MySQL_Exercise_12_Queries_that_Test_Relationships_Between_Test_Co

December 26, 2020

1 MySQL Exercise 12: Queries that Test Relationships Between Test Completion and Testing Circumstances

In this lesson, we are going to practice integrating more of the concepts we learned over the past few weeks to address whether issues in our Dognition sPAP are related to the number of tests dogs complete. We are going to focus on a subset of the issues listed in the "Features of Testing Circumstances" branch of our sPAP. You will need to look up new functions several times and the final queries at which we will arrive by the end of this lesson will be quite complex, but we will work up to them step-by-step.

To begin, load the sql library and database, and make the Dognition database your default database:

```
[1]: %load_ext sql
     %sql mysql://studentuser:studentpw@localhost/dognitiondb
     %sql USE dognitiondb
```

```
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

```
[1]: []
```

1.1 1. During which weekdays do Dognition users complete the most tests?

The first question we are going to address is whether there is a certain day of the week when users are more or less likely to complete Dognition tests. If so, targeting promotions or reminder emails to those times of the week might increase the number of tests users complete.

At first, the query we need to address this question might seem a bit intimidating, but once you can describe what the query needs to do in words, writing the query won't seem so challenging.

Ultimately, we want a count of the number of tests completed on each day of the week, with all of the dog_guids and user_guids the Dognition team flagged in their exclude column excluded. To achieve this, we are going to have to use the GROUP BY clause to break up counts of the records in the completed_tests table according to days of the week. We will also have to join the

completed_tests table with the dogs and users table in order to exclude completed_tests records that are associated with dog_guids or user_guids that should be excluded. First, though, we need a method for extracting the day of the week from a time stamp. In MySQL Exercise 2 we used a function called "DAYNAME". That is the most efficient function to use for this purpose, but not all database systems have this function, so let's try using a different method for the queries in this lesson. Search these sites to find a function that will output a number from 1-7 for time stamps where 1 = Sunday, 2 = Monday, ..., 7 = Saturday:

https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html
http://www.w3resource.com/mysql/mysql-functions-and-operators.php

Question 1: Using the function you found in the websites above, write a query that will output one column with the original created_at time stamp from each row in the completed_tests table, and another column with a number that represents the day of the week associated with each of those time stamps. Limit your output to 200 rows starting at row 50.

[5]:
```sql
%%sql
SELECT created_at, DAYOFWEEK(created_at)
FROM complete_tests
LIMIT 49, 3;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
3 rows affected.
```

[5]: [(datetime.datetime(2013, 2, 5, 22, 10, 6), 3),
  (datetime.datetime(2013, 2, 5, 22, 23, 49), 3),
  (datetime.datetime(2013, 2, 5, 22, 26, 36), 3)]

Of course, the results of the query in Question 1 would be much easier to interpret if the output included the name of the day of the week (or a relevant abbreviation) associated with each time stamp rather than a number index.

Question 2: Include a CASE statement in the query you wrote in Question 1 to output a third column that provides the weekday name (or an appropriate abbreviation) associated with each created_at time stamp.

[7]:
```sql
%%sql
SELECT created_at, DAYOFWEEK(created_at),
(CASE
WHEN DAYOFWEEK(created_at)=1 THEN "Su"
WHEN DAYOFWEEK(created_at)=2 THEN "Mo"
WHEN DAYOFWEEK(created_at)=3 THEN "Tu"
WHEN DAYOFWEEK(created_at)=4 THEN "We"
WHEN DAYOFWEEK(created_at)=5 THEN "Th"
WHEN DAYOFWEEK(created_at)=6 THEN "Fr"
WHEN DAYOFWEEK(created_at)=7 THEN "Sa"
END) AS daylabel
FROM complete_tests
```

```
LIMIT 49,3;
```

 * mysql://studentuser:***@localhost/dognitiondb
3 rows affected.

[7]: [(datetime.datetime(2013, 2, 5, 22, 10, 6), 3, 'Tu'),
     (datetime.datetime(2013, 2, 5, 22, 23, 49), 3, 'Tu'),
     (datetime.datetime(2013, 2, 5, 22, 26, 36), 3, 'Tu')]

Now that we are confident we have the correct syntax for extracting weekday labels from the created_at time stamps, we can start building our larger query that examines the number of tests completed on each weekday.

Question 3: Adapt the query you wrote in Question 2 to report the total number of tests completed on each weekday. Sort the results by the total number of tests completed in descending order. You should get a total of 33,190 tests in the Sunday row of your output.

[8]:
```sql
%%sql
SELECT sub.daylabel, count(created_at)
FROM
(SELECT created_at, DAYOFWEEK(created_at),
(CASE
WHEN DAYOFWEEK(created_at)=1 THEN "Su"
WHEN DAYOFWEEK(created_at)=2 THEN "Mo"
WHEN DAYOFWEEK(created_at)=3 THEN "Tu"
WHEN DAYOFWEEK(created_at)=4 THEN "We"
WHEN DAYOFWEEK(created_at)=5 THEN "Th"
WHEN DAYOFWEEK(created_at)=6 THEN "Fr"
WHEN DAYOFWEEK(created_at)=7 THEN "Sa"
END) AS daylabel
FROM complete_tests) AS sub
GROUP BY sub.daylabel
ORDER BY count(created_at) DESC;
```

 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.

[8]: [('Su', 33190),
     ('Mo', 30195),
     ('Tu', 27989),
     ('Sa', 27899),
     ('We', 26473),
     ('Th', 24420),
     ('Fr', 23080)]

So far these results suggest that users complete the most tests on Sunday night and the fewest tests on Friday night. We need to determine if this trend remains after flagged dog_guids and

3

user_guids are excluded. Let's start by removing the dog_guids that have an exclude flag. We'll exclude user_guids with an exclude flag in later queries.

Question 4: Rewrite the query in Question 3 to exclude the dog_guids that have a value of "1" in the exclude column (Hint: this query will require a join.) This time you should get a total of 31,092 tests in the Sunday row of your output.

```sql
[2]: %%sql
SELECT DAYOFWEEK(c.created_at),COUNT(c.created_at) AS numtests,
(CASE
WHEN DAYOFWEEK(c.created_at)=1 THEN "Su"
WHEN DAYOFWEEK(c.created_at)=2 THEN "Mo"
WHEN DAYOFWEEK(c.created_at)=3 THEN "Tu"
WHEN DAYOFWEEK(c.created_at)=4 THEN "We"
WHEN DAYOFWEEK(c.created_at)=5 THEN "Th"
WHEN DAYOFWEEK(c.created_at)=6 THEN "Fr"
WHEN DAYOFWEEK(c.created_at)=7 THEN "Sa"
END) AS daylabel
FROM complete_tests c JOIN dogs d
ON c.dog_guid=d.dog_guid
WHERE d.exclude IS NULL OR d.exclude=0
GROUP BY daylabel
ORDER BY numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.
```

```
[2]: [(1, 31092, 'Su'),
     (2, 28250, 'Mo'),
     (7, 26231, 'Sa'),
     (3, 25764, 'Tu'),
     (4, 24501, 'We'),
     (5, 22347, 'Th'),
     (6, 21028, 'Fr')]
```

Now we need to exclude the user_guids that have a value of "1" in the exclude column as well. One way to do this would be to join the completed_tests, dogs, and users table with a sequence of inner joins. However, we've seen in previous lessons that there are duplicate rows in the users table. These duplicates will get passed through the join and will affect the count calculations. To illustrate this, compare the following two queries.

Question 5: Write a query to retrieve all the dog_guids for users common to the dogs and users table using the traditional inner join syntax (your output will have 950,331 rows).

```sql
[6]: %%sql
SELECT dog_guid
FROM dogs d INNER JOIN users u
ON d.user_guid=u.user_guid
```

```
LIMIT 5;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
5 rows affected.
```

[6]:
```
[('fd27b272-7144-11e5-ba71-058fbc01cf0b',),
 ('fd417cac-7144-11e5-ba71-058fbc01cf0b',),
 ('fd27b5ba-7144-11e5-ba71-058fbc01cf0b',),
 ('fd3fb0f2-7144-11e5-ba71-058fbc01cf0b',),
 ('fd27b6b4-7144-11e5-ba71-058fbc01cf0b',)]
```

Question 6: Write a query to retrieve all the *distinct* dog_guids common to the dogs and users table using the traditional inner join syntax (your output will have 35,048 rows).

[5]:
```
%%sql
SELECT DISTINCT dog_guid
FROM dogs d INNER JOIN users u
ON d.user_guid=u.user_guid
LIMIT 5;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
5 rows affected.
```

[5]:
```
[('fd27b272-7144-11e5-ba71-058fbc01cf0b',),
 ('fd417cac-7144-11e5-ba71-058fbc01cf0b',),
 ('fd27b5ba-7144-11e5-ba71-058fbc01cf0b',),
 ('fd3fb0f2-7144-11e5-ba71-058fbc01cf0b',),
 ('fd27b6b4-7144-11e5-ba71-058fbc01cf0b',)]
```

The strategy we will use to handle duplicate rows in the users table will be to, first, write a subquery that retrieves the distinct dog_guids from an inner join between the dogs and users table with the appropriate records excluded. Then, second, we will join the result of this subquery to the complete_tests table and group the results according to the day of the week.

Question 7: Start by writing a query that retrieves distinct dog_guids common to the dogs and users table, excuding dog_guids and user_guids with a "1" in their respective exclude columns (your output will have 34,121 rows).

[8]:
```
%%sql
SELECT DISTINCT dog_guid
FROM dogs d INNER JOIN users u
ON d.user_guid=u.user_guid
WHERE (u.exclude =0 OR u.exclude IS NULL)
AND (d.exclude =0 OR d.exclude IS NULL)
LIMIT 5
;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
5 rows affected.
```

[8]: [('fd27b272-7144-11e5-ba71-058fbc01cf0b',),
('fd417cac-7144-11e5-ba71-058fbc01cf0b',),
('fd27b5ba-7144-11e5-ba71-058fbc01cf0b',),
('fd3fb0f2-7144-11e5-ba71-058fbc01cf0b',),
('fd27b6b4-7144-11e5-ba71-058fbc01cf0b',)]

Question 8: Now adapt your query from Question 4 so that it inner joins on the result of the subquery you wrote in Question 7 instead of the dogs table. This will give you a count of the number of tests completed on each day of the week, excluding all of the dog_guids and user_guids that the Dognition team flagged in the exclude columns.

[13]:
```sql
%%sql
SELECT DAYOFWEEK(c.created_at),COUNT(c.created_at) AS numtests,
(CASE
WHEN DAYOFWEEK(c.created_at)=1 THEN "Su"
WHEN DAYOFWEEK(c.created_at)=2 THEN "Mo"
WHEN DAYOFWEEK(c.created_at)=3 THEN "Tu"
WHEN DAYOFWEEK(c.created_at)=4 THEN "We"
WHEN DAYOFWEEK(c.created_at)=5 THEN "Th"
WHEN DAYOFWEEK(c.created_at)=6 THEN "Fr"
WHEN DAYOFWEEK(c.created_at)=7 THEN "Sa"
END) AS daylabel
FROM complete_tests c JOIN

(SELECT DISTINCT dog_guid
FROM dogs d INNER JOIN users u
ON d.user_guid=u.user_guid
WHERE (u.exclude =0 OR u.exclude IS NULL)
AND (d.exclude =0 OR d.exclude IS NULL)) AS sub


ON c.dog_guid=sub.dog_guid
GROUP BY daylabel
ORDER BY numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.
```

[13]: [(1, 31036, 'Su'),
(2, 28138, 'Mo'),
(7, 26149, 'Sa'),
(3, 25696, 'Tu'),
(4, 24433, 'We'),
(5, 22323, 'Th'),
(6, 21027, 'Fr')]

These results still suggest that Sunday is the day when the most tests are completed and Friday is the day when the fewest tests are completed. However, our first query suggested that more tests were completed on Tuesday than Saturday; our current query suggests that slightly more tests are completed on Saturday than Tuesday, now that flagged dog_guids and user_guids are excluded.

It's always a good idea to see if a data pattern replicates before you interpret it too strongly. The ideal way to do this would be to have a completely separate and independent data set to analyze. We don't have such a data set, but we can assess the reliability of the day of the week patterns in a different way. We can test whether the day of the week patterns are the same in all years of our data set.

Question 9: Adapt your query from Question 8 to provide a count of the number of tests completed on each weekday of each year in the Dognition data set. Exclude all dog_guids and user_guids with a value of "1" in their exclude columns. Sort the output by year in ascending order, and then by the total number of tests completed in descending order. HINT: you will need a function described in one of these references to retrieve the year of each time stamp in the created_at field:

https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html
http://www.w3resource.com/mysql/mysql-functions-and-operators.php

[16]:
```sql
%%sql
SELECT COUNT(c.created_at) AS numtests, YEAR(c.created_at) AS year,
(CASE
WHEN DAYOFWEEK(c.created_at)=1 THEN "Su"
WHEN DAYOFWEEK(c.created_at)=2 THEN "Mo"
WHEN DAYOFWEEK(c.created_at)=3 THEN "Tu"
WHEN DAYOFWEEK(c.created_at)=4 THEN "We"
WHEN DAYOFWEEK(c.created_at)=5 THEN "Th"
WHEN DAYOFWEEK(c.created_at)=6 THEN "Fr"
WHEN DAYOFWEEK(c.created_at)=7 THEN "Sa"
END) AS daylabel
FROM complete_tests c JOIN

(SELECT DISTINCT dog_guid
FROM dogs d INNER JOIN users u
ON d.user_guid=u.user_guid
WHERE (u.exclude =0 OR u.exclude IS NULL)
AND (d.exclude =0 OR d.exclude IS NULL)) AS sub


ON c.dog_guid=sub.dog_guid
GROUP BY year, daylabel
ORDER BY year ASC, numtests DESC;
```

 * mysql://studentuser:***@localhost/dognitiondb
21 rows affected.

```
[16]: [(8203, 2013, 'Su'),
      (6854, 2013, 'Sa'),
      (5740, 2013, 'Mo'),
      (5665, 2013, 'We'),
      (5393, 2013, 'Tu'),
      (4997, 2013, 'Fr'),
      (4961, 2013, 'Th'),
      (9309, 2014, 'Mo'),
      (9210, 2014, 'Su'),
      (9177, 2014, 'Tu'),
      (8857, 2014, 'We'),
      (8257, 2014, 'Sa'),
      (7286, 2014, 'Th'),
      (6475, 2014, 'Fr'),
      (13623, 2015, 'Su'),
      (13089, 2015, 'Mo'),
      (11126, 2015, 'Tu'),
      (11038, 2015, 'Sa'),
      (10076, 2015, 'Th'),
      (9911, 2015, 'We'),
      (9555, 2015, 'Fr')]
```

These results suggest that although the precise order of the weekdays with the most to fewest
completed tests changes slightly from year to year, Sundays always have a lot of completed tests,
and Fridays always have the fewest or close to the fewest completed tests. So far, it seems like it
might be a good idea for Dognition to target reminder or encouragement messages to customers
on Sundays. However, there is one more issue our analysis does not address. All of the time
stamps in the created_at column are in Coordinated Universal Time (abbreviated UTC). This is
a time convention that is constant around the globe. Nonetheless, as the picture below illustrates,
countries and states have different time zones. The same UTC time can correspond with local
times in different countries that are as much as 24 hours apart:

Therefore, the weekdays we have extracted so far may not accurately reflect the weekdays in the
local times of different countries. The only way to correct the time stamps for time zone differences
is to obtain a table with the time zones of every city, state, or country. Such a table was not available
to us in this course, but we can run some analyses that approximate a time zone correction for
United States customers.

**Question 10: First, adapt your query from Question 9 so that you only examine
customers located in the United States, with Hawaii and Alaska residents excluded.
HINTS: In this data set, the abbreviation for the United States is "US", the abbre-
viation for Hawaii is "HI" and the abbreviation for Alaska is "AK". You should have
5,860 tests completed on Sunday of 2013.**

```
[18]: %%sql
      SELECT COUNT(c.created_at) AS numtests, YEAR(c.created_at) AS year,
      (CASE
      WHEN DAYOFWEEK(c.created_at)=1 THEN "Su"
```

```
WHEN DAYOFWEEK(c.created_at)=2 THEN "Mo"
WHEN DAYOFWEEK(c.created_at)=3 THEN "Tu"
WHEN DAYOFWEEK(c.created_at)=4 THEN "We"
WHEN DAYOFWEEK(c.created_at)=5 THEN "Th"
WHEN DAYOFWEEK(c.created_at)=6 THEN "Fr"
WHEN DAYOFWEEK(c.created_at)=7 THEN "Sa"
END) AS daylabel
FROM complete_tests c JOIN

(SELECT DISTINCT dog_guid
FROM dogs d INNER JOIN users u
ON d.user_guid=u.user_guid
WHERE (u.exclude =0 OR u.exclude IS NULL)
AND (d.exclude =0 OR d.exclude IS NULL)
AND country = 'US'
AND  (u.state!="HI" AND u.state!="AK")) AS sub


ON c.dog_guid=sub.dog_guid
GROUP BY year, daylabel
ORDER BY year ASC, numtests DESC
LIMIT 7;
```

 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.

[18]: [(5860, 2013, 'Su'),
       (4674, 2013, 'Sa'),
       (3695, 2013, 'Mo'),
       (3496, 2013, 'We'),
       (3449, 2013, 'Tu'),
       (3163, 2013, 'Fr'),
       (3090, 2013, 'Th')]

The next step is to adjust the created_at times for differences in time zone. Most United States states (excluding Hawaii and Alaska) have a time zone of UTC time -5 hours (in the eastern-most regions) to -8 hours (in the western-most regions). To get a general idea for how much our weekday analysis is likely to change based on time zone, we will subtract 6 hours from every time stamp in the complete_tests table. Although this means our time stamps can be inaccurate by 1 or 2 hours, people are not likely to be playing Dognition games at midnight, so 1-2 hours should not affect the weekdays extracted from each time stamp too much.

The functions used to subtract time differ across database systems, so you should double-check which function you need to use every time you are working with a new database. We will use the date_sub function:

https://www.w3schools.com/sql/func_mysql_date_sub.asp

Question 11: Write a query that extracts the original created_at time stamps for rows

in the complete_tests table in one column, and the created_at time stamps with 6 hours subtracted in another column. Limit your output to 100 rows.

[19]:
```sql
%%sql
SELECT created_at, DATE_SUB(created_at, interval 6 hour) AS corrected_time
FROM complete_tests
LIMIT 5;
```

 * mysql://studentuser:***@localhost/dognitiondb
5 rows affected.

[19]: [(datetime.datetime(2013, 2, 5, 18, 26, 54), datetime.datetime(2013, 2, 5, 12, 26, 54)),
 (datetime.datetime(2013, 2, 5, 18, 31, 3), datetime.datetime(2013, 2, 5, 12, 31, 3)),
 (datetime.datetime(2013, 2, 5, 18, 32, 4), datetime.datetime(2013, 2, 5, 12, 32, 4)),
 (datetime.datetime(2013, 2, 5, 18, 32, 25), datetime.datetime(2013, 2, 5, 12, 32, 25)),
 (datetime.datetime(2013, 2, 5, 18, 32, 56), datetime.datetime(2013, 2, 5, 12, 32, 56))]

Question 12: Use your query from Question 11 to adapt your query from Question 10 in order to provide a count of the number of tests completed on each day of the week, with approximate time zones taken into account, in each year in the Dognition data set. Exclude all dog_guids and user_guids with a value of "1" in their exclude columns. Sort the output by year in ascending order, and then by the total number of tests completed in descending order. HINT: Don't forget to adjust for the time zone in your DAYOFWEEK statement and your CASE statement.

[22]:
```sql
%%sql
SELECT COUNT(c.created_at) AS numtests, YEAR(c.created_at) AS year,
(CASE
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=1 THEN "Su"
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=2 THEN "Mo"
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=3 THEN "Tu"
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=4 THEN "We"
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=5 THEN "Th"
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=6 THEN "Fr"
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=7 THEN "Sa"
END) AS daylabel
FROM complete_tests c JOIN

(SELECT DISTINCT dog_guid
FROM dogs d INNER JOIN users u
ON d.user_guid=u.user_guid
WHERE (u.exclude =0 OR u.exclude IS NULL)
AND (d.exclude =0 OR d.exclude IS NULL)
```

```
AND country = 'US'
AND  (u.state!="HI" AND u.state!="AK")) AS sub


ON c.dog_guid=sub.dog_guid
GROUP BY year, daylabel
ORDER BY year ASC, numtests DESC
LIMIT 7;
```

 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.

[22]: [(6061, 2013, 'Su'),
      (4754, 2013, 'Sa'),
      (3798, 2013, 'Mo'),
      (3410, 2013, 'We'),
      (3276, 2013, 'Tu'),
      (3079, 2013, 'Th'),
      (3049, 2013, 'Fr')]

You can try re-running the query with time-zone corrections of 5, 7, or 8 hours, and the results remain essentially the same. All of these analyses suggest that customers are most likely to complete tests around Sunday and Monday, and least likely to complete tests around the end of the work week, on Thursday and Friday. This is certainly valuable information for Dognition to take advantage of.

If you were presenting this information to the Dognition team, you might want to present the information in the form of a graph that you make in another program. The graph would be easier to read if the output was ordered according to the days of the week shown in standard calendars, with Monday being the first day and Sunday being the last day. MySQL provides an easy way to do this using the FIELD function in the ORDER BY statement:

https://www.virendrachandak.com/techtalk/mysql-ordering-results-by-specific-field-values/

Question 13: Adapt your query from Question 12 so that the results are sorted by year in ascending order, and then by the day of the week in the following order: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday.

[24]:
```sql
%%sql
SELECT COUNT(c.created_at) AS numtests, YEAR(c.created_at) AS year,
(CASE
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=1 THEN "Su"
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=2 THEN "Mo"
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=3 THEN "Tu"
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=4 THEN "We"
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=5 THEN "Th"
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=6 THEN "Fr"
WHEN DAYOFWEEK(DATE_SUB(c.created_at, interval 6 hour))=7 THEN "Sa"
END) AS daylabel
FROM complete_tests c JOIN
```

```
(SELECT DISTINCT dog_guid
FROM dogs d INNER JOIN users u
ON d.user_guid=u.user_guid
WHERE (u.exclude =0 OR u.exclude IS NULL)
AND (d.exclude =0 OR d.exclude IS NULL)
AND country = 'US'
AND  (u.state!="HI" AND u.state!="AK")) AS sub


ON c.dog_guid=sub.dog_guid
GROUP BY year, daylabel
ORDER BY  year ASC, FIELD(daylabel,'Mo','Tu','We','Th','Fr','Sa','Su')
limit 7;
```

 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.

[24]: [(3798, 2013, 'Mo'),
      (3276, 2013, 'Tu'),
      (3410, 2013, 'We'),
      (3079, 2013, 'Th'),
      (3049, 2013, 'Fr'),
      (4754, 2013, 'Sa'),
      (6061, 2013, 'Su')]

Unfortunately other database platforms do not have the ORDER BY FIELD functionality. To achieve the same result in other platforms, you would have to use a CASE statement or a more advanced solution:

http://stackoverflow.com/questions/1309624/simulating-mysqls-order-by-field-in-postgresql

The link provided above is to a discussion on stackoverflow.com. Stackoverflow is a great website that, in their words, "is a community of 4.7 million programmers, just like you, helping each other." You can ask questions about SQL queries and get help from other experts, or search through questions posted previously to see if somebody else has already asked a question that is relevant to the problem you are trying to solve. It's a great resource to use whenever you run into trouble with your queries.

1.2   2. Which states and countries have the most Dognition users?

You ended up with a pretty long and complex query in the questions above that you tested step-by-step. Many people save these types of queries so that they can be adapted for similar queries in the future without having to redesign and retest the entire query.

In the next two questions, we will practice repurposing previously-designed queries for new questions. Both questions can be answered through relatively minor modifications of the queries you wrote above.

Question 14: Which 5 states within the United States have the most Dognition customers, once all dog_guids and user_guids with a value of "1" in their exclude columns are removed? Try using the following general strategy: count how many unique user_guids are associated with dogs in the complete_tests table, break up the counts according to state, sort the results by counts of unique user_guids in descending order, and then limit your output to 5 rows. California ("CA") and New York ("NY") should be at the top of your list.

```
[38]: %%sql
SELECT sub.state, count(distinct sub.user_guid)
FROM complete_tests c JOIN
(SELECT distinct u.user_guid, d.dog_guid, u.state
FROM users u, dogs d
WHERE d.user_guid=u.user_guid
AND (u.exclude =0 OR u.exclude IS NULL)
AND (d.exclude =0 OR d.exclude IS NULL)
AND country = 'US') AS sub
ON c.dog_guid = sub.dog_guid
GROUP BY sub.state
ORDER BY count(distinct sub.user_guid) DESC
LIMIT 5
;
```

 * mysql://studentuser:***@localhost/dognitiondb
5 rows affected.

[38]: [('CA', 1363), ('NY', 628), ('TX', 536), ('FL', 502), ('NC', 467)]

The number of unique Dognition users in California is more than two times greater than any other state. This information could be very helpful to Dognition. Useful follow-up questions would be: were special promotions run in California that weren't run in other states? Did Dognition use advertising channels that are particularly effective in California? If not, what traits differentiate California users from other users? Can these traits be taken advantage of in future marketing efforts or product developments?

Let's try one more analysis that examines testing circumstances from a different angle.

Question 15: Which 10 countries have the most Dognition customers, once all dog_guids and user_guids with a value of "1" in their exclude columns are removed? HINT: don't forget to remove the u.country="US" statement from your WHERE clause.

```
[40]: %%sql
SELECT sub.country, count(distinct sub.user_guid)
FROM complete_tests c JOIN
(SELECT distinct u.user_guid, d.dog_guid, u.country
FROM users u, dogs d
WHERE d.user_guid=u.user_guid
AND (u.exclude =0 OR u.exclude IS NULL)
```

```
AND (d.exclude =0 OR d.exclude IS NULL)) AS sub
ON c.dog_guid = sub.dog_guid
GROUP BY sub.country
ORDER BY count(distinct sub.user_guid) DESC
LIMIT 10
;
```

 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.

[40]: [('US', 8936),
       ('N/A', 5466),
       ('CA', 484),
       ('AU', 142),
       ('GB', 123),
       ('DE', 40),
       ('NZ', 38),
       ('DK', 34),
       ('NO', 30),
       ('FR', 23)]

The United States, Canada, Australia, and Great Britain are the countries with the most Dognition users. N/A refers to "not applicable" which essentially means we have no usable country data from those rows. After Great Britain, the number of Dognition users drops quite a lot. This analysis suggests that Dognition is most likely to be used by English-speaking countries. One question Dognition might want to consider is whether there are any countries whose participation would dramatically increase if a translated website were available.

1.3 3. Congratulations!

You have now written many complex queries on your own that address real analysis questions about a real business problem. You know how to look up new functions, you know how to troubleshoot your queries by isolating each piece of the query until you are sure the syntax is correct, and you know where to look for help if you get stuck. You are ready to start using SQL in your own business ventures. Keep learning, keep trying new things, and keep asking questions. Congratulations for taking your career to the next level!

There is another video to watch, and of course, more exercises to work through using the Dillard's data set.

In the meantime, enjoy practicing any other queries you want to try here:

[ ]: