# Project Documentation - IntelliSQL

**Names:** B.Rohith Kumar,22BCT0330(VIT Vellore)
K.Geethika Varshini,22BCB0001(VIT Vellore)
M.Bhuvan Chowdary,22BKT0068(VIT Vellore)
N.Revanth Reddy,22BCE3158(VIT Vellore)

- **Table of Contents**

- **1. Introduction**

### 1.1 Purpose

The main objective of this document is to define and explain the requirements of the IntelliSQL project—a web-based platform that translates natural English queries into SQL commands using Google Gemini and executes them against a local SQLite database. This project is designed for educational, experimental, and research purposes to demonstrate the power of large language models (LLMs) when combined with structured databases.

### 1.2 Intended Audience

This document serves as a comprehensive guide for software developers, testers, system designers, AI researchers, project managers, and academic evaluators who are directly or indirectly involved in the conception, development, deployment, and testing of intelligent data querying systems.

### 1.3 Scope of Development Project

The IntelliSQL project addresses the need to simplify database querying for users without knowledge of SQL. The system's core functionality lies in converting human language into SQL and instantly providing answers. It will support interactive and responsive querying, future integration of multiple tables, user authentication, access control, and possibly cloud-based deployment for enterprise scalability.

## 1.4 Definition, Acronyms, Abbreviations

SQL – Structured Query Language

LLM – Large Language Model

DBMS – Database Management System

AI – Artificial Intelligence

UI – User Interface

NLP – Natural Language Processing

SQLite – Lightweight Embedded SQL Database

Gemini – Google’s Generative AI API

## 1.5 References

https://streamlit.io

https://ai.google.dev/gemini-api

https://sqlite.org

https://www.w3schools.com/sql/

https://www.geeksforgeeks.org/sql-tutorial/

- **2. Overall Description**

### 2.1 Product Perspective

IntelliSQL is a single-page web application that enhances user experience in querying relational databases. It uses a prompt-based design powered by Gemini API to interpret the user's intention and respond with the correct SQL statement. The SQL result is returned dynamically in a visually clean table.

### 2.2 Product Functions

- **Input:** Accept English query from the user
- **Processing:** Send query to Gemini API for SQL conversion
- **Execution:** Run the SQL on SQLite database
- **Output:** Display result in tabular format on Streamlit UI
- **Error Handling:** Provide friendly messages if syntax or logic errors occur

### 2.3 User Classes and Characteristics

- **Students**: Non-technical users querying academic datasets
- **Developers**: Test and build features or integrate with more complex DBs
- **Faculty**: Evaluate students' work, extend for lab demonstration
- **Admins**: Manage dataset, access control, API configuration

### 2.4 Operating Environment

- Web browser: Chrome, Firefox, Edge
- Internet: Required for Gemini API
- Local SQLite DB with Python environment
- Deployed via local server or Streamlit Cloud

### 2.5 Assumptions and Dependencies

- Gemini API must be functional with a valid API key
- Database structure should match prompt context
- Internet connection must be stable for real-time inference

### 2.6 Requirement

- Frontend: Streamlit + HTML/Markdown
- Backend: Python 3.10+
- Libraries: streamlit, google-generativeai, sqlite3, dotenv
- Environment: .env file for API key storage

### 2.7 Data Requirement

- Database Name: STUDENTS
- Columns: name (VARCHAR), class (VARCHAR), marks (INT), company (VARCHAR), city (VARCHAR), graduation_year (INT), internship_completed (BOOLEAN)
- Records: At least 35 entries to enable diverse testing

- **3. External Interface Requirements**

**3.1 User Interface**

- Responsive layout
- Dark-themed glowing input fields
- Orbitron/Roboto font styling
- Hover and animation effects for usability
- Result rendering in tables with pagination (future feature)

**3.2 Hardware Interfaces**

- Devices supported: Smartphone, Laptop, Desktop
- No specialized hardware required
- Compatible with both Windows and Linux systems

**3.3 Software Interfaces**

- SQLite for local DB
- Google Gemini API for AI response
- Streamlit frontend engine
- dotenv for environment variable management

**3.4 Communication Interfaces**

- REST-based HTTPS API requests to Gemini
- Local socket communication for frontend–backend integration

- **4. System Features**

### 4.1 Scope of the Work

- Enable intelligent SQL generation and execution
- Make databases usable for English-only users
- Showcase AI integration with databases for academic demos

### 4.2 Scope of the Product

- Interactive English-to-SQL interface
- Dynamic SQL validation and safety
- Real-time data query response
- Configurable database and prompt context

### 4.3 Functional Requirements

- Text field for English queries
- Send query to Gemini model
- Convert and validate SQL
- Execute using sqlite3
- Display result using Streamlit table
- Highlight keyword matches
- Error reporting for invalid requests

- **5. Other Nonfunctional Requirements**

### 5.1 Performance Requirements

- Handle ~10 concurrent users
- Query response < 3 seconds
- Uptime goal: 99% for hosted deployment

### 5.2 Safety Requirements

- Log every query executed (future add-on)
- Only SELECT queries allowed
- Prevent DROP, DELETE, UPDATE statements

### 5.3 Security Requirements

- Environment file for securing API key
- No personally identifiable information stored
- Future addition: role-based access control

### 5.4 Software Quality Attributes

- Clean modular Python codebase
- Easy to maintain and extend
- Portable across operating systems

### 5.5 Business Rules

- Prompt follows specific template structure
- Database schema must align with prompt
- Application must reject ambiguous queries

### 5.6 User Requirements

- Minimal computer knowledge required
- No SQL or programming experience needed
- Instructions provided in UI header

- **6. Domain Requirements**

  NLP model must be capable of generating SQL from diverse question types

  DB backend must be schema-mapped with prompt

  System must be robust to vague inputs and still generate valid queries

- **7. Other Requirements**

### 7.1 Data and Category Requirements

- Data must simulate real-world student dataset
- Must include textual and numeric fields
- Categories include academic performance, placement status, location

- **Appendices**

### Appendix A: Glossary

  - SQL: Structured Query Language
  - LLM: Large Language Model
  - NLP: Natural Language Processing
  - Gemini: Google Generative AI
  - Streamlit: Python UI Framework

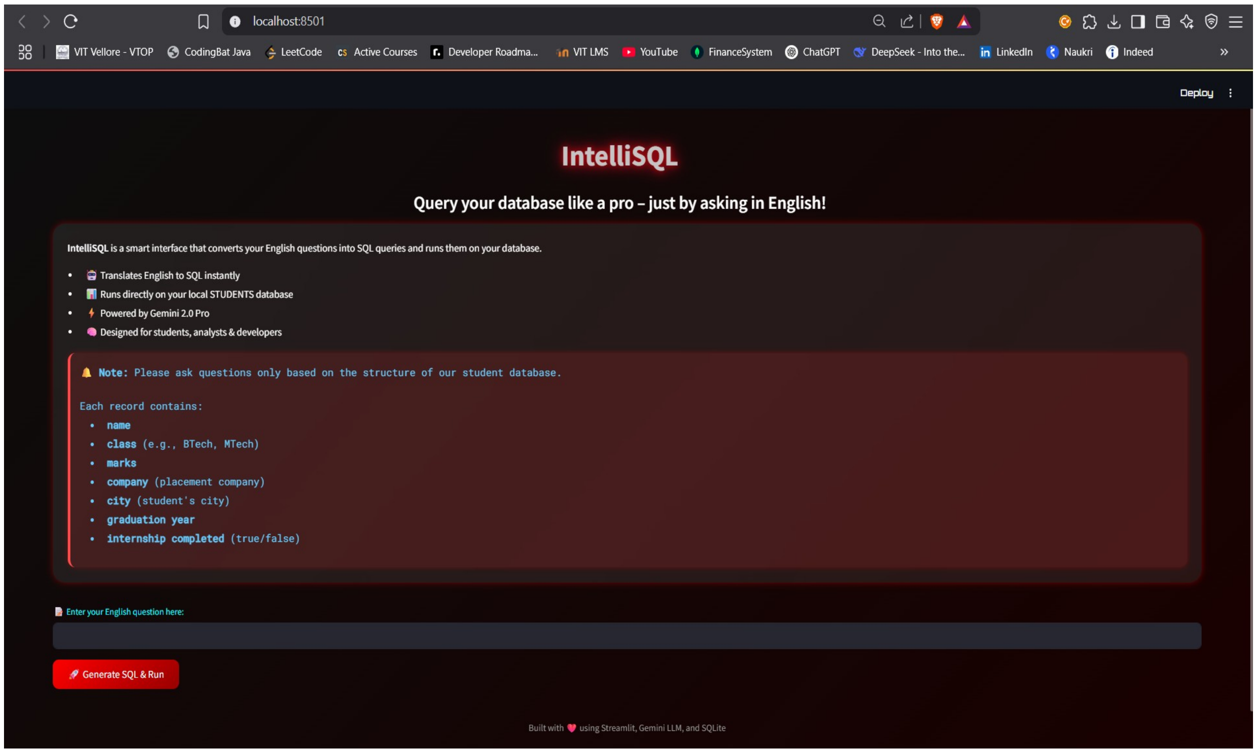
### Appendix B: Analysis Models

- Use Case: User → Gemini → SQL → SQLite → Streamlit
- Sequence Diagram: English Input → Prompt Template → Gemini API → SQL → DB Execution → Result

### Appendix C: To Be Determined List

- Multi-table query capability
- Query result export to Excel/CSV
- User login and session history logging
- Integration with other cloud DBs (e.g., PostgreSQL)

## 1. Demo Photos of Project



# IntelliSQL

**Query your database like a pro – just by asking in English!**

**IntelliSQL** is a smart interface that converts your English questions into SQL queries and runs them on your database.

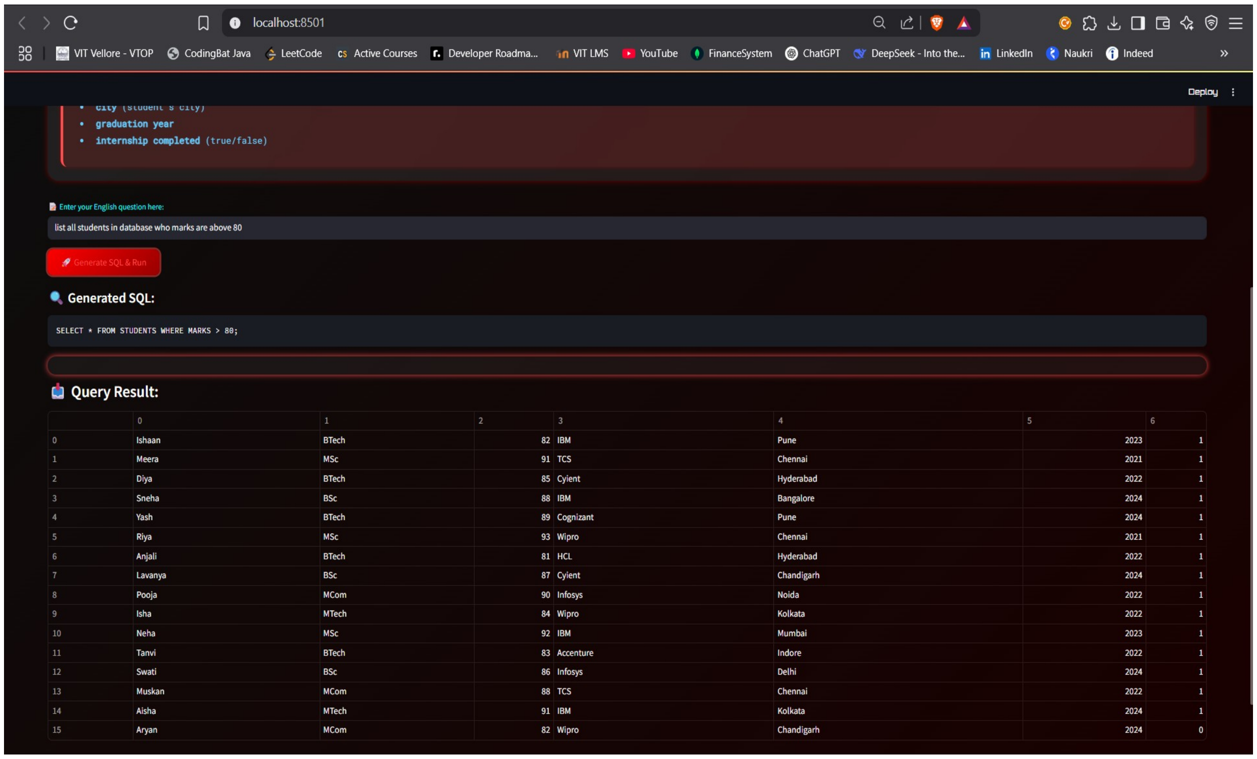
- Translates English to SQL instantly
- Runs directly on your local STUDENTS database
- Powered by Gemini 2.0 Pro
- Designed for students, analysts & developers

**Note:** Please ask questions only based on the structure of our student database.

Each record contains:

- **name**
- **class** (e.g., BTech, MTech)
- **marks**
- **company** (placement company)
- **city** (student's city)
- **graduation year**
- **internship completed** (true/false)

localhost:8501
Deploy
Enter your English question here:
Generate SQL & Run
Built with ❤️ using Streamlit, Gemini LLM, and SQLite



- **city** (student's city)
- **graduation year**
- **internship completed** (true/false)

localhost:8501
Deploy
Enter your English question here:
Generate SQL & Run

list all students in database who marks are above 80

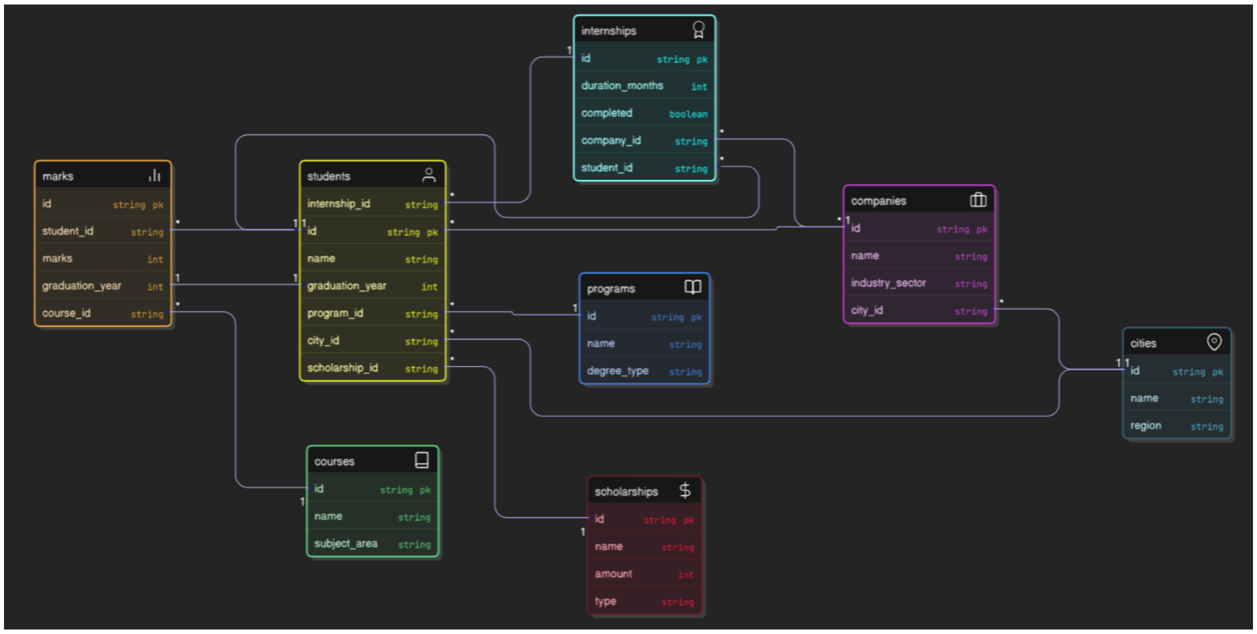
**Generated SQL:**

```
SELECT * FROM STUDENTS WHERE MARKS > 80;
```

**Query Result:**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | Ishaan | BTech | 82 | IBM | Pune | 2023 | 1 |
| 1 | Meera | MSc | 91 | TCS | Chennai | 2021 | 1 |
| 2 | Diya | BTech | 85 | Cyient | Hyderabad | 2022 | 1 |
| 3 | Sneha | BSc | 88 | IBM | Bangalore | 2024 | 1 |
| 4 | Yash | BTech | 89 | Cognizant | Pune | 2024 | 1 |
| 5 | Riya | MSc | 93 | Wipro | Chennai | 2021 | 1 |
| 6 | Anjali | BTech | 81 | HCL | Hyderabad | 2022 | 1 |
| 7 | Lavanya | BSc | 87 | Cyient | Chandigarh | 2024 | 1 |
| 8 | Pooja | MCom | 90 | Infosys | Noida | 2022 | 1 |
| 9 | Isha | MTech | 84 | Wipro | Kolkata | 2022 | 1 |
| 10 | Neha | MSc | 92 | IBM | Mumbai | 2023 | 1 |
| 11 | Tanvi | BTech | 83 | Accenture | Indore | 2022 | 1 |
| 12 | Swati | BSc | 86 | Infosys | Delhi | 2024 | 1 |
| 13 | Muskan | MCom | 88 | TCS | Chennai | 2022 | 1 |
| 14 | Aisha | MTech | 91 | IBM | Kolkata | 2024 | 1 |
| 15 | Aryan | MCom | 82 | Wipro | Chandigarh | 2024 | 0 |

## <u>ER Software Diagram of IntelliSQL Project</u>


internships
id string pk
duration_months int
completed boolean
company_id string
student_id string
marks
id string pk
student_id string
marks int
graduation_year int
course_id string
students
internship_id string
id string pk
name string
graduation_year int
program_id string
city_id string
scholarship_id string
companies
id string pk
name string
industry_sector string
city_id string
programs
id string pk
name string
degree_type string
cities
id string pk
name string
region string
courses
id string pk
name string
subject_area string
scholarships
id string pk
name string
amount int
type string
1
*