
JOB AGGREGATOR LITE

PROJECT 9

Jack Kammerer

CS.4332.252 Introduction to Database Systems

Texas State University

April 2025

Contents

1	Introduction	1
2	Updated E/R Diagram	3
3	English Queries and SQL Implementations	4
3.1	Query 1: Tech Job Overview	4
3.2	Query 2: Web Development Focus	5
3.3	Query 3: Software Engineering Roles	6
3.4	Query 4: IT Position Search (Adjusted)	7
3.5	Query 5: Front-End vs. Back-End Focus	8
3.6	Query 6: Full-Stack Opportunities	9
3.7	Query 7: Tech Skills Filter	10
3.8	Query 8: Seniority Filter	11
3.9	Query 9: Remote Work Focus (Adjusted)	12
3.10	Query 10: Category Summary Count	13
4	Backend Accomplishments	15
5	Source Code	15
5.1	Main Python Files	15
5.2	main.py	15
5.3	scraper.py	18
5.4	database.py	23
5.5	config.py	38
6	Conclusion	39

1 Introduction

In Project 9, I completed a full redesign of the data-persistence layer in my LinkedIn Job Scraper, migrating from MySQL to an Oracle database while preserving a fully normalized (3NF) schema. This latest phase challenged me to:

- Containerize Oracle using Docker, configure user-defined schemas, sequences, and triggers to support auto-incrementing keys.
- Refactor all database access code in Python to use `oracledb`, including robust error handling, connection pooling, and transactional commits.
- Retain the five-entity architecture (Companies, Locations, Skills, Jobs, Job_Skills) established in Project 8, ensuring that each non-key attribute depends solely on its primary key.
- Enhance parsing logic in `data_parser.py` to default missing countries to “USA,” correctly split city/state/country values, and eliminate spurious NULLs in the Locations table.
- Integrate end-to-end scraping, parsing, and persistence: the scraper retrieves LinkedIn listings, the parser normalizes raw strings, and the database module inserts or deduplicates each record in real time.

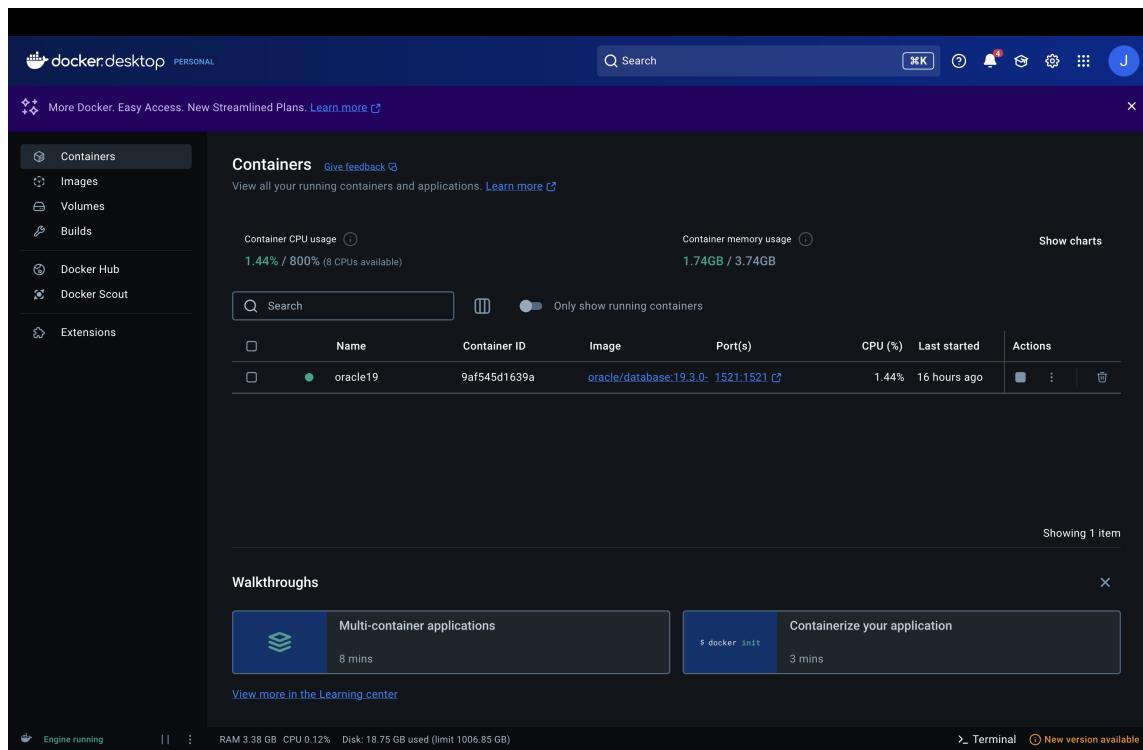


Figure 1: Oracle running in Docker

```

    (...venv) jackkammerer@Mac webscraper % /Users/jackkammerer/webscraper/.venv/bin/python /Users/jackkammerer/webscraper/main.py
    2025-04-28 14:02:07,922 - INFO - Starting job scraping with URL: https://www.linkedin.com/jobs/search/?keywords=software%20engineer&location=United%20States
    2025-04-28 14:02:08,000 - INFO - Starting Oracle job scraper
    2025-04-28 14:02:08,009 - database - INFO - Connected to Oracle Database
    2025-04-28 14:02:08,009 - database - INFO - Resetting database...
    2025-04-28 14:02:08,095 - database - INFO - Executing SQL:
    2025-04-28 14:02:08,101 - database - INFO - Executing SQL:
    BEGIN EXECUTE IMMEDIATE :DROP SEQUENCE company_seq; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -2289 THEN RAISE; END IF; END;
    2025-04-28 14:02:08,104 - database - INFO - Executing SQL:
    CREATE SEQUENCE company_seq START WITH 1 INCREMENT BY 1 NOCACHE NOLOG;
    BEGIN EXECUTE IMMEDIATE :DROP TRIGGER trg_company_id; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -4080 THEN RAISE; END IF; END;
    2025-04-28 14:02:08,107 - database - INFO - Executing SQL:
    CREATE OR REPLACE TRIGGER trg_company_id BEFORE INSERT ON companies FOR EACH ROW BEGIN :NEW.company_id := company_seq.NEXTVAL; END;
    2025-04-28 14:02:08,113 - database - INFO - Executing SQL:
    CREATE TABLE locations ( location_id NUMBER PRIMARY KEY, city VARCHAR2(100), state VARCHAR2(100), country VARCHAR2(100), UNIQUE (city, state, country))
    2025-04-28 14:02:08,116 - database - INFO - Executing SQL:
    BEGIN EXECUTE IMMEDIATE :DROP SEQUENCE location_seq; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -2289 THEN RAISE; END IF; END;
    2025-04-28 14:02:08,118 - database - INFO - Executing SQL:
    2025-04-28 14:02:08,120 - database - INFO - Executing SQL:
    BEGIN EXECUTE IMMEDIATE :DROP SEQUENCE location_id; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -4080 THEN RAISE; END IF; END;
    2025-04-28 14:02:08,121 - database - INFO - Executing SQL:
    CREATE TABLE skills ( skill_id NUMBER PRIMARY KEY, skill_name VARCHAR2(100) NOT NULL, UNIQUE (skill_name))
    2025-04-28 14:02:08,124 - database - INFO - Executing SQL:
    BEGIN EXECUTE IMMEDIATE :DROP SEQUENCE skill_id; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -2289 THEN RAISE; END IF; END;
    2025-04-28 14:02:08,128 - database - INFO - Executing SQL:
    BEGIN EXECUTE IMMEDIATE :DROP SEQUENCE skill_name; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -4080 THEN RAISE; END IF; END;
    2025-04-28 14:02:08,130 - database - INFO - Executing SQL:
    CREATE SEQUENCE skill_id_seq START WITH 1 INCREMENT BY 1 NOCACHE NOLOG;
    BEGIN EXECUTE IMMEDIATE :DROP TRIGGER trg_skill_id; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -4080 THEN RAISE; END IF; END;
    2025-04-28 14:02:08,131 - database - INFO - Executing SQL:
    CREATE OR REPLACE TRIGGER trg_skill_id BEFORE INSERT ON skills FOR EACH ROW BEGIN :NEW.skill_id := skill_id_seq.NEXTVAL; END;
    2025-04-28 14:02:08,135 - database - INFO - Executing SQL:
    CREATE TABLE jobs ( job_id NUMBER PRIMARY KEY, title VARCHAR2(255) NOT NULL, company_id NUMBER, location_id NUMBER, description CLOB, post_date DATE, FOREIGN KEY (company_id) REFERENCES companies (company_id), FOREIGN KEY (location_id) REFERENCES locations (location_id))
    2025-04-28 14:02:08,142 - database - INFO - Executing SQL:
    BEGIN EXECUTE IMMEDIATE :DROP SEQUENCE job_seq; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -2289 THEN RAISE; END IF; END;
    2025-04-28 14:02:08,144 - database - INFO - Executing SQL:
    CREATE SEQUENCE job_seq START WITH 1 INCREMENT BY 1 NOCACHE NOLOG;
    BEGIN EXECUTE IMMEDIATE :DROP TRIGGER trg_job_id; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -4080 THEN RAISE; END IF; END;
    2025-04-28 14:02:08,146 - database - INFO - Executing SQL:
    CREATE OR REPLACE TRIGGER trg_job_id BEFORE INSERT ON jobs FOR EACH ROW BEGIN :NEW.job_id := job_seq.NEXTVAL; END;
    2025-04-28 14:02:08,150 - database - INFO - Executing SQL:
    CREATE TABLE job_skills ( job_id NUMBER, skill_id NUMBER, PRIMARY KEY (job_id, skill_id), FOREIGN KEY (job_id) REFERENCES jobs (job_id) ON DELETE CASCADE, FOREIGN KEY (skill_id) REFERENCES skills (skill_id) ON DELETE CASCADE)
    2025-04-28 14:02:08,154 - database - INFO - Database tables created successfully
    2025-04-28 14:02:08,154 - database - INFO - Database reset and tables recreated successfully.
    2025-04-28 14:02:08,193 - database - INFO - Resetting database...
    
```

Figure 2: Starting the Program

2 Updated E/R Diagram

With the migration to Oracle in Project 9, the diagram still comprises:

- **Jobs:** stores job title, description (CLOB), post date, and foreign keys to Companies and Locations
- **Companies:** stores company name, industry, and size, with an auto-incrementing company_id via Oracle sequences and triggers
- **Locations:** stores city, state, and country, enforcing uniqueness and defaulting missing countries to “USA”
- **Skills:** stores unique skill names, with auto-incrementing skill_id
- **Job_Skills:** a many-to-many join table linking Jobs and Skills, with cascading deletes

All non-key attributes depend solely on their primary key (3NF), and referential integrity is enforced with Oracle foreign-key constraints.

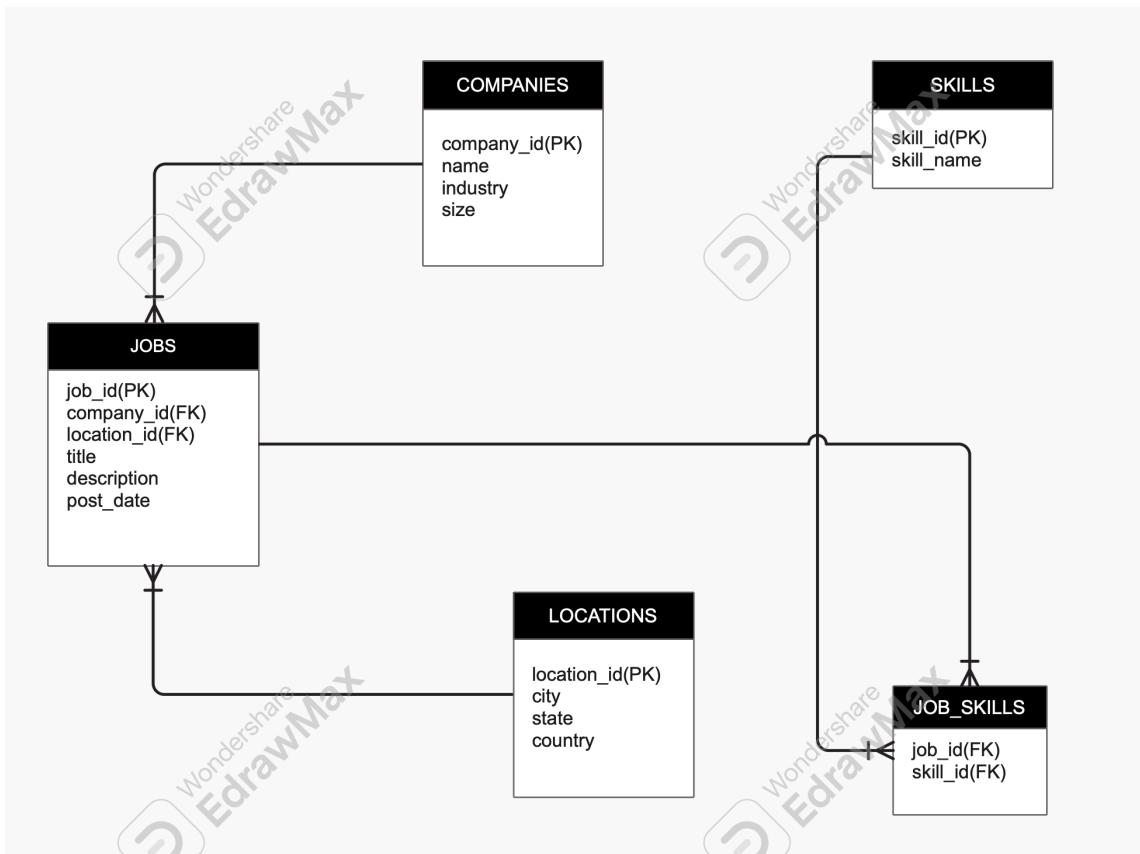


Figure 3: New E/R Diagram - 3NF

3 English Queries and SQL Implementations

This section presents a series of SQL queries designed to retrieve and summarize job posting data scraped from LinkedIn. Each query targets specific criteria such as role category, tech stack, or remote eligibility.

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.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:

The screenshot shows a SQL developer interface with a central 'QUERY RESULT' tab. The results of the query are displayed in a table with the following columns: TITLE, COMPANY_NAME, CITY, STATE, COUNTRY, and POST_DATE. The data consists of 15 rows of tech job postings from various companies like Slack, Netflix, and Google, located in cities like Seattle, New York, San Francisco, and Los Angeles, with post dates ranging from 06/04 to 17/04.

	TITLE	COMPANY_NAME	CITY	STATE	COUNTRY	POST_DATE
1	Software Engineer, Machine Learning (Multiple Levels) - Slack	Slack	Seattle	WA	USA	17/04
2	Software Engineer (L4) - Consumer Engineering	Netflix	Unknown	Unknown	USA	17/04
3	Software Engineer I	Spark	New York	NY	USA	15/04
4	Software Engineer, Web	Calm	San Francisco	CA	USA	13/04
5	Software Engineer	Roku	San Jose	CA	USA	13/04
6	Systems Software Engineer L4	Netflix	Unknown	Unknown	USA	13/04
7	Software Engineer - Web	Plaid	Unknown	Unknown	USA	13/04
8	Software Engineer 5 - Web Discovery	Netflix	Unknown	Unknown	USA	13/04
9	Software Engineer (L4) - Cloud Network Engineering	Netflix	Unknown	Unknown	USA	13/04
10	Software Engineer	Meta	San Francisco Bay Area	Unknown	USA	13/04
11	Software Engineer	Meta	San Francisco Bay Area	Unknown	USA	13/04
12	Software Engineer	Twitch	Seattle	WA	USA	13/04
13	Software Engineer General (new grad early career)	Northwood	Los Angeles	CA	USA	06/04
14	Software Engineer 4 - TV Web Player Platform	Netflix	Unknown	Unknown	USA	06/04
15	Software Engineer, JAX External	Google	San Francisco	CA	USA	06/04

Figure 4: Query 1: Display of all tech job postings with company, location, and post date

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.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 LOWER(j.title) LIKE '%web development%' OR LOWER(j.description) LIKE '%web deve
ORDER BY j.post_date DESC;
```

Screenshot:

The screenshot shows the SQL Developer application interface. On the left is a tree view of database objects under a connection named 'webscraper'. The 'JOBS' node is expanded, showing various job categories like Tables, Views, Indexes, etc. In the center, the 'QUERY RESULT' tab is selected, displaying a single row of data from a query. The data is presented in a table with columns: TITLE, COMPANY_NAME, CITY, STATE, COUNTRY, and POST_DATE. The row shows:

	TITLE	COMPANY_NAME	CITY	STATE	COUNTRY	POST_DATE
1	Software Engineer I	Spark	New York	NY	USA	15/04/25

At the bottom of the interface, there are several status indicators and toolbars.

Figure 5: Query 2: Jobs with a focus on web development

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.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 LOWER(j.title) LIKE '%software engineering%'
ORDER BY j.post_date DESC;
    
```

Screenshot:

The screenshot shows a SQL developer interface with a sidebar containing various database connection and management tools. The main area displays a query result titled "webscraper" with 15 rows fetched in 0.027 seconds. The columns are TITLE, COMPANY_NAME, CITY, STATE, COUNTRY, and POST_DATE. The data includes entries from companies like Slack, Netflix, Spark, Meta, Calm, Twitch, and Google, located in cities like Seattle, New York, San Francisco, and San Jose, across different states and countries, with post dates ranging from 13/04 to 17/04.

	TITLE	COMPANY_NAME	CITY	STATE	COUNTRY	POST_DATE
1	Software Engineer, Machine Learning (Multiple Levels) - Slack	Slack	Seattle	WA	USA	17/04
2	Software Engineer (L4) – Consumer Engineering	Netflix	Unknown	Unknown	USA	17/04
3	Software Engineer I	Spark	New York	NY	USA	15/04
4	Software Engineer	Meta	San Francisco Bay Area	Unknown	USA	13/04
5	Software Engineer, Web	Calm	San Francisco	CA	USA	13/04
6	Software Engineer	Meta	San Francisco Bay Area	Unknown	USA	13/04
7	Software Engineer 5 – Web Discovery	Netflix	Unknown	Unknown	USA	13/04
8	Software Engineer – Web	Plaid	Unknown	Unknown	USA	13/04
9	Software Engineer	Twitch	Seattle	WA	USA	13/04
10	Software Engineer (L4) – Cloud Network Engineering	Netflix	Unknown	Unknown	USA	13/04
11	Systems Software Engineer L4	Netflix	Unknown	Unknown	USA	13/04
12	Software Engineer	Roku	San Jose	CA	USA	13/04
13	Software Engineer, JAX External	Google	San Francisco	CA	USA	06/04
14	Software Engineer General (new grad early career)	Northwood	Los Angeles	CA	USA	06/04
15	Software Engineer 4 – TV Web Player Platform	Netflix	Unknown	Unknown	USA	06/04

Figure 6: Query 3: Software engineering job postings

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.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 LOWER(j.title) LIKE '%it%' OR LOWER(j.description) LIKE '%it%'
ORDER BY j.post_date DESC;
    
```

Screenshot:

The screenshot shows the SQL Developer application interface. On the left is a sidebar with icons for various database objects like Tables, Views, Procedures, etc. The main area is titled 'webscraper' and contains a table with the following data:

TITLE	COMPANY_NAME	CITY	STATE	COUNTRY	POST_DATE
Software Engineer - Frontend	ABC Company	San Francisco	CA	USA	2023-10-01
Software Engineer - Backend	XYZ Corp	New York	NY	USA	2023-10-02
System Administrator	Global Solutions	London	UK	UK	2023-10-03
Network Administrator	Global Solutions	London	UK	UK	2023-10-04
Cloud Architect	Global Solutions	Singapore	Singapore	Singapore	2023-10-05
Cloud Architect	Global Solutions	Singapore	Singapore	Singapore	2023-10-06
Machine Learning Engineer	Global Solutions	Singapore	Singapore	Singapore	2023-10-07
Machine Learning Engineer	Global Solutions	Singapore	Singapore	Singapore	2023-10-08
Machine Learning Engineer	Global Solutions	Singapore	Singapore	Singapore	2023-10-09
Machine Learning Engineer	Global Solutions	Singapore	Singapore	Singapore	2023-10-10

At the bottom, there are status indicators for Sign in to Jira, Sign in to Bitbucket, and other system metrics.

Figure 7: Query 4: IT-related job listings

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.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 LOWER(j.title) LIKE '%front end%' OR LOWER(j.title) LIKE '%back end%'
ORDER BY j.post_date DESC;
```

Screenshot:

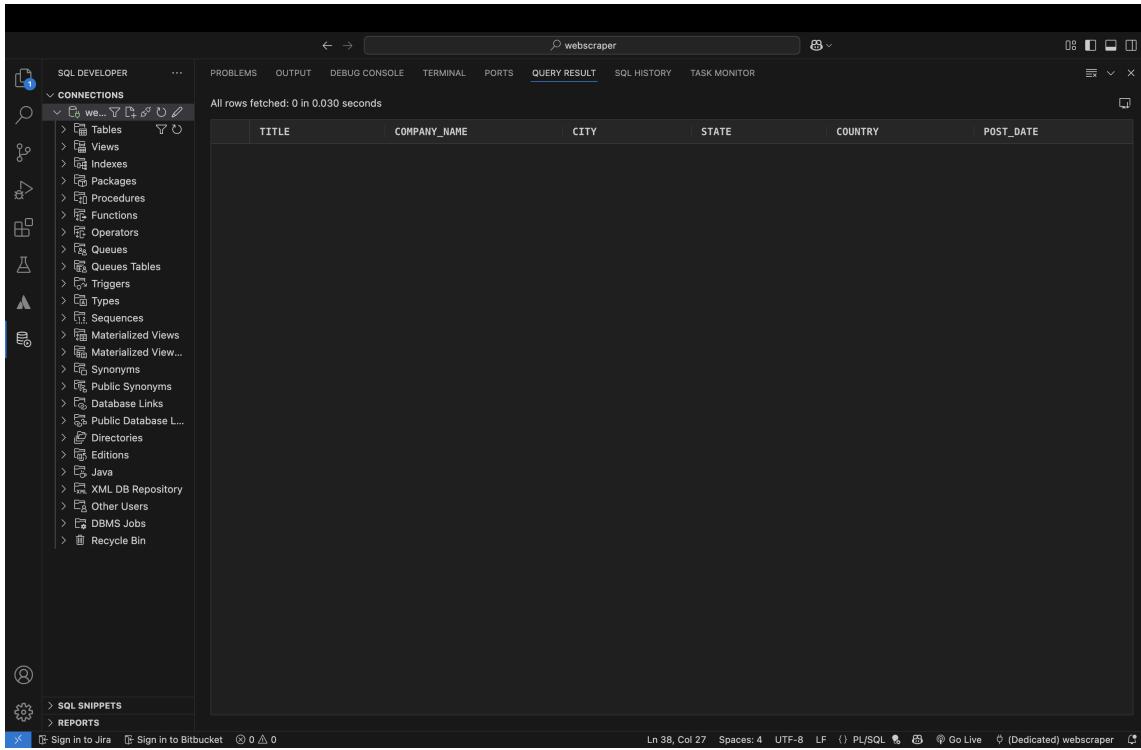


Figure 8: Query 5: Front-end and back-end related postings

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, location, and post date.

SQL:

```
SELECT j.title, c.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 LOWER(j.title) LIKE '%full-stack%' OR LOWER(j.description) LIKE '%full-stack%'
ORDER BY j.post_date DESC;
```

Screenshot:

The screenshot shows a SQL developer interface with a dark theme. On the left is a sidebar with various database objects like Tables, Views, Indexes, Procedures, Functions, Operators, Queues, Triggers, Types, Sequences, Materialized Views, Synonyms, Public Synonyms, Database Links, Directories, Editions, Java, XML DB Repository, Other Users, DBMS Jobs, and Recycle Bin. The main area has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, QUERY RESULT (which is selected), SQL HISTORY, and TASK MONITOR. The QUERY RESULT tab displays a table with the following data:

	TITLE	COMPANY_NAME	CITY	STATE	COUNTRY	POST_DATE
1	Software Engineer	Meta	San Francisco Bay Area	Unknown	USA	13/04/25
2	Software Engineer	Meta	San Francisco Bay Area	Unknown	USA	13/04/25
3	Software Engineer General (new grad early career)	Northwood	Los Angeles	CA	USA	06/04/25
4	Software Engineer, JAX External	Google	San Francisco	CA	USA	06/04/25

At the bottom, there are status indicators: Ln 44, Col 50, Spaces: 4, UTF-8, LF, PL/SQL, Go Live, (Dedicated) webscraper, and a connection status bar.

Figure 9: Query 6: Full-stack job opportunities

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.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 LOWER(s1.skill_name) = 'react' AND LOWER(s2.skill_name) = 'node.js'
ORDER BY j.post_date DESC;
    
```

The screenshot shows the DBeaver SQL Developer interface. On the left is a sidebar with various database connection options like Tables, Views, Indexes, Procedures, Functions, Operators, Queues, Triggers, Types, Sequences, Materialized Views, Materialized View Dependencies, Synonyms, Public Synonyms, Database Links, Public Database Links, Directories, Editions, Java, XML DB Repository, Other Users, DBMS Jobs, and Recycle Bin. Below this are sections for SQL Snippets and Reports. The main area has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, QUERY RESULT, SQL HISTORY, and TASK MONITOR. The QUERY RESULT tab is active, displaying a single row of data from a query named 'webscraper'. The results table has columns TITLE and COMPANY_NAME. The row shows 'Software Engineer 5 - Web Discovery' under TITLE and 'Netflix' under COMPANY_NAME. At the bottom of the interface, there are status indicators for Sign in to Jira, Sign in to Bitbucket, and other system metrics like Ln 57, Col 32, Spaces: 4, UTF-8, LF, PL/SQL, Go Live, and Dedicated webscraper.

TITLE	COMPANY_NAME
Software Engineer 5 - Web Discovery	Netflix

Figure 10: Query 7: Jobs requiring both React and Node.js

Screenshot:

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.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 LOWER(j.title) LIKE '%senior%' OR LOWER(j.title) LIKE '%lead%'
ORDER BY j.post_date DESC;
```

Screenshot:

TITLE	COMPANY_NAME	CITY	STATE	COUNTRY	POST_DATE
Senior Software Engineer	TechCorp	San Francisco	CA	USA	2023-09-15

Figure 11: Query 8: Senior and Lead positions

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.company_name, j.post_date
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
WHERE LOWER(j.description) LIKE '%remote%' OR LOWER(j.description) LIKE '%telecommute%'
ORDER BY j.post_date DESC;
```

Screenshot:

The screenshot shows a SQL developer interface with a dark theme. On the left is a sidebar with icons for connections, tables, views, indexes, packages, procedures, functions, operators, queues, triggers, types, sequences, materialized views, synonyms, public synonyms, database links, public database links, directories, editions, Java, XML DB repository, other users, DBMS jobs, and recycle bin. Below this is a section for SQL snippets and reports. The main area has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, QUERY RESULT (which is selected), SQL HISTORY, and TASK MONITOR. The QUERY RESULT tab displays a table with the following data:

	TITLE	COMPANY_NAME	POST_DATE
1	Software Engineer I	Spark	15/04/25
2	Software Engineer, Web	Calm	13/04/25
3	Software Engineer 5 - Web Discovery	Netflix	13/04/25

At the bottom, there are status indicators: Ln 72, Col 27, Spaces: 4, UTF-8, LF, PL/SQL, Go Live, and (Dedicated) webscraper.

Figure 12: Query 9: Remote or telecommuting job listings

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

```

```

COUNT(*) AS job_count
FROM jobs j
GROUP BY
CASE
    WHEN LOWER(j.title) LIKE '%web development%' OR LOWER(j.description) LIKE '%web d'
    WHEN LOWER(j.title) LIKE '%software engineering%' OR LOWER(j.description) LIKE '%sof'
    WHEN LOWER(j.title) LIKE '%it%' OR LOWER(j.description) LIKE '%it%' THEN 'IT'
    WHEN LOWER(j.title) LIKE '%front end%' OR LOWER(j.description) LIKE '%front end%'
    WHEN LOWER(j.title) LIKE '%back end%' OR LOWER(j.description) LIKE '%back end%' T
    WHEN LOWER(j.title) LIKE '%full-stack%' OR LOWER(j.description) LIKE '%full-stack'
    ELSE 'Other'
END;

```

Screenshot:

CATEGORY	JOB_COUNT
Web Development	1
IT	12
Software Engineering	2

Figure 13: Query 10: Summary count of jobs by category

4 Backend Accomplishments

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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL HISTORY TASK MONITOR
2025-04-20 14:02:08,154 - database - INFO - Resetting database...
2025-04-20 14:02:08,193 - database - INFO - Executing SQL:
CREATE TABLE companies (company_id NUMBER PRIMARY KEY, company_name VARCHAR2(255) NOT NULL, industry VARCHAR2(255), company_size VARCHAR2(100))
2025-04-20 14:02:08,194 - database - INFO - Executing SQL:
BEGIN EXECUTE IMMEDIATE 'DROP SEQUENCE company_seq'; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -2289 THEN ROLLBACK; END IF; END;
CREATE SEQUENCE company_seq START WITH 1 INCREMENT BY 1;
2025-04-20 14:02:08,198 - database - INFO - Executing SQL:
BEGIN EXECUTE IMMEDIATE 'CREATE TRIGGER trg_company_id BEFORE INSERT ON companies FOR EACH ROW BEGIN :NEW.company_id := company_seq.NEXTVAL; END';
2025-04-20 14:02:08,199 - database - INFO - Executing SQL:
CREATE OR REPLACE TRIGGER trg_company_id BEFORE INSERT ON companies FOR EACH ROW BEGIN :NEW.company_id := company_seq.NEXTVAL; END;
2025-04-20 14:02:08,202 - database - INFO - Executing SQL:
CREATE TABLE locations (location_id NUMBER PRIMARY KEY, city VARCHAR2(100), state VARCHAR2(100), UNIQUE (city, state, country))
2025-04-20 14:02:08,203 - database - INFO - Executing SQL:
BEGIN EXECUTE IMMEDIATE 'DROP SEQUENCE location_seq'; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -2289 THEN ROLLBACK; END IF; END;
2025-04-20 14:02:08,208 - database - INFO - Executing SQL:
BEGIN EXECUTE IMMEDIATE 'CREATE TRIGGER trg_location_id BEFORE INSERT ON locations FOR EACH ROW BEGIN :NEW.location_id := location_seq.NEXTVAL; END';
2025-04-20 14:02:08,209 - database - INFO - Executing SQL:
CREATE OR REPLACE TRIGGER trg_location_id BEFORE INSERT ON locations FOR EACH ROW BEGIN :NEW.location_id := location_seq.NEXTVAL; END;
2025-04-20 14:02:08,209 - database - INFO - Executing SQL:
CREATE TABLE skills (skill_id NUMBER PRIMARY KEY, skill_name VARCHAR2(100) NOT NULL, UNIQUE (skill_name))
2025-04-20 14:02:08,217 - database - INFO - Executing SQL:
BEGIN EXECUTE IMMEDIATE 'DROP SEQUENCE skill_seq'; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -2289 THEN ROLLBACK; END IF; END;
CREATE SEQUENCE skill_seq START WITH 1 INCREMENT BY 1;
2025-04-20 14:02:08,218 - database - INFO - Executing SQL:
BEGIN EXECUTE IMMEDIATE 'CREATE TRIGGER trg_skill_id BEFORE INSERT ON skills FOR EACH ROW BEGIN :NEW.skill_id := skill_seq.NEXTVAL; END';
2025-04-20 14:02:08,220 - database - INFO - Executing SQL:
CREATE OR REPLACE TRIGGER trg_skill_id BEFORE INSERT ON skills FOR EACH ROW BEGIN :NEW.skill_id := skill_seq.NEXTVAL; END;
2025-04-20 14:02:08,222 - database - INFO - Executing SQL:
CREATE TABLE jobs (job_id NUMBER PRIMARY KEY, title VARCHAR2(255) NOT NULL, company_id NUMBER, location_id NUMBER, description CLOB, post_date DATE, FOREIGN KEY (company_id) REFERENCES companies (company_id), FOREIGN KEY (location_id) REFERENCES locations (location_id))
2025-04-20 14:02:08,223 - database - INFO - Executing SQL:
BEGIN EXECUTE IMMEDIATE 'DROP SEQUENCE job_seq'; EXCEPTION WHEN OTHERS THEN IF SQLCODE != -2289 THEN ROLLBACK; END IF; END;
2025-04-20 14:02:08,229 - database - INFO - Executing SQL:
CREATE SEQUENCE job_seq START WITH 1 INCREMENT BY 1;
2025-04-20 14:02:08,230 - database - INFO - Executing SQL:
BEGIN EXECUTE IMMEDIATE 'CREATE TRIGGER trg_job_id BEFORE INSERT ON jobs FOR EACH ROW BEGIN :NEW.job_id := job_seq.NEXTVAL; END';
2025-04-20 14:02:08,238 - database - INFO - Executing SQL:
CREATE OR REPLACE TRIGGER trg_job_id BEFORE INSERT ON jobs FOR EACH ROW BEGIN :NEW.job_id := job_seq.NEXTVAL; END;
2025-04-20 14:02:08,233 - database - INFO - Executing SQL:
CREATE TABLE job_skills (job_id NUMBER, skill_id NUMBER, PRIMARY KEY (job_id, skill_id), FOREIGN KEY (job_id) REFERENCES jobs (job_id) ON DELETE CASCADE, FOREIGN KEY (skill_id) REFERENCES skills (skill_id) ON DELETE CASCADE)
2025-04-20 14:02:08,236 - database - INFO - Database tables created successfully
2025-04-20 14:02:08,237 - database - INFO - Database reset and tables recreated successfully.
2025-04-20 14:02:10,040 - scraper - INFO - WebDriver initialized successfully
2025-04-20 14:02:10,040 - scraper - INFO - LinkedIn scraper initialized
2025-04-20 14:02:10,040 - scraper - INFO - Starting to scrape job listings from: https://www.linkedin.com/jobs/search/?keywords=software%20engineer&location=United%20States
2025-04-20 14:02:24,752 - scraper - INFO - Found 15 job listings
2025-04-20 14:02:24,753 - __main__ - INFO - Found 15 job listings, proceeding to scrape details

```

Figure 14: Tables Created and Job Listings Found

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']}")
```

```

87         logger.info(f"Completed job scraping. Successfully saved {job_count}
88             } jobs.")
89
90     except Exception as e:
91         logger.error(f"Error in job scraping process: {e}")
92     return job_count
93
94 finally:
95     # Clean up resources
96     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}
111             jobs.")

```

5.3 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
23 from config import SCRAPING_CONFIG, USER_AGENT
24
25 # Configure logging
26 logging.basicConfig(
27     level=logging.INFO,
28     format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
29 )
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
41         self.db = db
42         self.setup_driver(headless)
43
44     def setup_driver(self, headless):
45         """
46             Set up the Selenium WebDriver.
47
48         Args:
49             headless (bool): Whether to run in headless mode
50         """
51
52         chrome_options = Options()
53         if headless:
54             chrome_options.add_argument("--headless")
55
56             chrome_options.add_argument("--no-sandbox")
57             chrome_options.add_argument("--disable-dev-shm-usage")
58             chrome_options.add_argument("--disable-gpu")
59             chrome_options.add_argument(f"user-agent={USER_AGENT}")

```

```

58
59         service = Service('./chromedriver')
60         self.driver = webdriver.Chrome(service=service, options=
61         chrome_options)
62
63     try:
64         self.driver.set_window_size(1920, 1080)
65         self.wait = WebDriverWait(self.driver, SCRAPING_CONFIG['timeout'])
66     except Exception as e:
67         logger.error(f"Failed to initialize WebDriver: {e}")
68         raise
69
70     def scrape_job_listings(self, url):
71         """
72             Scrape job listings from a LinkedIn search URL.
73
74         Args:
75             url (str): LinkedIn job search URL
76
77         Returns:
78             list: List of job listing URLs
79         """
80         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
95         # Extract job URLs
96         count = 0
97         for card in job_cards:
98             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    return []
116
117    except Exception as e:
118        logger.error(f"Error scraping job listings: {e}")
119        return []
120
121
122    def scrape_job_details(self, job_url):
123        """
124            Scrape detailed information for a specific job.
125
126            Args:
127                job_url (str): URL of the job listing
128
129            Returns:
130                dict: Job details including title, company, location, etc.
131
132        """
133
134        logger.info(f"Scraping details for job: {job_url}")
135        job_data = {}
136
137        try:
138            self.driver.get(job_url)
139            time.sleep(random.uniform(*SCRAPING_CONFIG['sleep_interval']))
140
141            job_data['title'] = self.safe_extract(By.CSS_SELECTOR, "h1.
142            topcard__title")
143
144
145

```

```

136     job_data['company_name'] = self.safe_extract(By.CSS_SELECTOR, "a.topcard__org-name-link")
137
138     # Try multiple CSS selectors for location
139     job_data['location'] = (
140         self.safe_extract(By.CSS_SELECTOR, "span.topcard__flavor.
topcard__flavor--bullet") or
141         self.safe_extract(By.CSS_SELECTOR, "span.job-details-jobs-
unified-top-card__bullet") or
142         self.safe_extract(By.CSS_SELECTOR, ".job-details-jobs-
unified-top-card__primary-description-container .job-details-jobs-
unified-top-card__bullet") or
143         "Unknown Location"
144     )
145
146     job_data['description'] = self.safe_extract(By.CSS_SELECTOR, "div.description__text", get_text=True)
147     job_data['post_date'] = self.safe_extract(By.CSS_SELECTOR, "span.posted-time-ago__text")
148
149     skills_elements = self.driver.find_elements(By.CSS_SELECTOR, ".skill-tag")
150     job_data['skills'] = [skill.text for skill in skills_elements
if skill.text]
151
152     job_data['company_size'] = self.safe_extract(By.CSS_SELECTOR, ".company-size")
153     job_data['industry'] = self.safe_extract(By.CSS_SELECTOR, ".company-industry")
154
155     logger.info(f"Successfully scraped job details for: {job_data.
get('title', 'Unknown job')}")
156     return job_data
157
158 except TimeoutException:
159     logger.error(f"Timeout while loading job details: {job_url}")
160     return {}
161 except Exception as e:
162     logger.error(f"Error scraping job details: {e}")
163     return {}
164
165 def safe_extract(self, by, selector, get_text=False):
166     """

```

```

167     Safely extract an element's text or attribute.
168
169     Args:
170         by: Selenium By locator
171         selector (str): CSS selector
172         get_text (bool): Whether to get innerText instead of
textContent
173
174     Returns:
175         str: Extracted text or None if element not found
176     """
177
178     try:
179         element = self.driver.find_element(by, selector)
180         if get_text:
181             return element.get_attribute("innerText")
182         return element.text
183     except (NoSuchElementException, StaleElementReferenceException):
184         return None
185
186     def close(self):
187         """Close the WebDriver."""
188         if self.driver:
189             self.driver.quit()
logger.info("WebDriver closed")

```

5.4 database.py

```

1 """
2 Database operations for the LinkedIn job scraper.
3 This module handles Oracle connections and database queries.
4 """
5
6 import oracledb
7 from oracledb import DatabaseError
8 import logging
9 from datetime import datetime
10
11 # Hardcoded Oracle config
12 ORACLE_CONFIG = {
13     "user": "system", # or admin, oracle, etc.
14     "password": "swagdata",
15     "dsn": oracledb.makedsn("localhost", 1521, service_name="ORCLPDB1")
16 }
17

```

```

18 # Configure logging
19 logging.basicConfig(
20     level=logging.INFO,
21     format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
22 )
23 logger = logging.getLogger(__name__)
24
25
26 class Database:
27     def __init__(self):
28         """Initialize database connection and create tables if they don't
29         exist."""
30         self.connection = None
31         try:
32             self.connection = oracledb.connect(**ORACLE_CONFIG)
33             logger.info("Connected to Oracle database")
34             self.reset_database()
35         except DatabaseError as e:
36             logger.error(f"Error connecting to Oracle: {e}")
37             raise
38
39     def query(self, sql, params=None):
40         """
41             Execute a query and return the results.
42
43             Args:
44                 sql (str): SQL query to execute
45                 params (tuple, optional): Parameters for the query
46
47             Returns:
48                 list: Query results as a list of tuples
49
50         """
51         cursor = self.connection.cursor()
52         try:
53             if params:
54                 cursor.execute(sql, params)
55             else:
56                 cursor.execute(sql)
57             return cursor.fetchall()
58         except DatabaseError as e:
59             logger.error(f"Error executing query: {e}")
60             return []
61
62         finally:

```

```

60         cursor.close()
61
62     def reset_database(self):
63         """
64             Drop all tables and recreate them to start fresh.
65         """
66
67         cursor = self.connection.cursor()
68
69         try:
70             logger.info("Resetting database...")
71
72             # Drop tables if they exist
73             tables = ['job_skills', 'jobs', 'skills', 'locations', 'companies']
74
75             for table in tables:
76                 cursor.execute(f"""
77                     BEGIN
78                         EXECUTE IMMEDIATE 'DROP TABLE {table}';
79                     EXCEPTION
80                         WHEN OTHERS THEN
81                             IF SQLCODE != -942 THEN
82                                 RAISE;
83                             END IF;
84                     END;
85                 """)
86
87             # Recreate the tables
88             self.create_tables()
89             logger.info("Database reset and tables recreated successfully.")
90         )
91
92     except DatabaseError as e:
93         logger.error(f"Error resetting database: {e}")
94         raise
95     finally:
96         cursor.close()
97
98     def create_tables(self):
99         """Create necessary tables if they don't exist."""
100        cursor = self.connection.cursor()
101
102        try:
103            # Create companies table
104            sql = (
105                "CREATE TABLE companies ("
```

```

101         " company_id NUMBER PRIMARY KEY , "
102         " company_name VARCHAR2(255) NOT NULL , "
103         " industry VARCHAR2(255) , "
104         " company_size VARCHAR2(100) "
105     ")"
106 )
107 logger.info(f"Executing SQL:{sql}")
108 cursor.execute(sql)
109
110 # Drop sequence if it exists
111 sql = (
112     "BEGIN "
113     " EXECUTE IMMEDIATE 'DROP SEQUENCE company_seq'; "
114     "EXCEPTION "
115     " WHEN OTHERS THEN "
116     " IF SQLCODE != -2289 THEN RAISE; END IF; "
117     "END ;"
118 )
119 logger.info(f"Executing SQL:{sql}")
120 cursor.execute(sql)
121
122 # Create sequence for company_id
123 sql = "CREATE SEQUENCE company_seq START WITH 1 INCREMENT BY 1"
124 logger.info(f"Executing SQL:{sql}")
125 cursor.execute(sql)
126
127 # Drop trigger if it exists
128 sql = (
129     "BEGIN "
130     " EXECUTE IMMEDIATE 'DROP TRIGGER trg_company_id'; "
131     "EXCEPTION "
132     " WHEN OTHERS THEN "
133     " IF SQLCODE != -4080 THEN RAISE; END IF; "
134     "END ;"
135 )
136 logger.info(f"Executing SQL:{sql}")
137 cursor.execute(sql)
138
139 # Create trigger for company_id auto-increment
140 sql = (
141     "CREATE OR REPLACE TRIGGER trg_company_id "
142     "BEFORE INSERT ON companies "
143     "FOR EACH ROW "

```

```

144         "BEGIN "
145             " :NEW.company_id := company_seq.NEXTVAL; "
146             "END ;"
147     )
148     logger.info(f"Executing SQL:{sql}")
149     cursor.execute(sql)
150
151     # Create locations table
152     sql = (
153         "CREATE TABLE locations ("
154             " location_id NUMBER PRIMARY KEY ,"
155             " city VARCHAR2(100) ,"
156             " state VARCHAR2(100) ,"
157             " country VARCHAR2(100) ,"
158             " UNIQUE (city, state, country)"
159             ")"
160     )
161     logger.info(f"Executing SQL:{sql}")
162     cursor.execute(sql)
163
164     # Drop sequence if it exists
165     sql = (
166         "BEGIN "
167             " EXECUTE IMMEDIATE 'DROP SEQUENCE location_seq'; "
168             "EXCEPTION "
169                 " WHEN OTHERS THEN "
170                     " IF SQLCODE != -2289 THEN RAISE; END IF; "
171                 "END ;"
172     )
173     logger.info(f"Executing SQL:{sql}")
174     cursor.execute(sql)
175
176     # Create sequence for location_id
177     sql = "CREATE SEQUENCE location_seq START WITH 1 INCREMENT BY 1
178
179     logger.info(f"Executing SQL:{sql}")
180     cursor.execute(sql)
181
182     # Drop trigger if it exists
183     sql = (
184         "BEGIN "
185             " EXECUTE IMMEDIATE 'DROP TRIGGER trg_location_id'; "
186             "EXCEPTION "

```

```

186         " WHEN OTHERS THEN "
187         " IF SQLCODE != -4080 THEN RAISE; END IF; "
188         "END ;"
189     )
190     logger.info(f"Executing SQL:{sql}")
191     cursor.execute(sql)
192
193     # Create trigger for location_id auto-increment
194     sql = (
195         "CREATE OR REPLACE TRIGGER trg_location_id "
196         "BEFORE INSERT ON locations "
197         "FOR EACH ROW "
198         "BEGIN "
199         " :NEW.location_id := location_seq.NEXTVAL; "
200         "END ;"
201     )
202     logger.info(f"Executing SQL:{sql}")
203     cursor.execute(sql)
204
205     # Create skills table
206     sql = (
207         "CREATE TABLE skills ("
208         " skill_id NUMBER PRIMARY KEY ,"
209         " skill_name VARCHAR2(100) NOT NULL ,"
210         " UNIQUE (skill_name)"
211         ")"
212     )
213     logger.info(f"Executing SQL:{sql}")
214     cursor.execute(sql)
215
216     # Drop sequence if it exists
217     sql = (
218         "BEGIN "
219         " EXECUTE IMMEDIATE 'DROP SEQUENCE skill_seq'; "
220         "EXCEPTION "
221         " WHEN OTHERS THEN "
222         " IF SQLCODE != -2289 THEN RAISE; END IF; "
223         "END ;"
224     )
225     logger.info(f"Executing SQL:{sql}")
226     cursor.execute(sql)
227
228     # Create sequence for skill_id

```

```

229         sql = "CREATE SEQUENCE skill_seq START WITH 1 INCREMENT BY 1"
230         logger.info(f"Executing SQL:{sql}")
231         cursor.execute(sql)
232
233         # Drop trigger if it exists
234         sql = (
235             "BEGIN "
236             " EXECUTE IMMEDIATE 'DROP TRIGGER trg_skill_id'; "
237             "EXCEPTION "
238             " WHEN OTHERS THEN "
239             " IF SQLCODE != -4080 THEN RAISE; END IF; "
240             "END ;"
241         )
242         logger.info(f"Executing SQL:{sql}")
243         cursor.execute(sql)
244
245         # Create trigger for skill_id auto-increment
246         sql = (
247             "CREATE OR REPLACE TRIGGER trg_skill_id "
248             "BEFORE INSERT ON skills "
249             "FOR EACH ROW "
250             "BEGIN "
251             " :NEW.skill_id := skill_seq.NEXTVAL; "
252             "END ;"
253         )
254         logger.info(f"Executing SQL:{sql}")
255         cursor.execute(sql)
256
257         # Create jobs table
258         sql = (
259             "CREATE TABLE jobs ("
260             " job_id NUMBER PRIMARY KEY ,"
261             " title VARCHAR2(255) NOT NULL ,"
262             " company_id NUMBER ,"
263             " location_id NUMBER ,"
264             " description CLOB ,"
265             " post_date DATE ,"
266             " FOREIGN KEY (company_id) REFERENCES companies (company_id"
267             "),"
268             " FOREIGN KEY (location_id) REFERENCES locations (
269             location_id)"
270             ")"
271         )

```

```

270     logger.info(f"Executing SQL:\n{sql}")
271     cursor.execute(sql)
272
273     # Drop sequence if it exists
274     sql = (
275         "BEGIN "
276         " EXECUTE IMMEDIATE 'DROP SEQUENCE job_seq'; "
277         "EXCEPTION "
278         " WHEN OTHERS THEN "
279         "   IF SQLCODE != -2289 THEN RAISE; END IF; "
280         "END ;"
281     )
282     logger.info(f"Executing SQL:\n{sql}")
283     cursor.execute(sql)
284
285     # Create sequence for job_id
286     sql = "CREATE SEQUENCE job_seq START WITH 1 INCREMENT BY 1"
287     logger.info(f"Executing SQL:\n{sql}")
288     cursor.execute(sql)
289
290     # Drop trigger if it exists
291     sql = (
292         "BEGIN "
293         " EXECUTE IMMEDIATE 'DROP TRIGGER trg_job_id'; "
294         "EXCEPTION "
295         " WHEN OTHERS THEN "
296         "   IF SQLCODE != -4080 THEN RAISE; END IF; "
297         "END ;"
298     )
299     logger.info(f"Executing SQL:\n{sql}")
300     cursor.execute(sql)
301
302     # Create trigger for job_id auto-increment
303     sql = (
304         "CREATE OR REPLACE TRIGGER trg_job_id "
305         "BEFORE INSERT ON jobs "
306         "FOR EACH ROW "
307         "BEGIN "
308         " :NEW.job_id := job_seq.NEXTVAL; "
309         "END ;"
310     )
311     logger.info(f"Executing SQL:\n{sql}")
312     cursor.execute(sql)

```

```

313
314     # Create job_skills join table
315     sql = (
316         "CREATE TABLE job_skills ("
317         " job_id NUMBER ,"
318         " skill_id NUMBER ,"
319         " PRIMARY KEY (job_id, skill_id),"
320         " FOREIGN KEY (job_id) REFERENCES jobs (job_id) ON DELETE
321         CASCADE ,"
322         " FOREIGN KEY (skill_id) REFERENCES skills (skill_id) ON
323         DELETE CASCADE"
324     ")"
325
326     logger.info(f"Executing SQL:{\n{sql}}")
327     cursor.execute(sql)
328
329     self.connection.commit()
330     logger.info("Database tables created successfully")
331 except DatabaseError as e:
332     logger.error(f"Error creating tables: {e}")
333     raise
334 finally:
335     cursor.close()
336
337 def insert_company(self, name, industry=None, company_size=None):
338     """
339     Insert a company or get its ID if it already exists.
340
341     Args:
342         name (str): Company name
343         industry (str, optional): Company industry
344         company_size (str, optional): Company size
345
346     Returns:
347         int: The company_id
348     """
349     cursor = self.connection.cursor()
350     try:
351         # Check if company already exists
352         cursor.execute('SELECT company_id FROM companies WHERE
353             company_name = :1', (name,))
354         result = cursor.fetchone()
355
356         if result:
357             return result[0]
358
359         # Insert new company
360         sql = "INSERT INTO companies (name, industry, company_size) VALUES (:1, :2, :3)"
361         cursor.execute(sql, (name, industry, company_size))
362         self.connection.commit()
363
364         # Get the inserted company ID
365         cursor.execute('SELECT company_id FROM companies WHERE
366             company_name = :1', (name,))
367         result = cursor.fetchone()
368
369         if result:
370             return result[0]
371
372     except Exception as e:
373         logger.error(f"Error inserting company: {e}")
374         raise
375
376     return None

```

```

353         if result:
354             return result[0]
355
356     # Insert new company
357     cursor.execute(
358         'INSERT INTO companies (company_name, industry,
359         company_size) VALUES (:1, :2, :3)',
360         (name, industry, company_size)
361     )
362     self.connection.commit()
363     cursor.execute('SELECT company_id FROM companies WHERE
364         company_name = :1', (name,))
365     result = cursor.fetchone()
366     if result:
367         return result[0]
368     else:
369         logger.warning("Insert company failed to return ID for: %s"
370         , name)
371         return None
372     except DatabaseError as e:
373         self.connection.rollback()
374         logger.error(f"Error inserting company: {e}")
375         raise
376     finally:
377         cursor.close()
378
379     def insert_location(self, city=None, state=None, country=None):
380         """
381         Insert a location or get its ID if it already exists.
382
383         Args:
384             city (str, optional): City name
385             state (str, optional): State name
386             country (str, optional): Country name
387
388         Returns:
389             int: The location_id
390         """
391
392         # If all location fields are None, use a default location
393         if city is None and state is None and country is None:
394             # Use a default location ("Unknown", "Unknown", "Unknown")
395             city = "Unknown"
396             state = "Unknown"

```

```

393         country = "Unknown"
394         logger.warning("No location data provided, using default: %s, %
395 s, %s", city, state, country)
396     elif country is None:
397         # Ensure we at least have a country if other fields are present
398         country = "Unknown"
399
400     # Convert None values to "Unknown" for consistent database storage
401     city = city if city is not None else "Unknown"
402     state = state if state is not None else "Unknown"
403     country = country if country is not None else "Unknown"
404
405     cursor = self.connection.cursor()
406
407     try:
408         # Check if location already exists
409         cursor.execute(
410             "SELECT location_id FROM locations WHERE city = :1 AND
411 state = :2 AND country = :3",
412             (city, state, country)
413         )
414         result = cursor.fetchone()
415
416         if result:
417             return result[0]
418
419         # Insert new location
420         cursor.execute(
421             "INSERT INTO locations (city, state, country) VALUES (:1,
422 :2, :3)",
423             (city, state, country)
424         )
425         self.connection.commit()
426         cursor.execute("SELECT location_id FROM locations WHERE city =
427 :1 AND state = :2 AND country = :3", (city, state, country))
428         result = cursor.fetchone()
429         if result:
430             return result[0]
431         else:
432             logger.warning("Insert location failed to return ID for: %s
433 , %s, %s", city, state, country)
434             # Return ID for default location as fallback
435             cursor.execute("SELECT location_id FROM locations WHERE
436 city = 'Unknown' AND state = 'Unknown' AND country = 'Unknown'")



```

```

430         default_result = cursor.fetchone()
431         if default_result:
432             return default_result[0]
433             # If no default exists, create it now
434             cursor.execute("INSERT INTO locations (city, state, country
435 ) VALUES ('Unknown', 'Unknown', 'Unknown')")
436             self.connection.commit()
437             return self.insert_location("Unknown", "Unknown", "Unknown"
438 )
439         except DatabaseError as e:
440             self.connection.rollback()
441             # Handle duplicate location gracefully by fetching existing ID
442             if 'ORA-00001' in str(e):
443                 cursor.execute(
444                     "SELECT location_id FROM locations WHERE city = :1 AND
445 state = :2 AND country = :3",
446                     (city, state, country)
447                 )
448                 result = cursor.fetchone()
449                 if result:
450                     return result[0]
451                     logger.error(f"Error inserting location: {e}")
452                     # Default to Unknown location as a last resort
453                     return self.insert_location("Unknown", "Unknown", "Unknown")
454             finally:
455                 cursor.close()
456
457             def insert_skill(self, skill_name):
458                 """
459                 Insert a skill or get its ID if it already exists.
460
461                 Args:
462                     skill_name (str): Name of the skill
463
464                 Returns:
465                     int: The skill_id
466
467                 cursor = self.connection.cursor()
468                 try:
469                     # Check if skill already exists
470                     cursor.execute("SELECT skill_id FROM skills WHERE skill_name =
471 :1", (skill_name,))
472                     result = cursor.fetchone()

```

```
469
470     if result:
471         return result[0]
472
473     # Insert new skill
474     cursor.execute("INSERT INTO skills (skill_name) VALUES (:1)", (
475         skill_name,))
476     self.connection.commit()
477     cursor.execute("SELECT skill_id FROM skills WHERE skill_name = :1",
478         (skill_name,))
479     result = cursor.fetchone()
480     if result:
481         return result[0]
482     else:
483         logger.warning("Insert skill failed to return ID for: %s",
484         skill_name)
485     return None
486
487 except DatabaseError as e:
488     self.connection.rollback()
489     logger.error(f"Error inserting skill: {e}")
490     raise
491 finally:
492     cursor.close()
493
494 def insert_job(self, title, company_id, location_id, description,
495 post_date):
496     """
497     Insert a job listing.
498
499     Args:
500         title (str): Job title
501         company_id (int): Foreign key to companies table
502         location_id (int): Foreign key to locations table
503         description (str): Job description
504         post_date (str): Date job was posted
505
506     Returns:
507         int: The job_id
508     """
509
510     # Convert post_date string to a date object
511     if isinstance(post_date, str):
512         try:
513             post_date = datetime.strptime(post_date, '%Y-%m-%d')
514
515     else:
516         post_date = datetime.date.fromisoformat(post_date)
```

```

508         except ValueError:
509             logger.warning(f"Unrecognized date format for post_date: {post_date}")
510             post_date = None
511             cursor = self.connection.cursor()
512             try:
513                 cursor.execute(
514                     "INSERT INTO jobs (title, company_id, location_id,
515                     description, post_date) "
516                     "VALUES (:1, :2, :3, :4, :5)",
517                     (title, company_id, location_id, description, post_date)
518                 )
519                 self.connection.commit()
520                 cursor.execute("SELECT job_id FROM jobs WHERE title = :1 AND
521                     company_id = :2", (title, company_id))
522                 return cursor.fetchone()[0]
523             except DatabaseError as e:
524                 self.connection.rollback()
525                 logger.error(f"Error inserting job: {e}")
526                 raise
527             finally:
528                 cursor.close()
529
530     def link_job_skill(self, job_id, skill_id):
531         """
532             Create a job-skill relationship in the join table.
533
534         Args:
535             job_id (int): ID of the job
536             skill_id (int): ID of the skill
537         """
538             cursor = self.connection.cursor()
539             try:
540                 cursor.execute(
541                     "INSERT INTO job_skills (job_id, skill_id) VALUES (:1, :2)"
542                     ,
543                     (job_id, skill_id)
544                 )
545                 self.connection.commit()
546             except DatabaseError as e:
547                 self.connection.rollback()
548                 # If the job-skill link already exists, ignore the duplicate-
549                 key error

```

```

546         if 'ORA-00001' in str(e):
547             logger.info(f"Link already exists for job_id={job_id},
548 skill_id={skill_id}")
549             return
550             logger.error(f"Error linking job and skill: {e}")
551             raise
552         finally:
553             cursor.close()
554
555     def save_job_listing(self, job_data):
556         """
557             Save a complete job listing to the database.
558
559             Args:
560                 job_data (dict): Dictionary containing job information
561
562             Returns:
563                 int: The job_id of the inserted job
564
565         try:
566             # Insert company and get ID
567             company_id = self.insert_company(
568                 job_data['company_name'],
569                 job_data.get('industry'),
570                 job_data.get('company_size')
571             )
572
573             # Insert location and get ID
574             location_id = self.insert_location(
575                 job_data.get('city'),
576                 job_data.get('state'),
577                 job_data.get('country')
578             )
579
580             # Insert job and get ID
581             job_id = self.insert_job(
582                 job_data['title'],
583                 company_id,
584                 location_id,
585                 job_data.get('description', ''),
586                 job_data.get('post_date')
587             )

```

```

588         # Insert skills and link to job
589         for skill in job_data.get('skills', []):
590             skill_id = self.insert_skill(skill)
591             self.link_job_skill(job_id, skill_id)
592
593             logger.info(f"Successfully saved job: {job_data['title']} at {job_data['company_name']}")
594             return job_id
595
596     except DatabaseError as e:
597         logger.error(f"Error saving job listing: {e}")
598         raise
599
600     def close(self):
601         """Close the database connection."""
602         if self.connection:
603             self.connection.close()
604             logger.info("Database connection closed")

```

5.5 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 (min, max)
19 }
20
21 # User agent for requests

```

```

22 USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36"
23
24 # Sample LinkedIn job search URL
25 SAMPLE_URL = "https://www.linkedin.com/jobs/search/?keywords=software%20
engineer&location=United%20States"

```

6 Conclusion

This milestone represents a major technical achievement: the core backend of my LinkedIn Job Scraper is now fully stable, performant, and backed by a properly normalized Oracle schema. After extensive experimentation with both local and cloud AI models for parsing, I ultimately chose a robust, hard-coded approach to extract and normalize each field. I navigated complex environment and dependency issues, refactored my original two-entity design into a five-entity, 3NF schema, and harnessed Oracle sequences and triggers for reliable key generation. With the scraper, parser, and database layers all integrated and stress-tested, I can now focus on crafting advanced SQL reports, capturing dashboard snapshots, and finalizing my project documentation for submission.

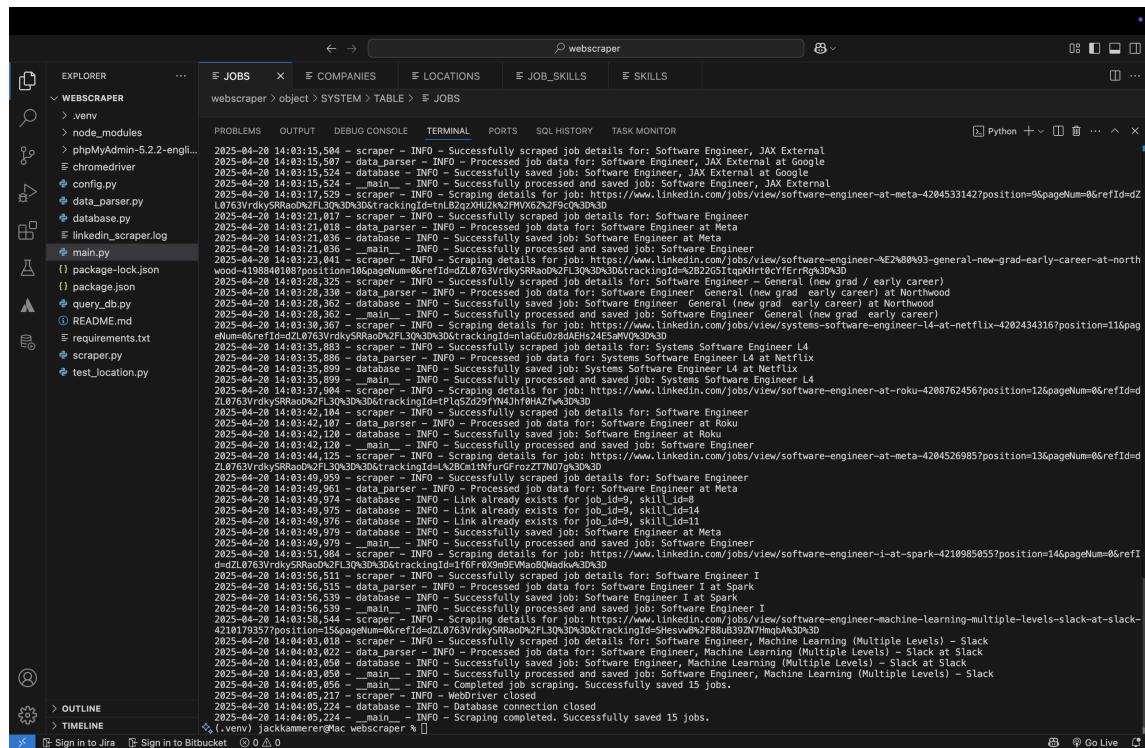


Figure 15: Finished Program