MySQL Exercise 3: Formatting Selected Data

In this lesson, we are going to learn about three SQL clauses or functionalities that will help you format and edit the output of your queries. We will also learn how to export the results of your formatted queries to a text file so that you can analyze them in other software packages such as Tableau or Excel.

Begin by loading the SQL library into Jupyter, connecting to the Dognition database, and setting Dognition as the default database.

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

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

```
0 rows affected.
```

Out[1]: []

1. Use AS to change the titles of the columns in your output

The AS clause allows you to assign an alias (a temporary name) to a table or a column in a table. Aliases can be useful for increasing the readability of queries, for abbreviating long names, and for changing column titles in query outputs. To implement the AS clause, include it in your query code immediately after the column or table you want to rename. For example, if you wanted to change the name of the time stamp field of the completed_tests table from "created_at" to "time_stamp" in your output, you could take advantage of the AS clause and execute the following query:

```
SELECT dog_guid, created_at AS time_stamp
FROM complete_tests
```

Note that if you use an alias that includes a space, the alias must be surrounded in quotes:

```
SELECT dog_guid, created_at AS "time stamp"
FROM complete_tests
```

You could also make an alias for a table:

```
SELECT dog_guid, created_at AS "time stamp"
FROM complete_tests AS tests
```

Since aliases are strings, again, MySQL accepts both double and single quotation marks, but some database systems only accept single quotation marks. It is good practice to avoid using SQL keywords in your aliases, but if you have to use an SQL keyword in your alias for some reason, the string must be enclosed in backticks instead of quotation marks.

Question 1: How would you change the title of the "start_time" field in the exam_answers table to "exam start time" in a query output? Try it below:

In [5]:
```
%%sql
SELECT start_time AS 'exam start time'
FROM exam_answers
LIMIT 5
```

```
5 rows affected.
```

Out[5]:

exam start time
2013-02-05 03:58:13
2013-02-05 03:58:31
2013-02-05 03:59:03
2013-02-05 03:59:10
2013-02-05 03:59:22

2. Use DISTINCT to remove duplicate rows

Especially in databases like the Dognition database where no primary keys were declared in each table, sometimes entire duplicate rows can be entered in error. Even with no duplicate rows present, sometimes your queries correctly output multiple instances of the same value in a column, but you are interested in knowing what the different possible values in the column are, not what each value in each row is. In both of these cases, the best way to arrive at the clean results you want is to instruct the query to return only values that are distinct, or different from all the rest. The SQL keyword that allows you to do this is called DISTINCT. To use it in a query, place it directly after the word SELECT in your query.

For example, if we wanted a list of all the breeds of dogs in the Dognition database, we could try the following query from a previous exercise:

```
SELECT breed
FROM dogs;
```

However, the output of this query would not be very helpful, because it would output the entry for every single row in the breed column of the dogs table, regardless of whether it duplicated the breed of a previous entry. Fortunately, we could arrive at the list we want by executing the following query with the DISTINCT modifier:

```
SELECT DISTINCT breed
FROM dogs;
```

Try it yourself (If you do not limit your output, you should get 2006 rows in your output):

In [8]:
```
%%sql
SELECT DISTINCT breed
FROM dogs
```

```
2006 rows affected.
```

If you scroll through the output, you will see that no two entries are the same. Of note, if you use the DISTINCT clause on a column that has NULL values, MySQL will include one NULL value in the DISTINCT output from that column.

When the DISTINCT clause is used with multiple columns in a SELECT statement, the combination of all the columns together is used to determine the uniqueness of a row in a result set.

For example, if you wanted to know all the possible combinations of states and cities in the users table, you could query:

```
SELECT DISTINCT state, city
FROM users;
```

Try it (if you don't limit your output you'll see 3999 rows in the query result, of which the first 1000 are displayed):

In [7]:
```
%%sql
SELECT DISTINCT state, city
FROM users
LIMIT 20
```

20 rows affected.

Out[7]:

state	city
ND	Grand Forks
MA	Barre
CT	Darien
IL	Winnetka
NC	Raleigh
WA	Auburn
CO	Fort Collins
WA	Seattle
WA	Bainbridge Island
WA	Bremerton
CA	Aptos
OH	North Ridgeville
VA	Charlottesville
NC	Hillsborough
NY	New York
CA	Los Angeles
WY	Worland
IL	Grayslake
VA	Falls Church
WA	Everett

If you examine the query output carefully, you will see that there are many rows with California (CA) in the state column and four rows that have Gainesville in the city column (Georgia, Arkansas, Florida, and Virginia all have cities named Gainesville in our user table), but no two rows have the same state and city combination.

When you use the DISTINCT clause with the LIMIT clause in a statement, MySQL stops searching when it finds the number of *unique* rows specified in the LIMIT clause, not when it goes through the number of rows in the LIMIT clause.

For example, if the first 6 entries of the breed column in the dogs table were:

Labrador Retriever
Shetland Sheepdog
Golden Retriever
Golden Retriever
Shih Tzu
Siberian Husky

The output of the following query:

```
SELECT DISTINCT breed
FROM dogs LIMIT 5;
```

would be the first 5 different breeds:

Labrador Retriever
Shetland Sheepdog
Golden Retriever
Shih Tzu
Siberian Husky

not the distinct breeds in the first 5 rows:

Labrador Retriever
Shetland Sheepdog
Golden Retriever
Shih Tzu

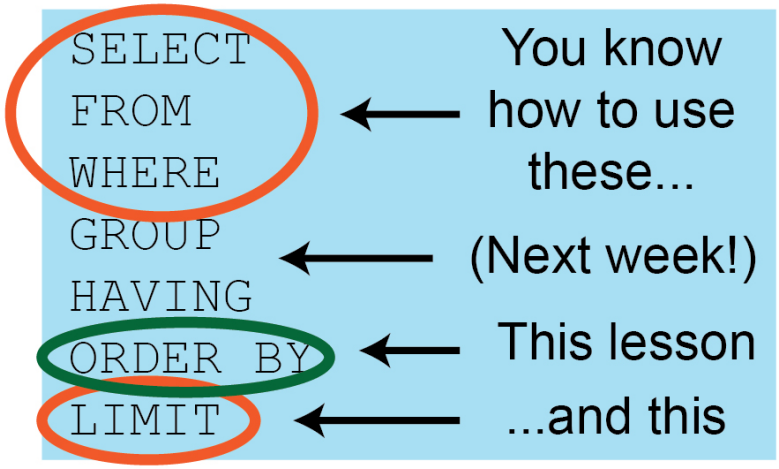
Question 2: How would you list all the possible combinations of test names and subcategory names in complete_tests table? (If you do not limit your output, you should retrieve 45 possible combinations)

In [ ]:
```
%%sql
SELECT DISTINCT test_name, subcategory_name
FROM complete_tests
```

3. Use ORDER BY to sort the output of your query

As you might have noticed already when examining the output of the queries you have executed thus far, databases do not have built-in sorting mechanisms that automatically sort the output of your query. However, SQL permits the use of the powerful ORDER BY clause to allow you to sort the output according to your own specifications. Let's look at how you would implement a simple ORDER BY clause.

Recall our query outline:



Your ORDER BY clause will come after everything else in the main part of your query, but before a LIMIT clause.

If you wanted the breeds of dogs in the dog table sorted in alphabetical order, you could query:

```
SELECT DISTINCT breed
FROM dogs
ORDER BY breed
```

Try it yourself:

In [9]:
```
%%sql
SELECT DISTINCT breed
FROM dogs
ORDER BY breed
```

```
2006 rows affected.
```

(You might notice that some of the breeds start with a hyphen; we'll come back to that later.)

The default is to sort the output in ascending order. However, you can tell SQL to sort the output in descending order as well:

```
SELECT DISTINCT breed
FROM dogs
ORDER BY breed DESC
```

Combining ORDER BY with LIMIT gives you an easy way to select the "top 10" and "last 10" in a list or column. For example, you could select the User IDs and Dog IDs of the 5 customer-dog pairs who spent the least median amount of time between their Dognition tests:

```
SELECT DISTINCT user_guid, median_ITI_minutes
FROM dogs
ORDER BY median_ITI_minutes
LIMIT 5
```

or the greatest median amount of time between their Dognition tests:

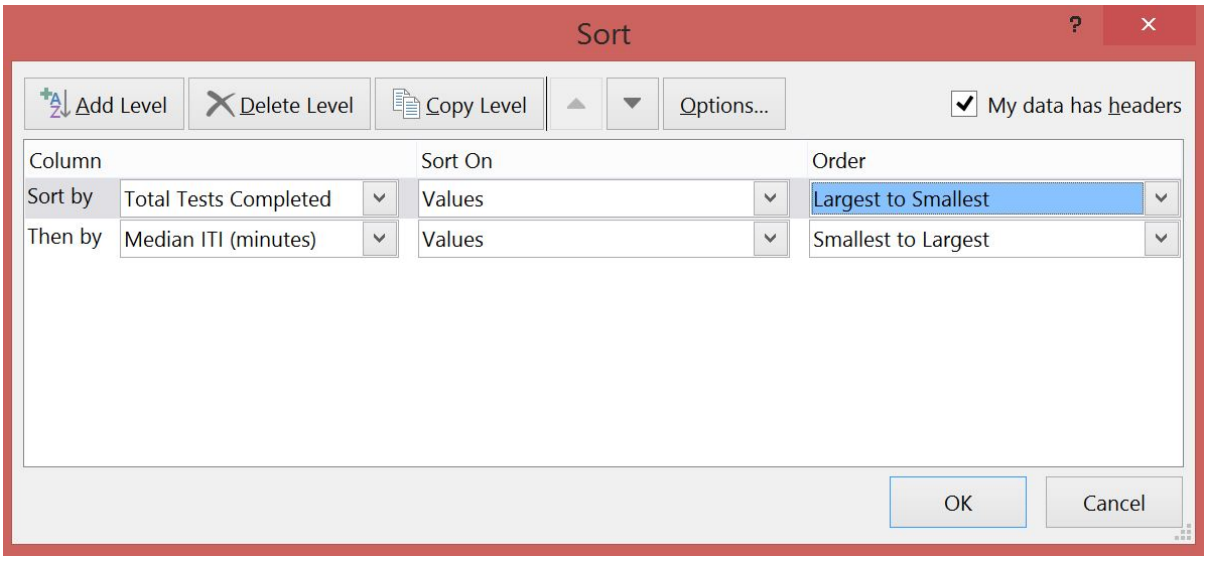
```
SELECT DISTINCT user_guid, median_ITI_minutes
FROM dogs
ORDER BY median_ITI_minutes DESC
LIMIT 5
```

You can also sort your output based on a derived field. If you wanted your inter-test interval to be expressed in seconds instead of minutes, you could incorporate a derived column and an alias into your last query to get the 5 customer-dog pairs who spent the greatest median amount of time between their Dognition tests in seconds:

```
SELECT DISTINCT user_guid, (median_ITI_minutes * 60) AS median_ITI_sec
FROM dogs
ORDER BY median_ITI_sec DESC
LIMIT 5
```

Note that the parentheses are important in that query; without them, the database would try to make an alias for 60 instead of median_ITI_minutes * 60.

SQL queries also allow you to sort by multiple fields in a specified order, similar to how Excel allows to include multiple levels in a sort (see image below):



To achieve this in SQL, you include all the fields (or aliases) by which you want to sort the results after the ORDER BY clause, separated by commas, in the order you want them to be used for sorting. You can then specify after each field whether you want the sort using that field to be ascending or descending.

If you wanted to select all the distinct User IDs of customers in the United States (abbreviated "US") and sort them according to the states they live in in alphabetical order first, and membership type second, you could query:

```
SELECT DISTINCT user_guid, state, membership_type
FROM users
WHERE country="US"
ORDER BY state ASC, membership_type ASC
```

Go ahead and try it yourself (if you do not limit the output, you should get 9356 rows in your output):

In [10]:
```
%%sql
SELECT DISTINCT user_guid, state, membership_type
FROM users
WHERE country="US"
ORDER BY state ASC, membership_type ASC
```

```
9356 rows affected.
```

You might notice that some of the rows have null values in the state field. You could revise your query to only select rows that do not have null values in either the state or membership_type column:

```
SELECT DISTINCT user_guid, state, membership_type
FROM users
WHERE country="US" AND state IS NOT NULL and membership_type IS NOT NULL
ORDER BY state ASC, membership_type ASC
```

Question 3: Below, try executing a query that would sort the same output as described above by membership_type first in descending order, and state second in ascending order:

In [11]:
```
%%sql
SELECT DISTINCT user_guid, state, membership_type
FROM users
WHERE country="US" AND state IS NOT NULL and membership_type IS NOT NULL
ORDER BY state ASC, membership_type ASC
```

```
9356 rows affected.
```

4. Export your query results to a text file

Next week, we will learn how to complete some basic forms of data analysis in SQL. However, if you know how to use other analysis or visualization software like Excel or Tableau, you can implement these analyses with the SQL skills you have gained already, as long as you can export the results of your SQL queries in a format other software packages can read. Almost every database interface has a different method for exporting query results, so you will need to look up how to do it every time you try a new interface (another place where having a desire to learn new things will come in handy!).

There are two ways to export your query results using our Jupyter interface.

1. You can select and copy the output you see in an output window, and paste it into another program. Although this strategy is very simple, it only works if your output is very limited in size (since you can only paste 1000 rows at a time).
2. You can tell MySQL to put the results of a query into a variable (for our purposes consider a variable to be a temporary holding place), and then use Python code to format the data in the variable as a CSV file (comma separated value file, a .CSV file) that can be downloaded. When you use this strategy, all of the results of a query will be saved into the variable, not just the first 1000 rows as displayed in Jupyter, even if we have set up Jupyter to only display 1000 rows of the output.

Let's see how we could export query results using the second method.

To tell MySQL to put the results of a query into a variable, use the following syntax:

```
variable_name_of_your_choice = %sql [your full query goes he
re];
```

In this case, you must execute your SQL query all on one line. So if you wanted to export the list of dog breeds in the dogs table, you could begin by executing:

```
breed_list = %sql SELECT DISTINCT breed FROM dogs ORDER BY b
reed;
```

Go ahead and try it:

In [12]:
```
breed_list = %sql SELECT DISTINCT breed FROM dogs ORDER BY breed
```

```
2006 rows affected.
```

Once your variable is created, using the above command tell Jupyter to format the variable as a csv file using the following syntax:

```
the_output_name_you_want.csv('the_output_name_you_want.csv')
```

Since this line is being run in Python, do NOT include the %sql prefix when trying to execute the line. We could therefore export the breed list by executing:

```
breed_list.csv('breed_list.csv')
```

When you do this, all of the results of the query will be saved in the text file but the results will not be displayed in your notebook. This is a convenient way to retrieve large amounts of data from a query without taxing your browser or the server.

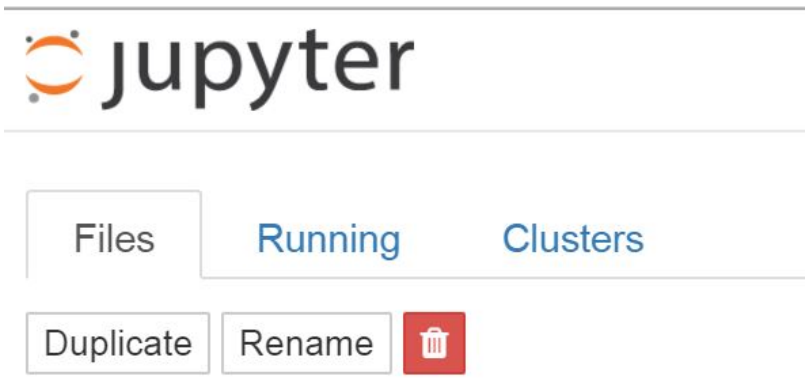
Try it yourself:

In [13]:
```
breed_list.csv('breed_list.csv')
```

Out[13]: CSV results (./files/breed_list.csv)

You should see a link in the output line that says "CSV results." You can click on this link to see the text file in a tab in your browser or to download the file to your computer (exactly how this works will differ depending on your browser and settings, but your options will be the same as if you were trying to open or download a file from any other website.)

You can also open the file directly from the home page of your Jupyter account. Behind the scenes, your csv file was written to your directory on the Jupyter server, so you should now see this file listed in your Jupyter account landing page along with the list of your notebooks. Just like a notebook, you can copy it, rename it, or delete it from your directory by clicking on the check box next to the file and clicking the "duplicate," "rename," or trash can buttons at the top of the page.



5. A Bird's Eye View of Other Functions You Might Want to Explore

When you open your breed list results file, you will notice the following:

1) All of the rows of the output are included, even though you can only see 1000 of those rows when you run the query through the Jupyter interface.

2) There are some strange values in the breed list. Some of the entries in the breed column seem to have a dash included before the name. This is an example of what real business data sets look like...they are messy! We will use this as an opportunity to highlight why it is so important to be curious and explore MySQL functions on your own.

If you needed an accurate list of all the dog breeds in the dogs table, you would have to find some way to "clean up" the breed list you just made. Let's examine some of the functions that could help you achieve this cleaning using SQL syntax rather than another program or language outside of the database.

I included these links to MySQL functions in an earlier notebook:
http://dev.mysql.com/doc/refman/5.7/en/func-op-summary-ref.html
(http://dev.mysql.com/doc/refman/5.7/en/func-op-summary-ref.html)
http://www.w3resource.com/mysql/mysql-functions-and-operators.php
(http://www.w3resource.com/mysql/mysql-functions-and-operators.php)

The following description of a function called REPLACE is included in that resource:

"REPLACE(str,from_str,to_str)
Returns the string str with all occurrences of the string from_str replaced by the string to_str. REPLACE() performs a case-sensitive match when searching for from_str."

One thing we could try is using this function to replace any dashes included in the breed names with no character:

```
SELECT DISTINCT breed,
REPLACE(breed,'-','') AS breed_fixed
FROM dogs
ORDER BY breed_fixed
```

In this query, we put the field/column name in the replace function where the syntax instructions listed "str" in order to tell the REPLACE function to act on the entire column. The "-" was the "from_str", which is the string we wanted to replace. The "" was the to_str, which is the character with which we want to replace the "from_str".

Try looking at the output:

In [14]:
```
%%sql
SELECT DISTINCT breed,
REPLACE(breed,'-','') AS breed_fixed
FROM dogs
ORDER BY breed_fixed
```

```
2006 rows affected.
```

That was helpful, but you'll still notice some issues with the output.

First, the leading dashes are indeed removed in the breed_fixed column, but now the dashes used to separate breeds in entries like 'French Bulldog-Boston Terrier Mix' are missing as well. So REPLACE isn't the right choice to selectively remove leading dashes.

Perhaps we could try using the TRIM function:

http://www.w3resource.com/mysql/string-functions/mysql-trim-function.php (http://www.w3resource.com/mysql/string-functions/mysql-trim-function.php)

```
SELECT DISTINCT breed, TRIM(LEADING '-' FROM breed) AS breed_fixed
FROM dogs
ORDER BY breed_fixed
```

Try the query written above yourself, and inspect the output carefully:

In [15]:
```
%%sql
SELECT DISTINCT breed, TRIM(LEADING '-' FROM breed) AS breed_fixed
FROM dogs
ORDER BY breed_fixed
```

```
2006 rows affected.
```

That certainly gets us a lot closer to the list we might want, but there are still some entries in the breed_fixed column that are conceptual duplicates of each other, due to poor consistency in how the breed names were entered. For example, one entry is "Beagle Mix" while another is "Beagle-Mix". These entries are clearly meant to refer to the same breed, but they will be counted as separate breeds as long as their breed names are different.

Cleaning up all of the entries in the breed column would take quite a bit of work, so we won't go through more details about how to do it in this lesson. Instead, use this exercise as a reminder for why it's so important to always look at the details of your data, and as motivation to explore the MySQL functions we won't have time to discuss in the course. If you push yourself to learn new SQL functions and embrace the habit of getting to know your data by exploring its raw values and outputs, you will find that SQL provides very efficient tools to clean real-world messy data sets, and you will arrive at the correct conclusions about what your data indicate your company should do.

Now it's time to practice using AS, DISTINCT, and ORDER BY in your own queries.

Question 4: How would you get a list of all the subcategories of Dognition tests, in alphabetical order, with no test listed more than once (if you do not limit your output, you should retrieve 16 rows)?

In [16]:
```
%%sql
SELECT DISTINCT subcategory_name
FROM complete_tests
ORDER BY subcategory_name
```

16 rows affected.

Out[16]:

subcategory_name
Communication
Cunning
Empathy
Expression Game
Impossible Task
Laterality
Memory
Numerosity
Perspective Game
Reasoning
Self Control Game
Shaker Game
Shell Game
Smell Game
Social Bias
Spatial Navigation

Question 5: How would you create a text file with a list of all the non-United States countries of Dognition customers with no country listed more than once?

In [20]:
```
non_us_countries = %sql SELECT DISTINCT city FROM users WHERE country != 'US';
non_us_countries.csv('non_us_countries')
```

782 rows affected.

Out[20]: CSV results (./files/non_us_countries)

Question 6: How would you find the User ID, Dog ID, and test name of the first 10 tests to ever be completed in the Dognition database?

In [22]:
```
%%sql
SELECT user_guid, dog_guid, test_name
FROM complete_tests
ORDER BY created_at
LIMIT 10
```

10 rows affected.

Out[22]:

user_guid	dog_guid	test_name
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Yawn Warm-up
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Yawn Game
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Eye Contact Warm-up
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Eye Contact Game
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Treat Warm-up
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Arm Pointing
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Foot Pointing
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Watching
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Turn Your Back
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Cover Your Eyes

Question 7: How would create a text file with a list of all the customers with yearly memberships who live in the state of North Carolina (USA) and joined Dognition after March 1, 2014, sorted so that the most recent member is at the top of the list?

In [26]:
```
NC_yearly_after_March_1_2014 = %sql SELECT DISTINCT user_guid, state, created_at
```

68 rows affected.

In [27]:
```
NC_yearly_after_March_1_2014.csv('NC_yearly_after_March_1_2014')
```

Out[27]: CSV results (./files/NC_yearly_after_March_1_2014)

Question 8: See if you can find an SQL function from the list provided at:

http://www.w3resource.com/mysql/mysql-functions-and-operators.php (http://www.w3resource.com/mysql/mysql-functions-and-operators.php)

that would allow you to output all of the distinct breed names in UPPER case. Create a query that would output a list of these names in upper case, sorted in alphabetical order.

In [28]:
```
%%sql
SELECT DISTINCT UPPER(breed)
FROM dogs
ORDER BY breed
```

```
2006 rows affected.
```

Practice any other queries you want to try below!

In [ ]:
```

```