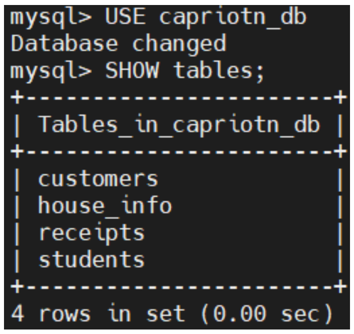
Nicholas Capriotti
CS 317 - Dr. Van Hilst

Assignment 2: Nested Queries

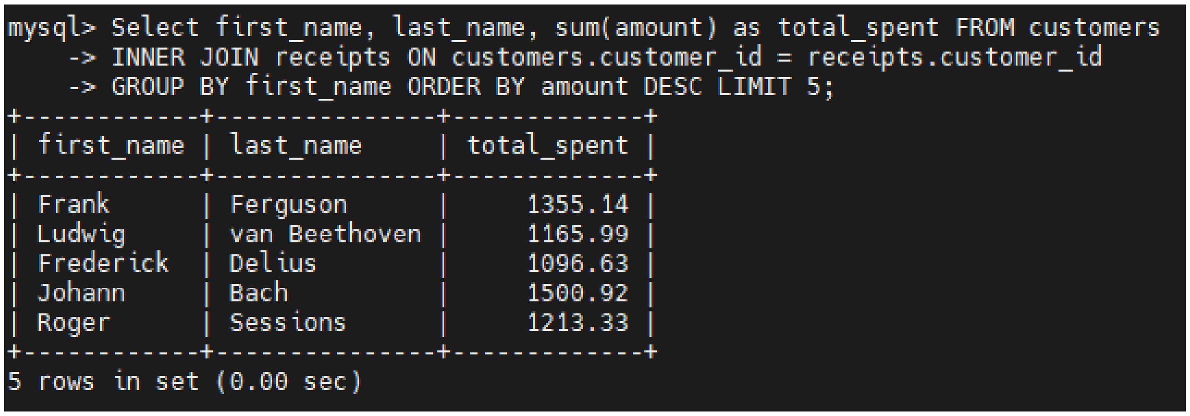
MySQl allows for the use of nested queries, which return values to the outer queries to further organize or restrict the data being shown. Sub queries are mainly used with the SELECT, WHERE, DELETE, and BETWEEN clauses. This document provides an introduction to nested queries and how to properly format your query based on the information given and needed. This all takes place on the MOBAXTerm virtual machine, so minimal LINUX proficiency is required.

The first step is to complete the login process described in the last document. Once the login is complete, select your own database with “USE username_db”, The next step is to upload the given tables into your own database so that we can manipulate them.



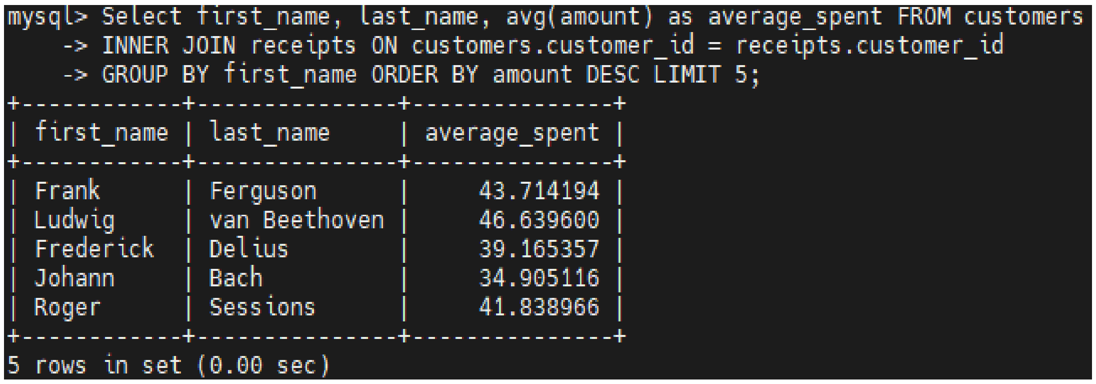
```
mysql> USE capriotn_db
Database changed
mysql> SHOW tables;
+----------------------+
| Tables_in_capriotn_db |
+----------------------+
| customers            |
| house_info           |
| receipts             |
| students             |
+----------------------+
4 rows in set (0.00 sec)
```

The first steps require us to use the customers and receipts tables. This will begin the introduction to nested queries. The first problem asks us to find out who were the 5 best customers based on the total purchase amount. This means we will need to join the customers and receipts table with the joint “customer_id” key. We want the first name, last name, and the amount that they spent in total, hence the SUM function. We also want to group by the first name, and sort them from highest to lowest, while only displaying the top 5 spenders.



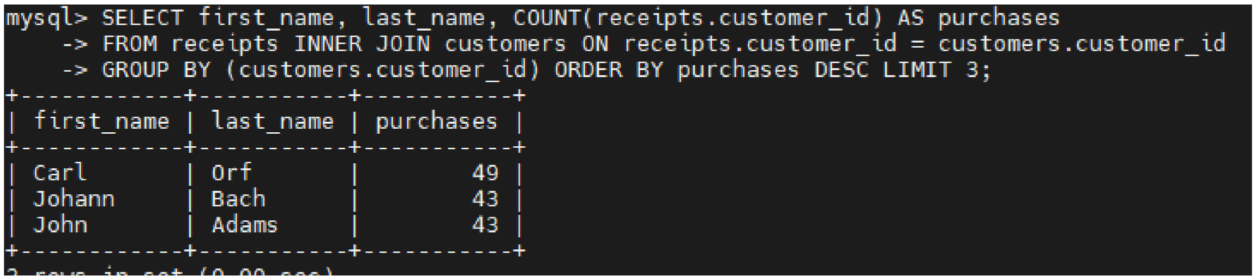
```
mysql> Select first_name, last_name, sum(amount) as total_spent FROM customers
    -> INNER JOIN receipts ON customers.customer_id = receipts.customer_id
    -> GROUP BY first_name ORDER BY amount DESC LIMIT 5;
+------------+---------------+-------------+
| first_name | last_name     | total_spent |
+------------+---------------+-------------+
| Frank      | Ferguson      |     1355.14 |
| Ludwig     | van Beethoven |     1165.99 |
| Frederick  | Delius        |     1096.63 |
| Johann     | Bach          |     1500.92 |
| Roger      | Sessions      |     1213.33 |
+------------+---------------+-------------+
5 rows in set (0.00 sec)
```

The next step wants the 5 biggest spenders based on the average purchase. This means we can use a very similar function with the same setup, but instead of asking for the SUM of all that is spent, we will check for the average amount spent per customer, and then sort them in a descending list showing the top 5 biggest average spenders.



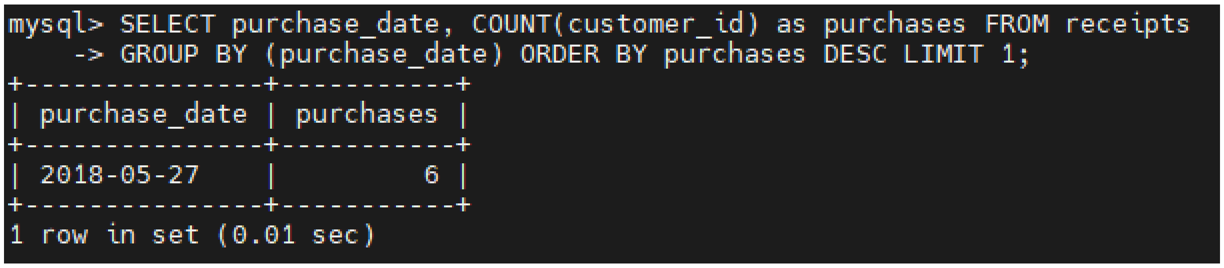
```
mysql> Select first_name, last_name, avg(amount) as average_spent FROM customers
    -> INNER JOIN receipts ON customers.customer_id = receipts.customer_id
    -> GROUP BY first_name ORDER BY amount DESC LIMIT 5;
+------------+---------------+---------------+
| first_name | last_name     | average_spent |
+------------+---------------+---------------+
| Frank      | Ferguson      |     43.714194 |
| Ludwig     | van Beethoven |     46.639600 |
| Frederick  | Delius        |     39.165357 |
| Johann     | Bach          |     34.905116 |
| Roger      | Sessions      |     41.838966 |
+------------+---------------+---------------+
5 rows in set (0.00 sec)
```

For the next step, we want to find out the 3 most frequent shoppers based on the number of purchases made. For this, we need to create a table that has the first names, last names, and number of purchases made by each customer_id. We will join the customers to the receipts table through the customer_id key again so we can use customer names, and this time we want to show the top 3 in a descending list, Car, Johann, and John.



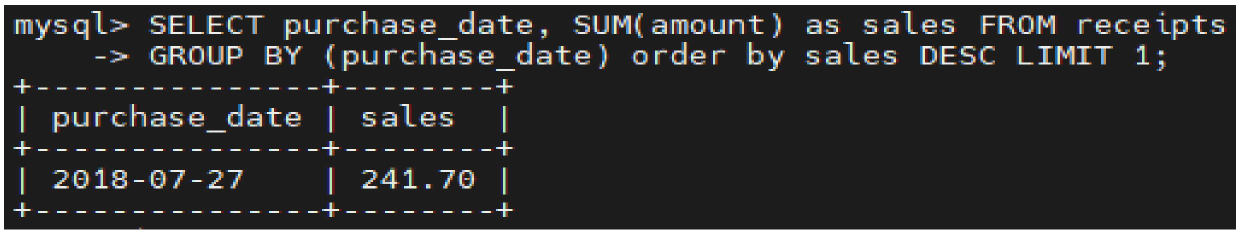
```
mysql> SELECT first_name, last_name, COUNT(receipts.customer_id) AS purchases
    -> FROM receipts INNER JOIN customers ON receipts.customer_id = customers.customer_id
    -> GROUP BY (customers.customer_id) ORDER BY purchases DESC LIMIT 3;
+------------+-----------+-----------+
| first_name | last_name | purchases |
+------------+-----------+-----------+
| Carl       | Orf       |        49 |
| Johann     | Bach      |        43 |
| John       | Adams     |        43 |
+------------+-----------+-----------+
```

Our next task is to find out which day was the busiest, this means we need the day with the highest number of purchases. To do this, we will create a table that shows the purchase_date and a count of every customer_id on that date. This means that we will have a table populated with a date that's associated with the highest count of customer_ids (purchases). Then we will make it a descending list to get the top values and limit it to 1 to get the single most profitable date, 05-27-2018.



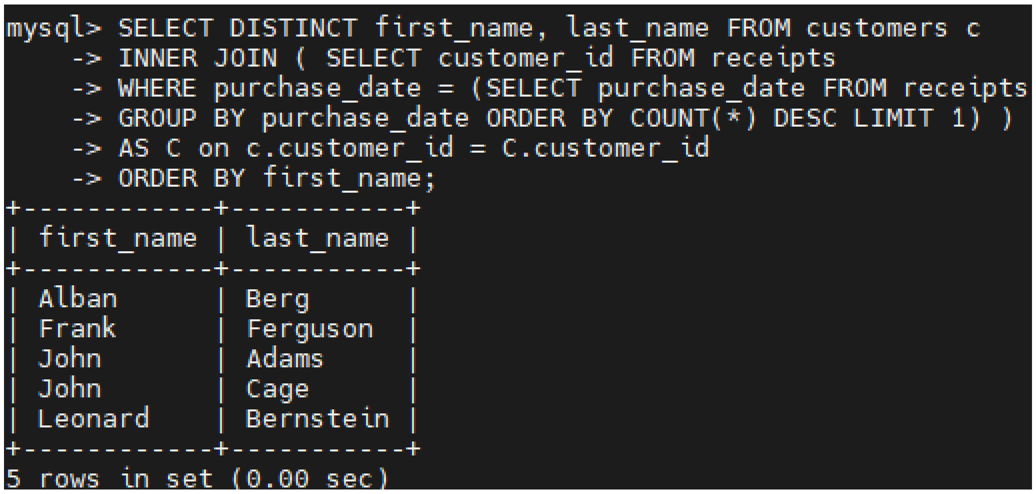
```
mysql> SELECT purchase_date, COUNT(customer_id) as purchases FROM receipts
    -> GROUP BY (purchase_date) ORDER BY purchases DESC LIMIT 1;
+---------------+-----------+
| purchase_date | purchases |
+---------------+-----------+
| 2018-05-27    |         6 |
+---------------+-----------+
1 row in set (0.01 sec)
```

In order to find out the most profitable day, we need to look at the sum of sales rather than the number of sales. We will display a table with a purchase_date, and the associated profit for that day, and then create a descending list with limit 1 to get the highest grossing day, which ends up being 07-27-2018, where the sales were $241.70.



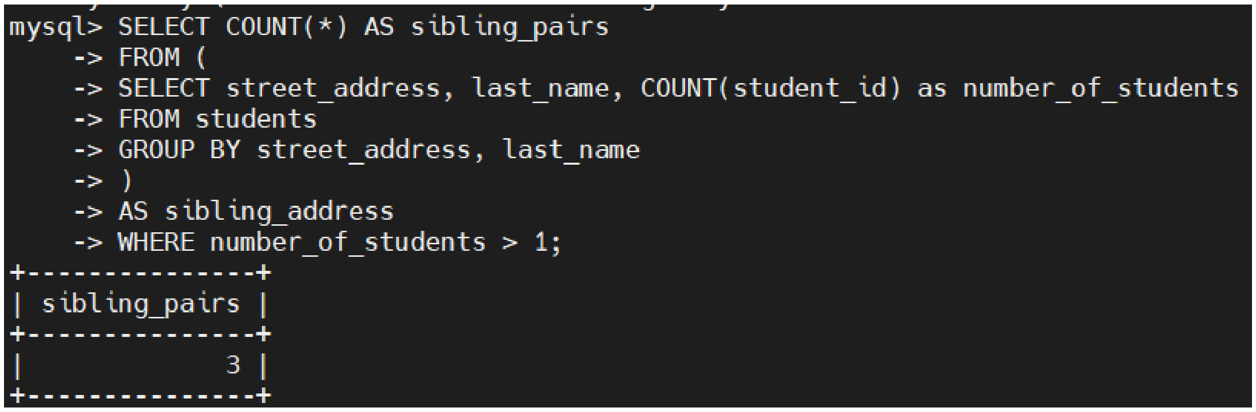
```
mysql> SELECT purchase_date, SUM(amount) as sales FROM receipts
    -> GROUP BY (purchase_date) order by sales DESC LIMIT 1;
+---------------+--------+
| purchase_date | sales  |
+---------------+--------+
| 2018-07-27    | 241.70 |
+---------------+--------+
```

Now, going back to the busiest day, rather than a count of how many purchases were made, we are looking to display all of the customers that made a purchase on that date, and to make sure we don't repeat the names of customers who have already made a purchase we use the DISTINCT clause, which means that the table won't repeat the same key that's already displayed in the table. We then want to use an INNER JOIN on another table we will create, this will help with performance as the INNER JOIN will only select the column required for this query. For this example we want a table that shows first and last name, from the customer table, with the INNER JOIN only pulling over the row where the purchase date has the highest number of purchases. We can use ORDER BY first_name to alphabetize the list.



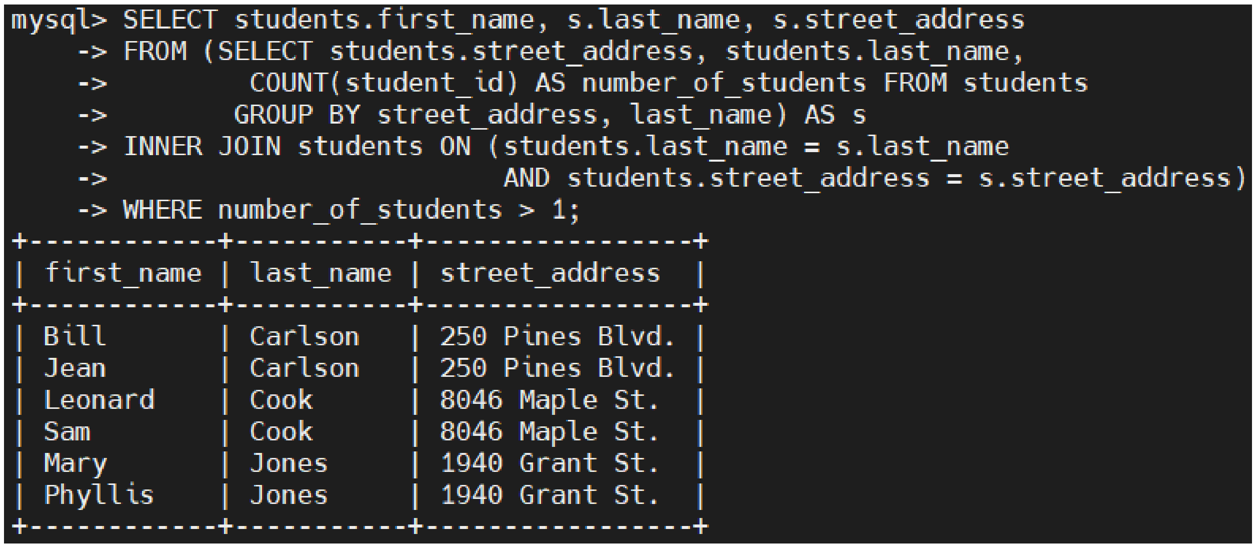
```
mysql> SELECT DISTINCT first_name, last_name FROM customers c
    -> INNER JOIN ( SELECT customer_id FROM receipts
    -> WHERE purchase_date = (SELECT purchase_date FROM receipts
    -> GROUP BY purchase_date ORDER BY COUNT(*) DESC LIMIT 1) )
    -> AS C on c.customer_id = C.customer_id
    -> ORDER BY first_name;
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Alban      | Berg      |
| Frank      | Ferguson  |
| John       | Adams     |
| John       | Cage      |
| Leonard    | Bernstein |
+------------+-----------+
5 rows in set (0.00 sec)
```

For the next section, we are tasked with working with the students table. The first problem asks us to find out how many sibling pairs there are. A sibling pair is a pair of kids who share the same last name and address. For this query, we want to display the count of sibling pairs. We will select from a smaller table that only contains street_address, last_name, and a count of the student IDs in that table and using GROUP BY, we can select the students who share the same address and name. We will also select to only show cases where there are more than 1 person with the same name at a house, so we don't print the values of the students without siblings.



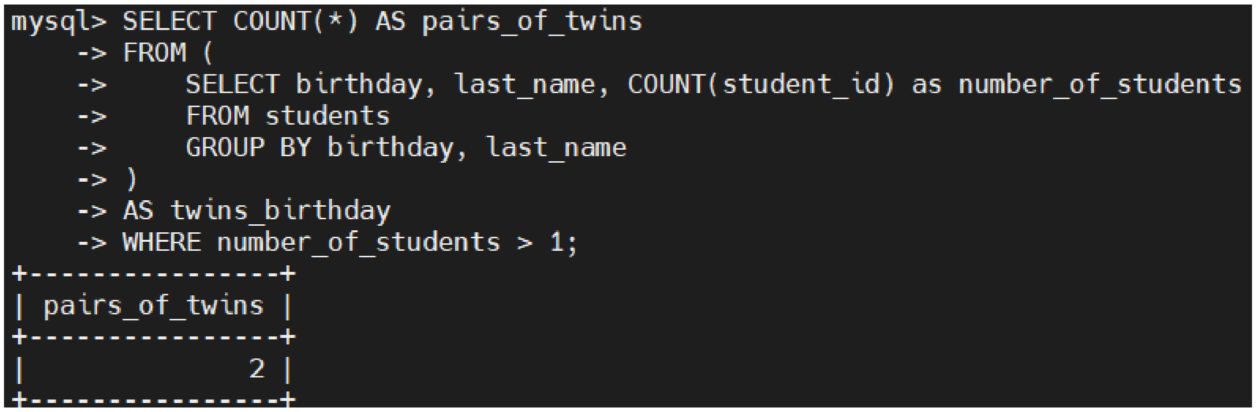
```
mysql> SELECT COUNT(*) AS sibling_pairs
    -> FROM (
    -> SELECT street_address, last_name, COUNT(student_id) as number_of_students
    -> FROM students
    -> GROUP BY street_address, last_name
    -> )
    -> AS sibling_address
    -> WHERE number_of_students > 1;
+---------------+
| sibling_pairs |
+---------------+
|             3 |
+---------------+
```

We are then asked to name all of the sibling pairs found in the last example. First we use a nested query to select only the street_address, last_name, and student_id count columns for our inner query. Next, we will use a self join, where we join the students table to itself where student.last_name = s.last_name and student.street_address = s.street_address so that we can get the names of students who are matching another student's address and last name. Our final query prints the first names, last names, and addresses of everyone who has a sibling.



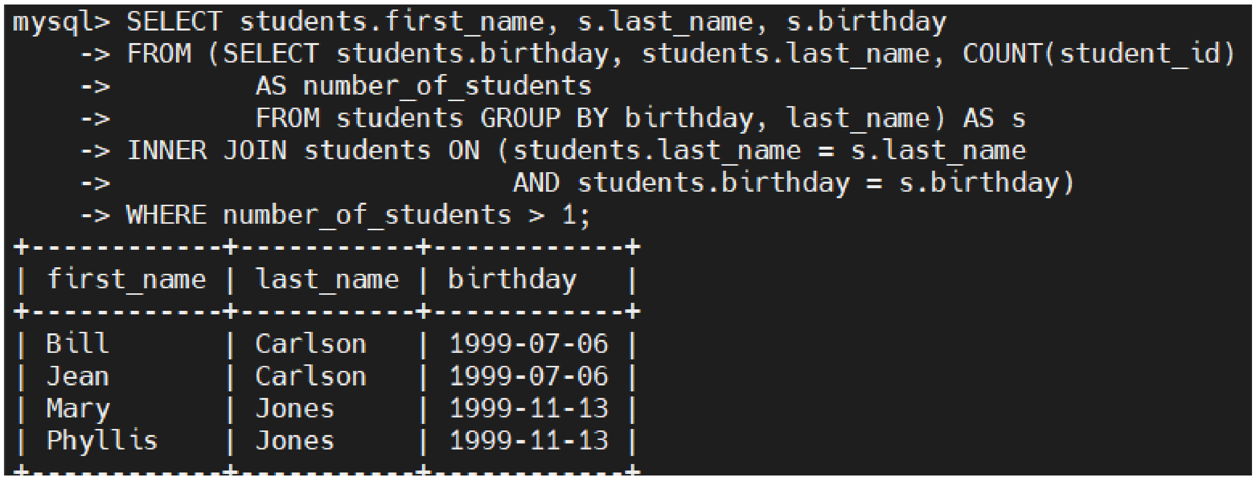
```
mysql> SELECT students.first_name, s.last_name, s.street_address
    -> FROM (SELECT students.street_address, students.last_name,
    ->         COUNT(student_id) AS number_of_students FROM students
    ->        GROUP BY street_address, last_name) AS s
    -> INNER JOIN students ON (students.last_name = s.last_name
    ->                         AND students.street_address = s.street_address)
    -> WHERE number_of_students > 1;
+------------+-----------+-----------------+
| first_name | last_name | street_address  |
+------------+-----------+-----------------+
| Bill       | Carlson   | 250 Pines Blvd. |
| Jean       | Carlson   | 250 Pines Blvd. |
| Leonard    | Cook      | 8046 Maple St.  |
| Sam        | Cook      | 8046 Maple St.  |
| Mary       | Jones     | 1940 Grant St.  |
| Phyllis    | Jones     | 1940 Grant St.  |
+------------+-----------+-----------------+
```

The next query asks us to find the number of twin pairs, to do this we can modify our previous query to find students with the same last_name, and birthday. This is the same query as finding siblings, but modified to find birthday matches as opposed to address matches.



```
mysql> SELECT COUNT(*) AS pairs_of_twins
    -> FROM (
    ->     SELECT birthday, last_name, COUNT(student_id) as number_of_students
    ->     FROM students
    ->     GROUP BY birthday, last_name
    -> )
    -> AS twins_birthday
    -> WHERE number_of_students > 1;
+----------------+
| pairs_of_twins |
+----------------+
|              2 |
+----------------+
```

The last problem asks us to name the twins, this can be done by taking the last query for naming sibling paris, and altering it so that it creates matches for students who share their birthday and last name. We use a similar self join operator and a nested query in the FROM statement to make our inner query table smaller.



```
mysql> SELECT students.first_name, s.last_name, s.birthday
    -> FROM (SELECT students.birthday, students.last_name, COUNT(student_id)
    ->         AS number_of_students
    ->         FROM students GROUP BY birthday, last_name) AS s
    -> INNER JOIN students ON (students.last_name = s.last_name
    ->                         AND students.birthday = s.birthday)
    -> WHERE number_of_students > 1;
+------------+-----------+------------+
| first_name | last_name | birthday   |
+------------+-----------+------------+
| Bill       | Carlson   | 1999-07-06 |
| Jean       | Carlson   | 1999-07-06 |
| Mary       | Jones     | 1999-11-13 |
| Phyllis    | Jones     | 1999-11-13 |
+------------+-----------+------------+
```

The use of nested queries is a good skill to master as it allows you to perform more precise operations on data. While the syntax is easy to remember, the hardest part of using nested queries is putting everything in the correct order so your final table displays the correct information. It's very easy to get a query that compiles fine, but does not display the intended information. I think this remains a powerful tool, but there are some areas of concern with it, as I feel the self join operators are more confusing than helpful most of the time.