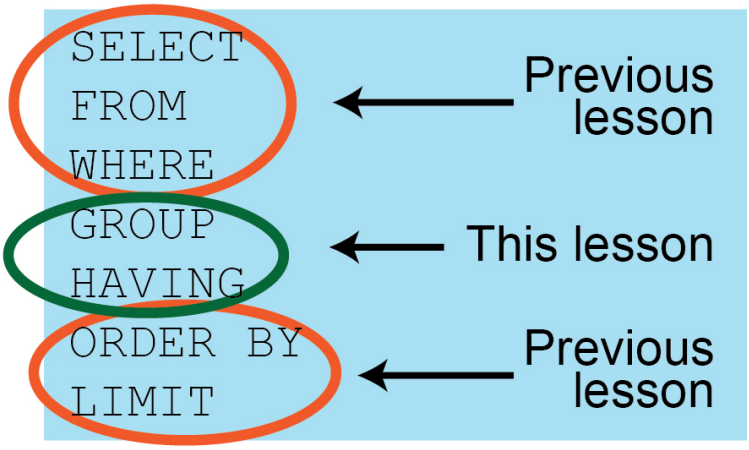
Copyright Jana Schaich Borg/Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)

MySQL Exercise 5: Summaries of Groups of Data

So far you've learned how to select, reformat, manipulate, order, and summarize data from a single table in database. In this lesson, you are going to learn how to summarize multiple subsets of your data in the same query. The method for doing this is to include a "GROUP BY" clause in your SQL query.

The GROUP BY clause

The GROUP BY clause comes after the WHERE clause, but before ORDER BY or LIMIT:



The GROUP BY clause is easy to incorporate into your queries. In fact, it might be a little too easy to incorporate into MySQL queries, because it can be used incorrectly in MySQL queries even when no error message is displayed. As a consequence, I suggest you adopt a healthy dose of caution every time you use the GROUP BY clause. By the end of this lesson, you will understand why. When used correctly, though, GROUP BY is one of the most useful and efficient parts of an SQL query, and once you are comfortable using it, you will use it very frequently.

To get started, load the SQL library and the Dognition database, and set the dognition database as the default:

In [2]:
```
% load_ext sql
%sql mysql://studentuser:studentpw@mysqlserver/dognitiondb
%sql USE dognitiondb
```

```
0 rows affected.
```

Out[2]: []

Let's return to a question from MySQL Exercise 4. How would you query the average rating for each of the 40 tests in the Reviews table? As we discussed, one very inefficient method to do that would

be to write 40 separate queries with each one having a different test name in the WHERE conditional clause. Then you could copy or transcribe the results from all 40 queries into one place. But that wouldn't be very pleasant. Here's how you could do the same thing using one query that has a GROUP BY clause:

```
SELECT test_name, AVG(rating) AS AVG_Rating
FROM reviews
GROUP BY test_name
```

This query will output the average rating for each test. More technically, this query will instruct MySQL to average all the rows that have the same value in the test_name column.

Notice that I included test_name in the SELECT statement. As a strong rule of thumb, if you are grouping by a column, you should also include that column in the SELECT statement. If you don't do this, you won't know to which group each row of your output corresponds.

To see what I mean, try the query above without test_name included in the SELECT statement:

In [4]:
```
%%sql
SELECT test_name, AVG(rating) AS AVG_Rating
FROM reviews
GROUP BY test_name
```

40 rows affected.

Out[4]:

test_name	AVG_Rating
1 vs 1 Game	3.9206
3 vs 1 Game	4.2857
5 vs 1 Game	3.9272
Arm Pointing	4.2153
Cover Your Eyes	2.6741
Delayed Cup Game	3.3514
Different Perspective	2.7647
Expression Game	4.0000
Eye Contact Game	2.9372
Eye Contact Warm-up	0.9632
Foot Pointing	4.0093
Impossible Task Game	3.0965
Impossible Task Warm-up	0.2174
Inferential Reasoning Game	4.5223
Inferential Reasoning Warm-up	4.3066
Memory versus Pointing	3.5584
Memory versus Smell	4.2623
Navigation Game	2.9841
Navigation Learning	2.0303
Navigation Warm-up	1.9805
Numerosity Warm-Up	2.6173
One Cup Warm-up	1.3693
Physical Reasoning Game	3.8492
Physical Reasoning Warm-up	1.6625
Self Control Game	3.8519
Shaker Game	4.6667
Shaker Warm-Up	2.1818
Shared Perspective	3.2778

Slide	4.5111
Smell Game	4.2857
Stair Game	4.2857
Switch	5.5676
Treat Warm-up	0.7909
Turn Your Back	3.1293
Two Cup Warm-up	1.6737
Warm-Up	1.2020
Watching	2.4594
Watching - Part 2	2.6570
Yawn Game	2.8477
Yawn Warm-up	2.0035

You can form groups using derived values as well as original columns. To illustrate this, let's address another question: how many tests were completed during each month of the year?

To answer this question, we need to take advantage of another datetime function described in the website below:

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

MONTH() will return a number representing the month of a date entry. To get the total number of tests completed each month, you could put the MONTH function into the GROUP BY clause, in this case through an alias:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Co
mpleted_Tests
FROM complete_tests
GROUP BY Month;
```

You can also group by multiple columns or derived fields. If we wanted to determine the total number of each type of test completed each month, you could include both "test_name" and the derived "Month" field in the GROUP BY clause, separated by a comma.

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Co
mpleted_Tests
FROM complete_tests
GROUP BY test_name, Month;
```

MySQL allows you to use aliases in a GROUP BY clause, but some database systems do not. If you are using a database system that does NOT accept aliases in GROUP BY clauses, you can still group by derived fields, but you have to duplicate the calculation for the derived field in the GROUP BY clause in addition to including the derived field in the SELECT clause:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Co
mpleted_Tests
FROM complete_tests
GROUP BY test_name, MONTH(created_at);
```

Try the query once with test_name first in the GROUP BY list, and once with Month first in the GROUP BY list below. Inspect the outputs:

In [9]:
```
%%sql
SELECT test_name, MONTH(created_at) AS month, COUNT(created_at) AS Num_Completed_
FROM complete_tests
GROUP BY test_name, month
LIMIT 20
```

```
20 rows affected.
```

Out[9]:

test_name	month	Num_Completed_Tests
1 vs 1 Game	1	25
1 vs 1 Game	2	28
1 vs 1 Game	3	22
1 vs 1 Game	4	12
1 vs 1 Game	5	13
1 vs 1 Game	6	18
1 vs 1 Game	7	36
1 vs 1 Game	8	17
1 vs 1 Game	9	28
1 vs 1 Game	10	27
1 vs 1 Game	11	15
1 vs 1 Game	12	14
3 vs 1 Game	1	35
3 vs 1 Game	2	28
3 vs 1 Game	3	34
3 vs 1 Game	4	16
3 vs 1 Game	5	34
3 vs 1 Game	6	42
3 vs 1 Game	7	37
3 vs 1 Game	8	23

In [10]:
```
%%sql
SELECT test_name, MONTH(created_at) AS month, COUNT(created_at) AS Num_Completed_
FROM complete_tests
GROUP BY test_name, MONTH(created_at)
LIMIT 20
```

20 rows affected.

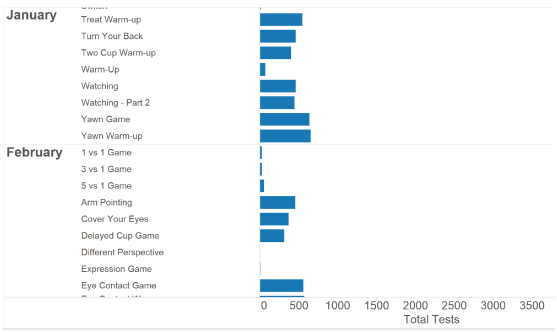
Out[10]:

test_name	month	Num_Completed_Tests
1 vs 1 Game	1	25
1 vs 1 Game	2	28
1 vs 1 Game	3	22
1 vs 1 Game	4	12
1 vs 1 Game	5	13
1 vs 1 Game	6	18
1 vs 1 Game	7	36
1 vs 1 Game	8	17
1 vs 1 Game	9	28
1 vs 1 Game	10	27
1 vs 1 Game	11	15
1 vs 1 Game	12	14
3 vs 1 Game	1	35
3 vs 1 Game	2	28
3 vs 1 Game	3	34
3 vs 1 Game	4	16
3 vs 1 Game	5	34
3 vs 1 Game	6	42
3 vs 1 Game	7	37
3 vs 1 Game	8	23

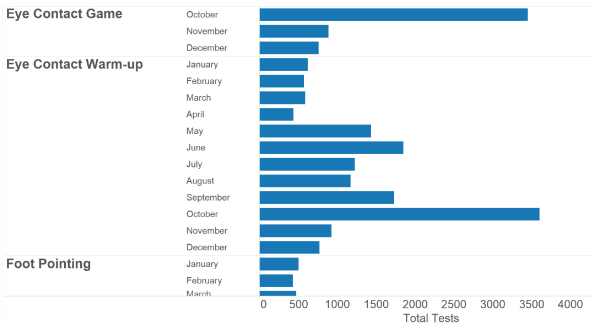
Notice that in the first case, the first block of rows share the same test_name, but are broken up into separate months (for those of you who took the "Data Visualization and Communication with Tableau" course of this specialization, this is similar to what would happen if you put test_name first and created_at second on the rows or columns shelf in Tableau).

In the second case, the first block of rows share the same month, but are broken up into separate tests (this is similar to what would happen if you put created_at first and test_name second on the rows or columns shelf in Tableau). If you were to visualize these outputs, they would look like the charts below.

GROUP BY MONTH(created_at), test_name;



GROUP BY test_name, MONTH(created_at)



Different database servers might default to ordering the outputs in a certain way, but you shouldn't rely on that being the case. To ensure the output is ordered in a way you intend, add an ORDER BY clause to your grouped query using the syntax you already know and have practiced:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Co
mpleted_Tests
FROM complete_tests
GROUP BY test_name, Month
ORDER BY test_name ASC, Month ASC;
```

Question 1: Output a table that calculates the number of distinct female and male dogs in each breed group of the Dogs table, sorted by the total number of dogs in descending order (the sex/breed_group pair with the greatest number of dogs should have 8466 unique Dog_Guids):

In [13]:
```
%%sql
SELECT gender, breed_group, COUNT(DISTINCT dog_guid) AS num_dogs
FROM dogs
GROUP BY breed_group, gender
ORDER BY num_dogs DESC
LIMIT 20
```

18 rows affected.

Out[13]:

gender	breed_group	num_dogs
male	None	8466
female	None	8367
male	Sporting	2584
female	Sporting	2262
male	Herding	1736
female	Herding	1704
male	Toy	1473
female	Toy	1145
male	Non-Sporting	1098
male	Working	1075
female	Non-Sporting	919
male	Terrier	919
female	Working	895
female	Terrier	794
male	Hound	725
female	Hound	614
male		147
female		127

Some database servers, including MySQL, allow you to use numbers in place of field names in the GROUP BY or ORDER BY fields to reduce the overall length of the queries. I tend to avoid this abbreviated method of writing queries because I find it challenging to troubleshoot when you are writing complicated queries with many fields, but it does allow you to write queries faster. To use this method, assign each field in your SELECT statement a number acording to the order the field appears in the SELECT statement. In the following statement:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Co
mpleted_Tests
```

test_name would be #1, Month would be #2, and Num_Completed_Tests would be #3. You could then rewrite the query above to read:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Co
mpleted_Tests
FROM complete_tests
GROUP BY 1, 2
ORDER BY 1 ASC, 2 ASC;
```

Question 2: Revise the query your wrote in Question 1 so that it uses only numbers in the GROUP BY and ORDER BY fields.

In [14]:
```
%%sql
SELECT gender, breed_group, COUNT(DISTINCT dog_guid) AS num_dogs
FROM dogs
GROUP BY 2, 1
ORDER BY 3 DESC
LIMIT 20
```

18 rows affected.

Out[14]:

gender	breed_group	num_dogs
male	None	8466
female	None	8367
male	Sporting	2584
female	Sporting	2262
male	Herding	1736
female	Herding	1704
male	Toy	1473
female	Toy	1145
male	Non-Sporting	1098
male	Working	1075
female	Non-Sporting	919
male	Terrier	919
female	Working	895
female	Terrier	794
male	Hound	725
female	Hound	614
male		147
female		127

The HAVING clause

Just like you can query subsets of rows using the WHERE clause, you can query subsets of aggregated groups using the HAVING clause. However, wheras the expression that follows a WHERE clause has to be applicable to each row of data in a column, the expression that follows a HAVING clause has to be applicable or computable using a group of data.

If you wanted to examine the number of tests completed only during the winter holiday months of November and December, you would need to use a WHERE clause, because the month a test was completed in is recorded in each row. Your query might look like this:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Co
mpleted_Tests
FROM complete_tests
WHERE MONTH(created_at)=11 OR MONTH(created_at)=12
GROUP BY 1, 2
ORDER BY 3 DESC;
```

If you then wanted to output only the test-month pairs that had at least 20 records in them, you would add a HAVING clause, because the stipulation of at least 20 records only makes sense and is only computable at the aggregated group level:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Co
mpleted_Tests
FROM complete_tests
WHERE MONTH(created_at)=11 OR MONTH(created_at)=12
GROUP BY 1, 2
HAVING COUNT(created_at)>=20
ORDER BY 3 DESC;
```

Question 3: Revise the query your wrote in Question 2 so that it (1) excludes the NULL and empty string entries in the breed_group field, and (2) excludes any groups that don't have at least 1,000 distinct Dog_Guids in them. Your result should contain 8 rows. (HINT: sometimes empty strings are registered as non-NULL values. You might want to include the following line somewhere in your query to exclude these values as well):

```
breed_group!=""
```

In [16]:
```
%%sql
SELECT gender, breed_group, COUNT(DISTINCT dog_guid) AS num_dogs
FROM dogs
WHERE breed_group IS NOT NULL AND breed_group != "None" AND breed_group != ""
GROUP BY breed_group, gender
HAVING num_dogs >= 1000
ORDER BY num_dogs DESC
```

```
8 rows affected.
```

Out[16]:

gender	breed_group	num_dogs
male	Sporting	2584
female	Sporting	2262
male	Herding	1736
female	Herding	1704
male	Toy	1473
female	Toy	1145
male	Non-Sporting	1098
male	Working	1075

We will review several issues that can be tricky about using GROUP BY in your queries in the next lesson, but those issues will make more sense once you are sure you are comfortable with the basic functionality of the GROUP BY and HAVING clauses.

Practice incorporating GROUP BY and HAVING into your own queries.

Question 4: Write a query that outputs the average number of tests completed and average mean inter-test-interval for every breed type, sorted by the average number of completed tests in descending order (popular hybrid should be the first row in your output).

In [17]:
```
%%sql
SELECT breed_type, AVG(total_tests_completed) AS TotTests, AVG(mean_iti_minutes)
AS AvgMeanITI
FROM dogs
GROUP BY breed_type
ORDER BY AVG(total_tests_completed) DESC
```

```
4 rows affected.

/opt/conda/lib/python3.5/site-packages/sqlalchemy/engine/default.py:470: Warnin
g: (1292, "Truncated incorrect DOUBLE value: '#VALUE!'")
  cursor.execute(statement, parameters)
```

Out[17]:

breed_type	TotTests	AvgMeanITI
Popular Hybrid	10.257530120481928	2834.3205728931534
Cross Breed	9.945900537634408	2872.351156110182
Pure Breed	9.871602824737856	3193.350493795222
Mixed Breed/ Other/ I Don't Know	9.54250850170034	3023.0711302156274

Question 5: Write a query that outputs the average amount of time it took customers to complete each type of test where any individual reaction times over 6000 hours are excluded and only average reaction times that are greater than 0 seconds are included (your output should end up with 67 rows).

In [21]:
```
%%sql
SELECT test_name, AVG(TIMESTAMPDIFF(hour, start_time, end_time)) AS avg_reaction_
FROM exam_answers
WHERE TIMESTAMPDIFF(minute, start_time, end_time) < 6000
GROUP BY test_name
HAVING AVG(TIMESTAMPDIFF(second, start_time, end_time)) > 0
ORDER BY avg_reaction_time DESC
```

67 rows affected.

Out[21]:

test_name	avg_reaction_time
Social-Quiz	2.3382
Social	0.7465
Diet	0.6735
Yawn Warm-up	0.5921
Activity	0.5096
Sociability	0.3241
Watching	0.2822
Set 3	0.2641
Surprise And Delight	0.2592
Confinement	0.1757
Navigation Warm-up	0.1637
Yawn Game	0.1556
Navigation Game	0.1273
Set 2	0.1259
Physical	0.1232
Emotions	0.1173
Impossible Task Warm-up	0.1040
Gender	0.0882
Watching - Part 2	0.0834
Stair Game	0.0793
Set 1	0.0790
Environment	0.0779
Turn Your Back	0.0759
Eye Contact Game	0.0738
Shaker Warm-Up	0.0644
Treat Warm-up	0.0601

Owner	0.0567
Cover Your Eyes	0.0555
Training	0.0550
Switch	0.0531
Toys	0.0521
Eye Contact Warm-up	0.0513
Different Perspective	0.0493
Warm-Up	0.0486
One Cup Warm-up	0.0472
Shy/Boldness	0.0436
Purina-Only	0.0430
Attachment	0.0381
Physical Reasoning Warm-up	0.0351
Perception	0.0339
Inferential Reasoning Warm-up	0.0336
Purina	0.0327
Puzzles	0.0310
Recall	0.0296
Foot Pointing	0.0281
Excitability	0.0263
Partnership	0.0244
Impossible Task Game	0.0238
Arm Pointing	0.0238
Memory versus Smell	0.0235
Navigation Learning	0.0226
Obedience	0.0169
Delayed Cup Game	0.0158
Inferential Reasoning Game	0.0157
Numerosity Warm-Up	0.0116
Two Cup Warm-up	0.0086
Physical Reasoning Game	0.0065
Memory versus Pointing	0.0057
Shared Perspective	0.0000

Expression Game	0.0000
5 vs 1 Game	0.0000
Self Control Game	0.0000
Slide	0.0000
Smell Game	0.0000
1 vs 1 Game	0.0000
3 vs 1 Game	0.0000
Shaker Game	0.0000

Question 6: Write a query that outputs the total number of unique User_Guids in each combination of State and ZIP code (postal code) in the United States, sorted first by state name in ascending alphabetical order, and second by total number of unique User_Guids in descending order (your first state should be AE and there should be 5043 rows in total in your output).

In [26]:
```
%%sql
SELECT state, zip, COUNT(user_guid) AS num_users
FROM users
WHERE country = 'US'
GROUP BY state, zip
ORDER BY state ASC, zip DESC
LIMIT 5
```

```
5 rows affected.
```

Out[26]:

state	zip	num_users
AE	9845	1
AE	9469	1
AE	9128	3
AE	9107	1
AE	9053	1

Question 7: Write a query that outputs the total number of unique User_Guids in each combination of State and ZIP code in the United States *that have at least 5 users*, sorted first by state name in ascending alphabetical order, and second by total number of unique User_Guids in descending order (your first state/ZIP code combination should be AZ/86303).

In [28]:
```
%%sql
SELECT state, zip, COUNT(DISTINCT user_guid) AS num_users
FROM users
WHERE country = 'US'
GROUP BY state, zip
HAVING COUNT(DISTINCT user_guid) >= 5
ORDER BY state ASC, zip DESC
LIMIT 10
```

10 rows affected.

Out[28]:

state	zip	num_users
AZ	86303	14
AZ	85749	5
AZ	85718	6
AZ	85711	5
AZ	85260	5
AZ	85254	5
AZ	85253	5
CA	95818	5
CA	95762	6
CA	95476	5

Be sure to watch the next video before beginning Exercise 6, the next set of MySQL exercises, and feel free to practice any other queries you wish below!

In [ ]:
```
```