# MySQL_Exercise_10_Useful_Logical_Functions

December 2, 2020

## 1 MySQL Exercise 10: Useful Logical Operators

There are a few more logical operators we haven't covered yet that you might find useful when designing your queries. Expressions that use logical operators return a result of "true" or "false", depending on whether the conditions you specify are met. The "true" or "false" results are usually used to determine which, if any, subsequent parts of your query will be run. We will discuss the IF operator, the CASE operator, and the order of operations within logical expressions in this lesson.

**Begin by loading the sql library and database, and making the Dognition database your default database:**

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

```
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

[1]:
```
[]
```

### 1.1 1. IF expressions

IF expressions are used to return one of two results based on whether inputs to the expressions meet the conditions you specify. They are frequently used in SELECT statements as a compact way to rename values in a column. The basic syntax is as follows:

```
IF([your conditions],[value outputted if conditions are met],[value outputted if conditions are
```

So we could write:

```
SELECT created_at, IF(created_at<'2014-06-01','early_user','late_user') AS user_type
FROM users
```

to output one column that provided the time stamp of when a user account was created, and a second column called user_type that used that time stamp to determine whether the user was an early or late user. User_type could then be used in a GROUP BY statement to segment summary calculations (in database systems that support the use of aliases in GROUP BY statements).

For example, since we know there are duplicate user_guids in the user table, we could combine a subquery with an IF statement to retrieve a list of unique user_guids with their classification as either an early or late user (based on when their first user entry was created):

```
SELECT cleaned_users.user_guid as UserID,
       IF(cleaned_users.first_account<'2014-06-01','early_user','late_user') AS user_type
FROM (SELECT user_guid, MIN(created_at) AS first_account
      FROM users
      GROUP BY user_guid) AS cleaned_users
```

We could then use a GROUP BY statement to count the number of unique early or late users:

```
SELECT IF(cleaned_users.first_account<'2014-06-01','early_user','late_user') AS user_type,
       COUNT(cleaned_users.first_account)
FROM (SELECT user_guid, MIN(created_at) AS first_account
      FROM users
      GROUP BY user_guid) AS cleaned_users
GROUP BY user_type
```

**Try it yourself:**

[2]:
```
%%sql
SELECT IF(cleaned_users.first_account<'2014-06-01','early_user','late_user') AS␣
 ↪user_type,
       COUNT(cleaned_users.first_account)
FROM (SELECT user_guid, MIN(created_at) AS first_account
      FROM users
      GROUP BY user_guid) AS cleaned_users
GROUP BY user_type;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

[2]:
```
[('early_user', 14470), ('late_user', 18723)]
```

**Question 1: Write a query that will output distinct user_guids and their associated country of residence from the users table, excluding any user_guids that have NULL values. You should get 16,261 rows in your result.**

[ ]:
```
%%sql
SELECT DISTINCT user_guid, country
FROM users
WHERE country IS NOT NULL
ORDER BY COUNTRY ASC;
```

**Question 2: Use an IF expression and the query you wrote in Question 1 as a subquery to determine the number of unique user_guids who reside in the United States (abbreviated "US") and outside of the US.**

[ ]:
```
%%sql
SELECT user_guid, IF(country="US", 'US', 'notUS') as residence
FROM (SELECT DISTINCT user_guid, country
FROM users
WHERE country IS NOT NULL
ORDER BY COUNTRY ASC) as u
ORDER BY residence;
```

Single IF expressions can only result in one of two specified outputs, but multiple IF expressions can be nested to result in more than two possible outputs. When you nest IF expressions, it is important to encase each IF expression–as well as the entire IF expression put together–in parentheses.

For example, if you examine the entries contained in the non-US countries category, you will see that many users are associated with a country called "N/A." "N/A" is an abbreviation for "Not Applicable"; it is not a real country name. We should separate these entries from the "Outside of the US" category we made earlier. We could use a nested query to say whenever "country" does not equal "US", use the results of a second IF expression to determine whether the outputed value should be "Not Applicable" or "Outside US." The IF expression would look like this:

```
IF(cleaned_users.country='US','In US', IF(cleaned_users.country='N/A','Not Applicable','Outside
```

Since the second IF expression is in the position within the IF expression where you specify "value outputted if conditions are not met," its two possible outputs will only be considered if cleaned_users.country='US' is evaluated as false.

The full query to output the number of unique users in each of the three groups would be:

```
SELECT IF(cleaned_users.country='US','In US',
          IF(cleaned_users.country='N/A','Not Applicable','Outside US')) AS US_user,
      count(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
      FROM users
      WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user
```

**Try it yourself. You should get 5,642 unique user_guids in the "Not Applicable" category, and 1,263 users in the "Outside US" category.**

[10]:
```
%%sql
SELECT IF(cleaned_users.country='US','In US',
          IF(cleaned_users.country='N/A','Not Applicable','Outside US')) AS␣
 ↪US_user,
      count(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
      FROM users
      WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
3 rows affected.
```

```
[10]: [('In US', 9356), ('Not Applicable', 5642), ('Outside US', 1263)]
```

The IF function is not supported by all database platforms, and some spell the function as IIF rather than IF, so be sure to double-check how the function works in the platform you are using.

If nested IF expressions seem confusing or hard to read, don't worry, there is a better function available for situations when you want to use conditional logic to output more than two groups. That function is called CASE.

## 1.2 2. CASE expressions

The main purpose of CASE expressions is to return a singular value based on one or more conditional tests. You can think of CASE expressions as an efficient way to write a set of IF and ELSEIF statements. There are two viable syntaxes for CASE expressions. If you need to manipulate values in a current column of your data, you would use this syntax:

Using this syntax, our nested IF statement from above could be written as:

```
SELECT CASE WHEN cleaned_users.country="US" THEN "In US"
            WHEN cleaned_users.country="N/A" THEN "Not Applicable"
            ELSE "Outside US"
            END AS US_user,
      count(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
      FROM users
      WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user
```

**Go ahead and try it:**

```
[12]: %%sql
SELECT CASE WHEN cleaned_users.country="US" THEN "In US"
            WHEN cleaned_users.country="N/A" THEN "Not Applicable"
            ELSE "Outside US"
            END AS US_user,
      COUNT(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
      FROM users
      WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
3 rows affected.
```

```
[12]: [('In US', 9356), ('Not Applicable', 5642), ('Outside US', 1263)]
```

Since our query does not require manipulation of any of the values in the country column, though, we could also take advantage of this syntax, which is slightly more compact:

Our query written in this syntax would look like this:

```
SELECT CASE cleaned_users.country
            WHEN "US" THEN "In US"
            WHEN "N/A" THEN "Not Applicable"
            ELSE "Outside US"
            END AS US_user,
      count(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
      FROM users
      WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user
```

**Try this query as well:**

[13]:
```
%%sql
SELECT CASE cleaned_users.country
            WHEN "US" THEN "In US"
            WHEN "N/A" THEN "Not Applicable"
            ELSE "Outside US"
            END AS US_user,
      COUNT(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
      FROM users
      WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
3 rows affected.
```

[13]:
```
[('In US', 9356), ('Not Applicable', 5642), ('Outside US', 1263)]
```

There are a couple of things to know about CASE expressions:

- Make sure to include the word END at the end of the expression
- CASE expressions do not require parentheses
- ELSE expressions are optional
- If an ELSE expression is omitted, NULL values will be outputted for all rows that do not meet any of the conditions stated explicitly in the expression
- CASE expressions can be used anywhere in a SQL statement, including in GROUP BY, HAVING, and ORDER BY clauses or the SELECT column list.

You will find that CASE statements are useful in many contexts. For example, they can be used to rename or revise values in a column.

**Question 3: Write a query using a CASE statement that outputs 3 columns: dog_guid, dog_fixed, and a third column that reads "neutered" every time there is a 1 in the "dog_fixed" column of dogs, "not neutered" every time there is a value of 0 in the "dog_fixed" column of dogs, and "NULL" every time there is a value of anything else in the "dog_fixed" column. Limit your results for troubleshooting purposes.**

[32]:
```
%%sql
SELECT dog_guid, dog_fixed,
    CASE dog_fixed
        WHEN "1" THEN "neutered"
        WHEN "0" THEN "Not neutered"
    END AS dog_condition
FROM dogs
LIMIT 100;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
100 rows affected.
```

[32]:
```
[('fd27b272-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27b5ba-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27b6b4-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
 ('fd27b79a-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
 ('fd27b86c-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
 ('fd27b948-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27ba1a-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27bbbe-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27c1c2-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27c5be-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27c74e-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27c7d0-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27c852-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27c8d4-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27c956-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27cb72-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27cd98-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27ce1a-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27cea6-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27cf28-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27cfaa-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27d02c-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
 ('fd27d0b8-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27d144-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27d1c6-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27d248-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27d2ca-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27d34c-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27d3d8-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27d45a-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27d4dc-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27d770-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27d9fa-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27db08-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27db8a-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27dc52-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27dd38-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27e026-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
 ('fd27e0d0-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27e1e8-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
 ('fd27e31e-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
 ('fd27e454-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27e580-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27e9a4-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
 ('fd27eae4-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
 ('fd27ed46-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
 ('fd27efb2-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27f110-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27f25a-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27f4c6-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27f732-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27f868-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd27f9a8-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd28010a-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
 ('fd280236-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd280344-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd280826-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd2808b2-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd28093e-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd2809c0-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3ccd24-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3ccf2c-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cd40e-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
 ('fd3cd4d6-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cd8d2-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cd99a-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cec50-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cf5c4-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cf678-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cf718-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cf8ee-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cf984-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cfa1a-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cfab0-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cfcfe-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cfd94-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cfe2a-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cfeb6-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3cff4c-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3d0078-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3d01ae-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
 ('fd3d03fc-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d0492-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d05be-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d064a-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d06e0-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d0776-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d080c-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d0898-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d0938-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d09ce-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d0b7c-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d0c12-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d0cb2-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d0d48-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d0dde-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d0f00-7144-11e5-ba71-058fbc01cf0b', 0, 'Not neutered'),
  ('fd3d0f96-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d102c-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered'),
  ('fd3d10cc-7144-11e5-ba71-058fbc01cf0b', 1, 'neutered')]
```

You can also use CASE statements to standardize or combine several values into one.

**Question 4: We learned that NULL values should be treated the same as "0" values in the exclude columns of the dogs and users tables. Write a query using a CASE statement that outputs 3 columns: dog_guid, exclude, and a third column that reads "exclude" every time there is a 1 in the "exclude" column of dogs and "keep" every time there is any other value in the exclude column. Limit your results for troubleshooting purposes.**

[ ]:
```
%%sql
SELECT dog_guid, exclude,
    CASE exclude
        WHEN "1" THEN "exclude"
        ELSE "keep"
    END AS cond
FROM dogs
LIMIT 100;
```

**Question 5: Re-write your query from Question 4 using an IF statement instead of a CASE statement.**

[ ]:
```
%%sql
SELECT dog_guid, exclude, IF(exclude="1", "exclude", "keep") as cond
FROM dogs
LIMIT 100;
```

Case expressions are also useful for breaking values in a column up into multiple groups that meet specific criteria or that have specific ranges of values.

**Question 6: Write a query that uses a CASE expression to output 3 columns:**

**dog_guid, weight, and a third column that reads...**
**"very small" when a dog's weight is 1-10 pounds**
**"small" when a dog's weight is greater than 10 pounds to 30 pounds**
**"medium" when a dog's weight is greater than 30 pounds to 50 pounds**
**"large" when a dog's weight is greater than 50 pounds to 85 pounds**
**"very large" when a dog's weight is greater than 85 pounds**
**Limit your results for troubleshooting purposes.**

[ ]:
```
%%sql
SELECT dog_guid, weight,
    CASE
        WHEN weight <= 10 THEN "very small"
        WHEN weight > 10 && weight <= 30 THEN "small"
        WHEN weight > 30 && weight <= 50 THEN "medium"
        WHEN weight > 50 && weight <= 85 THEN "large"
        WHEN weight > 85 THEN "very large"
    END as size
FROM dogs
LIMIT 100;
```

### 1.3 3. Pay attention to the order of operations within logical expressions

As you started to see with the query you wrote in Question 6, CASE expressions often end up needing multiple AND and OR operators to accurately describe the logical conditions you want to impose on the groups in your queries. You must pay attention to the order in which these operators are included in your logical expressions, because unless parentheses are included, the NOT operator is always evaluated before an AND operator, and an AND operator is always evaluated before the OR operator.

When parentheses are included, the expressions within the parenthese are evaluated first. That means this expression:

`CASE WHEN "condition 1" OR "condition 2" AND "condition 3"...`

will lead to different results than this expression:

`CASE WHEN "condition 3" AND "condition 1" OR "condition 2"...`

or this expression:

`CASE WHEN ("condition 1" OR "condition 2") AND "condition 3"...`

In the first case you will get rows that meet condition 2 and 3, or condition 1. In the second case you will get rows that meet condition 1 and 3, or condition 2. In the third case, you will get rows that meet condition 1 or 2, and condition 3.

Let's see a concrete example of how the order in which logical operators are evaluated affects query results.

**Question 7: How many distinct dog_guids are found in group 1 using this query?**

```
SELECT COUNT(DISTINCT dog_guid),
CASE WHEN breed_group='Sporting' OR breed_group='Herding' AND exclude!='1' THEN "group 1"
     ELSE "everything else"
     END AS groups
FROM dogs
GROUP BY groups
```

[46]:
```
%%sql
SELECT COUNT(DISTINCT dog_guid),
CASE WHEN breed_group='Sporting' OR breed_group='Herding' AND exclude!='1' THEN␣
↪"group 1"
     ELSE "everything else"
     END AS groups
FROM dogs
GROUP BY groups;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

[46]:
```
[(30179, 'everything else'), (4871, 'group 1')]
```

**Question 8: How many distinct dog_guids are found in group 1 using this query?**

```
SELECT COUNT(DISTINCT dog_guid),
CASE WHEN exclude!='1' AND breed_group='Sporting' OR breed_group='Herding' THEN "group 1"
     ELSE "everything else"
     END AS group_name
FROM dogs
GROUP BY group_name
```

[47]:
```
%%sql
SELECT COUNT(DISTINCT dog_guid),
CASE WHEN exclude!='1' AND breed_group='Sporting' OR breed_group='Herding' THEN␣
↪"group 1"
     ELSE "everything else"
     END AS group_name
FROM dogs
GROUP BY group_name;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

[47]:
```
[(31589, 'everything else'), (3461, 'group 1')]
```

**Question 9: How many distinct dog_guids are found in group 1 using this query?**

```
SELECT COUNT(DISTINCT dog_guid),
CASE WHEN exclude!='1' AND (breed_group='Sporting' OR breed_group='Herding') THEN "group 1"
     ELSE "everything else"
     END AS group_name
FROM dogs
GROUP BY group_name
```

[48]:
```
%%sql
SELECT COUNT(DISTINCT dog_guid),
CASE WHEN exclude!='1' AND (breed_group='Sporting' OR breed_group='Herding')␣
↪THEN "group 1"
     ELSE "everything else"
     END AS group_name
FROM dogs
GROUP BY group_name;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

[48]:
```
[(35004, 'everything else'), (46, 'group 1')]
```

So make sure you always pay attention to the order in which your logical operators are listed in your expressions, and whenever possible, include parentheses to ensure that the expressions are evaluated in the way you intend!

### 1.4 Let's practice some more IF and CASE statements

Question 10: For each dog_guid, output its dog_guid, breed_type, number of completed tests, and use an IF statement to include an extra column that reads "Pure_Breed" whenever breed_type equals 'Pure Breed" and "Not_Pure_Breed" whenever breed_type equals anything else. LIMIT your output to 50 rows for troubleshooting. HINT: you will need to use a join to complete this query.

[ ]:
```
%%sql
SELECT d.dog_guid, d.breed_type, COUNT(c.created_at) as CantTests,
        IF(d.breed_type = "Pure Breed", "Pure Breed", "Not Pure Breed") as␣
↪breedP
FROM dogs d, complete_tests c
WHERE d.dog_guid = c.dog_guid
GROUP BY d.dog_guid, d.breed_type, breedP
LIMIT 50;
```

Question 11: Write a query that uses a CASE statement to report the number of unique user_guids associated with customers who live in the United States and who are in the following groups of states:

Group 1: New York (abbreviated "NY") or New Jersey (abbreviated "NJ")
Group 2: North Carolina (abbreviated "NC") or South Carolina (abbreviated "SC")
Group 3: California (abbreviated "CA")
Group 4: All other states with non-null values

You should find 898 unique user_guids in Group1.

[59]:
```
%%sql
SELECT COUNT(DISTINCT user_guid),
    CASE
        WHEN state IN ("NY", "NJ") THEN "group 1"
        WHEN state IN ("Nc", "SC") THEN "group 2"
        WHEN state = "CA" THEN "group 3"
        ELSE "group 4"
    END as groups
FROM users
WHERE country = "US" AND state IS NOT NULL
GROUP BY groups;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

[59]:
```
[(898, 'group 1'), (653, 'group 2'), (1417, 'group 3'), (6388, 'group 4')]
```

**Question 12: Write a query that allows you to determine how many unique dog_guids are associated with dogs who are DNA tested and have either stargazer or socialite personality dimensions. Your answer should be 70.**

[60]:
```
%%sql
SELECT COUNT(DISTINCT dog_guid)
FROM dogs
WHERE dna_tested = 1 AND dimension IN("socialite", "stargazer");
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

[60]:
```
[(70,)]
```

**Feel free to practice any other queries you like here!**

[ ]:
```
```