# Week 07

Prepare For Exam: Create a PostgreSQL Server

Melvyn Ian Drag

October 17, 2019

**Abstract**

These notes describe the procedure for configuring a PostgreSQL Database Server on a Debian 10 Server. We will learn what a DB is, what are some basic things it can do, and how to make it accessible over the internet.

## 1 Warning!

As usual with computer programs, all of the commands in this document must be executed precisely as they are found herein. If an when you come across an error, please let me know so I can update the notes. All of the information in these notes was condensed from the following references:

1. `https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-`
2. `http://www.postgresqltutorial.com/postgresql-inner-join/`
3. `https://dba.stackexchange.com/questions/190674/can-a-database-table-not-have-a-prim`
4. `https://www.postgresql.org/message-id/001f01c018c2$830133b0$64898cd5@northlink.gr`
5. `https://support.plesk.com/hc/en-us/articles/115003321434-How-to-enable-remote-acces`

## 2 What's a database?

Need to fill this in. For now, just give a rough explanation. Show the inner join table and doodle on the board what we expect.

By the way, Postgres is a Relational Database. You may have heard word like "Microsoft Access", "Oracle", "PostgreSQL", "SQLite3", "mySQL" and more. These are all relational databases. They all have slightly different implementations, and the way you interact with them is *slightly* different, but they all accomplish the same task. They store tabular data and allow you to manipulate the tables. The language you use to interact with these databases is called "SQL", though, as I said, the implementations of SQL in these different

databases is *slightly* different in each implementation. So the stuff I'm going to show you today is conceptually portable between these different DBs, but you'll need to google the syntax. If you've used mysql before you may notice that the commands we type thorughout this lecture are a teensy bit different from mysql, but approximately the same.

### 2.1 This is a Linux Class not a SQL class, I hate this lecture

Linux is useful on laptops as you can see - we are all doing our regular computer stuff on linux laptops. Linux is also hugely popular as a server OS. Many people run windows on their personal laptops, for example, but still use Linux to run their websites, source control (e.g. git) servers, proxy servers, and - you guessed it - database servers. SQL databases ( aka relational databases ) are used in business, on website backends, and probably other places too. In your linux career you will likely be interacting with a database in some make, shape or form, so listen up! This is useful info, although you might not realize this for another two years. Thank me later.

To be more explicit when you sign up on a website for example it asks for you first name, last name, password, etc.. When you click 'submit' on the webpage, that info goes to some website code and then flows into a database. When you login again, somehow the website knows your name - it gets that info out of a database.

## 3 What is our goal today?

We want to be able to do the inner join operation we just sketched out remotely, from our computer, on a database that lives on a different linux computer.

## 4 Install PostgreSQL

### 4.1 Intro

Follow instuctions here precisely: `https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-18-04`

Then work through the sample exercises.

### 4.2 Create a new droplet

Create a new debian 10 digital ocean droplet. Make sure you add your ssh key to the droplet as we go along.

### 4.3 Install postgress

```
user@laptop$ ssh root@ipaddr
root@ipaddr$ apt update
# stuff happens
root@ipaddr$ apt install postgresql postgresql-contrib
```

```
# stuff happens. Say yes when prompted.
root@ipaddr$ sudo -i -u postgres
postgres@ipaddr$ createuser --interactive
# add user 'test'
# make user a superuser
postgres@ipaddr$ createdb test # make this the same name as
   the username you created
postgres@ipaddr$ exit
root@ipaddr$ adduser test # same name as db
root@ipaddr$ sudo -i -u  test
test@ipaddr$ psql # this will open a different looking ocmmand
   prompt. That is the postgres prompt
test=#
test=# \conninfo
# youll see some info about your database connection
test=# \password
# enter a new password. Note this somewhere.
# exit by typing \q
# reenter by typing psql as before.
```

Your database is installed an functioning! Now we can move on.

# 5 Perform an inner join

## 5.1 Create Table store

Create the table like this:

```
CREATE TABLE store (
        store_id   int PRIMARY KEY,
        price      int NOT NULL,
        about_item varchar (25) NOT NULL
);
```

Add some stuff to the table like this:

```
INSERT INTO store ( store_id, price, about_item ) VALUES ( 1,
   100, 'Fancy dress shirt' );
INSERT INTO store ( store_id, price, about_item ) VALUES ( 2,
   40, 'reasonable shirt');
INSERT INTO store ( store_id, price, about_item ) VALUES ( 3,
   40, 'branded tshirt');
```

Note that if you try to add an element with a primary key that is already there it will yell at you:

```
INSERT INTO store ( store_id, price, about_item ) VALUES ( 3,
   20, 'another shirt');
```

Server says:

```
Hello
```

That's because the PRIMARY KEY column - in this case store_id - should uniquely identify a product. Verify the stuff in your table with this command:

```
SELECT * FROM store;
```

## 5.2 Create Table warehouse

```
CREATE TABLE warehouse(
        uniq_id      int      PRIMARY KEY,
        quantity     smallint NOT NULL,
        warehouse_id int NOT NULL,
        FOREIGN KEY (warehouse_id) REFERENCES store (store_id)
);
```

And then we'll put some stuff in the table

```
INSERT INTO warehouse ( uniq_id, quantity, warehouse_id )
   VALUES ( 1, 10, 1 );
INSERT INTO warehouse ( uniq_id, quantity, warehouse_id )
   VALUES ( 2, 30, 2 );
SELECT * FROM warehouse;
```

## 5.3 Do the thing

```
SELECT
        store.store_id,
        warehouse.warehouse_id,
        store.price,
        store.about_item,
        warehouse.quantity
FROM
        store
INNER JOIN
        warehouse
ON
        store.store_id = warehouse.warehouse_id;
```

Output should look like this:

```
 store_id | warehouse_id | price |    about_item     | quantity
----------+--------------+-------+-------------------+----------
        1 |            1 |   100 | Fancy dress shirt |       10
        2 |            2 |    40 | reasonable shirt  |       30
        3 |            3 |    40 | branded tshirt    |       15
```

# 6 Configure server to be accessible on the internet

At this point, we have a database on our computer with some interesting tables in it. We've been able to add and manipulate data in the database. Now we just want to configure our server such that we can access the database remotely from the terminal on another computer.

Right now you cannot access this database. By default, postgreSQL only makes this database visible from the machine running the database server. Now we are going to expose it to the internet. You need to modify two files:

REad this stuff in case there is a mistake in lecture notes

1. `https://bosnadev.com/2015/12/15/allow-remote-connections-postgresql-database-server`
2. `https://support.plesk.com/hc/en-us/articles/115003321434-How-to-enable-remote-acces`

## 6.1 postgresql.conf

Find this file on your computer by typing:

```
find / -name postgresql.conf
```

For me the file is here. Yours should be in the same spot.

```
/etc/postgresql/11/main/postgresql.conf
```

Find the line in the file that says 'listen_addresses' and change it such that it says:

```
listen_addresses = '*'
```

Note that postgres runs on port 5432.

## 6.2 pg_hba.conf

For me the file is here:

```
/etc/postgresql/11/main/pg_hba.conf
```

Since we're all on a digital ocean debian 10 server, your location should be the same, but maybe not.

Add the following line to the bottom of that file:

```
host    all     all  0.0.0.0/0    md5
```

### 6.3 Restart postgres

Run

```
root@machine$ service postgresql restart
```

### 6.4 Connect remotely

Back on your laptop install a postgres client that can talk to the database. On ubuntu you install this with:

```
user@laptop$ sudo apt install postgresql-client-common postgresql-client
```

Gather your server ip address, database user name, database user password, and run the following command:

```
psql -h 67.205.173.100 -p 5432 -U melvyn
```

But supply your server's ip address and postgres username.
Enter your password when prompted.
Then run this command at the prompt:

```
user# \dt
```

You should see the tables you created. You can now rerun your inner join command, select commands, whatever database things you want to do, remotely.

## 7 A bit More Info

There is so much to know about databases. When we created tables I made the column types int, varchar, foreign key, primary key, small int. You can read more about this here: `http://www.postgresqltutorial.com/postgresql-data-types/`
Let's create tables with some other datatypes in it. ( 10 minute exercise or so )

## 8 What's a Primary Key and Do I need One?

`https://dba.stackexchange.com/questions/190674/can-a-database-table-not-have-a-primary-k`

## 9 Questions

As I've said before, I'm sort of in the dark about how these things work. A few things I need to know but don't are :

- in setting up the server we did 'sudo -i -u $username'. Need to look at that more carefully. It passes authentication later by routing you through some shells.

- Why do we need a linux user the same name as the postgres user the same name as the database? Just default postgres settings. Should play with that and see what breaks.
- When you createuser –interactive, what is the default password? Does it ask for a password? I can't remember.

# 10 Your exam

Next week you will come in, reset your droplet, and then configure it as we did. Then you will come to my laptop and show me you can execute an inner join on the server you created.

# 11 A few bash commands

## 11.1 curl

We'll talk more about curl at a later date. It's a powerful tool you use for sending commands over different network protocols to different servers. Generally you use it for web requests. More on that later. I've been using it to check your ip address. In the browser you can go to this website:

`ipinfo.io/ip`

And it will tell you your ip address. I use curl to get that info on the command line, when a browser isn't available.

```
user@machine$ curl ipinfo.io/ip
a.bb.c.ddd
```

## 11.2 tmux

Now you are ready to appreciate the power of tmux! When you are ssh-ed into another machine, you can't open another tab, or popup another window - all you have is one terminal connection to that machine. That's when tmux ( and screen ) become useful. You can split the screen into windows and use them to do stuff simultaneously. I showed you these tmux tricks:

CTRL B +:

- % = vertical split
- " = horizontal split
- ( arrow keys ) = move around splits
- x = kill current pane ( note in the bottom of the screen it asks if you want to close the pane )

Here's another cool thing you can detach a shell and come back later.

1. start tmux
2. start a runForever program.
3. detach tmux
4. disconnect from ssh
5. log back in with ssh
6. reattach tmux
7. process still going!

## 11.3 ssh

Guard your id_rsa with your life! The .pub can be distributed, but if you leak you r id_rsa youre in deep doo doo. People can hack you if they have access to that file.

alot to be said - and then again, very little. You are now using ssh to connect to a remote machine. And you saw that if you put an id_rsa.pub file on that machine you can login without even giving a password! Pretty cool stuff. How it works is elegant, we'll get to it later. It involves cryptography. ssh will be a tool you use every day if you keep working with linux.

You can use putty on windows. If time, just try to ssh with putty using a linux id_rsa key. This will take some time.