Shiny with Databases

Robert Kahne & Adam Sampson

Splash Analytics

2019-08-28

Reading flat files into R dataframes to store/filter/query your Shiny application data is a lot like using Excel to build a model to predict who will win the championship tournament.

Sure, you can do it ... But there's a better way.

Why Splash Uses Databases With Shiny

Overview

Why

- Speed
  - Loading
  - Indexes and Query Optimization
- Memory
  - Large Files
  - Multiple Users
- User Inputs
  - Persistent
  - Cross-session
  - Tracking User Activity

How

- Databases
  - Database Options
  - Database Optimization
  - Data Validations
- dplyr
  - Connecting to Database
  - Database Queries
  - Compute
  - Collect
  - Writing to Database

Using Databases for Speed

Databases can provide multiple speed advantages:

- Initial app load-time is faster
- Only load subsets of data from disk when/if they are neeeded
- Index data to improve filtering, sorting, and grouped summarizations
- Automated query optimization by SQL engine
- Lots of work has been done to optimize databases ... don't reinvent the wheel

Using Databases for Speed

Metric	db_time	readr_time	vroom_time
Initial Startup	0.02	5.69	1.34
Filter Crime Data	0.06	0.05	0.44 (0.08)
Plot Map	0.03	0.02	0.02
Join Crime and Weather Data	2.31	0.16	0.27 (0.18)
Plot Lineplot	0.02	0.02	0.02
Total	2.44	5.94	2.09

Based on bigTiming.R in Example App folder

Keep in mind, we are using sqlite for this demonstration. Using a hosted database would make the db speed even faster.

Using Databases for Memory

To store data in memory or to store data on the hard drive?

- R stores all data in memory (by default) for each instance
- Lots of work has been done to optimize databases ... don't reinvent the wheel
  - SQL has advanced caching to determine which data to store in memory and which to leave stored on disk
  - SQL caches work for all users simultaneously

Using Databases for Memory

But what about [data.table / feather / vroom]!?

- If you have a reason not to use a database then these might be helpful
  - Shiny server is read-only
  - You have to store data over network with lots of lag and aren't allowed to run a SQL server (can only use SQLite)
  - Your app will never have too much data/users
- Otherwise, if you want to try to do better than decades of computer scientist, you do you ...

Using Databases for Memory

Metric	db_mem	readr_mem	vroom_mem
Initial Startup	134.01	542.32	104.62
Filter Crime Data	116.36	516.24	110.99
Plot Map	117.3	518.82	112.92
Join Crime and Weather Data	109.99	515.76	115.05
Plot Lineplot	111.18	517.23	116.39

Based on bigTiming.R in Example App folder

User Inputs

Databases allow collection of data which can be stored for later use.

- Collect inputs which can be stored across sessions.
- Annotate data/plots/tables with information that can be shared across users.
- User data can be analyzed for performance improvement or finding other insights.

User Inputs - Database Validation Rules

What happens if a user enters an invalid response?

- In R you can check to make sure they never enter an invalid response
- In SQL you can define requirements for inputs to prevent an incorrect entry from being uploaded
- Don't forget to have R check for SQL injection attacks (enabled by default in `dbplyr`)

User Inputs - Concurrent Users

What happens if two users are both using an app to interactively enter information?

- With flat files users may overwrite each other's edits
- With flat files might get an error if trying to write from two instances at once
- Databases are made for concurrent requests
- Design databases to show most current input, but keep all inputs for a trail of activity
- `reactivePoll` can automatically update data

Example App - Big[ish] Data

For this app we will use:

The Louisville Crime Dataset

The Bowman Field Weather ISD Lite Dataset

1. Populate Shiny drop-down to select options
2. Assign the data to R object
3. When user selects a crime, generate plots for:
    - County map
    - Temperature to crime count lineplot

Connecting to Tables in a Database

```
db <- dbConnect(RSQLite::SQLite(), "big_app.sqlite")

crime_tbl <- tbl(db,'louisvilleCrimeData')
weather_tbl <- tbl(db,'louisvilleWeatherData')
comment_tbl <- tbl(db,'comments')
```

Using Collect to Get Data from Database

```
uniqueCrimeList <- crime_tbl %>%
    select(crime_type) %>%
    distinct() %>%
    collect() %>%
    pull(crime_type)

output$dynamicControls <- renderUI({
      selectInput(
        "crimeSelection",
        "Select Crime[s]:",
        # choices = c("All",uniqueCrimeList),
        choices = uniqueCrimeList,
        selected = c(NULL),
        multiple = TRUE
      ),
      div(
        style="text-align:center",
        "Note: empty selection == 'All'"
      )
    )
  })
```

Using Compute to Only Compute Once

```
filteredCrimeData <- eventReactive(list(selectedCrimeTypes(),
                                        selectedStartDate(),
                                        selectedEndDate()),{
  # Sometimes dbplyr doesn't like variable()
  crime_filter <- selectedCrimeTypes()
  start_date <- date_to_sqlite_numeric(selectedStartDate())
  end_date <- date_to_sqlite_numeric(selectedEndDate())

  # Iff a crime filter is required prepare it.
  if(crime_filter[[1]] != "All"){
    out <- crime_tbl %>% filter(crime_type %in% crime_filter)
  } else {out <- crime_tbl # No filter needed if "All"}

  # Finish preparing filters and then compute
  out <- out %>%
    select(date_occured,year,month,day,hour,crime_type,zip_code) %>%
    filter(zip_code %in% zip_filter) %>%
    filter(date_occured >= start_date) %>%
    filter(date_occured <= end_date) %>%
    compute()
  return(out)
})
```

Joining Data Between Tables

```
crimeDataWithTemp <- eventReactive(filteredCrimeData(),{
  out <- filteredCrimeData() %>%
    left_join(
      weather_tbl %>%
        select(year,month,day,hour,air_temp_c),
      by = c("year" = "year",
             "month" = "month",
             "day" = "day",
             "hour" = "hour")
    )
  return(out)
})
```

Joining Small Local to Large Remote

```
localSmall <- tibble(crime = c("DUI",
                               "DRUGS/ALCOHOL VIOLATIONS",
                               "DISTURBING THE PEACE"))

myJoin <- filteredCrimeData() %>%
  inner_join(localSmall,
    by = c("crime_type" = "crime"),
    copy = TRUE
  )
```

Joining Small Remote to Large Local

```
output$louisvilleMap <- renderLeaflet({
    shapesToPlot <- louisville_shapefile %>%
      left_join(
        filteredCrimeData() %>%
          select(zip_code) %>%
          group_by(zip_code) %>%
          summarise(crimes = n()) %>%
          collect(),
        by = c("ZCTA5CE10" = "zip_code")
      )
    pal <- colorNumeric("OrRd",
                        domain = shapesToPlot$crimes,
                        reverse = FALSE)
    leaflet::leaflet(shapesToPlot) %>%
      # addTiles() %>%
      addProviderTiles(providers$CartoDB.Positron) %>%
      addPolygons(fillOpacity = 0.75,
                  color = ~pal(crimes),
                  weight = 1)
  })
```

Uploading New Data to Database

```
check_comment <- validateComment(input$commentCrimeType,
                                 input$commentUserName,
                                 input$commentText,
                                 input$readyToSubmit)
if(check_comment){
  toSave <- tibble(
        datetime = Sys.time(),
        CrimeType = input$commentCrimeType,
        UserName = input$commentUserName,
        comment = input$commentText
      )
      dbWriteTable(db,'comments',toSave,append = TRUE)
      showModal(
        modalDialog(
          title = "Comment Uploaded",
          "Your comment has been uploaded. Thanks!",
          easyClose = TRUE
        )
      )
}
```

Polling for New Data

```
comment_data <- reactivePoll(
    5000, # Check every 5 seconds
    session,
    checkFunc = function(){
      # if this tally value changes
      comment_tbl %>% tally() %>% pull()
    },
    valueFunc = function(){
      # Do this if it changed
      comment_tbl %>% collect()
    }
  )
```

Fin

Questions???

Additional SQLite Help

```
sqlite_numeric_to_date <- function(number.in) {
    as_datetime(as.numeric(number.in),
                origin = "1970-01-01",
                tz = "GMT") #SQLite defaults to UCT/GMT
  }

date_to_sqlite_numeric <- function(date.in) {
    as.numeric(date.in)
  }
```