Querying Relational Data with Postgres

By this end of this activity, you will be able to:

1. View table and column definitions, and perform SQL queries using PgAdmin
2. Filter table rows and columns
3. Combine two tables by joining on a column
4. Visualize the results of our queries

Step 1. Open a terminal shell. Open your local terminal shell and go to your big-data-3/postgresSQL directory

```
PS C:\Users\             \Desktop\coursera\big-data-3\postgresSQL>
```

Step 2. Start Docker. Make sure to start Docker by opening Docker Desktop.

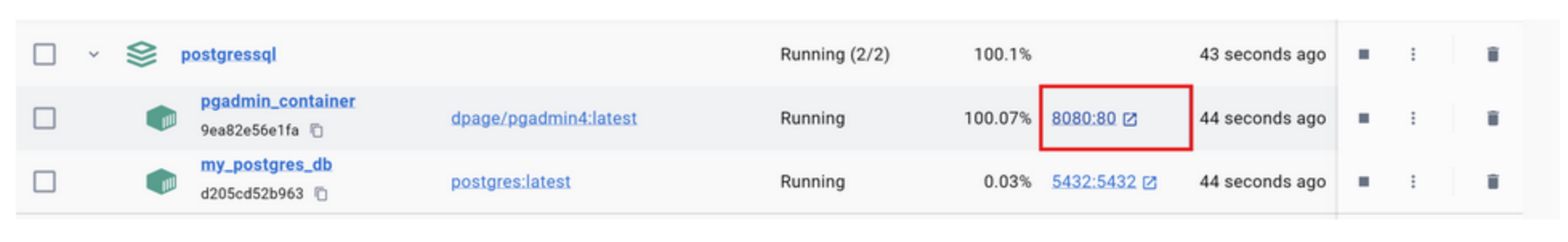
Step 3. Starting the containers. We need to start multiple containers for this activity. One container will host out postgresSQL server, while the other will host PgAdmin, which is essentially an interface to interact with our postgresSQL server.

Run *docker compose up -d*

```
docker compose up -d
```

Step 4. Verify your containers are running. Go to Docker Desktop and make sure your containers are running.

Once you make sure they are running, click con the 8080 port to access PgAdmin.

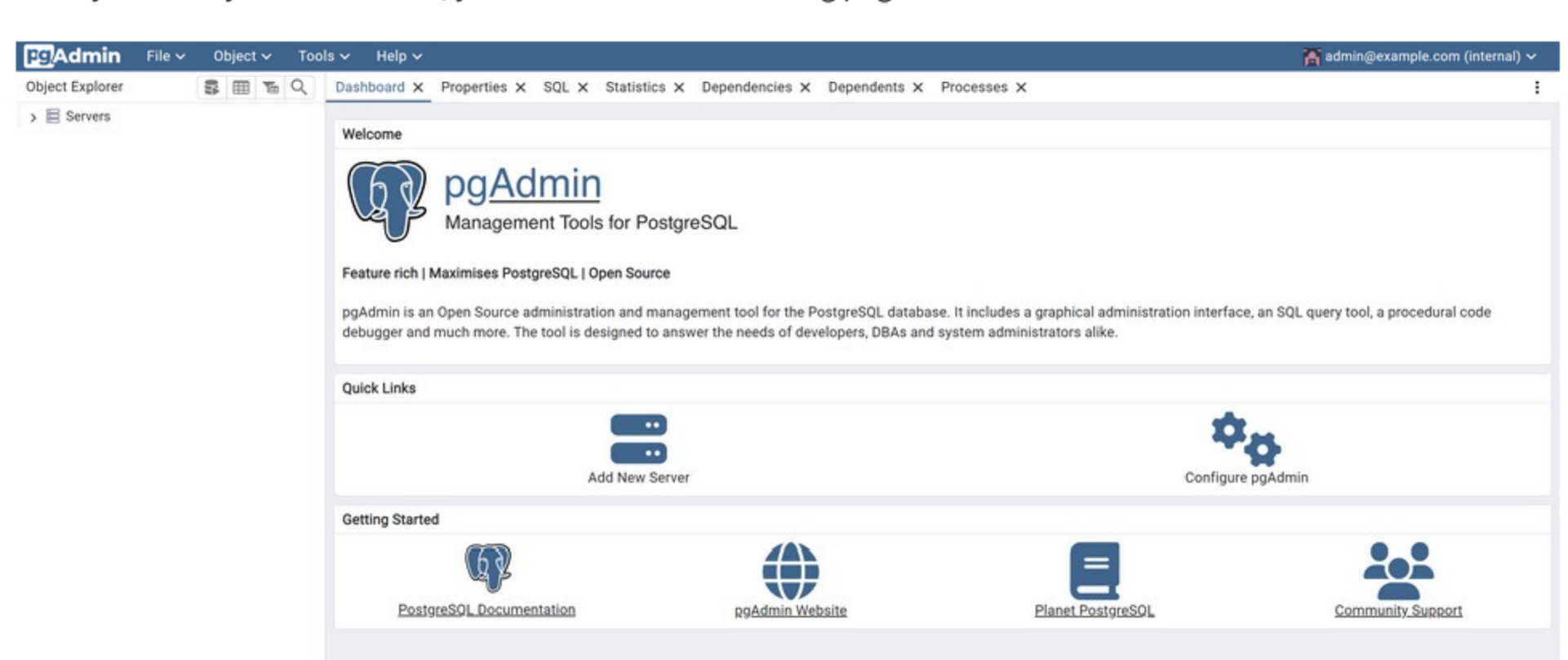


NOTE: It is possible that if you click quickly after starting the container, you get a *This site can't be reached* message from your browser. If that happens, wait for a couple of minutes and try again.

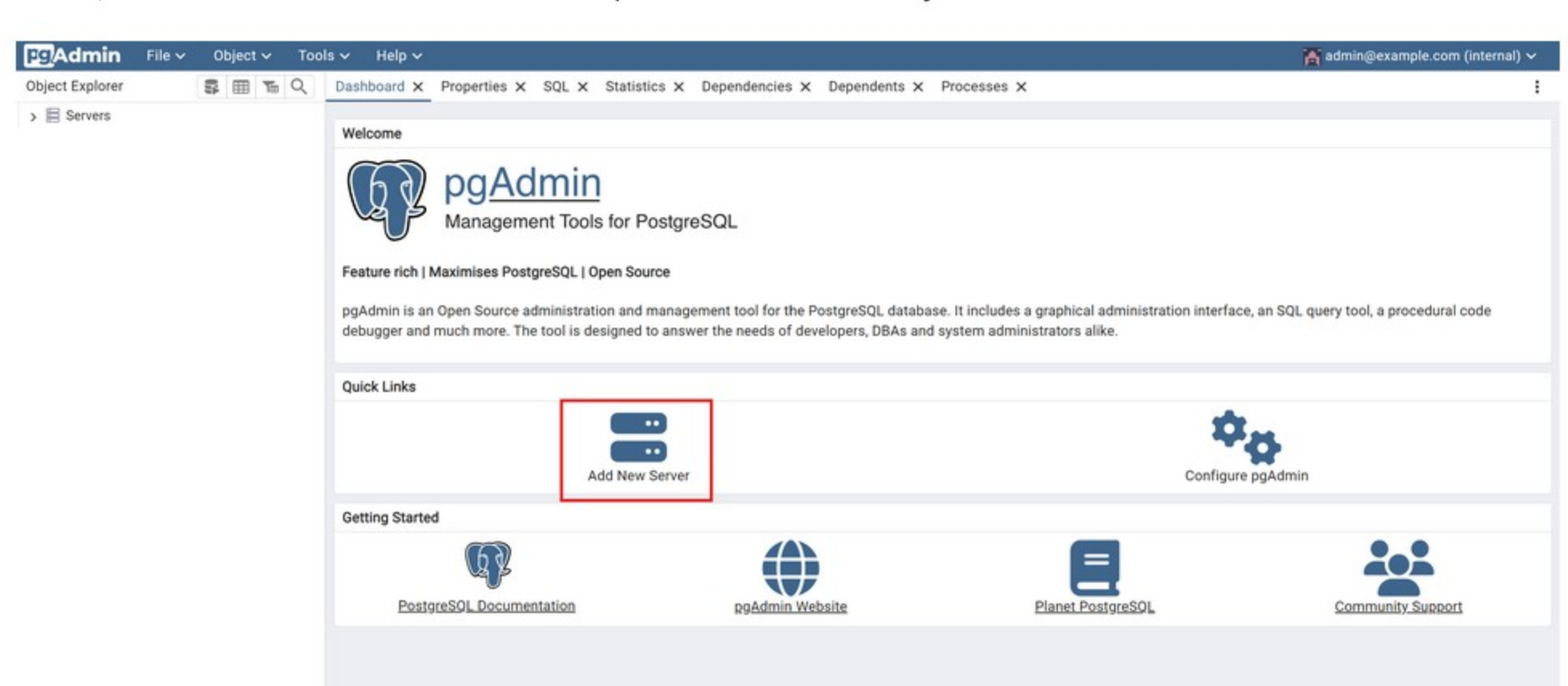
Once you are redirected to your localhost:8080, you should see the following page.

Step 5. Access PgAdmin. Enter admin@example.com as your Email Address/ Username and *adminpassword* as the password.

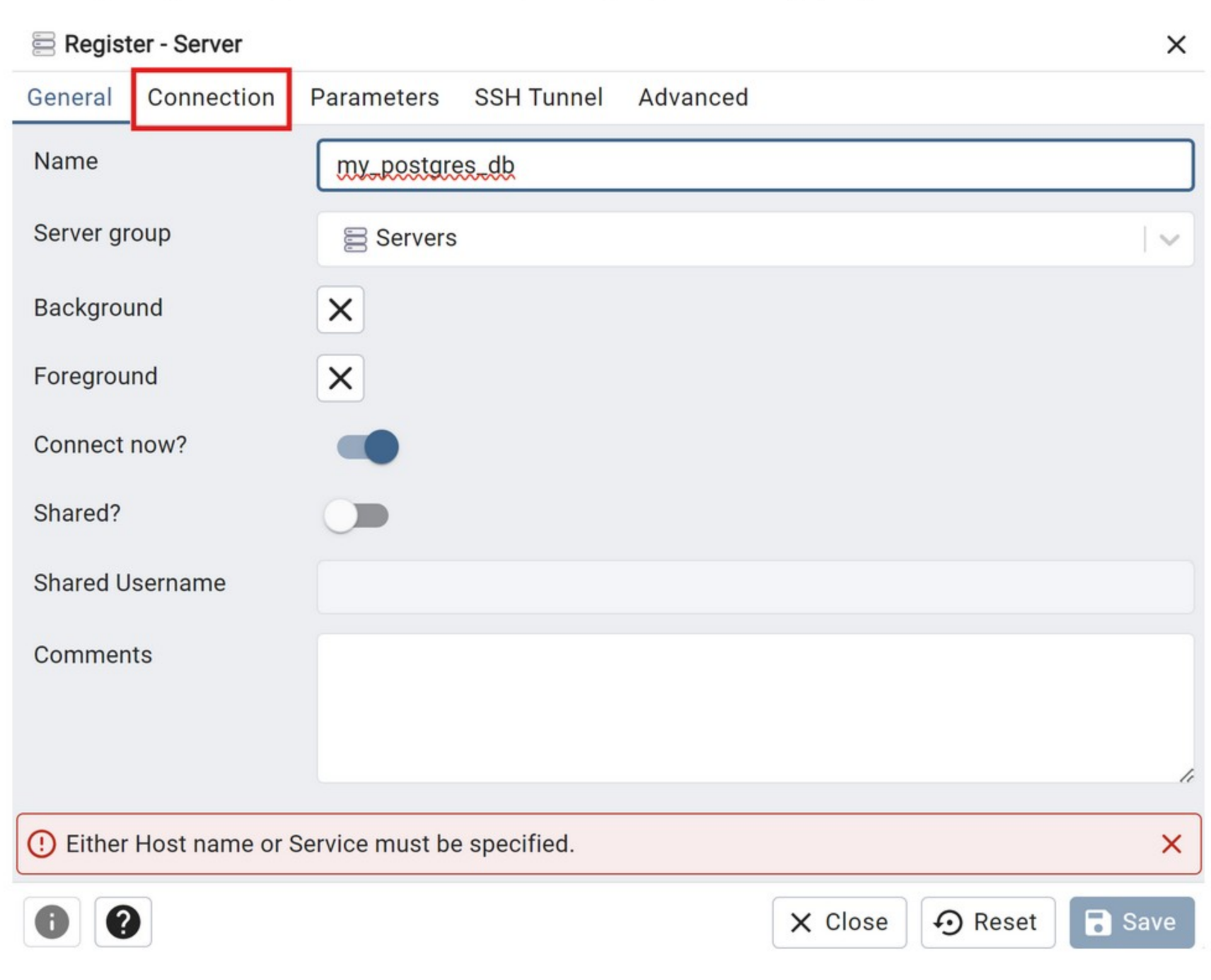
Once you enter your credentials, you should see the following page.



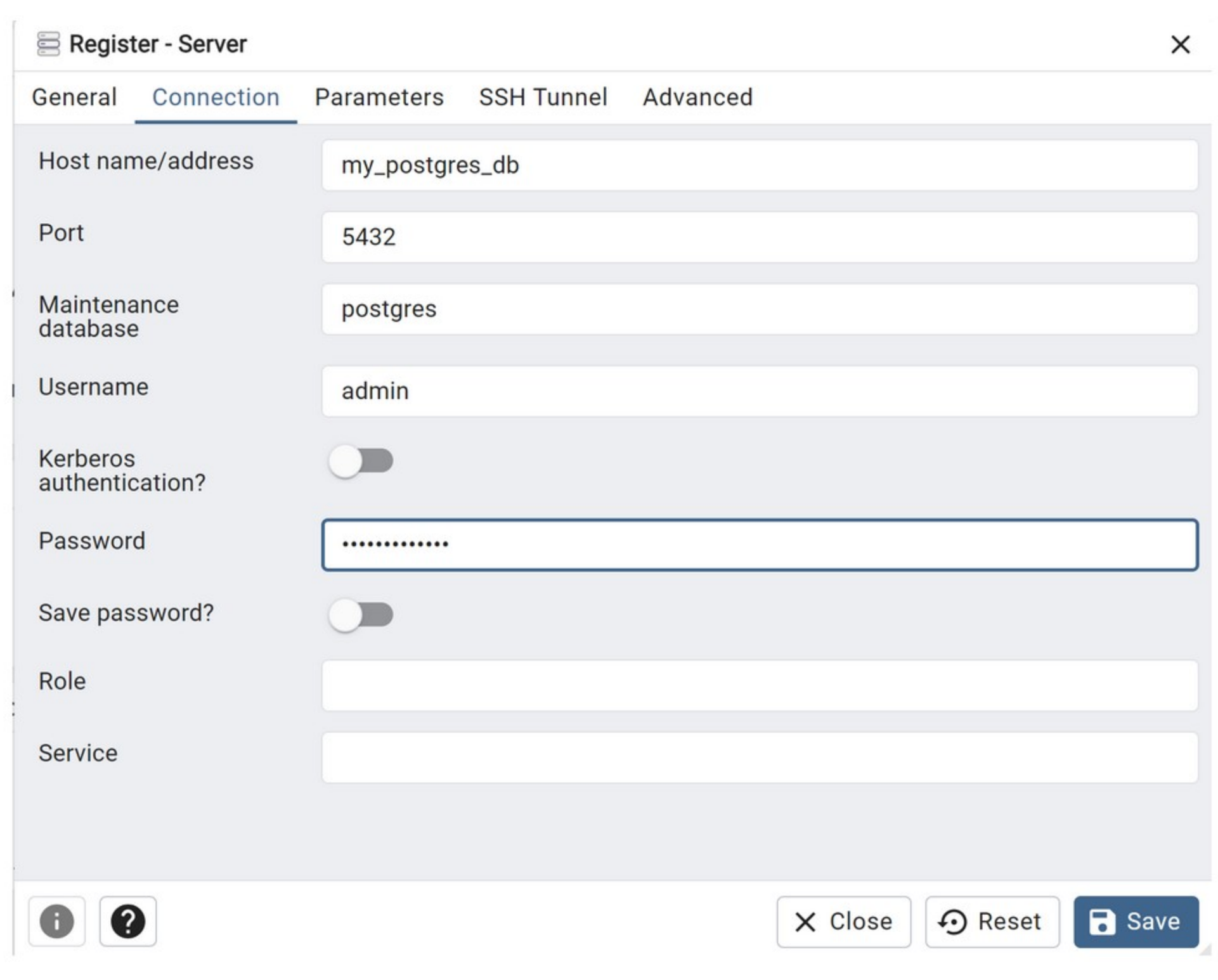
Step 6. Establish a connection with the postgresSQL server. Now we are going to connect to our postgresSQL server, which contains the data that we have provided for this activity. Click on *Add New Server*



You can give any name to your server. In this case, we are going to name it *my_postgres_db*

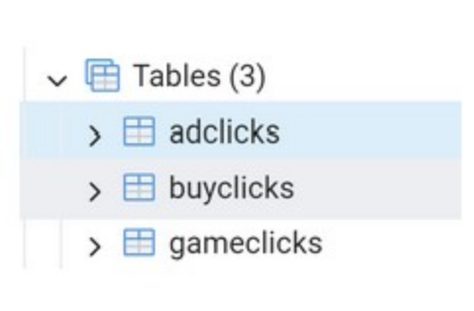


In *Hostname/Address* type *my_postgres_db*. Make sure *Port* is pointed to *5432*. In *Password* type *adminpassword*. Click *Save*.

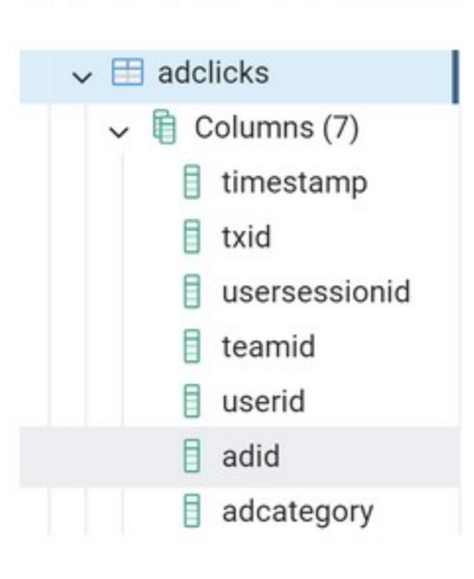


Step 7. Exploring the DB and the schema. Click on *my_postgres_db*, then on *databases* and finally on *sampledb*

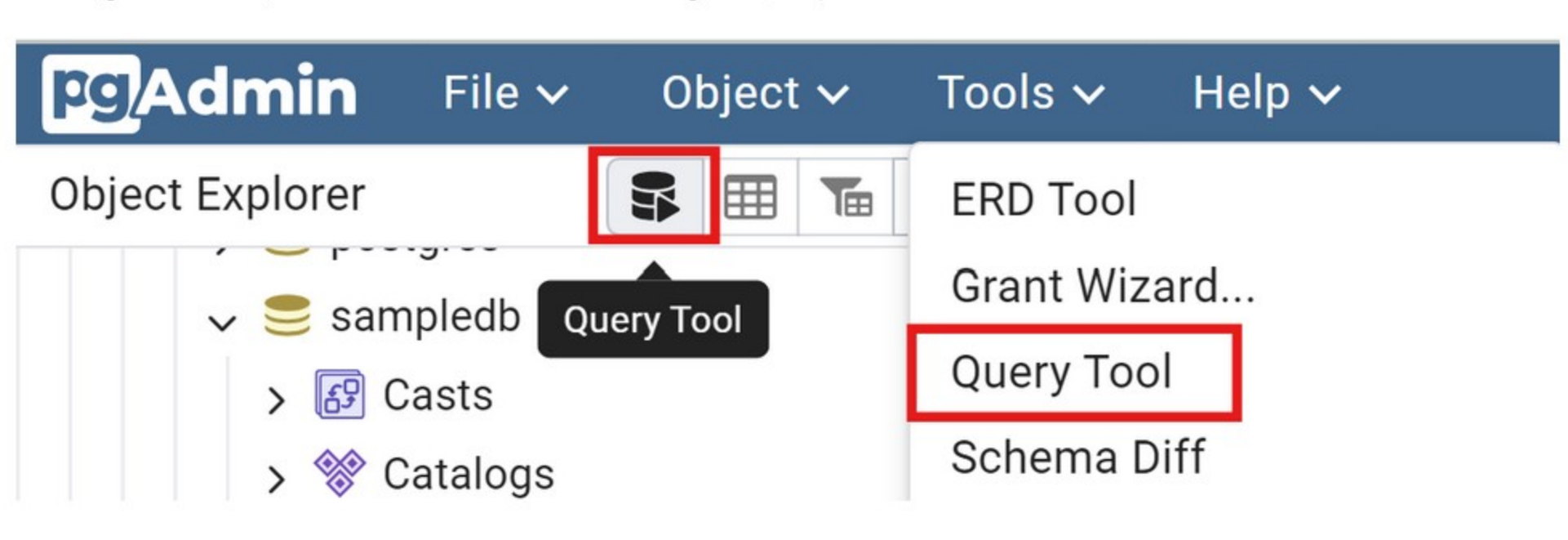
Click on S*chemas* and then on *Tables* to take a look at the tables that are part of the database.



Click on one of the tables and then click on Columns to take a look at the different columns of that particular table.

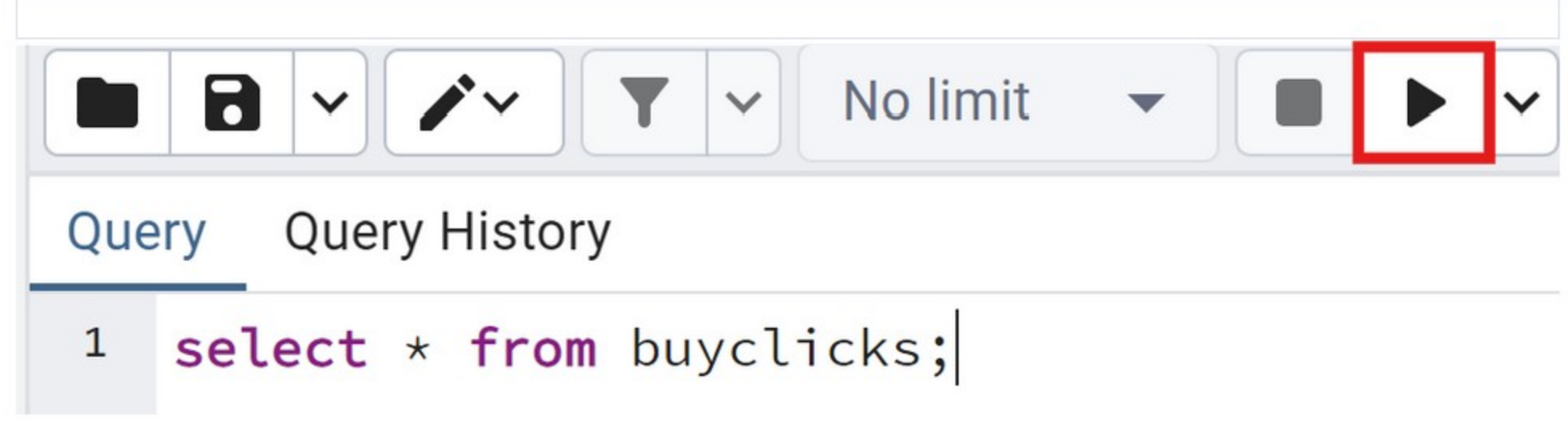


Step 8. Running queries to retrieve data. Select the *Query Tool,* either by clicking on the logo on the top left, or by clicking on the dropdown menu *Tools* and choosing the *Query Tool.*



Run *select * from buyclicks;* to see the contents from the buyclicks table. To execute the query, type *F5* on your keyboard, or click on the *Execute* button.

```
select * from buyclicks;
```



The *select ** means we want to query all the columns, and *from buyclicks* denotes which table to query. Notice the addition of the semicolon at the end of the query. While this addition is not strictly necessary on PgAdmin, it is a good practice to get used to its inclusion, as in most SQL systems queries cannot be executed unless they are closed with a semicolon.

The result of the query is:

	timestamp timestamp without time zone	txid integer	usersessionid integer	team integer	userid integer	buyid integer	price double precision
1	2016-05-26 15:36:54	6004	5820	9	1300	2	3
2	2016-05-26 15:36:54	6005	5775	35	868	4	10
3	2016-05-26 15:36:54	6006	5679	97	819	5	20
4	2016-05-26 16:36:54	6067	5665	18	121	2	3
5	2016-05-26 17:06:54	6093	5709	11	2222	5	20
6	2016-05-26 17:06:54	6094	5798	77	1304	5	20
7	2016-05-26 18:06:54	6155	5920	9	1027	5	20
8	2016-05-26 18:06:54	6156	5697	35	2199	2	3
9	2016-05-26 18:36:54	6183	5893	64	1544	5	20

Total rows: 1000 of 2947   Query complete 00:00:00.206

Step 9. Filter rows and columns. We can query only the *price* and *userid* columns with the following command:

```
select price, userid from buyclicks;
```

The result of this query is:

	price double precision	userid integer
1	3	1300
2	10	868
3	20	819
4	3	121
5	20	2222
6	20	1304
7	20	1027
8	3	2199

Total rows: 1000 of 2947   Query complete 00:00:00.146

We can also query rows that match a specific criteria. For example, the following command queries only rows with a price greater than 10:

```
select price, userid from buyclicks where price > 10;
```

The result is:

	price double precision	userid integer
1	20	819
2	20	2222
3	20	1304
4	20	1027
5	20	1544
6	20	1065
7	20	2221
8	20	1155

Total rows: 610 of 610   Query complete 00:00:00.131

Step 10. Perform aggregate operations. The SQL language provides many aggregate operations. We can calculate the average price:

```
select avg(price) from buyclicks;
```

The result is

	avg double precision
1	7.263997285374957

We can also calculate the total price:

```
select sum(price) from buyclicks;
```

The result is

	sum double precision
1	21407

The complete list of aggregate functions for Postgres can be found here: https://www.postgresql.org/docs/16/functions-aggregate.html

NOTE: The Postgres server used for this activity always uses the most recent version of Postgres (16 as of 2024). Check for the most recent version to consult the proper list of aggregate functions.

Step 11. Combine two tables. We combine the contents of two tables by matching or joining on a single column. If we look at the definition of the *adclicks* table

```
select column_name, data_type
from information_schema.columns
where table_name = 'adclicks';
```

	column_name name	data_type character varying
1	timestamp	timestamp without time zone
2	txid	integer
3	usersessionid	integer
4	teamid	integer
5	userid	integer
6	adid	integer
7	adcategory	character varying

We see that *adclicks* also has a column named *userid.* The following query combines the *adclicks* and *buyclicks* tables on the *userid* column in both tables:

```
select adid, buyid, adclicks.userid
from adclicks join buyclicks on adclicks.userid = buyclicks.userid;
```

This query shows the columns *adid* and *userid* from the *adclicks* table, and the *buyid* column from the *buyclicks* table. The *from adclicks join buyclicks* denotes that we want to combine these two tables, and *on adclicks.userid = buyclicks.userid* denotes which two columns to use when the tables are combined.

The result of the query is:

	adid integer	buyid integer	userid integer
1	2	5	611
2	2	4	611
3	2	4	611
4	2	5	611
5	2	4	611
6	2	1	611
7	21	1	1874
8	21	1	1874
9	21	3	1874

Total rows: 1000 of 103156   Query complete 00:00:00.116

Step 12. Generate a graph and download your results. Run the que following query

```
select avg(price) as avg_price, team
from buyclicks group by team order by team;
```

This query will calculate the mean price by team, name this avg column *avg_price*, select both columns and sort the results using the *team* column in ascending order. We could have included desc before the end of our query to order the data in descending order based on *team*.

Click on *Save results to file.* This will store your results into a csv file and download it locally.

	avg_price double precision	team
1	6.137931034482759	[illegible]
2	11.9375	8
3	5.23469387755102	9
4	9.171875	11
5	8.145454545454545	13
6	5.397435897435898	18
7	7.633333333333334	20
8	6.22093023255814	22

Save results to file F8

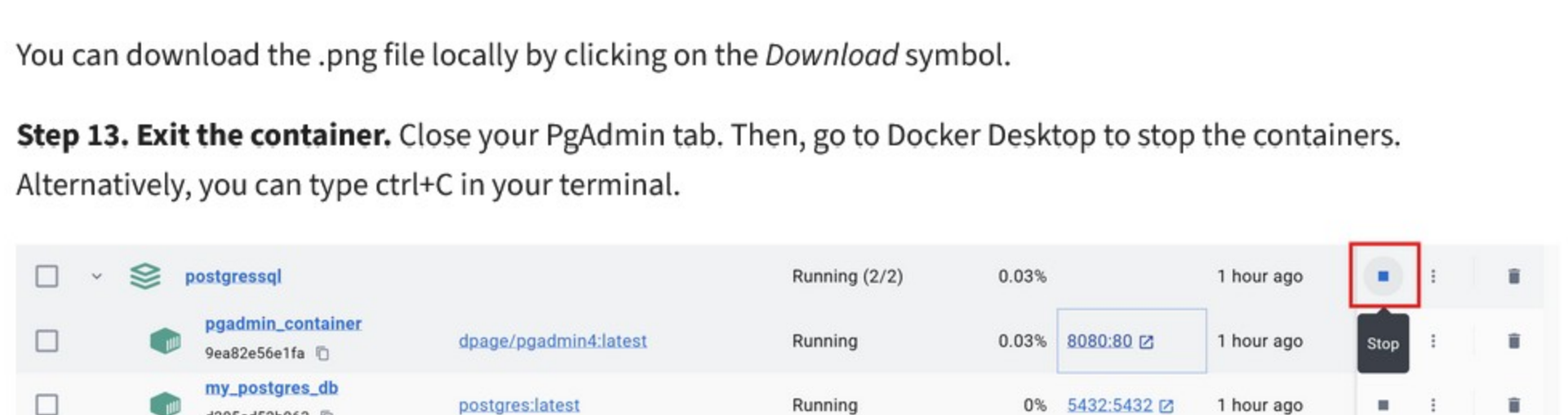
Click on *Graph Visualizer*

	avg_price double precision	team
1	6.137931034482759	2

Graph Visualiser

Select *Bar Chart* as the Graph Type, *team* on the X-axis and *avg_price* on the Y-axis. Click on *Generate.*

You should get the following chart



You can download the .png file locally by clicking on the *Download* symbol.

Step 13. Exit the container. Close your PgAdmin tab. Then, go to Docker Desktop to stop the containers. Alternatively, you can type ctrl+C in your terminal.

You can either keep the containers to come back later and keep practicing SQL queries, or delete them by clicking on the *Delete* button.

Mark as completed

Like   Dislike   Report an issue