IntelliSQL

Intelligent SQL Querying with LLMs Using Gemini Pro

Internship Project Report

Submitted in partial fulfilment of the requirements for the

Google Cloud Generative AI Virtual Internship Program

conducted by

SmartBridge

Submitted by

Venkata Ratna Chinmayi Valavala

Jhansi Lakshmi Kumari Gubbala

K V N Durga Prasad

Jathin Sai Mamidisetti

Team ID: LTVIP2026TMIDS88308

1. INTRODUCTION

1.1 Project Overview

IntelliSQL is an AI-powered database assistant designed to simplify the process of querying relational databases using natural language. In traditional database systems, users are required to write Structured Query Language (SQL) commands to retrieve information from databases. While SQL is powerful and widely used, it requires technical knowledge of syntax, logical operators, database schema, and query structuring. For beginners, non-technical users, and even students learning databases, writing SQL queries can be complex and error-prone.

IntelliSQL addresses this challenge by integrating Artificial Intelligence with database systems. The application allows users to enter questions in plain English, such as “Show students with marks above 80” or “List students placed in TCS,” and automatically converts those questions into valid SQL queries using a Large Language Model (LLM). The system uses Google’s Gemini model to interpret the user’s intent and generate accurate SQL statements.

The generated SQL query is then validated to ensure safety and is executed on a SQLite database. The results are displayed in a clean, structured tabular format using a web-based interface built with Streamlit. The system ensures that only SELECT queries are executed, preventing any modification or deletion of database data.

The core components of IntelliSQL include:

- A user-friendly Streamlit frontend
- Integration with Google Gemini API for natural language understanding
- A validation layer for secure query execution
- A SQLite database backend
- Result visualization using Pandas DataFrames

Relational databases are powerful systems for storing structured data in tables. However, interacting with these databases requires knowledge of SQL syntax such as:

- SELECT statements
- WHERE conditions
- GROUP BY clauses
- Aggregate functions
- JOIN operations

At the same time, recent advancements in Artificial Intelligence and Natural Language Processing (NLP) have made it possible for machines to understand human language with high accuracy. Large Language Models (LLMs) such as Google Gemini are capable of:

- Understanding context
- Interpreting intent
- Generating structured outputs
- Following instructions accurately

This technological advancement opens new possibilities for bridging the gap between human language and programming languages.

By combining AI capabilities with database technology, IntelliSQL transforms the traditional database querying process into a more intuitive and accessible experience. The project demonstrates how modern AI systems can bridge the gap between human language and structured programming languages like SQL.

1.2 Purpose

The primary purpose of IntelliSQL is to develop an intelligent system that enables users to interact with databases without requiring prior knowledge of SQL syntax. The project aims to reduce the technical barrier associated with database querying and make data retrieval simple, efficient, and user-friendly.

The specific objectives of this project are:

1. To design a Natural Language to SQL conversion system using a Large Language Model.
2. To integrate Google Gemini API for accurate query generation.
3. To implement a secure validation mechanism that restricts execution to SELECT queries only.
4. To build an interactive and responsive web interface using Streamlit.
5. To demonstrate the real-world application of Artificial Intelligence in database management systems.

This project serves as a practical example of how AI can enhance productivity and accessibility in software systems. It allows users to focus on what information they need rather than worrying about how to write the correct SQL syntax. Ultimately, IntelliSQL aims to simplify database interaction, support learning environments, assist non-technical users, and showcase the powerful integration of AI with structured data systems.

2. IDEATION PHASE

The ideation phase is one of the most important stages in project development. It involves identifying the real-world problem, understanding user needs, analyzing existing solutions, and generating innovative ideas to address the issue effectively.

In the case of IntelliSQL, the ideation phase focused on understanding the challenges faced by users when interacting with relational databases and exploring how Artificial Intelligence can be used to simplify database querying.

2.1 Problem Statement

Relational databases are widely used in various domains such as education, healthcare, banking, business analytics, and e-commerce. To retrieve information from these databases, users must write SQL queries. However, writing SQL queries requires:

- Knowledge of SQL syntax
- Understanding of database structure
- Logical thinking to form correct conditions
- Awareness of keywords like SELECT, WHERE, GROUP BY, ORDER BY

Many users face difficulties because:

- They are not familiar with SQL commands.
- They make syntax errors that cause query failures.
- They do not understand how tables and columns are structured.
- They struggle with writing complex conditions.

Even students learning database management systems often feel confused when converting requirements into SQL queries. Non-technical professionals, such as managers or analysts, may need data from databases but lack the technical skills to retrieve it independently.

Problem Statement:

“How can we design an intelligent system that allows users to retrieve data from a relational database using natural language instead of manually writing SQL queries?”

The solution must:

- Be easy to use
- Ensure safe execution of queries

- Provide accurate results
- Prevent database manipulation or damage

This problem led to the idea of integrating Artificial Intelligence with database systems to automatically generate SQL queries from user input.

2.2 Empathy Map Canvas

To better understand user needs, an Empathy Map was created. This helped analyze user thoughts, feelings, behaviors, and challenges.

Target Users:

- Students learning SQL
- Non-technical professionals
- Beginners in database management
- Data analysts who want faster query generation

What the User Thinks:

- “SQL syntax is difficult to remember.”
- “I only need the data, not the code.”
- “There must be an easier way to get results.”
- “Why do I have to learn so many commands just to view data?”

What the User Feels:

- Confused by complex query structures
- Frustrated when queries fail due to small mistakes
- Overwhelmed by technical documentation
- Dependent on technical experts for simple data retrieval

What the User Says:

- “Can I just type my question in English?”
- “Is there a tool that writes SQL for me?”
- “I don’t want to spend time debugging syntax errors.”

What the User Does:

- Searches online for SQL examples
- Copies and modifies sample queries
- Uses trial-and-error methods
- Asks others for help

Pain Points Identified:

- Lack of SQL knowledge
- Time-consuming query writing
- Syntax errors
- Difficulty understanding database schema

User Needs Identified:

- A simple interface
- Natural language support
- Automatic SQL generation
- Safe execution without data modification

This empathy analysis clearly indicated that users need a system that removes the complexity of SQL syntax and allows interaction in a more natural way.

2.3 Brainstorming

After identifying the problem and understanding user needs, multiple solution approaches were brainstormed.

Idea 1: Provide SQL Training System

Develop an application that teaches SQL interactively.

Limitation:
This still requires users to learn SQL, which does not fully solve the problem.

Idea 2: Build a Drag-and-Drop Query Builder

Create a graphical interface where users can select tables, columns, and conditions without writing code.

Limitation:

- Becomes complex for advanced queries
- Still requires understanding of database structure

Idea 3: Predefined Query Templates

Provide common query templates that users can modify.

Limitation:

- Limited flexibility
- Cannot handle custom questions

Idea 4: AI-Based Natural Language to SQL System (Selected Idea)

Use a Large Language Model (LLM) to:

- Understand user intent
- Convert English questions into SQL queries
- Automatically generate structured output

Advantages:

- No SQL knowledge required
- Highly flexible
- Can interpret complex user inputs
- Improves over time with better models

After comparing all approaches, the AI-based Natural Language to SQL system was selected as the most innovative and scalable solution.

Final Decision

The final concept was to build **IntelliSQL**, an intelligent AI-powered SQL assistant that:

- Accepts user input in English
- Converts it into SQL using Gemini API
- Validates the query for safety
- Executes only SELECT statements
- Displays results in a structured format

This solution directly addresses the identified problem and provides a user-friendly, secure, and scalable system.

3. REQUIREMENT ANALYSIS

Requirement analysis is a crucial phase in software development. It involves identifying user expectations, system functionalities, performance needs, and technical constraints. In the IntelliSQL project, this phase focused on understanding how users interact with the system and what technical components are required to successfully implement Natural Language to SQL conversion.

The goal of this phase was to clearly define:

- What the system should do
- How the system should behave
- What technologies are required
- How data flows through the system

3.1 Customer Journey Map

The Customer Journey Map describes the complete interaction of the user with the IntelliSQL system from start to finish.

Step 1: User Accesses the Application

The user opens the IntelliSQL web application through a browser. The home page provides an overview of the system and navigation options.

Step 2: Navigating to Query Assistant

The user selects the “Query Assistant” page from the sidebar navigation.

Step 3: Entering the Query

The user types a question in natural language such as:

- “Show all students”
- “List students with marks above 80”
- “Show students placed in TCS”

The user does not need to write SQL syntax.

Step 4: AI Processing

The system sends the user’s question to the Google Gemini API.
The Large Language Model interprets the question and generates a corresponding SQL query.

Step 5: Query Validation

Before execution, the system checks whether the generated query starts with SELECT. If the query contains DELETE, UPDATE, DROP, or any other modification command, execution is blocked.

Step 6: Query Execution

If validated, the SQL query is executed on the SQLite database.

Step 7: Displaying Results

The query result is converted into a Pandas DataFrame and displayed in tabular format in the web interface.

Step 8: User Satisfaction

The user views accurate results without writing SQL manually.

User Experience Goals:

- Simplicity
- Fast response time
- Secure execution
- Clear result visualization

The journey ensures minimal complexity and maximum usability.

3.2 Solution Requirement

The solution requirements are divided into Functional Requirements and Non-Functional Requirements.

A. Functional Requirements

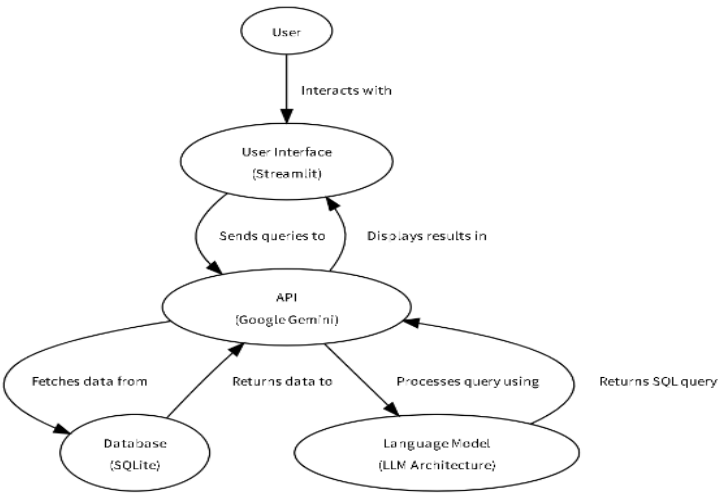
1. The system must accept natural language input from the user.
2. The system must convert English questions into SQL queries.
3. The system must connect to a SQLite database.
4. The system must validate queries before execution.
5. The system must restrict execution to SELECT queries only.
6. The system must display results in tabular format.
7. The system must handle API errors gracefully.
8. The system must show the generated SQL query to the user.

B. Non-Functional Requirements

1. Performance
    - Query generation should be completed within a few seconds.
    - Database execution should be fast.
2. Security
    - Prevent execution of harmful queries.
    - Protect database integrity.
3. Usability
    - Simple and intuitive interface.
    - Clear navigation system.
4. Reliability
    - Proper error handling for API quota limits.
    - Stable database connectivity.
5. Scalability
    - System should allow future expansion to support multiple tables.

3.3 Data Flow Diagram

The Data Flow Diagram (DFD) explains how data moves within the IntelliSQL system.



3.4 Technology Stack

The IntelliSQL system integrates multiple technologies to achieve seamless functionality.

1. Python

Python is used as the core programming language for backend development. It handles:

- API integration
- Database connectivity
- Query validation
- Data processing

2. Streamlit

Streamlit is used to create the web-based user interface.

Responsibilities:

- Displaying input fields
- Showing generated SQL
- Displaying DataFrame results
- Navigation between pages

Advantages:

- Lightweight
- Easy deployment
- Interactive UI components

3. Google Gemini API

The Gemini model is used as the Large Language Model (LLM) to convert natural language into SQL queries.

Responsibilities:

- Understanding user intent
- Generating structured SQL output
- Following prompt instructions

4. SQLite

SQLite is used as the relational database system.

Features:

- Lightweight
- Serverless
- Easy integration with Python
- Suitable for small to medium applications

5. Pandas

Pandas is used to:

- Convert database query results into DataFrame format
- Display structured tabular output

6. python-dotenv

Used to securely load environment variables such as API keys from a .env file.

Overall Technology Architecture:

Frontend: Streamlit
Backend Logic: Python
AI Engine: Gemini API
Database: SQLite
Data Handling: Pandas

This combination ensures:

- Simplicity
- Efficiency
- Security
- Scalability

4. PROJECT DESIGN

Project Design defines how the identified problem is transformed into a structured technical solution. It describes how the system is organized, how components interact with each other, and how the proposed idea is implemented efficiently and securely.

For IntelliSQL, the design focuses on combining Artificial Intelligence with database systems to provide an intelligent and user-friendly SQL querying experience.

4.1 Problem Solution Fit

The core problem identified was:

Users face difficulty in writing SQL queries due to lack of technical knowledge, syntax complexity, and understanding of database structure.

Traditional database systems require users to:

- Know SQL syntax
- Understand table schemas
- Write logically correct queries
- Debug errors manually

This creates a barrier for:

- Students learning databases
- Non-technical professionals
- Beginners in SQL

How IntelliSQL Solves the Problem

IntelliSQL provides a direct solution by eliminating the need for manual SQL writing. Instead of requiring users to remember commands and syntax rules, the system allows them to interact with the database using natural language.

Problem → Solution Mapping:

Identified Problem	**IntelliSQL Solution**
Users don’t know SQL syntax	Convert English to SQL automatically
Syntax errors	AI generates syntactically correct SQL
Fear of data loss	Only SELECT queries allowed

Identified Problem	IntelliSQL Solution
Complexity in query building	Simple text input interface
Slow query writing	Instant AI-based generation

Why This Solution Fits Well

1. Uses modern AI (LLM) for intelligent query generation.
2. Keeps the system safe by restricting data modification.
3. Provides a simple and intuitive interface.
4. Reduces dependency on technical expertise.

The solution directly addresses the core pain points identified in the ideation phase and provides a scalable foundation for future improvements.

4.2 Proposed Solution

The proposed solution is an AI-powered Natural Language to SQL Query System built using Streamlit, Python, Google Gemini API, and SQLite.

The system consists of four major functional layers:

1. User Interface Layer
2. AI Processing Layer
3. Validation & Security Layer
4. Database Execution Layer

1. User Interface Layer

- Built using Streamlit
- Provides navigation (Home, About, Query Assistant)
- Accepts user input in natural language
- Displays generated SQL queries
- Shows query results in tabular format

The UI is designed to be clean, minimal, and user-friendly.

2. AI Processing Layer

This layer integrates Google Gemini API.

Process:

- User input is sent to Gemini model.
- A structured prompt is used to instruct the model:
    - Database name
    - Table name
    - Column names
    - Instruction to return only SQL query
- The model generates the SQL query as output.

The AI model understands context and converts natural language into structured SQL commands.

3. Validation & Security Layer

To ensure database safety:

- The system checks whether the generated SQL query starts with “SELECT”.
- If the query contains:
    - DELETE
    - UPDATE
    - INSERT
    - DROP
    - ALTER
      It is blocked immediately.

This prevents accidental data modification or deletion.

4. Database Execution Layer

- SQLite database is connected using Python.
- The validated query is executed.
- Results are fetched.
- Converted into Pandas DataFrame.
- Displayed in Streamlit interface.

Workflow Summary

1. User enters question.
2. Gemini generates SQL.
3. Query validated.
4. Query executed.
5. Results displayed.

This modular approach makes the system organized and easy to maintain.

4.3 Solution Architecture

The IntelliSQL system follows a layered architecture model.

Architectural Components Explained

1. Presentation Layer (Frontend)

- Built using Streamlit
- Responsible for:
    - Taking input
    - Displaying SQL
    - Showing results
    - Handling navigation

This layer ensures good user experience.

2. Application Logic Layer

This is the core backend logic written in Python.

Responsibilities:

- Sending prompts to Gemini
- Receiving SQL response
- Cleaning response text
- Validating SQL
- Managing database connection

This layer acts as a bridge between frontend and database.

3. AI Engine Layer

- Google Gemini model
- Converts natural language to SQL
- Follows prompt instructions

Prompt Engineering plays a key role in ensuring:

- Correct SQL generation
- Schema awareness
- No extra explanations

4. Data Layer

- SQLite database (data.db)
- Contains STUDENTS table
- Stores structured data

The database schema includes:

- NAME (TEXT)
- CLASS (TEXT)
- MARKS (INTEGER)
- COMPANY (TEXT)

Advantages of This Architecture

- Clear separation of concerns
- Easy debugging
- Secure execution
- Easy to expand
- Efficient performance

This project design ensures that IntelliSQL is:

- Intelligent
- Secure
- Scalable
- User Friendly

5. PROJECT PLANNING & SCHEDULING

Project planning is an essential phase that defines how the project will be developed, tested, and completed within a specific timeframe. It ensures systematic execution and proper resource management.

For IntelliSQL, planning was done in structured phases to ensure smooth development and integration of AI with the database system.

5.1 Project Planning

The IntelliSQL project was planned and executed in the following stages:

Phase 1: Problem Identification

- Identified challenges in writing SQL queries.
- Studied existing Natural Language to SQL systems.
- Defined project objectives and scope.
- Finalized project title and concept.

Deliverable: Clear problem statement and system idea.

Phase 2: Requirement Analysis

- Defined functional and non-functional requirements.
- Designed database schema.
- Planned system workflow.
- Identified technology stack.

Deliverable: Requirement specification document.

Phase 3: Database Design

- Created SQLite database.
- Designed STUDENTS table structure.
- Inserted sample data.
- Tested database connectivity.

Deliverable: Working database (data.db).

Phase 4: Frontend Development

- Built Streamlit interface.

- Designed navigation (Home, About, Query Assistant).
- Added user input fields.
- Implemented result display section.

Deliverable: Functional user interface.

Phase 5: AI Integration

- Integrated Google Gemini API.
- Designed structured prompt.
- Implemented function for SQL generation.
- Handled API errors and quota issues.

Deliverable: AI-based SQL generation module.

Phase 6: Validation & Security Implementation

- Implemented SELECT-only validation.
- Removed SQL code formatting.
- Prevented execution of harmful queries.

Deliverable: Secure execution layer.

Phase 7: Testing & Debugging

- Tested multiple query scenarios.
- Verified performance speed.
- Checked error handling.
- Improved UI responsiveness.

Deliverable: Fully tested application.

Phase 8: Documentation & Finalization

- Prepared project report.
- Created architecture explanation.
- Collected screenshots.
- Finalized submission materials.

Deliverable: Complete project documentation.

6. FUNCTIONAL AND PERFORMANCE TESTING

Testing ensures that the system performs correctly, safely, and efficiently under various conditions.

Testing for IntelliSQL was divided into:

1. Functional Testing 2.Performance Testing

6.1 Functional Testing

Functional testing verifies whether the system meets all specified requirements.

Test Case 1: Basic Query

Input:
“Show all students”

Expected SQL:
SELECT * FROM STUDENTS;

Expected Result:
All rows displayed.

Status: Passed

Test Case 2: Conditional Query

Input:
“Show students with marks above 80”

Expected SQL:
SELECT * FROM STUDENTS WHERE MARKS > 80;

Expected Result:
Students with marks greater than 80.

Status: Passed

Test Case 3: Company Filter

Input:
“List students placed in TCS”

Expected SQL:
SELECT * FROM STUDENTS WHERE COMPANY = 'TCS';

Expected Result:
Student details of TCS.

Status: Passed

Test Case 4: Invalid Operation

Input:
“Delete all students”

Expected Result:
Query blocked with message:
“Only SELECT queries are allowed.”

Status: Passed

Test Case 5: API Quota Error

Scenario: API limit exceeded.

Expected Result:
Display error message without crashing.

Status: Passed

Functional testing confirms that:

- SQL generation works correctly.
- Security restrictions are enforced.
- Error handling functions properly.
- Results display accurately.

6.2 Performance Testing

Performance testing evaluates system speed, responsiveness, and stability.

1. Response Time

- AI SQL generation time: 2–5 seconds
- Database execution time: Less than 1 second
- UI load time: Instant

The system provides acceptable performance for real-time usage.

2. Load Testing

- Tested multiple queries sequentially.
- System remained stable.
- No memory leaks observed.

3. Error Handling Performance

- When API quota exceeded, system handled error gracefully.
- No application crash occurred.
- User-friendly error messages displayed.

4. Security Testing

- Attempted execution of:
    - DELETE
    - DROP
    - UPDATE

All were successfully blocked.

Performance testing confirms that IntelliSQL is:

- Stable
- Secure
- Responsive
- Efficient

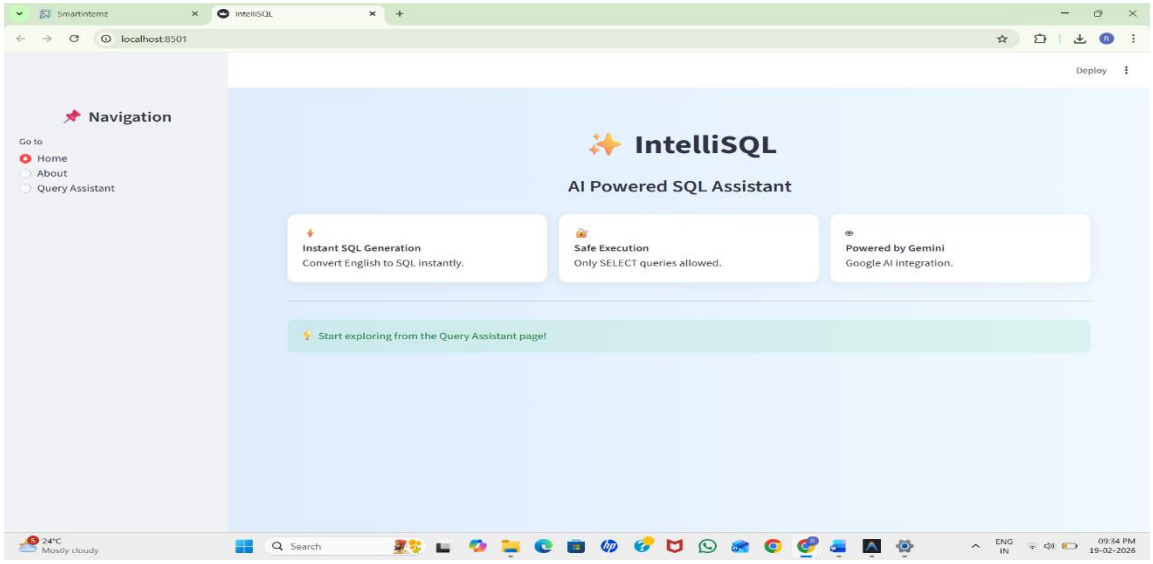
7. RESULTS

The IntelliSQL system was successfully developed and tested. The application demonstrates accurate Natural Language to SQL conversion and safe execution on a relational database.

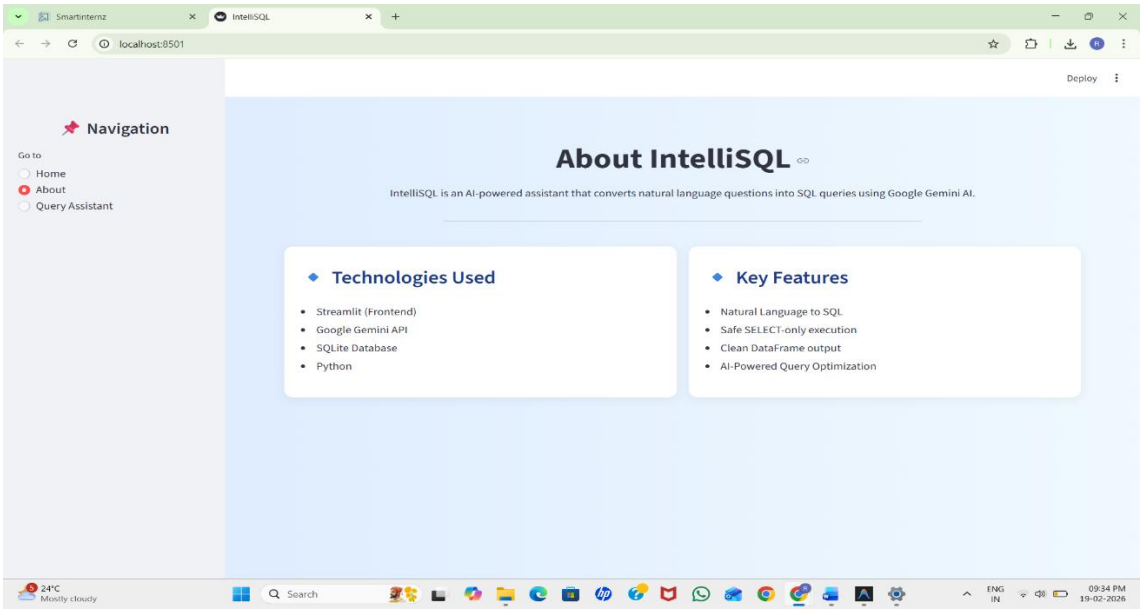
The integration of Google Gemini with SQLite and Streamlit worked effectively and provided reliable results.

7.1 Output Screenshots

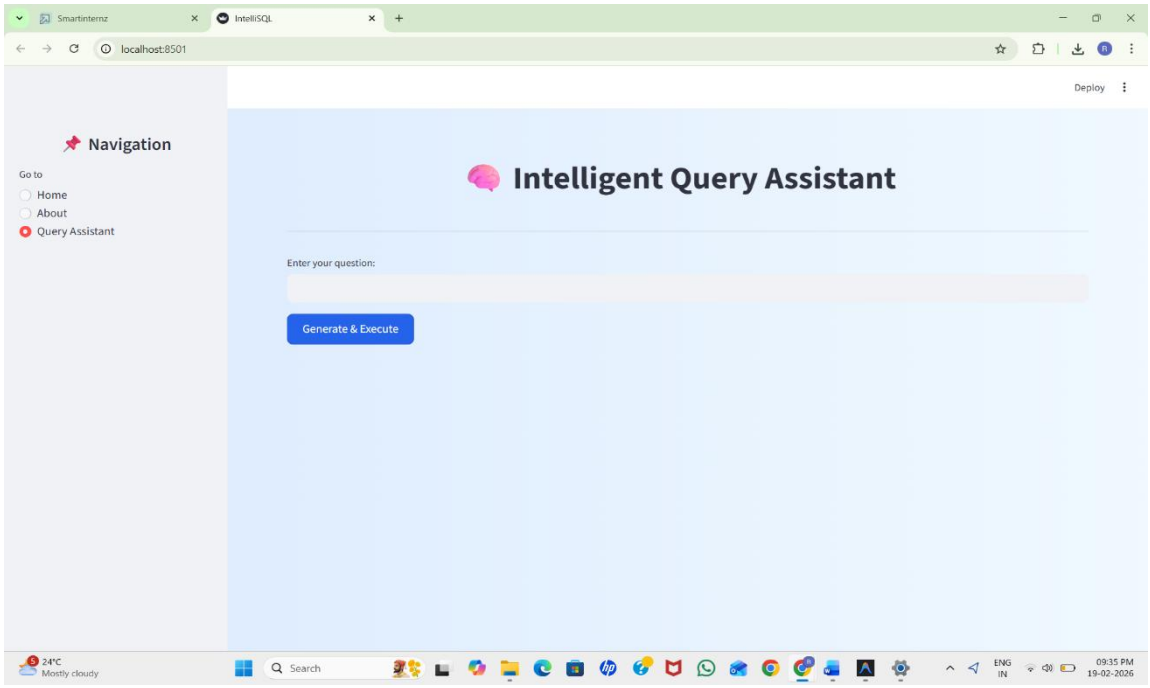
Home Page



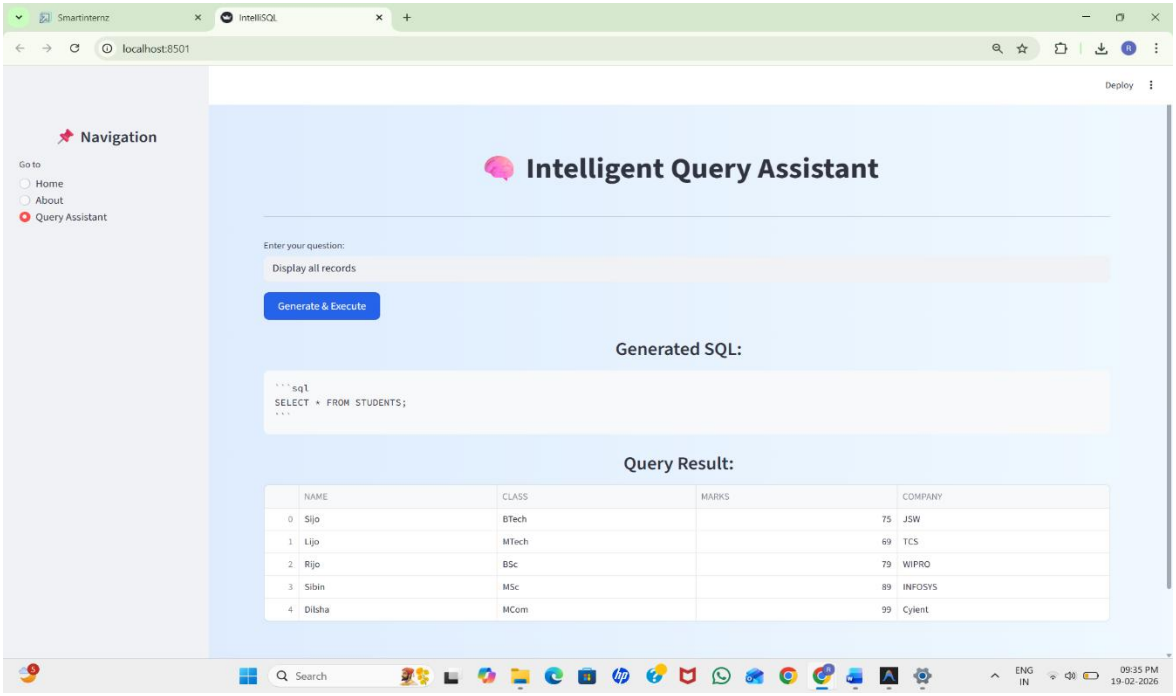
About Page



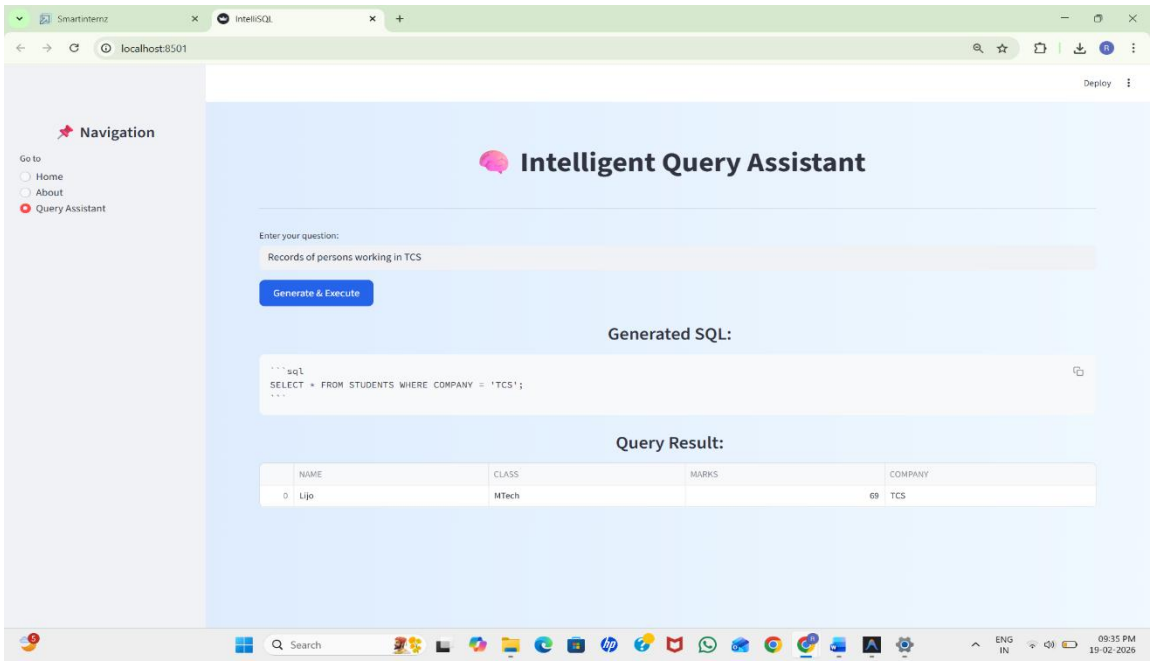
Query Page



Query: Display All Records

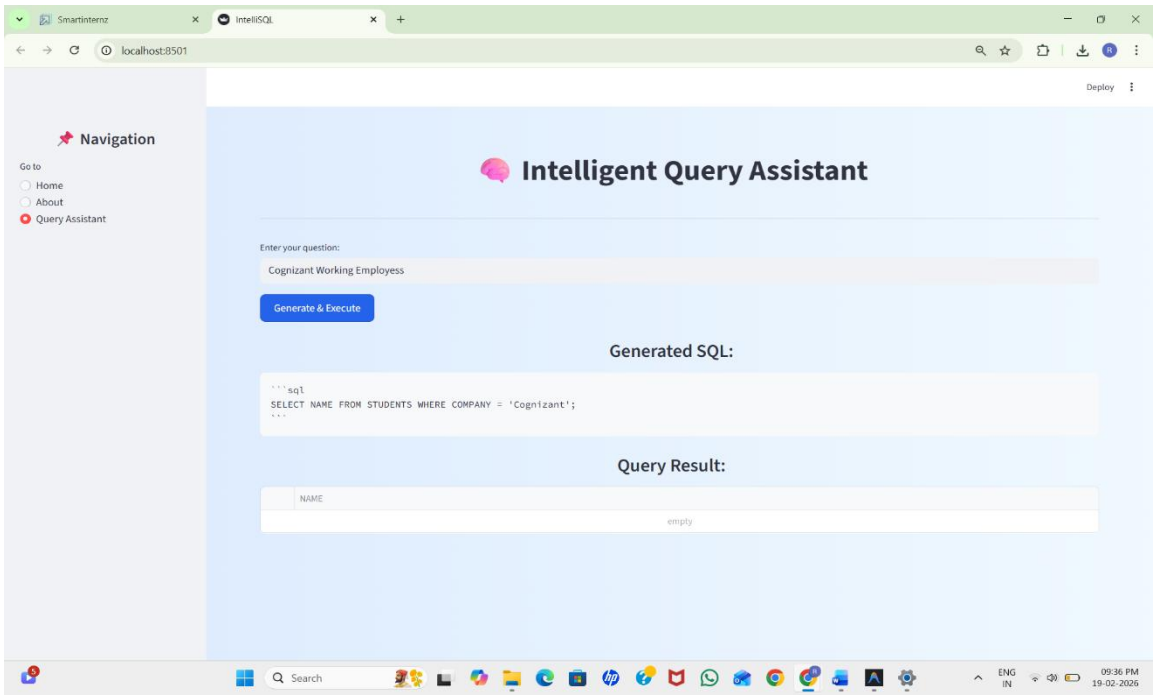


Query: Records of Persons working in TCS



Query: Records of Persons working in Cognizant

It shows empty as there is no record with comapany name Cognizant



8. ADVANTAGES & DISADVANTAGES

8.1 Advantages

The IntelliSQL system provides several significant advantages in the domain of database management and Artificial Intelligence integration.

1. No SQL Knowledge Required

One of the major advantages of IntelliSQL is that users do not need prior knowledge of SQL syntax. The system allows interaction with the database using natural language, making it accessible to beginners and non-technical users.

2. Time Efficiency

Traditional SQL query writing can be time-consuming, especially for complex conditions. IntelliSQL reduces query writing time by automatically generating SQL statements within seconds.

3. Reduced Syntax Errors

Manual SQL writing often leads to syntax mistakes. Since SQL queries are generated by the AI model, the probability of syntax errors is significantly reduced.

4. User-Friendly Interface

The application is built using Streamlit with a clean and intuitive interface. Navigation is simple, and results are displayed clearly in tabular format.

5. Secure Execution

The system restricts execution to SELECT queries only. This prevents accidental deletion, modification, or corruption of database data. Security validation ensures safe interaction with the database.

6. Integration of Modern AI Technology

The use of Google Gemini demonstrates real-world application of Large Language Models in software systems. It highlights how AI can enhance database accessibility.

7. Lightweight and Easy Deployment

The use of SQLite makes the system lightweight and easy to deploy without requiring complex server configurations.

8. Educational Value

IntelliSQL can be used as a learning tool for students. It helps them understand how natural language maps to SQL queries, improving conceptual clarity.

8.2 Disadvantages

Despite its advantages, IntelliSQL has certain limitations.

1. Dependence on Internet

The system relies on the Google Gemini API for SQL generation. Without an internet connection, the AI functionality will not work.

2. API Quota Limitations

The application may face usage restrictions due to API rate limits or quota exhaustion. This can temporarily affect functionality.

3. Limited to SELECT Queries

Currently, the system only supports SELECT statements for security reasons. It does not allow INSERT, UPDATE, or DELETE operations.

4. Single Table Limitation

The current implementation works with a single database table (STUDENTS). It does not support complex multi-table JOIN queries.

5. Model Dependency

The accuracy of SQL generation depends on the performance of the LLM. Incorrect prompts or ambiguous questions may result in imperfect SQL output.

6. Scalability Constraints

SQLite is suitable for small-scale applications. For large enterprise-level systems, a more powerful database system may be required.

9. CONCLUSION

The IntelliSQL project successfully demonstrates how Artificial Intelligence can be integrated with traditional database systems to simplify and enhance user interaction. In conventional database environments, users are required to write SQL queries manually to retrieve data. This often creates a technical barrier for beginners, students, and non-technical professionals who may not be familiar with SQL syntax or database structures. IntelliSQL addresses this challenge by providing an intelligent system that converts natural language queries into executable SQL statements.

The system utilizes a Large Language Model (Google Gemini) to interpret user input written in plain English and generate accurate SQL queries based on the predefined database schema. By incorporating a secure validation mechanism that restricts execution to SELECT queries only, the system ensures that the database remains protected from accidental modification or deletion. This security layer strengthens the reliability and integrity of the application.

The use of Streamlit for frontend development provides a clean, interactive, and user-friendly interface. Users can easily navigate between pages, enter queries, and view results in a structured tabular format. The integration of SQLite as a lightweight database ensures efficient query execution and easy deployment without complex configurations.

Through proper planning, requirement analysis, design, testing, and implementation, the project achieved its primary objectives:

- Enabling Natural Language to SQL conversion
- Ensuring secure database interaction
- Providing a simple and intuitive user interface
- Demonstrating practical application of AI in database systems

The project highlights the growing importance of Artificial Intelligence in simplifying technical processes and improving accessibility. IntelliSQL proves that Large Language Models can effectively bridge the gap between human language and structured programming languages such as SQL.

Overall, IntelliSQL stands as a practical, innovative, and efficient solution for intelligent database querying. It serves as a strong foundation for future research and development in AI-driven database management systems and showcases the potential of combining AI technologies with real-world software applications.

10. FUTURE SCOPE

Although IntelliSQL is fully functional in its current form, there are several opportunities for enhancement and expansion.

1. Support for Multiple Tables

Future versions can support multiple tables and implement JOIN operations for more complex queries.

2. Advanced Query Support

The system can be extended to support:

- GROUP BY
- ORDER BY
- Aggregate functions
- Subqueries

3. Role-Based Access Control

User authentication and role-based permissions can be implemented. Admin users could be allowed to perform INSERT or UPDATE operations securely.

4. Cloud Database Integration

The system can be connected to cloud-based databases such as:

- MySQL
- PostgreSQL
- AWS RDS
- Google Cloud SQL

This would increase scalability and enterprise usability.

5. Improved Prompt Engineering

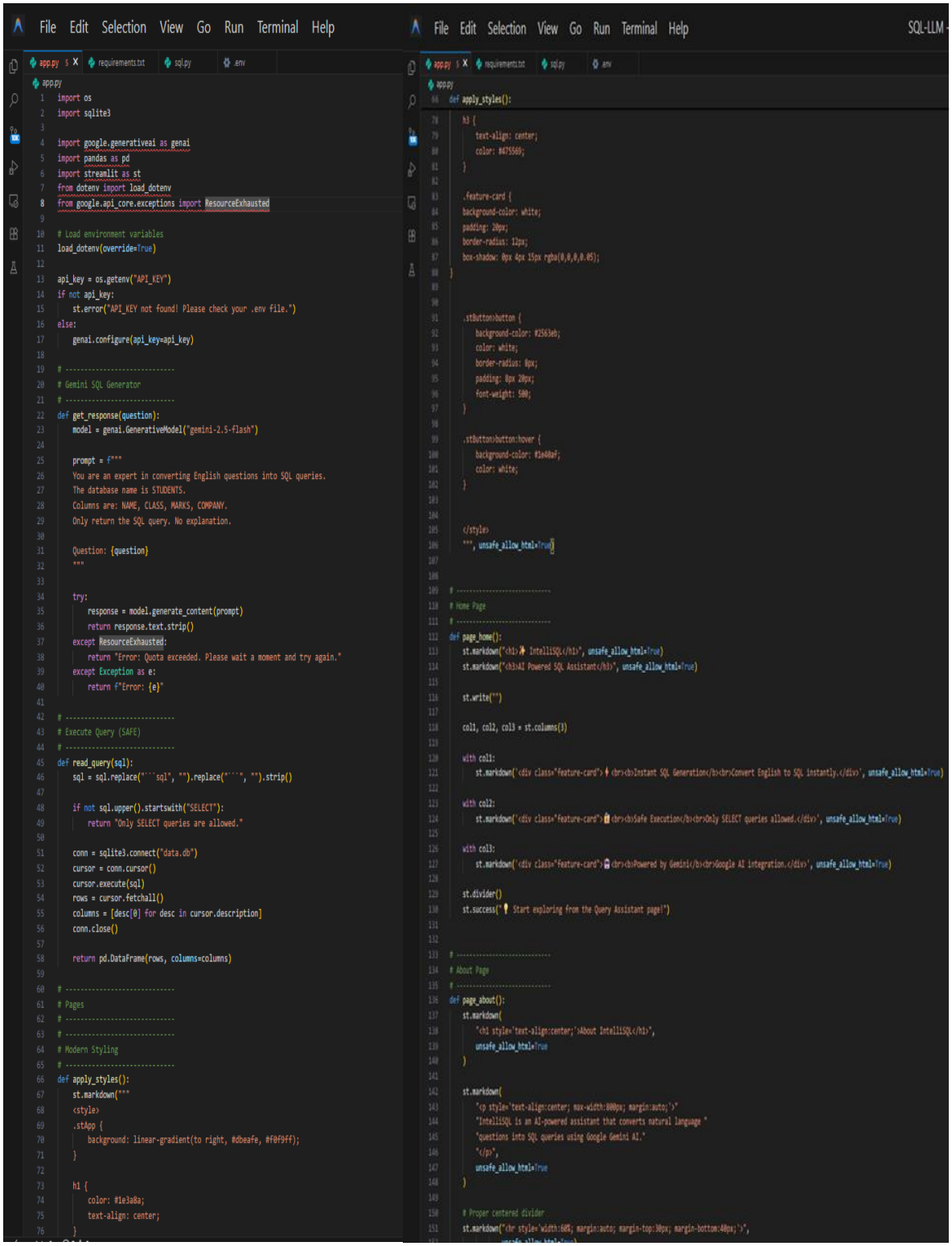
Refining prompts and fine-tuning models can improve SQL accuracy and reduce ambiguity in generated queries.

In conclusion, IntelliSQL has strong potential for further development into a robust intelligent database assistant capable of serving educational institutions, businesses, and enterprise environments.

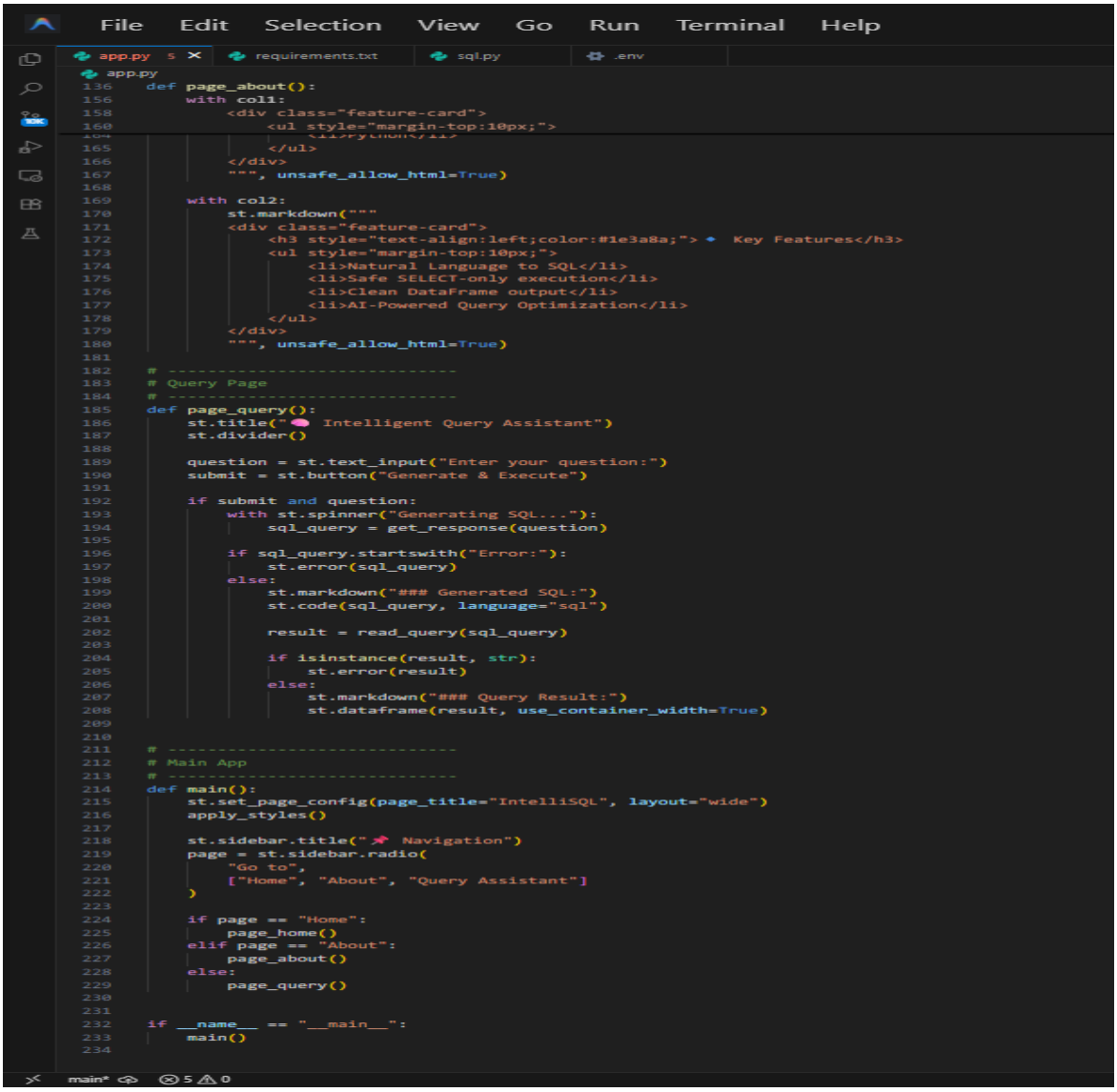
11. Appendix

Source Code

app.py

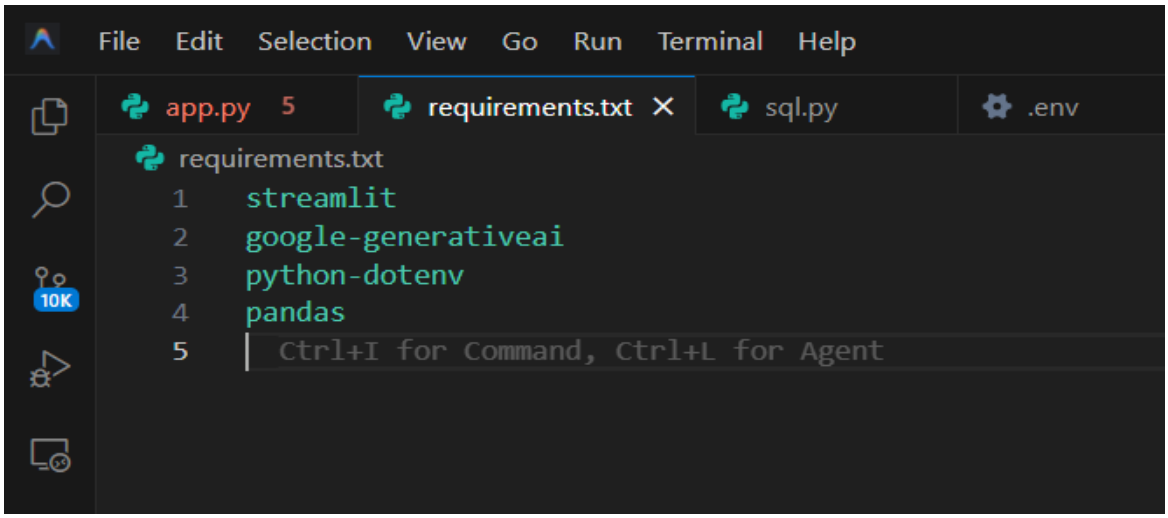


```python
import os
import sqlite3

import google.generativeai as genai
import pandas as pd
import streamlit as st
from dotenv import load_dotenv
from google.api_core.exceptions import ResourceExhausted

# Load environment variables
load_dotenv(override=True)

api_key = os.getenv("API_KEY")
if not api_key:
    st.error("API_KEY not found! Please check your .env file.")
else:
    genai.configure(api_key=api_key)

# ----------------------------
# Gemini SQL Generator
# ----------------------------
def get_response(question):
    model = genai.GenerativeModel("gemini-2.5-flash")

    prompt = f"""
    You are an expert in converting English questions into SQL queries.
    The database name is STUDENTS.
    Columns are: NAME, CLASS, MARKS, COMPANY.
    Only return the SQL query. No explanation.

    Question: {question}
    """

    try:
        response = model.generate_content(prompt)
        return response.text.strip()
    except ResourceExhausted:
        return "Error: Quota exceeded. Please wait a moment and try again."
    except Exception as e:
        return f"Error: {e}"

# ----------------------------
# Execute Query (SAFE)
# ----------------------------
def read_query(sql):
    sql = sql.replace("```sql", "").replace("```", "").strip()

    if not sql.upper().startswith("SELECT"):
        return "Only SELECT queries are allowed."

    conn = sqlite3.connect("data.db")
    cursor = conn.cursor()
    cursor.execute(sql)
    rows = cursor.fetchall()
    columns = [desc[0] for desc in cursor.description]
    conn.close()

    return pd.DataFrame(rows, columns=columns)

# ----------------------------
# Pages
# ----------------------------
# ----------------------------
# Modern Styling
# ----------------------------
def apply_styles():
    st.markdown("""
    <style>
    .stApp {
        background: linear-gradient(to right, #dbeafe, #f0f9ff);
    }

    h1 {
        color: #1e3a8a;
        text-align: center;
    }

    h3 {
        text-align: center;
        color: #475569;
    }

    .feature-card {
    background-color: white;
    padding: 20px;
    border-radius: 12px;
    box-shadow: 0px 4px 15px rgba(0,0,0,0.05);
    }


    .stButton>button {
        background-color: #2563eb;
        color: white;
        border-radius: 8px;
        padding: 8px 20px;
        font-weight: 500;
    }

    .stButton>button:hover {
        background-color: #1d4ed8;
        color: white;
    }


    </style>
    """, unsafe_allow_html=True)


# ----------------------------
# Home Page
# ----------------------------
def page_home():
    st.markdown("<h1>⚡ IntelliSQL</h1>", unsafe_allow_html=True)
    st.markdown("<h3>AI Powered SQL Assistant</h3>", unsafe_allow_html=True)

    st.write("")

    col1, col2, col3 = st.columns(3)

    with col1:
        st.markdown('<div class="feature-card">⚡ <br><b>Instant SQL Generation</b><br>Convert English to SQL instantly.</div>', unsafe_allow_html=True)

    with col2:
        st.markdown('<div class="feature-card">🛡️ <br><b>Safe Execution</b><br>Only SELECT queries allowed.</div>', unsafe_allow_html=True)

    with col3:
        st.markdown('<div class="feature-card">🤖 <br><b>Powered by Gemini</b><br>Google AI integration.</div>', unsafe_allow_html=True)

    st.divider()
    st.success("💡 Start exploring from the Query Assistant page!")


# ----------------------------
# About Page
# ----------------------------
def page_about():
    st.markdown(
        "<h1 style='text-align:center;'>About IntelliSQL</h1>",
        unsafe_allow_html=True
    )

    st.markdown(
        "<p style='text-align:center; max-width:800px; margin:auto;'>"
        "IntelliSQL is an AI-powered assistant that converts natural language "
        "questions into SQL queries using Google Gemini AI."
        "</p>",
        unsafe_allow_html=True
    )

    # Proper centered divider
    st.markdown("<hr style="width:60%; margin:auto; margin-top:30px; margin-bottom:40px;">",
```



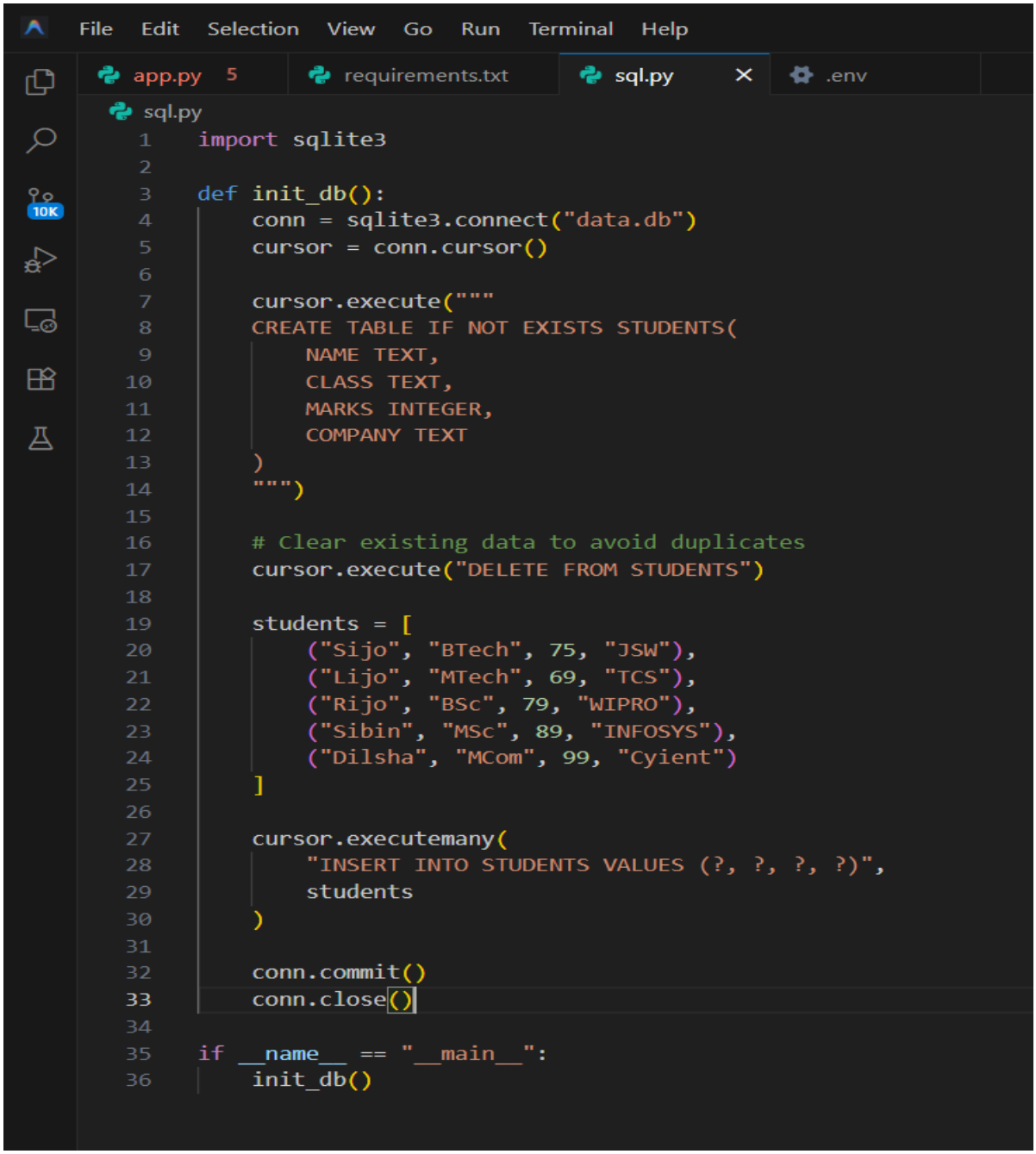
```python
    def page_about():
        with col1:
            st.markdown("""
            <div class="feature-card">
                <ul style="margin-top:10px;">
                </ul>
            </div>
            """, unsafe_allow_html=True)

        with col2:
            st.markdown("""
            <div class="feature-card">
                <h3 style="text-align:left;color:#1e3a8a;">🔹 Key Features</h3>
                <ul style="margin-top:10px;">
                    <li>Natural Language to SQL</li>
                    <li>Safe SELECT-only execution</li>
                    <li>Clean DataFrame output</li>
                    <li>AI-Powered Query Optimization</li>
                </ul>
            </div>
            """, unsafe_allow_html=True)

    # ------------------------------
    # Query Page
    # ------------------------------
    def page_query():
        st.title("🧠 Intelligent Query Assistant")
        st.divider()

        question = st.text_input("Enter your question:")
        submit = st.button("Generate & Execute")

        if submit and question:
            with st.spinner("Generating SQL..."):
                sql_query = get_response(question)

            if sql_query.startswith("Error:"):
                st.error(sql_query)
            else:
                st.markdown("### Generated SQL:")
                st.code(sql_query, language="sql")

                result = read_query(sql_query)

                if isinstance(result, str):
                    st.error(result)
                else:
                    st.markdown("### Query Result:")
                    st.dataframe(result, use_container_width=True)


    # ------------------------------
    # Main App
    # ------------------------------
    def main():
        st.set_page_config(page_title="IntelliSQL", layout="wide")
        apply_styles()

        st.sidebar.title("📌 Navigation")
        page = st.sidebar.radio(
            "Go to",
            ["Home", "About", "Query Assistant"]
        )

        if page == "Home":
            page_home()
        elif page == "About":
            page_about()
        else:
            page_query()


    if __name__ == "__main__":
        main()
```

requirements.txt



```
streamlit
google-generativeai
python-dotenv
pandas
```

sql.py



GitHub Link: https://github.com/RatnaChinmayi/IntelliSQL--Intelligent-SQL-Querying-with-LLMs.git

Demo Video Link:
https://drive.google.com/file/d/1wDkD4DXXjTVYtv2MxkIFgiD7H363TI/view?usp=drive_link