This assignment expands upon the MariaDB exercises we did last week, using the same `vets` database..

First, let's get back into my container, and start up MariaDB:

```
📦[b@mariadb-ppc ~]$ sudo service mariadb start
 * Starting MariaDB database server mariadbd                    [ OK ]
📦[b@mariadb-ppc ~]$ mariadb -u bf -p fisher
Enter password:
```

Now that I'm back in, I can start working on the problems in this assignment.

1: Alias column name:

Using `AS` we can rename a column before outputting, allowing us to tweak things to a more descriptive displayed output.

```
MariaDB [fisher]> SELECT COUNT(*) AS RowCount FROM vets;
+----------+
| RowCount |
+----------+
|    58253 |
+----------+
1 row in set (0.018 sec)
```

2. TRIM(state) = state;

Next, we are instructed to run the query `SELECT COUNT(*) FROM vets WHERE TRIM(state) = state;`, then observe and explain the output.

We can see there are 83 rows that match the query:

```
MariaDB [fisher]> SELECT COUNT(*) FROM vets WHERE TRIM(state) = state;
+----------+
| COUNT(*) |
+----------+
|       83 |
+----------+
1 row in set (0.012 sec)
MariaDB [fisher]>
```

To get a better idea of what is contained in these rows, I'm going to tweak the query to display the first five rows matching `TRIM(state) =`

`state` rather than the count.

I believe this should show five rows where the data in the state column is the same before and after trimming, meaning it should contain no whitespace.

```
MariaDB [fisher]> SELECT * FROM vets WHERE TRIM(state) = state
LIMIT 5;
+-------+-------+------+-------+
| lname | fname | town | state |
+-------+-------+------+-------+
|       |       |      |       |
|       |       |      |       |
|       |       |      |       |
|       |       |      |       |
|       |       |      |       |
+-------+-------+------+-------+
5 rows in set (0.003 sec)
```

Interesting, we see only blank entries. From this output, I can come to the conclusion that the 83 rows where TRIM(state) = state are rows where there are no surrounding whitespaces on the entries. I reran the query without `LIMIT 5`, and all of the 83 rows are just empty data.

This means only empty rows have no surrounding white space on the state data.

3. Delete empty rows

The next step of the assignment confirms my conclusion:
The original CSV file had blank lines, so they were read in as empty rows. This is one thing you should
be aware of when reading in CSV files.

Now, we're tasked to use a DELETE statement to clear out all the empty rows in the vets table.

```
MariaDB [fisher]> DELETE FROM vets WHERE TRIM(state) = state;
Query OK, 83 rows affected (0.025 sec)
```

Great! We can see our 83 empty rows have been deleted.

4. Trim whitespace on states

Now that we've deleted any empty rows, we should properly trim all of our values in the state column to remove any extra whitespace.

To do this, we'll use an `UPDATE` statement.

```
MariaDB [fisher]> UPDATE vets
    -> SET state = TRIM(state);
Query OK, 58170 rows affected (0.104 sec)
Rows matched: 58170  Changed: 58170  Warnings: 0
```

Looks good. Let's check our changes:

```
MariaDB [fisher]> SELECT * FROM vets WHERE TRIM(state) = state LIMIT 5;
+---------+--------------------+----------------+-------+
| lname   | fname              | town           | state |
+---------+--------------------+----------------+-------+
| AADLAND |  GERALD L          |  SISSETON      | SD    |
| AALUND  |  JAMES DOWNING     |  HOUSTON       | TX    |
| AAMOLD  |  DANIEL LAWRENCE   |  MOORHEAD      | MN    |
| AARDE   |  JAMES RAYMOND     |  KENT          | WA    |
| AARON   |  CHARLES EDWARD    |  FORGE VILLAGE | MA    |
+---------+--------------------+----------------+-------+
5 rows in set (0.000 sec)
```

Now, the same query that proved we had extra whitespace on all of our state columns is showing us that we've properly trimmed all of the entries of the state column.

5. Trim whitespace on everything else

We've seen how to trim leading and trailing whitespace in one column, now let's do that for each other column. I'll alter the `UPDATE` sequence and re-run for each column.

The following query will test to make sure each of the columns was properly trimmed. We'll also use a `LIKE` statement to display anybody whose first name starts with GUADALUPE, and is followed by anything else.

```
MariaDB [fisher]> SELECT fname as "First name", lname AS "Last Name", town, state FROM vets where fname like 'GUADALUPE%';
+------------------+-----------+------------+-------+
| First name       | Last Name | town       | state |
+------------------+-----------+------------+-------+
| GUADALUPE MASIAS | ALVAREZ   | DONNA      | TX    |
| GUADALUPE        | FLORES    | BEXAR      | TX    |
| GUADALUPE JR     | GALINDO   | ROCKPORT   | TX    |
| GUADALUPE B L    | GARIBAY   | SAN DIEGO  | CA    |
| GUADALUPE        | GONZALEZ  | ALICE      | TX    |
| GUADALUPE MENDOZA | LEAL     | QUITAQUE   | TX    |
```

Looks good!

6. Find towns starting with `LOS`

Our next task is to find how many towns in the data start with `LOS`.

```
MariaDB [fisher]> SELECT COUNT(*) FROM vets WHERE town LIKE 'LOS%';
+----------+
| COUNT(*) |
+----------+
|      586 |
+----------+
1 row in set (0.009 sec)
```

Looks like there are 586 towns in the data that start with `LOS`.

7. How many are NOT `LOS ANGELES`?

To find out how many of the 586 towns are Los Angeles, I'll just run a `SELECT` statement to find where the town matches `LOS ANGELES` exactly, rather than LOS%.

```
MariaDB [fisher]> SELECT COUNT(*) FROM vets WHERE town='LOS ANGELES';
+----------+
| COUNT(*) |
+----------+
|      538 |
+----------+
1 row in set (0.020 sec)
```

We can see 538 of the towns starting with `LOS` are `LOS ANGELES`.
586 - 538 = 48
So, 48 towns are not Los Angeles, and 538 are Los Angeles.

8. Los - Angeles: GROUP BY

We want a count by state of people who are not from Los Angeles, but are from other towns starting with 'LOS'.

First, let's show a count by state of all people from town starting with 'LOS'.

```
MariaDB [fisher]> SELECT state, COUNT(*) FROM vets WHERE town LIKE 'LOS%' GROUP BY state;
+-------+----------+
| state | COUNT(*) |
+-------+----------+
| CA    |      573 |
| IA    |        2 |
| KY    |        1 |
| NM    |        6 |
| TX    |        3 |
| WV    |        1 |
+-------+----------+
6 rows in set (0.024 sec)
```

Now, let's exclude Los Angeles.

```
MariaDB [fisher]> SELECT state, COUNT(*) FROM vets WHERE town LIKE 'LOS%' AND town!='LOS ANGELES'  GROUP BY state;
+-------+----------+
| state | COUNT(*) |
+-------+----------+
| CA    |       35 |
| IA    |        2 |
| KY    |        1 |
| NM    |        6 |
| TX    |        3 |
| WV    |        1 |
+-------+----------+
```

Perfect!

9. List of Vets from GUAM (GU) ordered by last name

Now, we'll use an `ORDER BY` clause to order our output display:

```
MariaDB [fisher]> SELECT * FROM vets WHERE state='GU' ORDER BY lname ASC LIMIT 5;
+-----------+-----------------+----------+-------+
| lname     | fname           | town     | state |
+-----------+-----------------+----------+-------+
| AGUON     | JOSE QUINATA    | UMATAC   | GU    |
| ASANOMA   | FRANCISCO M     | INARAJAN | GU    |
| BENAVENTE | DAVID GUERRERO  | DEDEDO   | GU    |
| BIAGINI   | MARK FREDERICK  | MANGILAO | GU    |
| BLAS      | ANTHONY MARTIN M | YONA     | GU    |
+-----------+-----------------+----------+-------+
```

I've limited the output to 5 rows, but we can see the rows are sorted by last name in ascending order.

10. Create new table States

The last part of this assignment is to create a new table `states`, and read in a CSV file `States.csv` into the table.

When we preview the CVS file, we can see all of the data entries are stuck between quotation marks:

```
"State","Abbreviation"
"Alabama","AL"
"Alaska","AK"
"Arizona","AZ"
"Arkansas","AR"
"California","CA"
"Colorado","CO"
"Connecticut","CT"
```

We don't want that reflected in our new table, so we'll be sure to remove that while reading in the file.

First, let's get a table created with basic attributes `stateName` and `Abbrev` (for abbreviation).

```
MariaDB [fisher]> CREATE TABLE states (
    -> stateName VARCHAR(30),
    -> Abbrev VARCHAR(10);
```

Next, let's read in the `states.csv` file. The `ENCLOSED BY` section states that we should ignore the " " that is surrounding each entry of the CSV file.

```
MariaDB [fisher]> LOAD DATA LOCAL INFILE '/home/b/Downloads/states.csv' into table states
    -> FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';
Query OK, 52 rows affected (0.008 sec)
Records: 52  Deleted: 0  Skipped: 0  Warnings: 0
```

Now that we've read the data in, let's examine:

```
MariaDB [fisher]> SELECT * FROM states LIMIT 5;
+-----------+--------------+
| StateName | Abbrev       |
+-----------+--------------+
| State     | Abbreviation |
| Alabama   | AL           |
| Alaska    | AK           |
| Arizona   | AZ           |
| Arkansas  | AR           |
+-----------+--------------+
5 rows in set (0.000 sec)
```

Looks like we accidentally read in the Header of the CSV file. Let's delete that row.

```
MariaDB [fisher]> DELETE FROM states WHERE StateName='State';
Query OK, 1 row affected (0.008 sec)

MariaDB [fisher]> SELECT * FROM states LIMIT 5;
+------------+--------+
| StateName  | Abbrev |
+------------+--------+
| Alabama    | AL     |
| Alaska     | AK     |
| Arizona    | AZ     |
| Arkansas   | AR     |
| California | CA     |
+------------+--------+
5 rows in set (0.000 sec)
```

After deleting the header row, we can see our state names and their abbreviations have been properly read into the table, and we have finished the assignment.