# LinkedIn Database Solutions

Jesse Aseoff

Chapman University

Fowler School of Engineering

## I. Problem to Address

As a soon-to-be college graduate, I continue to plan for the beginning of my professional career. I am studying Data Science and am hoping to get a full-time position where I can put my technical and behavioral skills to the test. Fortunately, there are plenty of opportunities out there. However, navigating the best opportunities for me (and I know many others) can be challenging, time consuming, and frustrating.

The reality is that we, Gen Z, are living in the Age of Information, where we have access to anything and anyone in a matter of seconds. This reality has plenty of benefits; however, it also has its drawbacks. One major disadvantage to having all this information at our fingertips is that it can sometimes make it more difficult to find what we are looking for.

When it comes to searching for jobs and career opportunities, there is no shortage of places to begin. Websites like LinkedIn, Indeed, Glass Door, Ripple Match and others all offer services (mostly free) to help find job openings that match the type of job you are looking for. They also allow you to apply filters that can help narrow down your search. These filters include location, job level, industry, etc.

In theory, these sites provide everything and more for a person looking for a new job. However, in practice, due to the magnitude of jobs and information even with a variety of filters applied, job seekers can end up spending hours navigating through thousands and thousands of jobs – many that don't even match the criteria they are looking for.

To address this large-scale issue, I will first analyze it on a micro scale. I will do this by only diving into the LinkedIn jobs platform (rather than all the popular job board sites). When looking at LinkedIn jobs specifically, there are a few problems that stand out:

1. **User can (effectively) only enter in one job title at a time**

   For example, if I search “Data Science”, I will see plenty of Data Science positions (and probably some unrelated roles too), but I will not see a lot of “Data Analyst” positions. This is a fundamental problem because more than likely, people are looking for not just one job title. Chances are there are multiple roles that can fit one’s skillset.

2. **User can (effectively) only enter in one location at a time**

   There is an option to keep the location filter blank (meaning showing jobs everywhere); however, it is beneficial to filter the location because there are already too many jobs for a given search. For this reason, adding more and more locations only makes the search result more cluttered and time consuming.

3. **Too much information…**

   In a given search for “Data Science” jobs in “Colorado”, LinkedIn returns 3,000+ jobs. For each job, there is a multi-paragraph description along with other pieces of information about the specific role. For a person to browse through all the thousands of jobs and read each description would take days… It is simply not practical.

For all the above reasons, I decided to use my knowledge from CPSC408 (Database Management) to create a Database Management System (DMBS) that simplifies and tracks one’s job search process. The DBMS will incorporate the user’s personal LinkedIn connections to allow for a custom, faster, and less frustrating professional experience.

## II. Related Applications

As far as I know, the closest thing to what I've created is LinkedIn jobs itself. However, like I said above, there are some fundamental flaws with the platform that inspired me to build this improved system. LinkedIn jobs is a powerful tool that people searching for jobs can surely utilize as a resource. However, the overload of (oftentimes irrelevant) information makes the site less attractive to those looking for a faster alternative to navigating the many jobs being posted on LinkedIn every day.

## III. Elements of Solution

My program, "LinkedIn Database Solutions" implements a few powerful tools that streamlines and enhances the LinkedIn jobs experience. Below I will breakdown how I created the application and how it works.

### 1. Web scrape LinkedIn Jobs

Like I keep saying, there is way too much information for any one person to scroll through on the LinkedIn jobs page. For this reason, to create a custom jobs Database based on one's professional interests, it wouldn't be realistic to manually go through LinkedIn Jobs and copy/paste job information into a new database. I needed a (much) faster alternative.

For this reason, I learned a new technical skill – web scraping using Python's selenium library. This allowed me to automatically obtain search results for many jobs with different filters. In short here is how my jobs scraping program worked:

1. Prompt user to enter LinkedIn jobs search results page URL(s) (user can enter as many or as few URLs as they want)
2. For every URL inputted by the user, the program will scrape the first x number of jobs (x can be changed by the user) on each page. The scraped job information will include all relevant details about the job including title, company, location, level, industry, description, link, etc.
3. The scraped job information will be appended to a custom csv file containing jobs from the links inputted by the user (the csv could contain thousands of relevant jobs with each's corresponding information)

**2. Add job information csv to "jobs" table**

Now that the job scraping program has successfully scraped x number of jobs for the user based on the links provided, it is time to move this information into the database. The jobs in the csv file will be inputted into a table called "jobs" which contains the same columns as those in the csv (all relevant information about each job)

**3. Create "companies" table using companies from "jobs" table**

Since it can be repetitive to have multiple jobs from the same company, I decided it would be better to create a "companies" table within the database. I created the companies table initially using the companies from all the jobs that were scraped. After creating the companies table which contains columns "company_id", "company_name",

and “company_description”, I converted all the companies in the jobs table to their corresponding company id (FK in the jobs table).

4. **Add “connections” to the Database**

   I didn’t know this before beginning this project, but LinkedIn allows all its users to download their connections information into a csv file. The csv contains information like first name, last name, title, company, and date connected. I figured that using this data could be a good way to make my program more customizable. I can now add a filter in the job search using connections (search for a job based on where connections work). I used my downloaded connections csv to create the “connections” table in the database. Each attribute in this table had a connection_id (PK) and company_id (FK).

5. **Create “saved jobs” table**

   LinkedIn allows its users to save jobs; however, it can sometimes be hard to keep track of these saved jobs. I know sometimes I scroll through my saved jobs on LinkedIn and wonder why I saved a particular job. For this reason, I thought it was important to allow the user the ability to enter in notes for a job they want to save and keep track of. I created a new table in the DB called “saved_jobs” which contains PK “saved_job_id”, FK “job_id”, and another column “notes”. Now it will be possible for a user to save a job they may be interested in and add notes to remember why they saved it (Ex: “Remember to apply” or “reach out to recruiter”).

6. **Create the interface**

Now for the fun part, the UI. I decided to use streamlit to bring the DBMS to life. The application is simple and effective. The main features include a sidebar with three tabs: my jobs, my connections, and my saved jobs. Within each section, the user can view results based on filters that will impact the SQL query used to extract results. For example, a user can enter in location, job title, and company and be outputted information that contains all those parameters. One of the most challenging and rewarding parts of the overall program was coding the "query_creator" method in Python. This function essentially uses all the input elements (and disregards those that are left blank) to create a custom query for results. In other words, I did not hard code specific queries for specific search results, rather, the program uses the given information to create the appropriate query.

Once input has been given, jobs or connections are printed in the main section of the page. The jobs or connections are listed as expanders containing the most relevant information on the outside (Job title, location, etc. or connection first and last name) and the more detailed information within (once user clicks to expand). This removes clutter on the page and allows the user to quickly access more information. The user then can decide which jobs or connections they want to learn more about (by opening the expander)

Within each job expander is an option to add a note and save the job. Once a note is added and the "Save job" button is pressed, the job gets put into the saved jobs table and can be viewed in the saved jobs section of the streamlit page. Now the user is not

only able to view relevant jobs, but they can also keep track of jobs that interest them and add corresponding notes.

There are certainly more features I hope to add in the future, which I will discuss in the discussion section.

## IV. Results

Overall, the application works well and does what I intended for it to do. A user can successfully input job or connection parameters, a SQL query is generated from the Python program, and the appropriate results are printed to the user. Moreover, within the jobs section, a user has the capability to add notes and save specific jobs (creating a new saved job). They are then able to view their saved jobs within the saved jobs section of the application and modify (change note) or delete a saved job entirely.

Throughout this project, I have certainly grown as a programmer, problem solver, and out-of-the-box thinker. I took on a large-scale problem that required new skills and lots of critical thinking. One of the toughest aspects of this project was building the jobs scraper. And although this was not a requirement for the project, it certainly enhanced what I built (allowing for me to use real data) and widened my skillset. I can say the same for streamlit – a program I had never heard of and now I have built a fully functioning application with it. For all the above reasons, I am proud of what I produced. However, I do want to eventually deploy this idea and know with more time, I can make it even better and more user friendly.

## V. Discussion and Next Steps

In the future, I hope to automate the entire process of scraping jobs and loading them into a MySQL database. Rather than scraping jobs, creating a CSV, and then using the CSV to create tables, it would be better to go straight from scraping to MySQL. This is a doable task that I will make sure to implement.

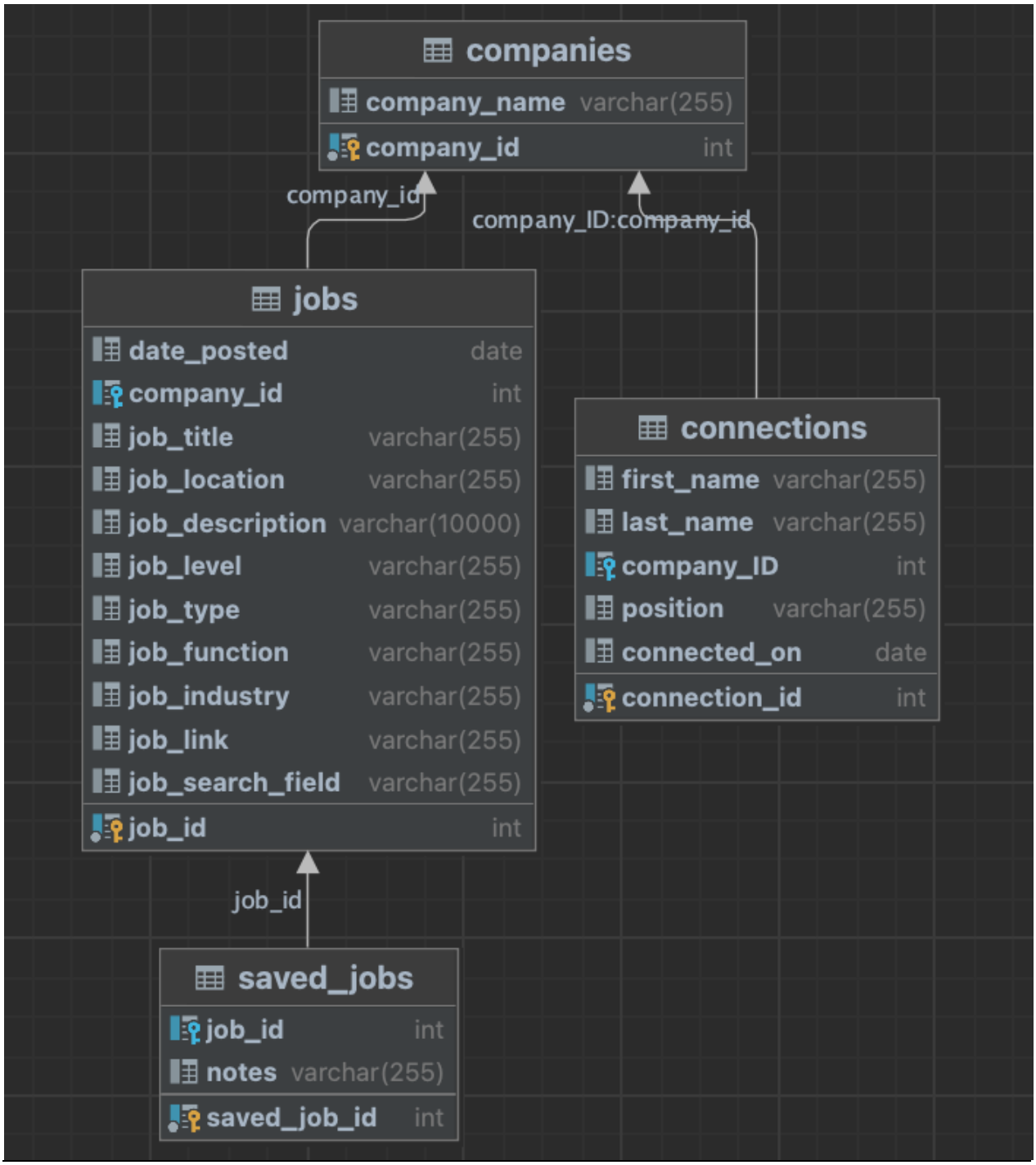
Another challenge I encountered was streamlit's auto refresh tendencies. Basically, streamlit automatically reruns on the backend every time a user interacts with an element (text input, button, etc.) on the front end. This made it difficult for my application to perform the way I intended for many reasons. It was particularly frustrating when it came to the saved jobs feature. Due to the streamlit nature, for a while, every time I tried to save a job within the job results, the page would refresh and the "save job" button click would not trigger the corresponding MySQL query required to create a new record and commit the changes to the database.

I found some workarounds to the constant refreshing and was able to get the jobs to save. That being said, I could not resolve all my related issues, and there are still some quirks within the program that I would like to eventually fix. **Note,** if you notice a button click seeming to not execute properly, follow these steps: 1) press it again (this will probably get the proper results to show) and if not 2) manually check to see if prompted changes were made to the DB in an application like Data Grip.

Lastly, I hope to turn my concept into a fully functioning organizational tools that students and professionals can utilize when searching for a job. This of course requires time (and possibly money); however, this project has provided me with a great framework to eventually turn this idea into a reality!

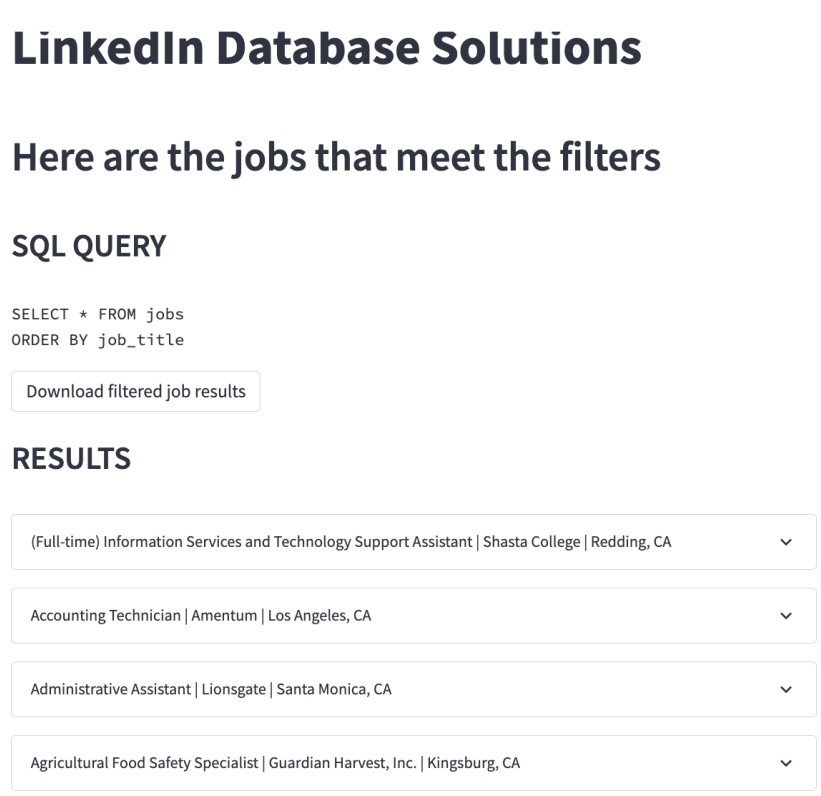
**Appendix**

Schema

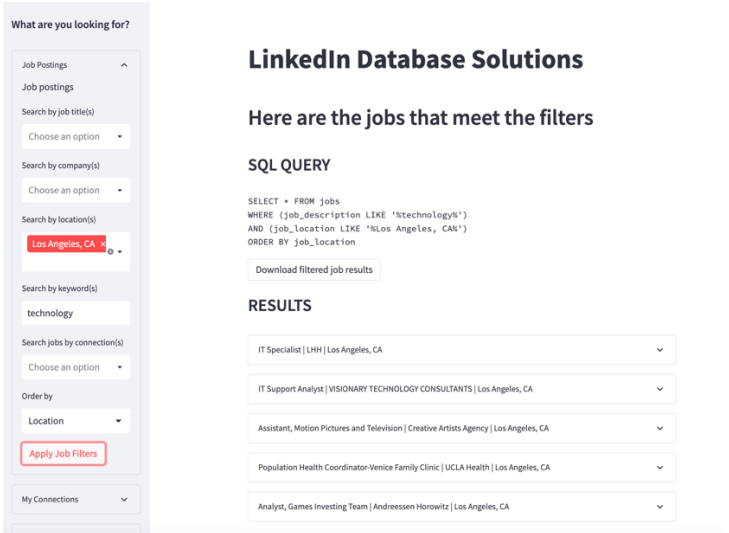


Project Requirements

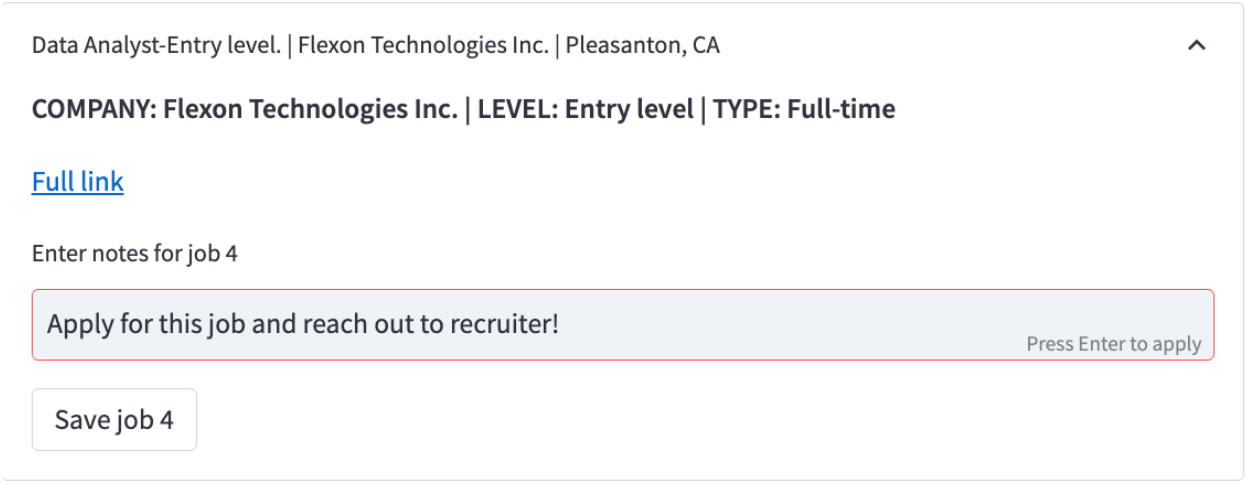
1. **Print/display records from your database/tables.**




2. **Query for data/results with various parameters/filters**

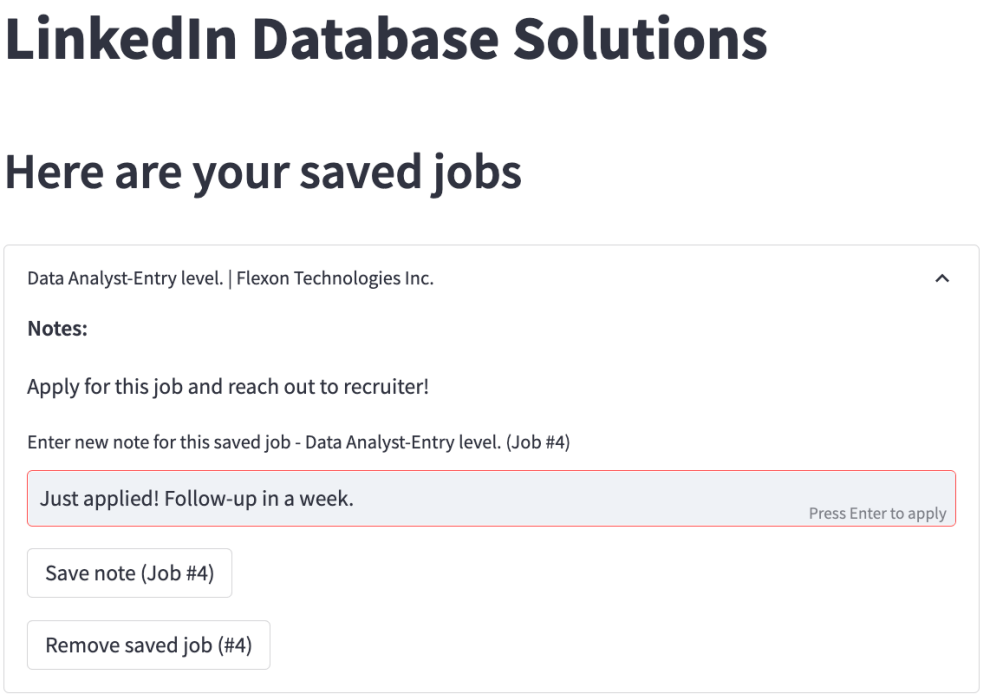


### 3. Create a new record




Pressing the "Save job x" button executes MySQL query to create new saved_job record containing the notes inputted by the user.

### 4. Delete/update records

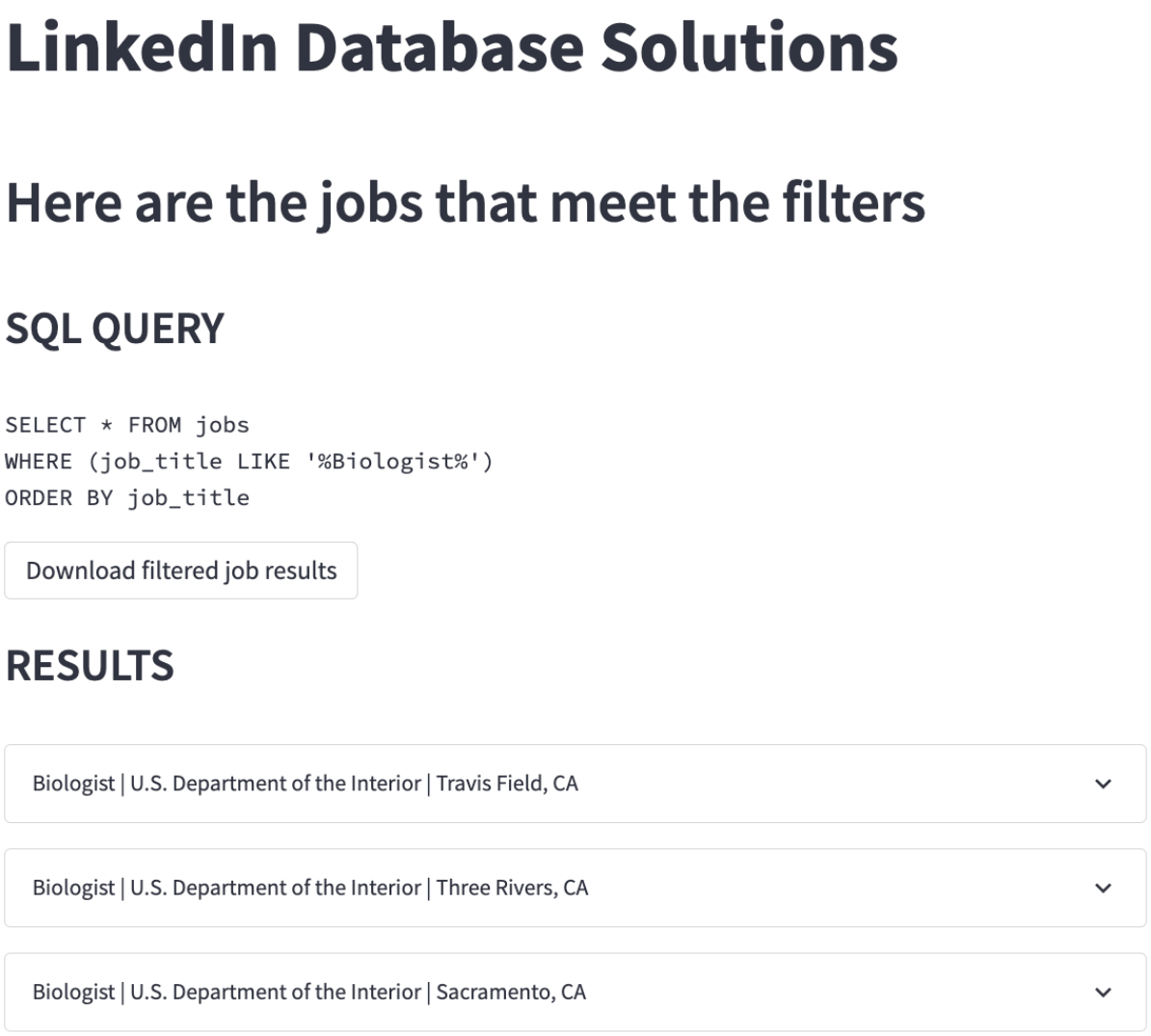



Within the saved jobs tab, users have the option to "update" their saved job by adding a new note or remove ("delete") the saved job if it is no longer relevant to them.

5. **Make use of transactions - commit**

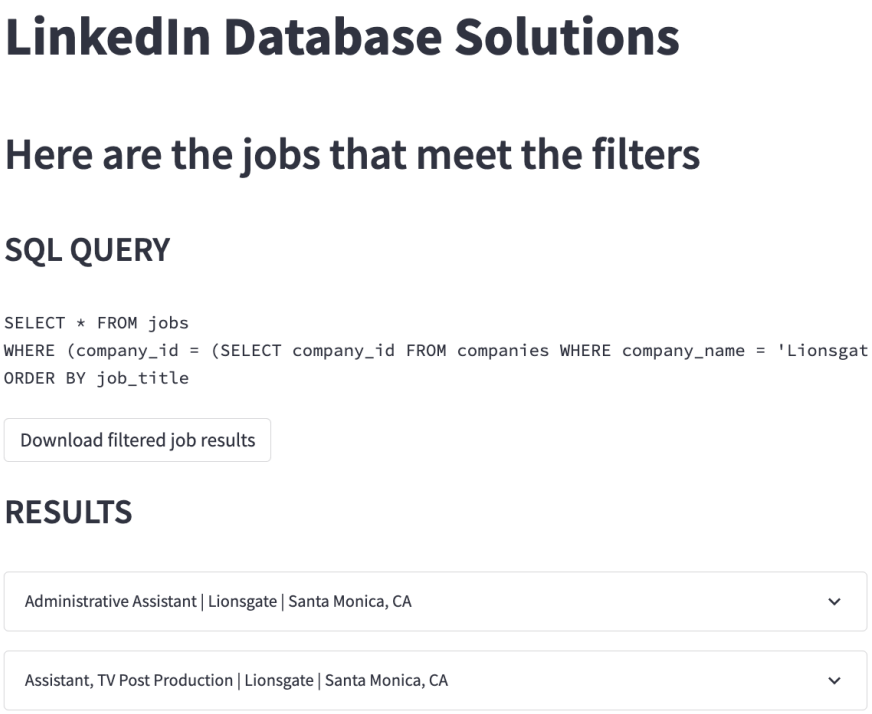
Whenever a button to create, modify, or remove a saved job is pressed, a changed is committed to the database. I could (and plan to) improve on this requirement here by utilizing rollback features.

6. **One query must perform an aggregation/group-by clause**



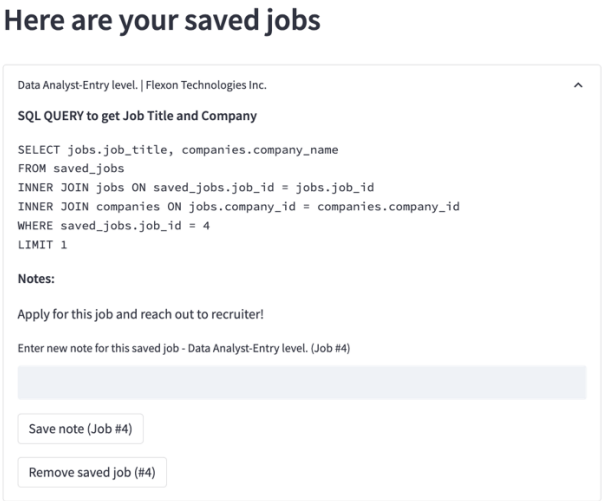

For the jobs and connections search features, the user has the option to sort information by a few different parameters. The query creator function within my Python program then creates and executes the appropriate aggregation.

7. **One query must contain a sub-query**




To pull company information for jobs/connections results, it is necessary to access the companies table within the database. Rather than using a join, I instead implemented a sub-query (dependent on user input) to extract company_id from company name.

8. **Two queries must involve joins across at least 3 tables - do it for saved jobs**




In order to extract a saved job's title and company, it is necessary to access the jobs table and the companies table. Therefore, two joins are required to get all of the information. For every expander, the position title (within jobs table) and company name (within companies table) are displayed. Therefore, the program will execute a different query (containing 2 joins) for each saved job. Above is an example of a query with 2 joins used to extract information about the displayed saved job.

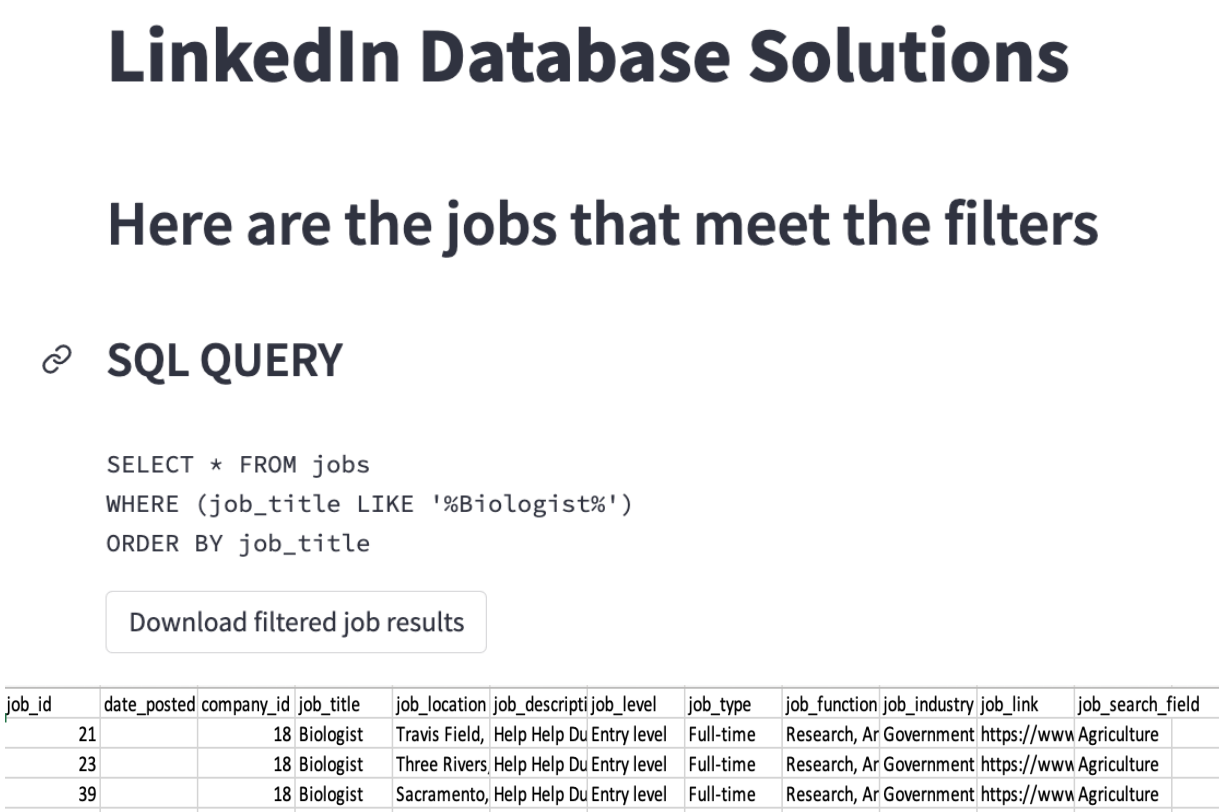
9. **Enforce referential integrality (PK/FK Constraints)**

Referential integrity is enforced throughout the database. In the future, I plan to add additionally constraints/triggers that ensure the user cannot compromise the database's logic.

10. **Use at least 5 entities (tables)**

I have four; however, since I'm working solo, I believe this is okay. In the future, I plan to add more (one in mind is past saved jobs).

11. **Extra credit: Generate reports that can be exported (excel or csv format)**




| job_id | date_posted | company_id | job_title | job_location | job_descripti | job_level | job_type | job_function | job_industry | job_link | job_search_field |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 21 | | 18 | Biologist | Travis Field, | Help Help Du | Entry level | Full-time | Research, Ar | Government | https://www | Agriculture |
| 23 | | 18 | Biologist | Three Rivers, | Help Help Du | Entry level | Full-time | Research, Ar | Government | https://www | Agriculture |
| 39 | | 18 | Biologist | Sacramento, | Help Help Du | Entry level | Full-time | Research, Ar | Government | https://www | Agriculture |

The "Download filtered job results" allows the user to download (into a csv) all the jobs or connections results that were outputted based on the filters they applied. I plan to eventually implement this feature within the saved jobs section as well.