
JOB AGGREGATOR LITE

PROJECT 8

Jack Kammerer

CS.4332.252 Introduction to Database Systems

Texas State University

April 2025

Contents

1	Introduction	1
2	Updated E/R Diagram	2
3	English Queries and SQL Implementations	3
3.1	Query 1: Tech Job Overview	4
3.2	Query 2: Web Development Focus	4
3.3	Query 3: Software Engineering Roles	5
3.4	Query 4: IT Position Search (Adjusted)	6
3.5	Query 5: Front-End vs. Back-End Focus	7
3.6	Query 6: Full-Stack Opportunities	8
3.7	Query 7: Tech Skills Filter	9
3.8	Query 8: Seniority Filter	10
3.9	Query 9: Remote Work Focus (Adjusted)	11
3.10	Query 10: Category Summary Count	12
4	Backend Accomplishments	13
5	Source Code	14
5.1	Main Python Files	14
5.2	main.py	14
5.3	query_db.py	17
5.4	scraper.py	26
5.5	database.py	30
5.6	config.py	39
6	Future Implementations	40
7	Conclusion	40

1 Introduction

At this milestone, I successfully finalized the backend functionality of my LinkedIn Job Scraper project. This phase required major pivots, refactoring, and plenty of trial and error to achieve a stable, fully functional scraper backed by a normalized database. The project originally began with a minimal design: using SQLite and just two basic entities. But to meet the expanded project scope and ensure proper database normalization, I scaled the system into a five-entity architecture, now running on MySQL and fully integrated with **phpMyAdmin** for easier data management and visibility. Early on, I explored using AI-powered parsing tools like **Ollama** and **CodeLlama** hosted locally, in an attempt to automate SQL generation and parsing tasks. However, due to their limited model sizes and inefficiencies with complex queries, I pivoted to building out custom, hard-coded parsing logic. This allowed me to reliably extract and structure job data aligned with the roles and industries I personally care about.

As of Project 8, my system achieves the following:

- Fully operational LinkedIn scraper with a hardcoded URL targeting relevant job listings
- Clean and efficient parsing logic for job titles, company names, locations, skills, and post dates
- Expansion from 2 to 5 entities, achieving a normalized database schema (3NF)
- Data storage in MySQL with full visibility and control via phpMyAdmin
- Real-time data scraping and insertion, verified through terminal outputs and phpMyAdmin dashboards

```

RENDERER   PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   SQL HISTORY   TASK MONITOR
config.py > ...   config.py > ...   database.py   main.py   data_parser.py   scraper.py   query_db.py
host: 'localhost'
jackkammerer@Jack-2 webscraper % source .venv/bin/activate
(.venv) jackkammerer@Jack-2 webscraper % python main.py
2025-04-13 18:37:48,536 - main - INFO - Starting job scraping with URL: https://www.linkedin.com/jobs/search/?keywords=software%20engineer&location=United%20States
2025-04-13 18:37:48,536 - main - INFO - Starting to scrap job with URL: https://www.linkedin.com/jobs/view/intern-software-engineer-at-ticketmaster-4191513835?position=1&pageNum=0&refId=d71d42b8xcnrczBtD7zBY6vBaK3%03dcrackingId=UfH4cStVpcS5Ym9gPMgk3Dk3D
2025-04-13 18:37:48,687 - database - INFO - Connected to MySQL database
2025-04-13 18:37:48,687 - database - INFO - Database tables created successfully
2025-04-13 18:37:48,687 - database - INFO - Resetting database...
2025-04-13 18:37:48,687 - database - INFO - Dropped table job_skills
2025-04-13 18:37:48,687 - database - INFO - Dropped table locations
2025-04-13 18:37:48,687 - database - INFO - Dropped table companies
2025-04-13 18:37:48,687 - database - INFO - Database reset and tables recreated successfully
2025-04-13 18:37:48,687 - database - INFO - Database reset and tables recreated successfully
2025-04-13 18:37:48,687 - database - INFO - Database initialized and reset
2025-04-13 18:37:50,160 - scraper - INFO - WebDriver initialized successfully
2025-04-13 18:37:50,161 - __main__ - INFO - LinkedIn scraper initialized
2025-04-13 18:37:50,162 - scraper - INFO - Starting to scrape job listings from: https://www.linkedin.com/jobs/search/?keywords=software%20engineer&location=United%20States
2025-04-13 18:37:55,714 - scraper - INFO - Found 15 job listings
2025-04-13 18:37:55,714 - __main__ - INFO - Found 15 job listings, proceeding to scrape details
2025-04-13 18:38:00,944 - scraper - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/intern-software-engineer-at-ticketmaster-4191513835?position=1&pageNum=0&refId=d71d42b8xcnrczBtD7zBY6vBaK3%03dcrackingId=UfH4cStVpcS5Ym9gPMgk3Dk3D
2025-04-13 18:38:00,944 - data_parser - INFO - Processed job data for: Intern, Software Engineer
2025-04-13 18:38:00,944 - __main__ - INFO - Successfully processed and saved job: Intern, Software Engineer at ticketmaster
2025-04-13 18:38:01,837 - __main__ - INFO - Successfully processed and saved job: Intern, Software Engineer
2025-04-13 18:38:03,943 - scraper - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/software-engineer-ai-platform-new-grad-at-nuro-4191146990?position=5&pageNum=0&refId=d71d42b8xcnrczBtD7zBY6vBaK3%03dcrackingId=k00X1H6wK7NYxJltQ4An3Dk3D
2025-04-13 18:38:03,943 - data_parser - INFO - Processed job data for: Software Engineer, AI Platform - New Grad
2025-04-13 18:38:06,735 - __main__ - INFO - Successfully scraped job details for: Software Engineer, AI Platform - New Grad at Nuro
2025-04-13 18:38:06,763 - database - INFO - Successfully saved job: Software Engineer, AI Platform - New Grad at Nuro
2025-04-13 18:38:06,763 - __main__ - INFO - Successfully processed and saved job: Software Engineer, AI Platform - New Grad
2025-04-13 18:38:08,766 - scraper - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/software-engineer-augmented-reality-split-compute-at-google-4185939?position=0&pageNum=0&refId=d71d42b8xcnrczBtD7zBY6vBaK3%03dcrackingId=UfH4cStVpcS5Ym9gPMgk3Dk3D
2025-04-13 18:38:11,412 - scraper - INFO - Successfully scraped job details for: Software Engineer, Augmented Reality, Split Compute
2025-04-13 18:38:14,015 - data_parser - INFO - Processed job data for: Software Engineer, Augmented Reality, Split Compute at Google
2025-04-13 18:38:14,023 - database - INFO - Successfully saved job: Software Engineer, Augmented Reality, Split Compute at Google
2025-04-13 18:38:14,023 - __main__ - INFO - Successfully processed and saved job: Software Engineer, Augmented Reality, Split Compute
2025-04-13 18:38:16,928 - scraper - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/software-engineer-front-end-multiple-levels-slack-at-slack-4206626878?position=4&pageNum=0&refId=d71d42b8xcnrczBtD7zBY6vBaK3%03dcrackingId=f5M2rFAncByV72gs1mYOj3Dk3D
2025-04-13 18:38:19,928 - scraper - INFO - Successfully scraped job details for: Software Engineer FrontEnd (Multiple Levels) - Slack
2025-04-13 18:38:19,928 - data_parser - INFO - Processed job data for: Software Engineer FrontEnd (Multiple Levels) - Slack at Slack
2025-04-13 18:38:19,931 - __main__ - INFO - Successfully processed and saved job: Software Engineer FrontEnd (Multiple Levels) - Slack
2025-04-13 18:38:19,931 - __main__ - INFO - Successfully processed and saved job: Software Engineer FrontEnd (Multiple Levels) - Slack
2025-04-13 18:38:21,036 - scraper - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/software-engineer-new-grad-program-at-sigma-41922080?position=5&pageNum=0&refId=d71d42b8xcnrczBtD7zBY6vBaK3%03dcrackingId=bfMnVKXP1m4E4%2FzNzX7A%3Dk3D
2025-04-13 18:38:21,036 - data_parser - INFO - Processed job data for: Software Engineer (New Grad Program) at Sigma
2025-04-13 18:38:26,961 - database - INFO - Successfully saved job: Software Engineer (New Grad Program) at Sigma
2025-04-13 18:38:26,973 - __main__ - INFO - Successfully processed and saved job: Software Engineer (New Grad Program)
2025-04-13 18:38:28,978 - scraper - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/software-engineer-integrations-at-notion-4194463239?position=6&pageNum=0&refId=d71d42b8xcnrczBtD7zBY6vBaK3%03dcrackingId=70d04WpVpENz24ZqzX%3Dk3D
2025-04-13 18:38:33,616 - scraper - INFO - Successfully scraped job details for: Software Engineer, Integrations

```

Ln 13, Col 25 Spaces: 4 UTF-8 LF () Python ⌂ 3.9.6 ('venv') ⌂ Go Live ⌂

Figure 1: main.py executing(part1)

Table	Action	Rows	Type	Collation	Size	Overhead
companies	Browse Structure Search Insert Empty Drop	13	InnoDB	utf8mb4_0900_ai_ci	32.0 KiB	-
jobs	Browse Structure Search Insert Empty Drop	15	InnoDB	utf8mb4_0900_ai_ci	208.0 KiB	-
job_skills	Browse Structure Search Insert Empty Drop	65	InnoDB	utf8mb4_0900_ai_ci	32.0 KiB	-
locations	Browse Structure Search Insert Empty Drop	9	InnoDB	utf8mb4_0900_ai_ci	32.0 KiB	-
skills	Browse Structure Search Insert Empty Drop	28	InnoDB	utf8mb4_0900_ai_ci	32.0 KiB	-
5 tables	Sum	130	InnoDB	utf8mb4_0900_ai_ci	336.0 KiB	0 B

Figure 2: Data Scrapped Tables

2 Updated E/R Diagram

The E/R diagram evolved substantially from the earlier phase. Originally designed with just job_sites and job_postings, I expanded the model to five entities:

- **Jobs:** Job title, description, post date, etc.

- **Companies:** Company name, industry, size
- **Locations:** City, state, country
- **Skills:** Technologies or skills required
- **Job_Skills:** Many-to-many relationship table between jobs and skills

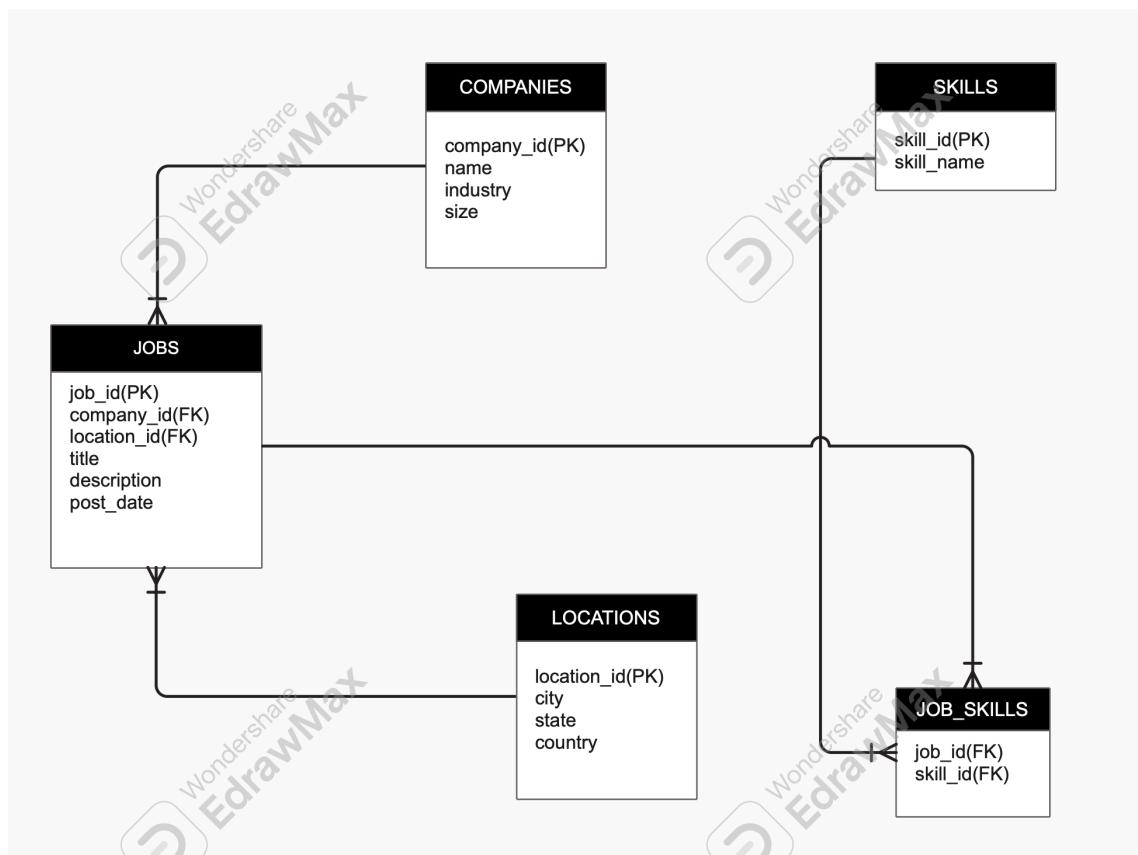


Figure 3: New E/R Diagram - 3NF

3 English Queries and SQL Implementations

Due to major changes in my project to fit the expansion standard (migrating from a 2-entity model to a 5-entity normalized database), my original English queries were revisited and adjusted to suit the final database design.

3.1 Query 1: Tech Job Overview

English:

List all tech job postings scraped from LinkedIn, showing the job title, company name, location (city, state, country), and the post date. **SQL:**

```
SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
JOIN locations l ON j.location_id = l.location_id
ORDER BY j.post_date DESC;
```

Screenshot:

title	company_name	city	state	country	post_date
Software Engineer FrontEnd (Multiple Levels) - Sla...	Slack	San Francisco	CA		2025-04-10
Software Engineer, Web	Calm	San Francisco	CA		2025-04-10
Software Engineer General (new grad early career...)	Northwood	Los Angeles	CA		2025-04-06
Software Engineer (New Grad Program)	Sigma	New York	NY		2025-04-06
Software Engineer	Lyft	San Francisco County	CA		2025-04-06
Software Engineer	Lyft	Seattle	WA		2025-04-06
Intern, Software Engineer	Ticketmaster	Greater Tucson Area			2025-03-30
Full Stack Software Engineer (L5), Content Middlew...	Netflix	Los Angeles	CA		2025-03-30
Software Engineer, AI Platform - New Grad	Nuro	Mountain View	CA		2025-03-30
Software Engineer, Augmented Reality, Split Comput...	Google	Mountain View	CA		2025-03-30
Intern, Software Engineer	Ticketmaster	New York	NY		2025-03-30
Software Engineer, Integrations	Notion	New York	NY		2025-03-30
Software Engineer - New Grad	Scale AI	San Francisco Bay Area			2025-03-30
Software Engineer, Front End	Meta	Menlo Park	CA		2025-03-14
Software Engineer, Front End	Meta	San Francisco	CA		2025-03-14

Figure 4: Query 1

3.2 Query 2: Web Development Focus

English:

Retrieve all job postings where the job title or description contains 'web development', and sort them by the most recent post date. **SQL:**

```

SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
JOIN locations l ON j.location_id = l.location_id
WHERE j.title LIKE '%web development%' OR j.description LIKE '%web development%'
ORDER BY j.post_date DESC;

```

Screenshot:

The screenshot shows the phpMyAdmin interface with the following details:

- Left Sidebar:** Shows a navigation tree with categories like CORE, MISCELLANEOUS, PROJECTS, BLOG, WEBSCRAPER, BPM, and WAZ.
- Database Selection:** The database 'linkedin_jobs' is selected.
- Query Results:**
 - Query 1 (Top):** Returns an empty result set for 'Web Development Focus' (Query took 0.0015 seconds).
 - Query 2 (Second from Top):** Returns an empty result set for 'Software Engineering Roles' (Query took 0.0003 seconds).
 - Query 3 (Bottom):** Returns an empty result set for 'Software Engineering Roles' (Query took 0.0003 seconds). A warning message at the bottom states: "Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available."
- Bottom Bar:** Includes tabs for Structure, SQL, Search, Query, Export, Import, Operations, Privileges, Routines, Events, More, and a Console tab.

Figure 5: Query 2

3.3 Query 3: Software Engineering Roles

English:

Find job postings with 'software engineering' in the title and display the associated company name, full location, and post date. **SQL:**

```

SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date
FROM jobs j
JOIN companies c ON j.company_id = c.company_id

```

```

JOIN locations l ON j.location_id = l.location_id
WHERE j.title LIKE '%software engineering%'
ORDER BY j.post_date DESC;

```

Screenshot:

The screenshot shows the phpMyAdmin interface with the following details:

- Left Sidebar:** Shows a tree view of databases: information_schema, linkedin_jobs (selected), mysql, performance_schema, and sys.
- Top Bar:** Shows the URL http://localhost/phpmyadmin/index.php?route=/database/sql&db=linkedin_jobs, Server: localhost, Database: linkedin_jobs, and various navigation tabs like Structure, SQL, Search, Query, Export, Import, Operations, Privileges, Routines, Events, and More.
- Query Results:**
 - Query 1:** MySQL returned an empty result set (i.e. zero rows). (Query took 0.0015 seconds.)
-- 2. Web Development Focus `SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date FROM jobs j JOIN companies c ON j.company_id = c.company_id JOIN locations l ON j.location_id = l.location_id WHERE j.title LIKE '%web development%' OR j.description LIKE '%web development%' ORDER BY j.post_date DESC;`
 - Query 2:** MySQL returned an empty result set (i.e. zero rows). (Query took 0.0003 seconds.)
-- 3. Software Engineering Roles `SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date FROM jobs j JOIN companies c ON j.company_id = c.company_id JOIN locations l ON j.location_id = l.location_id WHERE j.title LIKE '%software engineering%' ORDER BY j.post_date DESC;`
 - Current Selection:** A warning message: "Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available." (with a help icon)
- Bottom Bar:** Shows a Console tab.

Figure 6: Query 3

3.4 Query 4: IT Position Search (Adjusted)

English:

Display all IT-related job postings, including job title, company name, location, and post date. **SQL:**

```

SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
JOIN locations l ON j.location_id = l.location_id
WHERE j.title LIKE '%IT%' OR j.description LIKE '%IT%'
ORDER BY j.post_date DESC;

```

Screenshot:

The screenshot shows the phpMyAdmin interface with the following details:

- Left Sidebar:** Shows recent databases like 'CORE', 'Data Security...', 'Miscellaneous', and various projects.
- Top Bar:** Shows the URL http://localhost/phpmyadmin/index.php?route=/database/sql&db=linkedin_jobs, the server 'localhost', and the database 'linkedin_jobs'.
- Main Area:**
 - Structure:** Shows the table structure with columns: title, company_name, city, state, country, and post_date.
 - SQL:** Displays the SQL query used to generate the results:

```
-- 4. IT Position Search (modified, working)
SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date
FROM jobs j JOIN companies c ON j.company_id = c.company_id JOIN locations l ON j.location_id = l.location_id WHERE j.title LIKE '%IT%' OR j.description LIKE '%IT%' ORDER BY j.post_date DESC;
```
 - Results:** A table showing 14 rows of job postings. The columns are: title, company_name, city, state, country, and post_date. The results are ordered by post_date DESC.

title	company_name	city	state	country	post_date
Software Engineer FrontEnd (Multiple Levels) - Sla...	Slack	San Francisco	CA		2025-04-10
Software Engineer, Web	Calm	San Francisco	CA		2025-04-10
Software Engineer (New Grad Program)	Sigma	New York	NY		2025-04-06
Software Engineer: General (new grad, early career...)	Northwood	Los Angeles	CA		2025-04-06
Software Engineer	Lyft	Seattle	WA		2025-04-06
Software Engineer	Lyft	San Francisco County	CA		2025-04-06
Intern, Software Engineer	Ticketmaster	New York	NY		2025-03-30
Software Engineer, AI Platform - New Grad	Nuro	Mountain View	CA		2025-03-30
Software Engineer, Augmented Reality, Split Comput...	Google	Mountain View	CA		2025-03-30
Software Engineer, Integrations	Notion	New York	NY		2025-03-30
Intern, Software Engineer	Ticketmaster	Greater Tucson Area			2025-03-30
Full Stack Software Engineer (L5), Content Middlew...	Netflix	Los Angeles	CA		2025-03-30
Software Engineer - New Grad	Scale AI	San Francisco Bay Area			2025-03-30
Software Engineer, Front End	Meta	Menlo Park	CA		2025-03-14
Software Engineer, Front End	Meta	San Francisco	CA		2025-03-14

Figure 7: Query 4

3.5 Query 5: Front-End vs. Back-End Focus

English:

List job postings that include either 'front end' or 'back end' in the job title, with company name, location, and post date. **SQL:**

```
SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
JOIN locations l ON j.location_id = l.location_id
WHERE j.title LIKE '%front end%' OR j.title LIKE '%back end%'
ORDER BY j.post_date DESC;
```

Screenshot:

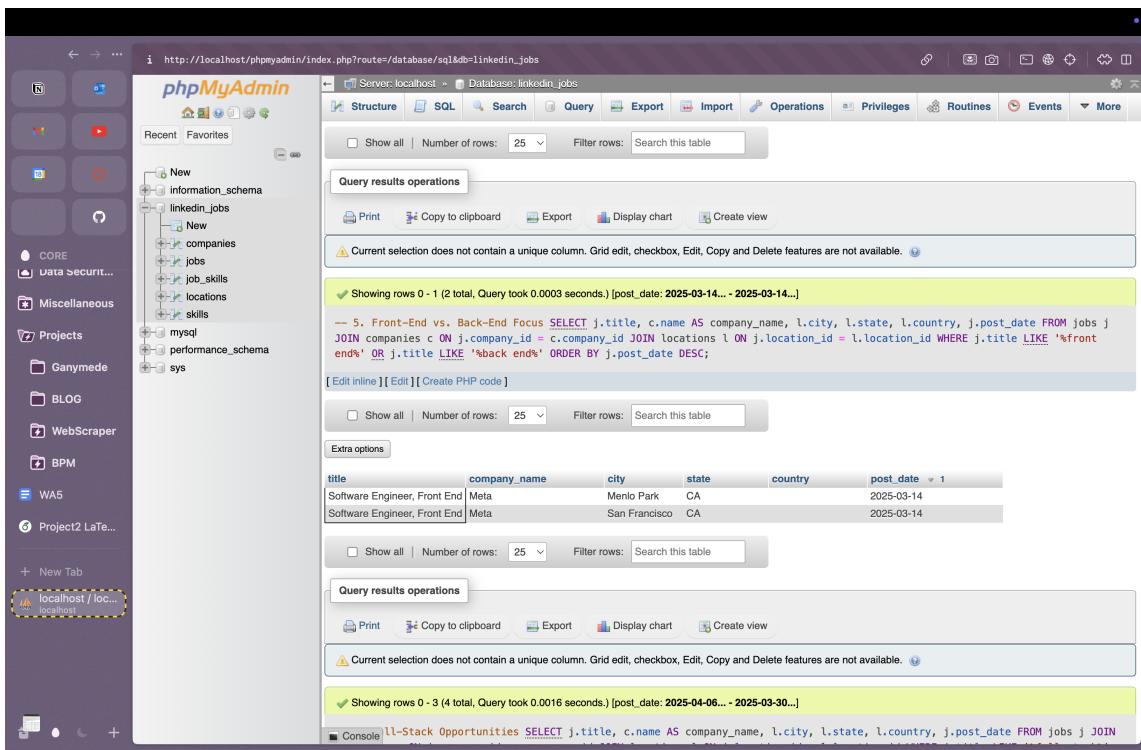


Figure 8: Query 5

3.6 Query 6: Full-Stack Opportunities

English:

Fetch job postings that mention 'full-stack' in the job title or description, displaying the job title, company name, and location. **SQL:**

```

SELECT j.title, c.name AS company_name, l.city, l.state, l.country
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
JOIN locations l ON j.location_id = l.location_id
WHERE j.title LIKE '%full-stack%' OR j.description LIKE '%full-stack%'
ORDER BY j.post_date DESC;
    
```

Screenshot:

The screenshot shows the phpMyAdmin interface for the 'linkedin_jobs' database. The left sidebar lists various databases and projects. The main area displays the results of a query:

```
-- 6. Full-Stack Opportunities
SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
JOIN locations l ON j.location_id = l.location_id
WHERE j.title LIKE '%full-stack%'
ORDER BY j.post_date DESC;
```

The results table shows the following data:

title	company_name	city	state	country	post_date
Software Engineer General (new grad early career...)	Northwood	Los Angeles	CA		2025-04-06
Software Engineer, Augmented Reality, Split Comput...	Google	Mountain View	CA		2025-03-30
Full Stack Software Engineer (L5), Content Middle...	Netflix	Los Angeles	CA		2025-03-30
Software Engineer - New Grad	Scale AI	San Francisco Bay Area			2025-03-30

Figure 9: Query 6

3.7 Query 7: Tech Skills Filter

English:

Retrieve job postings where the associated skills include both 'React' and 'Node.js', showing the job title and company name. **SQL:**

```
SELECT DISTINCT j.title, c.name AS company_name
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
JOIN job_skills js1 ON j.job_id = js1.job_id
JOIN skills s1 ON js1.skill_id = s1.skill_id
JOIN job_skills js2 ON j.job_id = js2.job_id
JOIN skills s2 ON js2.skill_id = s2.skill_id
WHERE s1.skill_name = 'React' AND s2.skill_name = 'Node.js'
ORDER BY j.post_date DESC;
```

Screenshot:

The screenshot shows the phpMyAdmin interface for the 'linkedin_jobs' database. The left sidebar shows various projects and databases. The main area displays the 'jobs' table with the following data:

title	company_name	city	state	country	post_date
Intern, Software Engineer	Ticketmaster	New York	NY		2025-03-30

Below the table, the SQL query executed is:

```
-- 7. Tech Skills Filter (React + Node.js) SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date FROM jobs j JOIN companies c ON j.company_id = c.company_id JOIN locations l ON j.location_id = l.location_id JOIN job_skills js1 ON j.job_id = js1.job_id JOIN skills s1 ON js1.skill_id = s1.skill_id JOIN job_skills js2 ON j.job_id = js2.job_id JOIN skills s2 ON js2.skill_id = s2.skill_id WHERE s1.skill_name = 'React' AND s2.skill_name = 'Node.js' GROUP BY j.job_id ORDER BY j.post_date DESC;
```

Figure 10: Query 7

3.8 Query 8: Seniority Filter

English:

List job postings for roles with seniority indicators like 'Senior' or 'Lead' in the title, along with company name, location, and post date. **SQL:**

```
SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
JOIN locations l ON j.location_id = l.location_id
WHERE j.title LIKE '%Senior%' OR j.title LIKE '%Lead%'
ORDER BY j.post_date DESC;
```

Screenshot:

The screenshot shows the phpMyAdmin interface for a database named 'linkedin_jobs'. The left sidebar lists databases like 'information_schema', 'linkedin_jobs' (selected), 'mysql', and 'sys'. The main area has two query panes.

Query 8:

```
-- 8. Seniority Filter
SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
JOIN locations l ON j.location_id = l.location_id
WHERE j.title LIKE '%Senior%' OR j.title LIKE '%Lead%'
ORDER BY j.post_date DESC;
```

Query 9:

```
-- 9. Remote Work Focus (modified, working)
SELECT j.title, c.name AS company_name, l.city, l.state, l.country, j.post_date
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
JOIN locations l ON j.location_id = l.location_id
WHERE j.description LIKE '%remote%' OR j.description LIKE '%telecommute%'
ORDER BY j.post_date DESC;
```

The results for Query 9 show one row:

title	company_name	city	state	country	post_date
Software Engineer, Web Calm	Calm	San Francisco	CA		2025-04-10

Figure 11: Query 8

3.9 Query 9: Remote Work Focus (Adjusted)

English:

Display all job postings that mention 'remote' or 'telecommute' in the description, including job title, company name, and post date. **SQL:**

```
SELECT j.title, c.name AS company_name, j.post_date
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
WHERE j.description LIKE '%remote%' OR j.description LIKE '%telecommute%'
ORDER BY j.post_date DESC;
```

Screenshot:

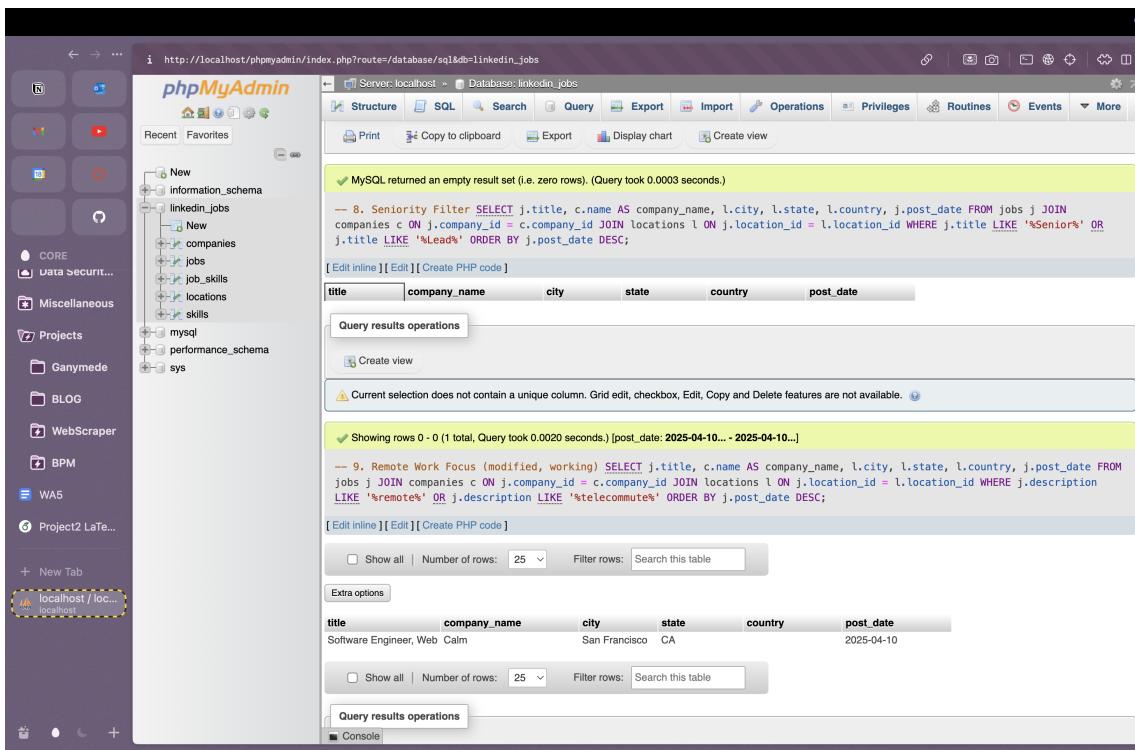


Figure 12: Query 9

3.10 Query 10: Category Summary Count

English:

Provide a summary count of job postings by category (e.g., web development, software engineering, IT, front-end, back-end, full-stack) based on keywords in the job title or description. **SQL:**

SELECT

CASE

```

WHEN j.title LIKE '%web development%' OR j.description LIKE '%web development%' THEN 'Web Development'
WHEN j.title LIKE '%software engineering%' OR j.description LIKE '%software engineering%' THEN 'Software Engineering'
WHEN j.title LIKE '%IT%' OR j.description LIKE '%IT%' THEN 'IT'
WHEN j.title LIKE '%front end%' OR j.description LIKE '%front end%' THEN 'Front End'
WHEN j.title LIKE '%back end%' OR j.description LIKE '%back end%' THEN 'Back End'
WHEN j.title LIKE '%full-stack%' OR j.description LIKE '%full-stack%' THEN 'Full Stack'
ELSE 'Other'

```

END AS category,

COUNT(*) AS job_count

```
FROM jobs j
GROUP BY category;
```

Screenshot:

category	job_count
IT	15

Figure 13: Query 10

4 Backend Accomplishments

- Clean and normalized database schema implemented
- Data stored dynamically in MySQL, visible in phpMyAdmin
- Python scraper successfully pulls live data from LinkedIn
- Parsing algorithm is hardcoded for precision
- CLI tool `query_db.py` works for interactive querying

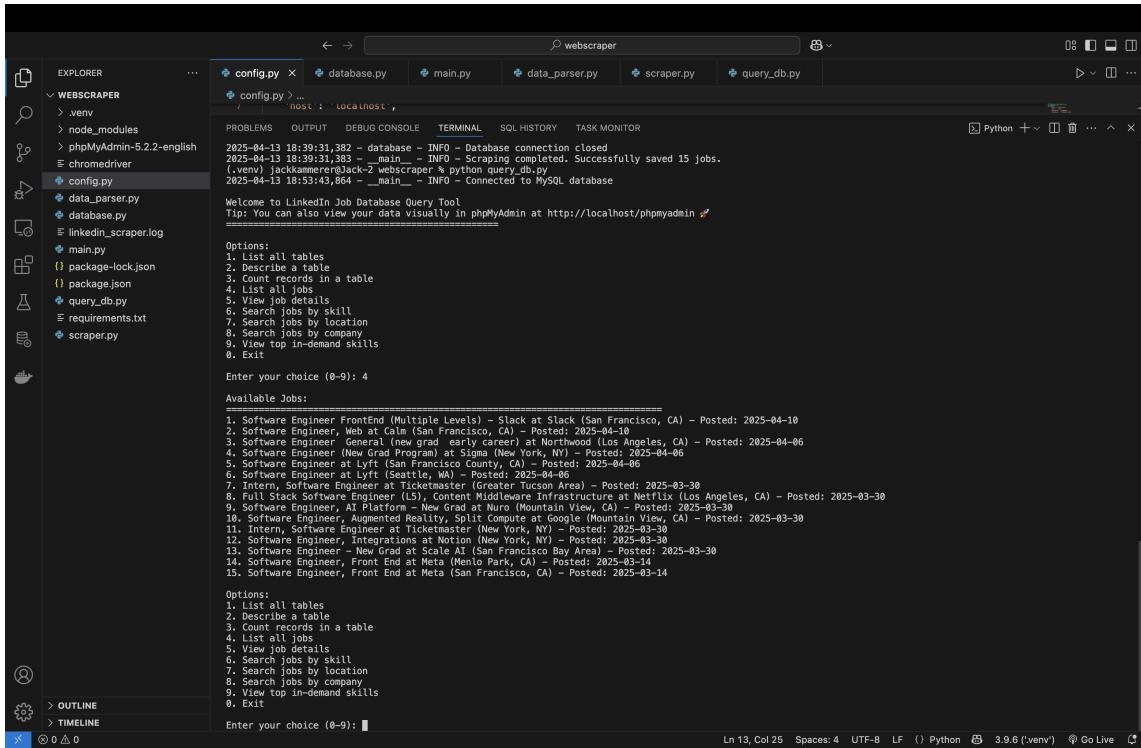


Figure 14: Developer File - To Test New Data

5 Source Code

5.1 Main Python Files

5.2 main.py

```

1 """
2 Main execution script for LinkedIn job scraper.
3 This script orchestrates the entire scraping, parsing, and saving process.
4 """
5
6 import sys
7 import logging
8 import time
9 from scraper import LinkedInScraper
10 from data_parser import process_job_data
11 from database import Database
12 from config import SAMPLE_URL
13
14 # Configure logging

```

```

15 logging.basicConfig(
16     level=logging.INFO,
17     format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
18     handlers=[
19         logging.FileHandler("linkedin_scraper.log"),
20         logging.StreamHandler(sys.stdout)
21     ]
22 )
23 logger = logging.getLogger(__name__)
24
25
26 def run_scraper(url):
27     """
28     Run the complete LinkedIn job scraping process.
29
30     Args:
31         url (str): LinkedIn job search URL
32
33     Returns:
34         int: Number of jobs successfully scraped and saved
35     """
36     logger.info("Starting LinkedIn job scraper")
37
38     db = None
39     scraper = None
40     job_count = 0
41
42     try:
43         # Initialize database
44         db = Database()
45         db.reset_database()
46         logger.info("Database initialized and reset")
47
48         # Initialize scraper
49         scraper = LinkedInScraper(db, headless=True)
50         logger.info("LinkedIn scraper initialized")
51
52         # Scrape job listing URLs
53         job_urls = scraper.scrape_job_listings(url)
54
55         if not job_urls:
56             logger.warning("No job listings found")
57             return 0

```

```

58
59         logger.info(f"Found {len(job_urls)} job listings, proceeding to
60             scrape details")
61
62     # Process each job URL
63     for job_url in job_urls:
64         try:
65             # Scrape job details
66             raw_job_data = scraper.scrape_job_details(job_url)
67
68             if not raw_job_data or 'title' not in raw_job_data:
69                 logger.warning(f"Skipping job, could not scrape details
70 : {job_url}")
71                 continue
72
73             # Process and clean job data
74             processed_job_data = process_job_data(raw_job_data)
75
76             # Save to database
77             db.save_job_listing(processed_job_data)
78             job_count += 1
79
80             logger.info(f"Successfully processed and saved job: {
81 processed_job_data['title']} ")
82
83             # Random sleep between requests
84             time.sleep(2) # Small delay between jobs
85
86         except Exception as e:
87             logger.error(f"Error processing job {job_url}: {e}")
88             continue
89
90     logger.info(f"Completed job scraping. Successfully saved {job_count}
91 } jobs.")
92     return job_count
93
94 except Exception as e:
95     logger.error(f"Error in job scraping process: {e}")
96     return job_count
97
98 finally:
99     # Clean up resources
100    if scraper:

```

```

97     scraper.close()
98
99     if db:
100         db.close()
101
102
103 if __name__ == "__main__":
104     # Get URL from command line or use sample URL
105     url = sys.argv[1] if len(sys.argv) > 1 else SAMPLE_URL
106
107     logger.info(f"Starting job scraping with URL: {url}")
108     success_count = run_scraper(url)
109
110     logger.info(f"Scraping completed. Successfully saved {success_count} jobs.")

```

5.3 query_db.py

```

1 """
2 Database query tool for LinkedIn job scraper.
3 This script helps query the normalized database tables.
4 """
5
6 import mysql.connector
7 import logging
8 from config import DB_CONFIG
9
10 # Configure logging
11 logging.basicConfig(
12     level=logging.INFO,
13     format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
14 )
15 logger = logging.getLogger(__name__)
16
17 class QueryTool:
18     def __init__(self):
19         """Initialize database connection."""
20         self.connection = None
21         try:
22             self.connection = mysql.connector.connect(**DB_CONFIG)
23             if self.connection.is_connected():
24                 logger.info("Connected to MySQL database")
25         except Exception as e:
26             logger.error(f"Error connecting to MySQL: {e}")

```

```

27         raise
28
29     def list_tables(self):
30         """List all tables in the database."""
31         cursor = self.connection.cursor()
32
33         try:
34             cursor.execute("SHOW TABLES")
35             tables = cursor.fetchall()
36             print("\nAvailable tables:")
37             for table in tables:
38                 print(f"- {table[0]}")
39             return tables
40         except Exception as e:
41             logger.error(f"Error listing tables: {e}")
42             return []
43
44         finally:
45             cursor.close()
46
47     def describe_table(self, table_name):
48         """Describe the structure of a specific table."""
49         cursor = self.connection.cursor()
50
51         try:
52             cursor.execute(f"DESCRIBE {table_name}")
53             columns = cursor.fetchall()
54             print(f"\nTable structure for '{table_name}':")
55             for col in columns:
56                 print(f"- {col[0]} ({col[1]})")
57             return columns
58         except Exception as e:
59             logger.error(f"Error describing table: {e}")
60             return []
61
62         finally:
63             cursor.close()
64
65     def count_records(self, table_name):
66         """Count the number of records in a table."""
67         cursor = self.connection.cursor()
68
69         try:
70             cursor.execute(f"SELECT COUNT(*) FROM {table_name}")
71             count = cursor.fetchone()[0]
72             print(f"\nNumber of records in '{table_name}': {count}")
73             return count
74         except Exception as e:
75

```

```

70         logger.error(f"Error counting records: {e}")
71         return 0
72     finally:
73         cursor.close()
74
75     def list_jobs(self):
76         """List all jobs with company and location details."""
77         cursor = self.connection.cursor(dictionary=True)
78         try:
79             query = """
80                 SELECT j.job_id, j.title, c.name AS company_name,
81                         l.city, l.state, l.country, j.post_date
82                 FROM jobs j
83                 JOIN companies c ON j.company_id = c.company_id
84                 JOIN locations l ON j.location_id = l.location_id
85                 ORDER BY j.post_date DESC
86             """
87
88             cursor.execute(query)
89             jobs = cursor.fetchall()
90
91             print("\nAvailable Jobs:")
92             print("=" * 80)
93             for idx, job in enumerate(jobs, 1):
94                 location = ', '.join(filter(None, [job['city'], job['state'],
95                     job['country']])))
96                 print(f"{idx}. {job['title']} at {job['company_name']} ({location}) - Posted: {job['post_date']}")
97
98             return jobs
99         except Exception as e:
100             logger.error(f"Error listing jobs: {e}")
101             return []
102     finally:
103         cursor.close()
104
105     def job_details(self, job_id):
106         """Get detailed information about a specific job."""
107         cursor = self.connection.cursor(dictionary=True)
108         try:
109             job_query = """
110                 SELECT j.job_id, j.title, c.name AS company_name, c.industry,
111                         c.size AS company_size,
112                         l.city, l.state, l.country, j.description, j.post_date

```

```

110     FROM jobs j
111     JOIN companies c ON j.company_id = c.company_id
112     JOIN locations l ON j.location_id = l.location_id
113     WHERE j.job_id = %s
114     """
115
116     cursor.execute(job_query, (job_id,))
117     job = cursor.fetchone()
118
119     if not job:
120         print(f"No job found with ID {job_id}")
121         return None
122
123     # Get skills for this job
124     skills_query = """
125     SELECT s.skill_name
126     FROM job_skills js
127     JOIN skills s ON js.skill_id = s.skill_id
128     WHERE js.job_id = %s
129     """
130
131     cursor.execute(skills_query, (job_id,))
132     skills = [row['skill_name'] for row in cursor.fetchall()]
133
134     # Print job details
135     print("\nJob Details:")
136     print("=" * 80)
137     print(f"Title: {job['title']}")
138     print(f"Company: {job['company_name']}")
139     print(f"Industry: {job['industry']} or 'Not specified'")
140     print(f"Company Size: {job['company_size']} or 'Not specified'")
141
142     location = ', '.join(filter(None, [job['city'], job['state'],
143                                   job['country']])))
144     print(f"Location: {location}")
145     print(f"Posted on: {job['post_date']}")
146
147     print("\nSkills:")
148     if skills:
149         for skill in skills:
150             print(f"- {skill}")
151     else:
152         print("No skills listed")

```

```

151         print("\nDescription:")
152         print("-" * 80)
153         print(job['description'])
154
155     return {**job, 'skills': skills}
156 except Exception as e:
157     logger.error(f"Error getting job details: {e}")
158     return None
159 finally:
160     cursor.close()
161
162 def search_jobs_by_skill(self, skill):
163     """Search for jobs requiring a specific skill."""
164     cursor = self.connection.cursor(dictionary=True)
165     try:
166         query = """
167             SELECT j.job_id, j.title, c.name AS company_name,
168                 l.city, l.state, l.country
169             FROM jobs j
170             JOIN companies c ON j.company_id = c.company_id
171             JOIN locations l ON j.location_id = l.location_id
172             JOIN job_skills js ON j.job_id = js.job_id
173             JOIN skills s ON js.skill_id = s.skill_id
174             WHERE s.skill_name LIKE %s
175             ORDER BY j.post_date DESC
176         """
177
178         cursor.execute(query, (f"%{skill}%",))
179         jobs = cursor.fetchall()
180
181         print(f"\nJobs requiring '{skill}':")
182         print("=" * 80)
183         if jobs:
184             for job in jobs:
185                 location = ', '.join(filter(None, [job['city'], job['state'], job['country']]))

186                 print(f"{job['job_id']}. {job['title']} at {job['company_name']} ({location})")
187             else:
188                 print(f"No jobs found requiring '{skill}'")
189
190     return jobs
191 except Exception as e:
192     logger.error(f"Error searching jobs by skill: {e}")

```

```

192         return []
193     finally:
194         cursor.close()
195
196     def search_jobs_by_location(self, location):
197         """Search for jobs in a specific location."""
198         cursor = self.connection.cursor(dictionary=True)
199
200         try:
201             query = """
202                 SELECT j.job_id, j.title, c.name AS company_name,
203                     l.city, l.state, l.country
204                 FROM jobs j
205                 JOIN companies c ON j.company_id = c.company_id
206                 JOIN locations l ON j.location_id = l.location_id
207                 WHERE l.city LIKE %s OR l.state LIKE %s OR l.country LIKE %
208                 ORDER BY j.post_date DESC
209             """
210
211             pattern = f"%{location}%""
212             cursor.execute(query, (pattern, pattern, pattern))
213             jobs = cursor.fetchall()
214
215             print(f"\nJobs in '{location}':")
216             print("=" * 80)
217             if jobs:
218                 for job in jobs:
219                     location = ', '.join(filter(None, [job['city'], job['state'], job['country']])))
220                     print(f"{job['job_id']}. {job['title']} at {job['company_name']} ({location})")
221             else:
222                 print(f"No jobs found in '{location}'")
223
224             return jobs
225         except Exception as e:
226             logger.error(f"Error searching jobs by location: {e}")
227             return []
228
229     def search_jobs_by_company(self, company):
230         """Search for jobs at a specific company."""
231         cursor = self.connection.cursor(dictionary=True)
232         try:

```

```

233     query = """
234     SELECT j.job_id, j.title, c.name AS company_name,
235         l.city, l.state, l.country
236     FROM jobs j
237     JOIN companies c ON j.company_id = c.company_id
238     JOIN locations l ON j.location_id = l.location_id
239     WHERE c.name LIKE %s
240     ORDER BY j.post_date DESC
241     """
242
243     cursor.execute(query, (f"%{company}%",))
244     jobs = cursor.fetchall()
245
246     print(f"\nJobs at '{company}':")
247     print("=" * 80)
248     if jobs:
249         for job in jobs:
250             location = ', '.join(filter(None, [job['city'], job['state'], job['country']])))
251             print(f"{job['job_id']}. {job['title']} at {job['company_name']} ({location})")
252     else:
253         print(f"No jobs found at '{company}'")
254
255     return jobs
256 except Exception as e:
257     logger.error(f"Error searching jobs by company: {e}")
258     return []
259 finally:
260     cursor.close()
261
262 def top_skills(self, limit=10):
263     """List the most in-demand skills based on job listings."""
264     cursor = self.connection.cursor(dictionary=True)
265     try:
266         query = """
267             SELECT s.skill_name, COUNT(js.job_id) as job_count
268             FROM skills s
269             JOIN job_skills js ON s.skill_id = js.skill_id
270             GROUP BY s.skill_name
271             ORDER BY job_count DESC
272             LIMIT %s
273             """
274
275         cursor.execute(query, (limit,))

```

```

274     skills = cursor.fetchall()
275
276     print(f"\nTop {limit} in-demand skills:")
277     print("=" * 50)
278     for i, skill in enumerate(skills, 1):
279         print(f"{i}. {skill['skill_name']} - Found in {skill['job_count']} job(s)")
280
281     return skills
282 except Exception as e:
283     logger.error(f"Error getting top skills: {e}")
284     return []
285 finally:
286     cursor.close()
287
288 def close(self):
289     """Close the database connection."""
290     if self.connection and self.connection.is_connected():
291         self.connection.close()
292     logger.info("Database connection closed")
293
294
295 if __name__ == "__main__":
296     tool = QueryTool()
297
298     try:
299         print("\nWelcome to LinkedIn Job Database Query Tool")
300         print("Tip: You can also view your data visually in phpMyAdmin at")
301         print("http://localhost/phpmyadmin      ")
302         print("=" * 50)
303
304         while True:
305             print("\nOptions:")
306             print("1. List all tables")
307             print("2. Describe a table")
308             print("3. Count records in a table")
309             print("4. List all jobs")
310             print("5. View job details")
311             print("6. Search jobs by skill")
312             print("7. Search jobs by location")
313             print("8. Search jobs by company")
314             print("9. View top in-demand skills")
315             print("0. Exit")

```

```

315
316     choice = input("\nEnter your choice (0-9): ")
317
318     if choice == '0':
319         print("Exiting...")
320         break
321     elif choice == '1':
322         tool.list_tables()
323     elif choice == '2':
324         table = input("Enter table name: ")
325         tool.describe_table(table)
326     elif choice == '3':
327         table = input("Enter table name: ")
328         tool.count_records(table)
329     elif choice == '4':
330         tool.list_jobs()
331     elif choice == '5':
332         job_id = input("Enter job ID: ")
333         tool.job_details(int(job_id))
334     elif choice == '6':
335         skill = input("Enter skill to search: ")
336         tool.search_jobs_by_skill(skill)
337     elif choice == '7':
338         location = input("Enter location to search: ")
339         tool.search_jobs_by_location(location)
340     elif choice == '8':
341         company = input("Enter company to search: ")
342         tool.search_jobs_by_company(company)
343     elif choice == '9':
344         try:
345             limit = int(input("Enter number of skills to show ("
346                         "default 10): ") or 10)
347             tool.top_skills(limit)
348         except ValueError:
349             print("Invalid input. Using default limit of 10.")
350             tool.top_skills()
351     else:
352         print("Invalid choice. Please try again.")
353
354     except Exception as e:
355         logger.error(f"An error occurred: {e}")
356

```

357 tool.close()

5.4 scraper.py

```

1 """
2 LinkedIn job scraper module.
3 This module handles the web scraping functionality.
4 """
5
6 import time
7 import random
8 import logging
9 import mysql.connector
10 from config import DB_CONFIG
11 from database import Database
12 from selenium import webdriver
13 from selenium.webdriver.chrome.service import Service
14 from selenium.webdriver.chrome.options import Options
15 from selenium.webdriver.common.by import By
16 from selenium.webdriver.support.ui import WebDriverWait
17 from selenium.webdriver.support import expected_conditions as EC
18 from selenium.common.exceptions import (
19     TimeoutException, NoSuchElementException,
20     StaleElementReferenceException
21 )
22 from config import SCRAPING_CONFIG, USER_AGENT
23
24 # Configure logging
25 logging.basicConfig(
26     level=logging.INFO,
27     format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
28 )
29 logger = logging.getLogger(__name__)
30
31
32 class LinkedInScraper:
33     def __init__(self, db, headless=True):
34         """
35             Initialize the LinkedIn scraper.
36
37         Args:
38             headless (bool): Whether to run the browser in headless mode
39         """

```

```

40         self.db = db
41         self.setup_driver(headless)
42
43     def setup_driver(self, headless):
44         """
45             Set up the Selenium WebDriver.
46
47         Args:
48             headless (bool): Whether to run in headless mode
49         """
50
51         chrome_options = Options()
52         if headless:
53             chrome_options.add_argument("--headless")
54
55             chrome_options.add_argument("--no-sandbox")
56             chrome_options.add_argument("--disable-dev-shm-usage")
57             chrome_options.add_argument("--disable-gpu")
58             chrome_options.add_argument(f"user-agent={USER_AGENT}")
59
60         service = Service('./chromedriver')
61         self.driver = webdriver.Chrome(service=service, options=
62             chrome_options)
63
64         try:
65             self.driver.set_window_size(1920, 1080)
66             self.wait = WebDriverWait(self.driver, SCRAPING_CONFIG['timeout']
67         ])
68             logger.info("WebDriver initialized successfully")
69         except Exception as e:
70             logger.error(f"Failed to initialize WebDriver: {e}")
71             raise
72
73     def scrape_job_listings(self, url):
74         """
75             Scrape job listings from a LinkedIn search URL.
76
77         Args:
78             url (str): LinkedIn job search URL
79
80         Returns:
81             list: List of job listing URLs
82         """
83
84         logger.info(f"Starting to scrape job listings from: {url}")

```

```

81     job_urls = []
82
83     try:
84         self.driver.get(url)
85         time.sleep(random.uniform(3, 5)) # Initial wait for page to
86         load
87
88         # Get job cards
89         job_cards = self.wait.until(
90             EC.presence_of_all_elements_located(
91                 (By.CSS_SELECTOR, ".jobs-search__results-list li")
92             )
93
94         # Extract job URLs
95         count = 0
96         for card in job_cards:
97             if count >= SCRAPING_CONFIG['max_jobs']:
98                 break
99
100            try:
101                job_link = card.find_element(By.CSS_SELECTOR, "a").
102                get_attribute("href")
103                job_urls.append(job_link)
104                count += 1
105            except (NoSuchElementException,
106                    StaleElementReferenceException) as e:
107                logger.warning(f"Could not extract job link: {e}")
108                continue
109
110            logger.info(f"Found {len(job_urls)} job listings")
111            return job_urls
112
113        except TimeoutException:
114            logger.error("Timeout while loading LinkedIn job search results
115        ")
116            return []
117
118    except Exception as e:
119        logger.error(f"Error scraping job listings: {e}")
120        return []
121
122    def scrape_job_details(self, job_url):
123        """

```

```

120     Scrape detailed information for a specific job.
121
122     Args:
123         job_url (str): URL of the job listing
124
125     Returns:
126         dict: Job details including title, company, location, etc.
127         """
128         logger.info(f"Scraping details for job: {job_url}")
129         job_data = {}
130
131     try:
132         self.driver.get(job_url)
133         time.sleep(random.uniform(*SCRAPING_CONFIG['sleep_interval']))
134
135         job_data['title'] = self.safe_extract(By.CSS_SELECTOR, "h1.topcard__title")
136         job_data['company_name'] = self.safe_extract(By.CSS_SELECTOR, "a.topcard__org-name-link")
137         location = self.safe_extract(By.CSS_SELECTOR, "span.topcard__flavor.topcard__flavor--bullet")
138         if location:
139             location_parts = location.split(',')
140             job_data['city'] = location_parts[0].strip() if len(location_parts) > 0 else None
141             job_data['state'] = location_parts[1].strip() if len(location_parts) > 1 else None
142             job_data['country'] = location_parts[-1].strip() if len(location_parts) > 2 else None
143
144             job_data['description'] = self.safe_extract(By.CSS_SELECTOR, "div.description__text", get_text=True)
145             job_data['post_date'] = self.safe_extract(By.CSS_SELECTOR, "span.posted-time-ago__text")
146
147             skills_elements = self.driver.find_elements(By.CSS_SELECTOR, ".skill-tag")
148             job_data['skills'] = [skill.text for skill in skills_elements
149             if skill.text]
150
151             job_data['company_size'] = self.safe_extract(By.CSS_SELECTOR, ".company-size")

```

```

151         job_data['industry'] = self.safe_extract(By.CSS_SELECTOR, ".company-industry")
152
153         logger.info(f"Successfully scraped job details for: {job_data.get('title', 'Unknown job')}")
154         return job_data
155
156     except TimeoutException:
157         logger.error(f"Timeout while loading job details: {job_url}")
158         return {}
159     except Exception as e:
160         logger.error(f"Error scraping job details: {e}")
161         return {}
162
163     def safe_extract(self, by, selector, get_text=False):
164         """
165             Safely extract an element's text or attribute.
166
167         Args:
168             by: Selenium By locator
169             selector (str): CSS selector
170             get_text (bool): Whether to get innerText instead of
textContent
171
172         Returns:
173             str: Extracted text or None if element not found
174         """
175
176         try:
177             element = self.driver.find_element(by, selector)
178             if get_text:
179                 return element.get_attribute("innerText")
180             return element.text
181         except (NoSuchElementException, StaleElementReferenceException):
182             return None
183
184     def close(self):
185         """Close the WebDriver."""
186         if self.driver:
187             self.driver.quit()
188             logger.info("WebDriver closed")

```

5.5 database.py

1 """\n

```

2 Database operations for the LinkedIn job scraper.
3 This module handles MySQL connections and database queries.
4 """
5
6 import mysql.connector
7 from mysql.connector import Error
8 import logging
9 from config import DB_CONFIG
10
11 # Configure logging
12 logging.basicConfig(
13     level=logging.INFO,
14     format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
15 )
16 logger = logging.getLogger(__name__)
17
18
19 class Database:
20     def __init__(self):
21         """Initialize database connection and create tables if they don't
22         exist."""
23         self.connection = None
24         try:
25             self.connection = mysql.connector.connect(**DB_CONFIG)
26             if self.connection.is_connected():
27                 logger.info("Connected to MySQL database")
28                 self.create_tables()
29         except Error as e:
30             logger.error(f"Error connecting to MySQL: {e}")
31             raise
32
33     def reset_database(self):
34         """
35         Drop all tables and recreate them to start fresh.
36         """
37         cursor = self.connection.cursor()
38         try:
39             logger.info("Resetting database...")
40
41             # Disable foreign key checks to avoid constraint errors during
42             drop
43             cursor.execute("SET FOREIGN_KEY_CHECKS = 0;")
```

```

43         # Drop tables if they exist
44         tables = ['job_skills', 'jobs', 'skills', 'locations', ,
45 companies']
46         for table in tables:
47             cursor.execute(f"DROP TABLE IF EXISTS {table};")
48             logger.info(f"Dropped table {table}")
49
50         # Re-enable foreign key checks
51         cursor.execute("SET FOREIGN_KEY_CHECKS = 1;")
52
53         # Recreate the tables
54         self.create_tables()
55         logger.info("Database reset and tables recreated successfully.")
56     )
57
58 except Error as e:
59     logger.error(f"Error resetting database: {e}")
60     raise
61 finally:
62     cursor.close()
63
64 def create_tables(self):
65     """Create necessary tables if they don't exist."""
66     cursor = self.connection.cursor()
67     try:
68         # Create companies table
69         cursor.execute('',
70             CREATE TABLE IF NOT EXISTS companies (
71                 company_id INT AUTO_INCREMENT PRIMARY KEY,
72                 name VARCHAR(255) NOT NULL,
73                 industry VARCHAR(255),
74                 size VARCHAR(100),
75                 UNIQUE (name)
76             )
77         , '')
78
79         # Create locations table
80         cursor.execute('',
81             CREATE TABLE IF NOT EXISTS locations (
82                 location_id INT AUTO_INCREMENT PRIMARY KEY,
83                 city VARCHAR(100),
84                 state VARCHAR(100),
85                 country VARCHAR(100),
86             )
87     )
88
89 
```

```

84             UNIQUE (city, state, country)
85         )
86     ''')
87
88     # Create skills table
89     cursor.execute('''
90         CREATE TABLE IF NOT EXISTS skills (
91             skill_id INT AUTO_INCREMENT PRIMARY KEY,
92             skill_name VARCHAR(100) NOT NULL,
93             UNIQUE (skill_name)
94         )
95     ''')
96
97     # Create jobs table
98     cursor.execute('''
99         CREATE TABLE IF NOT EXISTS jobs (
100            job_id INT AUTO_INCREMENT PRIMARY KEY,
101            title VARCHAR(255) NOT NULL,
102            company_id INT,
103            location_id INT,
104            description TEXT,
105            post_date DATE,
106            FOREIGN KEY (company_id) REFERENCES companies (
107                company_id),
108            FOREIGN KEY (location_id) REFERENCES locations (
109                location_id)
110        ''')
111
112     # Create job_skills join table
113     cursor.execute('''
114         CREATE TABLE IF NOT EXISTS job_skills (
115             job_id INT,
116             skill_id INT,
117             PRIMARY KEY (job_id, skill_id),
118             FOREIGN KEY (job_id) REFERENCES jobs (job_id) ON DELETE
119             CASCADE,
120             FOREIGN KEY (skill_id) REFERENCES skills (skill_id) ON
121             DELETE CASCADE
122         ''')
123
124     self.connection.commit()

```

```

123         logger.info("Database tables created successfully")
124     except Error as e:
125         logger.error(f"Error creating tables: {e}")
126         raise
127     finally:
128         cursor.close()
129
130     def insert_company(self, name, industry=None, size=None):
131         """
132             Insert a company or get its ID if it already exists.
133
134         Args:
135             name (str): Company name
136             industry (str, optional): Company industry
137             size (str, optional): Company size
138
139         Returns:
140             int: The company_id
141         """
142         cursor = self.connection.cursor()
143         try:
144             # Check if company already exists
145             cursor.execute("SELECT company_id FROM companies WHERE name = %s", (name,))
146             result = cursor.fetchone()
147
148             if result:
149                 return result[0]
150
151             # Insert new company
152             cursor.execute(
153                 "INSERT INTO companies (name, industry, size) VALUES (%s, %s, %s)",
154                 (name, industry, size)
155             )
156             self.connection.commit()
157             return cursor.lastrowid
158         except Error as e:
159             self.connection.rollback()
160             logger.error(f"Error inserting company: {e}")
161             raise
162         finally:
163             cursor.close()

```

```

164
165     def insert_location(self, city=None, state=None, country=None):
166         """
167             Insert a location or get its ID if it already exists.
168
169         Args:
170             city (str, optional): City name
171             state (str, optional): State name
172             country (str, optional): Country name
173
174         Returns:
175             int: The location_id
176         """
177
178         cursor = self.connection.cursor()
179
180         try:
181             # Check if location already exists
182             cursor.execute(
183                 "SELECT location_id FROM locations WHERE city = %s AND
184                 state = %s AND country = %s",
185                 (city, state, country)
186             )
187             result = cursor.fetchone()
188
189             if result:
190                 return result[0]
191
192             # Insert new location
193             cursor.execute(
194                 "INSERT INTO locations (city, state, country) VALUES (%s, %
195                 s, %s)",
196                 (city, state, country)
197             )
198             self.connection.commit()
199             return cursor.lastrowid
200
201         except Error as e:
202             self.connection.rollback()
203             logger.error(f"Error inserting location: {e}")
204             raise
205
206         finally:
207             cursor.close()
208
209     def insert_skill(self, skill_name):
210         """
211

```

```

205     Insert a skill or get its ID if it already exists.
206
207     Args:
208         skill_name (str): Name of the skill
209
210     Returns:
211         int: The skill_id
212     """
213
214     cursor = self.connection.cursor()
215     try:
216         # Check if skill already exists
217         cursor.execute("SELECT skill_id FROM skills WHERE skill_name = %s", (skill_name,))
218         result = cursor.fetchone()
219
220         if result:
221             return result[0]
222
223         # Insert new skill
224         cursor.execute("INSERT INTO skills (skill_name) VALUES (%s)", (skill_name,))
225         self.connection.commit()
226         return cursor.lastrowid
227     except Error as e:
228         self.connection.rollback()
229         logger.error(f"Error inserting skill: {e}")
230         raise
231     finally:
232         cursor.close()
233
234     def insert_job(self, title, company_id, location_id, description, post_date):
235         """
236             Insert a job listing.
237
238             Args:
239                 title (str): Job title
240                 company_id (int): Foreign key to companies table
241                 location_id (int): Foreign key to locations table
242                 description (str): Job description
243                 post_date (str): Date job was posted
244
245             Returns:

```

```

245         int: The job_id
246
247     cursor = self.connection.cursor()
248
249     try:
250         cursor.execute(
251             "INSERT INTO jobs (title, company_id, location_id,
252             description, post_date) "
253             "VALUES (%s, %s, %s, %s, %s)",
254             (title, company_id, location_id, description, post_date)
255         )
256         self.connection.commit()
257         return cursor.lastrowid
258     except Error as e:
259         self.connection.rollback()
260         logger.error(f"Error inserting job: {e}")
261         raise
262     finally:
263         cursor.close()
264
265     def link_job_skill(self, job_id, skill_id):
266
267         """
268         Create a job-skill relationship in the join table.
269
270         Args:
271             job_id (int): ID of the job
272             skill_id (int): ID of the skill
273
274             cursor = self.connection.cursor()
275
276             try:
277                 cursor.execute(
278                     "INSERT INTO job_skills (job_id, skill_id) VALUES (%s, %s)"
279
280                     ,
281                     (job_id, skill_id)
282                 )
283                 self.connection.commit()
284             except Error as e:
285                 self.connection.rollback()
286                 logger.error(f"Error linking job and skill: {e}")
287                 raise
288             finally:
289                 cursor.close()
290
291     def save_job_listing(self, job_data):

```

```

286     """
287     Save a complete job listing to the database.
288
289     Args:
290         job_data (dict): Dictionary containing job information
291
292     Returns:
293         int: The job_id of the inserted job
294     """
295
296     try:
297         # Insert company and get ID
298         company_id = self.insert_company(
299             job_data['company_name'],
300             job_data.get('industry'),
301             job_data.get('company_size')
302         )
303
304         # Insert location and get ID
305         location_id = self.insert_location(
306             job_data.get('city'),
307             job_data.get('state'),
308             job_data.get('country')
309         )
310
311         # Insert job and get ID
312         job_id = self.insert_job(
313             job_data['title'],
314             company_id,
315             location_id,
316             job_data.get('description', ''),
317             job_data.get('post_date')
318         )
319
320         # Insert skills and link to job
321         for skill in job_data.get('skills', []):
322             skill_id = self.insert_skill(skill)
323             self.link_job_skill(job_id, skill_id)
324
325             logger.info(f"Successfully saved job: {job_data['title']} at {job_data['company_name']}")
326
327             return job_id
328
329     except Error as e:

```

```

328         logger.error(f"Error saving job listing: {e}")
329         raise
330
331     def close(self):
332         """Close the database connection."""
333         if self.connection and self.connection.is_connected():
334             self.connection.close()
335         logger.info("Database connection closed")

```

5.6 config.py

```

1 """
2 Configuration settings for LinkedIn job scraper.
3 """
4
5 # Database configuration
6 DB_CONFIG = {
7     'host': 'localhost',
8     'user': 'root',
9     'password': 'swagdata',
10    'database': 'linkedin_jobs',
11 }
12
13 # Scraping configuration
14 SCRAPING_CONFIG = {
15     'timeout': 30, # Maximum seconds to wait for page loads
16     'jobs_per_page': 25,
17     'max_jobs': 15, # Maximum number of jobs to scrape
18     'sleep_interval': (2, 5), # Random sleep interval between requests (
19     min, max)
20 }
21 # User agent for requests
22 USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
23 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36"
24 # Sample LinkedIn job search URL
25 SAMPLE_URL = "https://www.linkedin.com/jobs/search/?keywords=software%20
engineer&location=United%20States"

```

6 Future Implementations

As I move forward with this project, my focus will be on refining and finalizing key components to ensure full functionality and presentation readiness. Specifically, I will:

- Finalize and test all SQL queries directly within phpMyAdmin to ensure accurate results
- Export the populated database schema for backup and grading purposes
- Prepare final screenshots of database tables, E/R diagram, and SQL query outputs
- Integrate English queries clearly mapped to their SQL counterparts within the report
- Insert all source code files into the final LaTeX report for complete project documentation
- Complete the final report and assemble the presentation deck

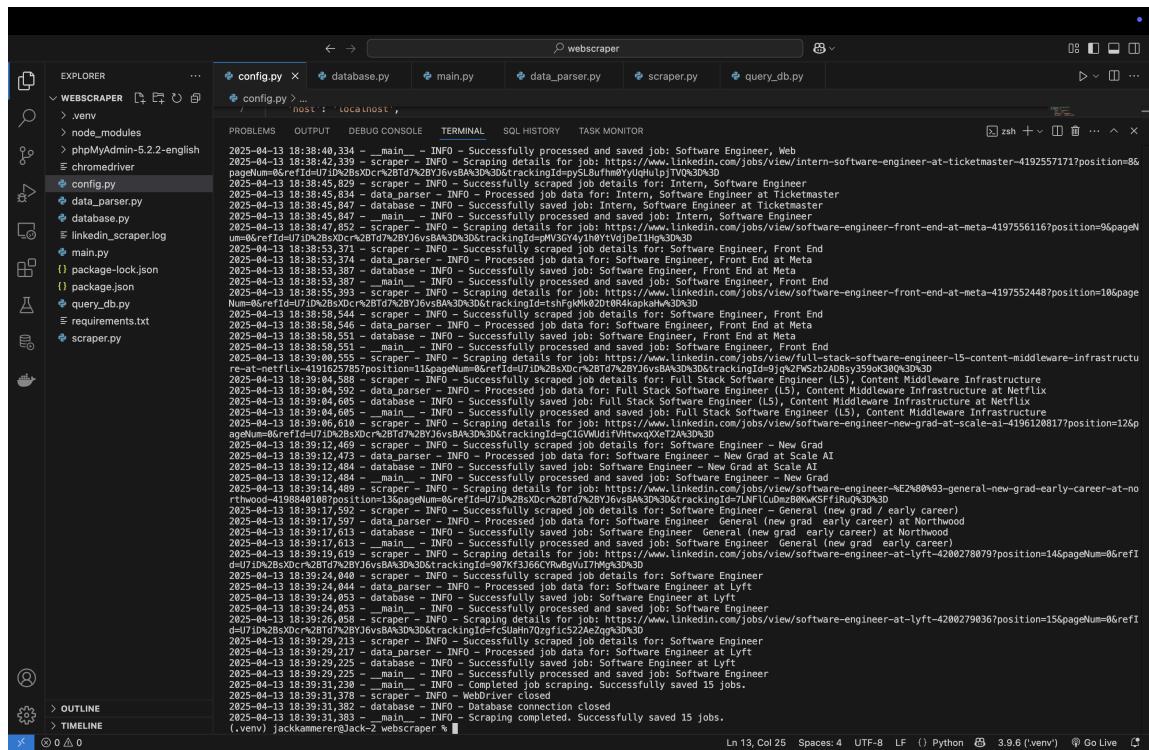
Additional Improvements:

- Fine-tune the parsing and scraping algorithms for better data consistency and accuracy
- Focus on UI/UX improvements to better showcase the scraped data and query results
- Consider adding pagination to enable scraping larger sets of job postings
- Optionally develop a Flask web interface for interactive data presentation
- Explore data export functionality for personal use in job applications

7 Conclusion

This milestone marks the most significant technical breakthrough in my project. After countless hours of trial and error, system restructures, and deep debugging, I successfully stabilized the backend functionality. Throughout the process, I experimented with multiple AI models, including smaller local LLMs and cloud-based tools, to assist with parsing and data manipulation. Despite initial setbacks, I realized that fully hardcoding my parsing algorithms and carefully mapping the data flow was the most reliable solution. I also navigated complex software dependencies, shifting between environments, and fine-tuned my schema design to meet full third normal form (3NF) standards. The database is now fully normalized and integrated with my DBMS (phpMyAdmin), providing clean, maintainable

data storage. This work required rethinking my initial design multiple times, significantly expanding from a simple two-entity system to a robust five-entity structure. With the core system now operational, I am positioned to finalize my SQL queries, capture reporting screenshots, and complete the final documentation for submission.



```

WEBSCRAPER ... config.py x database.py main.py data_parser.py scraper.py query_db.py
... > venv
... > node_modules
... > phpMyAdmin-5.2.2-english
... chromedriver
... config.py
... config.py > ...
...     host: 'localhost'
EXPLORER
PROBLEMS DEBUG CONSOLE TERMINAL SQL HISTORY TASK MONITOR
2025-04-13 18:38:42,339 - __main__ - INFO - Successfully processed and saved job: Software Engineer, Web pageNum=0&refId=U71D92B8xOcr%2BtD%2BYj6vSBa%3D&trackingId=pW30D30
2025-04-13 18:38:45,929 - __main__ - INFO - Successfully scraped job details for Intern, Software Engineer
2025-04-13 18:38:45,934 - data_parser - INFO - Processed job data for: Intern, Software Engineer at Ticketmaster
2025-04-13 18:38:45,947 - database - INFO - Successfully saved job: Intern, Software Engineer at Ticketmaster
2025-04-13 18:38:45,947 - __main__ - INFO - Successfully processed and saved job: Intern, Software Engineer
2025-04-13 18:38:45,951 - scraper - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/software-engineer-front-end-at-meta-4197556116?position=9&pageN
um=0&refId=U71D92B8xOcr%2BtD%2BYj6vSBa%3D&trackingId=pW30D4yh1YYVgJdeIIhg930n30
2025-04-13 18:38:53,371 - __main__ - INFO - Successfully scraped job details for: Software Engineer, Front End
2025-04-13 18:38:53,374 - data_parser - INFO - Processed job data for: Software Engineer, Front End at Meta
2025-04-13 18:38:53,387 - __main__ - INFO - Successfully processed and saved job: Software Engineer, Front End
2025-04-13 18:38:53,393 - data_parser - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/software-engineer-front-end-at-meta-4197552448?position=10&page
Num=0&refId=U71D92B8xOcr%2BtD%2BYj6vSBa%3D&trackingId=pW30D4kpkAhw30%30
2025-04-13 18:38:55,344 - __main__ - INFO - Successfully scraped job details for: Software Engineer, Front End
2025-04-13 18:38:55,348 - data_parser - INFO - Processed job data for: Software Engineer, Front End at Meta
2025-04-13 18:38:55,351 - database - INFO - Successfully saved job: Software Engineer, Front End at Meta
2025-04-13 18:38:55,351 - __main__ - INFO - Successfully processed and saved job: Software Engineer, Front End
2025-04-13 18:38:55,355 - data_parser - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/full-stack-software-engineer-l5-content-middleware-infrastruc
ture-new-grad-at-scale-a1-41962128817?position=125&pageNum=0&refId=U71D92B8xOcr%2BtD%2BYj6vSBa%3D&trackingId=pW30D30
2025-04-13 18:39:04,588 - __main__ - INFO - Successfully scraped job details for: Full Stack Software Engineer (L5), Content Middleware Infrastructure at Netflix
2025-04-13 18:39:04,605 - database - INFO - Successfully saved job: Full Stack Software Engineer (L5), Content Middleware Infrastructure at Netflix
2025-04-13 18:39:04,605 - __main__ - INFO - Successfully processed and saved job: Full Stack Software Engineer (L5), Content Middleware Infrastructure at Netflix
2025-04-13 18:39:06,610 - scraper - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/software-engineer-new-grad-at-scale-a1-41962128817?position=125&
pageNum=0&refId=U71D92B8xOcr%2BtD%2BYj6vSBa%3D&trackingId=pW30D30
2025-04-13 18:39:12,469 - __main__ - INFO - Successfully scraped job details for: Software Engineer - New Grad at Scale AI
2025-04-13 18:39:12,473 - data_parser - INFO - Processed job data for: Software Engineer - New Grad at Scale AI
2025-04-13 18:39:12,473 - database - INFO - Successfully saved job: Software Engineer - New Grad at Scale AI
2025-04-13 18:39:12,484 - __main__ - INFO - Successfully processed and saved job: Software Engineer - New Grad
2025-04-13 18:39:14,489 - __main__ - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/software-engineer-new-grad-at-scale-a1-41962128817?position=125&page
Num=0&refId=U71D92B8xOcr%2BtD%2BYj6vSBa%3D&trackingId=pW30D30
2025-04-13 18:39:24,040 - __main__ - INFO - Successfully scraped job details for: Software Engineer
2025-04-13 18:39:24,044 - data_parser - INFO - Processed job data for: Software Engineer at Lyft
2025-04-13 18:39:24,051 - database - INFO - Successfully saved job: Software Engineer at Lyft
2025-04-13 18:39:26,058 - scraper - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/software-engineer-at-lyft-4200278079?position=15&pageNum=0&refI
d=dU71D92B8xOcr%2BtD%2BYj6vSBa%3D&trackingId=fCSUahm7zqpfic52zAeZqg930%30
2025-04-13 18:39:29,213 - __main__ - INFO - Successfully scraped job details for: Software Engineer
2025-04-13 18:39:29,213 - data_parser - INFO - Processed job data for: Software Engineer at Northwood
2025-04-13 18:39:29,213 - database - INFO - Successfully saved job: Software Engineer General (new grad early career) at Northwood
2025-04-13 18:39:17,613 - __main__ - INFO - Successfully processed and saved job: Software Engineer General (new grad early career)
2025-04-13 18:39:19,619 - scraper - INFO - Scraping details for job: https://www.linkedin.com/jobs/view/software-engineer-at-lyft-4200278079?position=14&pageNum=0&refI
d=dU71D92B8xOcr%2BtD%2BYj6vSBa%3D&trackingId=fCSUahm7zqpfic52zAeZqg930%30
2025-04-13 18:39:31,225 - __main__ - INFO - Completed job scraping. Successfully saved 15 jobs.
2025-04-13 18:39:31,238 - __main__ - INFO - Completed job scraping. Successfully saved 15 jobs.
2025-04-13 18:39:31,378 - scraper - INFO - WebDriver closed.
2025-04-13 18:39:31,383 - __main__ - INFO - Scraping completed. Successfully saved 15 jobs.
(...,venv) jackhammerer@Jack-2 webscraper % █

```

Ln 13, Col 25 Spaces: 4 UTF-8 LF () Python ⌂ 3.9.6 ('venv') ⌂ Go Live ⌂

Figure 15: Finished Execution