LABORATORY REPORT

Application Development Lab (CS33002)

B.Tech Program in ECSc

Submitted By

Name: Shreyaa Venkateswaran

Roll No: 2230120

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

Spring 2024-2025

Table of Content

Exp No.	Title	Date of Experiment	Date of Submission	Remarks
1.	Experiment 1: Build a resume using HTML/CSS	07-01-2025	14-01-2025	
2.	Experiment 2: Machine Learning for Cat and Dog Classification	15-01-2025	20-01-2025	
3.	Experiment 3: Regression Analysis for Stock Prediction	21-01-2025	27-01-2025	
4.	Experiment 4: Conversational Chatbot with Any Files	04-02-2025	09-02-2025	
5.	Experiment 5: Web Scraper using LLMs	16-02-2025	17-03-2025	
6.	Experiment 6: Database Management Using Flask	11-03-2025	17-03-2025	
7.	Experiment 7: Natural Language Database Interaction with LLMs	18-03-2025	21-03-2025	
8.				
9.	Open Ended 1			
10.	Open Ended 2			

Experiment Number	7
Experiment Title	Natural Language Database Interaction with LLMs
Date of Experiment	18-03-2025
Date of Submission	21-03-2025

1. Objective:

To interact with databases using natural language queries powered by LLMs

2. Procedure:

1. Set up a MySQL database and populate it with sample data.

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

3. Develop a Flask backend to interact with the database.

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

3. Code:

app.py:

```
from flask import Flask, render_template, request, jsonify
from db import get_db_connection
from llm import generate_sql_query

app = Flask(__name__)

import os
from flask import Flask, render_template, request, jsonify
from db import get_db_connection
from llm import generate_sql_query

# Set template and static folder paths correctly
app                    =                    Flask(__name__,
template_folder="../frontend/templates",
static_folder="../frontend/static")
```

```
      @app.route("/")
      def index():
          return render_template("index.html")

      @app.route("/query", methods=["POST"])
      def query():
          data = request.json
          nl_query = data.get("query")

          if not nl_query:
               return jsonify({"error": "Query cannot be empty"}),
400

            print(f"Received NL Query: {nl_query}")  # Debugging
Line

          sql_query = generate_sql_query(nl_query)
          print(f"Generated SQL Query: {sql_query}")  # Debugging
Line

          conn = get_db_connection()
          cursor = conn.cursor(dictionary=True)

          try:
              cursor.execute(sql_query)
              results = cursor.fetchall()
              conn.commit()
          except Exception as e:
              print(f"SQL Execution Error: {e}")  # Debugging Line
              return jsonify({"error": str(e)}), 500
          finally:
              cursor.close()
              conn.close()

              return jsonify({"sql_query": sql_query, "results":
results})

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

db.py:

```
import mysql.connector
from config import DB_CONFIG

def get_db_connection():
    conn = mysql.connector.connect(**DB_CONFIG)
    return conn

```

llm.py:

```
import re
import requests

# Configure your Groq API details
GROQ_API_URL = "https://api.groq.com/openai/v1/chat/completions"
GROQ_API_KEY =
"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"

def generate_sql_query(natural_language_query):
      """ Sends a natural language query to the Groq LLM and
extracts the SQL query. """

    # Construct API request payload
    payload = {
        "model": "llama-3.3-70b-versatile",
        "messages": [
               {"role": "system", "content": "You are an AI that
converts natural language into MySQL queries."},
              {"role": "user", "content": f"Generate only a valid
MySQL query without explanation. Do not include markdown. Query:
{natural_language_query}"}
        ]
    }

        headers = {"Authorization": f"Bearer {GROQ_API_KEY}",
"Content-Type": "application/json"}

    try:
            response = requests.post(GROQ_API_URL, json=payload,
headers=headers)
        response.raise_for_status()
```

```
                                                      llm_response      =
response.json()["choices"][0]["message"]["content"]

        # Extract only the SQL query
        cleaned_sql = clean_sql_response(llm_response)
        return cleaned_sql

    except requests.exceptions.RequestException as e:
        print("Error communicating with Groq API:", e)
        return None

def clean_sql_response(llm_response):
    """Extracts only the SQL query from the LLM response."""
        match  =  re.search(r"```sql\n(.*?)\n```",  llm_response,
re.DOTALL)
    if match:
        return match.group(1).strip()

    return llm_response.split("\n")[0].strip()   # Fallback: First
line if no markdown

# Debugging
if __name__ == "__main__":
    test_query = "display the names of the employees"
    sql_query = generate_sql_query(test_query)
    print("Generated SQL Query:", sql_query)
```

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
            <meta   name="viewport"   content="width=device-width,
initial-scale=1.0">
    <title>NL to SQL Converter</title>
            <link   rel="stylesheet"   href="{{   url_for('static',
filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Natural Language to SQL Query</h1>
```

```
        <form id="query-form">
            <input type="text" id="nl-query" placeholder="Enter
your query..." required>
            <button type="submit">Generate SQL</button>
        </form>
        <div class="output">
            <p id="sql-query"></p>
            <p id="query-results"></p>
        </div>
    </div>

    <script>

document.getElementById("query-form").addEventListener("submit",
async function(event) {
            event.preventDefault();
            let userQuery =
document.getElementById("nl-query").value;

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

            let data = await response.json();

            document.getElementById("sql-query").innerText =
data.sql_query || "";

            // Handle results properly
            if (Array.isArray(data.results)) {

document.getElementById("query-results").innerText =
JSON.stringify(data.results, null, 2);
            } else if (typeof data.results === "object") {

document.getElementById("query-results").innerText =
JSON.stringify(data.results);
            } else {

document.getElementById("query-results").innerText = data.results
|| "";
```

```
            }
        });
    </script>
</body>
</html>
```

styles.css:

```
body {
    font-family: Arial, sans-serif;
    text-align: center;
     background: linear-gradient(to right, #ff9a9e, #fad0c4); /*
Pink Gradient */
    color: #333;
}

.container {
    max-width: 600px;
    margin: 50px auto;
    padding: 30px;
    background: rgba(255, 255, 255, 0.9);
    border-radius: 15px;
    box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.2);
}

h1 {
    color: #800080; /* Purple */
    font-size: 28px;
    margin-bottom: 20px;
}

input {
    width: 80%;
    padding: 10px;
    margin: 10px 0;
    border: 1px solid #ccc;
    border-radius: 5px;
    font-size: 16px;
}

button {
    background: #800080; /* Purple */
```

```css
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 5px;
    font-size: 16px;
    cursor: pointer;
    transition: 0.3s;
}

button:hover {
    background: #5d0078; /* Darker Purple */
}

.result-box {
    margin-top: 20px;
    padding: 20px;
    background: rgba(255, 255, 255, 0.8);
    border-radius: 10px;
    box-shadow: 0px 2px 8px rgba(0, 0, 0, 0.2);
}

.bold {
    font-weight: bold;
    font-size: 18px;
    color: #800080;
}
```

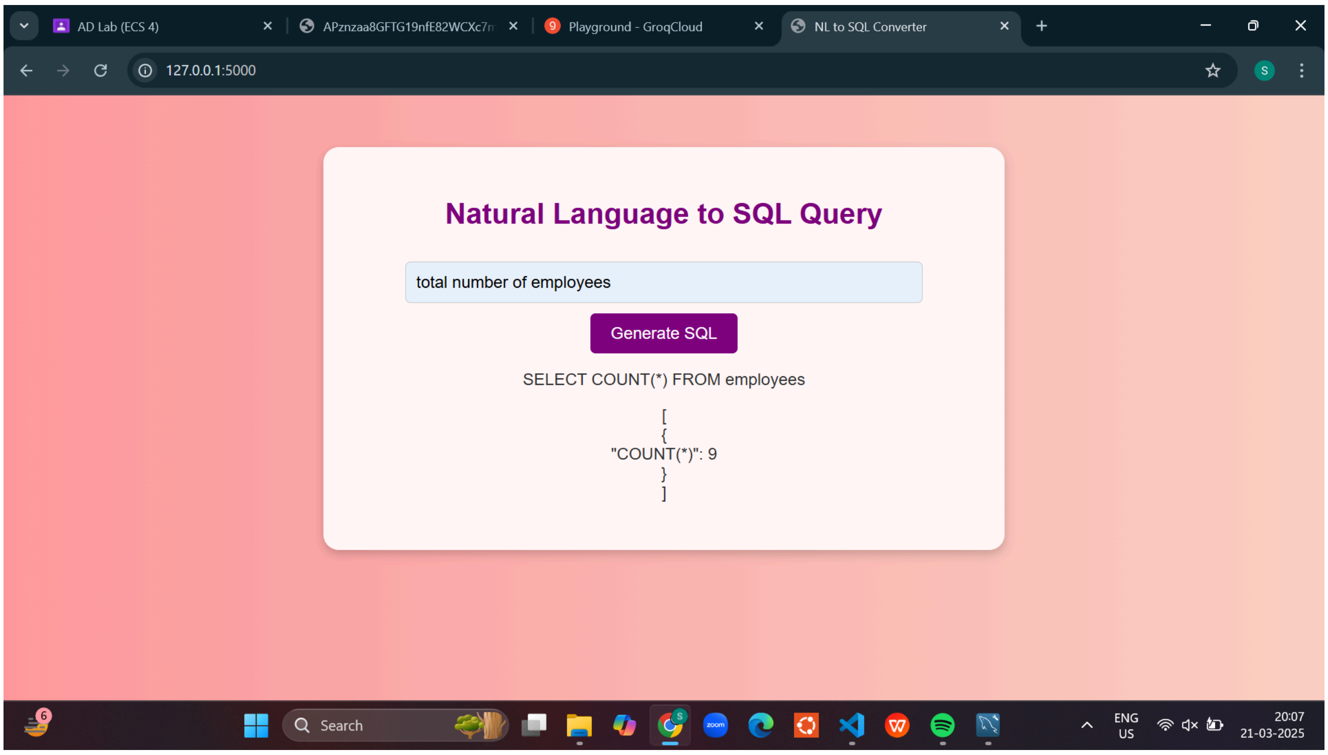
schema.sql:

```sql
CREATE DATABASE sql_ai_db;
USE sql_ai_db;

CREATE TABLE employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    department VARCHAR(100),
    salary INT
);

INSERT INTO employees (name, department, salary) VALUES
('Shreyaa Venkateswaran', 'Software Engineering', 800000),
('Akshara Prashant', 'Marketing', 500000),
('Shruti Pathak', 'HR', 600000),
('Radha Agarwal', 'Finance', 700000),
('Arjun Das', 'Management', 650000),
('Avinaash Saravanan', 'Product Development', 750000),
('Vijay Kumar', 'Marketing', 550000),
('Harris Jayaraj', 'Finance', 450000),
('Anirudh Ravi', 'Product Development', 580000);
```

4. Results/Output:



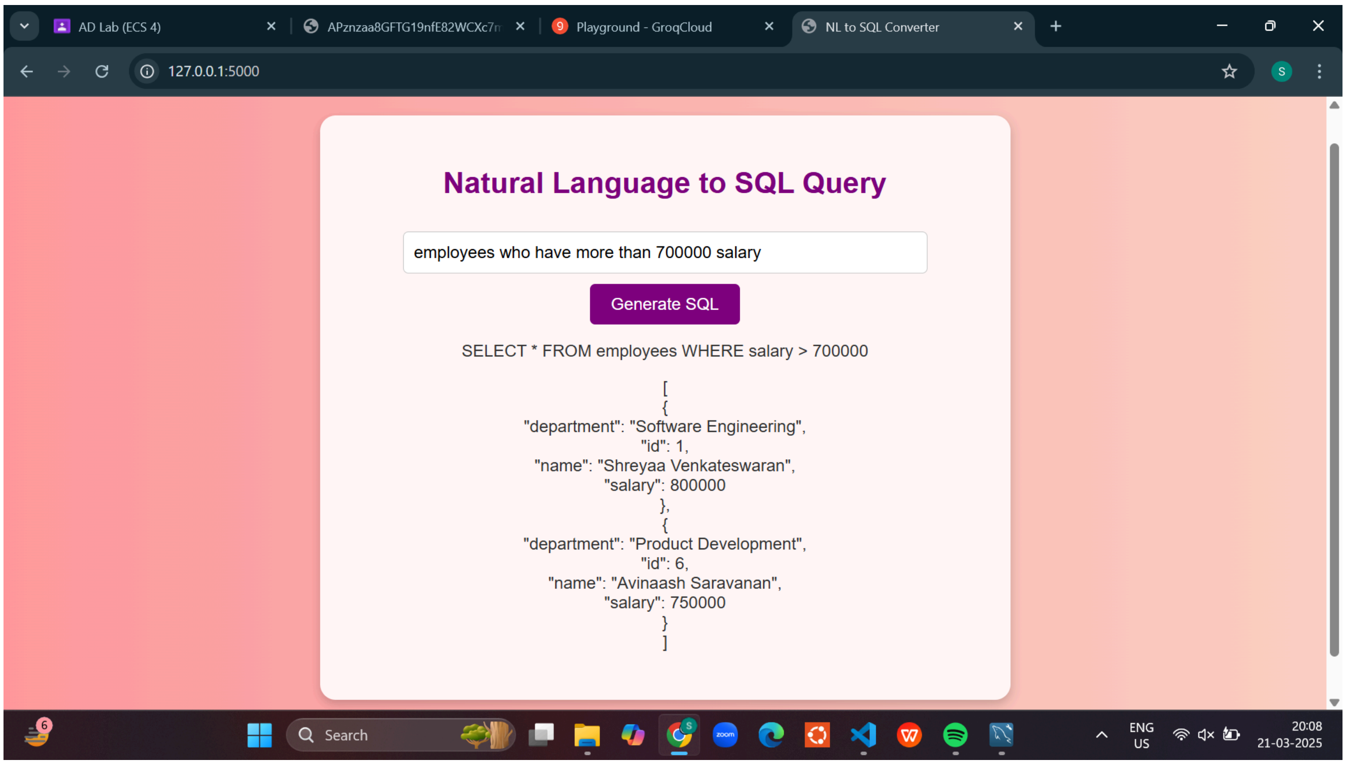
Natural Language to SQL Query

total number of employees

Generate SQL

SELECT COUNT(*) FROM employees

```
[
{
"COUNT(*)": 9
}
]
```



Natural Language to SQL Query

employees who have more than 700000 salary

SELECT * FROM employees WHERE salary > 700000

```
[
{
"department": "Software Engineering",
"id": 1,
"name": "Shreyaa Venkateswaran",
"salary": 800000
},
{
"department": "Product Development",
"id": 6,
"name": "Avinaash Saravanan",
"salary": 750000
}
]
```

5. Remarks:

Built an LLM based SQL query generator that converts natural language queries into SQL commands. The commands are then executed directly in the database to retrieve the desired results. Used Groq API and llama-3.3-70b-versatile model.

Website link: [NL_to_SQL]

GitHub link: [GitHub]

Shreyaa Venkateswaran

Signature of the Lab Coordinator