API and Web App for Bank Failure Database

Drake UNIVERSITY

Jaehyoek Choi, Jacob Danner, Dr. Eric Manley*, Dr. Sean Severe*

ABSTRACT

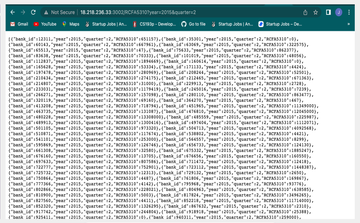
The Economics Data Science Research Group's goal is to use machine learning techniques to learn from previous data, so future bank failures can be predicted. But in order for this goal to be achieved, it's crucial for the data to be clean, organized and easily accessible.

Our data is publically available call reports, provided by every bank on a quarterly basis. The data is made available in bulk format by the FFIEC (Federal Financial Institutions Examination Council). The bulk format is extremely difficult to work with. This project includes the work organizing the data into an easily queryable relational database. In the database, there are over 3,000 tables, each containing data related to different banks at different times. In total it is over 50 GB of data. It is available online to economics researchers via a web API as well as a web based graphical interface.

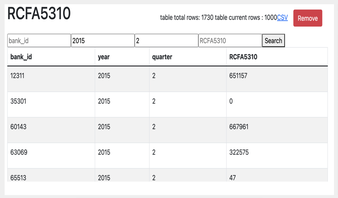
Our tools make it possible to easily isolate and merge variables across wide time series, enabling data analysis, predictive modeling, and visualization. A search tool allows researchers to find variables based on either a description or Federal Reserve series and item number. It allows easy programmatic access where complex SQL queries are replaced with simple URLs with search parameters. The web-app provides a GUI where users can query data using a simple input box and button and download the desired data in CSV format.

The API layer is written in Python's fastAPI framework. The UI layer uses Javascript's React library. Hosting was done with AWS services, such as EC2, RDS, and S3.

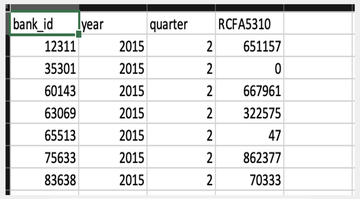
BULK FORMAT



WITH API



WITH WEB APP



bank_id	year	quarter	RCFA5310
12311	2015	2	651157
35301	2015	2	0
60143	2015	2	667961
63069	2015	2	322575
65513	2015	2	47
75633	2015	2	862377
83638	2015	2	70333

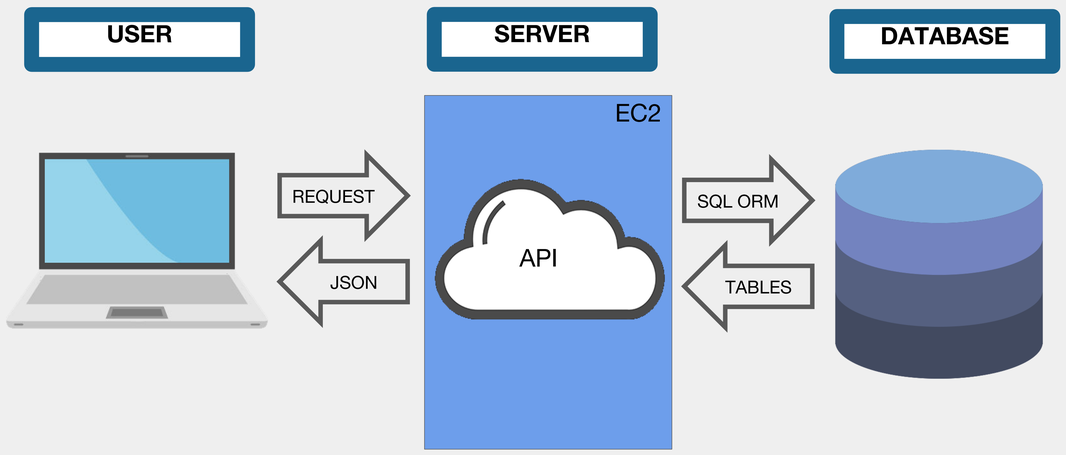
DOWNLOADABLE CSV

FUTURE

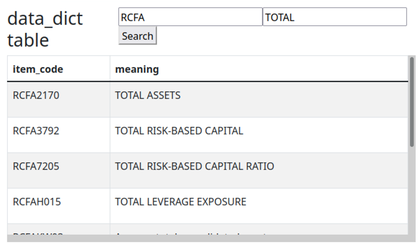
- Migrate the API into AWS Lambda instead of EC2 to save server cost
- Implement table joins in API layer and web UI layer
- Visualizations for different variables
- Frontend and Backend optimization for performance

ACKNOWLEDGEMNETS

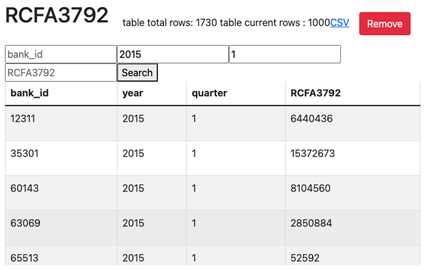
Dr. Eric Manley & Dr. Sean Severe - research supervisors, and organized AWS RDS database instance



UI LAYER



- When first entered, data_dict table will be displayed to show the item_code and its meaning. The data_dict table is functioning as the appendix for all the variables in the database
- In this case user searched for item_code that includes "RCFA" and meaning containing "Total"

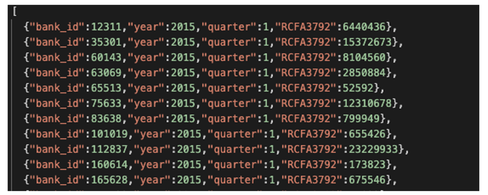


- Once the user double clicks the item_code from data_dict table, the item_code table, a variable from the database, is displayed.
- For this specific example RCFA3792 table has been used.
- The user can search each variables within the variables using inputs and search button
- In this case, the table displays data that have the value year of 2015 and quarter of 1.
- User can click CSV link to download the table in CSV format
- Once the user is done, user can click remove button to remove the display of the table

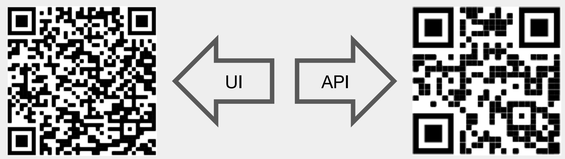
API LAYER

To search for a specific table, for example - RCFA3792, replace the data_dict in url with RCFA3792 and add search parameters such as http://18.218.236.33:3002/RCFA3792?year=2015&quarter=1

This query returns the following:

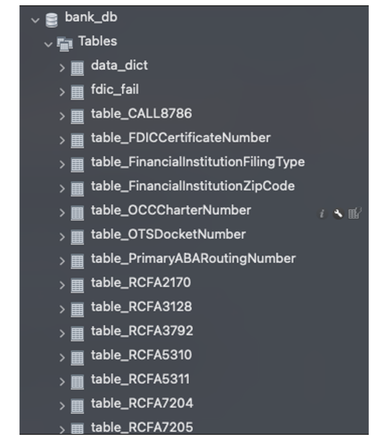


For the machine learning team, if they retrieve values using HTTP requests in their code and simply store JSON data as variables, they can easily create a dataFrame and the data would be in great form to train the models.



DB LAYER

Our database team, organized the bulk txt form of data into a relational mysql database. The database is currently hosted in AWS RDS service, and it is the backbone of all our work. Each variable is represented as a table, and each table has a value that changes over time - each quarter of a year.



For this specific project, the API is built to simplify the query process for the database. The web GUI is built using the API to make the process even easier.

DEPLOYMENT

Without the deployment, the API would not be accessible to the public. Our API server is running with an Amazon Web Services EC2 instance. Within the instance, there is a docker container running that has our API code.

The API code contains a Dockerfile, which is the basis for building the docker image. Once the docker image has been created using the Dockerfile, the docker image is run inside of the AWS EC2 instance. At this point, it's ready to take web requests. Once a request has been made, the server sends SQL queries to the AWS RDS instance holding our database.