Project Write Up

CS340 Dr. Willhoft

4/30/2021

Liam Fisher

# Executive Overview

The Goal of this project was to create a simplified Covid-19 dashboard for the United States. Dashboards are often loaded with too much functionality and complex charting. This makes them hard to navigate and interpret, especially for someone not experienced in data related fields. This dashboard is simplified and contains minimal functionality and charting. The outcome of this is that it is easier to navigate and draw insights from.

The dashboard pulls data from a MySQL database. The MySQL database is fed data from a Covid data API. The data is not prime for visualization as it is from the source, so MySQL allows modifications and additions to the retrieved data to be stored and used by the application.

## Technologies

R Shiny: Application software.

Shiny Apps: Server hosting application.

MySQL: DBMS.

Amazon Web Services RDS: Server hosting database.

## Data Source

https://covidtracking.com/data/api/

Open US Covid Data.

No API key required, open data.

## Link to Dashboard

<https://liam-fishers-shiny.shinyapps.io/coviddashboard/>

## Project Architecture

Diagram

Description automatically generated

# Database

## Database Diagram

Table

Description automatically generated

Functional Dependencies and Keys

statemetasecond

Key's: state

FD's: state → covid19Site, twitter, nameOfState, lat, long

histAmerica

Key's: dateDay

FD's: dateDay → states

dateDay, states → positive, negative, pending, hospitalizedCurrently hospitalizedCumulative inIcuCurrently, inIcuCumulative, onVentilatorCurrently, onVentilatorCumulative, death, hospitalized, totalTestResults

histState

Key's: state, dateDay

FD's: state, dateDay → positive, probableCases, negative, pending, totalTestResultsSource, totalTestResults, hospitalizedCurrently, hospitalizedCumulative, inIcuCurrently, inIcuCumulative, onVentilatorCurrently, onVentilatorCumulative, recovered, death, deathConfirmed

## Evaluation of Normal Forms

All relations are in Fourth Normal Form. No MVDs, FDs with non-key left sides, and transitive dependencies.

# Development Discussion

Data Input / Output:

R has functions in the RMariaDB library that make data input and output from a MySQL database very easy. dbGetQuery(con, …) allows you to pass an SQL query as a string to the database. It will then return the result as a dataframe ready for the R environment. Similarly, dbWriteTable(con, …) allows you to supply a dataframe and the name of a relation to write the dataframe to. It will make sure the dataframe matches the relation and append the data to the end of the SQL relation.

Where the bugs occurred was in setting up the download data feature to allow the user to download the data for themselves. Downloading the data as CSVs was fine but downloading to XML was not. The histState table would not convert to XML because of its size. Therefore, the dashboard only allows the other relations to be downloaded as XML.

Data:

Setting up the histState, and histAmerica relations were not challenging. That was just a matter of converting the table as it was from the data source minus a few unnecessary columns. The statemetasecond table that houses meta information about each state was the difficult one. To create the US Covid Map on the main page of the dashboard, the latitude and longitude of each state was required. For this, I had to go to the web and download a CSV containing the lat and long of each state. then merge the state meta table from the Covid API with the lat and long CSV. This merged data was then written to the database.

# Project Evaluation

Data:

The data source that I am using was deprecated as of March 7th, 2021. I need to find a replacement source of Covid data.

App Functionality:

The main purpose of the application was to make a simplified Covid dashboard that would be easier for general audiences to use. I have no proof, however, that I have accomplished this. I would like to test the dashboard on novice users and get feedback on ways that it could be made easier to use.

Security:

In the future, I would like to implement a login feature. This would require, however, that I enhance the security of other areas of the application. Both Shinyapps and Amazon Web Services use protected environments, and Shinyapps provides HTTPS certification for you. The security hole of the app is the connection from the application server to the database. I am only using password authentication and the connection has no encryption. If I were to implement a login feature, the DB connection would have to be secured. I would have to set up SSL or TLS on the database connection.

Code:

There are two main parts of the code that require an update. Firstly, the update button updates the database but does not refresh the app. The user must know to press the update data button then refresh the app to see the updated data. I should enhance the update date button to also refresh the app. This will make the updated data available to the user without having to manually refresh the app. Secondly, the code does not render the Twitter handles and Covid website's URLs Iteratively. There is a separate line of code for each state to prepare the Twitter handles and hyperlinks. The code would be much more readable if this were iterative.

# Appendix

## Database Schema

CREATE TABLE statemetasecond (

state VARCHAR(2),

covid19Site TEXT,

twitter VARCHAR(16),

nameOfState VARCHAR(24),

lat DOUBLE,

long DOUBLE);

CREATE TABLE histAmerica (

dateDay INT PRIMARY KEY,

states INT,

positive INT,

negative INT,

pending INT,

hospitalizedCurrently INT,

hospitalizedCumulative INT,

inIcuCurrently INT,

inIcuCumulative INT,

onVentilatorCurrently INT,

onVentilatorCumulative INT,

death INT,

hospitalized INT,

totalTestResults INT);

CREATE TABLE histState (

dateDay INT,

state CHAR(2),

positive INT,

probableCases INT,

negative INT,

pending INT,

totalTestResultsSource VARCHAR(100),

totalTestResults INT,

hospitalizedCurrently INT,

hospitalizedCumulative INT,

inIcuCurrently INT,

inIcuCumulative INT,

onVentilatorCurrently INT,

onVentilatorCumulative INT,

recovered INT,

death INT,

deathConfirmed INT,

PRIMARY KEY (dateDay, state));

ALTER TABLE histState

ADD CONSTRAINT stateConstraint

FOREIGN KEY (state) REFERENCES StateMeta(state);

CREATE INDEX idxstname ON statemetasecond (nameOfState);

CREATE INDEX idxst ON statemetasecond (state);

## Example Data

statemetasecond



histAmerica

 Cont.



histState



Cont.



Cont.



## Code Samples of Database Related features

Connects the application to the MySQL database.

con <- dbConnect(MariaDB(),

host = "mycoviddb3.cpzqhak4dus6.us-east-2.rds.amazonaws.com",

port = 3306,

dbname = "mydb",

user = "admin",

password = "cs340dbpw")

Closes connection to MySQL database when app is closed.

onSessionEnded(function() {

dbDisconnect(con)

})

Creates the function which grabs the historical Covid data for the whole US from the API. Data comes in as a JSON and is converted to a Tibble.

hist\_america <- function() {

url <- 'https://api.covidtracking.com/v1/us/daily.json'

r <- GET(url)

data = fromJSON(rawToChar(r$content))

return(as\_tibble(data) %>% select(1:11, 13:15) %>% rename(dateDay = date))

}

Creates the function which grabs the historical Covid data for each state from the API. Data comes in as a JSON and is converted to a Tibble.

hist\_state <- function() {

url <- 'https://api.covidtracking.com/v1/states/daily.json'

r <- GET(url)

data = fromJSON(rawToChar(r$content))

return(as\_tibble(data) %>% select(1:15, 19, 27) %>% rename(dateDay = date))

}

Function for updating the database. First it Queries the database and figures out what day the database has data up to. Then it grabs the data from the API and writes the data that is missing to the database.

update\_data <- function() {

us\_date <- as.integer(dbGetQuery(con, "SELECT MAX(dateDay) FROM histAmerica;")[1,1])

states\_date <- as.integer(dbGetQuery(con, "SELECT MAX(dateDay) FROM histState;")[1,1])

us\_data <- hist\_america() %>% filter(dateDay > us\_date)

states\_data <- hist\_state() %>% filter(dateDay > states\_date)

dbBegin(con)

dbWriteTable(con, "histAmerica", us\_data, append = TRUE)

dbWriteTable(con, "histState", states\_data, append = TRUE)

dbCommit(con)

}

Queries the database and grabs the three relations and converts them to Tibbles.

statemetasecond <- as\_tibble(dbGetQuery(con, "SELECT \* FROM statemetasecond;"))

americaData <- as\_tibble(dbGetQuery(con, "SELECT \* FROM histAmerica;"))

histState <- as\_tibble(dbGetQuery(con, "SELECT \* FROM histState;")) %>% select(1,2,3,9,10,16)

Grabs the twitter handle from the database of a certain state. Renders the result to text in preparation for the UI. This code runs for every state.

output$stateTwitterAlaska <-

renderText({paste("Alaska Twitter: ",as\_tibble(dbGetQuery(con,

paste0("SELECT twitter

FROM statemetasecond

WHERE nameOfState = '",

"Alaska",

"';")))[[1,1]])})

Grabs the Covid website URL from the database of a certain state. Renders the result to a hyperlink in preparation for the UI. This code runs for every state.

output$websiteURLAlaska <-

renderUI({tags$a(href = as\_tibble(dbGetQuery(con,

paste0("SELECT covid19Site

FROM statemetasecond

WHERE nameOfState = '",

"Alaska",

"';")))[[1,1]],

paste0("Alaska", " Covid Website"))})

Converts the table selected by the user to a CSV file and downloads it to their local machine.

output$downloadDataCSV <- downloadHandler(

filename = function() {

paste(input$datasetCSV, ".csv", sep = "")

},

content = function(file) {

write.csv(datasetInputCSV(), file, row.names = FALSE)

}

)

Converts the table selected by the user to an XML file and downloads it to their local machine. Does not allow downloading of the histState table as it takes too long to render to XML.

output$downloadDataXML <- downloadHandler(

filename = function() {

paste(input$datasetXML, ".xml", sep = "")

},

content = function(file) {

write.xml(as.data.frame(datasetInputXML()), file)

}

)