LABORATORY REPORT

Application Development Lab (CS33002)

B. Tech Program in ECSc

Submitted By

Name:- Akriti Patro

Roll No: 2230232

Kalinga Institute of Industrial Technology
(Deemed to be University)
Bhubaneswar, India

Spring 2024-2025

Table of Content

Experiment Number	7
Experiment Title	Natural Language Database Interaction with LLMs
Date of Experiment	18/03/25
Date of Submission	24/03/25

1. Objective:- To interact with databases using natural language queries powered by LLMs.

2. Procedure:-

1. I set up a MySQL database and populated it with sample data.

2. I integrated an LLM to convert natural language queries into SQL commands.

3. I developed a Flask backend to interact with the database.

4. I created a frontend for users to enter queries and view results.

3. Code:-

query.py

```
import os
import json
import re
import mysql.connector
from flask import Flask, request, jsonify, send_from_directory
from flask_cors import CORS
from dotenv import load_dotenv
import requests
from config import Config  # Import Config class

# Load environment variables
load_dotenv()

app = Flask(__name__, template_folder="templates")
CORS(app)  # Enable CORS for frontend communication

# MySQL Database Configuration
DB_CONFIG = {
    "host": Config.MYSQL_HOST,
    "user": Config.MYSQL_USER,
    "password": Config.MYSQL_PASSWORD,
    "database": Config.MYSQL_DB
}

# List of LLM models to try in fallback order
LLM_MODELS = ["mixtral-8x7b-32768", "llama3-8b", "gemma-7b"]

# Function to connect to MySQL
def connect_db():
    try:
        return mysql.connector.connect(**DB_CONFIG)
    except mysql.connector.Error as err:
        print(f"Database connection error: {err}")
        return None

# Function to get existing table names
def get_existing_tables():
    conn = connect_db()
    if not conn:
        return []
    try:
        cursor = conn.cursor()
        cursor.execute("SHOW TABLES;")
        tables = [row[0] for row in cursor.fetchall()]
        conn.close()
        return tables
    except mysql.connector.Error as err:
        print("❌ Error fetching tables:", err)
        conn.close()
        return []

# Function to get column names of a table
def get_table_columns(table_name):
    conn = connect_db()
    if not conn:
        return []
    try:
        cursor = conn.cursor()
        cursor.execute(f"SHOW COLUMNS FROM `{table_name}`;")
        columns = [row[0] for row in cursor.fetchall()]
        conn.close()
        return columns
    except mysql.connector.Error as err:
        print(f"❌ Error fetching columns for {table_name}:", err)
        conn.close()
        return []

# Function to convert natural language to SQL
def generate_sql(natural_query):
    """
    Converts a natural language query into an SQL query by detecting the table
    and requested columns.
    """

    existing_tables = get_existing_tables()
    detected_table = None

    # Find the table name in the query
    for table in existing_tables:
        if table in natural_query.lower():
            detected_table = table
            break

    if not detected_table:
            return use_llm_for_sql(natural_query)  # Fallback to LLM if no table is found

        # Extract potential column names
        query_words = natural_query.replace(detected_table, "").strip().split()
        table_columns = get_table_columns(detected_table)

        print(f"    Columns in `{detected_table}`: {table_columns}")

        # Filter valid columns
        valid_columns = [word.strip() for word in query_words if word in table_columns]

        # If valid columns are found, construct the SQL query
        if valid_columns:
            sql_query = f"SELECT {', '.join([f'`{col}`' for col in valid_columns])} FROM
`{detected_table}`;"
        else:
            sql_query = f"SELECT * FROM `{detected_table}`;"  # Default to all columns if none are
specified

        print(f"✅ Generated SQL Query: {sql_query}")
        return sql_query

    # Function to convert natural language to SQL using Groq LLM (fallback)
    def use_llm_for_sql(natural_query):
        url = "https://api.groq.com/openai/v1/chat/completions"
        headers = {
            "Authorization": f"Bearer {os.getenv('GROQ_API_KEY')}",
            "Content-Type": "application/json"
        }

        prompt_message = (
            "You are an expert MySQL query generator. "
            "For the given English query, return ONLY a valid SQL query in a single line, ending with a
semicolon. "
            "Do not include any explanation, comments, or additional text."
        )

        for model in LLM_MODELS:
            print(f"    Trying model: {model}")

            payload = {
                "model": model,
                "messages": [
                    {"role": "system", "content": prompt_message},
                    {"role": "user", "content": natural_query}
                ],
                "temperature": 0  # Low temperature for deterministic results
            }

            response = requests.post(url, headers=headers, json=payload)
            print(f"    {model} API Response:", response.text)  # Debugging API response

            if response.status_code == 200:
                response_json = response.json()
                try:
                    sql_query = response_json["choices"][0]["message"]["content"].strip()

                    # Validate SQL query
                    if
re.match(r"^\s*(SELECT|SHOW|INSERT|UPDATE|DELETE|CREATE|DROP|ALTER)\s+.*;$",
sql_query, re.IGNORECASE):
                    print(f"✅ Generated SQL Query: {sql_query}")
                    return sql_query
                    else:
                    print("❌ No valid SQL query detected.")
                except (KeyError, IndexError) as e:
                    print(f"Error extracting SQL from {model}: {e}")

        return None  # Return None if no valid SQL query was generated

    # Route to serve the frontend (index.html)
    @app.route("/")
    def home():
        return send_from_directory("templates", "index.html")

    # Route to handle user queries
    @app.route("/query", methods=["POST"])
    def process_query():
        data = request.get_json()
        if not data or "query" not in data:
            return jsonify({"error": "No query provided"}), 400

        user_query = data["query"]
        print("    Received query:", user_query)

        # Convert natural query to SQL
        sql_query = generate_sql(user_query)
        if not sql_query:
            return jsonify({"error": "Could not generate a valid SQL query."}), 400

        conn = connect_db()
        if conn is None:
            return jsonify({"error": "Database connection failed."}), 500

        try:
            cursor = conn.cursor(dictionary=True)

            # Set the correct database before executing the query
            cursor.execute(f"USE {Config.MYSQL_DB};")

            cursor.execute(sql_query)
            results = cursor.fetchall()
            conn.close()
            print("✅ SQL executed successfully.")
            return jsonify({"sql_query": sql_query, "results": results})
        except mysql.connector.Error as err:
            print("❌ SQL Execution Error:", err)
            return jsonify({"error": f"SQL Execution Error: {err}"})
```

```
if __name__ == "__main__":
    app.run(debug=True)
```

index.html

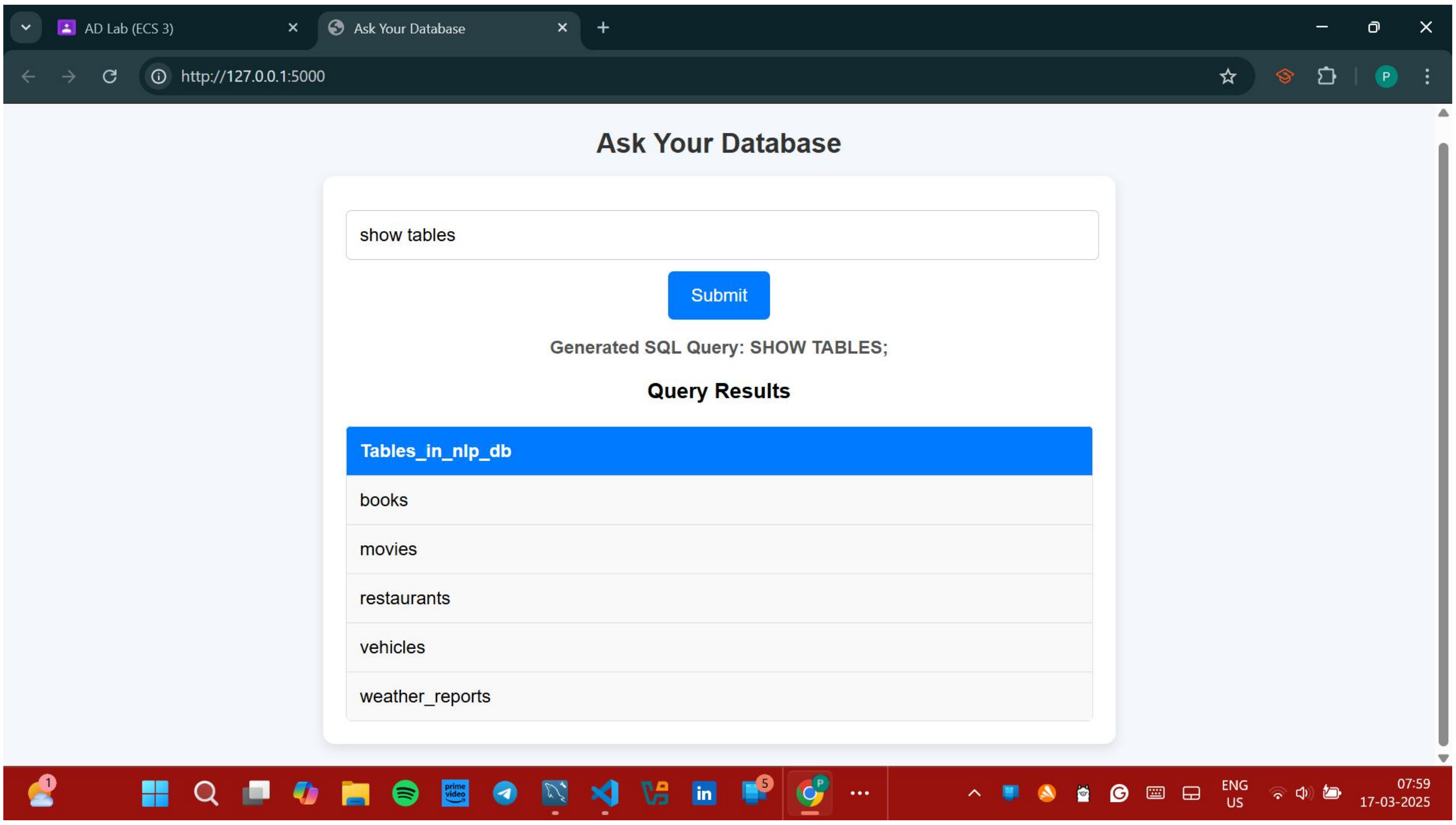
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ask Your Database</title>
    <style>
        body {
            font-family: 'Arial', sans-serif;
            max-width: 700px;
            margin: auto;
            text-align: center;
            padding: 20px;
            background-color: #f4f7fc;
        }
        h2 {
            color: #333;
            margin-bottom: 15px;
        }
        .container {
            background: white;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
        }
        input {
            width: calc(100% - 20px);
            padding: 12px;
            margin: 10px 0;
            border: 1px solid #ccc;
            border-radius: 5px;
            font-size: 16px;
        }
        button {
            padding: 12px 20px;
            background-color: #007bff;
            color: white;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            font-size: 16px;
            transition: 0.3s;
        }
        button:hover {
            background-color: #0056b3;
        }
        #sqlQuery {
            font-weight: bold;
            margin-top: 15px;
            word-wrap: break-word;
            color: #555;
        }
        #queryResults {
            margin-top: 20px;
            text-align: left;
        }
        table {
            width: 100%;
            border-collapse: collapse;
            margin-top: 15px;
            background: white;
            border-radius: 5px;
            overflow: hidden;
        }
        th, td {
            border: 1px solid #ddd;
            padding: 12px;
            text-align: left;
        }
        th {
            background-color: #007bff;
            color: white;
        }
        td {
            background-color: #f9f9f9;
        }
        .loading {
            font-style: italic;
            color: #777;
        }
        .hidden {
            display: none;
        }
    </style>
    <script>
        document.addEventListener("DOMContentLoaded", () => {
            const queryInput = document.getElementById("query");
            queryInput.addEventListener("keydown", (event) => {
                if (event.key === "Enter") {
                    submitQuery();
                }
            });
        });

        async function submitQuery() {
            const userQuery = document.getElementById("query").value;
            const sqlQueryElem = document.getElementById("sqlQuery");
            const queryResultsElem = document.getElementById("queryResults");
            const resultsContainer = document.getElementById("resultsContainer");

            if (!userQuery) {
```

```
            alert("Please enter a query!");
            return;
        }

        sqlQueryElem.innerText = "";
        queryResultsElem.innerHTML = "<p class='loading'>Processing your request...</p>";
        resultsContainer.classList.add("hidden");

        const apiUrl = `${window.location.origin}/query`;

        try {
            const response = await fetch(apiUrl, {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ query: userQuery })
            });

            const data = await response.json();
            queryResultsElem.innerHTML = "";
            resultsContainer.classList.remove("hidden");

            if (data.sql_query) {
                sqlQueryElem.innerText = "Generated SQL Query: " + data.sql_query;
            } else {
                sqlQueryElem.innerText = "Error generating SQL.";
            }

            if (data.results && data.results.length > 0) {
                queryResultsElem.innerHTML = generateTable(data.results);
            } else if (data.error) {
                queryResultsElem.innerHTML = `<p style="color: red;">Error: ${data.error}</p>`;
            } else {
                queryResultsElem.innerHTML = "<p>No results found.</p>";
            }
        } catch (error) {
            queryResultsElem.innerHTML = `<p style="color: red;">Server error: Unable to reach the
backend.</p>`;
        }
    }

    function generateTable(data) {
        let table = "<table><tr>";
        const headers = Object.keys(data[0]);
        headers.forEach(header => table += `<th>${header}</th>`);
        table += "</tr>";

        data.forEach(row => {
            table += "<tr>";
            headers.forEach(header => table += `<td>${row[header]}</td>`);
            table += "</tr>";
        });

        table += "</table>";
        return table;
    }
</script>
</head>
<body>
<h2>Ask Your Database</h2>
<div class="container">
    <input type="text" id="query" placeholder="Enter your question here">
    <button onclick="submitQuery()">Submit</button>

    <p id="sqlQuery"></p>

    <div id="resultsContainer" class="hidden">
        <h3>Query Results</h3>
        <div id="queryResults"></div>
    </div>
</div>
</body>
</html>
```

nlp.sql

```
CREATE DATABASE IF NOT EXISTS nlp_db;
USE nlp_db;

-- Drop tables if they exist
DROP TABLE IF EXISTS books;
DROP TABLE IF EXISTS movies;
DROP TABLE IF EXISTS restaurants;
DROP TABLE IF EXISTS weather_reports;
DROP TABLE IF EXISTS vehicles;

-- Books Table
CREATE TABLE IF NOT EXISTS books (
    book_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    author VARCHAR(255),
    genre VARCHAR(100),
    publication_year INT
);

INSERT INTO books (title, author, genre, publication_year) VALUES
('The Great Gatsby', 'F. Scott Fitzgerald', 'Classic', 1925),
('To Kill a Mockingbird', 'Harper Lee', 'Fiction', 1960),
('1984', 'George Orwell', 'Dystopian', 1949),
('The Catcher in the Rye', 'J.D. Salinger', 'Coming-of-Age', 1951),
('The Alchemist', 'Paulo Coelho', 'Philosophical', 1988);

-- Movies Table
CREATE TABLE IF NOT EXISTS movies (
    movie_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    director VARCHAR(255),
    release_year INT,
    rating DECIMAL(3,1)
);

INSERT INTO movies (title, director, release_year, rating) VALUES
('Inception', 'Christopher Nolan', 2010, 8.8),
('The Godfather', 'Francis Ford Coppola', 1972, 9.2),
('Titanic', 'James Cameron', 1997, 7.8),
('The Dark Knight', 'Christopher Nolan', 2008, 9.0),
('Parasite', 'Bong Joon-ho', 2019, 8.6);

-- Restaurants Table
CREATE TABLE IF NOT EXISTS restaurants (
    restaurant_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    cuisine VARCHAR(100),
    location VARCHAR(255),
    rating DECIMAL(3,1)
);

INSERT INTO restaurants (name, cuisine, location, rating) VALUES
('The Golden Spoon', 'Italian', 'New York', 4.5),
('Sushi Heaven', 'Japanese', 'Tokyo', 4.8),
('Spice Symphony', 'Indian', 'London', 4.3),
('BBQ Grill', 'American', 'Texas', 4.6),
('Café de Paris', 'French', 'Paris', 4.7);

-- Weather Reports Table
CREATE TABLE IF NOT EXISTS weather_reports (
    report_id INT AUTO_INCREMENT PRIMARY KEY,
    city VARCHAR(100),
    temperature DECIMAL(5,2),
    humidity INT,
    conditions VARCHAR(100),
    report_date DATE
);

INSERT INTO weather_reports (city, temperature, humidity, conditions, report_date) VALUES
('New York', 22.5, 60, 'Sunny', '2025-03-01'),
('London', 18.3, 70, 'Cloudy', '2025-03-01'),
('Tokyo', 25.2, 55, 'Clear', '2025-03-01'),
('Paris', 20.0, 65, 'Rainy', '2025-03-01'),
('Sydney', 28.4, 50, 'Windy', '2025-03-01');

-- Vehicles Table
CREATE TABLE IF NOT EXISTS vehicles (
    vehicle_id INT AUTO_INCREMENT PRIMARY KEY,
    make VARCHAR(100),
    model VARCHAR(100),
    year INT,
    price DECIMAL(10,2)
);

INSERT INTO vehicles (make, model, year, price) VALUES
('Toyota', 'Corolla', 2022, 25000),
('Honda', 'Civic', 2021, 23000),
('Tesla', 'Model 3', 2023, 45000),
('Ford', 'Mustang', 2020, 55000),
('BMW', 'X5', 2022, 60000);

-- Check All Tables
SHOW TABLES;

-- Verify Data
SELECT * FROM books;
SELECT * FROM movies;
SELECT * FROM restaurants;
SELECT * FROM weather_reports;
SELECT * FROM vehicles;
```

4. Results/Output:-



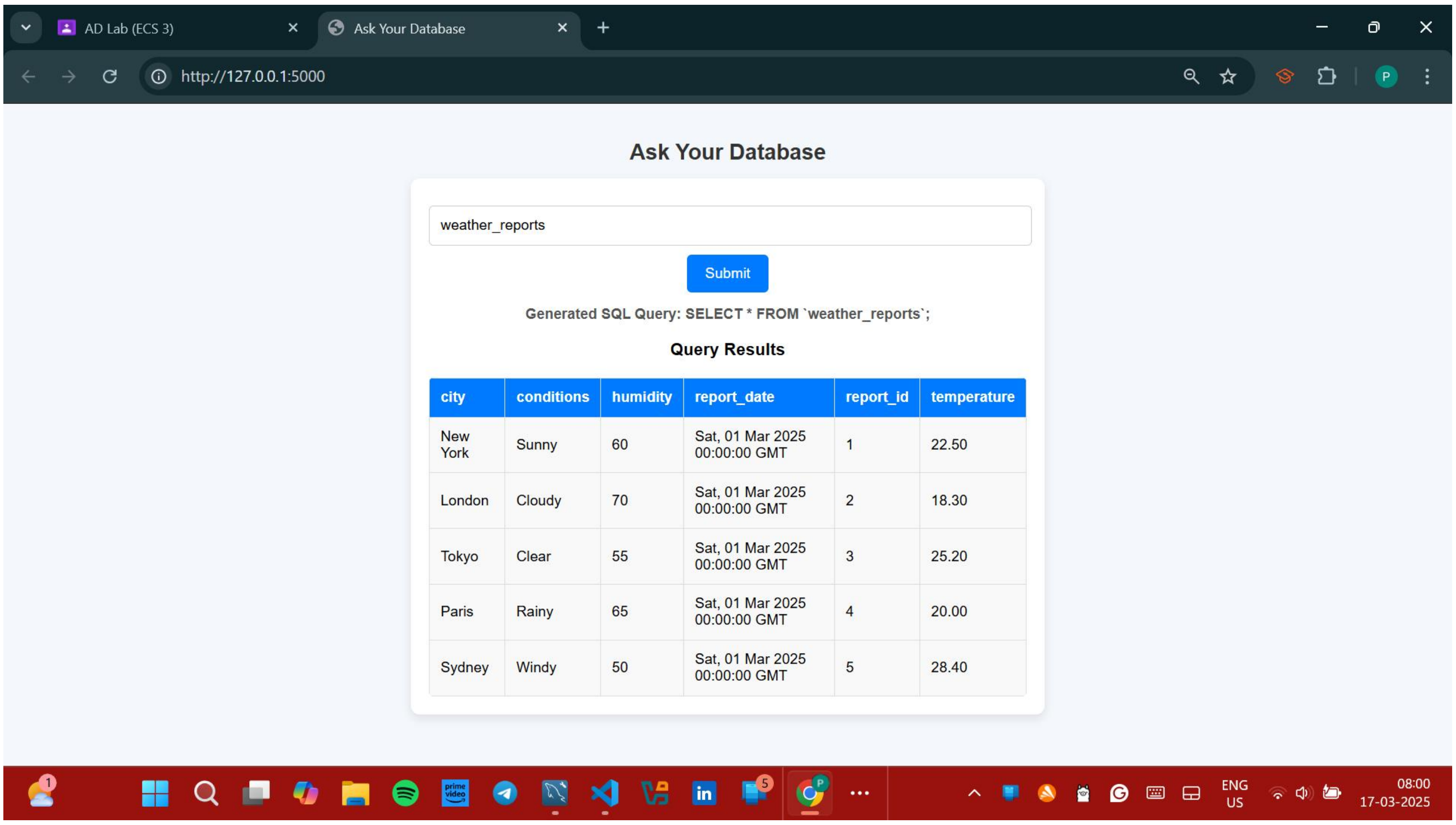
Ask Your Database

show tables

Generated SQL Query: SHOW TABLES;

Query Results

Tables_in_nlp_db
books
movies
restaurants
vehicles
weather_reports


Ask Your Database

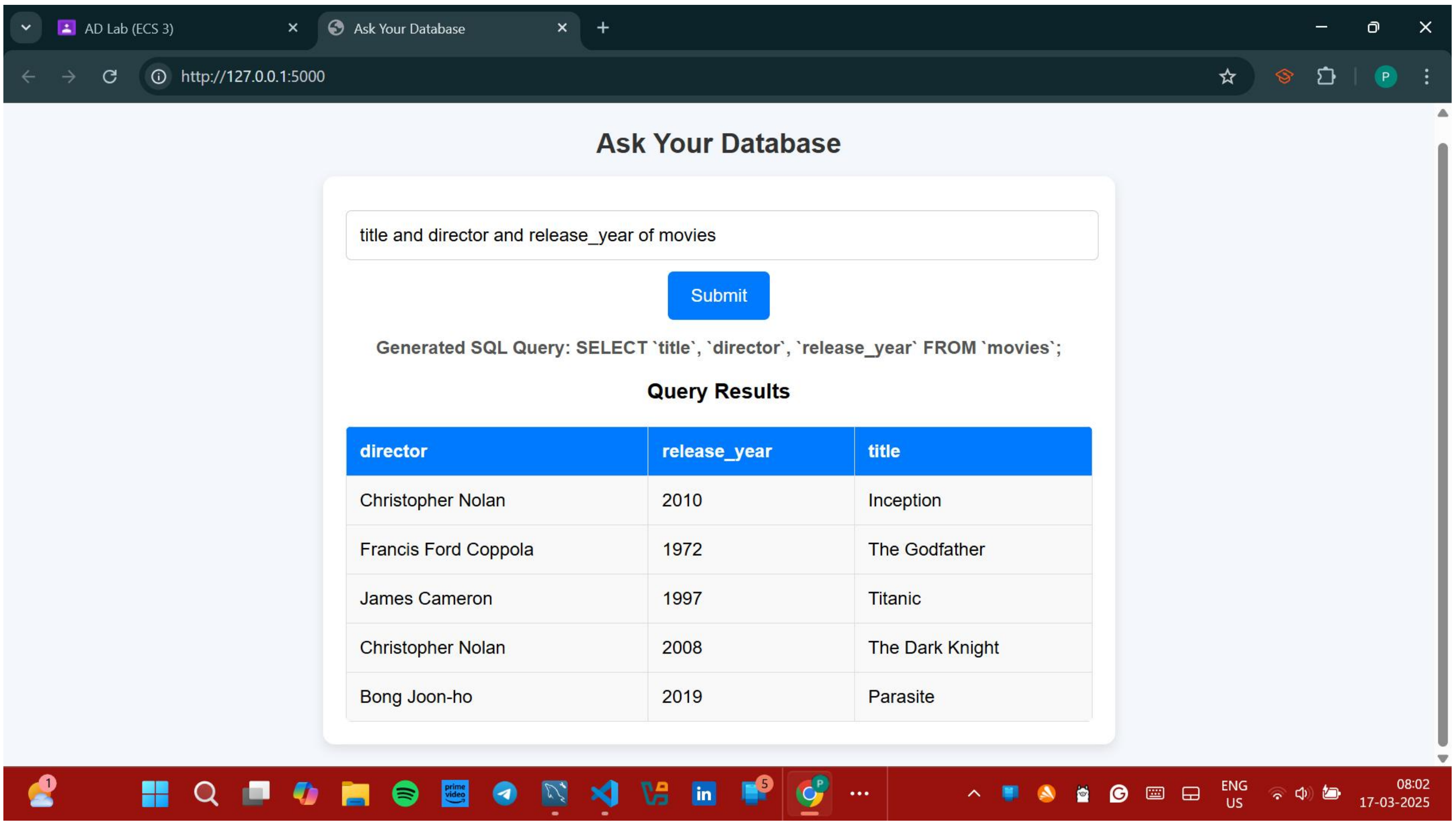
weather_reports

Submit

Generated SQL Query: SELECT * FROM `weather_reports`;

Query Results

city	conditions	humidity	report_date	report_id	temperature
New York	Sunny	60	Sat, 01 Mar 2025 00:00:00 GMT	1	22.50
London	Cloudy	70	Sat, 01 Mar 2025 00:00:00 GMT	2	18.30
Tokyo	Clear	55	Sat, 01 Mar 2025 00:00:00 GMT	3	25.20
Paris	Rainy	65	Sat, 01 Mar 2025 00:00:00 GMT	4	20.00
Sydney	Windy	50	Sat, 01 Mar 2025 00:00:00 GMT	5	28.40


Ask Your Database

title and director and release_year of movies

Submit

Generated SQL Query: SELECT `title`, `director`, `release_year` FROM `movies`;

Query Results

director	release_year	title
Christopher Nolan	2010	Inception
Francis Ford Coppola	1972	The Godfather
James Cameron	1997	Titanic
Christopher Nolan	2008	The Dark Knight
Bong Joon-ho	2019	Parasite

5. Remarks:-

In this experiment, I developed a natural language database interaction system using Flask, integrating an LLM (Large Language Model) to convert natural language queries into SQL commands. I used Flask-MySQLdb to interact with MySQL, enabling seamless communication between the user and the database.

The Flask backend processed user queries, dynamically generating SQL commands based on natural language input. If the system identified a matching table or column, it constructed an appropriate SQL query. Otherwise, it leveraged Groq's LLM (using models like Mixtral, LLaMA 3, and Gemma) to generate a valid SQL query. The backend executed these queries and returned the results to the frontend.

On the frontend, I built a responsive interface using HTML, and CSS allowing users to input natural language queries and view structured database results. The dashboard displayed query results in a tabular format.

Signature of the Student	Signature of the Lab Coordinator
Akriti Patro	Prof. Bhargav Appasani