

Federal Court Appeals Data - Web Application

Tyler Olson

Alex Zajichek

Prof. Rhonda DeCook

October 3, 2017

Contents

1	Files contained in <code>appeals.zip</code> folder	2
1.1	DatabaseApp Folder	2
1.2	Documentation Folder	2
1.3	SQLITE_ReferenceInformation Folder	3
2	Some SQL Background	3
2.1	SQL, SQLite, and RSQLite	3
2.2	Some Basic SQL commands	3
2.3	Example using the RSQLite package	4
3	‘Ground Zero’ creation of database from text file	5
4	Using the application	7
4.1	Initial setup	7
4.1.1	Installing R and other required packages (do this one time only)	7
4.1.2	Gathering files	7
4.1.3	Running the app	7
4.2	Interface	7
4.2.1	Query tab	7
4.2.2	Insert tab	8
4.2.3	Update tab	8
4.2.4	Visualize tab	8

1 Files contained in `appeals.zip` folder

1.1 DatabaseApp Folder

This folder contains the three necessary files for the database application to run: an SQLite database file and two R software source code files. The R files create the R shiny application or browser-based GUI that allows the user to interact (i.e. query, insert, update) with the database.

- **`appeals.sqlite`**

An original `.csv` file was used to create this SQLite database (`.sqlite` file extension). This is the dynamic database where all of the data are stored (and updated). If changes are made to the data through the database web application, they will be saved immediately in the `.sqlite` object. This file will not be opened directly, but will instead be accessed using SQL commands integrated into the R shiny application (i.e. the top-level GUI-user does not need to know SQL commands, instead the user's clicks are translated into SQL within the R shiny app).

Some recommendations/comments:

1. We strongly recommend frequently saving static copies of the database by either downloading the full database as a `.csv` file or saving the `.sqlite` file. This will allow you to have backups, and you can always return to a past version of the database, if desired. Names that include version information is best practice, e.g. `appealsYYYY-MM-DD.sqlite`.
2. The exact name of the database file, `appeals.sqlite`, is searched for when the database application is launched. Therefore it is important to leave your most up-to-date version with this same exact name in the **DatabaseApp** folder.

NOTE: If you would like a different version of the database to be used by the web application (e.g. `appealsYYYY-MM-DD.sqlite`), then you must change the requested database name in the `server.R` file in the `dbConnect` function toward the top of the file. This would most likely be done by Prof. DeCook.

- **`server.R`**

R software source code used to generate the R shiny application. Enables the functionality of the application, and allows communication with the database. This file works in conjunction with the `ui.R` file. Inclusion of new fields in the database would require modifying this file.

- **`ui.R`**

R software source code for the web application that controls the allowable inputs (i.e. dropdown lists) and format of the user interface (i.e. appearance of the GUI) for the web application. This file works in conjunction with the `server.R` file. Inclusion of new fields in the database would require modifying this file.

1.2 Documentation Folder

- **`AppealsDBDocumentation.pdf`**

The documentation coinciding with database web application. It is the document you are reading right now.

- **TexFolder**

Folder containing all relevant files used to generate the documentation PDF `AppealsDBDocumentation.pdf`. The PDF is generated using LaTeX and the files consist of graphics files, the `.tex` source code, and other files generated upon compilation. Changes to documentation can be made by changing the `.tex` source code. Once revisions are completed, replace the `AppealsDBDocumentation.pdf` in the Documentation folder with the newest version.

1.3 SQLITE Reference Information Folder

- preliminaryTasks_CSV_to_TXT_to_SQLITE.R

The original PATO and DCT spreadsheets were merged into a single `.csv` file (same columns, rows merged), and then converted into a `.sqlite` database file. This process started by running this R script on the `.csv` file, which applies necessary preliminary steps to ensure the data were in the correct format for database creation (e.g. date formats, delimiters, etc.). The last line of this R file writes-out the `appeals.txt` file, which is another file included in this reference information folder. It is this `.txt` file that is then converted to the `.sqlite` database file (see Sections 2.1.2 and 2.1.3).

- appeals.txt

Originally, this file contained the data from the most recent spreadsheets sent by Professor Rantanen on 4/19/2017. As the database has been updated using the app, and exported as a new `.csv` files, this file is meant to hold the most recent database in `.txt` format (see Section 2.1 for steps on recreating the database from this file, if necessary.)

- sqliteAPPEALS-CREATE_TABLE.txt

This text file contains miscellaneous comments and code for the programmer related to creating the database from a `.txt` file. Specifically, this text file includes the long “CREATE TABLE” statement used to create the sqlite database table, and this character string can be copied and pasted into the `sqlite>` command line for table creation (see Section 2.1.2).

2 Some SQL Background

This section is intended for R programmers who may be maintaining the `DatabaseApp`. If you are a user of the R shiny, you do not need to be familiar with any SQL language, which is the topic of this section.

2.1 SQL, SQLite, and RSQLite

There are many types of databases that are communicated with by SQL (Structured Query Language). This `DatabaseApp` application uses SQLite, which is a very simple and convenient framework. Tools from the `RSQLite` package are used allowing for convenient connection to an SQL database from within the R environment. In the following, we give a few examples of using SQL commands

***MAC OS users:** SQLite should be immediately available from your terminal command line.

***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 R shiny application.

2.2 Some Basic SQL commands

These are a few of the SQL commands that are utilized within the app. Again, the top-level user does not need to know such SQL commands because the app translates the user’s clicks into SQL commands automatically.

- 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 (i.e. Federal Circuit Decisions Database app), 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 a single row of data (i.e. a record) 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.

- Important Comment on TEXT entries with apostrophes:

As text must be entered with quotes, this poses an issue in SQL. The text provided by the user that contains an apostrophe (e.g. O'Malley) must be sanitized prior to translating to the SQL command. There are a variety of ways to deal with this, but if the input is not adjusted prior to entering into the app, the app will crash. See references in R for `dplyr::escape` as an example. See equivalent custom function called `sql_escape` used in the `server.R` file.

- 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 satisfies 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.

2.3 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 in R with the exact SQL 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"
```

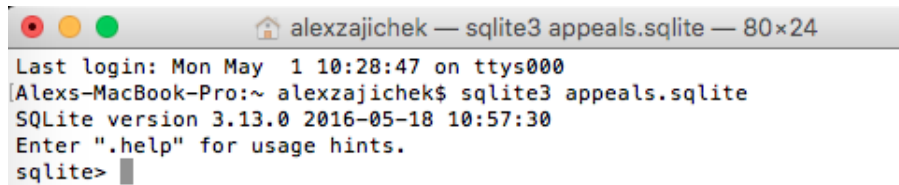
- Gives the column names stored in the supplied table name for a given connection.

3 ‘Ground Zero’ creation of database from text file

The original data provided by Prof. Rantanen on 4/19/2017 was in .csv format. This was converted into a .sqlite file in order to be used with the DatabaseApp. As the information in the database will continue to be updated, the app will be used to periodically export the database information as a .csv file, and this section provides information for translating the .csv into a .sqlite file.

1. Run the .csv file through the preliminary tasks in R as stated in Section 1.3 to create the `appeals.txt` file.
2. Open the terminal on your Mac and change the directory to the one containing `appeals.txt`. Then, initialize the database by opening `sqlite` and stating the the database name. Generically, it looks like `sqlite3 <database>.sqlite`. For this specific situation, we have...

```
$ sqlite3 appeals.sqlite
```



3. Create the table that will store the database information.

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

NOTE: For convenience, you can copy and paste the long SQLite command shown below from the file called `sqliteAPPEALS_CREATE_TABLE.txt`. If new columns have been added to the .csv file, then the command shown below must be edited to reflect the change.

```
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>
```

For this database, all data types are TEXT, except the unique ID which 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.

4. SIDENOTE: Closing database via closing `sqlite` and reopening `sqlite` and the database.

If you like, you can close the database and `sqlite`, and open the existing database again at a later time...

```
sqlite> .quit
$ sqlite3
sqlite> .open <databaseName>.sqlite
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 opened database.

5. Insert the text file into the created database.

Now that the database table has been created, we can load it with the data in the `appeals.txt` file (see Section 1.3 on appropriate pre-processing of the `.csv` file).

- (a) Place the data file (`appeals.txt`) in the same directory as the database file `appeals.sqlite`.
- (b) From a terminal window, change to the above directory.
- (c) Open the database in `sqlite` from this directory (see 4. above).
- (d) Change the delimiter in SQLite to the appropriate character for the `appeals.txt` file (see Section 1.3).

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

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

- (e) Import the data file into the database.

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

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

- (f) 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
PAT0|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> █
```

The SQLite file called `appeals.sqlite` can now be used in conjunction with the `ui.R` and `server.R` files to run the `DatabaseApp`.

4 Using the application

4.1 Initial setup

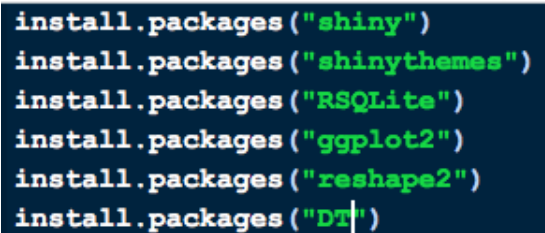
4.1.1 Installing R and other required packages (do this one time only)

To install R, go to <https://cran.r-project.org/>. RStudio can also be downloaded at <https://www.rstudio.com/products/rstudio/download2/>, but is not required for use of this application. Once installed, there are a few packages in R that will need to be downloaded in order for the application to run.

To install a package, open R and type

```
> install.packages(<packageName>)
```

The packages needed to run this application are `shiny`, `RSQLite`, `shinythemes`, `reshape2`, `ggplot2`, and `DT`. Below is a screenshot:



```
install.packages("shiny")
install.packages("shinythemes")
install.packages("RSQLite")
install.packages("ggplot2")
install.packages("reshape2")
install.packages("DT")
```

Type each line into the R console, hitting enter in between. **You will only need to install these packages once (unless R is updated or uninstalled), and then this step can be skipped when running the application.**

4.1.2 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.**

4.1.3 Running the app

Once the `server.R` and `ui.R` files have been stored together within a single directory, the application can be launched a few different ways. If you are using the standard R interface, `RGui`, the working directory must be changed to the location where the `server.R` and `ui.R` files are stored. This can be done by selecting **File** → **Change dir...**, and choosing the appropriate folder. Typing `runApp()` within the R console and hitting enter will now launch the application.

If you are using `RStudio` rather than `RGui`, which is recommended, a similar series of steps is required. To specify the appropriate working directory, select **Session** → **Set Working Directory** → **Choose Directory**. Running the `runApp()` command in the console will launch the application. Alternatively, if the `server.R` or `ui.R` files are open in the R script window, `RStudio` automatically recognizes that the file corresponds to a **Shiny** application and provides a **Run App** button in the right-hand corner of the window.

4.2 Interface

4.2.1 Query tab

When the application is initially launched, the **Query Data** tab is the default landing page. No data will appear until the **Display** tab is selected. The **Filter** tab allows users to subset the data according to different variable levels. Multiple levels can be specified by simply selecting the desired levels one at a time. The default date range is from the earliest date in the database to today's date. If data are inserted into the database that contain a new variable level, this level will automatically be added to the drop-down list.

The **Display** tab identifies which columns are shown in the table within the main panel of the page. The *ID*, *Case Date*, *Origin*, and *Case Name* variables are selected and displayed by default. Checking the box corresponding

to a particular variable adds the variable column to the table, and unchecking the box removes the column from the table. Once the data has been subset according to specific variables and levels, the subset being displayed in the table can be downloaded as a .csv file by clicking the **Download** button located at the bottom of the page.

4.2.2 Insert tab

The **Insert Record** functionality allows users to manually enter information regarding new cases and add this information to the database. The *New Record ID* located in the upper right-hand corner is automatically generated by the application and provides a unique identifier for each row. The *Session Insertion Count* keeps track of the number of insertions performed by a single user during the current session. The *Case Date* and *Duplicate?* fields default to the current date and “No.” All other fields are initially blank. If a field is left empty, a null will be used as the placeholder within the database. If a mistake is made during the insertion process, the **Update Record** tab can be used to rectify the error.

4.2.3 Update tab

The **Update** tab allows the user to edit any field in the database according to its unique ID value. This functionality gives the user the ability to maintain accurate data, even if it initially is entered wrong upon **Insert**.

After identifying a record that needs updates from the **Query** tab, simply type the unique ID into the text box in the **Update** tab and press “Get Record”. If the record requested does not have the supplied ID, a message will appear telling the user ‘ID XXXXX not found’. Otherwise, text boxes will be displayed, which are auto-populated with the current values of each column in the database associated with the supplied ID. The user can then edit any of columns as they wish. Once all edits are made on the record, scroll to the bottom of the page and click “Update”. The new data will be sent to the database, and be stored with the associated ID. The text boxes will then be cleared, and a confirmation message at the top displays ‘Record XXXXX updated.’

The user may then go back to the **Query** tab to confirm that the edit was a success. If it was not, simply go back to the **Update** tab and edit again.

Note: All date entries being sent to the database in either the **Insert** tab or **Update** tab must be of the form ‘YYYY-MM-DD’

4.2.4 Visualize tab

This tool creates bivariate stacked bar charts with the count/frequency as the y-variable. *Variable 1* is plotted on the x-axis, and the colors correspond to the different levels of *Variable 2*. The data for these plots is pulled directly from the current subset of data displayed in the **Query Data** tab. Therefore, the variables of interest must first be selected in the **Filter** tab in order to successfully generate bar plots. Plots can only be generated for categorical variables with a limited number of categories. The following variables cannot be visualized on either axis: *ID*, *Case Date*, *Case Name*, *Appeal Number*, *Notes*, and *URL*.