SQL Databases

Lacey Conrad
MSDS 610
Regis University

May, 2020

1 Introduction

Over its nearly fifty year lifespan, SQL has become synonymous with the management of databases. Today, SQL is considered the database query standard and powers many database applications on the internet. Applications are written at both the corporate and individual level, with the latter due to its many open-source options available (such as MySQL and PostgreSQL) (Jan, 2018).

SQL is all about data management, so it shouldn't be surprising that it is one of the top skills listed by many data scientists and data engineers. In the following lab exercise, we will be using SQL to query a relational database (in our case, PostgreSQL). SQL is used in relational database management in three ways: 1) retrieve data, 2) write data, 3) update data. As a data scientist, we need to be able to retrieve data in order to be able to work with it, and SQL is one of the more common languages used to interact with data in relational databases (De, 2019).

The goal of this lab was to become comfortable programming in SQL. After downloading PostgreSQL on my PC, I loaded the readychef database from the class website into previously created readychef database. I then determined the structure of the tables within the database to make future queries easier to envision. At this point, I practiced data retrieval from the readychef database by creating queries as listed in the lab exercise, or from other examples I found.

2 Written portion - Questions from week 3

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

   (a) PostgreSQL: PostgreSQL is free and is used frequently for web databases. PostgreSQL can manage structure and unstructured data. PROS: the management engine is scalable and can handle massive data sets. It also support JSON. There are also predefined functions, and a number of interfaces. CONS: Spotty documentation that may lead to searching online for support. Configuration can also be confusing. Speed can also suffer in the case of bulk operations (Arsenault, 2017).

   (b) Microsoft SQL server: Microsoft SQL server's database management engine works on cloud and local servers, and it can be set up to work on both at the same time. Temporal data support is also offered, allowing a user to track changes made to the database over time. PROS: Fast and stable. The engine can adjust performance levels. Not surprisingly, the Microsoft SQL server works very well with other Microsoft products. CONS: Enterprise pricing, which may be too pricey for smaller companies. Even with the engine adjusting performance, the server can still be a resource hog. Many users also have issues importing files with the Microsoft SQL server (Arsenault, 2017).

   (c) MySQL: MySQL is another database that is popular for web-based applications. Even though it is free, it is frequently updated. PROS: It's free and still offers a lot of functionality. There are a variety of user interfaces which are easy to use. It can work with other databases. CONS: It can be time consuming to get MySQL to do standard database functions automatically. While there is a free version, you'll need to pay for support (Arsenault, 2017).

(d) Oracle 12c: This newest version of the Oracle database is designed for the cloud and can be hosted on one or more servers. There are a number of editions of the Oracle database, which allow for flexibility depending upon a users needs. PROS: Oracle sets the bar for database management tools, and as such, the latest innovations started here. The Oracle database management tools are robust, and it is easy to find one to suit your needs. CONS: The cost can be prohibitive. The database system requires significant resources and may require hardware upgrades, further increasing costs (Arsenault, 2017).

2. Why is it useful to know SQL? SQL is a good tool for any programmer to know since it is the go-to language for managing databases. Data scientists in particular should know SQL as the main purpose of SQL is database interaction. In particular, SQL is able to retrieve data from a database, write data into a database, and update data in a database. These are all activities a data scientist is going to be performing regularly throughout their career. In particular, data scientist need to be able to retrieve data in order to work with it.

   Here are a few reasons to know SQL (from Prasannika, 2016):

   (a) Data monitoring: Basic queries can be used to 1) monitor table activity, 2) identify data at time intervals, and 3) view update events.

   (b) SQL programmers are needed currently: it is one of the top programming jobs listed on many job search engines. Also, many career websites list SQL as the top skill for data scientists and data engineers.

   (c) Data manipulation: SQL lets you see the data you're working with, which allows for easier manipulation. Also, basic queries allow for easy, dynamic modification and manipulation of databases.

   (d) Data merging: SQL makes the process of combing data from two or more sources straight-forward. Fields can be "merged" or entire databases can be combined.

   (e) Management of large data sets: SQL can be used to manage virtually any size of data.

   (f) SQL is the most commonly used database language: SQL, despite its age and the popularity of NoSQL databases, is still considered to be the universal interface for data analysis.

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

   Database normalization is where a database is reorganized such that queries and analyses can be carried out by users in a clear and consistent manner. While the definition of database normalization is somewhat vague, that is a result of the very broad scope of what data normalization accomplishes. There are several goals for data normalization: to get rid of duplicate or redundant data, to consolidate data, removing (or solving) conflicting data, and formatting the data. Normalization help databases to be used more efficiently, since this process removes data anomalies that can result from previous data deletions, insertions, or updates. These anomalies can create errors in addition to complicating the data, and their removal will benefit any of the future uses of that data. Database normalization is particularly important when managing Big Data. Given how much of Big Data is unstructured, organizing and structuring this data will make it easier for data storage, retrieval, and analysis ("What is data normalization and why is it important?," 2019).

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

   Both of these tools primarily function as means of data retrieval, yet perform that function in different ways. Hive is one of the many tools within Hadoop. It is data warehousing software designed to manage distributed data sets. It acts as an interface into Hadoop MapReduce and acts as a querying tool for the data stored in the Hadoop HDFS. Hive isn't intended for online use with heavy read/write use, and performs much slower than PostgreSQL in this capacity. Its method of querying data is called "schema on read" which allows users to define tables without interacting with data in the database. This means you can make sense out of unstructured data with Hive if you're prepared for null answers because the data didn't fit the users expectations. Hive is designed for use on large data sets and complex queries (Rathbone, 2015).

PostgreSQL is relational database management software that is widely used and open source. Unlike Hive, PostgreSQL is "schema on write," where users must define the schema before data can be added to the database. This allows to fast reading and writing of data. PostgreSQL is best suited as a database where performance is important, such as for frequently and quick data pulling. Also, PostgreSQL will perform better with relatively small data sets. Data modification is also more easily performed in PostgreSQL than Hive (Rathbone, 2015).

3 Methods and Code

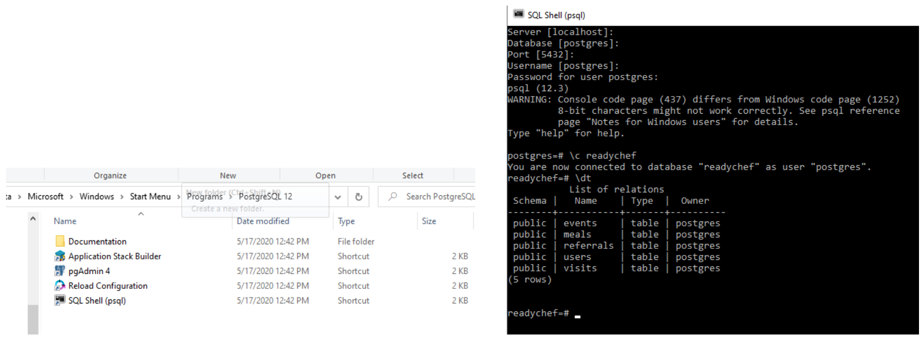
In order to perform any of this lab, I needed to be able to run PostgreSQL. Our lab handout gave us several options, and I decided upon a local installation of PostgreSQL on my PC (see figure 1). I followed the instructions on how to install the software, and eventually, I was able to open both user interfaces that came with the installation: psql and pgAdmin.

I began the coding portion of the lab by attempting to open and execute the code in the readychef sql file (`readychef.sql`), in pgAdmin. After receiving several errors, I decided to load the database using psql.

To load the database in psql, I executed the following command:

```
-- Connecting to the readychef database:
postgres=# \c readychef
-- Loading the readychef.sql file:
postgres=# \i C:\Users\Lacey!\Documents\MSDS610\readychef.sql
-- Displaying the tables in the database:
postgres=# \dt
```

To make sure the database was loaded correctly, I used the command `dt` to inspect the database:



Name	Date modified	Type	Size
Documentation	5/17/2020 12:42 PM	File folder	
Application Stack Builder	5/17/2020 12:42 PM	Shortcut	2 KB
pgAdmin 4	5/17/2020 12:42 PM	Shortcut	2 KB
Reload Configuration	5/17/2020 12:42 PM	Shortcut	2 KB
SQL Shell (psql)	5/17/2020 12:42 PM	Shortcut	2 KB

```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (12.3)
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=# \c readychef
You are now connected to database "readychef" as user "postgres".
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=# _
```

Figure 1: Left: Verification of local installation of PostgreSQL. Right: Verification of correct loading of `readychef.sql` in psql and displaying the loaded database structure.

At this point, I worked through the rest of the lab assignment in pgAdmin.

3.1 Readychef database organization

To make it easier to visualize the structure of the database and to help writing queries later, I viewed the properties of each table in pgAdmin and recreated the results here (for reference):

Table 1: Tables within the readychef database.

	Table Name
1	events
2	meals
3	referrals
4	users
5	visits

Table 2: Structure of the Events Table.

Column Name	Data Type
dt	date
userid	integer
meal_id	integer
event	character varying

Table 3: Structure of the Meals Table.

Column Name	Data Type
meal_id	integer
type	character varying
dt	date
price	integer

Table 4: Structure of the Referrals Table.

Column Name	Data Type
referred	integer
referred_by	integer

Table 5: Structure of the Users Table.

Column Name	Data Type
userid	integer
dt	date
campaign_id	character varying

Table 6: Structure of the Visits Table.

Column Name	Data Type
dt	date
userid	integer

3.2 SQL queries

I have included my SQL code below for each of the queries required in the lab exercise. Each SQL statement is then broken down to explain what the various clauses are doing to/with the data.

1. QUERY 1: Get the average, min and max price for each meal type.

```

-- QUERY 1:
SELECT type, AVG(price), MIN(price), MAX(price)
        FROM meals
        GROUP by type;
```

Description of the query:

Keyword	Result
SELECT type	Retrieve data from the type column
AVG(price)	of which I want to calculate the average price,
MIN(price)	the minimum price,
MAX(price)	and the maximum price.
FROM meals	The data is located in the meals table
GROUP BY type	and display the output by type of meal.
;	This is the end of the statement.

Table 7: Breaking down the SQL statement that collects the `AVG`, `MIN`, and `MAX` price for each meal type.

2. QUERY 2: Using the WHERE clause, write a new SELECT statement that returns all rows where Campaign_ID is equal to FB.

```

-- QUERY 2:
SELECT *
        FROM users
        WHERE campaign_id = 'FB'
        LIMIT 20;
```

Description of the query:

Keyword	Result
SELECT *	Retrieve all rows of data
FROM users	from the users table
WHERE campaign_id = 'FB'	where the value in the campaign_id column is equal to the string 'FB'.
LIMIT 20	I only want to see the first 20 results.
;	This is the end of the statement.

Table 8: Breaking down the SQL statement that returns all rows where campaign_id = 'FB'.

3. QUERY 3: Write a query to get the count of just the users who came from Facebook.

```

-- QUERY 3:
SELECT count(*)
        FROM users
        WHERE campaign_id = 'FB';
```

Description of the query:

Keyword	Result
SELECT count(*)	Retrieve the number of rows (count) of data
FROM users	from the users table
WHERE campaign_id = 'FB'	where the campaign_id is equal to the string 'FB'
;	This is the end of the statement.

Table 9: Breaking down the SQL statement which counts the number of users from Facebook.

4. QUERY 4: Count the number of users coming from each service. Here you'll have to group by the column you're selecting with a GROUP BY clause.

```

-- QUERY 4:
```

```sql
SELECT campaign_id, count(campaign_id)
        FROM users
        GROUP by campaign_id;
```

Description of the query:

Keyword	Result
SELECT campaign_id,	Retrive the rows of data from the campaign_id column
count(campaign_id)	and count the number of each type of campaign_id
FROM users	which is located in the users table
GROUP by campaign_id	and group the results according to campaign_id.
;	This is the end of the statement.

Table 10: Breaking down the SQL statement which counts the number of users coming from each service.

5. QUERY 5: Write a query to get one table that joins the events table with the users table (on userid)

```sql

-- QUERY 5:
SELECT e.userid, campaign_id, meal_id, event
        FROM events e
        JOIN users u on e.userid = u.userid
        LIMIT 20;
```

Description of the query:

Keyword	Result
SELECT e.userid,	Retrieve all results in the events userid column,
campaign_id,	the campaign_id column,
meal_id,	and the meal_id column
FROM events e	from the events table (and users table due to join)
JOIN users u on e.userid = u.userid	and combine the rows from the users and events
	table based on the related column, userid.
	e.userid is the primary key (userid of the events table)
	and u.userid (userid of the users table) is the foreign key
LIMIT 20	Limit the results to 20 lines of output.
;	This is the end of the statement.

Table 11: Breaking down the SQL statement which uses a `JOIN` clause to join the events table with the users table.

4 Results and Output

The following are the results from the 5 different queries I ran as instructed in the lab handout.

1. QUERY 1: Get the `AVG`, `MIN` and `MAX` price for each meal type.

	type	**AVG**	**MIN**	**MAX**
1	mexican	9.7	6	13
2	italian	11.3	7	16
3	chinese	9.5	6	13
4	french	11.5	7	16
5	japanese	9.4	6	13
6	vietnamese	9.3	6	13

Table 12: The average, minimum, and maximum price for each meal type.

As we can see from the table, the French meals had the highest average meal price and shared the highest maximum meal price with Italian meals. The lowest meal price was 6 and was shared among Mexican, Chinese, Japanese, and Vietnamese meals.

2. QUERY 2: Using the `WHERE` clause, write a new `SELECT` statement that returns all rows where campaign_ID is equal to FB.

	userid	**date**	**campaign_id**
1	3	2013-01-01	FB
2	4	2013-01-01	FB
3	5	2013-01-01	FB
4	6	2013-01-01	FB
5	8	2013-01-01	FB
6	9	2013-01-01	FB
7	12	2013-01-01	FB
8	17	2013-01-01	FB
9	19	2013-01-01	FB
10	24	2013-01-01	FB
	**		

Table 13: All rows of data where the campaign_id was listed as FB. **I have included the first 10 results for visual inspection. The total number of results for this query was 1000.

While there aren't any descriptive statistics to discuss here, this remains an important method for requesting data on a certain group within a database. In this case, we are querying the database to give us the user ID and date joined from anyone who (I assume) joined from a Facebook campaign.

3. QUERY 3: Write a query to get the count of just the users who came from Facebook.

	count
1	2192

Table 14: The number of users coming from Facebook.

Similar to the previous query, this is a simple way to count how many users fall into a group. In this case, we count users from the Facebook campaign. In the following, we can extend the count to include the different campaigns, and see how many users were recruited from each campaign type.

4. QUERY 4: Count the number of users coming from each service.

	campaign_id	**count**
1	FB	2192
2	RE	862
3	PI	588
4	TW	1882

Table 15: A count of users coming from each service.

From the table, we see that the Facebook campaign included the most users at 2192, followed by TW which brought in 1882. In third and forth places we have RE with 862 users and PI with 588 users.

5. QUERY 5: Write a query to get one table that joins the events table with the users table (on userid).

	userid	**campaign_id**	**meal_id**	**event**
1	3	FB	18	bought
2	7	PI	1	like
3	10	TW	29	bought
4	11	RE	19	share
5	15	RE	33	like
6	18	TW	4	share
7	18	TW	40	bought
8	21	RE	10	share
9	21	RE	4	like
10	22	RE	23	bought
	**			

Table 16: The results of joining the events table with the users table. **I have included the first 10 results for visual inspection. The total number of results for this query was 318120.

This table shows how we can increase the amount of data obtained from a query by using a join. Using userid as the primary and foreign key for the events and users table (respectively) we are able to include data columns from both the events and users tables.

5 Analysis

The simple SQL commands I used in this lab exercise indicate that it is a powerful tool for the extraction, organization, and management of structured data. Three to four lines of code can retrieve a very specific group of data from the database quickly. If this beginner-friendly SQL exercise code is any indication, it is easy to assume that SQL can be utilized for very intricate and accurate handling of data.

6 Conclusion

Even though this wasn't my first experience with SQL, I am still amazed at how simple the programming language is. It is hard to believe that a user can completely manage a database with under 100 command words. I have, thus far, no other database programming language to compare it to, so unfortunately I can only say what a useful tool PostgreSQL/SQL is and how refreshingly easy it seems to be to learn.

Structured data is almost exclusively what I have worked with throughout my career, with most of that data being kept in spreadsheets like excel. I always understood data to be structured, and in my mind, that was the only way it existed. I am very interested in learning about unstructured data, as it seems at odds with a lot of the basic statistical analyses and data handling I performed in the past.

Lastly, I am not sure where I stand in the debate on if SQL/relational databases will be around forever. My suspicion is if NoSQL or some other unstructured database model can enact ACID properties on their database, that relational/SQL databases may become a thing of the past. For example, institutions like banking will always require top-notch database security, and the health care industry will always need consistent data. There is no telling if a database model is already in the works that combines ACID properties with a NoSQL model. Right now, data sets are becoming so big, that the amount organizations spend on relational database infrastructure is sometimes more than the building it's being housed in. And since data is just getting bigger, this problem will just get worse. Yet, maybe relational databases will find a manner to scale better, although since they have been around so long it would be expected that they would have already figured out how.

7 References

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

2. De, S. (2019, September 26). *Is SQL needed to be a data scientist?* Tec4Tric. https://tec4tric.com/2018/03/is-sql-needed-for-data-scientist.html?utm_campaign=News&utm_medium=Community&utm_source=DataCamp.com

3. Jan, A. (2018, March 13). *What is SQL server and how does it work?* QuickStart. https://www.quickstart.com/blog/what-is-sql-server-and-how-does-it-work/

4. Prasannika. (2016, February 18). *6 reasons why you should learn SQL.* Stone River eLearning. https://blog.stoneriverelearning.com/6-reasons-why-you-should-learn-sql/

5. Rathbone, M. (2015, December 8). *Apache hive vs MySQL - What are the key differences?* Matthew Rathbone's Blog. https://blog.matthewrathbone.com/2015/12/08/hive-vs-mysql.html

6. *What is data normalization and why is it important?* (2019, May 7). Import.io. https://www.import.io/post/what-is-data-normalization-and-why-is-it-important/