Taylor Shrode
SQL Databases
2/2/2020
MSDS 610

Written Portion

1. What are some of the different SQL databases, and what are the pros and cons?

   There are many databases available for SQL, but a few of the most popular include Oracle 12c, MySQL, Microsoft SQL Server, and PostgreSQL (Arsenault, 2017). The newest version of Oracle is 12c, which has been designed for the cloud and can be hosted on multiple servers. Oracle offers the latest innovations and features and their database management tools are very strong. The downside of using Oracle is that it is expensive, which is limiting for smaller businesses (Arsenault, 2017). Oracle is best for large businesses that handle large databases and need the variety of features. Another popular database for web-based applications is MySQL. MySQL is available for free and offers a variety of user interfaces. However, an organization may have to spend a lot of time and effort to get MySQL to do things that other systems may do automatically, like create backups.

   Another popular cloud-based database is Microsoft SQL Server. Microsoft SQL is fast and stable and works well with other Microsoft products. Visualizations are also accessible on mobile devices (Arsenault, 2017). Although, the enterprise pricing may exceed what many organizations can afford. Finally, PostgreSQL is another popular free database that is designed for users that use both structured and unstructured data. PostgreSQL is scalable and can handle terabytes of data. However, configuration can be confusing, and documentation may be spotty. Meaning, so a user may find themselves online searching for a way to do something (Arsenault, 2017).

2. Why is it useful to know SQL?

   SQL was developed in the early 1970's by a computer scientist that worked for IBM. Since then, SQL has been widely adopted (Devlin, 2019). Some of the biggest names in tech use SQL. Names such as Uber, Netflix, Airbnb, Facebook, Google, and Amazon (Devlin, 2019). A quick job search on LinkedIn or Indeed.com will show that most data science jobs are looking for SQL skills. SQL is a universal language because when you use SQL, you're using other computer languages like C++, Javascript, Python, and others (Babu, 2018). While SQL may be older, it's everywhere (Devlin, 2019).

   SQL is also in high demand. Employers know that somebody who is skilled in SQL is highly favorably sought after (Babu, 2018). To demonstrate the importance of SQL in data-related jobs, Devlin (2019) analyzed 25,000 jobs on Indeed.com and found that 35.7% of those jobs had SQL mentioned as a job requirement. While, Python was only found in 25.7% of the jobs and R was mentioned in 17.9% of the jobs. SQL is becoming more popular than Python or R among data scientists and data engineers (Devlin, 2019).

3. What is database normalization and why is it important?

   Database normalization is a “process used to organize a database into tables and columns. The idea is that a table should be about a specific topic and that and only supporting topics included. (Wenzel, 2014)” In other words, normalization is a technique for organizing data (Li, 2019). There are three reasons to normalize a database: minimize duplicate data, minimize modification issues, and to simplify queries (Wenzel, 2014). Data duplicates increase storage and decrease performance. It also makes it more difficult to maintain data changes (Wenzel, 2014).

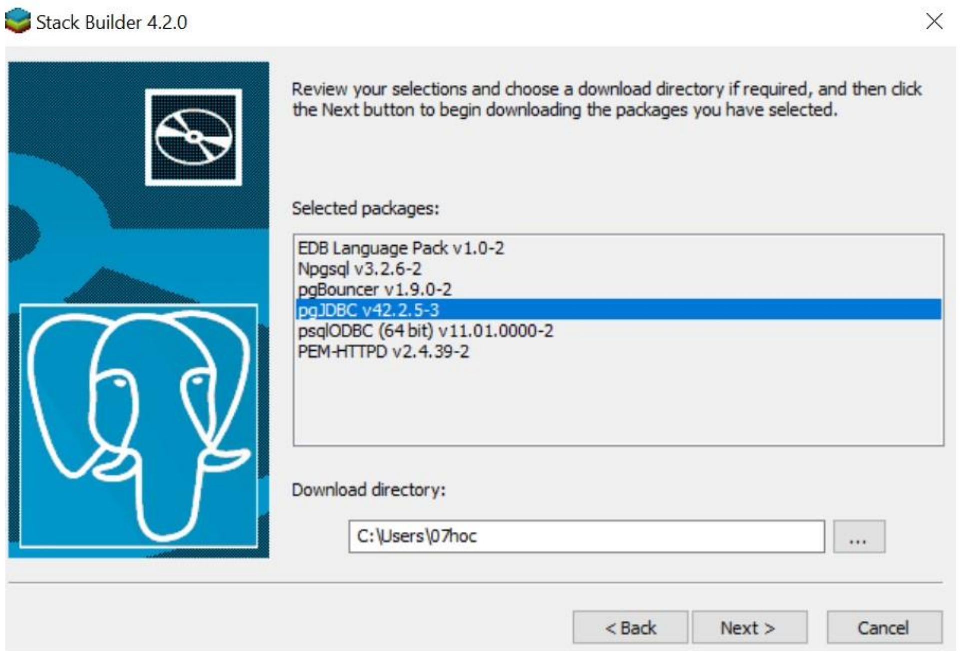
4. What is the difference between using Hive and something like PostgreSQL?

   The tools in Hive and in something like PostgreSQL retrieve data but they do it in very different ways. Hive is intended as a convenience for querying stored data, something like PostgreSQL is intended for online operations (Rathbone). Hive is better used when a user has large (terabytes or more) datasets to query. If extensibility is important, then Hive should also be used since Hive has a range of user function APIs that can be custom built (Rathbone). Something like PostgreSQL should be used if datasets are small (gigabytes) or performance is key. Hive is also an open source data system while something like PostgreSQL is used mainly for structured data processing (Educba).

Technical Portion

Structured Query Language (SQL) is used to communicate with a relational database. SQL is a nonprocedural language, which means the user specifies what needs to be done but does not specify how it is done (WorldClass). SQL is also relatively easy to learn since it contains a basic command set of less than 100 words and does not allow users to code logical statements such as “IF-THEN” statements (WorldClass).

For this assignment, we will be using a SQL database called PostgreSQL. PostgreSQL is a “general purpose and object-relational database management system” (What is PostgreSQL). It is also one of the most advanced open source database systems. PostgreSQL can be used on various platforms and is free and open source software. To set up and use a PostgreSQL database, PostgreSQL needs to be installed according to the user’s computer’s system type. While completing the “Setup” for PostgreSQL, a directory for the data to be stored, a password for the database superuser (postgres), the port number (5432), and the locale to be used by the database cluster will be created. After installation, Stack Builder needs to be set up. The following add-ons have been installed.



Stack Builder 4.2.0

Review your selections and choose a download directory if required, and then click the Next button to begin downloading the packages you have selected.

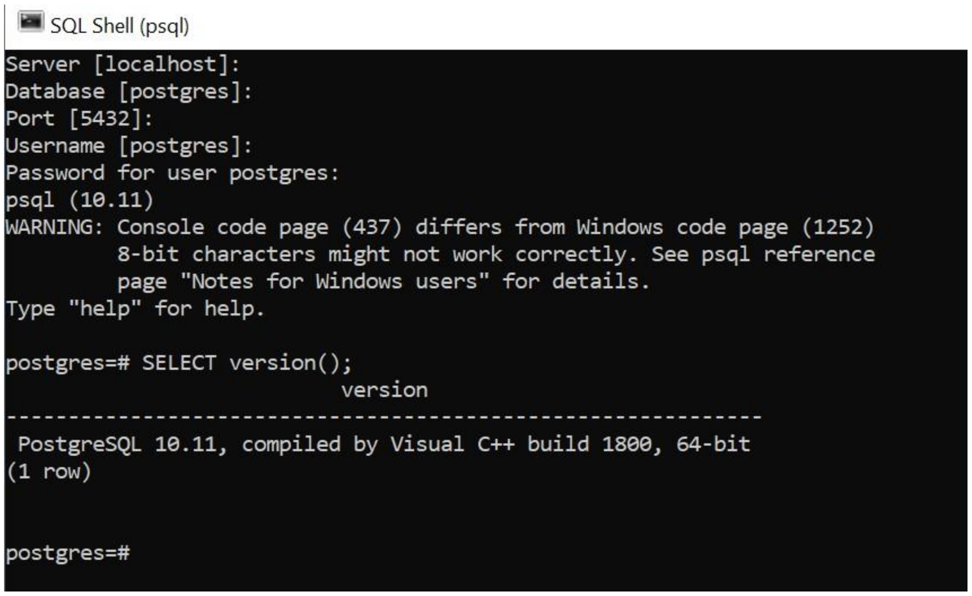
Selected packages:

EDB Language Pack v1.0-2
Npgsql v3.2.6-2
pgBouncer v1.9.0-2
pgJDBC v42.2.5-3
psqlODBC (64 bit) v11.01.0000-2
PEM-HTTPD v2.4.39-2

Download directory: C:\Users\07hoc

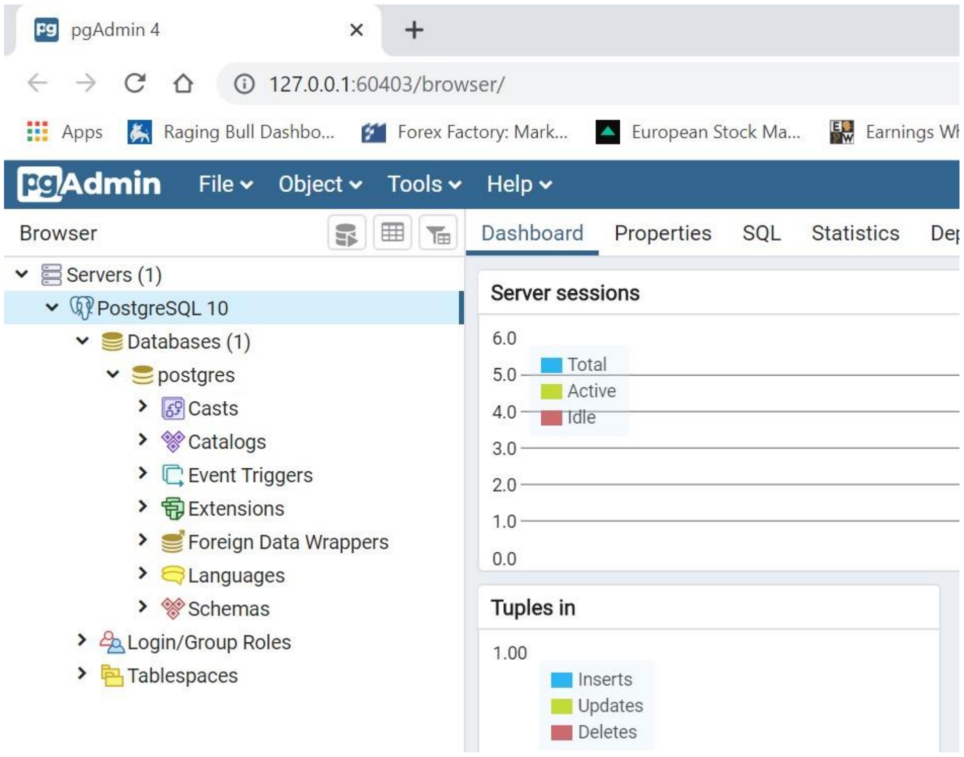
From here, we can connect to the PostgreSQL database server by launching **psql** (SQL Shell) from the start menu of our PC. After launching psql, the necessary information such as Server, Database, Username, and Password needs to be entered. Server, Database, and the Username is left as the default, but the Password is the password we created during the installation process. The SQL Shell will look like,

Taylor Shrode
SQL Databases
2/2/2020
MSDS 610


SQL Shell (psql)

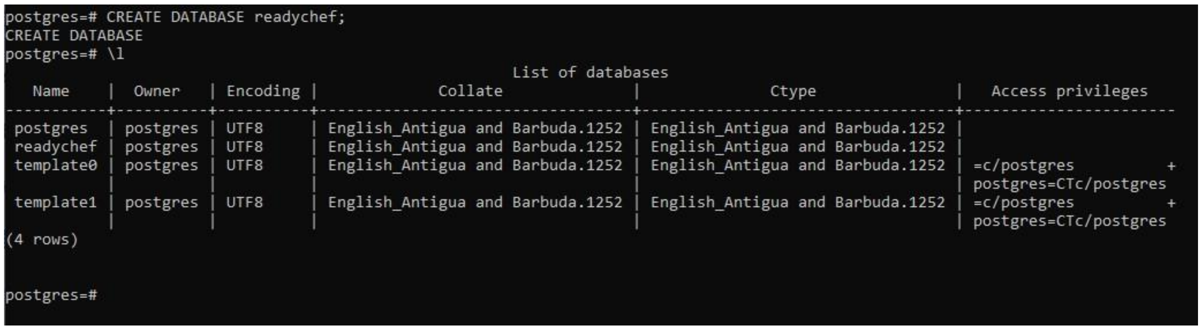
```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (10.11)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# SELECT version();
                           version
------------------------------------------------------------
 PostgreSQL 10.11, compiled by Visual C++ build 1800, 64-bit
(1 row)


postgres=#
```

To see the version of PostgreSQL that is being run, we use the command **SELECT version();** as seen above. We can also connect to the PostgreSQL database server using **pgAdmin**. By using the pgAdmin application, we can interact with PostgreSQL database via an intuitive user interface. Below is the pgAdmin page that is launched on the web browser.



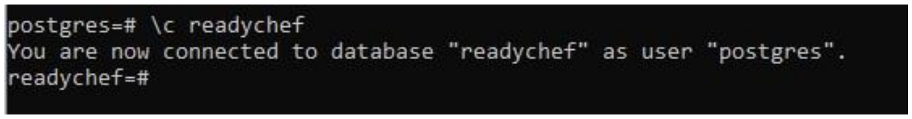
Taylor Shrode
SQL Databases
2/2/2020
MSDS 610

For this assignment, the SQL Shell was used. After launching psql again, and logging in with the necessary information, we can proceed by creating our **readychef** database.



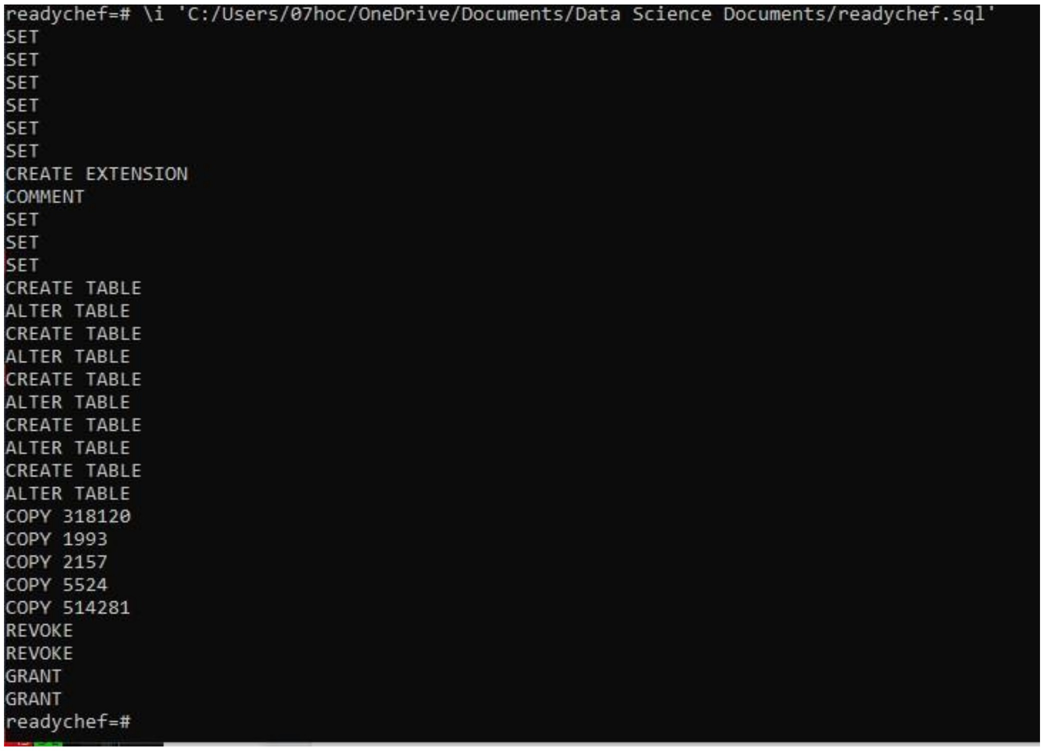
```
postgres=# CREATE DATABASE readychef;
CREATE DATABASE
postgres=# \l
                                              List of databases
   Name    |  Owner   | Encoding |             Collate              |              Ctype               |   Access privileges
-----------+----------+----------+----------------------------------+----------------------------------+-----------------------
 postgres  | postgres | UTF8     | English_Antigua and Barbuda.1252 | English_Antigua and Barbuda.1252 |
 readychef | postgres | UTF8     | English_Antigua and Barbuda.1252 | English_Antigua and Barbuda.1252 |
 template0 | postgres | UTF8     | English_Antigua and Barbuda.1252 | English_Antigua and Barbuda.1252 | =c/postgres          +
           |          |          |                                  |                                  | postgres=CTc/postgres
 template1 | postgres | UTF8     | English_Antigua and Barbuda.1252 | English_Antigua and Barbuda.1252 | =c/postgres          +
           |          |          |                                  |                                  | postgres=CTc/postgres
(4 rows)


postgres=#
```

We can view our databases, the command **\l** is used. To connect to a database, the command **\c <database name>** is used.



```
postgres=# \c readychef
You are now connected to database "readychef" as user "postgres".
readychef=#
```

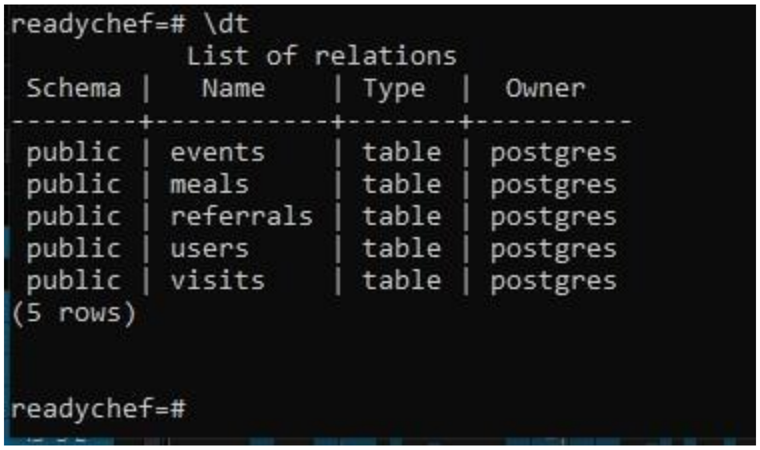
Notice how the prompt has been changed to **readychef=#**. Then, the **readychef.sql** file needs to be loaded by using the command **\i <path to file>** command.



```
readychef=# \i 'C:/Users/07hoc/OneDrive/Documents/Data Science Documents/readychef.sql'
SET
SET
SET
SET
SET
SET
CREATE EXTENSION
COMMENT
SET
SET
SET
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
COPY 318120
COPY 1993
COPY 2157
COPY 5524
COPY 514281
REVOKE
REVOKE
GRANT
GRANT
readychef=#
```

Taylor Shrode
SQL Databases
2/2/2020
MSDS 610

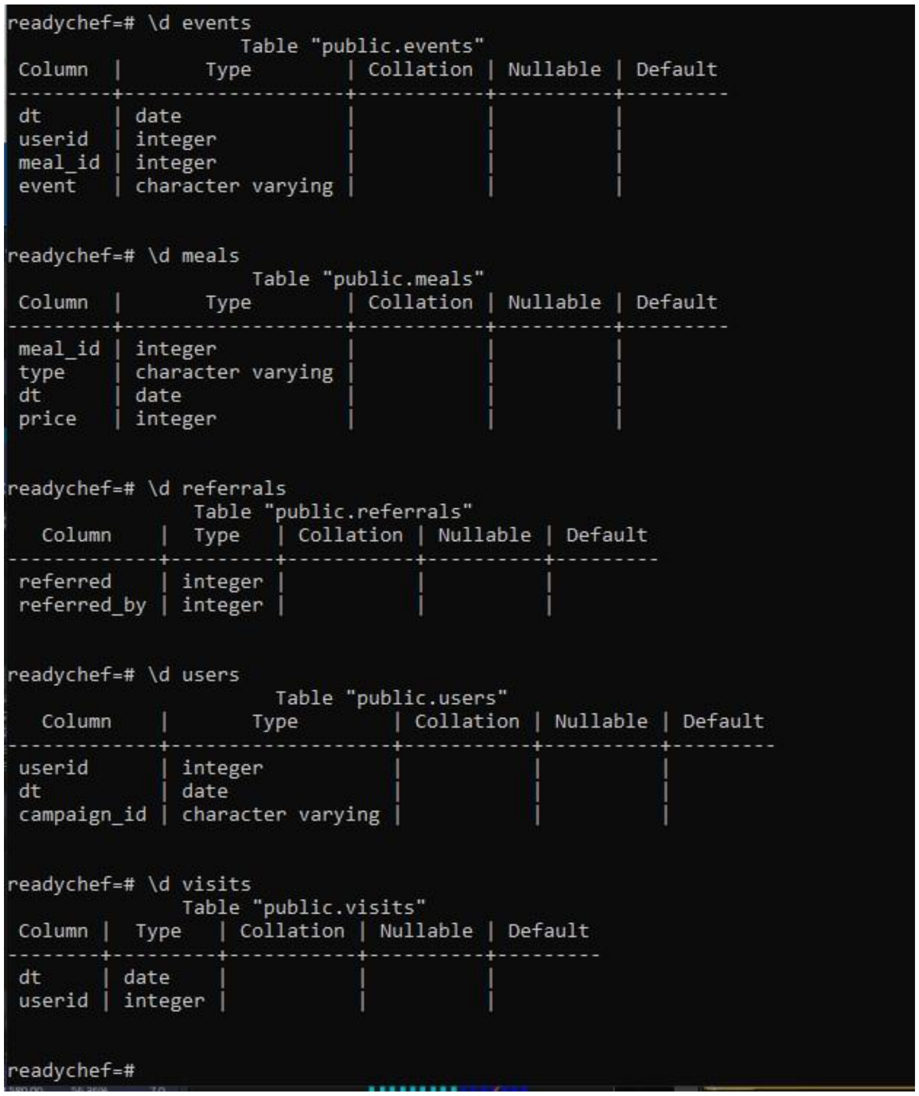
To view the tables in the **readychef** database, the command **\dt** is used.



```
readychef=# \dt
           List of relations
 Schema |   Name    | Type  |  Owner
--------+-----------+-------+----------
 public | events    | table | postgres
 public | meals     | table | postgres
 public | referrals | table | postgres
 public | users     | table | postgres
 public | visits    | table | postgres
(5 rows)


readychef=#
```

As seen above, there are five different tables in the **readychef** database labelled as **events, meals, referrals, users,** and **visits.** To view the structure of a table, we can use the command **\d <table name>**.

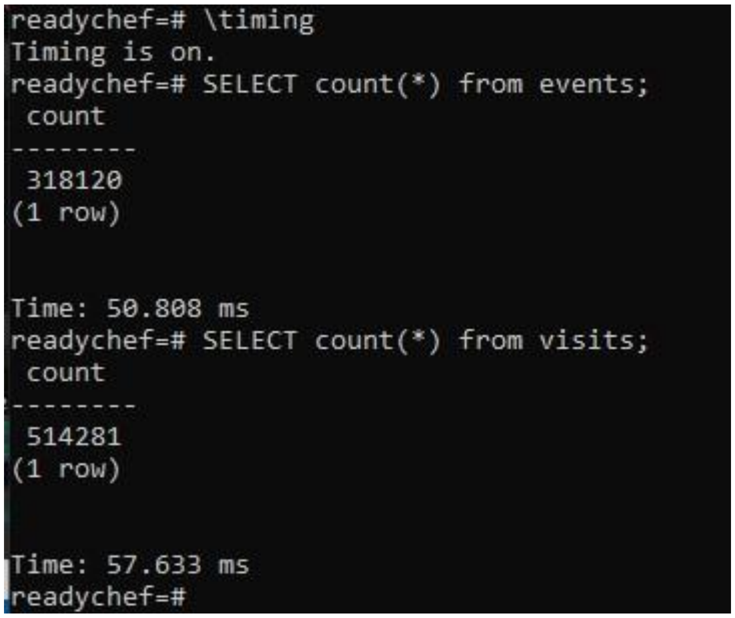
Taylor Shrode
SQL Databases
2/2/2020
MSDS 610



```
readychef=# \d events
                    Table "public.events"
 Column  |       Type        | Collation | Nullable | Default
---------+-------------------+-----------+----------+---------
 dt      | date              |           |          |
 userid  | integer           |           |          |
 meal_id | integer           |           |          |
 event   | character varying |           |          |


readychef=# \d meals
                     Table "public.meals"
 Column  |       Type        | Collation | Nullable | Default
---------+-------------------+-----------+----------+---------
 meal_id | integer           |           |          |
 type    | character varying |           |          |
 dt      | date              |           |          |
 price   | integer           |           |          |


readychef=# \d referrals
                Table "public.referrals"
   Column    |  Type   | Collation | Nullable | Default
-------------+---------+-----------+----------+---------
 referred    | integer |           |          |
 referred_by | integer |           |          |


readychef=# \d users
                       Table "public.users"
   Column    |       Type        | Collation | Nullable | Default
-------------+-------------------+-----------+----------+---------
 userid      | integer           |           |          |
 dt          | date              |           |          |
 campaign_id | character varying |           |          |


readychef=# \d visits
               Table "public.visits"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 dt     | date    |           |          |
 userid | integer |           |          |


readychef=#
```

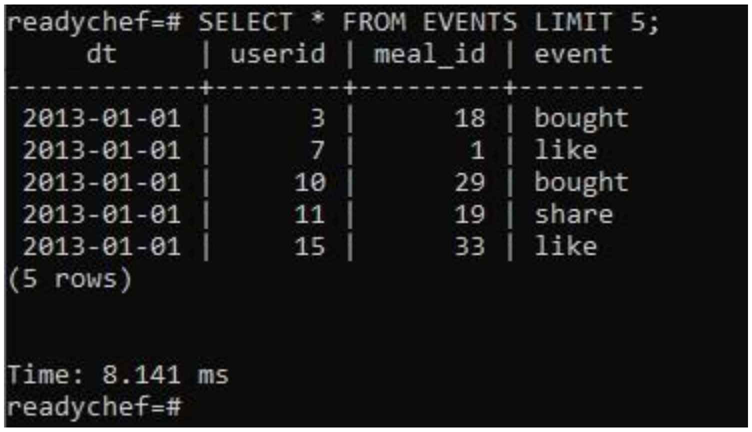
We can see that the **users** table has three columns labelled **userid, dt,** and **campaign_id**. It should be noted that the column **dt** refers to the date for each row in each table. To see how long it takes to run each query, the command **\timing** is used. We can also retrieve data from a table by using the **SELECT** command. For example, to count all the rows in a table, whether they contain NULL values or not, the command **SELECT count(*) from <table name>;** is used (Gravelle, 2018).

Taylor Shrode
SQL Databases
2/2/2020
MSDS 610



```
readychef=# \timing
Timing is on.
readychef=# SELECT count(*) from events;
 count
--------
 318120
(1 row)


Time: 50.808 ms
readychef=# SELECT count(*) from visits;
 count
--------
 514281
(1 row)


Time: 57.633 ms
readychef=#
```

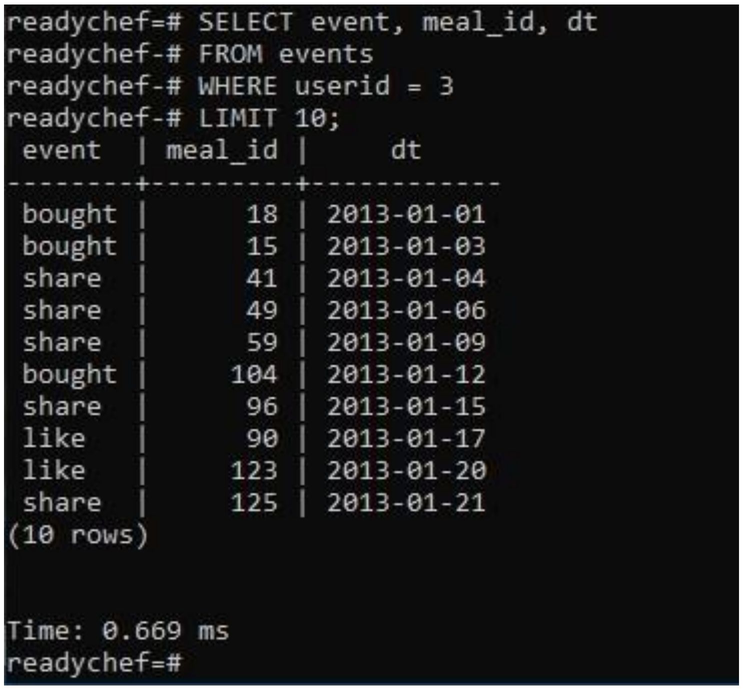
From above, we can see that the query to count the number of rows for the **events** table took 50.808 milliseconds and the query to count the number of rows for the **visits** table took 57.633 milliseconds. We can also view the rows of each table by using the command **SELECT * FROM <table name>;**. By adding **LIMIT <*x* number of rows>** after the <table name>, we can select *x* number of rows of that table. For example,



```
readychef=# SELECT * FROM EVENTS LIMIT 5;
     dt     | userid | meal_id | event
------------+--------+---------+--------
 2013-01-01 |      3 |      18 | bought
 2013-01-01 |      7 |       1 | like
 2013-01-01 |     10 |      29 | bought
 2013-01-01 |     11 |      19 | share
 2013-01-01 |     15 |      33 | like
(5 rows)


Time: 8.141 ms
readychef=#
```

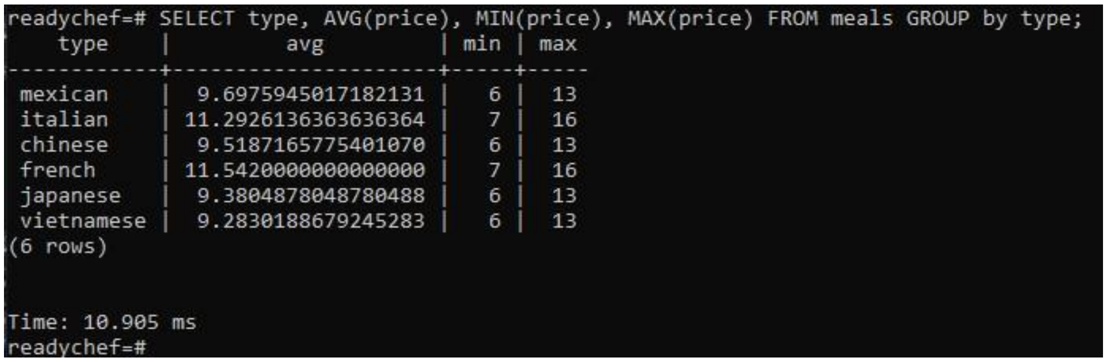
The **SELECT** command can also be used on several lines where the semi-colon tells psql to run the entire command. In the example below, the **WHERE** clause is used to query particular rows from a table that satisfy a certain condition (Practical psql Commands). The condition in this case is **WHERE userid = 3**.

Taylor Shrode
SQL Databases
2/2/2020
MSDS 610



```
readychef=# SELECT event, meal_id, dt
readychef-# FROM events
readychef-# WHERE userid = 3
readychef-# LIMIT 10;
 event  | meal_id |     dt
--------+---------+------------
 bought |      18 | 2013-01-01
 bought |      15 | 2013-01-03
 share  |      41 | 2013-01-04
 share  |      49 | 2013-01-06
 share  |      59 | 2013-01-09
 bought |     104 | 2013-01-12
 share  |      96 | 2013-01-15
 like   |      90 | 2013-01-17
 like   |     123 | 2013-01-20
 share  |     125 | 2013-01-21
(10 rows)


Time: 0.669 ms
readychef=#
```

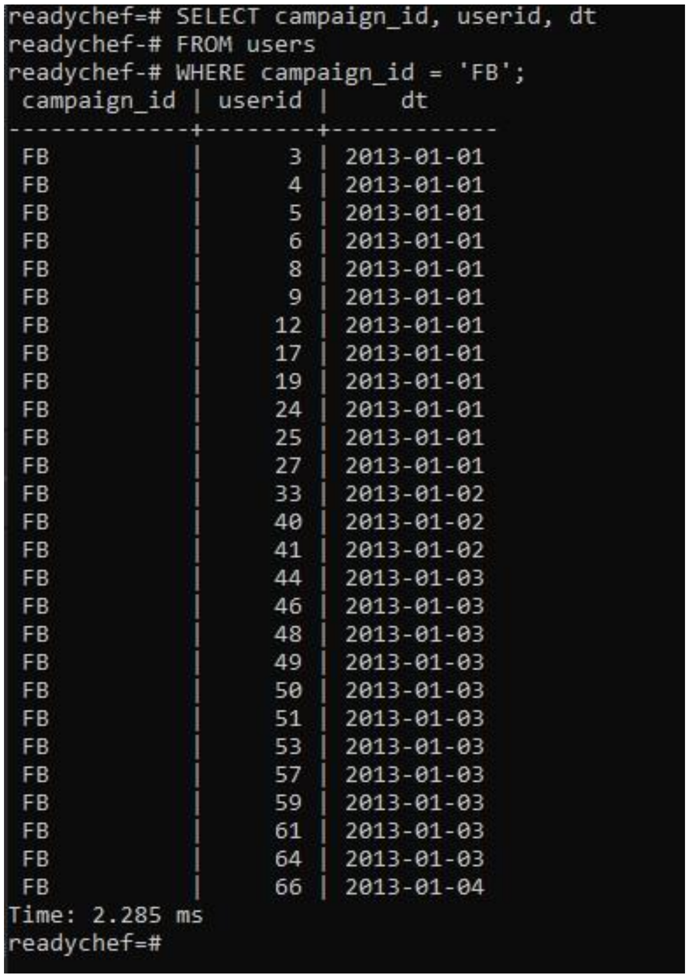
Next, we can find the average, minimum, and maximum price for each meal type by using the **GROUP BY** clause and the commands **AVG(), MIN(),** and **MAX()** where the item we want to calculate goes inside each function. We use the **GROUP BY** clause to divide the rows returned from the **SELECT** statement (Practical psql Commands). In this case, we want to calculate the average, minimum, and maximum price of each type of meal from the **meals** table. This can be seen below.



```
readychef=# SELECT type, AVG(price), MIN(price), MAX(price) FROM meals GROUP by type;
    type    |         avg         | min | max
------------+---------------------+-----+-----
 mexican    |  9.6975945017182131 |   6 |  13
 italian    | 11.2926136363636364 |   7 |  16
 chinese    |  9.5187165775401070 |   6 |  13
 french     | 11.5420000000000000 |   7 |  16
 japanese   |  9.3804878048780488 |   6 |  13
 vietnamese |  9.2830188679245283 |   6 |  13
(6 rows)


Time: 10.905 ms
readychef=#
```

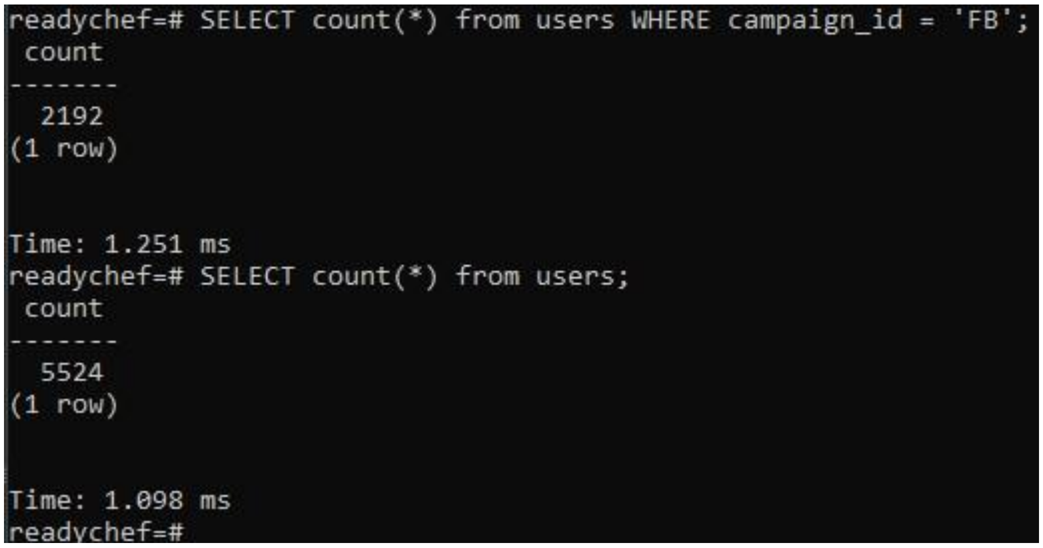
Using the **WHERE** clause, we can write a **SELECT** statement that returns all the rows from the **users** table where **campaign_id** is equal to **FB.**

Taylor Shrode
SQL Databases
2/2/2020
MSDS 610



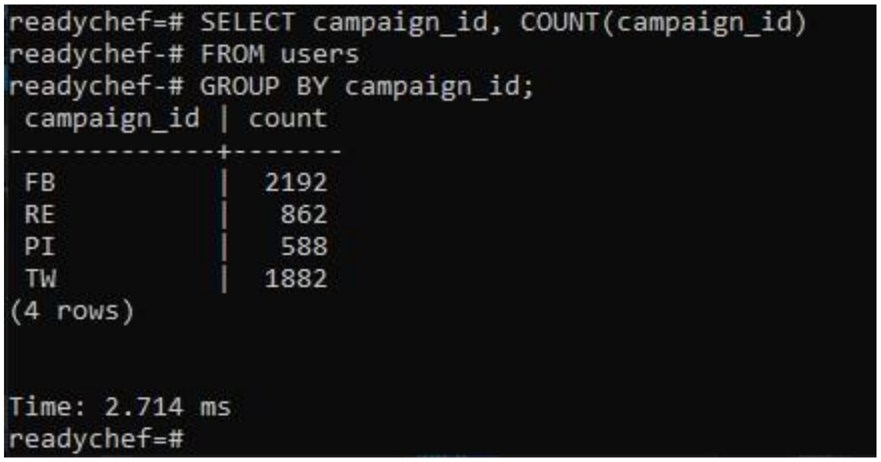
```
readychef=# SELECT campaign_id, userid, dt
readychef-# FROM users
readychef-# WHERE campaign_id = 'FB';
 campaign_id | userid |     dt
-------------+--------+------------
 FB          |      3 | 2013-01-01
 FB          |      4 | 2013-01-01
 FB          |      5 | 2013-01-01
 FB          |      6 | 2013-01-01
 FB          |      8 | 2013-01-01
 FB          |      9 | 2013-01-01
 FB          |     12 | 2013-01-01
 FB          |     17 | 2013-01-01
 FB          |     19 | 2013-01-01
 FB          |     24 | 2013-01-01
 FB          |     25 | 2013-01-01
 FB          |     27 | 2013-01-01
 FB          |     33 | 2013-01-02
 FB          |     40 | 2013-01-02
 FB          |     41 | 2013-01-02
 FB          |     44 | 2013-01-03
 FB          |     46 | 2013-01-03
 FB          |     48 | 2013-01-03
 FB          |     49 | 2013-01-03
 FB          |     50 | 2013-01-03
 FB          |     51 | 2013-01-03
 FB          |     53 | 2013-01-03
 FB          |     57 | 2013-01-03
 FB          |     59 | 2013-01-03
 FB          |     61 | 2013-01-03
 FB          |     64 | 2013-01-03
 FB          |     66 | 2013-01-04
Time: 2.285 ms
readychef=#
```

To count the number of users who came from Facebook (FB), we use the **COUNT()** command with the **WHERE** clause.



```
readychef=# SELECT count(*) from users WHERE campaign_id = 'FB';
 count
-------
  2192
(1 row)


Time: 1.251 ms
readychef=# SELECT count(*) from users;
 count
-------
  5524
(1 row)


Time: 1.098 ms
readychef=#
```

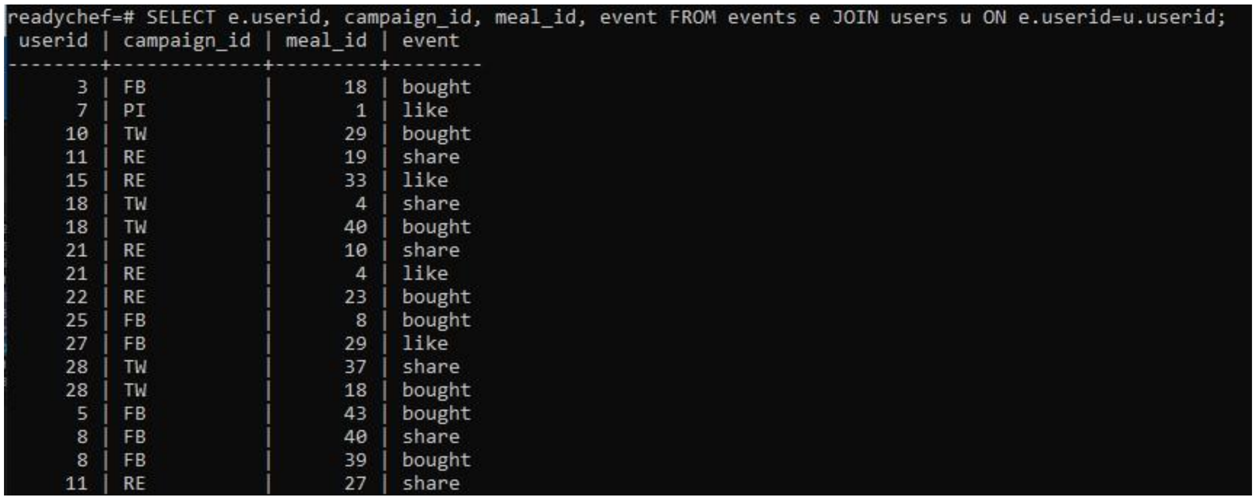
Similarly, we can count the number of users coming from each service. Here, we will also use the **GROUP BY** clause.



```
readychef=# SELECT campaign_id, COUNT(campaign_id)
readychef-# FROM users
readychef-# GROUP BY campaign_id;
 campaign_id | count
-------------+-------
 FB          |  2192
 RE          |   862
 PI          |   588
 TW          |  1882
(4 rows)


Time: 2.714 ms
readychef=#
```

Finally, we can also join tables together using the **JOIN** command. Here, we will write a query that joins the events table with the users table on **userid.**

Taylor Shrode
SQL Databases
2/2/2020
MSDS 610



```
readychef=# SELECT e.userid, campaign_id, meal_id, event FROM events e JOIN users u ON e.userid=u.userid;
 userid | campaign_id | meal_id | event
--------+-------------+---------+--------
      3 | FB          |      18 | bought
      7 | PI          |       1 | like
     10 | TW          |      29 | bought
     11 | RE          |      19 | share
     15 | RE          |      33 | like
     18 | TW          |       4 | share
     18 | TW          |      40 | bought
     21 | RE          |      10 | share
     21 | RE          |       4 | like
     22 | RE          |      23 | bought
     25 | FB          |       8 | bought
     27 | FB          |      29 | like
     28 | TW          |      37 | share
     28 | TW          |      18 | bought
      5 | FB          |      43 | bought
      8 | FB          |      40 | share
      8 | FB          |      39 | bought
     11 | RE          |      27 | share
```

As we can see above, PostgreSQL allows us to query data from a table by using techniques such as selecting columns, filtering rows, joining tables, and using different operations. The **SELECT** command allows us to query data from a single table. We can then filter that data by using commands **WHERE** and **LIMIT.** The data can be filtered further using the **GROUP BY** command which divides the rows into groups. The command **JOIN** allows to join multiple tables. We can also use other join methods such as the inner-join, left-join, or full-join, which was not shown here (What is PostgreSQL).

Taylor Shrode
SQL Databases
2/2/2020
MSDS 610

Resources

17 practical psql commands that you don't want to miss. (n.d.). Retrieved February 1, 2020, from https://www.postgresqltutorial.com/psql-commands/

Apache hive vs apache spark sql—13 amazing differences. (2018, March 11). *EDUCBA*. https://www.educba.com/apache-hive-vs-apache-spark-sql/

Arsenault, C. (2017, April 20). *The pros and cons of 8 popular databases*. KeyCDN. https://www.keycdn.com/blog/popular-databases

Babu, A. (2018, July 17). *Structured query language—Importance of learning sql*. https://codingsight.com/structured-query-language-importance-of-learning-sql/

Devlin, J. (2019, May 20). Why you need to learn sql if you want a job in data. *Dataquest*. https://www.dataquest.io/blog/why-sql-is-the-most-important-language-to-learn/

Gravelle, R. (2018, March 20). *Getting row counts in mysql(Part 1)*. https://www.navicat.com/en/company/aboutus/blog/695-getting-row-counts-in-mysql-part-1

Li, L. (2019, July 2). *Database normalization explained*. Medium. https://towardsdatascience.com/database-normalization-explained-53e60a494495

Rathbone, M. (n.d.). *Apache Hive vs MySQL - What are the key differences?* Matthew Rathbone's Blog. Retrieved February 2, 2020, from https://blog.matthewrathbone.com/2015/12/08/hive-vs-mysql.html

Taylor Shrode
SQL Databases
2/2/2020
MSDS 610

Untitled. (n.d.). Retrieved February 1, 2020, from https://worldclass.regis.edu/d2l/le/content/245302/Home?itemIdentifier=D2L.LE.Content.ContentObject.ModuleCO-3452009

Wenzel, K. (2014, June 17). Database normalization. *Essential SQL*. https://www.essentialsql.com/get-ready-to-learn-sql-database-normalization-explained-in-simple-english/

What is PostgreSQL. (n.d.). Retrieved February 1, 2020, from https://www.postgresqltutorial.com/what-is-postgresql/