MySQL Exercise 7: Joining Tables with Inner Joins

Before completing these exercises, I strongly recommend that you watch the video called "What are Joins?" that describe what joins are, and how different types of joins work.

As one of the last building blocks we need to address our Dognition analysis questions, in this lesson we will learn how to combine tables using inner joins.

1. Inner Joins between 2 tables

To begin, load the sql library, connect to the Dognition database, and set the Dognition database as the default.

In [ ]:

Recall that tables in relational databases are linked through primary keys and sometimes other fields that are common to multiple tables (as is the case with our Dognition data set). Our goal when we execute a JOIN or make a joined table is to use those common columns to let the database figure out which rows in one table match up to which rows in another table. Once that mapping is established using at least one common field or column, the database can pull any columns you want out of the mapped, or joined, tables and output the matched data to one common table.

An inner join is a join that outputs only rows that have an exact match in both tables being joined:

INNER\_JOIN

To illustrate how this works, let's find out whether dog owners that are particularly surprised by their dog's performance on Dognition tests tend to own similar breeds (or breed types, or breed groups) of dogs. There are many ways to address this question, but let's start by focusing on the dog owners who provided at least 10 ratings for one or more of their dogs in the ratings table. Of these owners, which 200 owners reported the highest average amount of surprise at their dog's performance, and what was the breed, breed\_type, and breed\_group of each of these owner's dog?

The surprise ratings are stored in the reviews table. The dog breed information is provided in the dogs table. There are two columns that are common to both tables: user\_guid and dog\_guid. How do we use the common columns to combine information from the two tables?

To join the tables, you can use a WHERE clause and add a couple of details to the FROM clause so that the database knows from what table each field in your SELECT clause comes.

First, start by adding all the columns we want to examine to the SELECT statement:

SELECT dog\_guid AS DogID, user\_guid AS UserID, AVG(rating) AS AvgRating,

COUNT(rating) AS NumRatings, breed, breed\_group, breed\_type

then list all the tables from which the fields we are interested in come, separated by commas (with no comma at the end of the list):

FROM dogs, reviews

then add the other restrictions:

GROUP BY user\_guid

HAVING NumRatings >= 10

ORDER BY AvgRating DESC

LIMIT 200

Try running this query and see what happens:

In [ ]:

You should receive an error message stating that the identity of dog\_guid and user\_guid in the field list is ambiguous. The reason is that the column title exists in both tables, and MySQL doesn't know which one we want. We have to specify the table name before stating the field name, and separate the two names by a period

SELECT dogs.dog\_guid AS DogID, dogs.user\_guid AS UserID, AVG(reviews.rating) AS AvgRating,

COUNT(reviews.rating) AS NumRatings, dogs.breed, dogs.breed\_group, dogs.breed\_type

FROM dogs, reviews

GROUP BY dogs.user\_guid

HAVING NumRatings >= 10

ORDER BY AvgRating DESC

LIMIT 200

You can also take advantage of aliases so that you don't have to write out the name of the tables each time. Here I will introduce another syntax for aliases that omits the AS completely. In this syntax, the alias is whatever word (or phrase, if you use quotation marks) follows immediately after the field or table name, separated by a space. So we could write:

SELECT d.dog\_guid AS DogID, d.user\_guid AS UserID, AVG(r.rating) AS AvgRating,

COUNT(r.rating) AS NumRatings, d.breed, d.breed\_group, d.breed\_type

FROM dogs d, reviews r

GROUP BY d.user\_guid

HAVING NumRatings >= 10

ORDER BY AvgRating DESC

LIMIT 200

I am tempted to tell you to run this query so that you will see what happens, but instead, I will explain what will happen and let you decide if you want to see what the output looks...and feels...like.

There is nothing built into the database table definitions that can instruct the server how to combine the tables on its own (remember, this is how relational databases save space and remain flexible). Further, the query as written does not tell the database how the two tables are related. As a consequence, rather than match up the two tables according to the values in the user\_id and/or dog\_id column, the database will do the only thing it knows how to do which is output every single combination of the records in the dogs table with the records in the reviews table. In other words, every single row of the dogs table will get paired with every single row of the reviews table. This is known as a Cartesian product. Not only will it be a heavy burden on the database to output a table that has the full length of one table multiplied times the full length of another (and frustrating to you, because the query would take a very long time to run), the output would be close to useless.

To prevent this from happening, tell the database how to relate the tables in the WHERE clause:

SELECT d.dog\_guid AS DogID, d.user\_guid AS UserID, AVG(r.rating) AS AvgRating,

COUNT(r.rating) AS NumRatings, d.breed, d.breed\_group, d.breed\_type

FROM dogs d, reviews r

WHERE d.dog\_guid=r.dog\_guid

GROUP BY d.user\_guid

HAVING NumRatings >= 10

ORDER BY AvgRating DESC

LIMIT 200

To be very careful and exclude any incorrect dog\_guid or user\_guid entries, you can include both shared columns in the WHERE clause:

SELECT d.dog\_guid AS DogID, d.user\_guid AS UserID, AVG(r.rating) AS AvgRating,

COUNT(r.rating) AS NumRatings, d.breed, d.breed\_group, d.breed\_type

FROM dogs d, reviews r

WHERE d.dog\_guid=r.dog\_guid AND d.user\_guid=r.user\_guid

GROUP BY d.user\_guid

HAVING NumRatings >= 10

ORDER BY AvgRating DESC

LIMIT 200

Try running this query now:

In [ ]:

The query should execute quickly. This would NOT have been the case if you did not include the WHERE clause to combine the two tables. If you accidentally request a Cartesian product from datasets with billions of rows, you could be waiting for your query output for days (and will probably get in trouble with your database administrator).

Let's examine our joined table a bit further. The joined table outputted by the query above should have 38 rows, despite the fact that we set our LIMIT at 200. The reason for this is that it turns out that a relatively small number of customers provided 10 or more reviews. If you remove the HAVING and LIMIT BY clause from the query, you should end up with 389 rows. Go ahead and try it:

In [ ]:

It's clear from looking at this output that (A) not many customers provided ratings, and (B) when they did, they usually were not very surprised by their dog's performance. Therefore, these ratings are probably not going to provide a lot of instructive insight into how to improve Dognition's completion rate. However, the ratings table still provides a great opportunity to illustrate the results of different types of joins.

To help prepare us for this:

Questions 1-4: How many unique dog\_guids and user\_guids are there in the reviews and dogs table independently?

In [ ]:

In [ ]:

In [ ]:

In [ ]:

These counts indicate some important things:

Many customers in both the reviews and the dogs table have multiple dogs

There are many more unique dog\_guids and user\_guids in the dogs table than the reviews table

There are many more unique dog\_guids and user\_guids in the reviews table than in the output of our inner join

Let's test one more thing.

Try the inner join query once with just the dog\_guid or once with just the user\_guid clause in the WHERE statement:

In [ ]:

In [ ]:

When you run the query by joining on the dog\_guid only, you still get 389 rows in your output. When you run the query by joining on the user\_guid only, you get 5586 rows in your output. This means 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

Perhaps most importantly for our current purposes, these COUNT queries show you that If you wanted to include all the dog\_guids or user\_guids in one or both of the tables, you would have to use an outer join, which we will practice in the next lesson.

Try an inner join on your own.

Question 5: How would you extract the user\_guid, dog\_guid, breed, breed\_type, and breed\_group for all animals who completed the "Yawn Warm-up" game (you should get 20,845 rows if you join on dog\_guid only)?

In [ ]:

2. Joining More than 2 Tables

In theory, you can join as many tables together as you want or need. To join multiple tables you take the same approach as we took when we were joining two tables together: list all the fields you want to extract in the SELECT statement, specify which table they came from in the SELECT statement, list all the tables from which you will need to extract the fields in the FROM statement, and then tell the database how to connect the tables in the WHERE statement.

To extract the user\_guid, user's state of residence, user's zip code, dog\_guid, breed, breed\_type, and breed\_group for all animals who completed the "Yawn Warm-up" game, you might be tempted to query:

SELECT c.user\_guid AS UserID, u.state, u.zip, d.dog\_guid AS DogID, d.breed, d.breed\_type, d.breed\_group

FROM dogs d, complete\_tests c, users u

WHERE d.dog\_guid=c.dog\_guid

AND c.user\_guid=u.user\_guid

AND c.test\_name="Yawn Warm-up";

This query focuses the relationships primarily on the complete\_tests table. However, it turns out that our Dognition dataset has only NULL values in the user\_guid column of the complete\_tests table. If you were to execute the query above, you would not get an error message, but your output would have 0 rows. However, the power of relational databases will come in handy here. You can use the dogs table to link the complete\_tests and users table (pay attention to the difference between the WHERE statement in this query vs. the WHERE statement in the query above):

SELECT d.user\_guid AS UserID, u.state, u.zip, d.dog\_guid AS DogID, d.breed, d.breed\_type, d.breed\_group

FROM dogs d, complete\_tests c, users u

WHERE d.dog\_guid=c.dog\_guid

AND d.user\_guid=u.user\_guid

AND c.test\_name="Yawn Warm-up";

Of note, joins are very resource intensive, so try not to join unnecessarily. In general, the more joins you have to execute, the slower your query performance will be.

Question 6: How would you extract the user\_guid, membership\_type, and dog\_guid of all the golden retrievers who completed at least 1 Dognition test (you should get 711 rows)?

In [ ]:

Practice inner joining your own tables!

Question 7: How many unique Golden Retrievers who live in North Carolina are there in the Dognition database (you should get 30)?

In [ ]:

Question 8: How many unique customers within each membership type provided reviews (there should be 3208 in the membership type with the greatest number of customers, and 18 in the membership type with the fewest number of customers)?

In [ ]:

Question 9: For which 3 dog breeds do we have the greatest amount of site\_activity data, (as defined by non-NULL values in script\_detail\_id)(your answers should be "Mixed", "Labrador Retriever", and "Labrador Retriever-Golden Retriever Mix"?

In [ ]:

Practice any other inner joins you would like to try here!

In [ ]: