Powerful, free, open source database manager

Ron Campbell, *NBC* Liz Lucas, *Kaiser Health News*

MySQL is a good choice for any journalist who

- wants to use data to enhance their stories
- knows or wants to learn SQL (Structured Query Language)
- works in Microsoft Access but is looking for something more powerful, that handles larger datasets

Installation

We won't cover installation and setup in this class, but here's a quick overview: you need to download the MySQL Community Server, and you'll need an additional program to connect to your MySQL databases. MySQL Workbench (which comes with most MySQL installs), Navicat (which has robust importing options but costs $$), SQLYog (free and easy to use) are all examples of interfaces that make querying data in MySQL easier.

Structured Query Language

SQL is a standard language for querying data in database managers. If you've used Microsoft Access, SQL Server or PostgreSQL before, you may recognize that most SQL is pretty standard. MySQL adds some functions and different syntax that we'll cover here, but let's start with the basic statements:

SELECT
FROM
WHERE
GROUP BY
ORDER BY

SELECT – think of this as a vertical slice of your data; you tell MySQL which columns from your data you want to see in your query results, or you can use the asterisk to select everything (ex: SELECT geography, total, native, noncitizen)

FROM – this just tells MySQL which table you're pulling data from (ex: FROM FL_nativity)

WHERE – this is a filter, or a horizontal slice of your data; you set criteria that has to be true in order for rows to be returned (ex: WHERE noncitizen = 311)
GROUP BY – this groups rows together by a certain field or fields so that you can do math on those groups, such as counting the number of records in each group or summing up the numbers in a field.
(ex: GROUP BY geography)

* this is paired with putting an aggregate function in your SELECT statement, such as COUNT(*) or SUM(noncitizen)
(ex: SELECT geography, COUNT(*)
FROM FL_nativity
GROUP BY geography)

ORDER BY – this sorts your results by a field or fields, ascending (*default*) or descending (DESC)
(ex: ORDER BY native DESC)

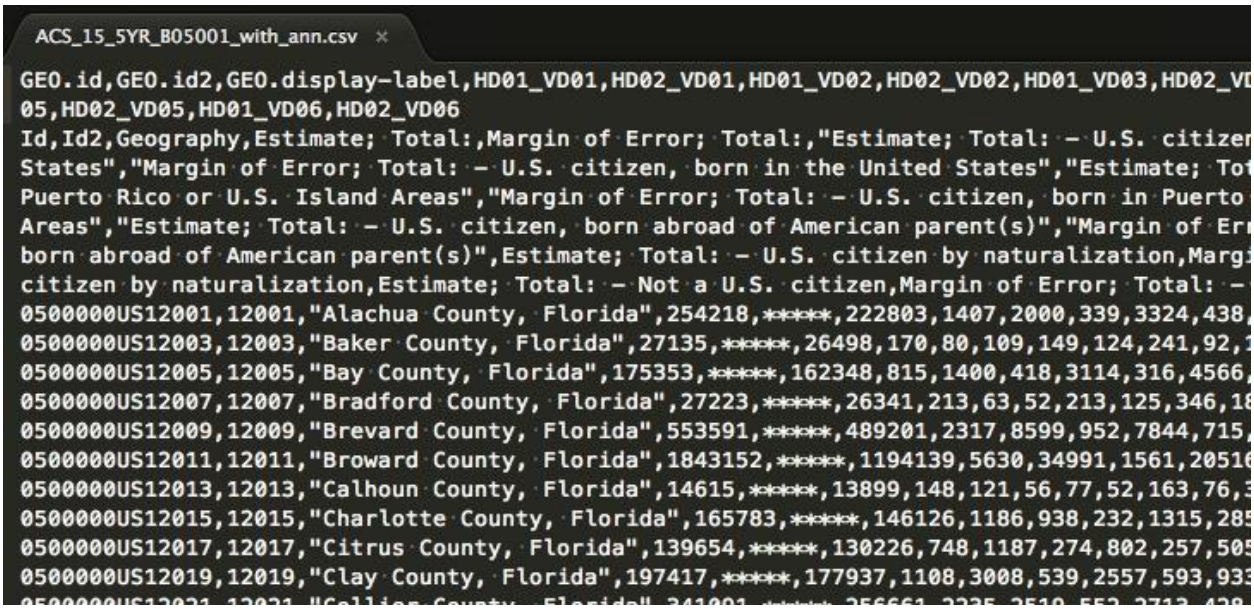
Some important notes SQL:

- the statements SELECT and FROM are mandatory, the rest are optional depending on what you want your results to look like
- the statements must always be in the order above, even if you're not using all of them
- SQL is *not* case sensitive (in MySQL)
- in the WHERE statement, when you're setting criteria for fields that are text, you must use quotes around the value you're looking for (ex: WHERE county = 'JACKSON'). Dates also require quotes; numbers do not.

We'll eventually be working with two different tables of data that go together, so we'll use SQL to **join** these two tables together. To do this, we'll add two statements to the list above:
INNER JOIN – list the second table you'd like to join to the table in FROM
(ex: INNER JOIN FL_poverty)
ON – tell MySQL what fields are supposed to match, so it nows how to bring the two tables together. To be clear which table each field comes from, add the tablename and a period before the fieldname
(ex: ON FL_nativity.ID2 = FL_poverty.ID2)

Importing

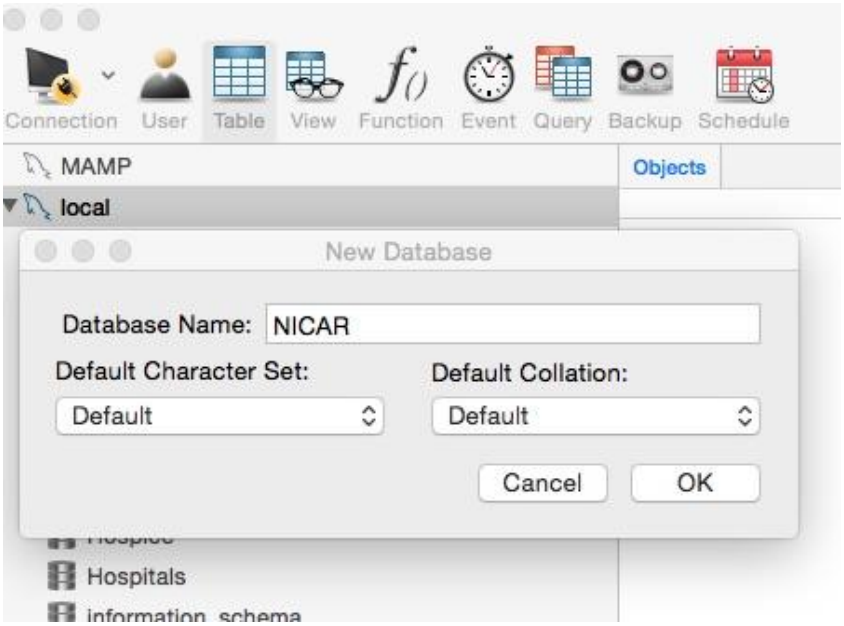
In this class we'll use Navicat to import some data from the U.S. Census Bureau: FL_nativity.csv (B05001). It has two field names on the first two lines; data begins on line 3:



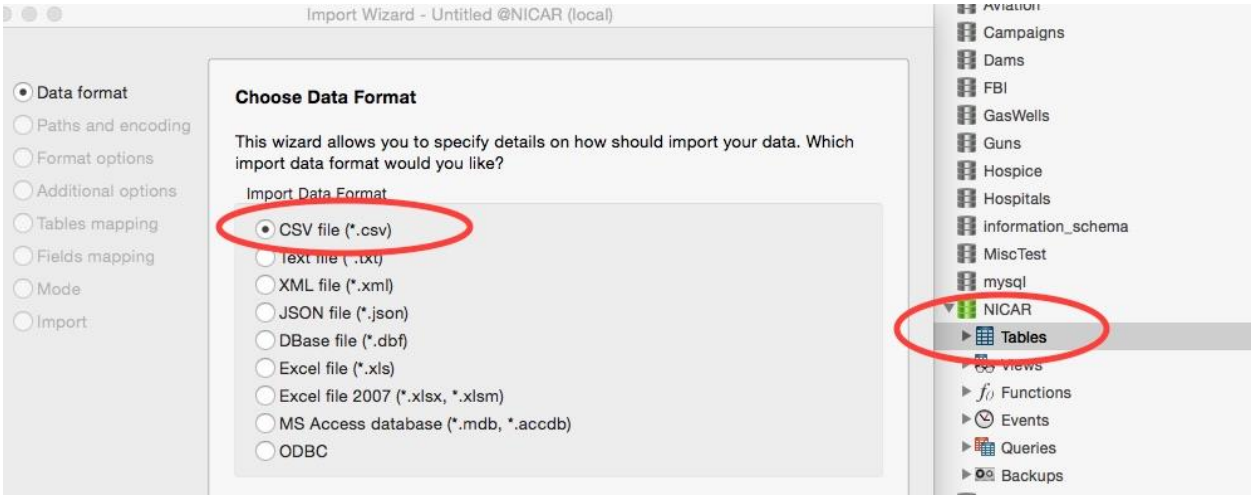
Note that each field is separated by a comma (the “delimiter”), and some fields have double quotation marks around them (the “text qualifier”).

Before we can import this file, we have to create a database in MySQL to hold it.
MySQL operates differently than Microsoft Access in that it puts all of your databases in a single place on your computer that you access through Navicat (or whatever interface you’re using). Each database is not a file that you move around. SQL Server and PostgreSQL also operate this way.

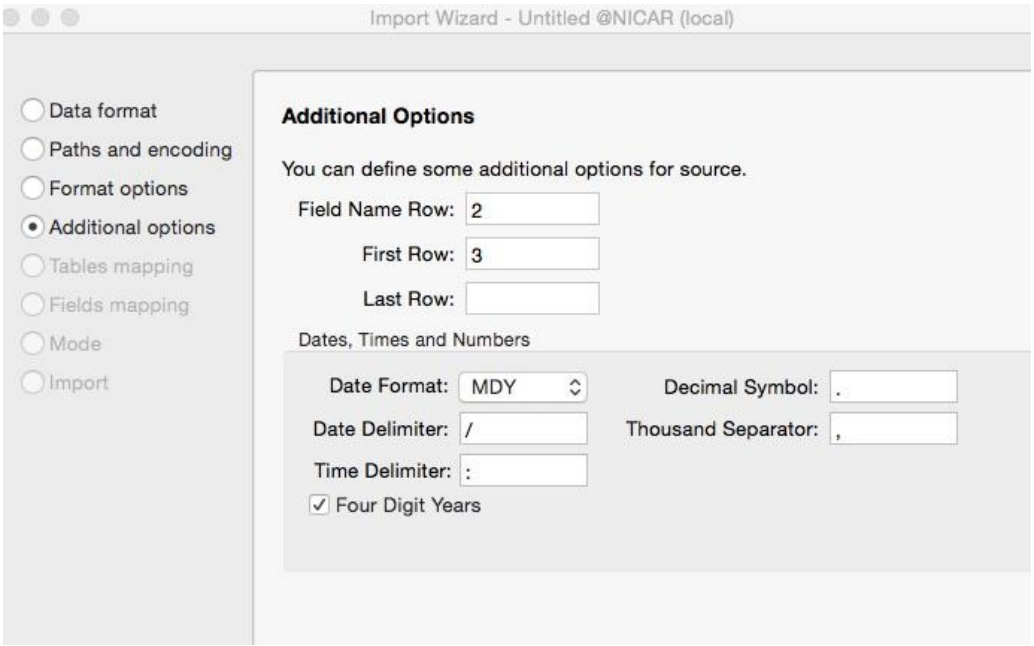
Create a new database. In Navicat, go to the Connection menu, click to drop down, select New Database.



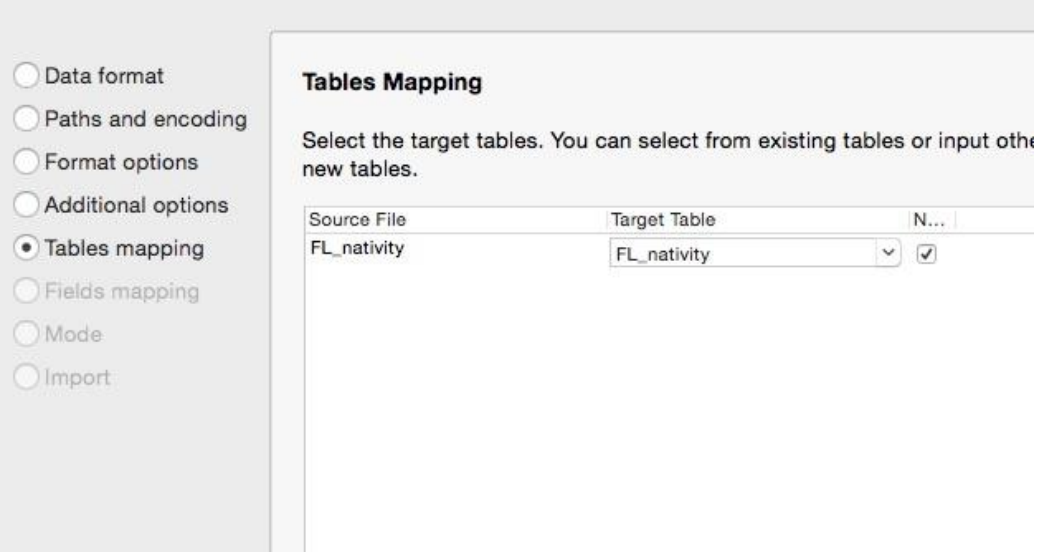
With the new database highlighted, double-click to display contents and click on Table. Then go to the Navicat File menu and click on the Import Wizard. CSV file is the default. Click “Continue” (not shown) for the next step. Navigate to find the file. The next two steps are easy.



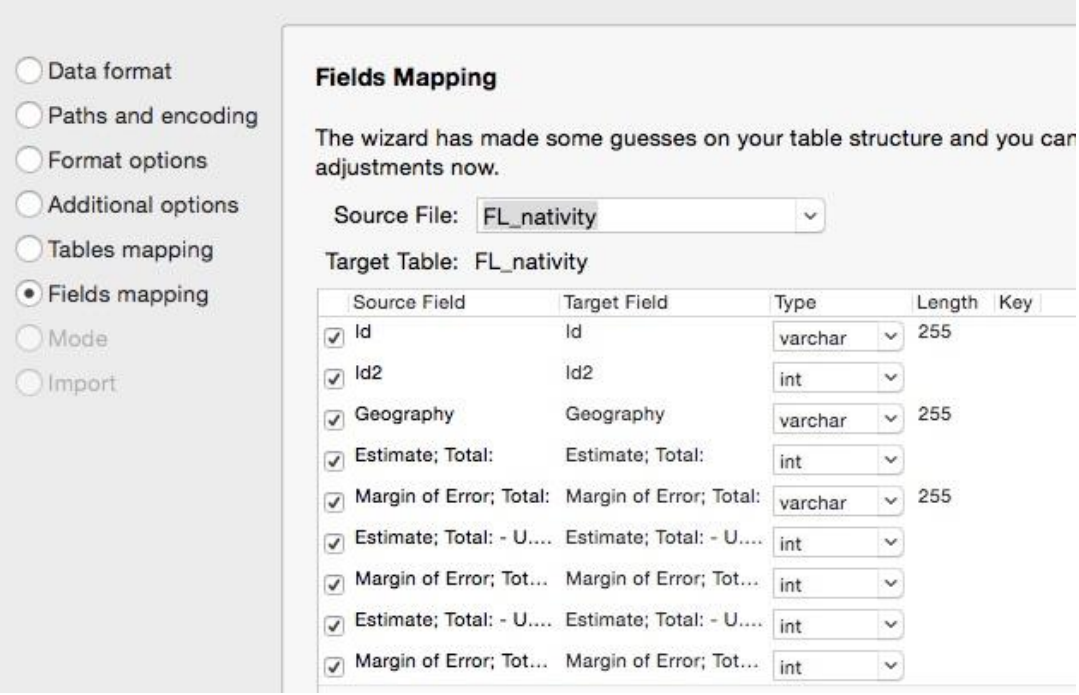
Here is where it gets tricky. If you looked at the file in a text editor, though, you know what to do. The field row names are in Row 2. “First row” refers to the row where the data starts – Row 3. It’s rare to specify the last row. There are no dates, so don’t worry about the date format; no dollar signs or thousand separators either.



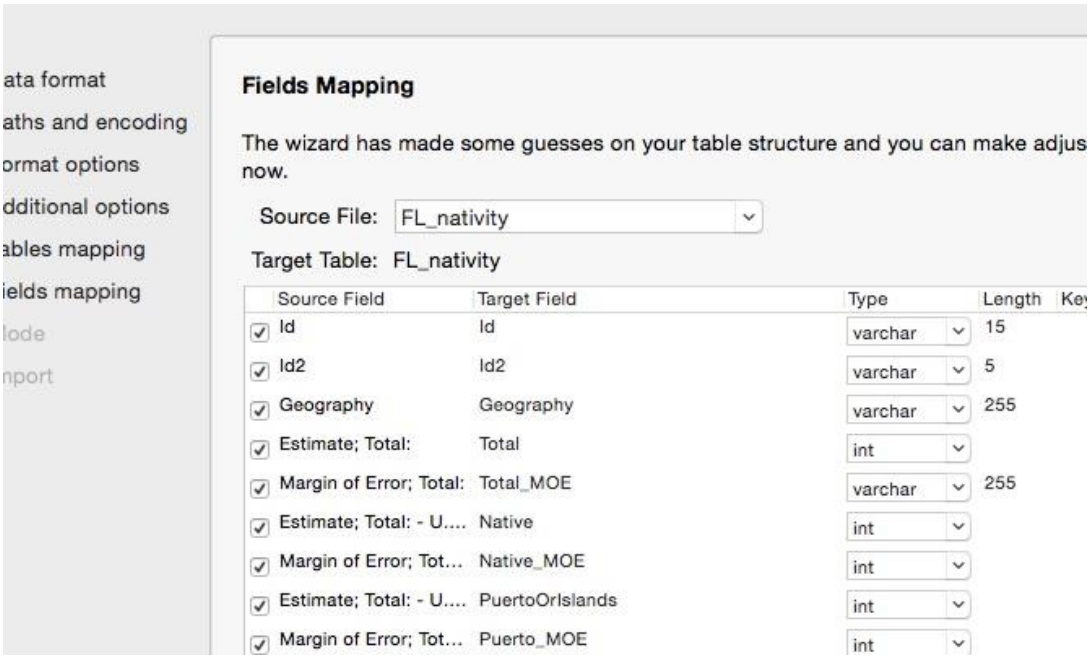
Name the file. We’ve made it easy and named it already, but often the text file comes with a name that only a techie could love.



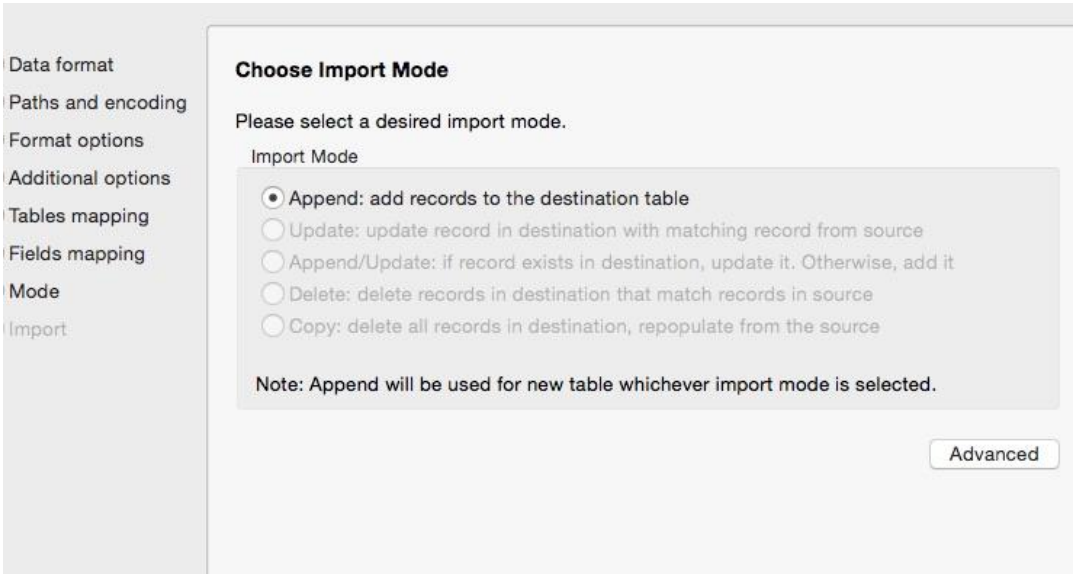
Here's the "Before" shot. Time to specify field types, lengths and field names. Field names can contain no spaces. Pro tip: Assign names whose meaning will still be clear to you months later.



The "After" shot. We've eliminated all spaces and standardized names. Notice that Id2 now is varchar instead of int (integer); notice too that both of the first two fields are much shorter than the standard varchar length; they're approximately as long as the actual length in the data.

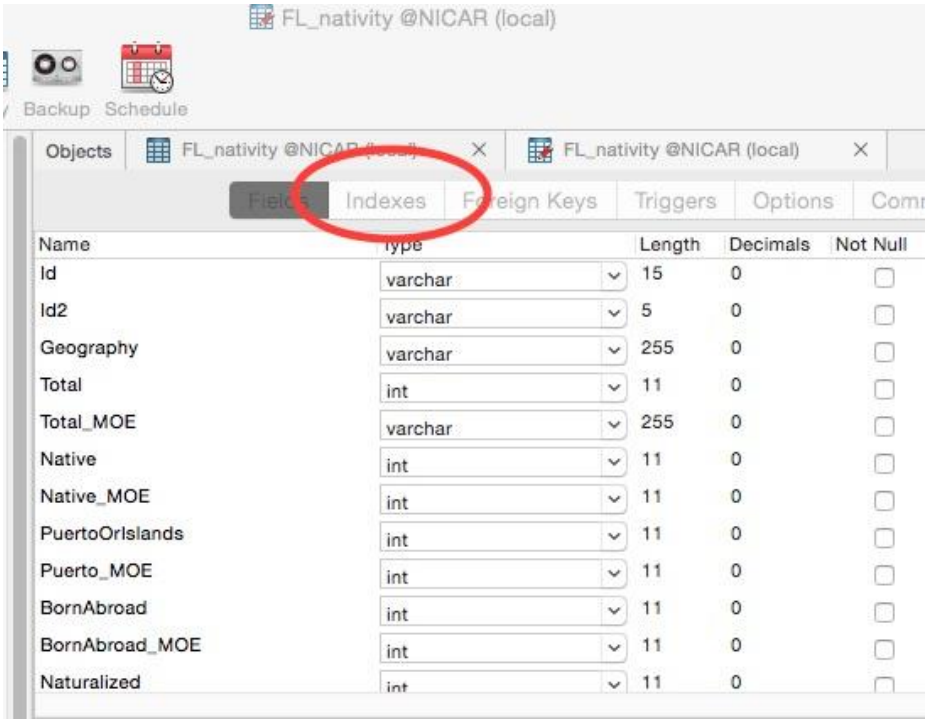


The final, Do you really want to do this, warning before you import. On the next screen, you'll be presented with a blue "Start" button. And then the data imports. Either it imports or you get an error. If it imports, you hit "Close" to exit.

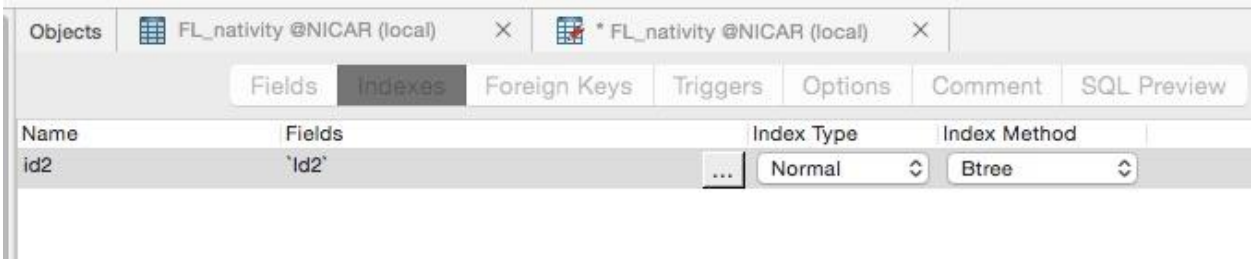


The table now appears in the list of tables in our database. Click on it and it will appear in the window to the right.

One last task – with the table highlighted, go to File in the Navicat menu, and click on Design Table. The Fields tab is highlighted; click on the Indexes tab immediately to the right.



The index screen will be blank. Go to the bottom left; you'll see a "+" and a "-" sign; click on the "+". This will bring up a line at the top. Enter "id2" as the name, "Btree" as one of the two index methods and, in the dropdown for fields, select the second field, id2. Click on the Save icon at the bottom. The table is now imported, indexed and ready to use.



Asking questions of the data

What's in the data? Do a basic query that selects all of the data so that you can take a look at it.

```
SELECT *
FROM FL_nativity
```

Take a look at all of the fields. What ones are you particularly interested in? Narrow your query so you can take a closer look at certain aspects of the data.

```
SELECT geography, total, native, naturalized, noncitizen
FROM fl_nativity
```

You can narrow your results even further by using the WHERE to home in on records of particular interest.

```
SELECT geography, total, native, naturalized, noncitizen
FROM fl_nativity
WHERE noncitizen= 311
```

While the "=" is a common operator, there are other ways to create filters: >, <, <>.

```
WHERE geography <> 'Union County, Florida'
WHERE noncitizen > naturalized
WHERE naturalized > noncitizen
WHERE noncitizen > native
WHERE naturalized > native
WHERE (naturalized + noncitizen) > native
```

Occasionally you'll want to add columns of your own, to specify something interesting in the data that isn't there.

1. Click on FL_nativity icon, then go to File | Design table.
2. In the fields tab, find the + sign just below the list of field names
3. Click once to add a field. Name it "Immigrant", type int, length 11.
4. Click on the + sign again. Name this field "Immigrant_per", type float, length 6, decimal
5. Save and close the design pane.
6. Close FL_nativity and double-click to re-open.
7. Click on the Query icon and open a query window.

Once you add the field, you can populate that field using the UPDATE query:

```
UPDATE FL_nativity
SET immigrant = (naturalized + noncitizen)
```

You may have noticed that this deviates from our basic SQL. This is SQL that *changes* the data, doesn't just SELECT it. So in this query, SELECT and FROM are not mandatory. There are three different types of queries that deviate from the SELECT, FROM pattern: queries that CREATE, UPDATE, or DELETE.

```
UPDATE FL_nativity
SET immigrant_per = (immigrant / total) * 100
```

Close FL_nativity and re-open. The Immigrant and Immigrant_per columns have been filled.

```
SELECT geography, total, native, immigrant, immigrant_per
FROM FL_nativity
WHERE immigrant_per >= 10
```

In order to see the most interesting results at the top, try sorting your results by adding one of the following ORDER BY statements:

```
ORDER BY immigrant_per
ORDER BY immigrant_per DESC
```

More queries, and joining

We'll switch datasets now to work with a small slice of campaign contributions from the Federal Election Commission's data. We'll have to load the data, but MySQL has a handy SQL statement called LOAD DATA that allows us to script imports from text files.

See the **load_indiv.sql** and **load_cand.sql** files in the repo.

For the queries to this data, look at the **querying_campaign_finance.sql** file.