MySQL Exercise 1: Welcome to your first notebook!

Database interfaces vary greatly across platforms and companies. The interface you will be using here, called Jupyter, is a web application designed for data science teams who need to create and share documents that share live code. We will be taking advantage of Jupyter's ability to integrate instructional text with areas that allow you to write and run SQL queries. In this course, you will practice writing queries about the Dognition data set in these Jupyter notebooks as I give you step-by-step instructions through written text. Then once you are comfortable with the query syntax, you will practice writing queries about the Dillard's data set in Teradata Viewpoint's more traditional SQL interface, called SQL scratchpad. Your assessments each week will be based on the exercises you complete using the Dillard's dataset.

Jupyter runs in a language called Python, so some of the commands you will run in these MySQL exercises will require incorporating small amounts of Python code, along with the MySQL query itself. Python is a very popular programming language with many statistical and visualization libraries, so you will likely encounter it in other business analysis settings. Since many data analysis projects do not use Python, however, I will point out to you what parts of the commands are specific to Python interfaces.

The first thing you should do every time you start working with a database: load the SQL library

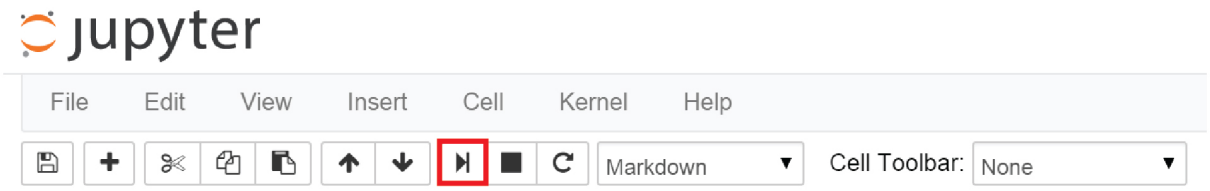
Since Jupyter is run through Python, the first thing you need to do to start practicing SQL queries is load the SQL library. To do this, type the following line of code into the empty cell below:

```
%load_ext sql
```

In [1]:
```
%load_ext sql
```

The "%" in this line of code is syntax for Python, not SQL. The "cell" I am referring to is the empty box area beside the "In [ ]:" you see on the left side of the screen.

Once you've entered the line of code, press the "run" button on the Jupyter toolbar that looks like an arrow. It's located under “cell” in the drop-down menu bar, and next to the stop button (the solid square). The run button is outlined in red in this picture:



TIP: Whenever instructions say “run” or "execute" a command in future exercises, type the appropriate code into the empty cell and execute it by pressing this same button with the arrow.

When the library has loaded, you will see a number between the brackets that preceed the line of code you executed:

For example, it might look like this:

```
In [2]:
```

The second thing you must do every time you want to start working with a database: connect to the database you need to use.

Now that the SQL library is loaded, we need to connect to a database. The following command will log you into the MySQL server at mysqlserver as the user 'studentuser' and will select the database named 'dognitiondb' :

```
mysql://studentuser:studentpw@mysqlserver/dognitiondb
```

However, to execute this command using SQL language instead of Python, you will need to begin the line of code with:

```
%sql
```

Thus, the complete line of code is:

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

Connect to the database by typing this command into the cell below, and running it (note: you can copy and paste the command from above rather than typing it, if you choose):

```
In [2]: %sql mysql://studentuser:studentpw@mysqlserver/dognitiondb
Out[2]: 'Connected: studentuser@dognitiondb'
```

Every time you run a line of SQL code in Jupyter, you will need to preface the line with "%sql". Remember to do this, even though I will not explicitly instruct you to do so for the rest of the exercises in this course.

Once you are connected, the output cell (which reads "Out" followed by brackets) will read: "Connected:studentuser@dognitiondb". To make this the default database for our queries, run this "USE" command:

```
%sql USE dognitiondb
```

In [3]:
```
%sql USE dognitiondb
```

```
0 rows affected.
```

Out[3]: []

You are now ready to run queries in the Dognition database!

The third thing you should do every time you start working with a new database: get to know your data

The data sets you will be working with in business settings will be big. REALLY big. If you just start making queries without knowing what you are pulling out, you could hang up your servers or be staring at your computer for hours before you get an output. Therefore, even if you are given an ER diagram or relational schema like we learned about in the first week of the course, before you start querying I strongly recommend that you (1) confirm how many tables each database has, and (2) identify the fields contained in each table of the database. To determine how many tables each database has, use the SHOW command:

```
SHOW tables
```

Try it yourself (TIP: if you get an error message, it's probably because you forgot to start the query with "%sql"):

In [4]:
```
%sql SHOW tables
```

```
6 rows affected.
```

Out[4]:

Tables_in_dognitiondb
complete_tests
dogs
exam_answers
reviews
site_activities
users

The output that appears above should show you there are six tables in the Dognition database. To determine what columns or fields (we will use those terms interchangeably in this course) are in each table, you can use the SHOW command again, but this time (1) you have to clarify that you want to see columns instead of tables, and (2) you have to specify from which table you want to examine the columns.

The syntax, which sounds very similar to what you would actually say in the spoken English language, looks like this:

```
SHOW columns FROM (enter table name here)
```

or if you have multiple databases loaded:

```
SHOW columns FROM (enter table name here) FROM (enter database name here)
```

or

```
SHOW columns FROM databasename.tablename
```

Whenever you have multiple databases loaded, you will need to specify which database a table comes from using one of the syntax options described above.

As I said in the earlier "Introduction to Query Syntax" video, it makes it easier to read and troubleshoot your queries if you always write SQL keywords in UPPERCASE format and write your table and field names in their native format. We will only use the most important SQL keywords in this course, but a full list can be found here:

https://dev.mysql.com/doc/refman/5.5/en/keywords.html (https://dev.mysql.com/doc/refman/5.5/en/keywords.html)

Question 1: How many columns does the "dogs" table have? Enter the appropriate query below to find out:

In [5]:
```
%sql SHOW columns FROM dogs
```

```
21 rows affected.
```

Out[5]:

Field	Type	Null	Key	Default	Extra
gender	varchar(255)	YES		None	
birthday	varchar(255)	YES		None	
breed	varchar(255)	YES		None	
weight	int(11)	YES		None	
dog_fixed	tinyint(1)	YES		None	
dna_tested	tinyint(1)	YES		None	
created_at	datetime	NO		None	
updated_at	datetime	NO		None	
dimension	varchar(255)	YES		None	

You should have determined that the "dogs" table has 21 columns.

An alternate way to learn the same information would be to use the DESCRIBE function. The syntax is:

```
DESCRIBE tablename
```

Question 2: Try using the DESCRIBE function to learn how many columns are in the "reviews" table:

In [6]:
```
%sql DESCRIBE reviews
```

```
7 rows affected.
```

Out[6]:

Field	Type	Null	Key	Default	Extra
rating	int(11)	YES		None	
created_at	datetime	NO		None	
updated_at	datetime	NO		None	
user_guid	varchar(60)	YES	MUL	None	
dog_guid	varchar(60)	YES	MUL	None	
subcategory_name	varchar(60)	YES		None	
test_name	varchar(60)	YES		None	

You should have determined that there are 7 columns in the "reviews" table.

The SHOW and DESCRIBE functions give a lot more information about the table than just how many columns, or fields, there are. Indeed, the first column of the output shows the title of each field in the table. The column next to that describes what type of data are stored in that column. There are 3 main types of data in MySQL: text, number, and datetime. There are many subtypes of data within these three general categories, as described here:

http://support.hostgator.com/articles/specialized-help/technical/phpmyadmin/mysql-variable-types (http://support.hostgator.com/articles/specialized-help/technical/phpmyadmin/mysql-variable-types)

The next column in the SHOW/DESCRIBE output indicates whether null values can be stored in the field in the table. The "Key" column of the output provides the following information about each field of data in the table being described (see https://dev.mysql.com/doc/refman/5.6/en/show-columns.html (https://dev.mysql.com/doc/refman/5.6/en/show-columns.html) for more information):

- Empty: the column either is not indexed or is indexed only as a secondary column in a multiple-column, nonunique index.
- PRI: the column is a PRIMARY KEY or is one of the columns in a multiple-column PRIMARY KEY.
- UNI: the column is the first column of a UNIQUE index.
- MUL: the column is the first column of a nonunique index in which multiple occurrences of a given value are permitted within the column.

The "Default" field of the output indicates the default value that is assigned to the field. The "Extra" field contains any additional information that is available about a given field in that table.

Questions 3-6: In the cells below, examine the fields in the other 4 tables of the Dognition database:

In [7]:
```
%sql DESCRIBE complete_tests
```

```
6 rows affected.
```

Out[7]:

Field	Type	Null	Key	Default	Extra
created_at	datetime	NO		None	
updated_at	datetime	NO		None	
user_guid	varchar(60)	YES	MUL	None	
dog_guid	varchar(60)	YES	MUL	None	
test_name	varchar(60)	YES		None	
subcategory_name	varchar(60)	YES		None	

In [8]:
```
%sql DESCRIBE exam_answers
```

```
8 rows affected.
```

Out[8]:

Field	Type	Null	Key	Default	Extra
script_detail_id	int(11)	YES		None	
subcategory_name	varchar(255)	YES		None	
test_name	varchar(255)	YES		None	
step_type	varchar(255)	YES		None	
start_time	datetime	YES		None	
end_time	datetime	YES		None	
loop_number	int(11)	YES		None	
dog_guid	varchar(60)	YES		None	

In [9]:
```
%sql DESCRIBE site_activities
```

```
11 rows affected.
```

Out[9]:

Field	Type	Null	Key	Default	Extra
activity_type	varchar(150)	YES	MUL	None	
description	text	YES		None	
membership_id	int(11)	YES		None	
category_id	int(11)	YES		None	
script_id	int(11)	YES		None	
created_at	datetime	NO		None	
updated_at	datetime	NO		None	
user_guid	varchar(255)	YES	MUL	None	
script_detail_id	int(11)	YES		None	
test_name	varchar(255)	YES		None	
dog_guid	varchar(255)	YES	MUL	None	

In [10]:
```
%sql DESCRIBE users
```

```
16 rows affected.
```

Out[10]:

Field	Type	Null	Key	Default	Extra
sign_in_count	int(11)	YES		0	
created_at	datetime	NO		None	
updated_at	datetime	NO		None	
max_dogs	int(11)	YES		0	
membership_id	int(11)	YES		None	
subscribed	tinyint(1)	YES		0	
exclude	tinyint(1)	YES		None	
free_start_user	tinyint(1)	YES		None	
last_active_at	datetime	YES		None	

As you examine the fields in each table, you will notice that none of the Dognition tables have primary keys declared. However, take note of which fields say "MUL" in the "Key" column of the DESCRIBE output, because these columns can still be used to link tables together. An important thing to keep in mind, though, is that because these linking columns were not configured as primary keys, it is possible the linking fields contain NULL values or duplicate rows.

If you do not have a ER diagram or relational schema of the Dognition database yet, consider making one at this point, because you will need to refer back to table and column names constantly throughout designing your queries (if you don't remember what primary keys, secondary keys, ER diagrams, or relational schemas are, review the material discussed in the first week of this course). Some database interfaces do provide notes or visual representations about the information in each table for easy reference, but since the Jupyter interface does not provide this, take your own notes and make your own diagrams, and keep them handy. As you will see, these diagrams and notes will save you a lot of time when you design your queries to pull data.

Using SELECT to look at your raw data

Once you have an idea of what is in your tables look like and how they might interact, it's a good idea to look at some of the raw data itself so that you are aware of any anomalies that could pose problems for your analysis or interpretations. To do that, we will use arguably the most important SQL statement for analysts: the SELECT statement.

SELECT is used anytime you want to retrieve data from a table. In order to retrieve that data, you always have to provide at least two pieces of information:

```
(1) what you want to select, and
(2) from where you want to select it.
```

I recommend that you always format your SQL code to ensure that these two pieces of information are on separate lines, so they are easy to identify quickly by eye.

The skeleton of a SELECT statement looks like this:

```
SELECT
FROM
```

To fill in the statement, you indicate the column names you are interested in after "SELECT" and the table name (and database name, if you have multiple databases loaded) you are drawing the information from after "FROM." So in order to look at the breeds in the dogs table, you would execute the following command:

```
SELECT breed
FROM dogs;
```

Remember:

- SQL syntax and keywords are case insensitive. I recommend that you always enter SQL keywords in upper case and table or column names in either lower case or their native format to make it easy to read and troubleshoot your code, but it is not a requirement to do so. Table or column names are often case insensitive as well, but defaults may vary across database platforms so it's always a good idea to check.
- Table or column names with spaces in them 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. In all database systems, if a table or column name contains an SQL keyword, the name 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`
```

- The semi-colon at the end of a query is only required when you have multiple separate queries saved in the same text file or editor. That said, I recommend that you make it a habit to always include a semi-colon at the end of your queries.

An important note for executing queries in Jupyter: in order to tell Python that you want to execute SQL language on multiple lines, you must include two percent signs in front of the SQL prefix instead of one. Therefore, to execute the above query, you should enter:

```
%%sql
SELECT breed
FROM dogs;
```

When Jupyter is busy executing a query, you will see an asterisk in the brackets next to the output field:

```
Out [*]
```

When the query is completed, you will see a number in the brackets next to the output field.

```
Out [5]
```

Try it yourself:

In [11]:

```
%%sql
SELECT breed
FROM dogs;
```

```
35050 rows affected.
```

Out[11]:

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

When you do so, you will see a line at the top of the output panel that says "35050 rows affected". This means that there are 35050 rows of data in the dogs table. Each row of the output lists the name of the breed of the dog represented by that entry. Notice that some breed names are listed multiple times, because several dogs of that breed have participated in the Dognition tests.

If you scroll all the way down to the bottom of the output, you will see a notification that says "35050 rows, truncated to displaylimit of 1000." We have set up a display limit of 1000 rows in these notebooks to ensure that our database servers are not overloaded, and to reduce the amount of time you have to wait for the query output. However, in a actual scenario these limits would not necessarily be in place for you. Therefore, before we go any further, I want to show you how you could restrict the number of rows outputted by a query.

CAUTION!!! Every time you select all the data from a column or a table without knowing how much data you are about to pull out, you run the risk of slowing down your network or staring at your screen for hours as you wait for your queries of billions of rows to finish. When in doubt, limit your output. You can always look at more data later.

Using LIMIT to restrict the number of rows in your output (and prevent system crashes)

The MySQL clause you should use is called LIMIT, and it is always placed at the very end of your query. The simplest version of a limit statement looks like this:

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

The "5" in this case indicates that you will only see the first 5 rows of data you select.

Question 7: In the next cell, try entering a query that will let you see the first 10 rows of the breed column in the dogs table.

In [12]:
```
%%sql
SELECT breed
FROM dogs LIMIT 10
```

```
10 rows affected.
```

Out[12]:

breed
Labrador Retriever
Shetland Sheepdog
Golden Retriever
Golden Retriever
Shih Tzu
Siberian Husky
Shih Tzu
Mixed
Labrador Retriever
Shih Tzu-Poodle Mix

You can also select rows of data from different parts of the output table, rather than always just starting at the beginning. To do this, use the OFFSET clause after LIMIT. The number after the OFFSET clause indicates from which row the output will begin querying. Note that the offset of Row 1 of a table is actually 0. Therefore, in the following query:

```
SELECT breed
FROM dogs LIMIT 10 OFFSET 5;
```

10 rows of data will be returned, starting at Row 6.

An alternative way to write the OFFSET clause in the query is:

```
SELECT breed
FROM dogs LIMIT 5, 10;
```

In this notation, the offset is the number before the comma, and the number of rows returned is the number after the comma.

Try it yourself:

In [13]:
```
%%sql
SELECT breed
FROM dogs LIMIT 10 OFFSET 5;
```

```
10 rows affected.
```

Out[13]:

breed
Siberian Husky
Shih Tzu
Mixed
Labrador Retriever
Shih Tzu-Poodle Mix
German Shepherd Dog-Pembroke Welsh Corgi Mix
Vizsla
Pug
Boxer
German Shepherd Dog-Nova Scotia Duck Tolling Retriever Mix

The LIMIT command is one of the pieces of syntax that can vary across database platforms. MySQL uses LIMIT to restrict the output, but other databases including Teradata use a statement called "TOP" instead. Oracle has yet another syntax:

http://www.tutorialspoint.com/sql/sql-top-clause.htm (http://www.tutorialspoint.com/sql/sql-top-clause.htm)

Make sure to look up the correct syntax for the database type you are using.

Using SELECT to query multiple columns

Now that we know how to limit our output, we are ready to make our SELECT statement work a little harder. The SELECT statement can be used to select multiple columns as well as a single column. The output of the query will depend on the order of the columns you enter after the SELECT statement in your query. When you enter column names, separate each name with a comma, but do NOT include a comma after the last column name.

Try the following query with different orders of the column names to observe the differences in output (I will include a LIMIT statement in many of the examples I use in the rest of the course, but feel free to change or remove them to explore different aspects of the data):

```
SELECT breed, breed_type, breed_group
FROM dogs LIMIT 5, 10;
```

In [15]:
```
%%sql
SELECT breed_type,breed,breed_group
FROM dogs
LIMIT 5,10
```

```
10 rows affected.
```

Out[15]:

breed_type	breed	breed_group
Pure Breed	Siberian Husky	Working
Pure Breed	Shih Tzu	Toy
Mixed Breed/ Other/ I Don't Know	Mixed	None
Pure Breed	Labrador Retriever	Sporting
Cross Breed	Shih Tzu-Poodle Mix	None
Cross Breed	German Shepherd Dog-Pembroke Welsh Corgi Mix	None
Pure Breed	Vizsla	Sporting
Pure Breed	Pug	Toy
Pure Breed	Boxer	Working
Cross Breed	German Shepherd Dog-Nova Scotia Duck Tolling Retriever Mix	None

Another trick to know about when using SELECT is that you can use an asterisk as a "wild card" to return all the data in a table. (A wild card is defined as a character that will represent or match any character or sequence of characters in a query.) Take note, this is very risky to do if you do not limit your output or if you don't know how many data are in your database, so use the wild card with caution. However, it is a handy tool to use when you don't have all the column names easily available or when you know you want to query an entire table.

The syntax is as follows:

```
SELECT *
FROM dogs LIMIT 5, 10;
```

Question 8: Try using the wild card to query the reviews table:

In [14]:
```
%%sql
SELECT *
FROM reviews LIMIT 5, 10;
```

```
10 rows affected.
```

Out[14]:

rating	created_at	updated_at	user_guid	dog_guid	subcategory_name	test_
6	2014-05-01 22:38:33	2014-05-01 22:38:33	ce47172c-7144-11e5-ba71-058fbc01cf0b	ce405c52-7144-11e5-ba71-058fbc01cf0b	Empathy	Yawn Game
3	2014-05-01 22:45:24	2014-05-01 22:45:24	ce47172c-7144-11e5-ba71-058fbc01cf0b	ce405c52-7144-11e5-ba71-058fbc01cf0b	Empathy	Eye Conta Game
0	2014-05-02 01:32:49	2014-05-02 01:32:49	ce471eca-7144-11e5-ba71-058fbc01cf0b	ce405e28-7144-11e5-ba71-058fbc01cf0b	Communication	Treat Warm

NOTE: if you do this for the dogs table, your output will be too wide to see at one time in the dialog box. Use the scroll bars at bottom and to the right of the dialog box to see the entire output table.

SELECT statements can also be used to make new derivations of individual columns using "+" for addition, "-" for subtraction, "*" for multiplication, or "/" for division. For example, if you wanted the median inter-test intervals in hours instead of minutes or days, you could query:

```
SELECT median_iti_minutes / 60
FROM dogs LIMIT 5, 10;
```

Question 9: Go ahead and try it, adding in a column to your output that shows you the original median_iti in minutes.

In [16]:
```
%%sql
SELECT median_iti_minutes/60, median_iti_minutes
FROM dogs LIMIT 5, 10;
```

```
10 rows affected.
```

Out[16]:

median_iti_minutes/60	median_iti_minutes
0.085555555285	5.1333333171
0.0080555537245	0.48333322347
0.11138889105833334	6.6833334635
0.081111109755	4.8666665853
0.09333333570666666	5.6000001424
0.14027777682833334	8.4166666097
0.10750000088166667	6.4500000529
0.09583333319833334	5.7499999919
0.094444443495	5.6666666097
0.09055555474166667	5.4333332845

Now it's time to practice writing your own SELECT statements.

Question 10: How would you retrieve the first 15 rows of data from the dog_guid, subcategory_name, and test_name fields of the Reviews table, in that order?

In [17]:
```
%%sql
SELECT dog_guid, subcategory_name, test_name FROM reviews
LIMIT 15
```

15 rows affected.

Out[17]:

dog_guid	subcategory_name	test_name
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

Question 11: How would you retrieve 10 rows of data from the activity_type, created_at, and updated_at fields of the site_activities table, starting at row 50? What do you notice about the created_at and updated_at fields?

In [19]:
```
%%sql
SELECT activity_type, created_at, updated_at
FROM site_activities
LIMIT 49,10;
```

10 rows affected.

Out[19]:

activity_type	created_at	updated_at
point_in_cat	2013-07-25 20:55:08	2013-07-25 20:55:08
point_in_cat	2013-07-25 20:55:07	2013-07-25 20:55:07
point_in_cat	2013-07-25 20:55:50	2013-07-25 20:55:50
point_in_cat	2013-07-25 20:56:09	2013-07-25 20:56:09
point_in_cat	2013-07-25 20:56:21	2013-07-25 20:56:21
point_in_cat	2013-07-25 21:00:31	2013-07-25 21:00:31
point_in_cat	2013-07-25 21:02:29	2013-07-25 21:02:29
point_in_cat	2013-07-25 21:02:31	2013-07-25 21:02:31
point_in_cat	2013-07-25 21:02:45	2013-07-25 21:02:45
point_in_cat	2013-07-25 21:05:41	2013-07-25 21:05:41

Question 12: How would you retrieve 20 rows of data from all the columns in the users

Question 12: How would you retrieve 20 rows of data from all the columns in the users table, starting from row 2000? What do you notice about the free_start_user field?

In [20]:
```
%%sql
SELECT * FROM users LIMIT 1999,20;
```

```
20 rows affected.
```

Out[20]:

sign_in_count	created_at	updated_at	max_dogs	membership_id	subscribed	exclu
1	2013-02-21 23:19:08	2015-01-28 20:51:53	1	1	1	None
3	2013-02-21 23:25:03	2015-01-28 20:51:53	1	2	1	None
1	2013-02-22 01:21:20	2015-01-28 20:51:53	1	1	1	None

You have already learned how to see all the data in your database! Congratulations!

Feel free to practice any other queries you are interested in below:

In [ ]:
```
```