MySQL_Exercise_02_Selecting_Data_Subsets_using_WHERE

January 18, 2021

1 MySQL Exercise 2: Using WHERE to select specific data

When you are querying a business-related data set, you are usually doing so to answer a question about a subset of the data. In this lesson you will learn how to select subsets of rows of data that meet criteria you specify, to help you prepare for these types of business questions. The mechanism within a SQL query that allows you specify which subset of data you want to retrieve is the WHERE clause.

Before we begin, let's load the SQL library and Dognition database, and make the Dognition database our default database. As a reminder, these are the lines of code you should input:

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

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

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

[1]: []

Recall the general syntax structure we learned from the "Introduction to Query Syntax" video at the beginning of the week:

This guide indicates that whenever the data we select need to meet certain criteria (specified using a "WHERE" clause), we specify those criteria after we have specified where the data come from.

Let's say we want to know which Dognition customers received access to Dognition's first four tests for free. These customers have a 1 in the "free_start_user" column of the users table. The syntax you would use to select the data for these customers would be:

```
SELECT user_guid
FROM users
WHERE free_start_user=1;
```

(Note: user_guid is the field that specifies the unique User ID number of each customer in the users table)

If you wanted to double-check that the outputted data indeed met the criteria you specified, you could include a second column in your output that would give you the value in the free_start_user field for each row of the output:

```
SELECT user_guid, free_start_user
FROM users
WHERE free_start_user=1;
```

Try this on your own below. Remember to use %%sql to indicate that your query will span multiple lines, and consider whether you would like to limit the number of results you ouput using the syntax we learned last lesson. If you do use a LIMIT statement, remember that it has to be the last item in your query, so this time you will place it after your WHERE statement instead of after your FROM statement.

[2]:
```
%%sql
SELECT user_guid, free_start_user
FROM users
WHERE free_start_user=1
LIMIT 5;
```

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

[2]:
```
[('ce28a468-7144-11e5-ba71-058fbc01cf0b', 1),
 ('ce28ac4c-7144-11e5-ba71-058fbc01cf0b', 1),
 ('ce28acba-7144-11e5-ba71-058fbc01cf0b', 1),
 ('ce28ad1e-7144-11e5-ba71-058fbc01cf0b', 1),
 ('ce28ad82-7144-11e5-ba71-058fbc01cf0b', 1)]
```

Question 1: How would you select the Dog IDs for the dogs in the Dognition data set that were DNA tested (these should have a 1 in the dna_tested field of the dogs table)? Try it below (if you do not limit your output, your query should output data from 1433 dogs):

[3]:
```
%%sql
SELECT dog_guid, dna_tested
FROM dogs
WHERE dna_tested=1
LIMIT 5;
```

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

[3]:
```
[('fd27b6b4-7144-11e5-ba71-058fbc01cf0b', 1),
 ('fd27cd98-7144-11e5-ba71-058fbc01cf0b', 1),
 ('fd27ce1a-7144-11e5-ba71-058fbc01cf0b', 1),
 ('fd27d144-7144-11e5-ba71-058fbc01cf0b', 1),
```

```
('fd27d1c6-7144-11e5-ba71-058fbc01cf0b', 1)]
```

The SELECT statement can be used to interact with all data types, and there are many operators and functions that allow you to interact with the data in different ways. Here are some resources that describe these operators and functions:

http://dev.mysql.com/doc/refman/5.7/en/func-op-summary-ref.html
http://www.w3resource.com/mysql/mysql-functions-and-operators.php

Some of the most common operators include: =,<,>,<=, and >=. If you want to select something that is NOT a specific value, use != or <>. You can also use logical operators, such as AND and OR.

Let's start by examining how operators can be used with numerical data.

If you wanted to examine the Dog IDs of dogs who weighed between 10 and 50 pounds, you could query:

```
SELECT dog_guid, weight
FROM dogs
WHERE weight BETWEEN 10 AND 50;
```

The above query provided an example of how to use the BETWEEN operator (described in the links provided above), as well as an example of how AND can be used to specify multiple criteria. If you wanted to examine the Dog IDs of dogs who were "fixed" (neutered) OR DNA tested, you could use OR in the following query:

```
SELECT dog_guid, dog_fixed, dna_tested
FROM dogs
WHERE dog_fixed=1 OR dna_tested=1;
```

If you wanted to examine the Dog IDs of dogs who were fixed but NOT DNA tested, you could query:

```
SELECT dog_guid, dog_fixed, dna_tested
FROM dogs
WHERE dog_fixed=1 AND dna_tested!=1;
```

Question 2: How would you query the User IDs of customers who bought annual subscriptions, indicated by a "2" in the membership_type field of the users table? (If you do not limit the output of this query, your output should contain 4919 rows.)

[4]:
```
%%sql
SELECT user_guid, membership_type
FROM users
WHERE membership_type=2
LIMIT 5;
```

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

```
[4]: [('ce134e42-7144-11e5-ba71-058fbc01cf0b', 2),
      ('ce135e14-7144-11e5-ba71-058fbc01cf0b', 2),
      ('ce135e14-7144-11e5-ba71-058fbc01cf0b', 2),
      ('ce136ac6-7144-11e5-ba71-058fbc01cf0b', 2),
      ('ce136c24-7144-11e5-ba71-058fbc01cf0b', 2)]
```

Now let's try using the WHERE statement to interact with text data (called "strings").

Strings need to be surrounded by quotation marks in SQL. MySQL accepts both double and single quotation marks, but some database systems only accept single quotation marks. Whenever a string contains an SQL keyword, the string must be enclosed in backticks instead of quotation marks.

```
'the marks that surrounds this phrase are single quotation marks'
"the marks that surrounds this phrase are double quotation marks"
`the marks that surround this phrase are backticks`
```

Strings enclosed in quotation or backticks can be used with many of the same operators as numerical data. For example, imagine that you only wanted to look at data from dogs of the breed "Golden Retrievers." You could query (note that double quotation marks could have been used in this example is well):

```
SELECT dog_guid, breed
FROM dogs
WHERE breed='golden retriever';
```

The IN operator allows you to specify multiple values in a WHERE clause. Each of these values must be separated by a comma from the other values, and the entire list of values should be enclosed in parentheses. If you wanted to look at all the data from Golden Retrievers and Poodles, you could certainly use the OR operator, but the IN operator would be even more efficient (note that single quotation marks could have been used in this example, too):

```
SELECT dog_guid, breed
FROM dogs
WHERE breed IN ("golden retriever","poodle");
```

The LIKE operator allows you to specify a pattern that the textual data you query has to match. For example, if you wanted to look at all the data from breeds whose names started with "s", you could query:

```
SELECT dog_guid, breed
FROM dogs
WHERE breed LIKE ("s%");
```

In this syntax, the percent sign indicates a wild card. Wild cards represent unlimited numbers of missing letters. This is how the placement of the percent sign would affect the results of the query:

- WHERE breed LIKE ("s%") = the breed must start with "s", but can have any number of letters after the "s"
- WHERE breed LIKE ("%s") = the breed must end with "s", but can have any number of letters before the "s"

- WHERE breed LIKE ("%s%") = the breed must contain an "s" somewhere in its name, but can have any number of letters before or after the "s"

Question 3: How would you query all the data from customers located in the state of North Carolina (abbreviated "NC") or New York (abbreviated "NY")? If you do not limit the output of this query, your output should contain 1333 rows.

[18]:
```
%%sql
SELECT *
FROM users
WHERE state IN ("NC","NY")
LIMIT 5;
```

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

[18]:
```
[(181, datetime.datetime(2013, 2, 5, 17, 54, 42), datetime.datetime(2015, 1, 28,
20, 51, 49), 13, 2, 1, 1, 0, None, 2, 'ce135e14-7144-11e5-ba71-058fbc01cf0b',
'Raleigh', 'NC', '27606', 'US', '-5'),
 (181, datetime.datetime(2013, 2, 5, 17, 54, 42), datetime.datetime(2015, 1, 28,
20, 51, 49), 13, 2, 1, 1, 0, None, 2, 'ce135e14-7144-11e5-ba71-058fbc01cf0b',
'Raleigh', 'NC', '27606', 'US', '-5'),
 (65, datetime.datetime(2013, 2, 5, 0, 52, 16), datetime.datetime(2015, 1, 28,
20, 51, 49), 3, 2, 1, None, None, None, 2,
'ce134be0-7144-11e5-ba71-058fbc01cf0b', 'Hillsborough', 'NC', '27278', 'US',
'-5'),
 (7, datetime.datetime(2013, 2, 6, 0, 40, 59), datetime.datetime(2015, 1, 28,
20, 51, 50), 1, 2, 1, None, None, None, 2,
'ce1371a6-7144-11e5-ba71-058fbc01cf0b', 'New York', 'NY', '10023', 'US', '-5'),
 (15, datetime.datetime(2013, 2, 6, 14, 13, 42), datetime.datetime(2015, 1, 28,
20, 51, 50), 1, 2, 1, None, None, None, 2,
'ce137c78-7144-11e5-ba71-058fbc01cf0b', 'Durham', 'NC', '27713', 'US', '-5')]
```

Next, let's try using the WHERE statement to interact with datetime data. Time-related data is a little more complicated to work with than other types of data, because it must have a very specific format. MySQL comes with the following data types for storing a date or a date/time value in the database:

DATE - format YYYY-MM-DD
DATETIME - format: YYYY-MM-DD HH:MI:SS
TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
YEAR - format YYYY or YY

One of the interesting things about time-related data is that SQL has commands to break the data into different "time parts" or "date parts" as described here:

http://www.tutorialspoint.com/mysql/mysql-date-time-functions.htm

A time stamp stored in one row of data might look like this:

`2013-02-07 02:50:52`

The year part of that entry would be 2013, the month part would be "02" or "February" (depending on the requested format), the seconds part would be "52", and so on. SQL functions easily allow you to convert those parts into formats you might need for specific analyses. For example, imagine you wanted to know how many tests Dognition customers complete on different days of the week. To complete this analysis, you would need to convert the time stamps of each completed test to a variable that outputted the correct day of the week for that date. DAYNAME is a function that will do this for you. You can combine DAYNAME with WHERE to select data from only a single day of the week:

```
SELECT dog_guid, created_at
FROM complete_tests
WHERE DAYNAME(created_at)="Tuesday"
```

You can also use common operators like =,<,>,<=,>=,!=, or <> with dates just like you would with other types of data, but whether you refer to the date as a number or text will depend on whether you are selecting individual date parts or treating the date/time entry as a single clause. For example, you could select all the Dog IDs and time stamps of tests completed after the 15 of every month with this command that extracts the "DAY" date part out of each time stamp:

```
SELECT dog_guid, created_at
FROM complete_tests
WHERE DAY(created_at) > 15
```

You could also select all the Dog IDs and time stamps of completed tests from after February 4, 2014 by treating date entries as text clauses with the following query:

```
SELECT dog_guid, created_at
FROM complete_tests
WHERE created_at > '2014-02-04'
```

Note that you have to use a different set of functions than you would use for regular numerical data to add or subtract time from any values in these datetime formats. For example, instead of using a minus sign to find the difference in time between two time stamps or dates, you would use the TIMEDIFF or DATEDIFF function. See the references provided above for a list of these functions.

Question 4: Now that you have seen how datetime data can be used to impose criteria on the data you select, how would you select all the Dog IDs and time stamps of Dognition tests completed before October 15, 2015 (your output should have 193,246 rows)?

[13]:
```
%%sql
SELECT dog_guid, created_at
FROM complete_tests
WHERE created_at < '2015-10-15'
LIMIT 5;
```

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

[13]:
```
[('fd27b86c-7144-11e5-ba71-058fbc01cf0b', datetime.datetime(2013, 2, 5, 18, 26,
54)),
```

```
 ('fd27b86c-7144-11e5-ba71-058fbc01cf0b', datetime.datetime(2013, 2, 5, 18, 31,
3)),
 ('fd27b86c-7144-11e5-ba71-058fbc01cf0b', datetime.datetime(2013, 2, 5, 18, 32,
4)),
 ('fd27b86c-7144-11e5-ba71-058fbc01cf0b', datetime.datetime(2013, 2, 5, 18, 32,
25)),
 ('fd27b86c-7144-11e5-ba71-058fbc01cf0b', datetime.datetime(2013, 2, 5, 18, 32,
56))]
```

Last, let's use the WHERE statement in combination with two very important operators: IS NULL and IS NOT NULL. IS NULL will indicate rows of data that have null values. IS NOT NULL will indicate rows that do not have null values. We saw in previous exercises that many of the entries in the free_start_user field of the user table in the Dognition data set had NULL values. To select only the rows that have non-null data you could query:

```
SELECT user_guid
FROM users
WHERE free_start_user IS NOT NULL;
```

To select only the rows that have null data so that you can examine if these rows share something else in common, you could query:

```
SELECT user_guid
FROM users
WHERE free_start_user IS NULL;
```

Question 5: How would you select all the User IDs of customers who do not have null values in the State field of their demographic information (if you do not limit the output, you should get 17,985 from this query – there are a lot of null values in the state field!)?

[16]:
```
%%sql
SELECT user_guid, state
FROM users
WHERE state IS NOT NULL
LIMIT 5;
```

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

[16]:
```
[('ce134e42-7144-11e5-ba71-058fbc01cf0b', 'ND'),
 ('ce1353d8-7144-11e5-ba71-058fbc01cf0b', 'MA'),
 ('ce135ab8-7144-11e5-ba71-058fbc01cf0b', 'CT'),
 ('ce13507c-7144-11e5-ba71-058fbc01cf0b', 'IL'),
 ('ce135e14-7144-11e5-ba71-058fbc01cf0b', 'NC')]
```

1.1 Practice writing your own SELECT and WHERE statements!

1.1.1 These queries will combine what you've learned in the past two lessons.

Question 6: How would you retrieve the Dog ID, subcategory_name, and test_name fields, in that order, of the first 10 reviews entered in the Reviews table to be submitted in 2014?

[2]:
```
%%sql
SELECT dog_guid, subcategory_name, test_name
FROM reviews
WHERE YEAR(created_at) = '2014'
LIMIT 10;
```

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

[2]:
```
[('ce3ac77e-7144-11e5-ba71-058fbc01cf0b', 'Empathy', 'Yawn Warm-up'),
 ('ce2aedcc-7144-11e5-ba71-058fbc01cf0b', 'Empathy', 'Eye Contact Warm-up'),
 ('ce2aedcc-7144-11e5-ba71-058fbc01cf0b', 'Empathy', 'Eye Contact Game'),
 ('ce2aedcc-7144-11e5-ba71-058fbc01cf0b', 'Communication', 'Treat Warm-up'),
 ('ce405c52-7144-11e5-ba71-058fbc01cf0b', 'Empathy', 'Yawn Warm-up'),
 ('ce405c52-7144-11e5-ba71-058fbc01cf0b', 'Empathy', 'Yawn Game'),
 ('ce405c52-7144-11e5-ba71-058fbc01cf0b', 'Empathy', 'Eye Contact Game'),
 ('ce405e28-7144-11e5-ba71-058fbc01cf0b', 'Communication', 'Treat Warm-up'),
 ('ce405e28-7144-11e5-ba71-058fbc01cf0b', 'Cunning', 'Turn Your Back'),
 ('ce2609c4-7144-11e5-ba71-058fbc01cf0b', 'Communication', 'Treat Warm-up')]
```

** Question 7: How would you select all of the User IDs of customers who have female dogs whose breed includes the word "terrier" somewhere in its name (if you don't limit your output, you should have 1771 rows in your output)? **

[34]:
```
%%sql
SELECT user_guid, gender, breed
FROM dogs
WHERE gender="female" AND breed LIKE ("%terrier%")
LIMIT 5;
```

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

[34]:
```
[('ce138722-7144-11e5-ba71-058fbc01cf0b', 'female', 'Australian Terrier'),
 ('ce13b152-7144-11e5-ba71-058fbc01cf0b', 'female', 'Bedlington Terrier'),
 ('ce21d7d2-7144-11e5-ba71-058fbc01cf0b', 'female', 'Russell Terrier'),
 ('ce2202e8-7144-11e5-ba71-058fbc01cf0b', 'female', 'Boston Terrier-Chihuahua
Mix'),
 ('ce2203f6-7144-11e5-ba71-058fbc01cf0b', 'female', 'American Pit Bull
Terrier')]
```

Question 8: How would you select the Dog ID, test name, and subcategory associated with each completed test for the first 100 tests entered in October, 2014?

[4]:
```
%%sql
SELECT dog_guid, test_name, subcategory_name
FROM complete_tests
WHERE YEAR(created_at)="2014" AND MONTH(created_at)="10"
LIMIT 10;
```

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

[4]:
```
[('fd6a3480-7144-11e5-ba71-058fbc01cf0b', 'Delayed Cup Game', 'Memory'),
 ('fd6a3480-7144-11e5-ba71-058fbc01cf0b', 'Inferential Reasoning Warm-up',
'Reasoning'),
 ('fd6a3480-7144-11e5-ba71-058fbc01cf0b', 'Inferential Reasoning Game',
'Reasoning'),
 ('fd6a3480-7144-11e5-ba71-058fbc01cf0b', 'Physical Reasoning Warm-up',
'Reasoning'),
 ('fd6a3480-7144-11e5-ba71-058fbc01cf0b', 'Physical Reasoning Game',
'Reasoning'),
 ('fd6a2350-7144-11e5-ba71-058fbc01cf0b', 'Memory versus Smell', 'Memory'),
 ('fd6924aa-7144-11e5-ba71-058fbc01cf0b', 'Yawn Warm-up', 'Empathy'),
 ('fd6924aa-7144-11e5-ba71-058fbc01cf0b', 'Yawn Game', 'Empathy'),
 ('fd6924aa-7144-11e5-ba71-058fbc01cf0b', 'Eye Contact Warm-up', 'Empathy'),
 ('fd6924aa-7144-11e5-ba71-058fbc01cf0b', 'Eye Contact Game', 'Empathy')]
```

There are many more operators you can use in your WHERE clauses to restrict the data you select as well. We do not have the space to go over each one individually in this lesson, but I encourage you to explore them on your own. *This is a great area to practice being fearless and bold in your desire to learn new things! The more you try, the more you will learn.*

Feel free to practice any other functions or operators you discover in the space below:

[ ]:
```

```