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.

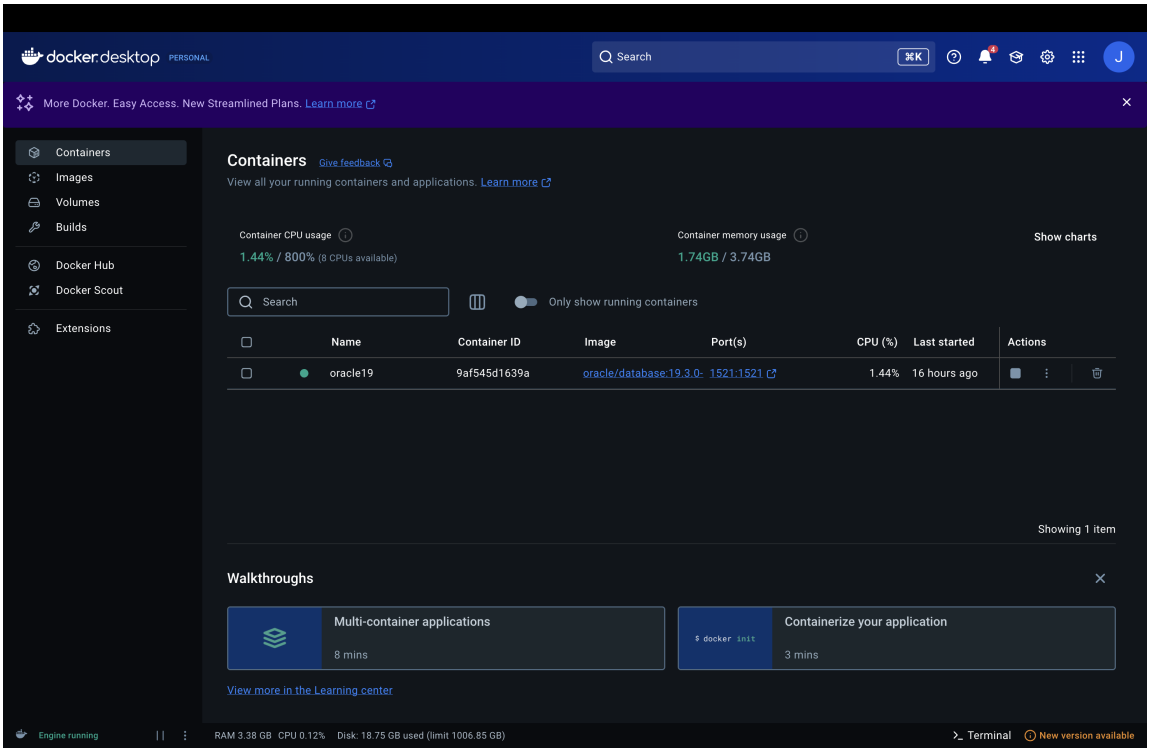

Figure 1: Oracle running in Docker

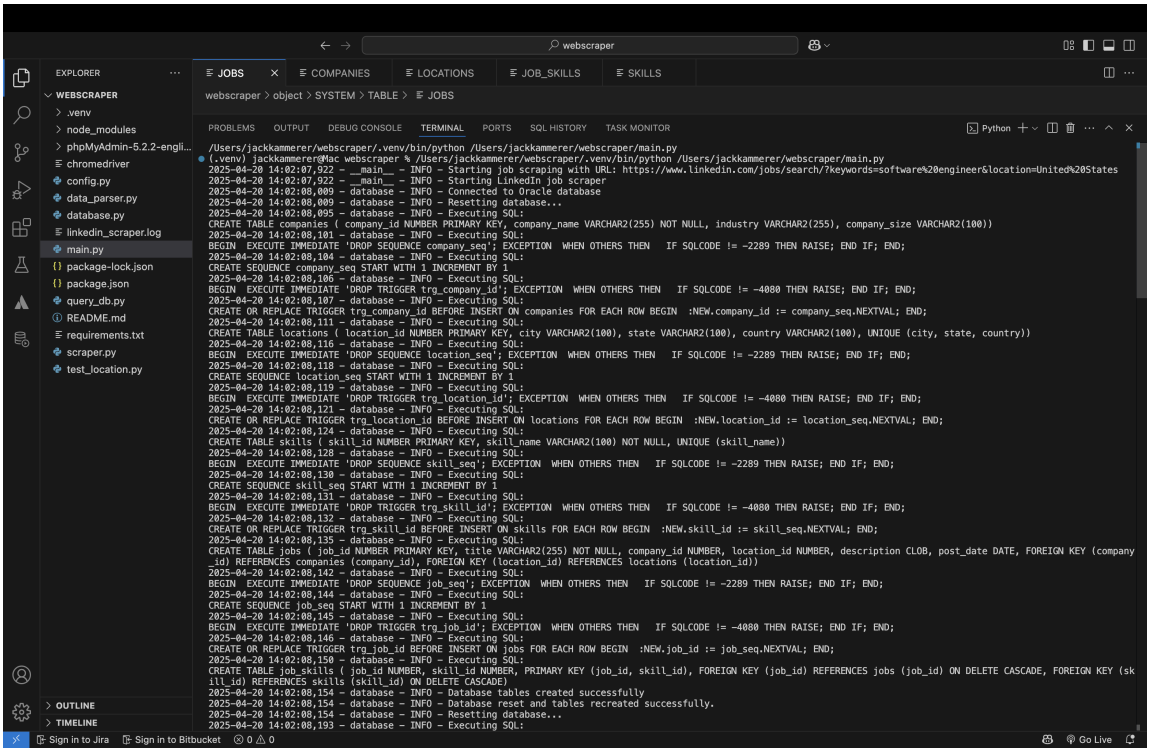

Figure 2: Starting the Program

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.

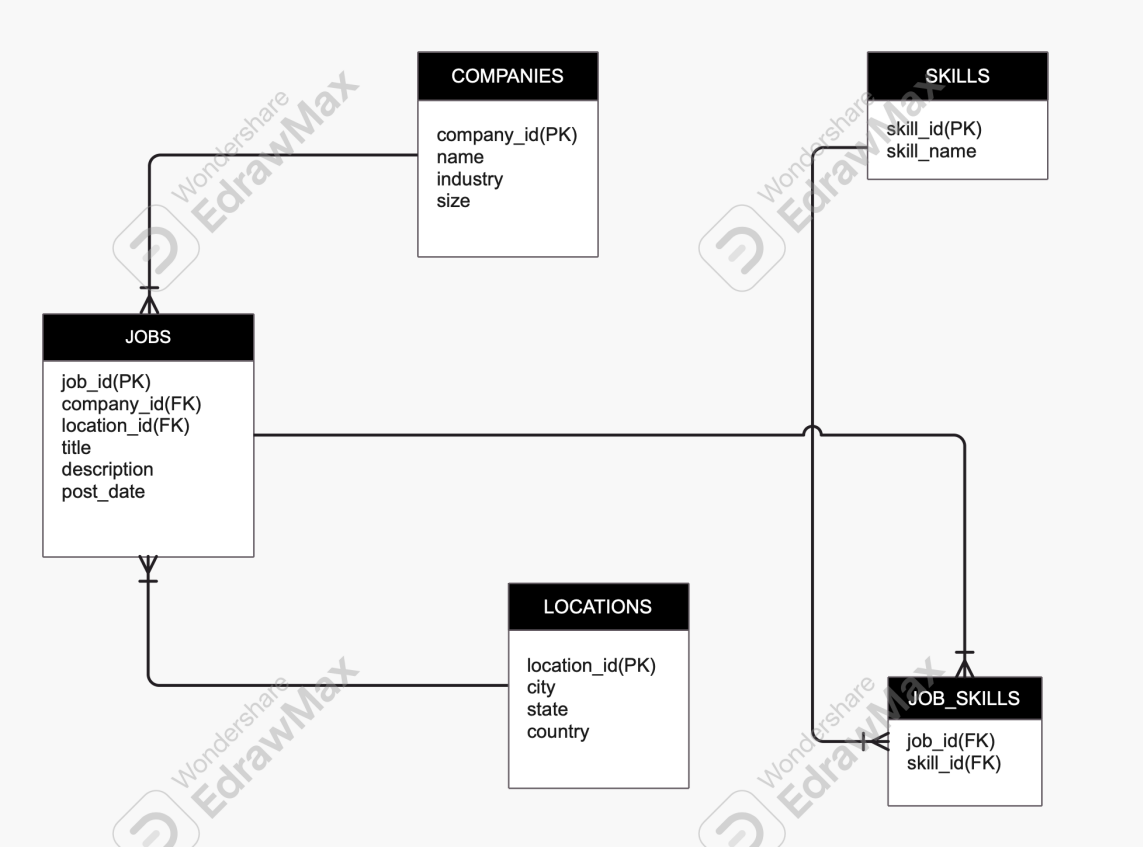

Figure 3: New E/R Diagram - 3NF

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:

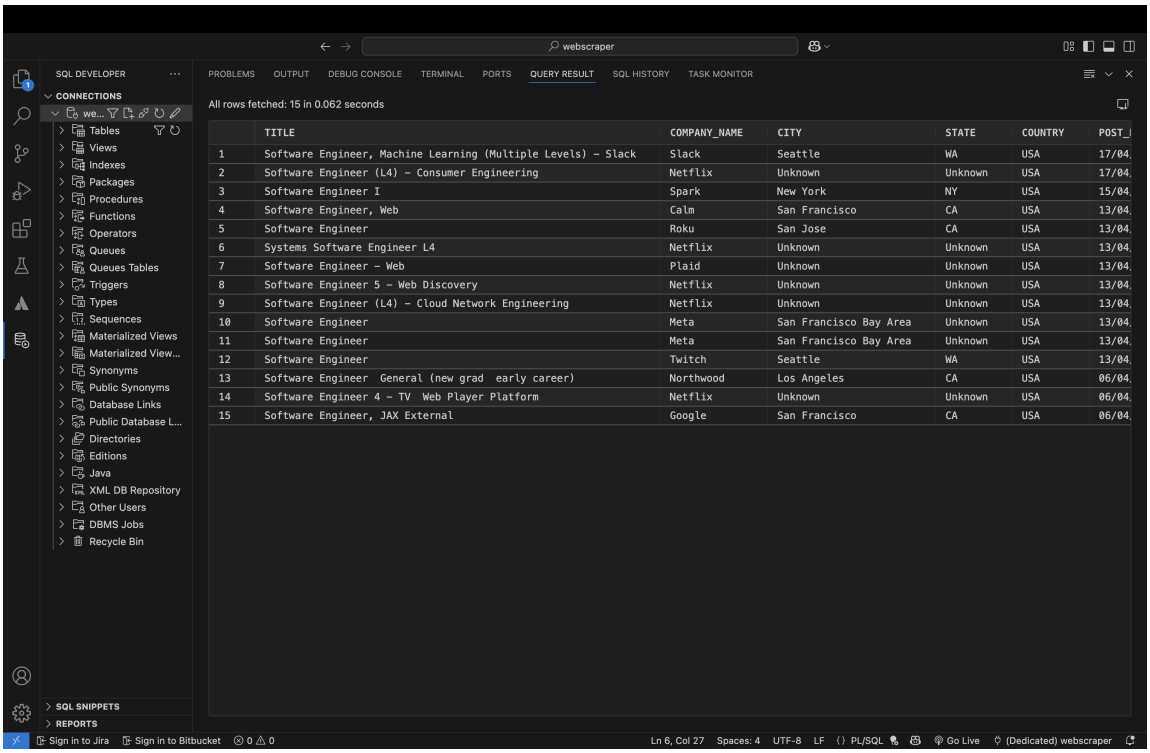

Figure 4: Query 1: Display of all tech job postings with company, location, and post date

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:

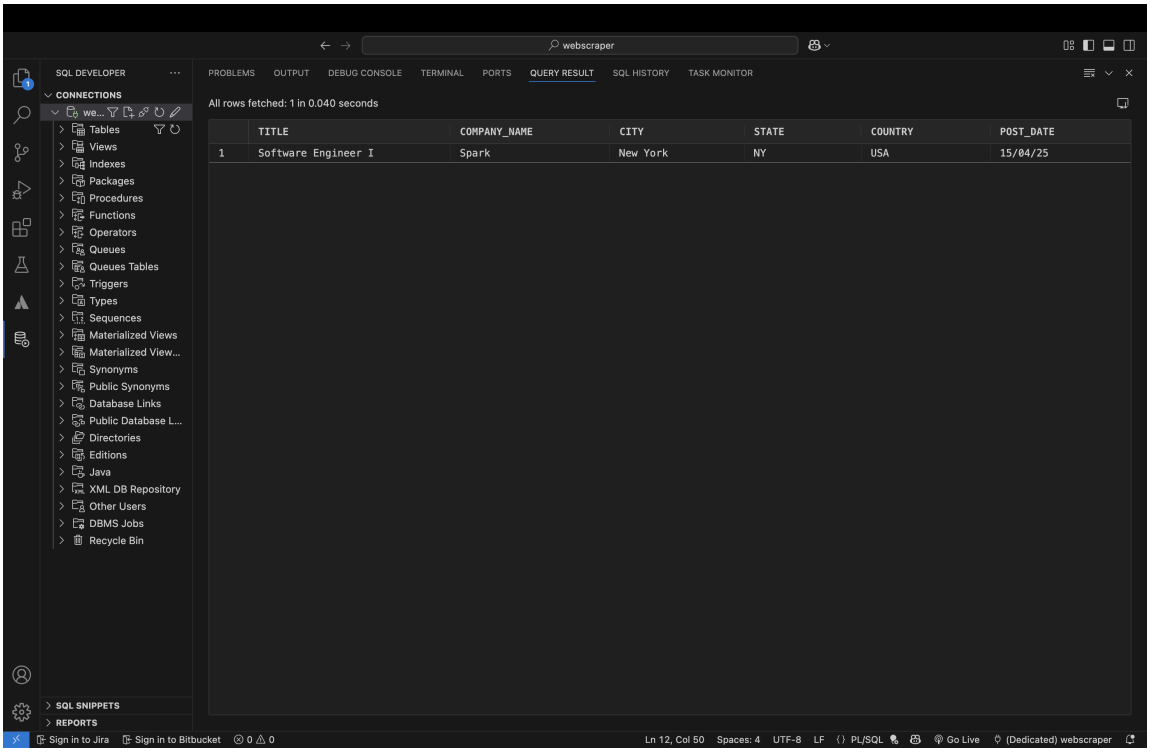

Figure 5: Query 2: Jobs with a focus on web development

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:

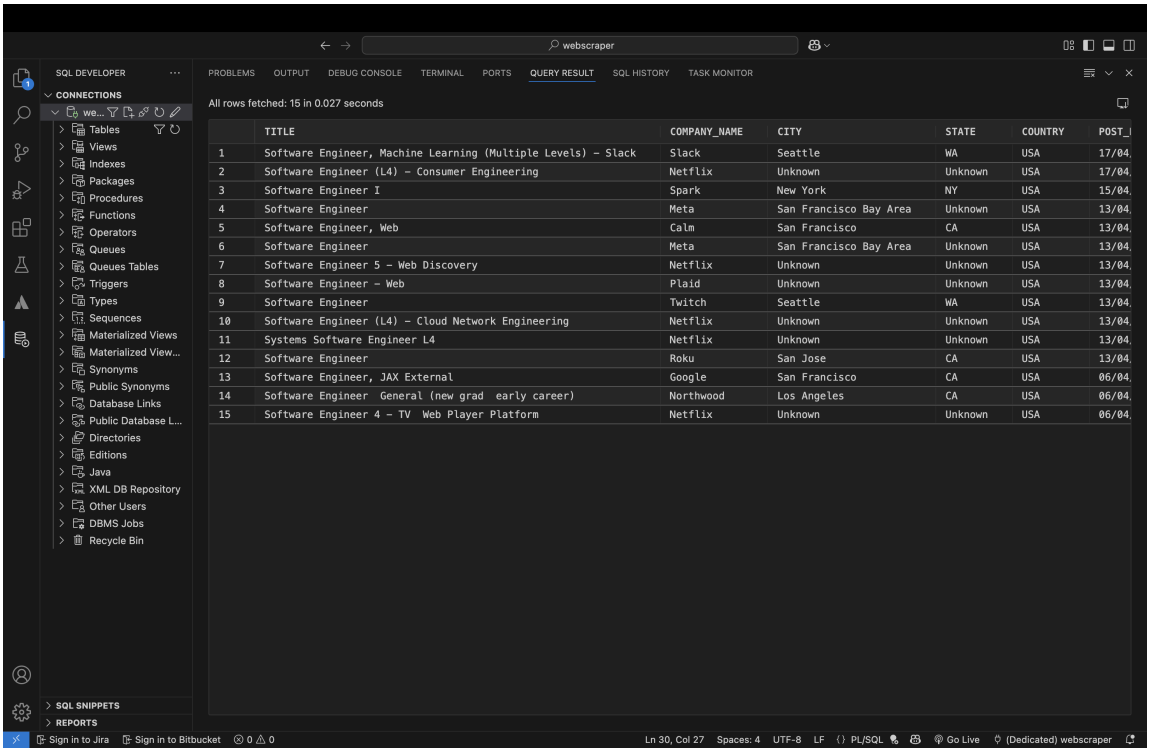

Figure 6: Query 3: Software engineering job postings

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:

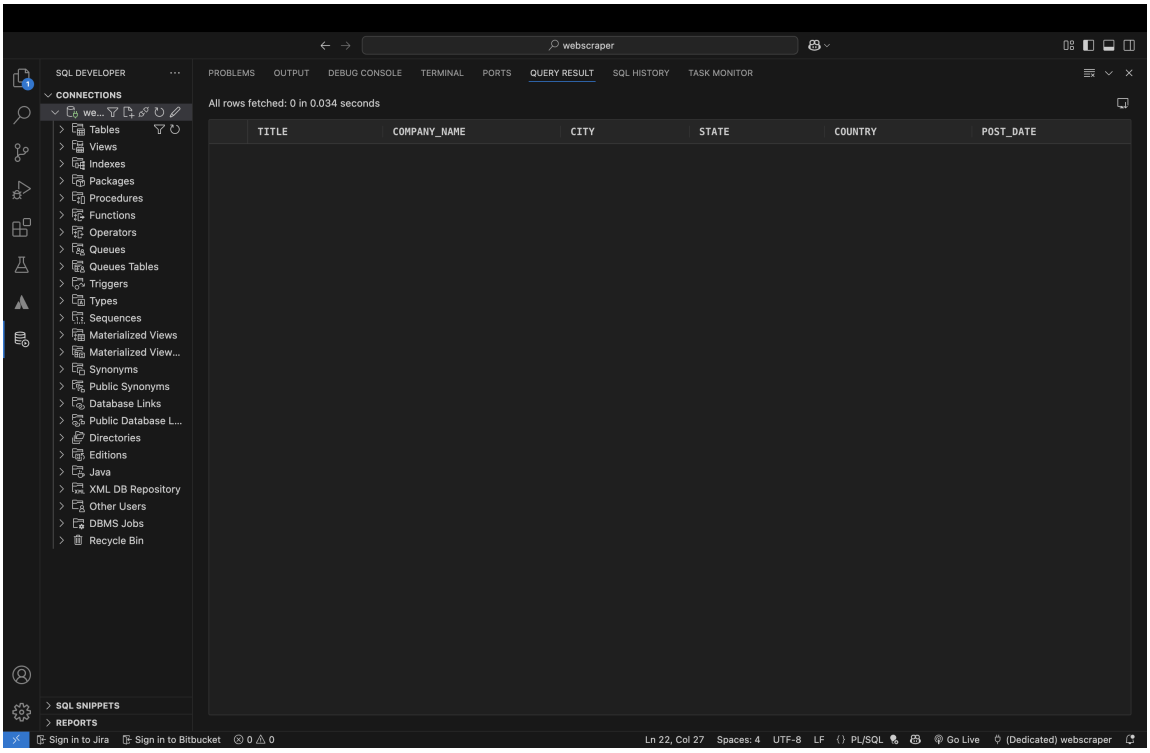

Figure 7: Query 4: IT-related job listings

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:

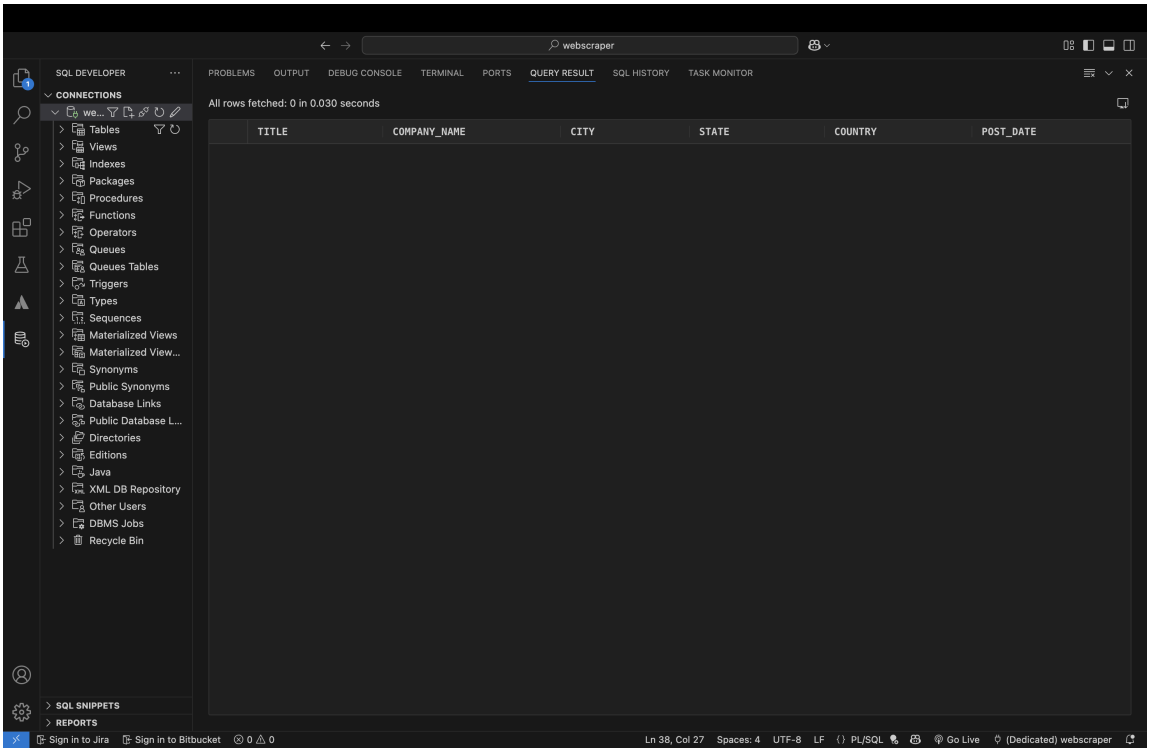

Figure 8: Query 5: Front-end and back-end related postings

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:

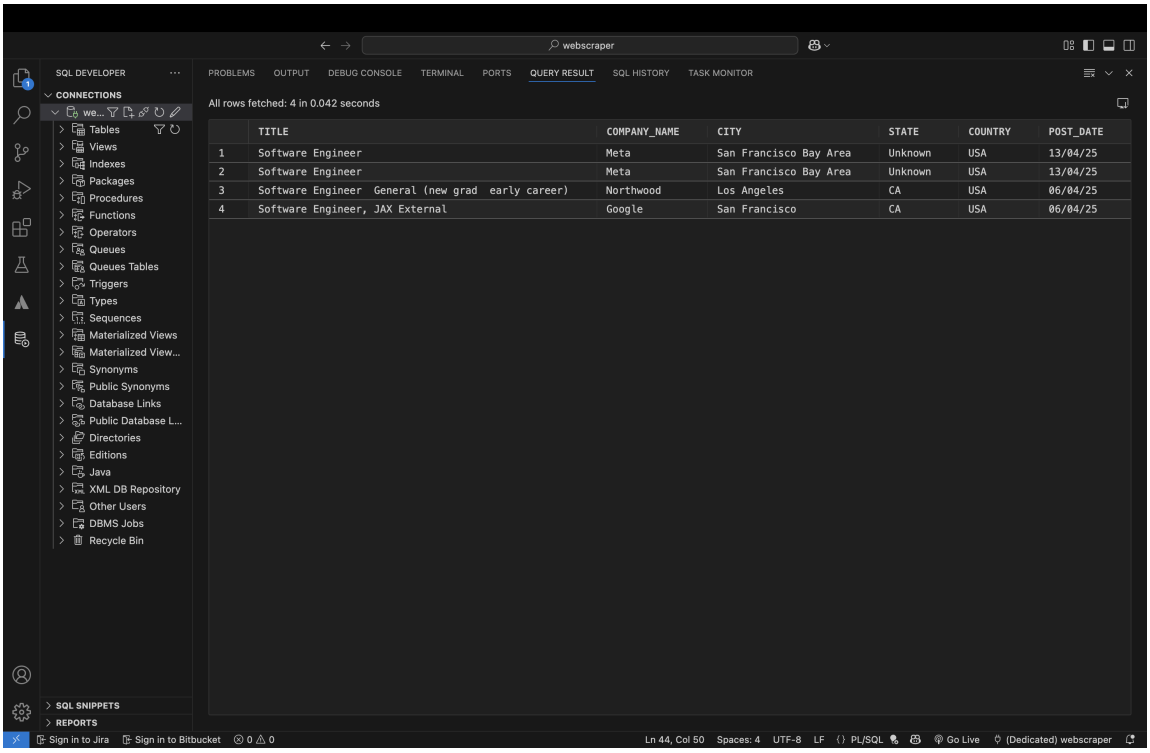

Figure 9: Query 6: Full-stack job opportunities

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;
```

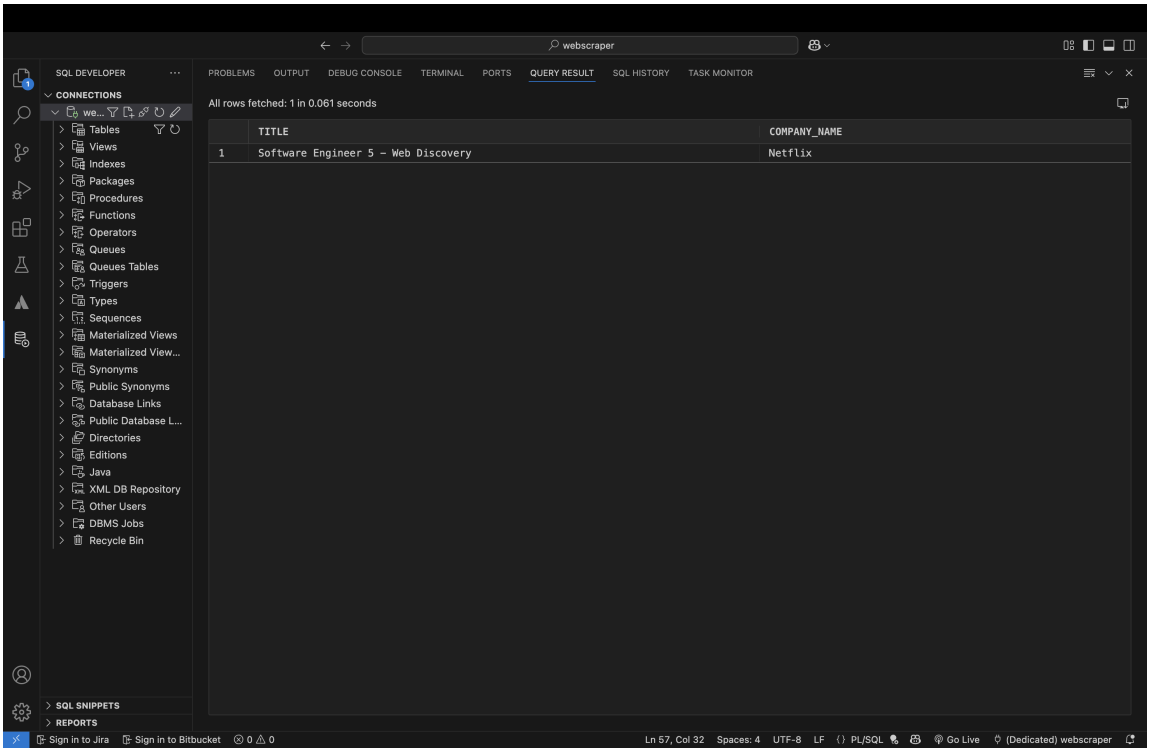

Figure 10: Query 7: Jobs requiring both React and Node.js

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:

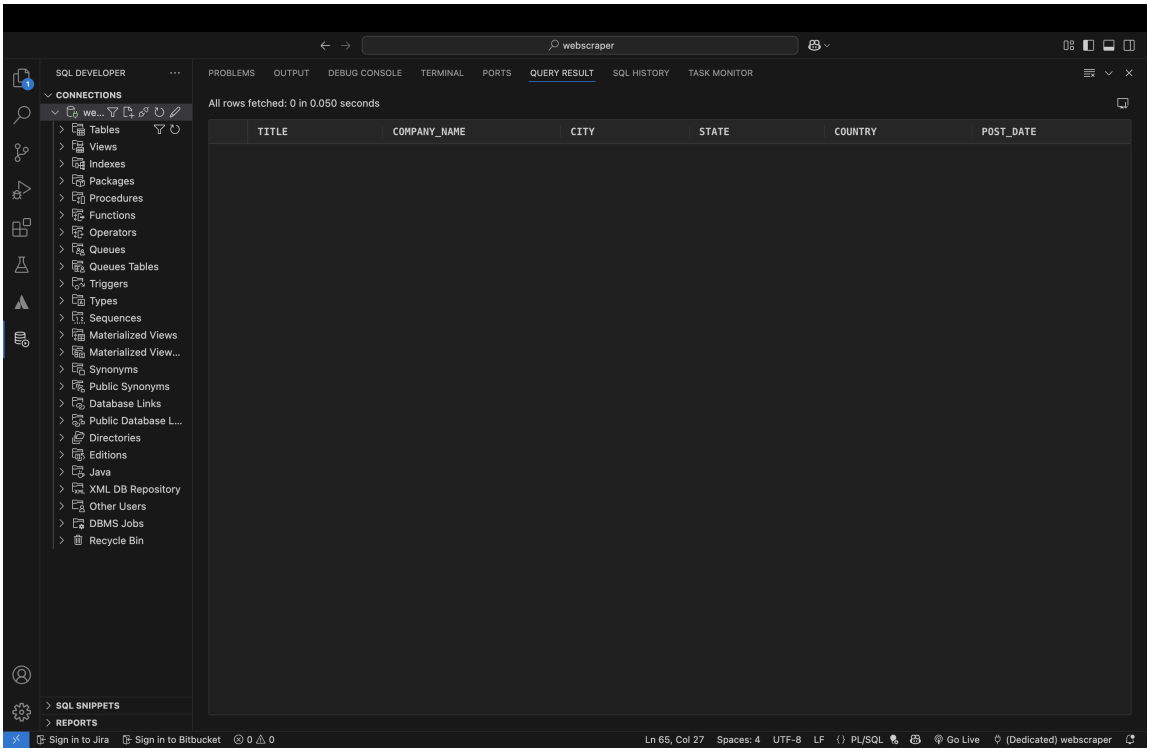

Figure 11: Query 8: Senior and Lead positions

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:

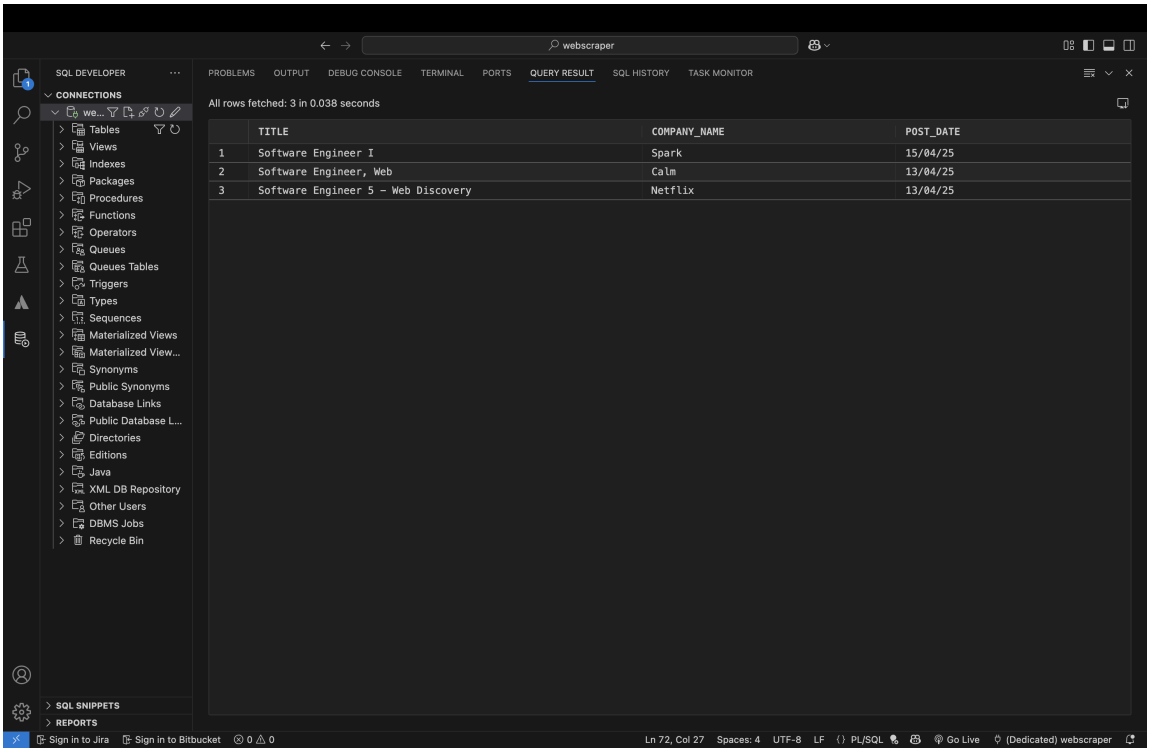

Figure 12: Query 9: Remote or telecommuting job listings

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 d
    WHEN LOWER(j.title) LIKE '%software engineering%' OR LOWER(j.description) LIKE '%
    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 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 '%
    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:

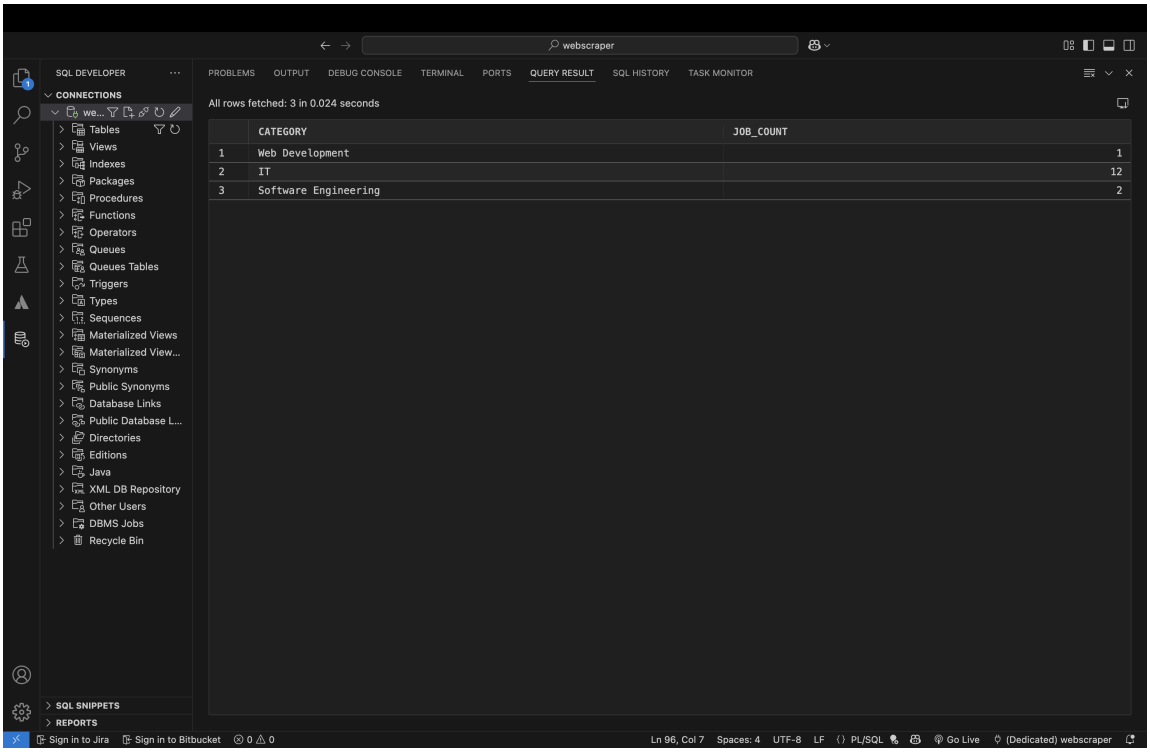

Figure 13: Query 10: Summary count of jobs by category

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

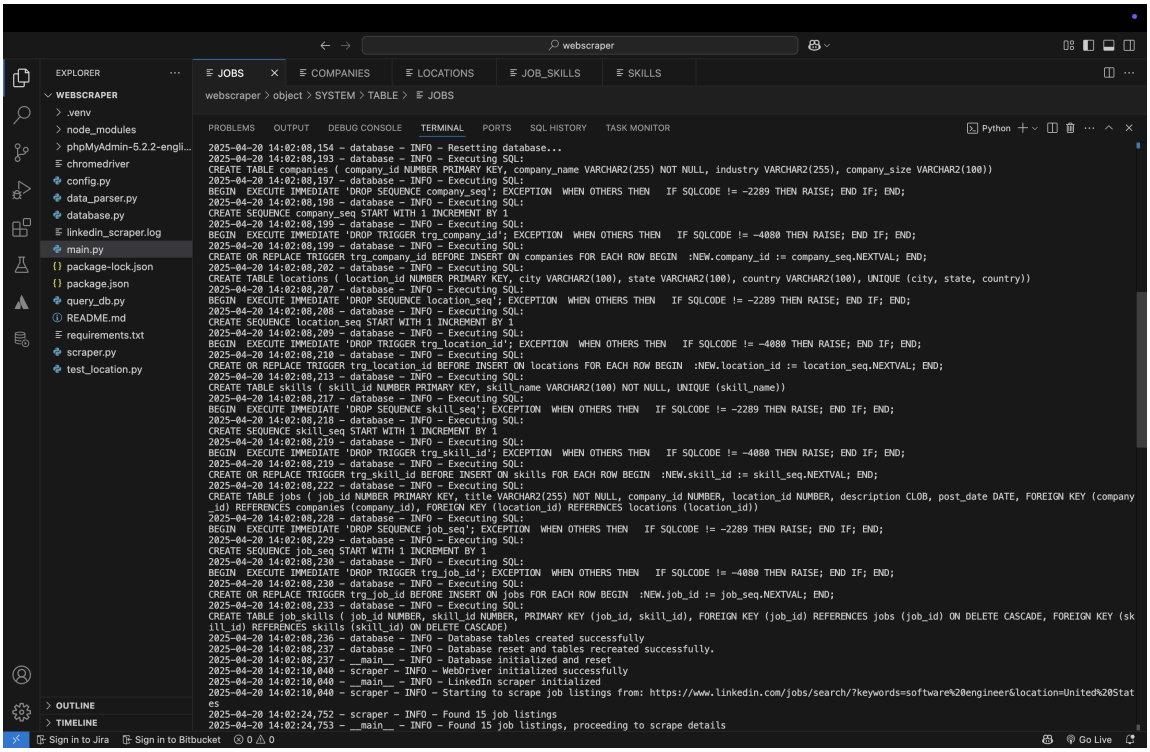

Figure 14: Tables Created and Job Listings Found

5 Source Code

5.1 Main Python Files

5.2 main.py

```
"""
Main execution script for LinkedIn job scraper.
This script orchestrates the entire scraping, parsing, and saving process.
"""
```

```
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
    scrape details")
60
61        # Process each job URL
62        for job_url in job_urls:
63            try:
64                # Scrape job details
65                raw_job_data = scraper.scrape_job_details(job_url)
66
67                if not raw_job_data or 'title' not in raw_job_data:
68                    logger.warning(f"Skipping job, could not scrape details
    : {job_url}")
69                    continue
70
71                # Process and clean job data
72                processed_job_data = process_job_data(raw_job_data)
73
74                # Save to database
75                db.save_job_listing(processed_job_data)
76                job_count += 1
77
78                logger.info(f"Successfully processed and saved job: {
    processed_job_data['title']}")
79
80                # Random sleep between requests
81                time.sleep(2)  # Small delay between jobs
82
83            except Exception as e:
84                logger.error(f"Error processing job {job_url}: {e}")
85                continue
86
```

```
        logger.info(f"Completed job scraping. Successfully saved {job_count} jobs.")
        return job_count

    except Exception as e:
        logger.error(f"Error in job scraping process: {e}")
        return job_count

    finally:
        # Clean up resources
        if scraper:
            scraper.close()

        if db:
            db.close()


if __name__ == "__main__":
    # Get URL from command line or use sample URL
    url = sys.argv[1] if len(sys.argv) > 1 else SAMPLE_URL

    logger.info(f"Starting job scraping with URL: {url}")
    success_count = run_scraper(url)

    logger.info(f"Scraping completed. Successfully saved {success_count} jobs.")
```

5.3 scraper.py

```
"""
LinkedIn job scraper module.
This module handles the web scraping functionality.
"""

import time
import random
import logging
import mysql.connector
from config import DB_CONFIG
from database import Database
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
```

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import (
    TimeoutException, NoSuchElementException,
    StaleElementReferenceException
)

from config import SCRAPING_CONFIG, USER_AGENT

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)


class LinkedInScraper:
    def __init__(self, db, headless=True):
        """
        Initialize the LinkedIn scraper.

        Args:
            headless (bool): Whether to run the browser in headless mode
        """
        self.db = db
        self.setup_driver(headless)

    def setup_driver(self, headless):
        """
        Set up the Selenium WebDriver.

        Args:
            headless (bool): Whether to run in headless mode
        """
        chrome_options = Options()
        if headless:
            chrome_options.add_argument("--headless")

        chrome_options.add_argument("--no-sandbox")
        chrome_options.add_argument("--disable-dev-shm-usage")
        chrome_options.add_argument("--disable-gpu")
        chrome_options.add_argument(f"user-agent={USER_AGENT}")
```

```
58
59          service = Service('./chromedriver')
60          self.driver = webdriver.Chrome(service=service, options=
    chrome_options)
61
62          try:
63              self.driver.set_window_size(1920, 1080)
64              self.wait = WebDriverWait(self.driver, SCRAPING_CONFIG['timeout
    '])
65              logger.info("WebDriver initialized successfully")
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
    load
86
87              # Get job cards
88              job_cards = self.wait.until(
89                  EC.presence_of_all_elements_located(
90                      (By.CSS_SELECTOR, ".jobs-search__results-list li")
91                  )
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").
    get_attribute("href")
102                     job_urls.append(job_link)
103                     count += 1
104                 except (NoSuchElementException,
    StaleElementReferenceException) as e:
105                     logger.warning(f"Could not extract job link: {e}")
106                     continue
107
108             logger.info(f"Found {len(job_urls)} job listings")
109             return job_urls
110
111         except TimeoutException:
112             logger.error("Timeout while loading LinkedIn job search results
    ")
113             return []
114         except Exception as e:
115             logger.error(f"Error scraping job listings: {e}")
116             return []
117
118     def scrape_job_details(self, job_url):
119         """
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
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         """
```

```
        Safely extract an element's text or attribute.

        Args:
            by: Selenium By locator
            selector (str): CSS selector
            get_text (bool): Whether to get innerText instead of
    textContent

        Returns:
            str: Extracted text or None if element not found
        """
        try:
            element = self.driver.find_element(by, selector)
            if get_text:
                return element.get_attribute("innerText")
            return element.text
        except (NoSuchElementException, StaleElementReferenceException):
            return None

    def close(self):
        """Close the WebDriver."""
        if self.driver:
            self.driver.quit()
            logger.info("WebDriver closed")
```

5.4 database.py

```
"""
Database operations for the LinkedIn job scraper.
This module handles Oracle connections and database queries.
"""

import oracledb
from oracledb import DatabaseError
import logging
from datetime import datetime

# Hardcoded Oracle config
ORACLE_CONFIG = {
    "user": "system",  # or admin, oracle, etc.
    "password": "swagdata",
    "dsn": oracledb.makedsn("localhost", 1521, service_name="ORCLPDB1")
}

```

```
# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)


class Database:
    def __init__(self):
        """Initialize database connection and create tables if they don't exist."""
        self.connection = None
        try:
            self.connection = oracledb.connect(**ORACLE_CONFIG)
            logger.info("Connected to Oracle database")
            self.reset_database()
        except DatabaseError as e:
            logger.error(f"Error connecting to Oracle: {e}")
            raise

    def query(self, sql, params=None):
        """
        Execute a query and return the results.

        Args:
            sql (str): SQL query to execute
            params (tuple, optional): Parameters for the query

        Returns:
            list: Query results as a list of tuples
        """
        cursor = self.connection.cursor()
        try:
            if params:
                cursor.execute(sql, params)
            else:
                cursor.execute(sql)
            return cursor.fetchall()
        except DatabaseError as e:
            logger.error(f"Error executing query: {e}")
            return []
        finally:
```

```
            cursor.close()

    def reset_database(self):
        """
        Drop all tables and recreate them to start fresh.
        """
        cursor = self.connection.cursor()
        try:
            logger.info("Resetting database...")

            # Drop tables if they exist
            tables = ['job_skills', 'jobs', 'skills', 'locations', '
companies']
            for table in tables:
                cursor.execute(f"""
                BEGIN
                    EXECUTE IMMEDIATE 'DROP TABLE {table}';
                EXCEPTION
                    WHEN OTHERS THEN
                        IF SQLCODE != -942 THEN
                            RAISE;
                        END IF;
                END;
                """)

            # Recreate the tables
            self.create_tables()
            logger.info("Database reset and tables recreated successfully."
)

        except DatabaseError as e:
            logger.error(f"Error resetting database: {e}")
            raise
        finally:
            cursor.close()

    def create_tables(self):
        """Create necessary tables if they don't exist."""
        cursor = self.connection.cursor()
        try:
            # Create companies table
            sql = (
                "CREATE TABLE companies ("
```

```
                " company_id NUMBER PRIMARY KEY,"
                " company_name VARCHAR2(255) NOT NULL,"
                " industry VARCHAR2(255),"
                " company_size VARCHAR2(100)"
                ")"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Drop sequence if it exists
            sql = (
                "BEGIN "
                " EXECUTE IMMEDIATE 'DROP SEQUENCE company_seq'; "
                "EXCEPTION "
                " WHEN OTHERS THEN "
                "  IF SQLCODE != -2289 THEN RAISE; END IF; "
                "END;"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Create sequence for company_id
            sql = "CREATE SEQUENCE company_seq START WITH 1 INCREMENT BY 1"
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Drop trigger if it exists
            sql = (
                "BEGIN "
                " EXECUTE IMMEDIATE 'DROP TRIGGER trg_company_id'; "
                "EXCEPTION "
                " WHEN OTHERS THEN "
                "  IF SQLCODE != -4080 THEN RAISE; END IF; "
                "END;"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Create trigger for company_id auto-increment
            sql = (
                "CREATE OR REPLACE TRIGGER trg_company_id "
                "BEFORE INSERT ON companies "
                "FOR EACH ROW "
```

```
                "BEGIN "
                " :NEW.company_id := company_seq.NEXTVAL; "
                "END;"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Create locations table
            sql = (
                "CREATE TABLE locations ("
                " location_id NUMBER PRIMARY KEY,"
                " city VARCHAR2(100),"
                " state VARCHAR2(100),"
                " country VARCHAR2(100),"
                " UNIQUE (city, state, country)"
                ")"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Drop sequence if it exists
            sql = (
                "BEGIN "
                " EXECUTE IMMEDIATE 'DROP SEQUENCE location_seq'; "
                "EXCEPTION "
                " WHEN OTHERS THEN "
                "  IF SQLCODE != -2289 THEN RAISE; END IF; "
                "END;"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Create sequence for location_id
            sql = "CREATE SEQUENCE location_seq START WITH 1 INCREMENT BY 1
    "
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Drop trigger if it exists
            sql = (
                "BEGIN "
                " EXECUTE IMMEDIATE 'DROP TRIGGER trg_location_id'; "
                "EXCEPTION "
```

```
                "  WHEN OTHERS THEN "
                "    IF SQLCODE != -4080 THEN RAISE; END IF; "
                "END;"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Create trigger for location_id auto-increment
            sql = (
                "CREATE OR REPLACE TRIGGER trg_location_id "
                "BEFORE INSERT ON locations "
                "FOR EACH ROW "
                "BEGIN "
                " :NEW.location_id := location_seq.NEXTVAL; "
                "END;"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Create skills table
            sql = (
                "CREATE TABLE skills ("
                " skill_id NUMBER PRIMARY KEY,"
                " skill_name VARCHAR2(100) NOT NULL,"
                " UNIQUE (skill_name)"
                ")"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Drop sequence if it exists
            sql = (
                "BEGIN "
                " EXECUTE IMMEDIATE 'DROP SEQUENCE skill_seq'; "
                "EXCEPTION "
                " WHEN OTHERS THEN "
                "   IF SQLCODE != -2289 THEN RAISE; END IF; "
                "END;"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Create sequence for skill_id
```

```
            sql = "CREATE SEQUENCE skill_seq START WITH 1 INCREMENT BY 1"
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Drop trigger if it exists
            sql = (
                "BEGIN "
                " EXECUTE IMMEDIATE 'DROP TRIGGER trg_skill_id'; "
                "EXCEPTION "
                " WHEN OTHERS THEN "
                "  IF SQLCODE != -4080 THEN RAISE; END IF; "
                "END;"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Create trigger for skill_id auto-increment
            sql = (
                "CREATE OR REPLACE TRIGGER trg_skill_id "
                "BEFORE INSERT ON skills "
                "FOR EACH ROW "
                "BEGIN "
                " :NEW.skill_id := skill_seq.NEXTVAL; "
                "END;"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Create jobs table
            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)"
                ")"
            )
```

```
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Drop sequence if it exists
            sql = (
                "BEGIN "
                " EXECUTE IMMEDIATE 'DROP SEQUENCE job_seq'; "
                "EXCEPTION "
                " WHEN OTHERS THEN "
                "  IF SQLCODE != -2289 THEN RAISE; END IF; "
                "END;"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Create sequence for job_id
            sql = "CREATE SEQUENCE job_seq START WITH 1 INCREMENT BY 1"
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Drop trigger if it exists
            sql = (
                "BEGIN "
                " EXECUTE IMMEDIATE 'DROP TRIGGER trg_job_id'; "
                "EXCEPTION "
                " WHEN OTHERS THEN "
                "  IF SQLCODE != -4080 THEN RAISE; END IF; "
                "END;"
            )
            logger.info(f"Executing SQL:\n{sql}")
            cursor.execute(sql)

            # Create trigger for job_id auto-increment
            sql = (
                "CREATE OR REPLACE TRIGGER trg_job_id "
                "BEFORE INSERT ON jobs "
                "FOR EACH ROW "
                "BEGIN "
                " :NEW.job_id := job_seq.NEXTVAL; "
                "END;"
            )
            logger.info(f"Executing SQL:\n{sql}")
            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
    CASCADE,"
321                 " FOREIGN KEY (skill_id) REFERENCES skills (skill_id) ON
    DELETE CASCADE"
322                 ")"
323             )
324             logger.info(f"Executing SQL:\n{sql}")
325             cursor.execute(sql)
326
327             self.connection.commit()
328             logger.info("Database tables created successfully")
329         except DatabaseError as e:
330             logger.error(f"Error creating tables: {e}")
331             raise
332         finally:
333             cursor.close()
334
335     def insert_company(self, name, industry=None, company_size=None):
336         """
337         Insert a company or get its ID if it already exists.
338
339         Args:
340             name (str): Company name
341             industry (str, optional): Company industry
342             company_size (str, optional): Company size
343
344         Returns:
345             int: The company_id
346         """
347         cursor = self.connection.cursor()
348         try:
349             # Check if company already exists
350             cursor.execute('SELECT company_id FROM companies WHERE
    company_name = :1', (name,))
351             result = cursor.fetchone()
352
```

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

```
            country = "Unknown"
            logger.warning("No location data provided, using default: %s, %s, %s", city, state, country)
        elif country is None:
            # Ensure we at least have a country if other fields are present
            country = "Unknown"

        # Convert None values to "Unknown" for consistent database storage
        city = city if city is not None else "Unknown"
        state = state if state is not None else "Unknown"
        country = country if country is not None else "Unknown"

        cursor = self.connection.cursor()
        try:
            # Check if location already exists
            cursor.execute(
                "SELECT location_id FROM locations WHERE city = :1 AND state = :2 AND country = :3",
                (city, state, country)
            )
            result = cursor.fetchone()

            if result:
                return result[0]

            # Insert new location
            cursor.execute(
                "INSERT INTO locations (city, state, country) VALUES (:1, :2, :3)",
                (city, state, country)
            )
            self.connection.commit()
            cursor.execute("SELECT location_id FROM locations WHERE city = :1 AND state = :2 AND country = :3", (city, state, country))
            result = cursor.fetchone()
            if result:
                return result[0]
            else:
                logger.warning("Insert location failed to return ID for: %s, %s, %s", city, state, country)
                # Return ID for default location as fallback
                cursor.execute("SELECT location_id FROM locations WHERE city = 'Unknown' AND state = 'Unknown' AND country = 'Unknown'")
```

```
                default_result = cursor.fetchone()
                if default_result:
                    return default_result[0]
                # If no default exists, create it now
                cursor.execute("INSERT INTO locations (city, state, country
) VALUES ('Unknown', 'Unknown', 'Unknown')")
                self.connection.commit()
                return self.insert_location("Unknown", "Unknown", "Unknown"
)
        except DatabaseError as e:
            self.connection.rollback()
            # Handle duplicate location gracefully by fetching existing ID
            if 'ORA-00001' in str(e):
                cursor.execute(
                    "SELECT location_id FROM locations WHERE city = :1 AND
state = :2 AND country = :3",
                    (city, state, country)
                )
                result = cursor.fetchone()
                if result:
                    return result[0]
            logger.error(f"Error inserting location: {e}")
            # Default to Unknown location as a last resort
            return self.insert_location("Unknown", "Unknown", "Unknown")
        finally:
            cursor.close()

    def insert_skill(self, skill_name):
        """
        Insert a skill or get its ID if it already exists.

        Args:
            skill_name (str): Name of the skill

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

```
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,
    description, post_date) "
515                 "VALUES (:1, :2, :3, :4, :5)",
516                 (title, company_id, location_id, description, post_date)
517             )
518             self.connection.commit()
519             cursor.execute("SELECT job_id FROM jobs WHERE title = :1 AND
    company_id = :2", (title, company_id))
520             return cursor.fetchone()[0]
521         except DatabaseError as e:
522             self.connection.rollback()
523             logger.error(f"Error inserting job: {e}")
524             raise
525         finally:
526             cursor.close()
527 
528     def link_job_skill(self, job_id, skill_id):
529         """
530         Create a job-skill relationship in the join table.
531 
532         Args:
533             job_id (int): ID of the job
534             skill_id (int): ID of the skill
535         """
536         cursor = self.connection.cursor()
537         try:
538             cursor.execute(
539                 "INSERT INTO job_skills (job_id, skill_id) VALUES (:1, :2)"
    ,
540                 (job_id, skill_id)
541             )
542             self.connection.commit()
543         except DatabaseError as e:
544             self.connection.rollback()
545             # If the job-skill link already exists, ignore the duplicate-
    key error
```

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

```
            # Insert skills and link to job
            for skill in job_data.get('skills', []):
                skill_id = self.insert_skill(skill)
                self.link_job_skill(job_id, skill_id)

            logger.info(f"Successfully saved job: {job_data['title']} at {job_data['company_name']}")
            return job_id

        except DatabaseError as e:
            logger.error(f"Error saving job listing: {e}")
            raise

    def close(self):
        """Close the database connection."""
        if self.connection:
            self.connection.close()
            logger.info("Database connection closed")
```

5.5 config.py

```
"""
Configuration settings for LinkedIn job scraper.
"""

# Database configuration
DB_CONFIG = {
    'host': 'localhost',
    'user': 'root',
    'password': 'swagdata',
    'database': 'linkedin_jobs'
}

# Scraping configuration
SCRAPING_CONFIG = {
    'timeout': 30,  # Maximum seconds to wait for page loads
    'jobs_per_page': 25,
    'max_jobs': 15,  # Maximum number of jobs to scrape
    'sleep_interval': (2, 5),  # Random sleep interval between requests (min, max)
}

# User agent for requests
```

```
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"

# Sample LinkedIn job search URL
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.

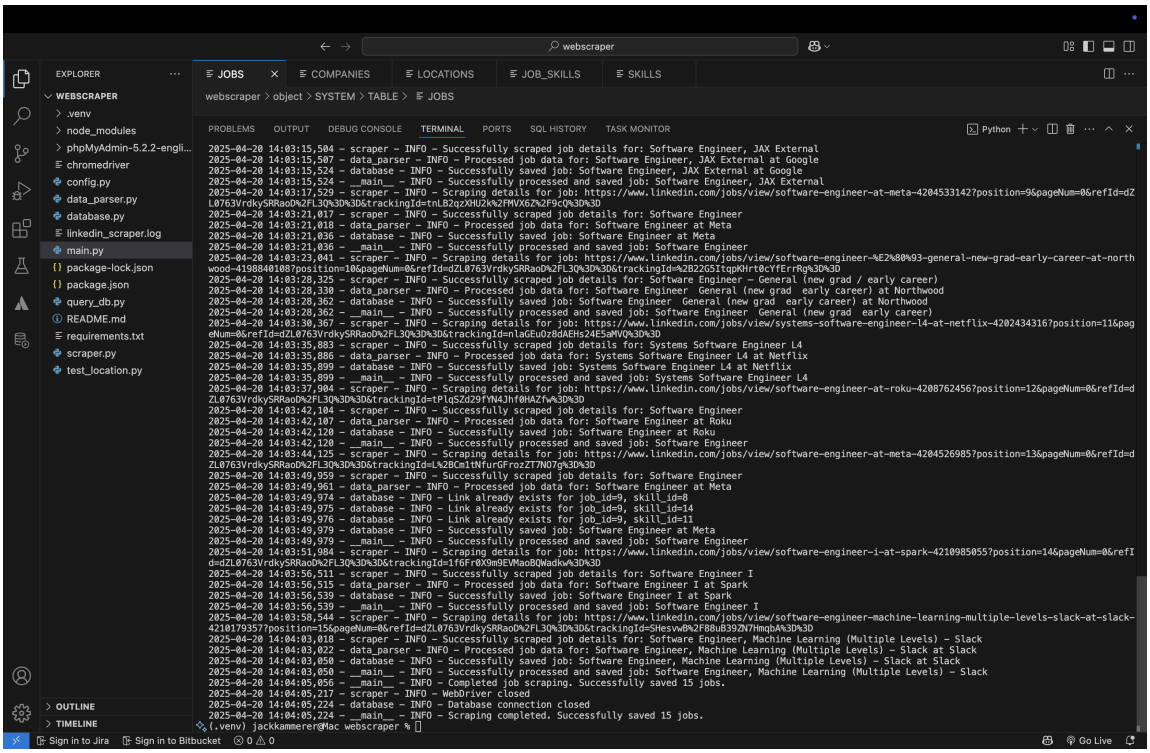

Figure 15: Finished Program