Using R with Databases and SQL

Adam Victor

October 20, 2017

Welcome to The Twilight Zone

Imagine waking up one day and finding, to your great confusion, that no one has ever heard of pizza. Sure, they're familiar with the food, but that's not pizza, that's 'dough topped with a tomato spread and melted cheese'. "Here's your dough topped with a tomato spread and melted cheese!" exclaims your waiter as he brings out some pizza and hands you a spoon to eat it with. You look down at your plate and the spoon in your hand as you struggle to use it to eat the pizza - or rather dough topped with a tomato spread and melted cheese. Your hands shake, you close your eyes and scream, desperate to awaken from this nightmare.

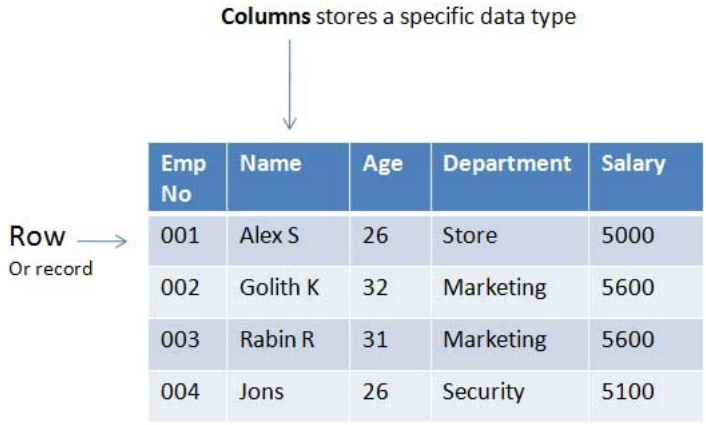
As someone that has had some exposure to relational databases and SQL, mainly through CS186, learning about R's data frames felt like entering a strange Twilight Zone where relational tables were called data frames and using clunky subsetting to manipulate data was the norm, rather than the clean SQL statements I had grown to know. Eventually I learned about dplyr, which, while much better than using subsetting to access data, still felt like a strange perversion of SQL.

I don't intend to argue that R's way of doing things is wrong or bad, but I assume I can't be alone in my initial confusion to seeing R's "tables by any other name". Interestingly, there do exist ways of doing data manipulation in R in a more SQL-ish way - by using SQL itself! In this post, I will discuss sqldf, an R package for using SQL to get data from data frames. I intend to highlight how to use it, who it could be useful for, and some potential benefits. Then, in a complete reversal I'll discuss how R can be used to access relational databases using idiomatic dplyr, as well as potential uses of this.

What are Relational Databases? What is SQL?

Since I've arguably put the cart before the horse here, let's take a step back and pause for a moment so I can explain what in the world I'm talking about. R as a language is primarily used for data manipulation, and it has some great built in structures, like data frames, for doing exactly that. However, in general this isn't the only way.

Relational Databases store data in a very similar way to data frames. Relational Databases contain several tables, each of which has columns describing that table's fields. The tables then have rows that provide specific instances of data described by the columns. This is actually very similar to data frames which also have columns and rows with similar meanings! Here's an example of what a table might look like in a database.



Emp No	Name	Age	Department	Salary
001	Alex S	26	Store	5000
002	Golith K	32	Marketing	5600
003	Rabin R	31	Marketing	5600
004	Jons	26	Security	5100

In R, we have subsetting and dplyr to access data contained in data frames. With most relational databases, SQL is used to access data contained in the tables. SQL stands for Structured Query Language and is a declarative language, meaning you tell it what to do, not how to do it. Here's a quick example. Let's say you have a table called "Students" where each student has a name and score. You could get the names of every student using SQL with the code `SELECT Students.name FROM Students`. Notice that we don't specify how to get what we want, just what we want. R can do things very similarly, either with subsetting via `Students[, "name"]` or with dplyr, using `select(Students, name)`. That code even looks pretty similar to the SQL statement!

Querying Data Frames with SQL

Enter sqldf, an R library intended to bridge the gap between R and SQL. With sqldf, you can write SQL queries to select from the contents of R data frames. Let's try it out to see how it works! Note that you will first need to install the library with `install.packages("sqldf")`.

```
#Let's create a Students data frame to play around with.
Students <- data.frame(
  name = c("Adam", "John", "Pauline", "Christine", "George"),
  score = c(99, 55, 86, 100, 10))
#Let's try using sqldf now!
#(note: I suppress warnings and messages when loading the libraries only to not clutter the output.)
suppressWarnings(suppressMessages(library(sqldf)))
sqldf("SELECT Students.name FROM Students")
```

```
##         name
## 1       Adam
## 2       John
## 3   Pauline
## 4 Christine
## 5    George
```

```
topStudentsSqldf = sqldf("SELECT Students.name FROM Students ORDER BY Students.Score DESC LIMIT 2")
topStudentsSqldf
```

```
##        name
## 1 Christine
## 2      Adam
```

```
#Compared to Dplyr
suppressWarnings(suppressMessages(library(dplyr)))
topStudentsDplyr = Students %>% arrange(desc(score)) %>% slice(1:2) %>% select(name)
topStudentsDplyr
```

```
## # A tibble: 2 x 1
##        name
##      <fctr>
## 1 Christine
## 2      Adam
```

```
is.data.frame(topStudentsSqldf)
```

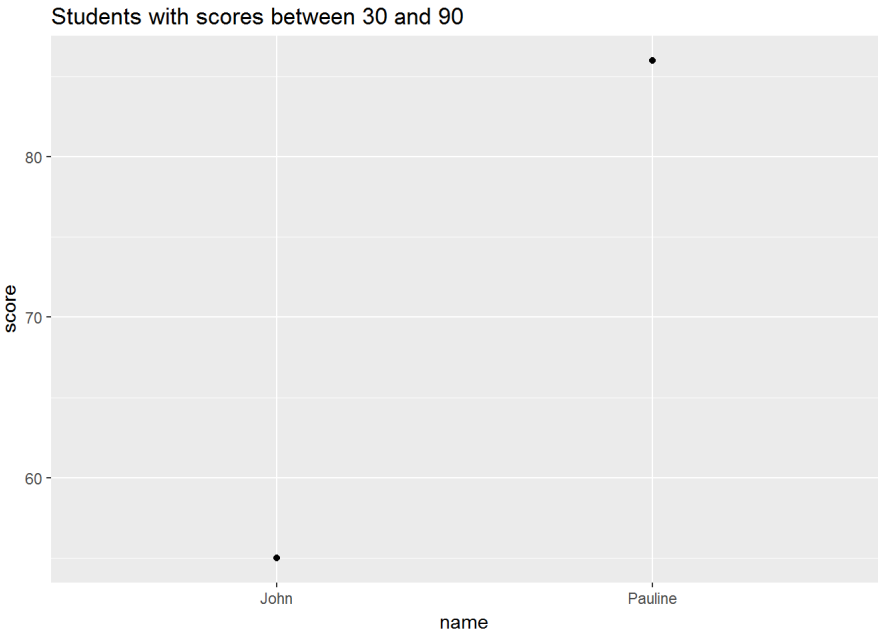
```
## [1] TRUE
```

```
is.data.frame(topStudentsDplyr)
```

```
## [1] TRUE
```

As you can see, both sqldf and dplyr can be used to get the correct outputs and both as data frames. Just for fun, let's try combining them both.

```
suppressWarnings(suppressMessages(library(ggplot2)))
data <- sqldf("SELECT Students.name, Students.score
              FROM Students WHERE Students.score < 90") %>% filter(score >= 30)
ggplot(data, aes(name, score)) + geom_point() + ggtitle("Students with scores between 30 and 90")
```



Now what do we have here? Above is a graph of all students and their scores for students with scores between 30 and 90. Nothing too exciting about the graph, except for how we created it. We didn't just use dplyr or sqldf, we used both! First we use sqldf to get students with scores below 90 and then we take the new data frame created from this and pass it to a dplyr filter to eliminate any students with scores below 30. This shows that sqldf creates standard data frames that can be graphed or used with dplyr.

What's the point?

We've shown that we can do some neat stuff with sqldf, but why use it when we can do the same things with dplyr? As I see it, there are a few main reasons. First, if you've come from a heavy SQL background but don't have much R experience, using SQL in R gives you a chance to use what you're familiar with while giving you the chance to mix in dplyr to ease into R. Another benefit is that some things may just seem more natural in one or the other, and this let's you choose what you prefer. R-bloggers explains, "it is not so matter whether R or SQL can express it but how easy it is to express. Some queries are simply much easier to express in one notation or the other." And a third, and possibly surprising benefit, is that in some cases using sqldf can provide a speedup over using normal R. According to a StackOverflow answer, using sqldf allowed them to pull in 2GB of text data in under 5 minutes. They had previously tried the task with R's `read.csv`, but had given up after it took an entire night and still didn't finish!

Querying Databases with R

Up to this point, I've discussed how SQL, a language typically used to query databases, can be used to instead query R data tables. Now, I will briefly discuss an antiparallel topic - using R (namely dplyr) to query databases! This uses the dbplyr, dplyr, and RSQLite libraries, so make sure to install those if you want to try the code out. I've included in the data directory a students.sqlite file, which defines an SQLite database with a single students table, the same one being used earlier. For the following code, it may seem like we're querying the data as if it's in a data frame. That's the magic of this technique - we can use dplyr as normal and abstract away the fact that our data is actually in a database!

```
suppressWarnings(suppressMessages(library(dplyr)))
suppressWarnings(suppressMessages(library(dbplyr)))
suppressWarnings(suppressMessages(library(RSQLite)))
#Establish a connection to the SQLite database.
studentsDB <- DBI::dbConnect(RSQLite::SQLite(), "data/students.sqlite")
#Show info on the database, including the tables in it.
src_dbi(studentsDB)
```

```
## src:  sqlite 3.19.3 [C:\Users\Adam\stat133\post1\data\students.sqlite]
## tbls: Students
```

```
#Get the students table from the database
studentsTable <- tbl(studentsDB, "students")
#Run a dplyr query to get the names of students with scores above 60.
studentsTable %>% filter(score > 60) %>% select(name)
```

```
## # Source:   lazy query [?? x 1]
## # Database: sqlite 3.19.3
## #   [C:\Users\Adam\stat133\post1\data\students.sqlite]
##       name
##      <chr>
## 1     Adam
## 2   Pauline
## 3 Christine
```

```
dbDisconnect(studentsDB)
```

As you can see, we've managed to query the students database with the dplyr functions we've grown used to. If it wasn't for the lines connecting and disconnecting the database, you might not even realize we're accessing an external database.

What's the Point This Time?

This is neat and all, but why bother when we can just stick with regular data frames? To answer that, let's ponder why anyone uses databases in the first place. The simple reason is that not everything can fit into the computer's memory (that is, RAM) at a time. The fictional students table may not be a great example, but imagine a massive dataset with millions of entries. It may eventually grow to the point that we can't fit that much data into memory at once, but using a database system, like SQLite, allows us to access the data still! Now that we can access databases from R, we aren't constrained by the size of the datasets we're dealing with. Additionally, we can mix regular R code, dplyr, and SQL to create more complex programs for data manipulation.

Conclusion

Throughout this post, I've discussed ways of bridging the gap between R's data frames and the tables used in relational databases. With sqldf, we can query R data frames using SQL, a language normally used to query databases. Additionally, the potent combination of dplyr, dbplyr, and RSQLite allows us to connect to external databases and query tables using either SQL or dplyr commands. All of this is significant because it can make it easier for people with different backgrounds to work together. Data scientists familiar with R and database engineers familiar with databases and SQL can cooperate on more complex data projects, combining their skillsets together. Even more, these libraries all allow programmers to overcome limitations of R. As mentioned before, sqldf can potentially give a speedup when doing certain tasks, like reading a large file into a data frame. Dbplyr and RSQLite allow programmers to work with larger datasets by first loading them into a database and then querying them through R. While this post focused on simple examples, all of these libraries allow some really amazing and complex tasks to be done. I have no doubt these libraries will be invaluable to the R community.

References

1. https://www.r-bloggers.com/make-r-speak-sql-with-sqldf/
2. https://db.rstudio.com/
3. http://www.burns-stat.com/translating-r-sql-basics/
4. https://github.com/ggrothendieck/sqldf
5. http://www.datacarpentry.org/R-ecology-lesson/05-r-and-databases.html
6. https://stackoverflow.com/questions/1727772/quickly-reading-very-large-tables-as-dataframes-in-r/1820610#1820610
7. http://www.plus2net.com/sql_tutorial/images/table.jpg
8. https://cran.r-project.org/web/packages/RSQLite/vignettes/RSQLite.html