How To Install and Use PostgreSQL on Ubuntu 12.04

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=227*

Install Postgres

Before we install postgres, we should quick perform a quick update of the apt-get repository:

```
apt-get update
```

Once apt-get has updated go ahead and download Postgres and its helpful accompanying dependencies:

```
sudo apt-get install postgresql postgresql-contrib
```

With that, postgres is installed on your server.

Create Your PostgreSQL Roles and Databases

Once Postgres has been installed on your server, you can start to configure the database.

Postgres uses the concept of roles to distinguish the variety of users that can connect to a database. When it is first installed on a server, the default postgres user is actual named "postgresâ€•. The other users are specified in one of variety of ways. The common methods are *ident* and *md5*.

The postgres default is to use *ident* authentication, tying each server user to a Postgres account. The alternative which can be set in the authentication configuration, located in "/etc/postgresql/9.1/main/pg_hba.conf " is md5 which asks the client to supply an encrypted password. To begin creating custom users, first switch into the default user:

```
su â€“ postgres
```

Once logged in as this user, you can move forward to create more roles in your PostgreSQL system:

```
createuser

Enter name of role to add: newuser
Shall the new role be a superuser? (y/n) y
```

To outfit your user with a password, you can add the words â€“pwprompt to the createuser command:

```
createuser --pwprompt
```

Connecting to the Postgres Databases

With the users that you want to use to log into your Postgres shell set up, you can proceed to make a

database for them to use. You can create the Postgres database as a superuser. In this case, we will use the default super user. Go ahead and switch into the postgres user once again:

```
su â€“ postgres
```

As postgres, you can start to create your first usable postgres database:

```
 createdb events
```

And with that you can finally connect to the postgres shell.

How to Create and Delete a Postgres Tables

Once we log into the correct database (using the `psql -d events` command where **events** is that database's name), we can create tables within it.

Let"s imagine that we are planning a get together of friends. We can use Postgres to track the details of the event.

Let"s create a new Postgres table:

```
CREATE TABLE potluck (name VARCHAR(20),
food VARCHAR(30),
confirmed CHAR(1),
signup_date DATE);
```

This command accomplishes a number of things:

1. It has created a table called potluck within the database, newdb.
2. We have set up 4 columns in the table—name, food, confirmed, and signup date.
3. The "nameâ€• column has been limited by the VARCHAR command to be under 20 characters long.
4. The "foodâ€• column designates the food each person will bring. The VARCHAR limits text to be under 30 characters.
5. The "confirmedâ€• column records whether the person has RSVP"d with one letter, Y or N.
6. The "dateâ€• column will show when they signed up for the event. Postgres requires that dates be written as yyyy-mm-dd

Once entered, postgres should confirm the table creation with the following line:

```
CREATE TABLE
```

You can additionally see all of the tables within the database with the following command:

```
\dt
```

The result, in this case, should look like this:

```
postgres=# \dt
        List of relations
 Schema |  Name   | Type  | Owner
--------+---------+-------+-------
 public | potluck | table | root
(1 row)
```

How to Add Information to a Postgres Table

We have a working table for our party. Now it"s time to start filling in the details.

Use this format to insert information into each row:

```
INSERT INTO potluck (name, food, confirmed, signup_date) VALUES('John',
'Casserole', 'Y', '2012-04-11');
```

Once you input that in, you will see the words:

```
INSERT 0 1
```

Let"s add a couple more people to our group:

```
INSERT INTO potluck (name, food, confirmed, signup_date) VALUES('Sandy', 'Key
Lime Tarts', 'N', '2012-04-14');
INSERT INTO potluck (name, food, confirmed, signup_date)VALUES ('Tom',
'BBQ','Y', '2012-04-18');
INSERT INTO potluck (name, food, confirmed, signup_date) VALUES('Tina', 'Salad',
'Y','2012-04-18');
```

We can take a look at our table:

```
SELECT * FROM potluck;
 name  |      food      | confirmed | signup_date
-------+----------------+-----------+-------------
 John  | Casserole      | Y         | 2012-04-11
 Sandy | Key Lime Tarts | N         | 2012-04-14
 Tom   | BBQ            | Y         | 2012-04-10
 Tina  | Salad          | Y         | 2012-04-18
(4 rows)
```

Should we want to, then, follow up by removing an unlucky attendee, in this John and his casserole, from our potluck we can accomplish this with the Delete command:

```
 DELETE FROM potluck WHERE name = 'John' ;
```

How to Add and Delete a Column

PDF Generated by PHPKB Knowledge Base Script

We are creating a handy chart, but it is missing some important information: our attendees" emails. We can easily add this:

```
ALTER TABLE potluck ADD email VARCHAR(40);
```

This command puts the new column called "email" at the end of the table by default, and the VARCHAR command limits it to 40 characters. Just as you can add a column, you can delete one as well:

```
ALTER TABLE potluck DROP email;
```

I guess we will never know how to reach the picnickers.

How to Update Information in the Table

Now that we have started our potluck list, we can address any possible changes. For example: Sandy has confirmed that she is attending, so we are going to update that in the table.

```
UPDATE potluck set confirmed = 'Y' WHERE name = 'Sandy';
```

You can also use this command to add information into specific cells, even if they are empty.

PDF Generated by PHPKB Knowledge Base Script