Shawn Houser
STA 574 - Statistical Software Packages
Midterm Graduate Presentation - Using Sqlite3 and SQL databases
2019-11-05

What is a Database Management System
- A tool that provides a layer of abstraction
  over the data that you interact with.

Without a Database
- You have to interact with a bunch of files
  - Imagine doing analysis on MainStreet
  - There will be a bunch of different data types
  - There could be many csv's, json, xml, or worse
- You may have to deal with custom datatypes
  - USM may have a custom format that you need
    to extract data from.

A database will solve these issues
Sqlite3 is a specific database manager (it manages the data)

Why Sqlite3
- It saves all of the data in one file
- You interact with this data using SQL
- It can be easier to ask for data in SQL
  - than it would be to write code in R
- It has been tried and tested
- Universal, works on nearly anything
- Used everywhere
  - Web browsers, cars, planes
- Databases it produces are a single file that
  - can be transferred without fear of different
    computers doing nasty things to the data

What does a database look like?
- Collection of tables
- Each table is like its own csv file
- Relations between tables
  - Different tables can share values

Example Database
- words.csv
- ---->

```
word_string | word_id
------------|--------
 Strawberry |    1
 Apple      |    2
 ...        |  ...
 Saturday   | 93,784
```

counts.csv
---->

```
 word_id | word_count
 --------|-----------
     1   |  3992
     2   |    42
 93,784  |   112
```

Installing sqlite3

install.packages("RSQLite")
This should be all you need.
Usure if this will install sqlite3 itself,
or only allow connection if it is already installed

Some of the examples in this document may not make sense on their own. They all require datafiles to be run correctly. This file was meant to be mostly a compilation of my notes for this presentation. The presentation is meant to be interactive, and many of the results would have been shown in realtime.

The data can be found here
https://drive.google.com/open?id=1IGfAOdZQ9DyTz8RLnj-ucnIOHG6elLYF
You must be signed into a USM email address.

This is a small demo to show how to interact with a sqlite3 database from R, and its counterpart of interacting with a csv file.

Regular.r

```
paste("Starting at ", Sys.time())

words <-
read.csv("/Users/shawn/Desktop/ssp_midterm/data/words.
csv")
head(words)

paste("Ending at ", Sys.time())
```

Sql.r

```
paste("Starting at ", Sys.time())

# https://db.rstudio.com/databases/sqlite/
library(DBI)
database <- dbConnect(RSQLite::SQLite(),
"/Users/shawn/Desktop/ssp_midterm/data/word_database.d
b")
res <- dbSendQuery(database, "SELECT * FROM words
ORDER BY word_id asc LIMIT 6")
dbFetch(res)

paste("Ending at ", Sys.time())
```

After running both these files, we can see that the sqlite3 version ran much faster. The reason is because we don't need to read the entire file, only the parts that we want to.

Not all interactions with sqlite3 need to be stored in a file, information can be stored in the RAM of your computer and will exist only as long as the R program exists. This is useful because you will not have to keep extra files that are useless to you after computation has been completed.

Regular.r

```
    subset(mtcars, cyl == 6)
```

Sql.r

```
    library(DBI)

    # Connects to a database that isn't on the harddrive
    # Useful for something that you do not want to deal
    # with afterwards, or start with a clean slate each
    time
    database <- dbConnect(RSQLite::SQLite(), ":memory:")

    # mtcars is the cars dataset. It is already loaded
    head(mtcars)
    cat("\n")

    # Stores the cars dataframe into a new table called
    mtcars
    dbWriteTable(database, "mtcars", mtcars)

    # Shows what the table looks like
    dbListFields(database, "mtcars")
    cat("\n")

    res <- dbSendQuery(database, "SELECT * FROM mtcars
    WHERE cyl = 6")
    dbFetch(res)
```

We can see that the sqlite3 version took a lot more setup than its pure R counterpart. This may be inefficient for small projects and tables like this, but for bigger problems the sqlite3 solution will be simpler.

There are functions that are very similar to R in sql. An example of this is the max function. It will iterate over all rows in a table and select the row with the maximum of that column.

Regular.r

```
paste("Starting at ", Sys.time())

counts <-
read.csv("/Users/shawn/Desktop/ssp_midterm/data/counts.csv")
max_word <- counts[which.max(counts$word_count), ]
paste("The max word_id was", max_word$word_id, "with a count
of", max_word$word_count)

paste("Ending at ", Sys.time())
```

Sql.r

```
paste("Starting at ", Sys.time())

# https://db.rstudio.com/databases/sqlite/
library(DBI)
database <- dbConnect(RSQLite::SQLite(),
"/Users/shawn/Desktop/ssp_midterm/data/word_database.d
b")
res <- dbSendQuery(database, paste(
    "SELECT word_id, max(word_count)",
    "FROM counts"
))
max_word <- dbFetch(res)

# Doesn't work because `max(word_count)` changes the
name of
# the row, you must either change the name here or in
the query
paste("The max word_id was", max_word$word_id, "with a
count of", max_word$word_count)
cat("\n")
paste("The max word_id was", max_word$word_id, "with a
count of", max_word$max)

paste("Ending at ", Sys.time())
```

Similar to 03_max, but with count. It counts the number of rows.

Regular.r

```
nrow(subset(mtcars, cyl == 6))
```

Sql.r

```
library(DBI)

database <- dbConnect(RSQLite::SQLite(), ":memory:")
dbWriteTable(database, "mtcars", mtcars)
res <- dbSendQuery(database, paste(
    "SELECT count(*)",
    "FROM mtcars",
    "WHERE cyl = 6"
))
dbFetch(res)
```

The real utility in sql is being able to work with multiple tables at once. This can become tiresome if you are working with just csv files.

Regular.r

```
paste("Starting at ", Sys.time())

counts <-
read.csv("/Users/shawn/Desktop/ssp_midterm/data/counts
.csv")
words <-
read.csv("/Users/shawn/Desktop/ssp_midterm/data/words.
csv")

#
https://www.rdocumentation.org/packages/base/versions/
3.6.1/topics/merge
#
https://stackoverflow.com/questions/1299871/how-to-joi
n-merge-data-frames-inner-outer-left-right
words_and_counts <- merge(counts, words, by =
"word_id")
max_word <-
words_and_counts[which.max(words_and_counts$word_count
), ]
paste(
    "The max word was `", max_word$word, "`",
    "with word_id", max_word$word_id,
    "and a count of", max_word$word_count
)

paste("Ending at ", Sys.time())
```

Sql code on next page

Sql.r

```
paste("Starting at ", Sys.time())

# https://db.rstudio.com/databases/sqlite/
library(DBI)
database <- dbConnect(RSQLite::SQLite(),
"/Users/shawn/Desktop/ssp_midterm/data/word_database.d
b")
res <- dbSendQuery(database, paste(
    "SELECT max(word_count), word_id, word",
    "FROM words",
    "INNER JOIN counts USING (word_id)"
))
max_word <- dbFetch(res)
paste(
    "The max word was `", max_word$word, "`",
    "with word_id", max_word$word_id,
    "and a count of", max_word$max
)

paste("Ending at ", Sys.time())
```

We can see that again sqlite3 is much faster than reading a csv file and doing work on that. The reason for this is the database is smart about how it stores the data. It can do operations much faster than we could.

Many times, the queries that you want to run could be very complex. When the query becomes that complex it is useful to move it into its own file. R can then read the file and use the sql in there to run a query.

Sql.r

```
      library(DBI)
      library(readr)

      database <- dbConnect(RSQLite::SQLite(),
      "/Users/shawn/Desktop/ssp_midterm/data/senators.db")
      res <- dbSendQuery(database,
      read_file("/Users/shawn/Desktop/ssp_midterm/06_scripts/probl
      em2.sql"))
      dbFetch(res)
```

Problem2.sql

```
      select distinct
          stname,
          cname,
          sname,
          age,
          totrev,
          population
      from senators
      inner join corporations using (stname)
      inner join states using (stname)
      where
          totrev > (3*population)/2
          and gender = 'Female'
          and party = 'Democrat'
      order by stname, cname, sname
```

Correct_output.txt

```
       stname |           cname            |     sname     |
      age |   totrev   | population
      --------+----------------------------+---------------+-
      ----+------------+------------
       CA     | Acme Mountain Gear         | Feinstein     |
      46 |   50000000 |   29760021
       CA     | Rockemwell Aerospace       | Feinstein     |
      46 | 2147483647 |   29760021
       IL     | Da Bulls Novelty Co.       | Moseley-Braun |
      47 |   24235683 |   11430602
```

```
 MD     | Northeast Burro Transport | Mikulski      |
49 |    8347902 |    4781468
(4 rows)
```

As we can see we can make the whole project a lot cleaner. At this point you do not need to edit the R code to get different results from the query, and then the same query could be used in another project in possibly a different language entirely.