(http://datacarpentry.org)

Basic queries

Learning objectives

1. Write and build queries.
2. Filter data given various criteria.
3. Sort the results of a query.

Writing my first query

Let's start by using the **surveys** table. Here we have data on every individual that was captured at the site, including when they were captured, what plot they were captured on, their species ID, sex and weight in grams.

Let's write an SQL query that selects only the year column from the surveys table. SQL queries can be written in the box located under the "Execute SQL" tab. Click 'Run SQL' to execute the query in the box.

```
SELECT year
FROM surveys;
```

We have capitalized the words SELECT and FROM because they are SQL keywords. SQL is case insensitive, but it helps for readability, and is good style.

If we want more information, we can just add a new column to the list of fields, right after `SELECT`:

```
SELECT year, month, day
FROM surveys;
```

Or we can select all of the columns in a table using the wildcard *

```
SELECT *
FROM surveys;
```

Unique values

If we want only the unique values so that we can quickly see what species have been sampled we use `DISTINCT`

```
SELECT DISTINCT species_id
FROM surveys;
```

If we select more than one column, then the distinct pairs of values are returned

```
SELECT DISTINCT year, species_id
FROM surveys;
```

Calculated values

We can also do calculations with the values in a query. For example, if we wanted to look at the mass of each individual on different dates, but we needed it in kg instead of g we would use

```
SELECT year, month, day, weight/1000.0
FROM surveys;
```

When we run the query, the expression `weight / 1000.0` is evaluated for each row and appended to that row, in a new column. Expressions can use any fields, any arithmetic operators (`+`, `-`, `*`, and `/`) and a variety of built-in functions. For example, we could round the values to make them easier to read.

```
SELECT plot_id, species_id, sex, weight, ROUND(weight / 1000.0, 2)
FROM surveys;
```

> ## Challenge
>
> Write a query that returns the year, month, day, species_id and weight in mg

SOLUTION

```
SELECT day, month, year, species_id, weight * 1000
FROM surveys;
```

Filtering

Databases can also filter data – selecting only the data meeting certain criteria. For example, let's say we only want data for the species *Dipodomys merriami*, which has a species code of DM. We need to add a `WHERE` clause to our query:

```
SELECT *
FROM surveys
WHERE species_id='DM';
```

We can do the same thing with numbers. Here, we only want the data since 2000:

```
SELECT * FROM surveys
WHERE year >= 2000;
```

We can use more sophisticated conditions by combining tests with `AND` and `OR`. For example, suppose we want the data on *Dipodomys merriami* starting in the year 2000:

```
SELECT *
FROM surveys
WHERE (year >= 2000) AND (species_id = 'DM');
```

Note that the parentheses are not needed, but again, they help with readability. They also ensure that the computer combines `AND` and `OR` in the way that we intend.

If we wanted to get data for any of the *Dipodomys* species, which have species codes `DM`, `DO`, and `DS`, we could combine the tests using OR:

```
SELECT *
FROM surveys
WHERE (species_id = 'DM') OR (species_id = 'DO') OR (species_id = 'DS');
```

> ## Challenge
>
> Write a query that returns the day, month, year, species_id, and weight (in kg) for individuals caught on Plot 1 that weigh more than 75 g

SOLUTION

```
SELECT day, month, year, species_id, weight / 1000.0
FROM surveys
WHERE plot_id = 1
AND weight > 75;
```

Building more complex queries

Now, lets combine the above queries to get data for the 3 *Dipodomys* species from the year 2000 on. This time, let's use `IN` as one way to make the query easier to understand. It is equivalent to saying `WHERE (species_id = 'DM') OR (species_id = 'DO') OR (species_id = 'DS')`, but reads more neatly:

```
SELECT *
FROM surveys
WHERE (year >= 2000) AND (species_id IN ('DM', 'DO', 'DS'));
```

We started with something simple, then added more clauses one by one, testing their effects as we went along. For complex queries, this is a good strategy, to make sure you are getting what you want. Sometimes it might help to take a subset of the data that you can easily see in a temporary database to practice your queries on before working on a

larger or more complicated database.

When the queries become more complex, it can be useful to add comments. In SQL, comments are started by `--`, and end at the end of the line. For example, a commented version of the above query can be written as:

```
-- Get post 2000 data on Dipodomys' species
-- These are in the surveys table, and we are interested in all columns
SELECT * FROM surveys
-- Sampling year is in the column `year`, and we want to include 2000
WHERE (year >= 2000)
-- Dipodomys' species have the `species_id` DM, DO, and DS
AND (species_id IN ('DM', 'DO', 'DS'));
```

Although SQL queries often read like plain English, it is *always* useful to add comments; this is especially true of more complex queries.

Sorting

We can also sort the results of our queries by using `ORDER BY`. For simplicity, let's go back to the **species** table and alphabetize it by taxa.

First, let's look at what's in the **species** table. It's a table of the species_id and the full genus, species and taxa information for each species_id. Having this in a separate table is nice, because we didn't need to include all this information in our main **surveys** table.

```
SELECT *
FROM species;
```

Now let's order it by taxa.

```
SELECT *
FROM species
ORDER BY taxa ASC;
```

The keyword `ASC` tells us to order it in Ascending order. We could alternately use `DESC` to get descending order.

```
SELECT *
FROM species
ORDER BY taxa DESC;
```

`ASC` is the default.

We can also sort on several fields at once. To truly be alphabetical, we might want to order by genus then species.

```
SELECT *
FROM species
ORDER BY genus ASC, species ASC;
```

> ## Challenge
>
> Write a query that returns year, species_id, and weight in kg from the surveys table, sorted with the largest weights at the top.

SOLUTION

```
SELECT year, species_id, weight / 1000.0
FROM surveys ORDER BY weight DESC;
```

Order of execution

Another note for ordering. **We don't actually have to display a column to sort by it**. For example, let's say we want to order the birds by their species ID, but we only want to see genus and species.

```
SELECT genus, species
FROM species
WHERE taxa = 'Bird'
ORDER BY species_id ASC;
```

We can do this because **sorting occurs earlier in the computational pipeline** than field selection.

The computer is basically doing this:

1. Filtering rows according to `WHERE`
2. Sorting results according to `ORDER BY`
3. Displaying requested columns or expressions.

Clauses are written in a fixed order: `SELECT`, `FROM`, `WHERE`, then `ORDER BY`. It is possible to write a query as a single line, but for readability, we recommend to put each clause on its own line.

> ## Challenge
>
> Let's try to combine what we've learned so far in a single query. Using the surveys table write a query to display the three date fields, `species_id`, and weight in kilograms (rounded to two decimal places), for individuals captured in 1999, ordered alphabetically by the `species_id`. Write the query as a single line, then put each clause on its own line, and see how more legible the query becomes!

SOLUTION

```
SELECT year, month, day, species_id, ROUND(weight / 1000.0, 2)
FROM surveys
WHERE year = 1999
ORDER BY species_id;
```

Previous: SQL Introduction (00-sql-introduction.html) Next: SQL Aggregation. (02-sql-aggregation.html)