác định điểm lưu trữ và quay trở lại điểm lưu trữ sẽ bị loại bỏ, nhưng những thay đổi trước điểm lưu trữ được giữ lại.

Ghi nhớ cơ sở dữ liệu ngân hàng, giả sử chúng ta ghi nợ 100,00 đô la từ tài khoản của Alice và ghi có vào tài khoản của Bob, chỉ để sau đó phát hiện ra rằng lẽ ra chúng ta nên ghi có vào tài khoản của Wally. Chúng tôi có thể làm điều đó bằng cách sử dụng các điểm lưu trữ như thế này:

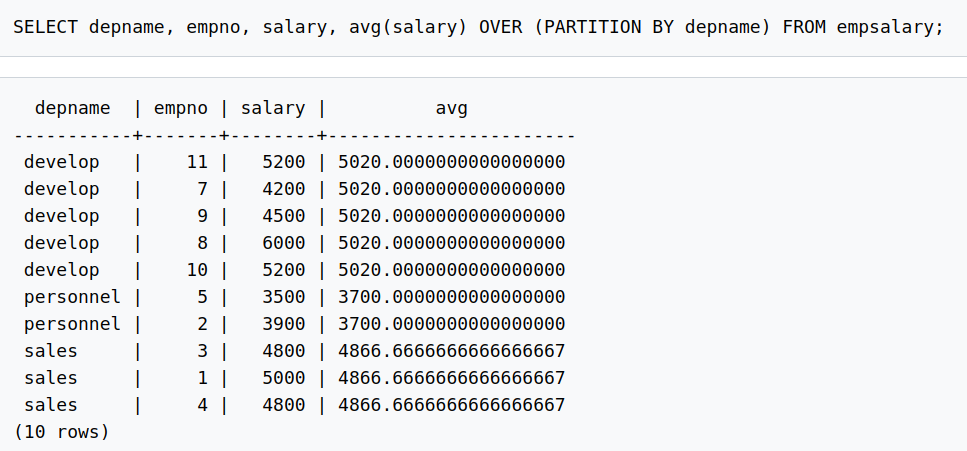
**ROLLBACK TO** là cách duy nhất để lấy lại quyền kiểm soát **Transaction block.** đã bị hệ thống đặt ở trạng thái bị hủy do lỗi, không thể khôi phục hoàn toàn và bắt đầu lại.

### 3.4. Window Functions

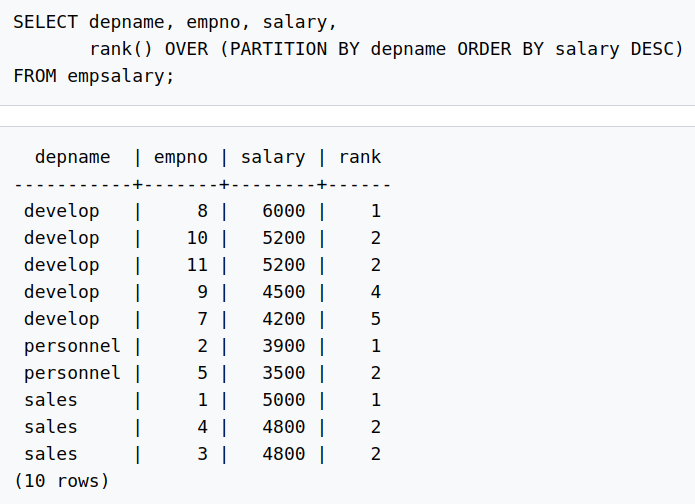
Window Functions thực hiện phép tính trên một tập hợp các hàng của bảng có liên quan nào đó đến hàng hiện tại. Điều này có thể so sánh với loại tính toán có thể được thực hiện với hàm tổng hợp (aggregate function).

Tuy nhiên, các chức năng của **Window Functions** không làm cho các hàng được nhóm lại thành một hàng đầu ra giống như các lệnh gọi tổng hợp **non-window**. Thay vào đó, các hàng giữ lại danh tính riêng biệt của họ. Chức năng cửa sổ có thể truy cập nhiều hơn chỉ hàng hiện tại của kết quả truy vấn.

Ví dụ cho thấy cách so sánh mức lương của từng nhân viên với mức lương trung bình trong bộ phận của họ:

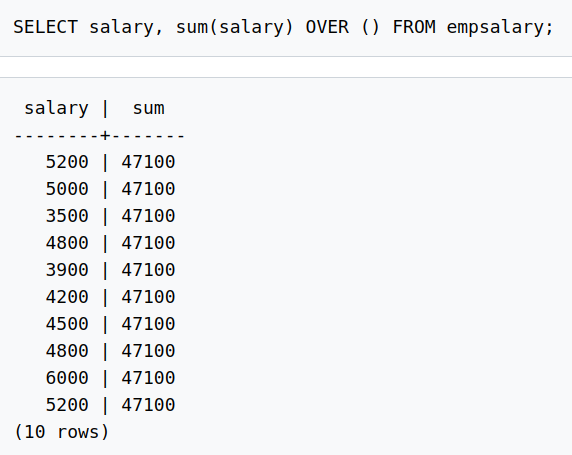
Một lệnh gọi **Window Functions** luôn chứa một mệnh đề **OVER** ngay sau tên và (các) đối số của **Window** **Functions**. Đây là điểm phân biệt về mặt cú pháp với hàm thông thường hoặc tổng hợp **non-window**. Mệnh đề **OVER** xác định chính xác cách các hàng của truy vấn được phân chia để chức năng cửa sổ xử lý. Mệnh đề **PARTITION** **BY** bên trong **OVER** chia các hàng thành các nhóm hoặc phân vùng có cùng giá trị của (các) biểu thức **PARTITION** **BY**. Đối với mỗi hàng, chức năng **window** được tính trên các hàng rơi vào cùng một phân vùng với hàng hiện tại.

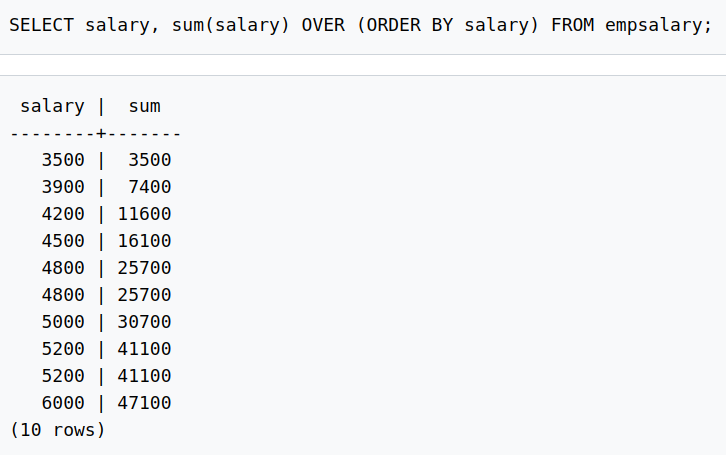
Bạn cũng có thể kiểm soát thứ tự các hàng được xử lý bởi các **Window Functions** bằng cách sử dụng **ORDER** **BY** trong **OVER**. (Cửa sổ **ORDER** **BY** thậm chí không phải khớp với thứ tự mà các hàng được xuất ra.) Đây là một ví dụ:

Như được hiển thị ở đây, **rank** **function** tạo ra một thứ hạng số cho từng giá trị **ORDER** **BY** riêng biệt trong phân vùng của hàng hiện tại, sử dụng thứ tự được xác định bởi mệnh đề **ORDER** **BY**. Thứ hạng không cần tham số rõ ràng, bởi vì hành vi của nó hoàn toàn được xác định bởi mệnh đề **OVER**.

Chúng ta đã thấy rằng **ORDER** **BY** có thể được bỏ qua nếu thứ tự các hàng không quan trọng. Cũng có thể bỏ qua **PARTITION** **BY**, trong trường hợp đó có một phân vùng duy nhất chứa tất cả các hàng.

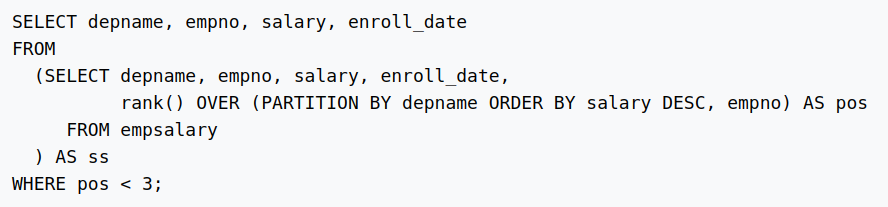
Có một khái niệm quan trọng khác liên quan đến **Window Functions**: đối với mỗi hàng, có một tập hợp các hàng trong phân vùng của nó được gọi là window frame của nó. Một số **Window Functions** chỉ hoạt động trên các hàng của **window** **frame** chứ không phải trên toàn bộ phân vùng. Theo mặc định, nếu **ORDER** **BY** được cung cấp thì **frame** bao gồm tất cả các hàng từ đầu phân vùng cho đến hàng hiện tại, cộng với mọi hàng tiếp theo bằng với hàng hiện tại theo mệnh đề **ORDER** **BY**. Khi **ORDER** **BY** bị bỏ qua, **frame** mặc định bao gồm tất cả các hàng trong phân vùng. Đây là một ví dụ sử dụng tổng:

Ở trên, vì không có **ORDER** **BY** trong mệnh đề **OVER,** nên **window** **frame** cũng giống như phân vùng, thiếu **PARTITION** **BY** là toàn bộ bảng; nói cách khác, mỗi tổng được lấy trên toàn bộ bảng và vì vậy chúng tôi nhận được kết quả tương tự cho mỗi hàng đầu ra. Nhưng nếu chúng ta thêm mệnh đề **ORDER** **BY,** chúng ta sẽ nhận được kết quả rất khác:

Ở đây, sum được lấy từ mức lương đầu tiên (thấp nhất) cho đến mức lương hiện tại, bao gồm bất kỳ bản sao nào của mức lương hiện tại (chú ý kết quả cho các mức lương trùng lặp).

**Window Functions** chỉ được phép mệnh đề **SELECT** và mệnh đề **ORDER** **BY** của truy vấn. Chúng bị cấm ở những nơi khác, chẳng hạn như trong các mệnh đề **GROUP** **BY**, **HAVING** và **WHERE**. Ngoài ra, **Window Functions** thực thi sau các hàm tổng hợp **non-window**. Điều này có nghĩa là hợp lệ khi bao gồm lệnh gọi hàm tổng hợp trong các đối số của **Window Function**, nhưng không phải ngược lại.

Nếu cần lọc hoặc nhóm các hàng sau khi thực hiện tính toán window, bạn có thể sử dụng sub-select.

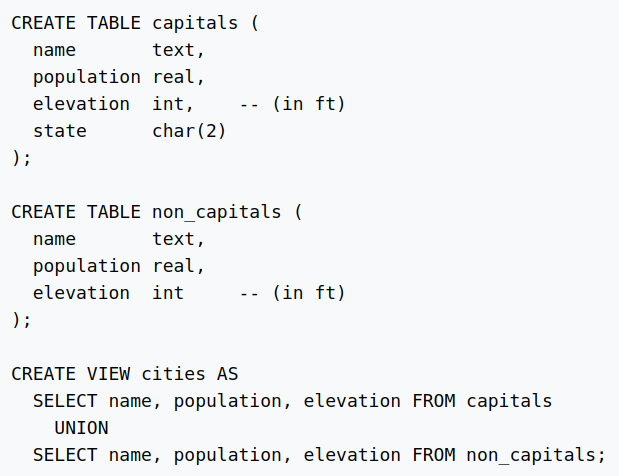
Truy vấn trên chỉ hiển thị các hàng từ truy vấn bên trong có rank ít hơn 3.

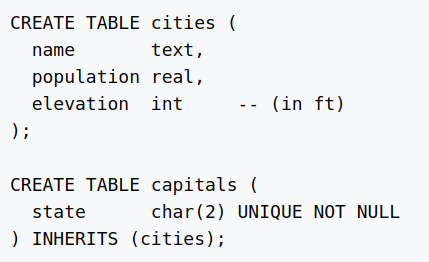
Khi một truy vấn liên quan đến nhiều **window** **functions**, có thể viết ra từng chức năng bằng một mệnh đề **OVER** riêng biệt, nhưng điều này trùng lặp và dễ bị lỗi nếu muốn có cùng một hành vi cửa sổ cho một số chức năng. Thay vào đó, mỗi hành vi cửa sổ có thể được đặt tên trong mệnh đề **WINDOW** và sau đó được tham chiếu trong **OVER**.

### 3.5. Inheritance

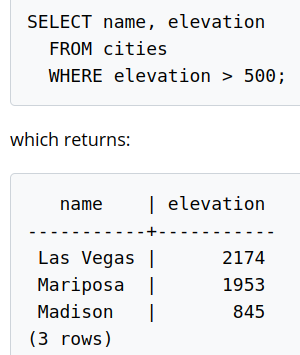
**Inheritance** là một khái niệm từ cơ sở dữ liệu hướng đối tượng. Nó mở ra những khả năng mới thú vị của thiết kế cơ sở dữ liệu.

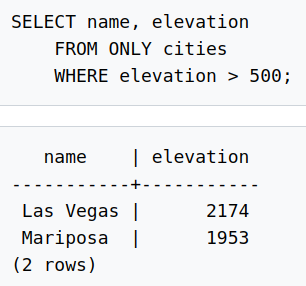
Hãy tạo hai bảng: Một bảng **cities** và một bảng **capitals**. Đương nhiên, **capitals** cũng là **cities**, vì vậy bạn muốn có một số cách để hiển thị **capitals** khi bạn liệt kê tất cả cities.

Một các khác tốt hơn:

Trong trường hợp này, một hàng **capitals*****kế thừa*** tất cả các cột (name, population, và elevation) từ **cities**. Bảng **capitals** có một cột bổ sung, **state**, hiển thị chữ viết tắt trạng thái của nó.

Ví dụ 1: Tìm tên của tất cả **cities**, bao gồm cả thủ phủ của bang (có nghĩa là **cities** đó cũng là **capitals)**, nằm ở **elevation** trên 500 feet:

Ví dụ 2: Tìm tất cả các **cities** không phải là thủ phủ của bang (có nghĩa là **cities** đó không là **capitals**) và nằm ở **elevation** trên 500 feet:



Ở đây **ONLY** trước **cities** chỉ ra rằng truy vấn chỉ được chạy trên bảng **cities** chứ không phải bảng bên dưới **cities** trong hệ thống phân cấp **inheritance**

Mặc dù **tính kế thừa** thường hữu ích, nhưng nó không được tích hợp với các ràng buộc **unique constraints** or **foreign keys**, điều này hạn chế tính hữu dụng của nó.

# Part 2. The SQL Language (Ngôn ngữ SQL)

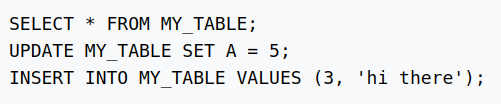
## Chapter 4. SQL Syntax (Cú pháp SQL)

### 4.1. Lexical Structure (Cấu trúc từ ngữ)

Đầu vào SQL bao gồm một chuỗi các lệnh. Một lệnh bao gồm một chuỗi các mã thông báo, được kết thúc bằng dấu chấm phẩy (“;”). Kết thúc luồng đầu vào cũng là kết thúc một lệnh. Mã thông báo nào hợp lệ tùy thuộc vào cú pháp của lệnh cụ thể.

Mã thông báo có thể là từ khóa, mã định danh, mã định danh được trích dẫn, chữ (hoặc hằng số) hoặc ký hiệu ký tự đặc biệt. Các mã thông báo thường được phân tách bằng khoảng trắng (dấu cách, tab, dòng mới), nhưng không cần thiết nếu không có sự mơ hồ (điều này thường chỉ xảy ra nếu một ký tự đặc biệt liền kề với một số loại mã thông báo khác).

Ví dụ về 1 đoạn lệnh đúng:

Một vài mã thông báo đầu tiên thường là tên lệnh, vì vậy trong ví dụ trên, chúng ta thường nói về lệnh “**SELECT**”, lệnh “**UPDATE**” và lệnh “**INSERT**”. Nhưng chẳng hạn, lệnh **UPDATE** luôn yêu cầu mã thông báo **SET** xuất hiện ở một vị trí nhất định và biến thể cụ thể này của INSERT cũng yêu cầu **VALUES** để hoàn thành

#### 4.1.1. Identifiers and Key Words (Định danh và từ khóa)

Các mã thông báo như **SELECT**, **UPDATE** hoặc **VALUES** trong ví dụ trên là các ví dụ về từ khóa, nghĩa là các từ có nghĩa cố định trong ngôn ngữ SQL. Mã thông báo **MY\_TABLE** và A là ví dụ về **identifiers**. Chúng xác định tên của bảng, cột hoặc các đối tượng cơ sở dữ liệu khác, tùy thuộc vào lệnh mà chúng được sử dụng. Do đó, đôi khi chúng được gọi đơn giản là "tên". Từ khóa và định danh có cấu trúc từ vựng giống nhau, nghĩa là người ta không thể biết token là **identifiers** hay **key** **words** nếu không biết ngôn ngữ.

Mã định danh SQL và từ khóa phải bắt đầu bằng một chữ cái (a- z, ngoài ra còn có các chữ cái có dấu phụ và chữ cái không phải chữ Latinh) hoặc dấu gạch dưới (\_). Các ký tự tiếp theo trong một mã định danh hoặc từ khóa có thể là các chữ cái, dấu gạch dưới, chữ số (0- 9) hoặc ký hiệu đô la ($). Lưu ý rằng các ký hiệu đô la không được phép trong các mã định danh theo chữ cái của tiêu chuẩn SQL, vì vậy việc sử dụng chúng có thể khiến các ứng dụng ít di động hơn.

Hệ thống sử dụng không quá **NAMEDATALEN-1** byte của mã định danh; tên dài hơn có thể được viết bằng lệnh, nhưng chúng sẽ bị cắt bớt. Theo mặc định, **NAMEDATALEN** là 64 nên độ dài định danh tối đa là 63 byte. Nếu giới hạn này có vấn đề, thì có thể tăng giới hạn này bằng cách thay đổi hằng số **NAMEDATALEN** trong **src/include/pg\_config\_manual.h.**

**Key** **words** và **identifiers** không được trích dẫn không phân biệt chữ hoa chữ thường. Ví dụ:

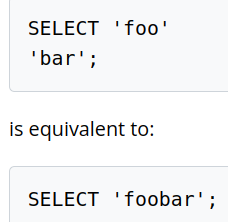
#### 4.1.2. Constants (Hằng số)

Có 3 loại hằn số trong PostgreSQL: strings, bit strings and numbers. Các hằng số cũng có thể được chỉ định với các loại rõ ràng, có thể cho phép biểu diễn chính xác hơn và hệ thống xử lý hiệu quả hơn.

***4.1.2.1. String Constants (Hằng chuỗi)***

Hằng chuỗi trong SQL là một chuỗi ký tự tùy ý được giới hạn bởi dấu nháy đơn (**'**), chẳng hạn **'**This is a string**'**.

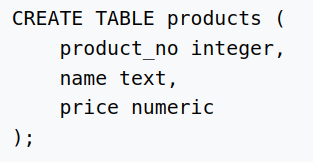
Hai hằng số chuỗi chỉ được phân tách bằng khoảng trắng với ít nhất một dòng mới được nối và xử lý hiệu quả như thể chuỗi đã được viết dưới dạng một hằng số. Ví dụ:

Tuy nhiên đoạn lệnh sau đây sẽ không hợp lệ:

## Chapter 5. Data Definition (Định nghĩa dữ liệu)

### 5.1. Table Basics (Khái niệm cơ bản về bảng)

Cách tạo một bảng:

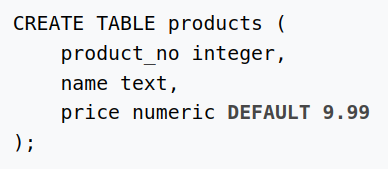
Khi bạn tạo nhiều bảng có liên quan với nhau, bạn nên chọn một mẫu đặt tên nhất quán cho các bảng và cột.

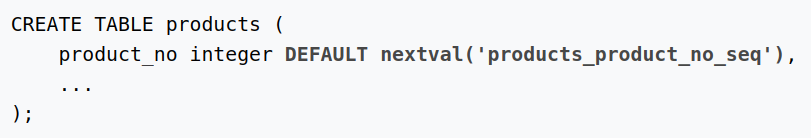
Cách xóa một bảng:

### 5.2. Default Values (Giá trị mặc định)

Một cột có thể được gán một giá trị mặc định. Khi một hàng mới được tạo và không có giá trị nào được chỉ định cho một số cột, các cột đó sẽ được lấp đầy bằng các giá trị mặc định tương ứng của chúng.

Nếu không có giá trị mặc định nào được khai báo rõ ràng, thì giá trị mặc định là giá trị null. Điều này thường có ý nghĩa vì giá trị null có thể được coi là đại diện cho dữ liệu chưa biết.

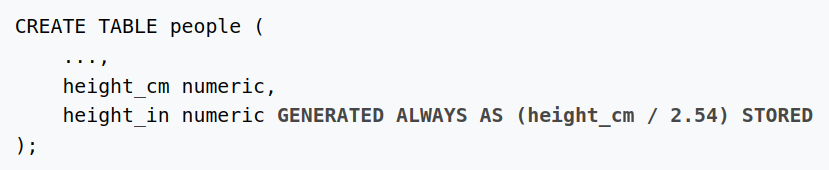
Giá trị mặc định có thể là một biểu thức, biểu thức này sẽ được đánh giá bất cứ khi nào giá trị mặc định được chèn vào (không phải khi bảng được tạo). Một ví dụ phổ biến là cột **timestamp** có giá trị mặc định là **CURRENT\_TIMESTAMP** để cột này được đặt thành thời điểm chèn hàng. Một ví dụ phổ biến khác là tạo "số sê-ri" cho mỗi hàng.

Cách viết tắt của đoạn lệnh trên:

### 5.3. Generated Columns

**Generated Columns** là một cột đặc biệt luôn được tính toán từ các cột khác. Vì vậy, đối với các cột thì dạng xem đối với các bảng là như thế nào. Có hai loại cột được tạo: **được** **lưu trữ và ảo**. Một **Generated** **Columns** được lưu trữ được tính toán khi nó được ghi (chèn hoặc cập nhật) và chiếm dung lượng lưu trữ như thể nó là một cột bình thường. **Generated** **Columns** ảo không chiếm dung lượng lưu trữ và được tính toán khi nó được đọc. Do đó, **Generated** **Columns** ảo tương tự như chế độ xem và cột được tạo được lưu trữ tương tự như chế độ xem cụ thể hóa (ngoại trừ việc nó luôn được cập nhật tự động). PostgreSQL hiện chỉ triển khai các Generated Columns được lưu trữ.

Để tạo **Generated** **Columns**, hãy sử dụng mệnh đề **GENERATED** **ALWAYS** **AS** trong **CREATE** **TABLE**

Từ khóa **STORED** phải được chỉ định để chọn loại **Generated** **Columns** được lưu trữ.

Một **Generated** **Columns**  không thể được ghi trực tiếp. Trong các lệnh **INSERT** hoặc **UPDATE**, không thể chỉ định giá trị cho **Generated** **Columns**, nhưng **DEFAULT** có thể chỉ định từ khóa.

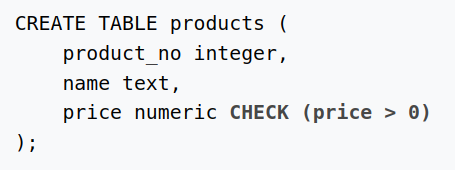
Một số hạn chế của **Generated** **Columns:**

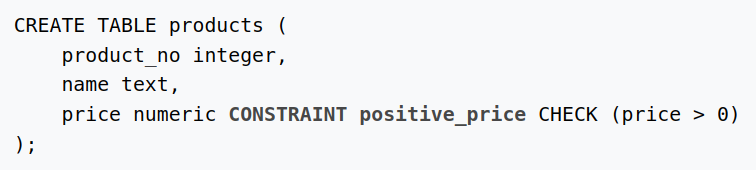
* **Generation expression** chỉ có thể sử dụng các hàm bất biến và không thể sử dụng các truy vấn con hoặc tham chiếu bất kỳ thứ gì khác ngoài hàng hiện tại theo bất kỳ cách nào.
* **Generation expression** không thể tham chiếu cột được tạo khác.
* **Generation expression** không thể tham chiếu cột hệ thống, ngoại trừ tableoid.
* **Generated** **Columns** không thể có mặc định cột hoặc định nghĩa danh tính.
* **Generated** **Columns** không thể là một phần của khóa phân vùng.
* Các **foreign table** có thể có các **Generated** **Columns**. Xem [**CREATE FOREIGN TABLE**](https://www-postgresql-org.translate.goog/docs/current/sql-createforeigntable.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) để biết chi tiết.
* Đối với thừa kế:
  + Nếu cột mẹ là **Generated** **Columns** thì cột con cũng phải là **Generated** **Columns** bằng cách sử dụng cùng một biểu thức. Trong định nghĩa của cột con, bỏ qua mệnh đề **GENERATED**, vì nó sẽ được sao chép từ cột cha.
  + Trường hợp đa thừa kế, nếu một cột cha là cột sinh thì tất cả các cột cha phải là cột sinh và có cùng biểu thức.
  + Nếu cột cha không phải là **Generated** **Columns**, thì cột con có thể được xác định là **Generated** **Columns** hoặc không.

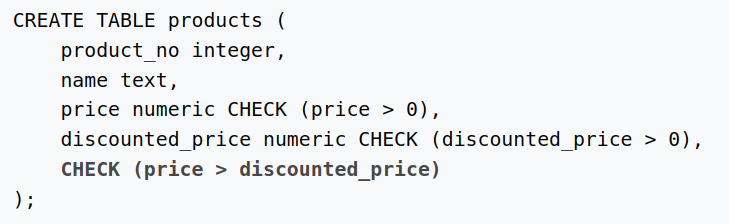
### 5.4. Constraints (Ràng buộc)

#### 5.4.1. Check Constraints (Ràng buộc kiểm tra)

Một ràng buộc **CHECK** là loại ràng buộc chung nhất. Nó cho phép bạn chỉ định rằng giá trị trong một cột nhất định phải đáp ứng biểu thức Boolean (giá trị thực). Chẳng hạn, để yêu cầu giá sản phẩm dương. Ví dụ:

Bạn cũng có thể đặt tên cho ràng buộc:

Một ràng buộc CHECK cũng có thể tham chiếu đến một số cột. Giả sử bạn lưu trữ giá thông thường và giá chiết khấu và bạn muốn đảm bảo rằng giá chiết khấu thấp hơn giá thông thường:



# Part 3. Server Administration

# Part 4. Client Interfaces

# Part 5. Server Programming

# Part 6. Reference

# Part 7. Internals

# Part 8. Appendixes