My project is called the Academic Keywords Analytics Dashboard. It allows the user to explore keywords in depth, such as by finding relevant professors and publications for their input keyword. This should mostly be helpful for anyone who wants to do research on a specific topic, since it can help them find publications to help with research or professors who they could contact that are relevant to their specific keywords.

As my project is mostly a dash dashboard, you simply need to install the prerequisite packages (dash, dash_bootstrap_components, plotly, pandas, networkxx) and run the app.py file on your local terminal. I installed each by using (pip install) on each. To connect to the databases, enter the fields according to the config parameters stated at the start of the code. Also, it should be noted that I included a file that contains how I made my views and triggers in the GIT repository.

The dashboard has 6 widgets: one each to explore keywords for professors, publications, and universities. The professor one lets you input a keyword, then it will return the top N (you can choose how many you want to see) professors whose keyword scores are the highest for the input keyword. Clicking on the name of the professor will pull up a card that has their photo, position @ university, email, phone, research interest, more keywords, and their 5 most recent publications. The publication one shows a scatterplot of all the publications written that have the input keyword, with the year as the x axis and their corresponding score as the y axis. Clicking on a point will show you the venue, authors, and number of citations for the publication. The university widget is a little bit different in that instead of inputting a keyword, you instead input the university name, and it returns the university's logo and a pie chart showing the top N (I limited it to 20) keywords based on the total scores from professors at that university. All of these widgets run by performing SQL queries on views created to simplify the queries by pre performing the various joins necessary for these queries.

Then there is one widget that allows you to find keywords that are similar to your input keyword, and you can then find professors and publications with your original keyword and any amount of the similar keywords. The graph will show you how connected these keywords are, whether there's only 2 or 192 professors and/or publications that share the two keywords. For example, if you input "internet", you can also find professors and/or publications with the keywords "internet", "network", and "email". Clicking on a professor's name will pull up a similar card to the one for the above professor widget. This works by running a Neo4j query to find the keyword relationships, then the results are then passed into an SQL query to show the keyword matches (as per the extra credit capability).

The last two widgets allow you to make changes to the database. The first one lets you edit a professor or publication's information, while the second one lets you edit keyword scores. You are allowed to edit anything related to the input professor or publication, and those changes will be reflected in the professor cards, as these edits and the professor cards are both connected to the MongoDB database. However, changes made to a professor's keywords will only change the keywords shown on their card, and will not affect searches within the widgets. This also applies to publications. The info displayed when clicking on a scatter plot point will be changed, but the info used to query the publication will not be changed. The keyword editor is instead connected to the SQL database, so any changes made here will be reflected when doing keyword searches. These are all connected to triggers that occur when a change is made to ensure data consistency.

I mostly used dash-bootstrap-components to implement the dashboard layout, though I used networkxx to create the keyword connections graph. I also used plotly to create my scatterplot for my publications widget and my pie chart for the university widget.

The three database techniques I chose to implement are views, triggers, and prepared statements. As mentioned before, I used views to simplify some of my SQL queries because they involved a large amount of joins. The queries perform noticeably faster with views compared to running the queries without them. While the original data had the keywords attached to the professors and publications, that data was separated when uploading the data to SQL, so the views simply rejoin the data to make for faster lookups. I also use triggers to help monitor the keyword updates to ensure consistency whenever a score is updated. Lastly, I use prepared statements in my queries so that the queries can receive parameters whenever I repeatedly call them for the widgets. For example, using prepared statements lets me run a query to find the professors with the highest keyword scores for biology or physics without having to write separate queries for each.

As also previously mentioned, I use multi database querying for my keyword similarity tab. Neo4j is best for finding relationships between different tuples, so I first used neo4j to find keywords that have some connection between each other. Then I use a regular SQL query to find the top professors and publications with the shared keywords. Originally I had these as separate widgets, but I realized that the second widget would be pointless if keyword “aaa” had no matches with keyword “zzz”, and the first widget guarantees that there’s a match between the queried keywords.

Lastly, as I worked on this project myself, I did 100% of the work on this project.