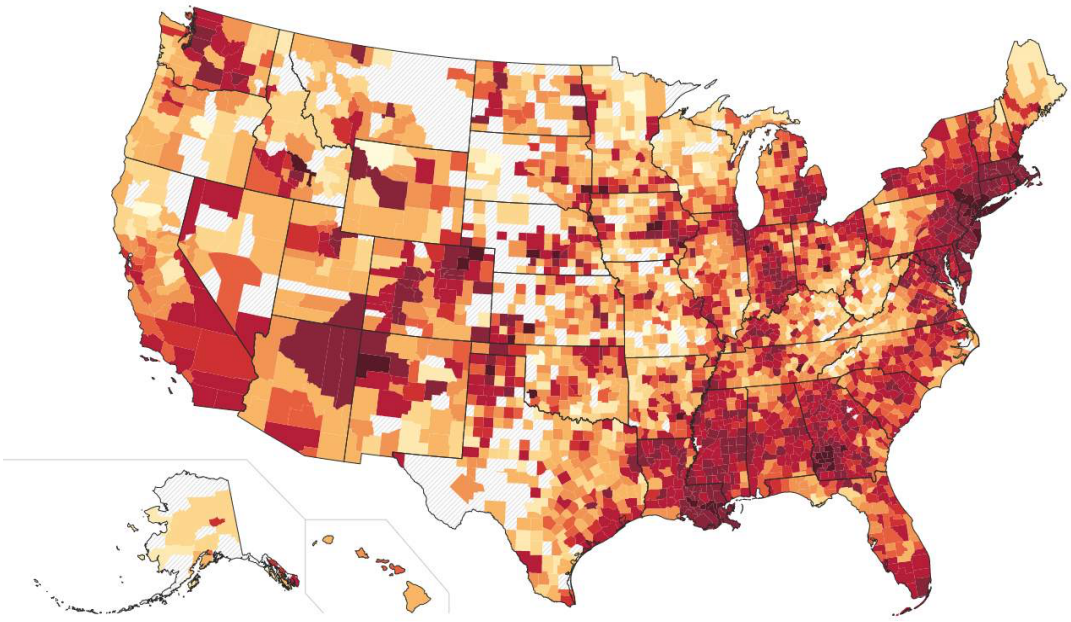
# Coronavirus Archive:
# A Look at the Last Few Months

**Ryan Burczak and Brandon Lewis**
**CS4430 Final Project Report**

## Abstract

The purpose of this project was to create a program that can connect to, collect, and process data from a MySQL database. Our goal of this project was to help users view where the Coronavirus was affecting the country, and where the most cases and deaths were occurring. In this report, we will discuss what was implemented into this application, also how and why this project was created.

## Introduction

The novel coronavirus SARS-CoV-2 is the cause of the coronavirus disease 2019 (COVID-19) and both the virus and the disease have spread throughout the United States. Its presence affects many aspects of our daily lives, from the activities we participate in to who we keep in touch with, and how. There is no cure for COVID-19. It is highly transmissible and it's hard to avoid. Stories of COVID-19 dominate the news cycle; be it updates on the number of cases and deaths, or various stories of people who got sick or died from this pandemic. Many countries, after a few months, were able to flatten the curve and get the virus under control. The United States, unfortunately, has not.

This project, created by Database Management Systems students Ryan Burczak and Brandon Lewis, seeks to show curious users the statistics of the pandemic between March 24 and July 21.

## Technology Implemented

1. **Java Eclipse**
2. **Java GUI**
3. **MySQL Databases**
4. **Excel Spreadsheets**

## What's in the Project

Because COVID-19 is such a pervasive issue in people's lives, we based our Database Management Systems project on that. It would have been ideal to find a way to automatically collect data in real time like most COVID-19 tracking sites, but that was an impossibility. However, we were able to access early COVID-19 data from Johns Hopkins University which was downloadable from GitHub. As a result, we decided to make our project an archive of early coronavirus data. Johns Hopkins University has coronavirus data for the United States arranged two different ways: by state only, and by state and county. To provide for the most geographically specific data queries, we

decided to collect the data by state and county.

According to worldometer.com, daily data shows that every Tuesday, there is a spike in the number of reported coronavirus cases and related deaths. Tuesday is the day of the week when most tracking sites catch up from incomplete reporting over the weekend. So, we decided to import the data from every Tuesday from March 24 to July 21. March 21 was the earliest day that Johns Hopkins started keeping tallies of COVID-19 cases, deaths, recoveries, and other information by county in the United States. The reason we stopped collecting data after July 21 is quite complex. When downloading the data from GitHub, the spreadsheets from every day leading up to the day before the data are downloaded. One cannot download a single spreadsheet at a time. We downloaded the data on July 24, so that marks the endpoint of our data.

Before loading the data into MySQL, it had to be processed on Microsoft Excel. There were 12 columns of data on each spreadsheet, but we only needed 7 for our project. The 5 columns of data irrelevant to the queries and other calculations we needed to perform were eliminated: FIPS, latitude, longitude, country (this project is only monitoring cases in the United States), and the redundant “combined key” (the county name and state, separated by a comma). Additionally, the format of each column needed to be manually set (Number for the confirmed cases, deaths, recoveries, and active cases, Date for the latest update, and General for the county and state names). Without this formatting, MySQL would reject new spreadsheets. Eventually, another way was found to import the new spreadsheets, where MySQL automatically created the variables and column names.

The coronavirus data from the 18 weeks spanning March 24 to July 21 was arranged into 18 tables. Each table held the data from a certain week, and was labeled with each respective Tuesday. The option of loading all 18 weeks worth of data into a single table was considered. However, there are two reasons we decided against it. First, we weren’t sure that MySQL could handle over 50000 tuples at a time. Second, if a user wanted to look up data for a particular county on a given day, the user may end up seeing the data for that one county for all the days logged in the database. By keeping the daily data separate, this mistake could be circumvented. Even with each day’s data in its own separate table, a user could still perform queries that require the acquisition of data from more than a single day. For example: How many new cases arose in St. Tammany Parish, Louisiana between March 31 and April 21? Which counties had more confirmed cases on April 7 than Bexar County, Texas did on March 24?

We came up with a series of sample queries or questions like the previously mentioned two that a typical user might ask using the database. In preparation for integrating these queries into Java, we first tested them on MySQL. Java allows for SQL queries with a connection to the desired database, and we were able to implement that. Afterwards, we explored options for displaying the queries on our Java GUI (Graphical User Interface). We both have experience in Python, but remembered using a Java SQL connector in a previous homework assignment and felt it would be useful in this project. We decided to implement our project in Java instead.

We researched numerous types of Java graphics, including JFrame, JPanel, JLabel, JButton and JComboBox, that we could use to create our GUI. We used JLabels to add the title and a description of the application on the homepage. We used JButtons to take the user to a different page, and to execute any query the user wanted. To set up the queries, we used JComboBoxes, which allow users to select from multiple choices, be it the date, the state, or the county.

## Challenges Faced

MySQL rejected data from the Johns Hopkins University spreadsheets for incorrect formatting. To fix this issue, we had to open up each CSV file downloaded from JHU in Microsoft Excel, and manually set the formatting of each column that we needed.

At first, we were unsure of how to establish a connection from eclipse to MySQL with a JDBC driver. We did an internet search to find the Java code needed to establish this connection.

We were at first unfamiliar with the different Java GUI components, but we quickly learned about the ones we needed online at Oracle Java Documentation.

It took at least 1600 lines of code to implement the Java GUI in our program. To tackle this beast of a workload, we worked late hours in the night typing out this code.

The toughest task in creating the program was, surprisingly, figuring out how to put a JScrollPane into the JTextArea where the results of the queries would be displayed. Very extensive internet research was required to find instructions for this.

When we tested the program, we would occasionally see errors, such as Java exceptions thrown. We copied and pasted the error messages from the console into Google, and followed the examples of other users who encountered the same errors.

Finally, we would have used current coronavirus statistics, but it was not possible to download live data from any website. Furthermore, there would be no way to make MySQL automatically accept new data whenever it comes in. We decided to make an “archive”, with past COVID records that we could download.

## Conclusion and Future Work

In this project, our goal was to allow users to be able to interact and to view data from different counties and states.

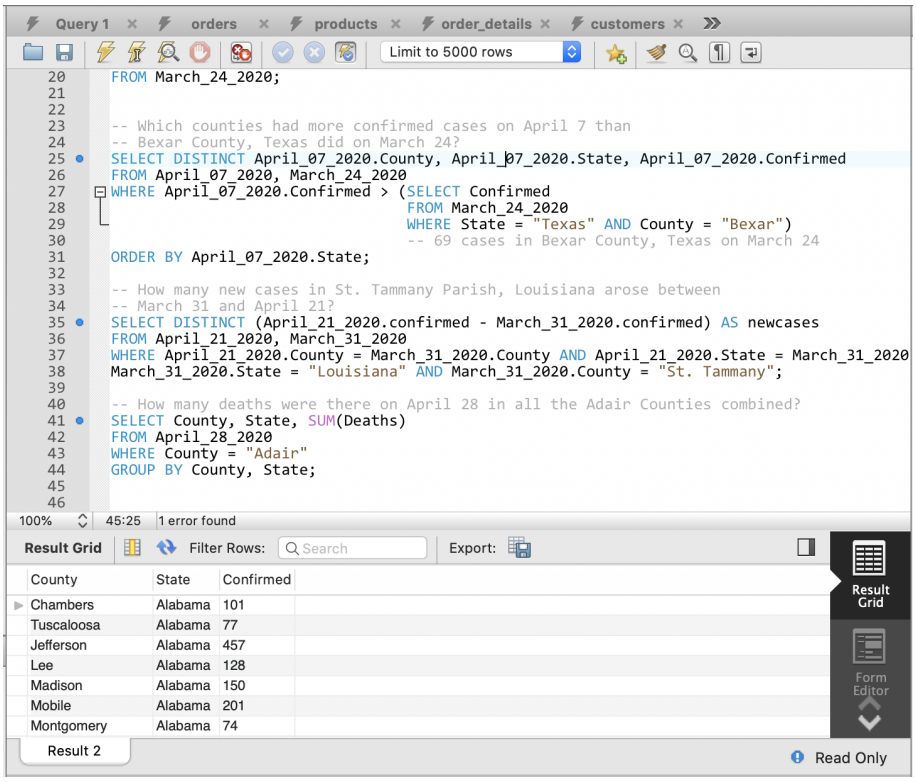
One of the countywide queries, which returns all counties that had more or less confirmed cases / deaths / recoveries / active cases than another county, takes a while to load when a lot of results are returned. Perhaps our algorithms could have been a little more efficient in this regard.

In the future, there is an opportunity to add more statistical information to be queried. For example, users may be interested in the number of COVID-19 tests taken per day, the number of positive test results per day, and the number of deaths per day, for each county and state.

In the end, our project amounts to a very useful tool that interested users can use to get an idea of how the coronavirus affected each area of the United States in the pandemic’s early days.

## Appendices

Testing SQL queries in MySQL first, before entering them into Eclipse




The SQL query, written in Java, is circled in red

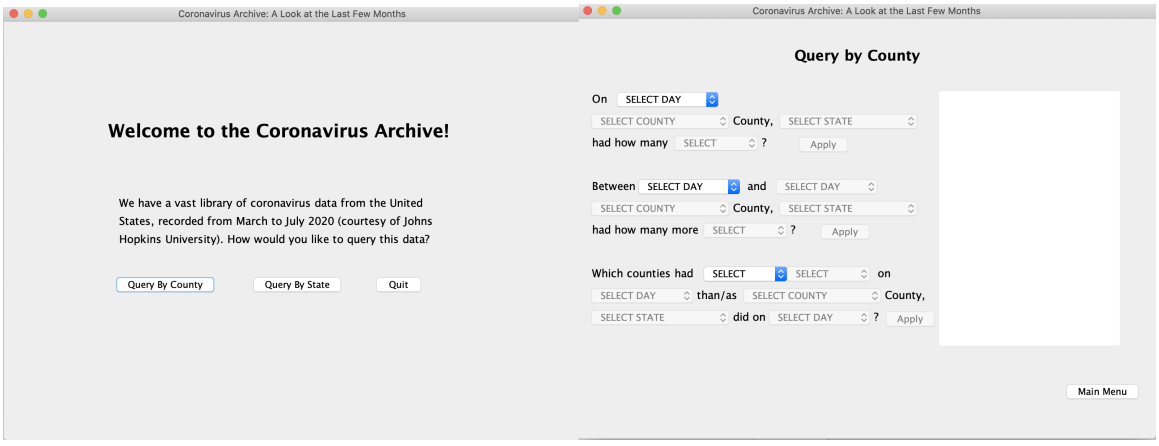
```
715         /** 6th **/
716         apply2.addActionListener(new ActionListener() {
717             public void actionPerformed(ActionEvent e)
718             {
719                 // How many new cases/deaths/recoveries/actives came up in this county between the two selected days?
720                 try
721                 {
722                     final String selectedDateA = (String) date2a.getSelectedItem();
723                     final String selectedDateB = (String) date2b.getSelectedItem();
724                     String sqlDateA = sqlDateFormat(selectedDateA);
725                     String sqlDateB = sqlDateFormat(selectedDateB);
726 
727                     final String selectedState = (String) states2.getSelectedItem();
728                     final String selectedCounty = (String) counties2.getSelectedItem();
729                     final String selectedCDRA = (String) cdra2.getSelectedItem();
730 
731                     result.setText(null); // clears the result pane
732                     result.setColumns(1); // puts 1 column in the result pane
733 
734                     ResultSet cdraResult;
735                     // I warn you, this is going to get dizzying...
736                     cdraResult = stmt.executeQuery("SELECT DISTINCT (" + sqlDateB + "." + selectedCDRA + " - " + sqlDateA + "." + selectedCDRA + ") AS " + selectedCDRA
737                             + " FROM " + sqlDateB + ", " + sqlDateA
738                             + " WHERE " + sqlDateB + ".County = " + sqlDateA + ".County AND " + sqlDateB + ".State = " + sqlDateA + ".State AND "
739                             + sqlDateA + ".State = '" + selectedState + "' AND " + sqlDateA + ".County = '" + selectedCounty + "';");
740                     while (cdraResult.next())
741                     {
742                         // result is the name of the JTextArea to which the query answer will be outputted
743                         result.append( Integer.toString(cdraResult.getInt(selectedCDRA)) );
744                     }
745 
746                 }
747                 catch (SQLException e1)
748                 {
749                     // TODO Auto-generated catch block
750                     e1.printStackTrace();
751                 }
752             }
753         });
754
```

Code to create the GUI graphics needed to display a sample query.

Corona.java

```
168                 }
169             }
170         });
171
172         // Now, we need to provide 3 types of queries...
173         /** For our first query, we need 4 drop-down menus and an APPLY button! **/
174
175         /** 1st **/
176         // date combo-box
177         JComboBox<String> date1 = new JComboBox<String>();
178         date1.addItem("SELECT DAY");
179         date1.addItem("March 24, 2020");
180         date1.addItem("March 31, 2020");
181         date1.addItem("April 7, 2020");
182         date1.addItem("April 14, 2020");
183         date1.addItem("April 21, 2020");
184         date1.addItem("April 28, 2020");
185         date1.addItem("May 5, 2020");
186         date1.addItem("May 12, 2020");
187         date1.addItem("May 19, 2020");
188         date1.addItem("May 26, 2020");
189         date1.addItem("June 2, 2020");
190         date1.addItem("June 9, 2020");
191         date1.addItem("June 16, 2020");
192         date1.addItem("June 23, 2020");
193         date1.addItem("June 30, 2020");
194         date1.addItem("July 7, 2020");
195         date1.addItem("July 14, 2020");
196         date1.addItem("July 21, 2020");
197         date1.setEnabled(true); // this combo-box is to ALWAYS be enabled! :)
198
199         /** 2nd **/
200         // state combo-box
201         JComboBox<String> states1 = new JComboBox<String>();
202         states1.addItem("SELECT STATE");
203         // don't add anything until a date is selected
204         states1.setEnabled(false); // this should be disabled if no date is selected
205
206         /** 3rd **/
207         // county combo-box
208         JComboBox<String> counties1 = new JComboBox<String>();
209         counties1.addItem("SELECT COUNTY");
210         // don't add anything until a state is selected
211         counties1.setEnabled(false); // this should be disabled if no state is selected
212
213         /** 4th **/
214         // confirmed (cases), deaths, recoveries, or active?
215         JComboBox<String> cdra1 = new JComboBox<String>();
216         // add items to the combo box
217         cdra1.addItem("SELECT");
218         cdra1.addItem("confirmed");
219         cdra1.addItem("deaths");
220         cdra1.addItem("recovered");
221         cdra1.addItem("active");
222         cdra1.setEnabled(false); // this should be disabled if no county is selected
223
224         /** 5th **/
225         // first apply button (for determining the number of cases)
226         JButton apply1 = new JButton("Apply");
227         Dimension a1Size = apply1.getPreferredSize();
228         apply1.setBounds(300, 160, a1Size.width, a1Size.height);
229         apply1.setEnabled(false); // this should be disabled if no selection is made in CDRA
230
231         /** Finally, a bonus... we need to provide a JTextArea to
232             output the results of ANY query! **/
233         JTextArea result = new JTextArea();
234         JScrollPane scrollable = new JScrollPane(result);
235         // this will change (in terms of number of columns) based on the query performed
236
```

## What the GUI (Graphical User Interface) looks like




# References

**Oracle. (2020, June 24). Retrieved August 02, 2020, from https://docs.oracle.com/javase/7/docs/api/javax/swing/JComboBox.html**

**Minh, N. H. (2019, July 6). JComboBox basic tutorial and examples. Retrieved www.codejava.net/java-se/swing/jcombobox-basic-tutorial-and-examples**

**Johns Hopkins University. (2020, July 24). JHU CSSE COVID-19 Dataset. Retrieved August 08, 2020, from https://github.com/CSSEGISandData/COVID-19/tree/b3262a11b6915e4ed2cd7b9563324fcb8a0d6a85/csse_covid_19_data**