R and SQL

Darren Homrighausen

10/06/2017

Introduction

These notes give an introduction to using SQL from within R. Data that looks like spreadsheets/matrices are known as a **flat databases** or **flat files**. SQL is a standard interface for storing/accessing **relational databases**.

Relational databases

Relational databases are a common way for information to be stored and distributed. In some cases, relational structures allow for much less redundancy. For instance, suppose I have a database with information about two different sections of a statistics class, each containing 3 students, and their (Post test) - (Pre test) scores.

Using a relational database, I would have two tables:

```
## Loading required package: knitr
```

```
## Warning: package 'knitr' was built under R version 3.4.1
```

Table 1: STAT101_1

ID	diff_score
s_101_1a	12
s_101_1b	5
s_101_1c	9

Table 2: STAT101_2

ID	diff_score
s_101_2a	4
s_101_2b	-3
s_101_2c	6

Converting this to a flat file would entail:

ID	Class	diff_score
s_101_1a	STAT101_1	12
s_101_1b	STAT101_1	5
s_101_1c	STAT101_1	9
s_101_2a	STAT101_2	4
s_101_2b	STAT101_2	-3
s_101_2c	STAT101_2	6

Notice that even in this tiny example, there is 1.5 times as much information stored, much of it redundant.

SQL in R

When *RSQLite* is installed, *SQLite* comes with it. There is another package *sqldf* which loads/downloads the relevant parts of both *RSQLite* and *SQLite*. This is what we are going to be referencing. Install it first (install.packages('sqldf')) and then load it into memory:

```
  require('sqldf')
```

```
## Loading required package: sqldf
## Warning: package 'sqldf' was built under R version 3.4.1
## Loading required package: gsubfn
## Loading required package: proto
## Loading required package: RSQLite
## Warning: package 'RSQLite' was built under R version 3.4.1
```

```
#Note: if you get an error like:
#  unable to load shared object '/Library/Frameworks/R.framework/Resources/modules//R_X11.so':
#Then, at least on Mac OS, you need to reinstall "XQuartz", which builds X11: https://www.xquartz.org/
#You'll need to log out/in and then open/close XQuartz (this re-builds the symlinks that the warning
#                                                        is referring to)
```

RSQLite's *dbConnect()* function opens a connection to a database. If the named database does not yet exist, one is created. The command below opens a connection to the *example.sqlite* database. If the database does not yet exist, one is created in R's working directory

```
db = dbConnect(SQLite(), dbname='example.sqlite')
getwd()
```

```
## [1] "/Users/darrenho/Box Sync/teaching/STAT6306/homeworks/homeworks/3homeworkSpam"
```

```
'example.sqlite' %in% dir()#This checks if example.sqlite is in cwd
```

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

Technically, a database hasn't been created, yet. Merely a connection to a place on the harddrive has been opened, along with a label for that connection. First, we have to make sure there isn't a Table with the same name (for instance, if we have run the code multiple times)

```
if('STAT101_1' %in% dbListTables(db)){dbRemoveTable(db, 'STAT101_1')}
if('STAT101_2' %in% dbListTables(db)){dbRemoveTable(db, 'STAT101_2')}
```

Now, let's put in the column names and data types via a **SQL query** :

```
dbSendQuery(conn = db,
       'CREATE TABLE STAT101_1
       (ID TEXT,Diff_score INTEGER)')
```

```
## <SQLiteResult>
##   SQL  CREATE TABLE STAT101_1
##        (ID TEXT,Diff_score INTEGER)
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

Now, we have a SQL database initialized, but it still doesn't have any data. First, we will manually add some observations

```
dbSendQuery(conn = db,
         "INSERT INTO STAT101_1
         VALUES ('s_101_1a',12)")
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL  INSERT INTO STAT101_1
##          VALUES ('s_101_1a',12)
##   ROWS Fetched: 0 [complete]
##        Changed: 1
```

```
dbSendQuery(conn = db,
         "INSERT INTO STAT101_1
         VALUES ('s_101_1b',5)")
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL  INSERT INTO STAT101_1
##          VALUES ('s_101_1b',5)
##   ROWS Fetched: 0 [complete]
##        Changed: 1
```

```
dbSendQuery(conn = db,
         "INSERT INTO STAT101_1
         VALUES ('s_101_1c',9)")
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL  INSERT INTO STAT101_1
##          VALUES ('s_101_1c',9)
##   ROWS Fetched: 0 [complete]
##        Changed: 1
```

Let's inspect this database to see what it looks like. Note that you wouldn't actually print out the data from a database in practice as it would be too much output.

```
dbListTables(db)
```

```
## Warning: Closing open result set, pending rows
```

```
## [1] "STAT101_1"
```

```
dbListFields(db, 'STAT101_1')
```

```
## [1] "ID"         "Diff_score"
```

```
dbReadTable(db, 'STAT101_1')
```

```
##         ID Diff_score
## 1 s_101_1a         12
## 2 s_101_1b          5
## 3 s_101_1c          9
```

```
dbDisconnect(db)
```

Likewise, we could create a second SQL table that has the information from STAT101_2.

In general, this procedure for populating the database is too cumbersome/error prone to use in practice. Instead, we can directly load existing (separate) flat files in databases. First, let's pretend we have two files on our hard drives from the two class (practically speaking, we are creating those files now)

```
tab1 = data.frame('ID'=c('s_101_1a','s_101_1b','s_101_1c'),
                  'diff_score' = c(12,5,9))
write.csv(tab1,file='STAT101_1.csv',row.names = FALSE)
tab2 = data.frame('ID'=c('s_101_2a','s_101_2b','s_101_2c'),
                  'diff_score' = c(4,-3,6))
write.csv(tab2,file='STAT101_2.csv',row.names = FALSE)
```

Let's load in the data from csv files. Again, we need to make sure the table names are unused (say, if this code has been run multiple times. Alternatively, you can use the 'overwrite = TRUE' parameter in *dbWriteTable* to, you guessed it, overwrite any existing information. Likewise, if you have a large flat file, you can load it in chunks and add it to the database via 'append = TRUE'):

```
db = dbConnect(SQLite(), dbname='example.sqlite')
if('STAT101_1' %in% dbListTables(db)){dbRemoveTable(db, 'STAT101_1')}
if('STAT101_2' %in% dbListTables(db)){dbRemoveTable(db, 'STAT101_2')}
```

```
STAT101_1 = read.csv(file='STAT101_1.csv')
dbWriteTable(conn = db, name = 'STAT101_1',
             value = STAT101_1, row.names = FALSE)
rm(STAT101_1)#removes object from R's memory as it is now saved in a data base
STAT101_2 = read.csv(file='STAT101_2.csv')
dbWriteTable(conn = db, name = 'STAT101_2',
             value = STAT101_2, row.names = FALSE)
rm(STAT101_2)
```

Let's inspect the results

```
dbListFields(db, 'STAT101_1')
```

```
## [1] "ID"         "diff_score"
```

```
dbReadTable(db, 'STAT101_1')
```

```
##          ID diff_score
## 1 s_101_1a         12
## 2 s_101_1b          5
## 3 s_101_1c          9
```

```
dbReadTable(db, 'STAT101_2')
```

```
##          ID diff_score
## 1 s_101_2a          4
## 2 s_101_2b         -3
## 3 s_101_2c          6
```

Now, if we want to run an analysis, we need to convert this relational data base back into a flat file:

```
STAT101_1 = dbReadTable(db, 'STAT101_1')
STAT101_2 = dbReadTable(db, 'STAT101_2')
(STAT101 = rbind(STAT101_1,STAT101_2))
```

```
##          ID diff_score
## 1 s_101_1a         12
## 2 s_101_1b          5
## 3 s_101_1c          9
## 4 s_101_2a           4
## 5 s_101_2b          -3
## 6 s_101_2c           6
```

Lastly, it is good practice to close connections after you are done with them:

```
dbDisconnect(db)
```