Basic SQL joining exercise using SQL Lite
Using Census data on housing units, by place
Created by @MaryJoWebster
January 2017

The goal of this exercise is to introduce how joining tables works in SQL. We're going to use two tables from the Census Bureau with the number of housing units in each "place" (incorporated cities) in Minnesota – one has data from 2000 and the other has data from 2013. I want to put them in the same file so it's easier to calculate the percentage change in housing units between the two points in time. Doing this kind of matching in a SQL software is much easier than trying to line up two sets of data in Excel, especially in this situation because there are "places" that are in one of the datasets, but not in the other.

We're going to start with two .csv files that need to be imported to SQL Lite – housing2000.csv and housing2013.csv

Create a new database and then import each file, naming them "housing2000" and "housing2013"

Let's look at the data. Both tables have a field called "geography," where it lists the name and the state; Both also have a field for the number of housing units. The 2013 data also has a MOE (Margin of error) field, because this data came from the American Community Survey 5-year data. The 2000 data is from the decennial census that year.

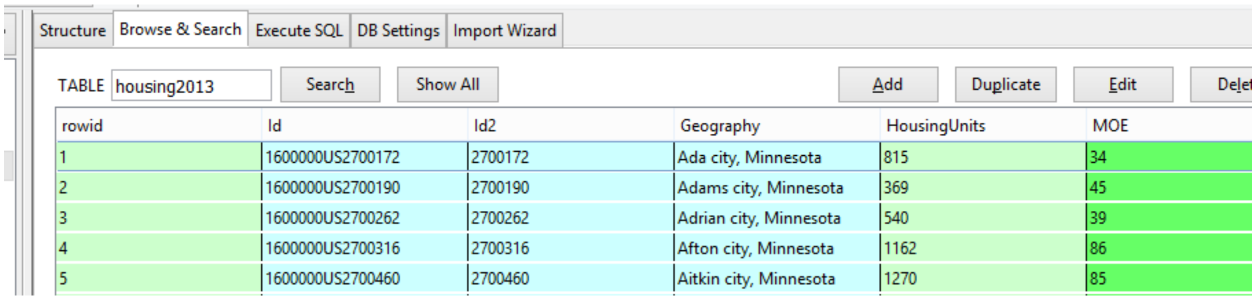


TABLE housing2013

rowid	Id	Id2	Geography	HousingUnits	MOE
1	1600000US2700172	2700172	Ada city, Minnesota	815	34
2	1600000US2700190	2700190	Adams city, Minnesota	369	45
3	1600000US2700262	2700262	Adrian city, Minnesota	540	39
4	1600000US2700316	2700316	Afton city, Minnesota	1162	86
5	1600000US2700460	2700460	Aitkin city, Minnesota	1270	85

You'll see there are two ID fields in these tables. They are basically the same – the first, though, has "1600000US" tacked on to the front. These are FIPS codes – 27 is for the state of Minnesota and the remaining five digits are the "place" FIPS codes.

How joining works:

Joining two tables together means that we want to line them up side-by-side . In this case, we want the 2000 data for Minneapolis to be on the same row as the 2013 data, then the same for St. Paul, and Bloomington and Duluth and on and on.

Some people confuse joining with "appending." Appending is when you want to add more "rows" or records below an existing dataset.

Joining means we are essentially adding more "columns" to a table.

There are a couple circumstances when joining is necessary. The first is what we're doing here. We have two datasets that came to us in separate files, but we want the information to be together.

The second is when you have a "relational" database where the data is stored in multiple tables.

For example, a dataset that is commonly stored as a relational database is campaign finance records. Typically there is one table that has one record for the committee/candidate, with their basic info (address, party affiliation, position they are running for, etc). And then there is another table with one record for each donation that each of those committees received, but usually that table only contains the ID number of the committee. You need to join the two tables to line up the name of the committee/candidate receiving the money with the name and other info of the person/committee giving the money.

Most of the time, you're going to be joining tables that were intended to go together. So for example, the campaign finance data has an ID number for the committee and that field shows up in both tables. We just need to tell our database software to "match" the two tables using those fields.

In some rare circumstances you might want to do what's called an "enterprise join" – where you are matching two tables that weren't intended to go together, but have some fields that contain the same information. For example, you might have one table with names of school bus drivers and another table with names of people convicted of felonies and you want to see if the school district is failing its job of backgrounding new drivers. That's more complex and we'll save that for another tutorial.

Let's get back to this Census data….

The first thing you need to figure out is what field(s) "match" between the two tables.

Your first inclination would probably be to join on the names, right? Seems like that would be the simplest way to go, but it's not.

Names tend to change and they offer more opportunity for error (one little typo in the spelling, for example), so it's far more likely that we'll have trouble matching. The problem is that SQL Lite (and any data program, for that matter) cannot discern that "Minneapolis" and "Mpls" are the same thing or that "Minneapolis, Minnesota" and "Minneapolis, MN" are the same thing. The program wants it to be exactly the same – down to the punctuation and spaces.

So the best way to join two tables is if you have ID numbers that are intended to be the same from one table to the next. FIPS codes are one of many examples out there of "codes" that show up in datasets for this very purpose—so you can match two datasets together.

As we mentioned above, there are two ID fields that are basically the same. For simplicity sake, I'm going to use "ID2". Conveniently, this field is named "ID2" in both tables (it doesn't always work that way and you don't need to have matching field names in order to join!)

In Structured Query Language (SQL), the joining is done in the FROM line. Up until now, the FROM line has been the easiest part of our queries.

Essentially we are now going to tell SQL Lite that our query is FROM two tables and, as part of that, we need to tell the program how to match the two tables together.

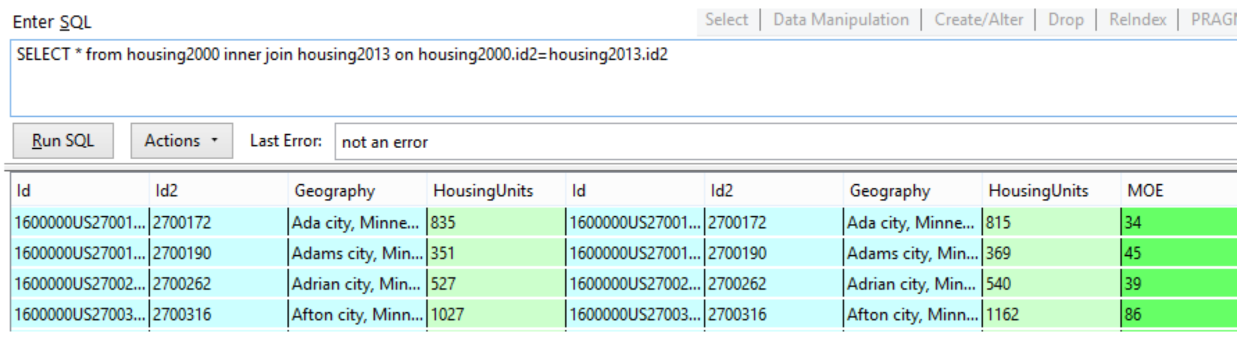
Here's the general syntax of the FROM line for a join query:

FROM tablename1 INNER JOIN tablename2 ON tablename1.fieldname=tablename2.fieldname

Notice that we have "tablename1.fieldname" and "tablename2.fieldname". This syntax is putting the tablename in front of the fieldname. This is to expressly ensure SQL Lite knows which table to pull the field from. This is REQUIRED for situations where there are fieldnames that are the same between the two tables. In this case, "ID2" exists in both tables. In fact, pretty much every field name is the same in these two tables. If your field names are different, then it's NOT required, however you can always use this syntax without causing any problems.

So let's put this to practice. We'll start with a simple query that tells SQL Lite to return all the columns from both tables. And because we're not using a WHERE statement, it will also return all the rows. (more on that in a minute)

SELECT * from housing2000 INNER JOIN housing2013 on housing2000.id2=housing2013.id2


Enter SQL
Select | Data Manipulation | Create/Alter | Drop | ReIndex | PRAGI
Run SQL
Actions

```
SELECT * from housing2000 inner join housing2013 on housing2000.id2=housing2013.id2
```

Last Error: not an error

Id	Id2	Geography	HousingUnits	Id	Id2	Geography	HousingUnits	MOE
1600000US27001...	2700172	Ada city, Minne...	835	1600000US27001...	2700172	Ada city, Minne...	815	34
1600000US27001...	2700190	Adams city, Min...	351	1600000US27001...	2700190	Adams city, Min...	369	45
1600000US27002...	2700262	Adrian city, Min...	527	1600000US27002...	2700262	Adrian city, Min...	540	39
1600000US27003...	2700316	Afton city, Minn...	1027	1600000US27003...	2700316	Afton city, Minn...	1162	86

Notice that it's giving us the fields from Housing2000 on the left and then the Housing2013 fields are on the right. It's kind of confusing, though, because the fields have the same names – it's hard to know which is which. The reason I know the 2000 fields are on the left is because that was the first table I listed in the FROM line.

The first thing you should look at is whether you got all of your records back. This query shows we got 863 rows.

We need to know how many records there were in our original tables. So let's run these two queries:

SELECT count(*) from housing2000

That says there were 867 records in that table.

```
SELECT count(*) from housing2013
```

Answer is: 905 records.

So we've got a situation here where we're not going to get a perfect match. So what would the IDEAL match be?

Our choices here are:

1) Include only cities that existed in both 2000 and 2013. It will exclude any cities that have data in one year, but not the other.
2) Get a table back that has all the cities that existed in 2013, showing the 2000 data as well if the city existed back then, but it would be blank for ones that did not. This approach would exclude any cities that existed in 2000, but are not in the 2013 data (maybe it lost its city status and is now a township, for example – or maybe it merged with another city)
3) Get a table back that has all the cities that existed back in 2000, with data for 2013 if that city is still around. This one would exclude any cities that did not exist in 2000, but are in the 2013 data

Your choice is going to be dependent on your purpose with this analysis. But I also think it's best to see what you're missing, so I usually like to go with option 2 or option 3—at least initially.

The query we ran is the option 1 version – it's known as an INNER JOIN

Options 2 and 3 are known as an OUTER JOIN. This is one place where SQL LITE is a bit lacking compared to other database programs. SQL LITE only allows you to do what's called a 'LEFT OUTER JOIN' – keeping all the records from the table that you listed first in the FROM line. (Other programs allow you to do a RIGHT OUTER JOIN, as well).

So if we want to do option 3 – returning all the records from 2000 and only those that match from 2013, we would change our syntax just slightly to this:

```
SELECT * from housing2000 LEFT OUTER JOIN housing2013 on housing2000.id2=housing2013.id2
```

We get 867 records back—the same number that there are in the 2000 table. If you scroll down a ways you'll find some cities – like New Market city – that don't have matches from 2013. (a record for New Market city doesn't exist in the 2013 table; this is a case where New Market was merged with another city and is now called Elko-New Market and it has an entirely different FIPS code)

If we want to do option 2 – get all the records from 2013 and only the matches from 2000, then we need to rearrange the order, so that we list "housing2013" first in the FROM line:

```
SELECT * from housing2013 LEFT OUTER JOIN housing2000 on housing2013.id2=housing2000.id2
```

This time we get 905 records. The same number as in the 2013 table. So the join did what it said…gave us all the 2013 records.

You'll see that there are a few that have blanks in the fields where you'd expect to find the 2000 data. (For example, look at Angle Inlet CDP). You'll see that there are a bunch of records for these CDP entities – I think these are places that used to be treated as townships, but more recently they've been identified as "places," so now they show up in this data.

There are a couple that aren't CDP's – Elko New Market and Columbus. What's going on with these? I would want to make sure there is a legitimate reason my join didn't work on these (and not a case of a typo). So typically I would try to do a manual search in the 2000 table and just make sure it's not really there. If you go looking for Elko New Market in the 2000 data, you'll find "Elko city" and "New Market City" and a little research in past news stories will show you that these two cities merged sometime between 2000 and 2013. If you go looking for Columbus, you won't find anything. The news clips, though, will show you that this used to be a township and it incorporated a few years back.

So we've now confirmed that all the places that don't match up have legitimate reasons for not matching.

Most likely you care most about the places that exist currently – so you'd want to keep all the records from 2013. So we'll stick with the query we just ran.

Let's fine tune our query a little more:

The Select line is where we put what fields we want to show in our answer. And once you make the join, you can pick and choose whichever fields from the two tables you want. In this case, we're pulling all of them. But maybe we don't want all of them?

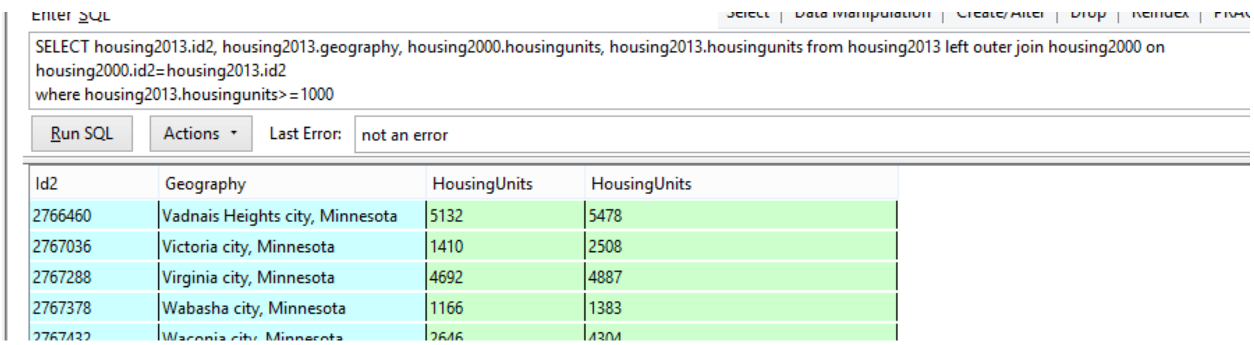
For example, we could set it up so that we just have the ID and Name from the 2013 data, and then the housing unit counts from both tables, like this:

```
SELECT Housing2013.Id2, Housing2013.Geography, Housing2000.HousingUnits,
Housing2013.HousingUnits
FROM Housing2013 RIGHT OUTER JOIN Housing2000 ON Housing2000.Id2 = Housing2013.Id2
```

Notice that we're using the "tablename.fieldname" syntax in the SELECT line to clarify which fields we want returned.

Once you have your join set up and working the way you want, then you can run queries as if it's just one table. As we've done above, you can pull fields from either table. You can also limit your results using a WHERE clause. Or order the results with an ORDER BY.

For example, let's say we want to only return places with more than 1,000 housing units.



```
SELECT housing2013.id2, housing2013.geography, housing2000.housingunits, housing2013.housingunits from housing2013 left outer join housing2000 on
housing2000.id2=housing2013.id2
where housing2013.housingunits>=1000
```

not an error

Id2	Geography	HousingUnits	HousingUnits
2766460	Vadnais Heights city, Minnesota	5132	5478
2767036	Victoria city, Minnesota	1410	2508
2767288	Virginia city, Minnesota	4692	4887
2767378	Wabasha city, Minnesota	1166	1383
2767432	Waconia city, Minnesota	2646	4304

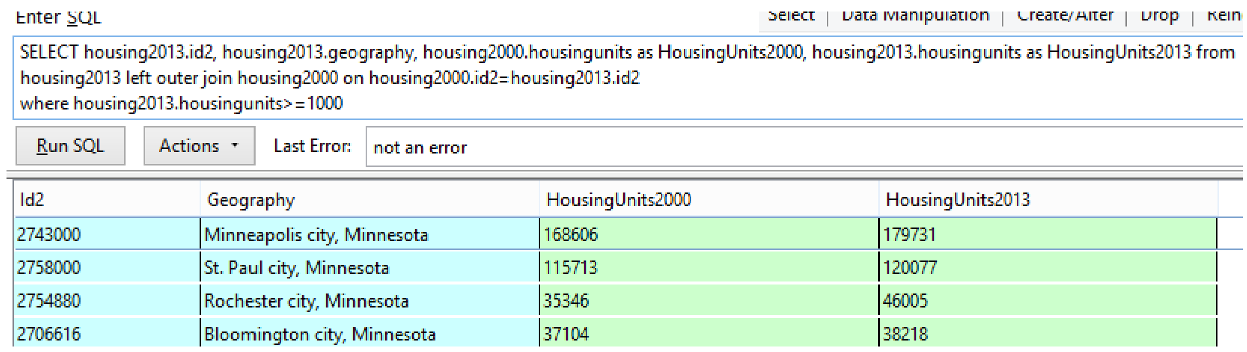
Then let's add an ORDER BY:

SELECT housing2013.id2, housing2013.geography, housing2000.housingunits, housing2013.housingunits
FROM housing2013 left outer join housing2000 on housing2000.id2=housing2013.id2
WHERE housing2013.housingunits>=1000
ORDER BY 4 desc

Let's do one more housekeeping thing that will make life a little easier. Notice that we now have two fields called 'Housingunits'. It's hard to know which is from 2000 and which is from 2013. We can fix that by assigning those fields an "alias."

Here's how:

SELECT housing2013.id2, housing2013.geography, housing2000.housingunits as HousingUnits2000, housing2013.housingunits as HousingUnits2013 FROM housing2013 left outer join housing2000 on housing2000.id2=housing2013.id2
WHERE housing2013.housingunits>=1000
ORDER BY 4 desc



```
SELECT housing2013.id2, housing2013.geography, housing2000.housingunits as HousingUnits2000, housing2013.housingunits as HousingUnits2013 from
housing2013 left outer join housing2000 on housing2000.id2=housing2013.id2
where housing2013.housingunits>=1000
```

not an error

Id2	Geography	HousingUnits2000	HousingUnits2013
2743000	Minneapolis city, Minnesota	168606	179731
2758000	St. Paul city, Minnesota	115713	120077
2754880	Rochester city, Minnesota	35346	46005
2706616	Bloomington city, Minnesota	37104	38218

In this example, we joined 2 tables together. However, it is possible to join many tables at the same time. Of course, it gets more complicated as you add more tables – especially when it comes to making sure you're getting all the records that you expect to get back.

Sometimes you will use JOIN as part of your analysis and you'll just keep running queries with the JOIN set up in the FROM line, for any queries where you need to pull information from more than one table.

Other times you might want to create a new table from this result. Here's the syntax you would use to do that (notice that I've changed the SQL back so that we're not excluding any records)

```
CREATE TABLE HousingMatch
as
SELECT housing2013.id2, housing2013.geography, housing2000.housingunits as HousingUnits2000,
housing2013.housingunits as HousingUnits2013
FROM housing2013 left outer join housing2000 on housing2000.id2=housing2013.id2
```