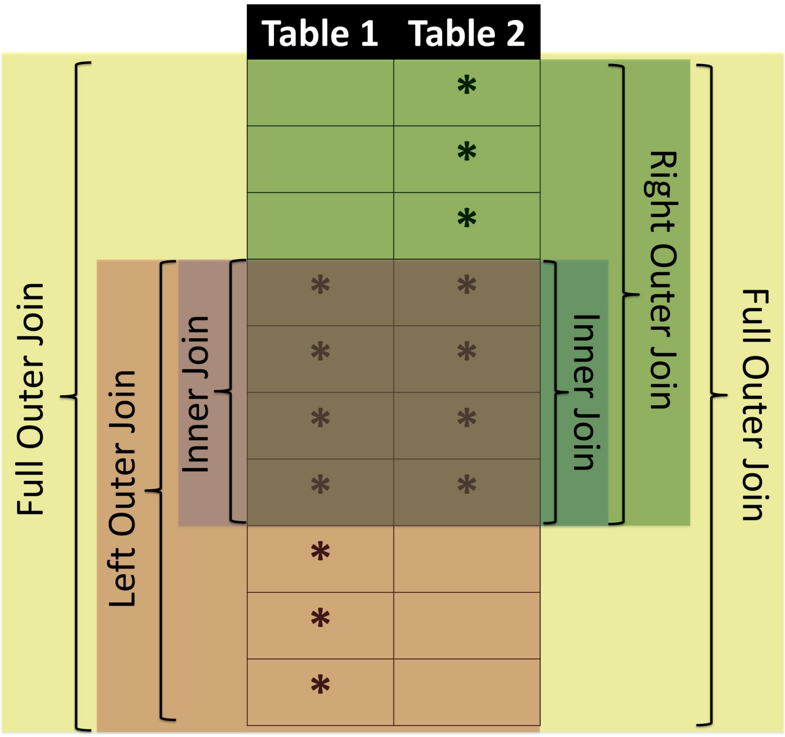
MySQL Exercise 8: Joining Tables with Outer Joins

We focused on inner joins in the last set of exercises. Most of the time inner joins will give you the results you are looking for. Occasionally, though, you might want to include all the data from a table in your calculations or your output, even if those data do not all match up with the data from the other tables you are joining with.

For example, if you have a table with customer demographic information and a table with information about which customers were sent a free sample, your might want to analyze the characteristics of customers who did and did not receive the sample. The best way to do that in a program like Tableau would be to have all your customer information in one table with an extra column indicating whether the customer received the free sample or not. Alternatively, you might want to generate a list of customers who did *not* receive the free sample, so that you can arrange for the customers to receive a free sample in the future.

In these types of situations, you will use outer joins to connect tables. Outer joins include left joins, right joins, or full outer joins (recall that full outer joins are NOT supported in MySQL). Refer to the videos "What are Joins?" and "Joins With Many to Many Relationships and Duplicates" for more information about joins.

Here's a picture to remind you of the general concepts behind outer joins:



To begin practicing outer joins, load the sql library, connect to the Dognition database, and make the Dognition database your default database:

In [1]:

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

```
0 rows affected.
```

Out[1]:

```
[]
```

Left and Right Joins

Left and right joins use a different sytax than we used in the lesson about inner joins. The method I showed you to execute inner joins tells the database how to relate tables in a WHERE clause like this:

```
WHERE d.dog_guid=r.dog_guid
```

I find this syntax -- called the "equijoin" syntax -- to be very intuitive, so I thought it would be a good idea to start with it. However, we can re-write the inner joins in the same syntax used by outer joins. To use this more traditional syntax, you have to tell the database how to connect the tables using an ON clause that comes right after the FROM clause. Make sure to specify the word "JOIN" explicitly. This traditional version of the syntax frees up the WHERE clause for other things you might want to include in your query. Here's what one of our queries from the inner join lesson would look like using the traditional syntax:

```
SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating, COUN
T(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type
FROM dogs d JOIN reviews r
  ON d.dog_guid=r.dog_guid AND d.user_guid=r.user_guid
GROUP BY d.user_guid
HAVING NumRatings > 9
ORDER BY AvgRating DESC
LIMIT 200
```

You could also write "INNER JOIN" instead of "JOIN" but the default in MySQL is that JOIN will mean inner join, so including the word "INNER" is optional.

If you need a WHERE clause in the query above, it would go after the ON clause and before the GROUP BY clause.

Here's an example of a different query we used in the last lesson that employed the equijoin syntax:

```
SELECT d.user_guid AS UserID, d.dog_guid AS DogID,
       d.breed, d.breed_type, d.breed_group
FROM dogs d, complete_tests c
WHERE d.dog_guid=c.dog_guid AND test_name='Yawn Warm-up';
```

Question 1: How would you re-write this query using the traditional join syntax?

In [3]:

```
%%sql
SELECT d.user_guid AS UserID, d.dog_guid AS DogID, d.breed, d.breed_type, d.br
eed_group
FROM dogs d
JOIN complete_tests c
ON d.dog_guid=c.dog_guid
WHERE test_name='Yawn Warm-up'
LIMIT 20;
```

```
20 rows affected.
```

Out[3]:

UserID	DogID	breed	breed_type	breed_group
ce134e42-7144-11e5-ba71-058fbc01cf0b	fd27b272-7144-11e5-ba71-058fbc01cf0b	Labrador Retriever	Pure Breed	Sporting
ce1353d8-7144-11e5-ba71-058fbc01cf0b	fd27b5ba-7144-11e5-ba71-058fbc01cf0b	Shetland Sheepdog	Pure Breed	Herding
ce135ab8-7144-11e5-ba71-058fbc01cf0b	fd27b6b4-7144-11e5-ba71-058fbc01cf0b	Golden Retriever	Pure Breed	Sporting
ce13507c-7144-11e5-ba71-058fbc01cf0b	fd27b79a-7144-11e5-ba71-058fbc01cf0b	Golden Retriever	Pure Breed	Sporting
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	Pure Breed	Toy
ce13615c-7144-11e5-ba71-058fbc01cf0b	fd27b948-7144-11e5-ba71-058fbc01cf0b	Siberian Husky	Pure Breed	Working
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27ba1a-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	Pure Breed	Toy
ce135f2c-7144-11e5-ba71-058fbc01cf0b	fd27bbbe-7144-11e5-ba71-058fbc01cf0b	Mixed	Mixed Breed/ Other/ I Don't Know	None
ce136a1c-7144-11e5-ba71-058fbc01cf0b	fd27c1c2-7144-11e5-ba71-058fbc01cf0b	Labrador Retriever	Pure Breed	Sporting
ce136ac6-7144-11e5-ba71-058fbc01cf0b	fd27c5be-7144-11e5-ba71-058fbc01cf0b	Shih Tzu-Poodle Mix	Cross Breed	None
ce136c24-7144-11e5-ba71-058fbc01cf0b	fd27c74e-7144-11e5-ba71-058fbc01cf0b	German Shepherd Dog-Pembroke Welsh Corgi Mix	Cross Breed	None
ce136e36-7144-11e5-ba71-058fbc01cf0b	fd27c7d0-7144-11e5-ba71-058fbc01cf0b	Vizsla	Pure Breed	Sporting
ce136ee0-7144-11e5-ba71-058fbc01cf0b	fd27c852-7144-11e5-ba71-058fbc01cf0b	Pug	Pure Breed	Toy

ce136f94-7144-11e5-ba71-058fbc01cf0b	fd27c8d4-7144-11e5-ba71-058fbc01cf0b	Boxer	Pure Breed	Working
ce134be0-7144-11e5-ba71-058fbc01cf0b	fd27c956-7144-11e5-ba71-058fbc01cf0b	German Shepherd Dog-Nova Scotia Duck Tolling Retriever Mix	Cross Breed	None
ce1371a6-7144-11e5-ba71-058fbc01cf0b	fd27cb72-7144-11e5-ba71-058fbc01cf0b	Beagle	Pure Breed	Hound
ce136f94-7144-11e5-ba71-058fbc01cf0b	fd27cd98-7144-11e5-ba71-058fbc01cf0b	Beagle	Pure Breed	Hound
ce136f94-7144-11e5-ba71-058fbc01cf0b	fd27ce1a-7144-11e5-ba71-058fbc01cf0b	Beagle	Pure Breed	Hound
ce1373ae-7144-11e5-ba71-058fbc01cf0b	fd27cea6-7144-11e5-ba71-058fbc01cf0b	Mixed	Mixed Breed/ Other/ I Don't Know	None
ce13750c-7144-11e5-ba71-058fbc01cf0b	fd27cf28-7144-11e5-ba71-058fbc01cf0b	Chesapeake Bay Retriever	Pure Breed	Sporting

Now that we've seen the join syntax, we can begin practicing outer joins. Our Dognition data set will make outer joins more challenging than usual, due to the lack of declared primary keys in the original database, the many-to-many relationships, and the presence of duplicate rows and NULL values in columns we will be using to combine tables. Mastering outer joins in this challenging context, though, will ensure that you understand the fundamental concepts behind joins, which will be a terrific benefit when you start writing queries in other company databases.

The first query I described above was originally written to address the question of whether dog owners who are particularly surprised by their dog's performance on Dognition tests tend to own similar breeds of dogs. When we designed this query in the last lesson, we wanted to focus on the dog owners who reported the highest average amount of surprise at their dog's performance, and provided at least 10 ratings for one or more of their dogs in the ratings. We also wanted to examine the breed, breed_type, and breed_group of each of these owner's dogs.

In examining the query, we learned that:

- All of the user_guids in the reviews table are in the dogs table
- Only 389 of the over 5000 dog_guids in the reviews table are in the dogs table

The inner join we executed resulted in 389 rows of output, because it only included the data from rows that have equivalent values in both tables being joined. But what if we wanted the full list of dogs in the reviews table and an indication of whether or not they were in the dogs table, rather than only a list of review information from dogs in the dogs table? To achieve this list, we could execute an outer join.

Let's start by using a left outer join to get the list we want. When we use the traditional join syntax to write inner joins, the order you enter the tables in your query doesn't matter. In outer joins, however, the order matters a lot. A left outer join will include all of the rows of the table to the left of the LEFT JOIN clause. A right outer join will include all of the rows of the table to the right of the RIGHT JOIN clause. So in order to retrieve a full list of dogs who completed at least 10 tests in the reviews table, and include as much breed information as possible, we could query:

```
SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_g
uid AS dUserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed,
 d.breed_group, d.breed_type
FROM reviews r LEFT JOIN dogs d
  ON r.dog_guid=d.dog_guid AND r.user_guid=d.user_guid
WHERE r.dog_guid IS NOT NULL
GROUP BY r.dog_guid
HAVING NumRatings >= 10
ORDER BY AvgRating DESC;
```

Question 2: How could you retrieve this same information using a RIGHT JOIN?

In [3]:

```
%%sql
SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_guid AS dUserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type
FROM dogs d RIGHT JOIN reviews r
ON r.dog_guid=d.dog_guid AND r.user_guid=d.user_guid
WHERE r.dog_guid IS NOT NULL
GROUP BY r.dog_guid
HAVING NumRatings >= 10
ORDER BY AvgRating DESC
LIMIT 20
```

```
20 rows affected.
```

Out[3]:

rDogID	dDogID	rUserID	dUserID	AvgRating	NumRatings	breed
fdbf39f8-7144-11e5-ba71-058fbc01cf0b	fdbf39f8-7144-11e5-ba71-058fbc01cf0b	ce987914-7144-11e5-ba71-058fbc01cf0b	ce987914-7144-11e5-ba71-058fbc01cf0b	8.0000	12	Canaa Dog
ce47553e-7144-11e5-ba71-058fbc01cf0b	None	ce6ca9ba-7144-11e5-ba71-058fbc01cf0b	None	7.8750	16	None
ce6f07e6-7144-11e5-ba71-058fbc01cf0b	None	ce7091e2-7144-11e5-ba71-058fbc01cf0b	None	7.5000	10	None
ce45ae5a-7144-11e5-ba71-058fbc01cf0b	None	ce67562c-7144-11e5-ba71-058fbc01cf0b	None	7.3529	17	None
ce2a68ac-7144-11e5-ba71-058fbc01cf0b	None	ce2a45c0-7144-11e5-ba71-058fbc01cf0b	None	7.1333	15	None
ce72be0e-7144-11e5-ba71-058fbc01cf0b	None	ce73f7a6-7144-11e5-ba71-058fbc01cf0b	None	7.0000	12	None
ce93d206-7144-11e5-ba71-058fbc01cf0b	None	ce8be19a-7144-11e5-ba71-058fbc01cf0b	None	6.7273	11	None
ce97cd7a-7144-11e5-ba71-058fbc01cf0b	None	ce948926-7144-11e5-ba71-058fbc01cf0b	None	6.7273	11	None
ce7038dc-7144-11e5-ba71-058fbc01cf0b	None	ce71a230-7144-11e5-ba71-058fbc01cf0b	None	6.6667	18	None
ce7dc3da-7144-11e5-ba71-058fbc01cf0b	None	ce7cbba2-7144-11e5-ba71-058fbc01cf0b	None	6.3750	16	None

ce866fda-7144-11e5-ba71-058fbc01cf0b	None	ce80c12a-7144-11e5-ba71-058fbc01cf0b	None	6.3000	10	None
ce2d9c98-7144-11e5-ba71-058fbc01cf0b	None	ce2d9108-7144-11e5-ba71-058fbc01cf0b	None	6.0000	15	None
ce94fc6c-7144-11e5-ba71-058fbc01cf0b	None	ce918db6-7144-11e5-ba71-058fbc01cf0b	None	6.0000	14	None
ce720978-7144-11e5-ba71-058fbc01cf0b	None	ce734fcc-7144-11e5-ba71-058fbc01cf0b	None	5.9500	20	None
ce719f88-7144-11e5-ba71-058fbc01cf0b	None	ce72ed7a-7144-11e5-ba71-058fbc01cf0b	None	5.9286	14	None
ce75ec8c-7144-11e5-ba71-058fbc01cf0b	None	ce76fd16-7144-11e5-ba71-058fbc01cf0b	None	5.8947	19	None
ce459c3a-7144-11e5-ba71-058fbc01cf0b	None	ce673e8a-7144-11e5-ba71-058fbc01cf0b	None	5.8333	18	None
ce8424b4-7144-11e5-ba71-058fbc01cf0b	None	ce8127b4-7144-11e5-ba71-058fbc01cf0b	None	5.8182	11	None
ce93d382-7144-11e5-ba71-058fbc01cf0b	None	ce8be19a-7144-11e5-ba71-058fbc01cf0b	None	5.8000	10	None
ce7aa06a-7144-11e5-ba71-058fbc01cf0b	None	ce7b7742-7144-11e5-ba71-058fbc01cf0b	None	5.7692	13	None

Notice in the output of both the left and the right version of the outer join, all the rows that had a dog_guid in the reviews table but did NOT have a matching dog_guid in the dogs table have the word "None" entered in output columns related to the dogs table. "None", in this case, is Jupyter's way of saying the value is NULL. This becomes clear when you query a list of only the dog_guids that were NOT in the dogs table:

```
SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_g
uid AS dUserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed,
 d.breed_group, d.breed_type
FROM reviews r LEFT JOIN dogs d
  ON r.dog_guid=d.dog_guid AND r.user_guid=d.user_guid
WHERE d.dog_guid IS NULL
GROUP BY r.dog_guid
HAVING NumRatings >= 10
ORDER BY AvgRating DESC;
```

Go ahead and try it yourself (you should get 894 rows in your query):

In [5]:

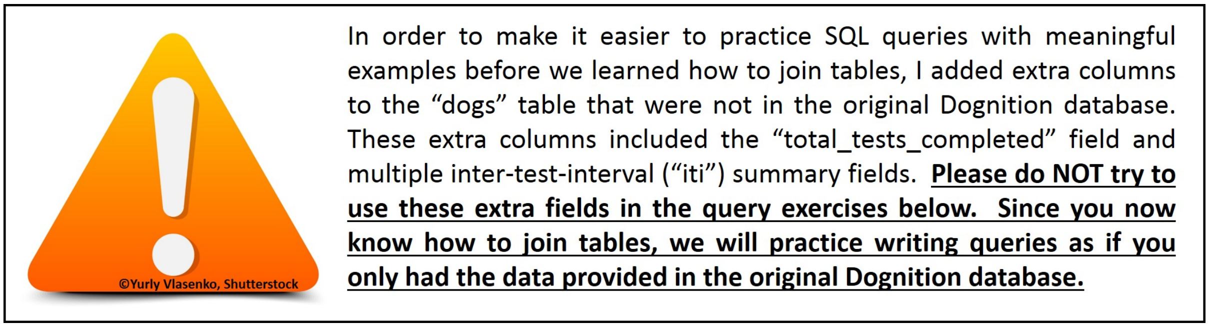
```
%%sql
SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_guid AS dUserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type
FROM reviews r LEFT JOIN dogs d
  ON r.dog_guid=d.dog_guid AND r.user_guid=d.user_guid
WHERE d.dog_guid IS NULL
GROUP BY r.dog_guid
HAVING NumRatings >= 10
ORDER BY AvgRating DESC
LIMIT 20
```

```
20 rows affected.
```

Out[5]:

rDogID	dDogID	rUserID	dUserID	AvgRating	NumRatings	breed	breed_gr
ce47553e-7144-11e5-ba71-058fbc01cf0b	None	ce6ca9ba-7144-11e5-ba71-058fbc01cf0b	None	7.8750	16	None	None
ce6f07e6-7144-11e5-ba71-058fbc01cf0b	None	ce7091e2-7144-11e5-ba71-058fbc01cf0b	None	7.5000	10	None	None
ce45ae5a-7144-11e5-ba71-058fbc01cf0b	None	ce67562c-7144-11e5-ba71-058fbc01cf0b	None	7.3529	17	None	None
ce2a68ac-7144-11e5-ba71-058fbc01cf0b	None	ce2a45c0-7144-11e5-ba71-058fbc01cf0b	None	7.1333	15	None	None
ce72be0e-7144-11e5-ba71-058fbc01cf0b	None	ce73f7a6-7144-11e5-ba71-058fbc01cf0b	None	7.0000	12	None	None
ce93d206-7144-11e5-ba71-058fbc01cf0b	None	ce8be19a-7144-11e5-ba71-058fbc01cf0b	None	6.7273	11	None	None
ce97cd7a-7144-11e5-ba71-058fbc01cf0b	None	ce948926-7144-11e5-ba71-058fbc01cf0b	None	6.7273	11	None	None
ce7038dc-7144-11e5-ba71-058fbc01cf0b	None	ce71a230-7144-11e5-ba71-058fbc01cf0b	None	6.6667	18	None	None
ce7dc3da-7144-11e5-ba71-058fbc01cf0b	None	ce7cbba2-7144-11e5-ba71-058fbc01cf0b	None	6.3750	16	None	None
ce866fda-7144-11e5-ba71-058fbc01cf0b	None	ce80c12a-7144-11e5-ba71-058fbc01cf0b	None	6.3000	10	None	None

ce2d9c98-7144-11e5-ba71-058fbc01cf0b	None	ce2d9108-7144-11e5-ba71-058fbc01cf0b	None	6.0000	15	None	None
ce94fc6c-7144-11e5-ba71-058fbc01cf0b	None	ce918db6-7144-11e5-ba71-058fbc01cf0b	None	6.0000	14	None	None
ce720978-7144-11e5-ba71-058fbc01cf0b	None	ce734fcc-7144-11e5-ba71-058fbc01cf0b	None	5.9500	20	None	None
ce719f88-7144-11e5-ba71-058fbc01cf0b	None	ce72ed7a-7144-11e5-ba71-058fbc01cf0b	None	5.9286	14	None	None
ce75ec8c-7144-11e5-ba71-058fbc01cf0b	None	ce76fd16-7144-11e5-ba71-058fbc01cf0b	None	5.8947	19	None	None
ce459c3a-7144-11e5-ba71-058fbc01cf0b	None	ce673e8a-7144-11e5-ba71-058fbc01cf0b	None	5.8333	18	None	None
ce8424b4-7144-11e5-ba71-058fbc01cf0b	None	ce8127b4-7144-11e5-ba71-058fbc01cf0b	None	5.8182	11	None	None
ce93d382-7144-11e5-ba71-058fbc01cf0b	None	ce8be19a-7144-11e5-ba71-058fbc01cf0b	None	5.8000	10	None	None
ce7aa06a-7144-11e5-ba71-058fbc01cf0b	None	ce7b7742-7144-11e5-ba71-058fbc01cf0b	None	5.7692	13	None	None
ce406288-7144-11e5-ba71-058fbc01cf0b	None	ce4725be-7144-11e5-ba71-058fbc01cf0b	None	5.7273	11	None	None



©Yurly Vlasenko, Shutterstock

In order to make it easier to practice SQL queries with meaningful examples before we learned how to join tables, I added extra columns to the "dogs" table that were not in the original Dognition database. These extra columns included the "total_tests_completed" field and multiple inter-test-interval ("iti") summary fields. **Please do NOT try to use these extra fields in the query exercises below. Since you now know how to join tables, we will practice writing queries as if you only had the data provided in the original Dognition database.**

Question 3: How would you use a left join to retrieve a list of all the unique dogs in the dogs table, and retrieve a count of how many tests each one completed? Include the dog_guids and user_guids from the dogs and complete_tests tables in your output. (If you do not limit your query, your output should contain 35050 rows. HINT: use the dog_guid from the dogs table to group your results.)

In [10]:

```
%%sql
SELECT COUNT(DISTINCT c.test_name), d.dog_guid AS DogID, d.user_guid AS UserID
FROM dogs d
LEFT JOIN complete_tests c
ON d.dog_guid = c.dog_guid
GROUP BY d.dog_guid
LIMIT 20
```

```
20 rows affected.
```

Out[10]:

COUNT(DISTINCT c.test_name)	DogID	UserID
21	fd27b272-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b
20	fd27b5ba-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b
2	fd27b6b4-7144-11e5-ba71-058fbc01cf0b	ce135ab8-7144-11e5-ba71-058fbc01cf0b
11	fd27b79a-7144-11e5-ba71-058fbc01cf0b	ce13507c-7144-11e5-ba71-058fbc01cf0b
30	fd27b86c-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b
20	fd27b948-7144-11e5-ba71-058fbc01cf0b	ce13615c-7144-11e5-ba71-058fbc01cf0b
27	fd27ba1a-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b
0	fd27baec-7144-11e5-ba71-058fbc01cf0b	ce1362ba-7144-11e5-ba71-058fbc01cf0b
20	fd27bbbe-7144-11e5-ba71-058fbc01cf0b	ce135f2c-7144-11e5-ba71-058fbc01cf0b
0	fd27be84-7144-11e5-ba71-058fbc01cf0b	ce13697c-7144-11e5-ba71-058fbc01cf0b
0	fd27bf60-7144-11e5-ba71-058fbc01cf0b	ce1352ac-7144-11e5-ba71-058fbc01cf0b
0	fd27c032-7144-11e5-ba71-058fbc01cf0b	ce1352ac-7144-11e5-ba71-058fbc01cf0b
0	fd27c0fa-7144-11e5-ba71-058fbc01cf0b	ce136a1c-7144-11e5-ba71-058fbc01cf0b
20	fd27c1c2-7144-11e5-ba71-058fbc01cf0b	ce136a1c-7144-11e5-ba71-058fbc01cf0b
0	fd27c294-7144-11e5-ba71-058fbc01cf0b	ce1366e8-7144-11e5-ba71-058fbc01cf0b
0	fd27c35c-7144-11e5-ba71-058fbc01cf0b	ce1366e8-7144-11e5-ba71-058fbc01cf0b
0	fd27c424-7144-11e5-ba71-058fbc01cf0b	ce1366e8-7144-11e5-ba71-058fbc01cf0b
0	fd27c4ec-7144-11e5-ba71-058fbc01cf0b	ce1366e8-7144-11e5-ba71-058fbc01cf0b

20	fd27c5be-7144-11e5-ba71-058fbc01cf0b	ce136ac6-7144-11e5-ba71-058fbc01cf0b
0	fd27c64a-7144-11e5-ba71-058fbc01cf0b	ce136c24-7144-11e5-ba71-058fbc01cf0b

Sometimes you can get so focused on writing your join statement that you don't pay close attention to the fields and tables you put in your other clauses, especially when you are joining a lot of tables. Often your query will still run successfully, even if you haven't entered the criteria or grouping clause you intended. The next question will illustrate how easy it is for this to happen.

Question 4: Repeat the query you ran in Question 3, but intentionally use the dog_guids from the completed_tests table to group your results instead of the dog_guids from the dogs table. (Your output should contain 17987 rows)

In [12]:

```
%%sql
SELECT COUNT(DISTINCT c.test_name), d.dog_guid AS DogID, d.user_guid AS UserID
FROM dogs d
LEFT JOIN complete_tests c
ON d.dog_guid = c.dog_guid
GROUP BY c.dog_guid
LIMIT 20
```

```
20 rows affected.
```

Out[12]:

COUNT(DISTINCT c.test_name)	DogID	UserID
0	fd27baec-7144-11e5-ba71-058fbc01cf0b	ce1362ba-7144-11e5-ba71-058fbc01cf0b
21	fd27b272-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b
20	fd27b5ba-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b
2	fd27b6b4-7144-11e5-ba71-058fbc01cf0b	ce135ab8-7144-11e5-ba71-058fbc01cf0b
11	fd27b79a-7144-11e5-ba71-058fbc01cf0b	ce13507c-7144-11e5-ba71-058fbc01cf0b
30	fd27b86c-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b
20	fd27b948-7144-11e5-ba71-058fbc01cf0b	ce13615c-7144-11e5-ba71-058fbc01cf0b
27	fd27ba1a-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b
20	fd27bbbe-7144-11e5-ba71-058fbc01cf0b	ce135f2c-7144-11e5-ba71-058fbc01cf0b
20	fd27c1c2-7144-11e5-ba71-058fbc01cf0b	ce136a1c-7144-11e5-ba71-058fbc01cf0b
20	fd27c5be-7144-11e5-ba71-058fbc01cf0b	ce136ac6-7144-11e5-ba71-058fbc01cf0b
14	fd27c74e-7144-11e5-ba71-058fbc01cf0b	ce136c24-7144-11e5-ba71-058fbc01cf0b
20	fd27c7d0-7144-11e5-ba71-058fbc01cf0b	ce136e36-7144-11e5-ba71-058fbc01cf0b
20	fd27c852-7144-11e5-ba71-058fbc01cf0b	ce136ee0-7144-11e5-ba71-058fbc01cf0b
20	fd27c8d4-7144-11e5-ba71-058fbc01cf0b	ce136f94-7144-11e5-ba71-058fbc01cf0b
11	fd27c956-7144-11e5-ba71-058fbc01cf0b	ce134be0-7144-11e5-ba71-058fbc01cf0b
20	fd27cb72-7144-11e5-ba71-058fbc01cf0b	ce1371a6-7144-11e5-ba71-058fbc01cf0b
20	fd27cd98-7144-11e5-ba71-058fbc01cf0b	ce136f94-7144-11e5-ba71-058fbc01cf0b

7	fd27ce1a-7144-11e5-ba71-058fbc01cf0b	ce136f94-7144-11e5-ba71-058fbc01cf0b
2	fd27cea6-7144-11e5-ba71-058fbc01cf0b	ce1373ae-7144-11e5-ba71-058fbc01cf0b

This time your query ran successfully, but you retrieved many fewer DogIDs because the GROUP BY clause grouped your results according to the dog_guids in the completed_tests table rather than the dog_guid table. As a result, even though you implemented your join correctly, all of the dog_guids that were in the dogs table but not in the completed_tests table got rolled up into one row of your output where completed_tests.dogs_guid = NULL. This is a good opportunity to remind ourselves about the differences between SELECT/GROUP BY and COUNT DISTINCT.

Question 5: Write a query using COUNT DISTINCT to determine how many distinct dog_guids there are in the completed_tests table.

In [14]:
```
%%sql
SELECT COUNT(DISTINCT dog_guid)
FROM complete_tests
```

```
1 rows affected.
```

Out[14]:

COUNT(DISTINCT dog_guid)
17986

The result of your COUNT DISTINCT clause should be 17,986 which is one row less than the number of rows you retrieved from your query in Question 4. That's because COUNT DISTINCT does NOT count NULL values, while SELECT/GROUP BY clauses roll up NULL values into one group. If you want to infer the number of distinct entries from the results of a query using joins and GROUP BY clauses, remember to include an "IS NOT NULL" clause to ensure you are not counting NULL values.

These exercises are a good illustration of why it is very helpful to save your queries when you are doing an analysis. Saving your queries allows you and your team members to double-check your work later. As you can see, the concepts behind SQL aren't themselves too tricky, but it is easy to make mistakes, especially when your queries get long and more complicated.

One more situation where joins can cause some confusion is when you have duplicate rows in a table you are joining. If you ignore what we've discussed about set theory and the way databases compute their joins, the behavior databases exhibit when you have duplicate rows in a joined table will seem utterly baffling. With this knowledge, though, the behavior will make perfect sense. Let's walk through what happens.

Question 6: We want to extract all of the breed information of every dog a user_guid in the users table owns. If a user_guid in the users table does not own a dog, we want that information as well. Write a query that would return this information. Include the dog_guid from the dogs table, and user_guid from both the users and dogs tables in your output. (HINT: you should get 952557 rows in your output!)

In [19]:

```
%%sql
SELECT d.dog_guid AS dDogID, d.user_guid AS dUserID, u.user_guid AS uUserID,
 d.breed
FROM users u
LEFT JOIN dogs d
ON u.user_guid = d.user_guid
LIMIT 20
```

```
20 rows affected.
```

Out[19]:

dDogID	dUserID	uUserID	breed
fd27b272-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b	Labrador Retriever
fd417cac-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b	Mixed
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b	Shetland Sheepdog
fd3fb0f2-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b	Shetland Sheepdog
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	ce135ab8-7144-11e5-ba71-058fbc01cf0b	ce135ab8-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
fd27b79a-7144-11e5-ba71-058fbc01cf0b	ce13507c-7144-11e5-ba71-058fbc01cf0b	ce13507c-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
fd27b86c-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
fd27e9a4-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
fd27ed46-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
fd3cf718-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
fd3d587a-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
fd3fbfe8-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
fd41c400-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
fd42e33a-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
fd3cffe2-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
fd42e196-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
fd453b6c-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
fd43c0c0-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu

fd27b948-7144-11e5-ba71-058fbc01cf0b	ce13615c-7144-11e5-ba71-058fbc01cf0b	ce13615c-7144-11e5-ba71-058fbc01cf0b	Siberian Husky

There are only 35050 distinct dog_guids in the dogs table. Why is the database outputting almost a million rows? That can't be right. Let's figure out what is going on.

Question 7: Adapt the query you wrote above so that it counts the number of rows the join will output per user_id. Sort the results by this count in descending order. Remember that if you include dog_guid or breed fields in this query, they will be randomly populated by only one of the values associated with a user_guid (see MySQL Exercise 6; there should be 33,193 rows in your output).

In [21]:

```
%%sql
SELECT d.dog_guid AS dDogID, d.user_guid AS dUserID, u.user_guid AS uUserID,
 d.breed, count(*) AS numrows
FROM users u
LEFT JOIN dogs d
ON u.user_guid = d.user_guid
GROUP BY u.user_guid
ORDER BY numrows DESC
LIMIT 20
```

```
20 rows affected.
```

Out[21]:

dDogID	dUserID	uUserID	breed	numrows
fd7bfb52-7144-11e5-ba71-058fbc01cf0b	ce7b75bc-7144-11e5-ba71-058fbc01cf0b	ce7b75bc-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	913138
fd423714-7144-11e5-ba71-058fbc01cf0b	ce225842-7144-11e5-ba71-058fbc01cf0b	ce225842-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	442
fd40bd62-7144-11e5-ba71-058fbc01cf0b	ce2258a6-7144-11e5-ba71-058fbc01cf0b	ce2258a6-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	320
fd27b86c-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	130
fd46b014-7144-11e5-ba71-058fbc01cf0b	ce29675e-7144-11e5-ba71-058fbc01cf0b	ce29675e-7144-11e5-ba71-058fbc01cf0b	Labrador Retriever- Mix	110
fd68ed6e-7144-11e5-ba71-058fbc01cf0b	ce6676d0-7144-11e5-ba71-058fbc01cf0b	ce6676d0-7144-11e5-ba71-058fbc01cf0b	Golden Retriever	64
fd7b3faa-7144-11e5-ba71-058fbc01cf0b	ce7adeea-7144-11e5-ba71-058fbc01cf0b	ce7adeea-7144-11e5-ba71-058fbc01cf0b	Labrador Retriever	49
fd401614-7144-11e5-ba71-058fbc01cf0b	ce47264a-7144-11e5-ba71-058fbc01cf0b	ce47264a-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	36
fdb54786-7144-11e5-ba71-058fbc01cf0b	ce8c2d08-7144-11e5-ba71-058fbc01cf0b	ce8c2d08-7144-11e5-ba71-058fbc01cf0b	Basenji	36
fd68e80a-7144-11e5-ba71-058fbc01cf0b	ce66713a-7144-11e5-ba71-058fbc01cf0b	ce66713a-7144-11e5-ba71-058fbc01cf0b	Puggle	30
fdbbdd4e-7144-11e5-ba71-058fbc01cf0b	ce964888-7144-11e5-ba71-058fbc01cf0b	ce964888-7144-11e5-ba71-058fbc01cf0b	Mixed	30
fd436e7c-7144-11e5-ba71-058fbc01cf0b	ce26b266-7144-11e5-ba71-058fbc01cf0b	ce26b266-7144-11e5-ba71-058fbc01cf0b	Pug	25
fd6cbb42-7144-11e5-ba71-058fbc01cf0b	ce6e67e6-7144-11e5-ba71-058fbc01cf0b	ce6e67e6-7144-11e5-ba71-058fbc01cf0b	Australian Shepherd	25

fd71a2f6-7144-11e5-ba71-058fbc01cf0b	ce728bf0-7144-11e5-ba71-058fbc01cf0b	ce728bf0-7144-11e5-ba71-058fbc01cf0b	Labrador Retriever-Golden Retriever Mix	25
fd3ccf2c-7144-11e5-ba71-058fbc01cf0b	ce135766-7144-11e5-ba71-058fbc01cf0b	ce135766-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	24
fd692aa4-7144-11e5-ba71-058fbc01cf0b	ce66b9b0-7144-11e5-ba71-058fbc01cf0b	ce66b9b0-7144-11e5-ba71-058fbc01cf0b	Mixed	24
fdad96a8-7144-11e5-ba71-058fbc01cf0b	ce83d2ca-7144-11e5-ba71-058fbc01cf0b	ce83d2ca-7144-11e5-ba71-058fbc01cf0b	Cockapoo	24
fd80bebc-7144-11e5-ba71-058fbc01cf0b	ce7da332-7144-11e5-ba71-058fbc01cf0b	ce7da332-7144-11e5-ba71-058fbc01cf0b	Mixed	20
fd40e206-7144-11e5-ba71-058fbc01cf0b	ce134492-7144-11e5-ba71-058fbc01cf0b	ce134492-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	18
fdc12bbe-7144-11e5-ba71-058fbc01cf0b	ce9a381c-7144-11e5-ba71-058fbc01cf0b	ce9a381c-7144-11e5-ba71-058fbc01cf0b	Chihuahua	18

This query told us that user 'ce7b75bc-7144-11e5-ba71-058fbc01cf0b' would be associated with 913,138 rows in the output of the outer join we designed! Once again, why? We are going to work with the second user_guid in the output you just generated, 'ce225842-7144-11e5-ba71-058fbc01cf0b', because it would be associated with 442 output rows, and 442 rows are much easier to work with than 913,138.

Question 8: How many rows in the *users* table are associated with user_guid 'ce225842-7144-11e5-ba71-058fbc01cf0b'?

In [22]:

```
%%sql
SELECT COUNT(*)
FROM users
WHERE user_guid = "ce225842-7144-11e5-ba71-058fbc01cf0b"
```

```
1 rows affected.
```

Out[22]:

COUNT(*)
17

There are 17 entries associated with that user_guid in the users table. If you examine all the columns in the entries, you will see that the rows are exact duplicates of each other. That's unfortunate, but also something that can happen in real life data sets, especially those from new companies or governmental agencies.

Ok, now...

Question 9: Examine all the rows in the *dogs* table that are associated with user_guid 'ce225842-7144-11e5-ba71-058fbc01cf0b'?

In [23]:

```
%%sql
SELECT COUNT(*)
FROM dogs d
WHERE user_guid = "ce225842-7144-11e5-ba71-058fbc01cf0b"
```

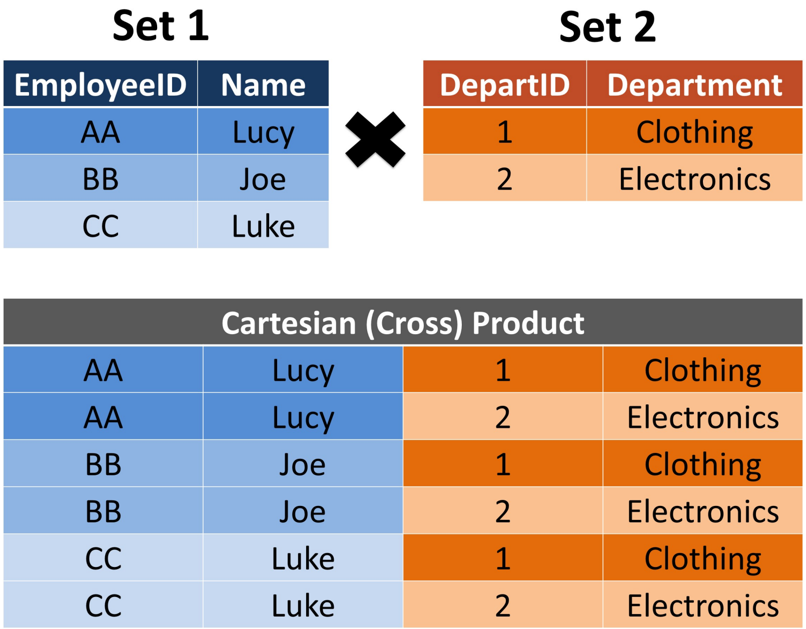
```
1 rows affected.
```

Out[23]:

COUNT(*)
26

You should see there are 26 rows associated with that UserID in the dogs table. When you examine the dogs table, you see that there are a lot of entries that have "Shih Tzu" in the breed column and "190" in the weight column. This was Dognition's internal convention for indicating test accounts. So these dog_guids and user_guids do not represent real data. Nonetheless, they provide a great example of what happens when you join on fields that have duplicate entries.

Recall the general strategy relational databases use to join tables:



Set 1

EmployeeID	Name
AA	Lucy
BB	Joe
CC	Luke

✖

Set 2

DepartID	Department
1	Clothing
2	Electronics

Cartesian (Cross) Product

AA	Lucy	1	Clothing
AA	Lucy	2	Electronics
BB	Joe	1	Clothing
BB	Joe	2	Electronics
CC	Luke	1	Clothing
CC	Luke	2	Electronics

When databases join tables, they output the result of every pair of entries that meet certain criteria in the linking column of one table with the linking column of another table. Our join statement imposed the criteria that the output should only include pairs whose user_guids matched in the two linking columns. However, since there were multiple rows that had the same user_guid in the users table, *each one of these rows got paired up with each row in the dogs table that had the same user_guid.* The result was 442 rows, because 17 (instances of the user_guid in the users table) x 26 (instances of the user_guid in the dogs table) = 442.

Having seen this, perhaps you can now appreciate why some database experts emphasize terminology that differentiates between set *theory* and real database *implementation*. Database operations, like those that join tables, are based on set theory that assumes there are no duplicate rows in your tables. In real life, duplicate rows get entered all the time, and it can cause you to misinterpret your queries if you do not understand the consequences. If you've been impacted enough times by the differences between real and theoretical databases, it makes sense that it would be important to you to use language that clearly distinguishes between theory and real life.

The important things I want you to remember from this example of joins with duplicates are that duplicate rows and table relationships that have table-to-table mappings of greater than 1 have multiplicative effects on your query results, due to the way relational databases combine tables. If you write queries that aggregate over a lot of joined tables, it can be very difficult to catch issues that output results you don't intend, because the aggregated results will hide clues from you. To prevent this from happening, I recommend you adopt the following practices:

- Avoid making assumptions about your data or your analyses. For example, rather than assume that all the values in a column are unique just because some documentation says they should be, check for yourself!
- Always look at example outputs of your queries before you strongly interpret aggregate calculations. Take extra care to do this when your queries require joins.
- When your queries require multiple layers of functions or joins, examine the output of each layer or join first before you combine them all together.
- Adopt a healthy skepticsm of all your data and results. If you see something you don't expect, make sure you explore it before interpreting it strongly or incorporating it into other analyses.

One more type of join to mention that I discussed in the joins videos is a full outer join. Full outer joins include all of the rows in both tables in an ON clause, regardless of whether there is a value that links the row of one table with a row in the other table. As with left or right joins, whenever a value in a row does not have a matching value in the joined table, NULLs will be entered for all values in the joined table.

Outer joins are used very rarely. The most practical application is if you want to export all of your raw data to another program for visualization or analysis. The syntax for outer joins is the same as for inner joins, but you replace the word "inner" with " full outer":

```
SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_g
uid AS dUserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed,
 d.breed_group, d.breed_type
FROM reviews r FULL OUTER JOIN dogs d
  ON r.dog_guid=d.dog_guid AND r.user_guid=d.user_guid
WHERE r.dog_guid IS NOT NULL
GROUP BY r.dog_guid
ORDER BY AvgRating DESC;
```

HOWEVER! MySQL does not support full outer joins.

If you wanted to imitate a full outer join in mySQL, you could follow one of the methods described at this website:

http://www.xaprb.com/blog/2006/05/26/how-to-write-full-outer-join-in-mysql/ (http://www.xaprb.com/blog/2006/05/26/how-to-write-full-outer-join-in-mysql/)

Practice outer joining your own tables!¶

Question 10: How would you write a query that used a *left* join to return the number of distinct user_guids that were in the users table, but not the dogs table (your query should return a value of 2226)?

In [31]:

```
%%sql
SELECT COUNT(DISTINCT u.user_guid)
FROM users u LEFT JOIN dogs d
ON u.user_guid=d.user_guid
WHERE d.user_guid IS NULL
```

```
1 rows affected.
```

Out[31]:

COUNT(DISTINCT u.user_guid)
2226

Question 11: How would you write a query that used a *right* join to return the number of distinct user_guids that were in the users table, but not the dogs table (your query should return a value of 2226)?

In [33]:

```
%%sql
SELECT COUNT(DISTINCT u.user_guid)
FROM dogs d
RIGHT JOIN users u
ON d.user_guid = u.user_guid
WHERE d.user_guid IS NULL
```

```
1 rows affected.
```

Out[33]:

COUNT(DISTINCT u.user_guid)
2226

Question 12: Use a left join to create a list of all the unique dog_guids that are contained in the site_activities table, but not the dogs table, and how many times each one is entered. Note that there are a lot of NULL values in the dog_guid of the site_activities table, so you will want to exclude them from your list. (Hint: if you exclude null values, the results you get will have two rows with words in their site_activities dog_guid fields instead of real guids, due to mistaken entries)

In [38]:

```
%%sql
SELECT s.dog_guid AS SA_dogs_not_present_in_dogs_table, COUNT(*) AS
NumEntries
FROM site_activities s LEFT JOIN dogs d
ON s.dog_guid=d.dog_guid
WHERE d.dog_guid IS NULL AND s.dog_guid IS NOT NULL
GROUP BY SA_dogs_not_present_in_dogs_table;
```

```
2 rows affected.
```

Out[38]:

SA_dogs_not_present_in_dogs_table	NumEntries
Membership	5587
PortalContent	12