Federal Court Appeals Web Application

Tyler Olson Alex Zajichek

May 1, 2017

1 Introduction

1.1 Files contained in `appeals.zip`

2 Information on backend

This section is to provide insight into the backend of the application for troubleshooting, or future use.

2.1 SQL, SQLite, and `RSQLite`

There are many types of databases that are communicated with by SQL (Structured Query Language). This application uses SQLite, which is a very simple and convenient framework. Below are some preliminary steps to, if needed, create a database from scratch on the **Mac OS only**.

***Windows users**: You must download SQLite from www.sqlite.org and use the command shell (or some other interface) to create the database. However, you do **not** need to do this if you are only wanting to be a user of the `shiny` application.

2.1.1 Basic SQL commands

- CREATE TABLE - Creates a database table for data to be stored

  ```
  CREATE TABLE <tableName> (<column1> <dataType>, <column2> <dataType>,..)
  ```

  SQLite has a limited number of data types. For this application, the INTEGER and TEXT were the only data types used. If dates are desired, you can declare it as a TEXT data type, and insert your data in the form 'YYYY-MM-DD', which will allow date ranges to be maintained.

- INSERT - Insert single rows of data into the database

  ```
  INSERT INTO <tableName> VALUES (<val1>, <val2>, ...)
  ```

  In this form, you must supply an input value for all columns in the database in the same order they are created in the CREATE TABLE statement. There is more specific syntax to add to the INSERT statement that will allow you to list the columns to insert values for. TEXT data types must have quotes around the string upon insert, while INTEGERS do not.

- UPDATE - Update a set of records in a table

  ```
  UPDATE <tableName> SET <column1> = <val1>, <column2> = <val2>,..., WHERE <condition>
  ```

  The desired columns to be updated are the only ones that are needed to be listed for the supplied table name. A condition can be specified to only update a set (or single) row that satisfy the criteria. The same insert value formats hold for updating.

- SELECT - Query records from the data base

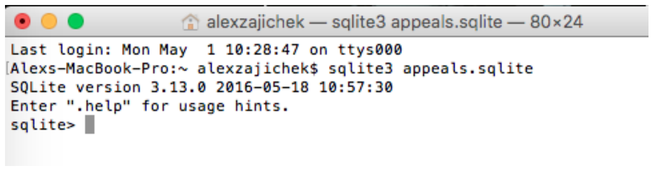
```
SELECT * FROM <tableName> WHERE <condition>
```

The asterisk is the simple way to return all columns into the result set. Specific column names can be listed with comma separation if only a subset are desired.

2.1.2 Setting up database in terminal

1. Open the terminal on your Mac, and initialize database.

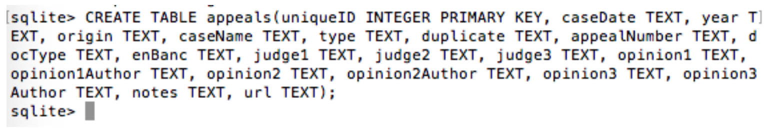
```
sqlite3 <databaseName>.sqlite
```



```
Last login: Mon May  1 10:28:47 on ttys000
Alexs-MacBook-Pro:~ alexzajichek$ sqlite3 appeals.sqlite
SQLite version 3.13.0 2016-05-18 10:57:30
Enter ".help" for usage hints.
sqlite> 
```

2. Create a table to store the data.

```
CREATE TABLE <tableName>(<column1> <dataType>, <column2> <dataType>,...)
```

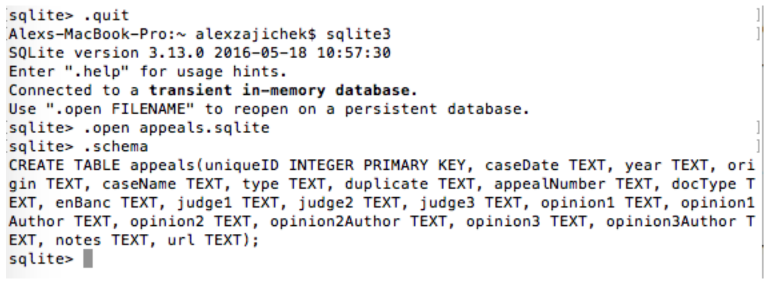


```
sqlite> CREATE TABLE appeals(uniqueID INTEGER PRIMARY KEY, caseDate TEXT, year T
EXT, origin TEXT, caseName TEXT, type TEXT, duplicate TEXT, appealNumber TEXT, d
ocType TEXT, enBanc TEXT, judge1 TEXT, judge2 TEXT, judge3 TEXT, opinion1 TEXT,
opinion1Author TEXT, opinion2 TEXT, opinion2Author TEXT, opinion3 TEXT, opinion3
Author TEXT, notes TEXT, url TEXT);
sqlite> 
```

All data types are TEXT, except the unique ID is an INTEGER PRIMARY KEY. This allows the database to enforce a constraint that every row has a *different* unique ID. It will not allow you to insert duplicate ones.

3. Closing and reopening the database

```
.quit
sqlite3
.open <databaseName>.sqlite
.schema
```



```
sqlite> .quit
Alexs-MacBook-Pro:~ alexzajichek$ sqlite3
SQLite version 3.13.0 2016-05-18 10:57:30
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open appeals.sqlite
sqlite> .schema
CREATE TABLE appeals(uniqueID INTEGER PRIMARY KEY, caseDate TEXT, year TEXT, ori
gin TEXT, caseName TEXT, type TEXT, duplicate TEXT, appealNumber TEXT, docType T
EXT, enBanc TEXT, judge1 TEXT, judge2 TEXT, judge3 TEXT, opinion1 TEXT, opinion1
Author TEXT, opinion2 TEXT, opinion2Author TEXT, opinion3 TEXT, opinion3Author T
EXT, notes TEXT, url TEXT);
sqlite> 
```

After reopening the database, the `.schema` command shows the tables that exist in the database.

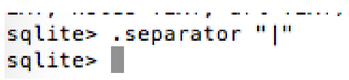
2.1.3 Inserting text file into the database

Now that the database table has been created, it is of interest to load it with data. The data loaded here was initially downloaded as a .csv file, and written back out to `appeals.txt` with '`|`' as the delimiter due to many characters being present. See `preliminaryTasks.R` for some minor preprocessing that was done on the original data.

1. Place data file (`appeals.txt`) in the same directory as the database, and then open the database in terminal

2. Change the delimiter in SQLite

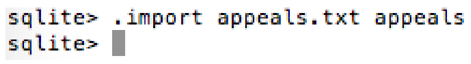
   ```
   .separator "<character>"
   ```

   ```
   sqlite> .separator "|"
   sqlite>
   ```

3. Import the data file

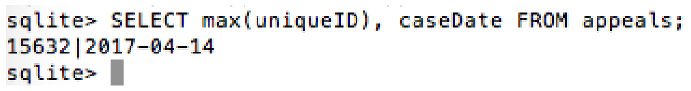
   ```
   .import <file>.txt <tableName>
   ```

   ```
   sqlite> .import appeals.txt appeals
   sqlite>
   ```

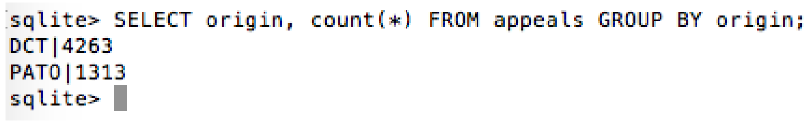
4. Run some test queries

   - Return the case date and ID for the row which has the maximum ID number.

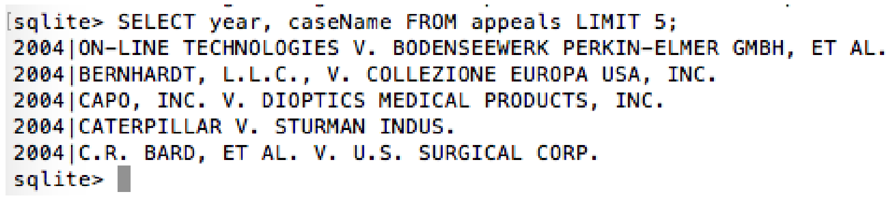
     ```
     sqlite> SELECT max(uniqueID), caseDate FROM appeals;
     15632|2017-04-14
     sqlite>
     ```

   - Count the number of rows in each level of origin

     ```
     sqlite> SELECT origin, count(*) FROM appeals GROUP BY origin;
     DCT|4263
     PATO|1313
     sqlite>
     ```

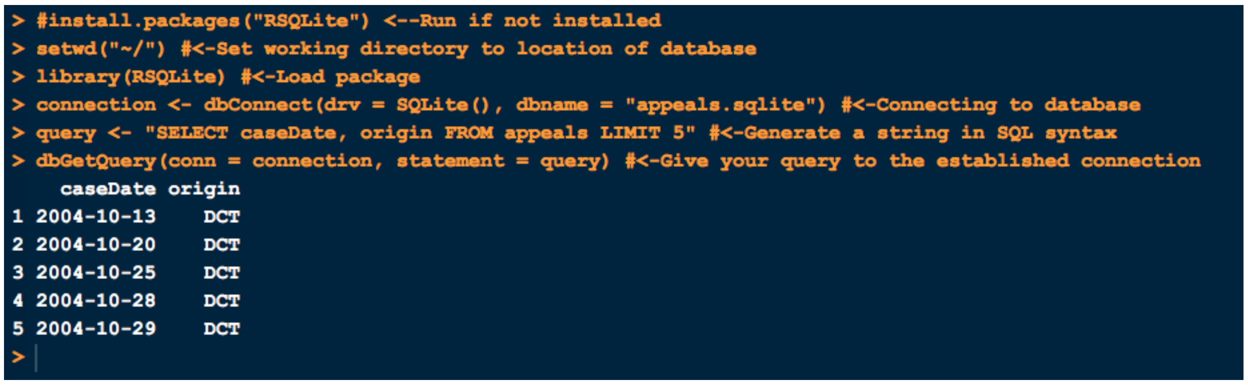
   - Get the year and case name of the first 5 records

     ```
     sqlite> SELECT year, caseName FROM appeals LIMIT 5;
     2004|ON-LINE TECHNOLOGIES V. BODENSEEWERK PERKIN-ELMER GMBH, ET AL.
     2004|BERNHARDT, L.L.C., V. COLLEZIONE EUROPA USA, INC.
     2004|CAPO, INC. V. DIOPTICS MEDICAL PRODUCTS, INC.
     2004|CATERPILLAR V. STURMAN INDUS.
     2004|C.R. BARD, ET AL. V. U.S. SURGICAL CORP.
     sqlite>
     ```

2.1.4 Example using the `RSQLite` package

The basic idea of how to use the `RSQLite` package in R, is that SQL statements will be created as character strings with the same exact syntax, and will simply be sent to the database. Below is a simple example in R.



```
> #install.packages("RSQLite") <--Run if not installed
> setwd("~/") #<-Set working directory to location of database
> library(RSQLite) #<-Load package
> connection <- dbConnect(drv = SQLite(), dbname = "appeals.sqlite") #<-Connecting to database
> query <- "SELECT caseDate, origin FROM appeals LIMIT 5" #<-Generate a string in SQL syntax
> dbGetQuery(conn = connection, statement = query) #<-Give your query to the established connection
    caseDate origin
1 2004-10-13    DCT
2 2004-10-20    DCT
3 2004-10-25    DCT
4 2004-10-28    DCT
5 2004-10-29    DCT
>
```

Result sets from sending a `SELECT` query will be an R data frame. There is only one other function used in the application:

```
> dbListFields(conn = connection, name = "appeals")
 [1] "uniqueID"       "caseDate"       "year"           "origin"
 [5] "caseName"       "type"           "duplicate"      "appealNumber"
 [9] "docType"        "enBanc"         "judge1"         "judge2"
[13] "judge3"         "opinion1"       "opinion1Author" "opinion2"
[17] "opinion2Author" "opinion3"       "opinion3Author" "notes"
[21] "url"
```

- Gives the column names stored in the supplied table name for a given connection.

3 Using the application

3.1 Initial setup

3.1.1 Installing `R` and `shiny`

3.1.2 Gathering files

Keep the following files together in a *single* directory (other files may be present):

1. `server.R` - Gives functionality to the application
2. `ui.R` - Creates the user interface
3. `appeals.sqlite` - Database containing all of the data

***I would suggest making copies of `appeals.sqlite` for backup, and storing most recent copies on Github, Dropbox, etc.**

3.1.3 Running the app

3.2 Interface

3.2.1 Query tab

3.2.2 Insert tab

3.2.3 Update tab

3.2.4 Visualize tab