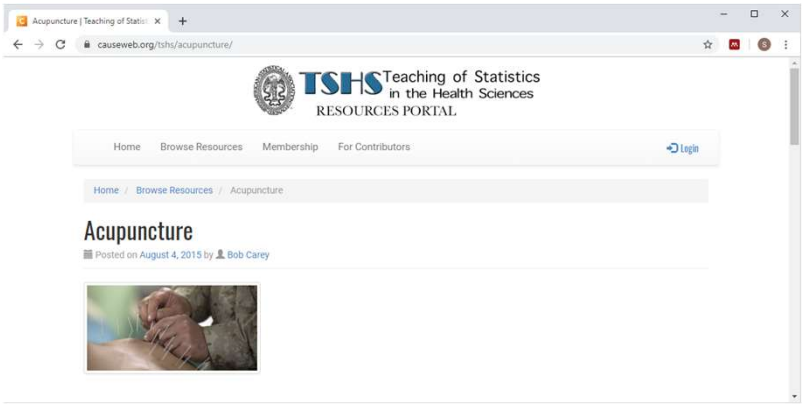
Simple joins

Suman Sahil, Steve Simon

October 13, 2017

New database, description (1/3)



Screenshot of TSHS resource page

I've created a database using the Teaching of Statistics in the Health Sciences resource page.

New database, description (2/3)

- Available on the web
- Originally from a study, Acupuncture for chronic headache in primary care: large, pragmatic, randomized trial by Vickers et al, published in BMJ in 2004.
- The full data set is available in Whose data set is it anyway? Sharing raw data from randomized trials by Andrew Vickers in Trials, 2006.

You can find this study on the web. It was from a BMJ paper published in 2004. I've adapted it to this class by adding three small tables to the main table.

New database, description (3/3)

- A subset of this data was submitted to the Teaching of Statistics in the Health Sciences Resources Portal.
- Available using an open source license.
- Converted to Oracle and SQLite with three small tables added.

I've adapted it to this class by adding three small tables to the main table.

New database, list files

– SQL code

```
select tbl_name
  from sqlite_master
```

– Output

```
##         tbl_name
## 1  results_table
## 2      sex_table
## 3 migraine_table
## 4    group_table
```

Anytime you encounter a new database, your first step should be to list all the table names. This code works with SQLite, but you have to make some changes to get it to work with Oracle.

New database, peek at results_table

– SQL code

```
select id, sex, migraine, grp
  from results_table
  limit 4
```

– Output

```
##    id sex migraine grp
## 1 100   1        1   1
## 2 101   1        1   0
## 3 104   1        1   0
## 4 105   1        1   0
```

Here's the first four records of results_table. Normally, I would print every field, but the output would not fit easily on a single slide, even with just four records.

New database, peek at sex_table

– SQL code

```
select *
  from sex_table
  limit 4
```

– Output

```
##   sex_code sex_label
## 1        0      Male
## 2        1    Female
```

Here's the data from sex_table. Notice that I put a limit on the number of rows returned, but only because I did not know that this table only had two recrods. Actually, I did know, but in most circumstances, you don't know this information ahead of time. The safer thing to do is to place limits on the number of rows retrieved until you know what you are doing.

New database, peek at migraine table

– SQL code

```
select *
  from migraine_table
  limit 4
```

– Output

```
##   migraine_code migraine_label
## 1             0       Migraine
## 2             1   Tension-type
```

Here's the data from migraine_table.

New database, peek at group_table

– SQL code

```
select *
  from group_table
  limit 4
```

– Output

```
##   group_code group_label
## 1          0     Control
## 2          1   Treatment
```

Here's the data from group_table.

Purpose of these extra tables

- This is a common strategy
  - Use number codes in big table
  - Define labels in smaller table
  - Use join to combine
- Advantages of this approach
  - Saves storage
  - Redefine labels in a single location
- Simplest case
  - Every code has a label in small table
  - Every label has a code in the big table

The three small tables provide information about then number codes used in the big table, results_table. This allows you to prepare data sets that use the informative labels instead of or in addition to the number codes. When you want a data set that has these labels, you use a join command.

This can get tricky at times. We're considering in this lecture the simplest case. In the simplest case, every code found in the big table can be matched to a label in the small table. If there are no matches, then you have two choices, leave the code in and insert a null value for the label. Or you can toss the unmatched record out.

A second important condition in the simplest case is that every label in the small table corresponds to at least one code in the big table. If some labels in the small table are unused, you again have two choices. Put in the label and use null values for everything in the big table or toss out the label.

If everything matches up: no labels without codes and no codes without labels, your life is simpler. I want to keep things simpler for you as we talk about this example, but keep in mind that down the road you will have to decide between different strategies when things don't match perfectly.

Using group labels (1 of 2)

– SQL code

```
select id, group_label
  from results_table
  join group_table
  where grp=group_code
  limit 4
```

– Output

```
##    id group_label
## 1 100   Treatment
## 2 101     Control
## 3 104     Control
## 4 105     Control
```

Here's the first five rows of results_table.

Using group labels (2 of 2)

– SQL code

```
select id, group_label
  from results_table
  join group_table
  on grp=group_code
  limit 4
```

– Output

```
##    id group_label
## 1 100   Treatment
## 2 101     Control
## 3 104     Control
## 4 105     Control
```

Difference between the where keyword and on keyword

- Execution order
  - on keyword runs during the join
  - where keyword runs after the join
- Not an issue for simplest joins
  - Problematic for joins that produce null values
- Recommendation
  - Reserve the where keyword to filtering operations
  - Use the on keyword to define how join is done
  - Consult an expert about complex situations

The distinction between the on and where keywords is somewhat analogous to the distinction between the where and having keywords. The on keyword specifies a condition that applies during the join and the where keyword specifies a condition that applies after the join.

In our simple example, these two queries both produced the same result. I've not encountered any settings yet where there is a difference, but it can happen. If there are complex joins (left join, right join, and outer join), then you might get different results, mostly for the records that do not match. The null values that are sometimes produces in left, right, and outer joins are not directly responsible for this, but the rule of thumb with databases is that if you have trouble with something, there are usually null values hanging around in the vicinity.

The distinctions between the on keyword and the where keyword are often subtle. I've included a few resources in the recommended readings that try to explain this, but I don't want you to worry too much about the details.

Here's a good rule of thumb that. Use the where keyword for filtering and use the on keyword to specify how the records in the two tables should be matched. At a minimum, would make your code more readable. It might also improve the efficiency of your code.

Maybe not, as most modern databases take a careful look at your SQL code and find ways to optimize things, even if your SQL code was not written optimally. These companies that sell databases have been at it for many decades and they have a very strong incentive to make their databases faster than the competition.

A simple and very good recommendation is to reserve the where keyword for filtering operations, operations that create a smaller subset of your data after the join is done. Then use the on keyword to describe how to link the two tables.

It is always a good idea to consult with an expert for any complex query.

Three competing examples (1/3)

– SQL code

```
select id, age, group_label
  from results_table
  join group_table
  on grp=group_code and age<40
  limit 4
```

– Output

```
##    id age group_label
## 1 104  32     Control
## 2 141  37   Treatment
## 3 149  23   Treatment
## 4 151  35     Control
```

Suppose you want to use the above join where the condition for matching is grp=group_code, but you also want to filter out anyone 40 years or older.

One way to add the age<40 condition to the on statement.

Three competing examples (2/3)

– SQL code

```
select id, age, group_label
  from results_table
  join group_table
  where grp=group_code and age<40
  limit 4
```

– Output

```
##    id age group_label
## 1 104  32     Control
## 2 141  37   Treatment
## 3 149  23   Treatment
## 4 151  35     Control
```

You could have the two conditions in the where statement.

Three competing examples (3/3)

– SQL code

```
select id, age, group_label
  from results_table
  join group_table
  on grp=group_code
  where age<40
  limit 4
```

– Output

```
##    id age group_label
## 1 104  32     Control
## 2 141  37   Treatment
## 3 149  23   Treatment
## 4 151  35     Control
```

The recommended approach, however, is to place the grp=group_code statement in the on statement, where it explains how the two tables are linked and then add the filter age<40 to a where statement that comes after the on statement.

Specifying table name to avoid ambiguity

– SQL code

```
select
  results_table.id,
  group_table.group_label
  from results_table
  join group_table
  on results_table.grp=group_table.group_code
  limit 4
```

– Output

```
##    id group_label
## 1 100   Treatment
## 2 101     Control
## 3 104     Control
## 4 105     Control
```

It's a good idea to include the table_name along with the field name when you are running a query involving multiple tables. It will resolve conflicts if there is the same field name in two different tables. Even if there are no conflicts, it will make your SQL query easier to understand.

Table aliases can help a lot

– SQL code

```
select
  r.id,
  g.group_label
  from results_table as r
  join group_table as g
  on r.grp=g.group_code
  limit 4
```

– Output

```
##    id group_label
## 1 100   Treatment
## 2 101     Control
## 3 104     Control
## 4 105     Control
```

With even moderate length table names, you will end up doing a lot of extra typing. It is common practice to use a brief abbreviation (sometimes as short as a single letter) as a table alias. You do this with the as keyword.

Alternative: conditional functions

– SQL code

```
select
  id, case grp
    when 0 then 'Control'
    when 1 then 'Treatment'
  end as group_label
  from results_table
  limit 4
```

– Output

```
##    id group_label
## 1 100   Treatment
## 2 101     Control
## 3 104     Control
## 4 105     Control
```

If you've been paying attention, you'll realize that there is an alternative. You can use the case keyword to create a set of labels. Here's an example of how that would work. By the way, you really need to define an alias for any variable created by the case keyword. The name that SQL uses is very long and unwieldy.

Disadvantages of conditional functions

- Does not work when you have many labels
  - Example, labels for ICD-10 codes
- Places more burden on the programmer
- Difficult to update labels

The conditional function is actually a pretty good choice for simple settings. It works fine when you have a handful of labels, but anything more than 4 or 5 requires a lot of coding. For example, there are nice labels for ICD-10 codes (International Classification of Disease) and it is fairly easy to import them into a table. But putting them in your code is rather awkward. There are over 68 thousand different codes and embedding them into a SQL query would be next to impossible.

Another disadvantage is that the case keyword places the burden of knowing the labels on the programmer. if you link to a table, you don't need and don't care about what the actual labels are. You just have to have enough information that you can link thigns together.

A third problem is that you might want to update your labels from time to time. You might, for example, want to change one of the race labels from "Black" to "African American." If the labels are in a single table then a change to that table will immediately be reflected in any query, including queries written before you made your change. If you rely on the case keyword, then you have to find and update every SQL program that uses your data.

Using single row functions

– SQL code

```
select
  r.id,
  substr(s.sex_label, 1, 1) as letter_code
  from results_table as r
  join sex_table as s
  on r.grp=s.sex_code
  limit 4
```

– Output

```
##    id letter_code
## 1 100           F
## 2 101           M
## 3 104           M
## 4 105           M
```

This example uses the single row functions that you learned about earlier to produce a single letter code, M and F, instead of the full label.

This might be useful if you wanted to use the single letter as text values in a plot.

Using aggregated data (1/2)

– SQL code

```
select
  sex, count(*) as n
  from results_table as r
  group by sex
```

– Output

```
##   sex   n
## 1   0  64
## 2   1 337
```

This example uses the statistical summary function to get a count of males and females in your table. This is a nicer example

Using aggregated data (2/2)

– SQL code

```
select
  s.sex_label, count(*) as n
  from results_table as r
  join sex_table as s
  on r.sex=s.sex_code
  group by s.sex_label
```

– Output

```
##   sex_label   n
## 1    Female 337
## 2      Male  64
```

This example uses the statistical summary function to get a count of males and females in your table. This is a nicer example

Conclusion

– What have you learned.
  - Using join to add labels to a table with number codes.
  - Simplest case, every code has a label and every label has a code
  - Distinction between on keyword and where keyword
  - Using single row functions and aggregated data with labels

In this lecture, you learned about a simple (but very common) use of multiple tables. The extra tables allow you to easily replace number codes with more decriptive labels. If there is a perfect match: every code in the big table has a label in the small table and every label in the small table has a code in the big table, then things work out easily. We talked about how the on keyword is best reserved for information linking two tables while the where keyword is best used for filtering your data after the join is completed. You also saw examples utilizing single row functions and aggregated data with labels.

Homework

1. Use the same database shown in this video. It is available on the Insights platform, or you can download a sqlite file from Canvas.
2. List id and migraine_label for the first ten records after joining the results_table and migraine_table.
3. Get a count of the number of records in the database in the control group and the treatment group. Use the label for group and not the number code.
4. Get a count of the numbers of males and females where you restrict age to be less than 40. Use the label for sex, but convert it to all uppercase

Here is your homework. Use the same database.