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

There are a few ways to classify databases and Structured Query Language (SQL) databases. They can be broken down into SQL (relational) and NoSQL (non-relational) databases. Both have their niche and one is not better than the other. SQL databases are appropriate for more complex queries while NoSQL databases are more appropriate for horizontal scaling and flexible requirements.

Some of the more popular databases today are MySQL, MariaDB, MongoDB, Redis, and PostgreSQL. MySQL is an open-source SQL (relational) database. It is popular in web development stacks. Pros are that it is open-source (free) and it has high performance when dealing with large databases. Cons are that it is difficult to create incremental backups, and that it also has no support for Extensible Markup Language (XML) or Online Analytical Processing (OLAP).

MariaDB is a variation of MySQL after MySQL was purchased by the Oracle Corporation. Pros are that it is scalable and its performance is good, and it implements encryption at multiple levels. Cons are that migration of the data is not always so simple.

MongoDB is actually NoSQL (non-relational) but is a tool for database management application. It is free and open-source which is a pro for users. It is a document-oriented database and is an appropriate choice for semi-structured data such as text documents, news headlines, etc. It has support for JSON and is high performance for large amounts of data. Some cons of MongoDB are that it uses a lot of memory, has limited default security, and has limited nesting of files.

Redis is a new tool and means "Remote Dictionary Server". It is also open-source, like so many of the other tools listed. It is also NoSQL in addition to MongoDB and Redis is actually very similar to MongoDB. It is primarily used for key-value stores. Pros are that it is high speed and configuration is easy. Many data types are supported. Some cons are that it takes a high amount of memory to run and has no support for join queries.

PostgreSQL is a tool that primarily utilizes object-relational DBMS. It is highly scalable and supports JSON format. Some cons are that it is not straightforward to configure, and doesn't necessarily have high performance at high-usage.

- Why is it useful to know SQL?

It is useful to know SQL because, put simply, it is relevant in many circles of the data science and data engineering world. SQL is a very popular language for data analytics and any engineer or person who knows the SQL language will find themselves more desirable than one who does not.

- What is database normalization and why is it important?

Database normalization is essentially organizing a database. It involves creating tables and relationships between those tables in order to protect the data and at the same time make it more flexible. There are "normal forms" that describe how the data has been normalized according to the few rules for data normalization. The data can be said to be in its first normal form if the first rule of normalization has been followed. Third normal form is the highest level of normalization necessary for applications. Database normalization is important because it organizes the data and reduces redundancy. It is an iterative process and breaks down a large table into smaller more manageable units.

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

With respect to Hive, PostgreSQL is the preferred tool overall. It has many more features and plugins and is easy to setup. There have also been some reviews stating that Hive is a bit slow compared to other languages. Hive is built on Hadoop so it may be a better choice for those with existing Hadoop experience. It has also been stated that Hive handles complicated data more effectively than SQL. SQL would be better suited for less-complicated data sets. Hive is not a free service either, so that must be considered.

Also submit SQL code (.sql files with comments explaining what you are doing in the code) for querying the SQL database to find the following from the readychef.sql database:

1. Get the average, min and max price for each meal type.
2. Using the `WHERE` clause, write a new `SELECT` statement that returns all rows where `Campaign_ID` is equal to `FB` .
3. Write a query to get the count of just the users who came from Facebook.
4. Now, 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.
5. Write a query to get one table that joins the events table with the users table (on `userid`).

Technical Section

To complete the assignment, installation of PostgreSQL was necessary. Rather than run inside a container, I wanted to run PostgreSQL inside of my existing VM. To install PostgrSQL on my Red Hat Linux VM, I performed the following:

```
# Install the repository RPM:
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm

# Disable the built-in PostgreSQL module:
sudo dnf -qy module disable postgresql

# Install PostgreSQL:
sudo dnf install -y postgresql14-server

# Optionally initialize the database and enable automatic start:
sudo /usr/pgsql-14/bin/postgresql-14-setup initdb
sudo systemctl enable postgresql-14
sudo systemctl start postgresql-14
```

FROM https://www.postgresql.org/download/linux/redhat/

After these initial commands, I also had to install the following packages: postgresql-server, postgresql-client, postgresql-contrib, and postgresql-devel.

I then saw that the postgres user's home directory is /var/lib/pgsql, so I copied the readychef.sql file to that directory. I was then able to access the .sql file in PostgreSQL. I entered 'psql' to open PostgreSQL and used the CREATE command to create a new database for the readychef dataset.

```
postgres=# CREATE DATABASE readychef;
CREATE DATABASE
postgres=# \q
```

Using the left carrot symbol, I was then able to load the file into the PostgreSQL database.

```
[postgres@localhost data]$ psql readychef < 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
```

It took me a few tries but I learned I had to enter '\c readychef' to enter the database I just created.

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

Before finding the average, min, and max of the price per type, I wanted to find the average, min, and max of the prices as a whole, to get a feel for the data. Shown below is that effort.

```
readychef=# SELECT MIN(price) from meals;
 min
-----
   6
(1 row)

readychef=# SELECT MAX(price) from meals;
 max
-----
  16
(1 row)

readychef=# SELECT AVG(price) from meals;
         avg
---------------------
 10.6522829904666332
(1 row)
```

I then queried the average, min, and max of the price per type.

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

I then began to explore the users table to find users which had come from facebook. In this case, their campaign_id would be 'FB'.

```
readychef=# SELECT userid, campaign_id from users WHERE campaign_id = 'FB';
 userid | campaign_id
--------+-------------
      3 | FB
      4 | FB
      5 | FB
      6 | FB
      8 | FB
      9 | FB
     12 | FB
     17 | FB
     19 | FB
     24 | FB
     25 | FB
     27 | FB
     33 | FB
     40 | FB
     41 | FB
     44 | FB
     46 | FB
     48 | FB
     49 | FB
     50 | FB
     51 | FB
     53 | FB
     57 | FB
     59 | FB
```

```
readychef=# SELECT campaign_id, COUNT(userid) FROM users WHERE campaign_id = 'FB' GROUP BY campaign_id;
 campaign_id | count
-------------+-------
 FB          |  2192
(1 row)
```

After finding the count of just users with the campaign_id 'FB', I then expanded that command to include the count of all campaign_id's.

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

I then joined the users table and the events table on the userid column. However, I found I had to make these column names distinct from each other as previously they were both named userid. I changed these column names to userid_users and userid_events. I then joined the tables with the following command:

```
readychef=# SELECT * FROM users INNER JOIN events ON userid_users = userid_events;
 userid_users |     dt     | campaign_id |     dt     | userid_events | meal_id | event
--------------+------------+-------------+------------+---------------+---------+--------
            3 | 2013-01-01 | FB          | 2013-01-01 |             3 |      18 | bought
            7 | 2013-01-01 | PI          | 2013-01-01 |             7 |       1 | like
           10 | 2013-01-01 | TW          | 2013-01-01 |            10 |      29 | bought
           11 | 2013-01-01 | RE          | 2013-01-01 |            11 |      19 | share
           15 | 2013-01-01 | RE          | 2013-01-01 |            15 |      33 | like
           18 | 2013-01-01 | TW          | 2013-01-01 |            18 |       4 | share
           18 | 2013-01-01 | TW          | 2013-01-01 |            18 |      40 | bought
           21 | 2013-01-01 | RE          | 2013-01-01 |            21 |      10 | share
           21 | 2013-01-01 | RE          | 2013-01-01 |            21 |       4 | like
           22 | 2013-01-01 | RE          | 2013-01-01 |            22 |      23 | bought
           25 | 2013-01-01 | FB          | 2013-01-01 |            25 |       8 | bought
           27 | 2013-01-01 | FB          | 2013-01-01 |            27 |      29 | like
           28 | 2013-01-01 | TW          | 2013-01-01 |            28 |      37 | share
           28 | 2013-01-01 | TW          | 2013-01-01 |            28 |      18 | bought
            5 | 2013-01-01 | FB          | 2013-01-02 |             5 |      43 | bought
            8 | 2013-01-01 | FB          | 2013-01-02 |             8 |      40 | share
            8 | 2013-01-01 | FB          | 2013-01-02 |             8 |      39 | bought
           11 | 2013-01-01 | RE          | 2013-01-02 |            11 |      27 | share
           20 | 2013-01-01 | RE          | 2013-01-02 |            20 |      11 | like
           21 | 2013-01-01 | RE          | 2013-01-02 |            21 |      35 | share
           21 | 2013-01-01 | RE          | 2013-01-02 |            21 |      36 | like
           23 | 2013-01-01 | TW          | 2013-01-02 |            23 |      36 | share
           25 | 2013-01-01 | FB          | 2013-01-02 |            25 |      28 | like
           26 | 2013-01-01 | TW          | 2013-01-02 |            26 |       9 | share
           26 | 2013-01-01 | TW          | 2013-01-02 |            26 |      33 | like
           27 | 2013-01-01 | FB          | 2013-01-02 |            27 |       3 | like
           32 | 2013-01-02 | RE          | 2013-01-02 |            32 |      41 | share
           32 | 2013-01-02 | RE          | 2013-01-02 |            32 |      14 | like
           35 | 2013-01-02 | RE          | 2013-01-02 |            35 |      12 | share
           35 | 2013-01-02 | RE          | 2013-01-02 |            35 |      35 | bought
            3 | 2013-01-01 | FB          | 2013-01-03 |             3 |      15 | bought
            7 | 2013-01-01 | PI          | 2013-01-03 |             7 |      48 | share
            8 | 2013-01-01 | FB          | 2013-01-03 |             8 |      21 | share
```

This fulfilled all the tasks which were assigned to us this week. I enjoyed the assignment this week! I've heard a lot about SQL but have not used it much myself so it was fun to finally get my hands a little dirty and learn what SQL is all about.

Thank you,

Jeremy

References

1) Smallcombe, M. (2020, November 18). *Hive vs. SQL: Which One Performs Data Analysis Better?* Integrate.Io. Retrieved May 22, 2022, from https://www.integrate.io/blog/hive-vs-sql/
2) *Compare Hive and PostgreSQL | Cloudflare*. (n.d.). G2. Retrieved May 22, 2022, from https://www.g2.com/compare/hive-vs-postgresql
3) M. (2022, May 5). *Database normalization description - Office*. Microsoft Docs. Retrieved May 22, 2022, from https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description
4) Babeni, S. (2020, May 28). *Most Popular Databases in 2020: Here's How They Stack Up*. Ormuco. Retrieved May 22, 2022, from https://ormuco.com/blog/most-popular-databases
5) *PostgreSQL: Linux downloads (Red Hat family)*. (n.d.). The PostgreSQL Global Development Group. Retrieved May 22, 2022, from https://www.postgresql.org/download/linux/redhat/