

How To Create Data Queries in PostgreSQL By Using the Select Command

Authored by: **ASPHostServer Administrator** [asphostserver@gmail.com]

Saved From: <http://faq.asphosthelpdesk.com/article.php?id=230>

What is PostgreSQL?

PostgreSQL is an open source database management system that utilized the SQL querying language. PostgreSQL, or simply "Postgres", is a very useful tool on a server because it can handle the data storage needs of websites and other applications.

In this guide, we will examine how to query a PostgreSQL database. This will allow us to instruct Postgres to return all of the data it manages that matches the criteria we are looking for.

This tutorial assumes that you have installed Postgres on your machine. We will be using Ubuntu 12.04, but any modern Linux distribution should work.

Log into PostgreSQL

We will be downloading a sample database to work with from the internet.

First, log into the default Postgres user with the following command:

```
sudo su - postgres
```

We will acquire the database file by typing:

```
wget http://pgfoundry.org/frs/download.php/527/world-1.0.tar.gz
```

Extract the gzipped archive and change to the content directory:

```
tar xzvf world-1.0.tar.gz
cd dbsamples-0.1/world
```

Create a database to import the file structure into:

```
createdb -T template0 worldddb
```

Finally, we will use the .sql file as input into the newly created database:

```
psql worldddb < world.sql
```

We are now ready to log into our newly create environment:

```
psql worldddb
```

How to Show Data in PostgreSQL

Before we begin, let's get an idea of what kind of data we just imported. To see the list of tables, we can use the following command:

```
\d+
```

List of relations					
Schema	Name	Type	Owner	Size	Description
public	city	table	postgres	264 kB	
public	country	table	postgres	48 kB	
public	countrylanguage	table	postgres	56 kB	

(3 rows)

We have three tables here. If we want to see the columns that make up the "city" table, we can issue this command:

```
\d city
```

Table "public.city"		
Column	Type	Modifiers
id	integer	not null
name	text	not null
countrycode	character(3)	not null
district	text	not null
population	integer	not null

Indexes:

"city_pkey" PRIMARY KEY, btree (id)

Referenced by:

TABLE "country" CONSTRAINT "country_capital_fkey" FOREIGN KEY (capital)
REFERENCES city(id)

We can see information about each of the columns, as well as this table's relationship with other sets of data.

How to Query Data with Select in PostgreSQL

We query (ask for) information from Postgres by using "select" statements. These statements use this general syntax:

```
SELECT columns_to_return FROM table_name;
```

For example, if we issue "\d country", we can see that the "country" table has many columns. We can create a query that lists the name of the country and the continent it is on with the following:

```
SELECT name,continent FROM country;
```

name	continent
-----+-----	

Afghanistan		Asia
Netherlands		Europe
Netherlands Antilles		North America
Albania		Europe
Algeria		Africa
American Samoa		Oceania
Andorra		Europe
. . .		

To view all of the columns in a particular table, we can use the asterisk (*) wildcard character. This means "match every possibility" and, as a result, will return every column.

```
SELECT * FROM city;
```

id		name		countrycode		district
population						

-----+-----+-----+-----						
1		Kabul		AFG		Kabol
		1780000				
2		Qandahar		AFG		Qandahar
		237500				
3		Herat		AFG		Herat
		186800				
4		Mazar-e-Sharif		AFG		Balkh
		127800				
5		Amsterdam		NLD		Noord-Holland
		731200				
6		Rotterdam		NLD		Zuid-Holland
		593321				
7		Haag		NLD		Zuid-Holland
		440900				
. . .						

Here, we see the "city" table in its entirety.

Ordering Query Results in PostgreSQL

You can organize the results of your query by using the "order by" clause. This allows you to specify a sort order to the returned data.

The "order by" clause comes after the normal select statement. This is the general syntax:

```
SELECT columns FROM table ORDER BY column_names [ ASC | DESC ];
```

If we wanted to select the country and continent from the country table, and then order by continent, we can give the following:

```
SELECT name,continent FROM country ORDER BY continent;
```

name	continent
Algeria	Africa
Western Sahara	Africa
Madagascar	Africa
Uganda	Africa
Malawi	Africa
Mali	Africa
Morocco	Africa
Côte d'Ivoire	Africa
. . .	

As you can see, by default the order statement organizes data in ascending order. This means that it starts at the beginning of lettered organizations and the lowest number for numerical searches.

If we want to reverse the sort order, we can type "desc" after the "order by" column declaration:

```
SELECT name,continent FROM country ORDER BY continent DESC;
```

name	continent
Paraguay	South America
Bolivia	South America
Brazil	South America
Falkland Islands	South America
Argentina	South America
Venezuela	South America
Guyana	South America
Chile	South America
. . .	

We can also choose to sort by more than one column. We can have a primary sort field, and then additional sort fields that are used if multiple records have the same value in the primary sort field.

For instance, we can sort by continent, and then by country to get an alphabetical list of country records in each continent:

```
SELECT name,continent FROM country ORDER BY continent,name;
```

name	continent
Angola	Africa
Burundi	Africa
Benin	Africa
Burkina Faso	Africa
Botswana	Africa
Central African Republic	Africa
Côte d'Ivoire	Africa
Cameroon	Africa
Congo, The Democratic Republic of the	Africa
. . .	

We now have alphabetical sorting in two columns.

Filtering Query Results in PostgreSQL

We have learned how to select only certain information from a table by specifying the columns we want, but Postgres provides more fine-grained filtering mechanisms.

We can filter results by including a "where" clause. A where clause is followed by a description of the results we would like to receive.

For example, if we wanted to select all of the cities in the United States, we could tell Postgres to return the name of cities where the three digit country code is "USA":

```
SELECT name FROM city WHERE countrycode = 'USA';
```

name
New York
Los Angeles
Chicago
Houston
Philadelphia
Phoenix
San Diego
Dallas
San Antonio
. . .

String values, like USA above, must be placed in single-quotations to be interpreted correctly by Postgres.

The previous query used "=" to compare whether the column value is an exact match for the value given on the right of the expression. We can search more flexibly though with the "like" comparison operator.

The like operator can use "_" as a wildcard to match a single character and "%" as a wildcard that matches zero or more characters.

We can also combine filtering terms with either "and" or "or". Let's combine some filtering to find cities in the US that have names starting with "N":

```
SELECT name FROM city WHERE countrycode = 'USA' AND name LIKE 'N%';
```

```
      name
-----
New York
Nashville-Davidson
New Orleans
Newark
Norfolk
Newport News
Naperville
New Haven
North Las Vegas
Norwalk
New Bedford
Norman
(12 rows)
```

We can, of course, sort these results just like with regular, unfiltered select data.

```
SELECT name FROM city WHERE countrycode = 'USA' AND name LIKE 'N%' ORDER BY
name;
```

```
      name
-----
Naperville
Nashville-Davidson
Newark
New Bedford
New Haven
New Orleans
Newport News
New York
Norfolk
Norman
North Las Vegas
Norwalk
(12 rows)
```

Advanced Select Operations in PostgreSQL

We are going to examine some more complex queries. Consider the following:

```
SELECT country.name AS country,city.name AS capital,continent FROM country JOIN
```

```
city ON country.capital = city.id ORDER BY continent,country;
```

continent	country	capital
-----------	---------	---------

Algeria	Alger	
Africa		
Angola	Luanda	
Africa		
Benin	Porto-Novo	
Africa		
Botswana	Gaborone	
Africa		
Burkina Faso	Ouagadougou	
Africa		
Burundi	Bujumbura	
Africa		
Cameroon	Yaoundé	
Africa		
Cape Verde	Praia	
Africa		
Central African Republic	Bangui	
Africa		
Chad	N´Djaména	
Africa		
. . .		

This query has a few different parts. Let's start at the end and work backwards.

The "order by" section of the statement (ORDER BY continent,country) should be familiar.

This section of the statement tells Postgres to sort based on continent first, and then sort the entries with matching continent values by the country column.

To explain the next portion, the table specification, we will learn about table joins.

Selecting Data From Multiple Tables in PostgreSQL with Join

Postgres allows you to select data from different, related tables using the "join" clause. Tables are related if they each have a column that can that refers to the same data.

In our example database, our "country" and "city" table share some data. We can see that the "country" table

references the "city" table by typing:

```
\d country
```

```
. . .  
. . .
```

Foreign-key constraints:

```
"country_capital_fkey" FOREIGN KEY (capital) REFERENCES city(id)  
. . .  
. . .
```

This statement tells us that the "capital" column within the "country" table is a reference to the "id" column within the "city" table. This means that we can almost treat these two tables as one giant table by matching up the values in those columns.

In our query, the table selection reads "FROM country JOIN city ON country.capital = city.id".

In this statement, we are telling Postgres to return information from both tables. The "join" statement specifies the default join, which is also called an "inner join".

An inner join will return the information that is present in both tables. For example, if we were matching columns that were not explicitly related as foreign-keys, we could run into a situation where one table had values that weren't matched in the other table. These would not be returned using a regular join.

The section after the "on" keyword specifies the columns that the tables share so that Postgres knows how the data is related. This information is given by specifying:

```
table_name.column_name
```

In our case, we are selecting records that have matching values in both tables, where the capital column of the country table should be compared to the id column of the city table.

Naming Selection Criteria For Table Joins in PostgreSQL

We are now to the beginning of our query statement. The part that selects the columns. This part should be fairly simple to decipher based on the last section.

We now have to name the table that the columns are in if we are trying to select a column name that is present in both tables.

For instance, we've selected the "name" column from both the country table and the city table. If we were to leave off the "table_name." portion of the selection, the match would be ambiguous and Postgres would not know which data to return.

We get around this problem by being explicit about which table to select from when there are naming collisions. It is sometimes helpful to name the tables regardless of whether the names are unique in order to maintain readability.

Conclusion

You should now have a basic idea of how to formulate queries. This gives you the ability to return specific

data from various sources. This is helpful for building or using applications and interactive webpages around this technology.