02 Functions

SQL also has the ability to transform or perform computations on the data in our tables. We use functions to do this.

This will be a new type of item to work with. We've used `()` before to group queries, but in this context they will have a different meaning. You use these functions inside of your SELECT statement items, and they will act upon the contents of that column.

`MIN()` and `MAX()`

For example, our FolderNumber column has many different numbers in it and appears to be incrementing up. The data is already sorted pretty well, so we can scroll to the bottom and top to see what we presume are the minimum and maximum values. But are we sure? There are two options for this:

1. we could select all the data, sort it by FolderNumber, and then look at the top and bottom. For some smaller data this might be just fine! But this isn't going to work well for 600 rows, or working with databases too big to display all at one.
2. You can directly compute the `min` and `max` of the column FolderNumber.

```
SELECT MIN(FolderNumber) FROM pettigrew;
```

This results in 1, and knowing that our data that seems right. Now let's try `MAX`.

```
SELECT MAX(FolderNumber) FROM pettigrew;
```

This gives us 99, which we know to be incorrect. That's because we read in the entire data file as text. Strings are sorted differently from numbers, and from that perspective "99" would come before "899" because "9" is greater than "8". Again, this is a sitution where knowing your data well will save you a lot of trouble.

We could either go back and change how we read in the data to say that this column is a string, or we can change it into numerical as part of our SQL statement.

`CAST()`

You may not always have control over how your data is read in and how the tables are created, or you may not want to alter the database that you have been given to work with. Most of the time you'll be in situations where coded values that are represented with integers were automatically read is as numerical, and you need to

change them to be text. Imagine a zip code here. It is composed of numbers, but doesn't really have a "numerical" meaning.

However, here we have the opposite. We have a column of numbers that were read in as text. Consequently, sorting and calculation operations are not behaving as desired.

We can use the `CAST()` function within our select statement to transform this on the fly. This function will operate on a single column, and has the syntax: `CAST(column_name as data_type`. You can read more about recasting optios in the documentaiton.

Consider our original select statement that yielded a result of `"99"`: `SELECT MAX(FolderNumber) FROM pettigrew;`. In this case we need to transform `FolderNumber` into a numerical data type. There are complex differences between the data types for numbers, which will not be discussed here. Let's just change this number into a `numeric` type. So we need to place `CAST(FolderNumber as numeric)` into our select statement.

Let's first cast the column as numeric and check that the data all looks intact.

```
SELECT CAST(FolderNumber as numeric) FROM pettigrew;
```

Once these all look right, we can add our function around the column we want.

```
SELECT MAX(CAST(FolderNumber as numeric)) FROM pettigrew;
```

That's a better result, `610`.

AS keyword

By default, whatever piece of SQL code that creates that column in the SELECT statement area will end up as the column name in the table view. There may also be situations where you might want to change an exising column name to something more descriptive.

Either way, you can control the header name with the `AS` keyword. Place this after the column decleration in the `SELECT` area. Example: `BoxNumber as box`.

```
SELECT DISTINCT BoxNumber as box FROM pettigrew;
```

When we run this, we get the same contents as we have before, but now the column header is `box`. Let's add our COUNT() back in there.

```
SELECT COUNT(DISTINCT BoxNumber) as numberOfBoxes FROM pettigrew;
```

We can also make these aggregation column names much more specific, so we can remember what they

stand for. Sometimes aggregations or transformations can get lengthy, so you'll want to change them to make your results more readable.

Summary

We're going to be using these functions more in our next section on aggregation.

Next up

Aggregation