How To Create, Remove, & Manage Tables in PostgreSQL on a Cloud Server

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=229*

What is PostgreSQL?

PostgreSQL is a database management system that uses the SQL querying language. It is a very stable and feature-rich database system that can be used to store the data from other applications on your server.

In this article, we will discuss how to create and manage tables within the postgreSQL interface. You will learn how to properly configure tables and use them to store your information.

How to Install and Log Into PostgreSQL on Ubuntu

In this guide, we will install PostgreSQL on Ubuntu 12.04, but it should be available in most other distributions' default repositories.

Type the following commands to install:

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

After installation, create a new user to manage the database we'll be creating:

```
sudo adduser postgres_user
```

Log into the default PostgreSQL user (called "postgres") to create a database and assign it to the new user:

```
sudo su - postgres
psql
```

You will be dropped into the PostgreSQL command prompt.

Create a new user that matches the system user you created. Then create a database managed by that user:

```
CREATE USER postgres_user WITH PASSWORD 'password';
CREATE DATABASE my_postgres_db OWNER postgres_user;
```

Exit out of the interface with the following command:

```
\q
```

Exit out of the default "postgres" user account and log into the user you created with the following commands:

```
exit
sudo su - postgres_user
```

Sign into the database you created with the following command:

```
psql my_postgres_db
```

We are now ready to learn about table management.

Table Creation Syntax in PostgreSQL

Our database does not have any tables yet. We can verify this by asking PostgreSQL to give us a listing of the available tables with this command:

```
\d

No relations found.
```

We can create a new table by adhering to the following syntax:

```
CREATE TABLE new_table_name (
 table_column_title TYPE_OF_DATA column_constraints,
 next_column_title TYPE_OF_DATA column_constraints,
 table_constraint
 table_constraint
) INHERITS existing_table_to_inherit_from;
```

The part after the closing parenthesis up until the semi-colon is an optional construction to inherit all columns from an existing table in addition to the columns listed in the earlier definition.

The part inside of the parentheses is divided into two parts: column definitions and table constraints.

PostgreSQL Column and Table Definitions

Column definitions follow this syntax pattern:

```
column_name data_type (optional_data_length_restriction) column_constraints
```

The column name should be self-explanatory.

PostgreSQL Data Types

The data type can be any of the following:

boolean: Use "boolean" or "bool" to declare a true or false value.
character values **char**: holds a single character **char (#)**: holds # number of characters. Spaces will be inserted to fill any extra room. **varchar (#)**: holds a maximum of # number of character. Can contain less.
integer values **smallint**: whole number between -32768 and 32767. **int**: whole number between -214783648 and 214783647. **serial**: Auto-populated integer number.
floating-point values **float (#)**: floating point number with at least # points of precision. **real**: 8-byte floating point number **numeric (#,after_dec)**: real number with # number of digits, and after_dec digits after

decimal
date and time values **date**: stores a date value **time**: stores a time value **timestamp**: stores a date and time value **timestamptz**: stores a timestamp that includes timezone data **interval**: stores the difference between two timestamp values
geometric data **point**: stores a pair of coordinates that define a point **line**: stores a set of points that map out a line **lseg**: stores data that defines a line segment **box**: stores data that defines a rectangle **polygon**: stores data that defines any enclosed space
device specifications **inet**: stores an IP address **macaddr**: stores a device MAC address

PostreSQL Column and Table Constraints

Column definitions can also have constraints that provide rules for the type of data found in the column. The following can be used as space-separated values following the data type:

NOT NULL: column cannot have null value **UNIQUE**: column value must not be the same for any record. Null is always considered a unique value **PRIMARY KEY**: combination of the above two constraints. Can only be used once per table **CHECK**: ensure that a condition is true for values in the column **REFERENCES**: value must exist in a column in another table
After the columns are defined, table-wide constraints may be declared. Table-wide constraints can be either UNIQUE, PRIMARY KEY, CHECK, or REFERENCES.

How to Create a Table in PostgreSQL

Let's create a test table to practice on. We will create a table called "pg_equipment" that defines various pieces of playground equipment. Type the following table definition:

```
CREATE TABLE pg_equipment (
 equip_id serial PRIMARY KEY,
 type varchar (50) NOT NULL,
 color varchar (25) NOT NULL,
 location varchar(25) check (location in ('north', 'south', 'west', 'east',
'northeast', 'southeast', 'southwest', 'northwest')),
 install_date date
 );

NOTICE:  CREATE TABLE will create implicit sequence "pg_equipment_equip_id_seq"
for serial column "pg_equipment.equip_id"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index
"pg_equipment_pkey" for table "pg_equipment"
CREATE TABLE
```

We can see our new table by typing "\d" into the prompt:

```
\d

                      List of relations
 Schema |           Name             |   Type   |     Owner
--------+----------------------------+----------+---------------
 public | pg_equipment               | table    | postgres_user
 public | pg_equipment_equip_id_seq | sequence | postgres_user
(2 rows)
```

The table is listed, as well as the sequence created by the "equip_id" serial data type declaration.

How to Change Table Data in PostgreSQL

We can change the definition of our tables with the following general syntax:

```
ALTER TABLE table_name  Action_TO_Take;
```

For example, we can add a column to our "pg_equipment" table by entering this command:

```
ALTER TABLE pg_equipment ADD COLUMN functioning bool;

ALTER TABLE
```

We can see the extra column by typing:

```
\d pg_equipment

    Column    |         Type          |                            Modifiers
```

```
--------------+-----------------------+-----------------------------------------------
 equip_id     | integer               | not null default
nextval('pg_equipment_equip_id_seq'::regclass)
 type         | character varying(50) | not null
 color        | character varying(25) | not null
 location     | character varying(25) |
 install_date | date                  |
```

```
 functioning  | boolean               |
 . . .
```

To add a default value that specifies that "equipment should be considered working unless otherwise noted", give the following command:

```
ALTER TABLE pg_equipment ALTER COLUMN functioning SET DEFAULT 'true';
```

If we want to ensure that the value is also not null, we can do this:

```
ALTER TABLE pg_equipment ALTER COLUMN functioning SET NOT NULL;
```

To rename the column, use the following syntax:

```
ALTER TABLE pg_equipment RENAME COLUMN functioning TO working_order;
```

To remove the column we just created, enter this command:

```
ALTER TABLE pg_equipment DROP COLUMN working_order;
```

We can rename the entire table with this command:

```
ALTER TABLE pg_equipment RENAME TO playground_equip;
```

Deleting Tables in PostgreSQL

We can delete the table we created by typing:

```
DROP TABLE playground_equip;

DROP TABLE
```

If we give that command to a table that does not exist, we will receive the following error:

```
ERROR: table "playground_equip" does not exist
```

To avoid this error, we can tell postgreSQL to delete the table if it exists and return successfully either way. We do this by issuing the following command:

```
DROP TABLE IF EXISTS playground_equip;

NOTICE:  table "playground_equip" does not exist, skipping
DROP TABLE
```

This time, it tells us that the table was not found, but continues instead of throwing an error.

Conclusion

You should now know enough to create and manage simple tables in PostgreSQL. These skills will be helpful if you are managing data from another application, or learning the how to control PostgreSQL from the command line.