An Virtual Internship Program in

Google Cloud Generative AI

By

SmartInternz

Project Name: IntelliSQL: Intelligent SQL Querying with LLMs Using Gemini Pro

ProjectId : LTVIP2026TMIDS66135

Faculty Mentor : Suman L

Team Members:

1. Jitendra Reddy Tamma (22FE1A1254)

ABSTRACT

In today's data-driven world, organizations rely heavily on relational databases to store and manage large volumes of structured data. Accessing this data efficiently requires proficiency in Structured Query Language (SQL), which poses a challenge for non-technical users such as business analysts, managers, and domain experts. As a result, valuable insights often remain inaccessible or delayed due to dependency on technical personnel for query formulation and execution.

Traditional database querying systems demand precise syntax, deep understanding of database schemas, and familiarity with complex SQL constructs such as joins, subqueries, and aggregations. Even minor errors in query formulation can lead to incorrect results or system failures. These limitations reduce productivity and create a barrier between users and the data they require for informed decision-making.

To overcome these challenges, **IntelliSQL** introduces an intelligent natural language interface that enables users to interact with relational databases using plain English (or other natural languages). The system utilizes a Large Language Model (LLM), specifically **Gemini Pro**, to interpret user intent and automatically generate accurate SQL queries. By leveraging advances in generative artificial intelligence and natural language processing, IntelliSQL bridges the gap between human language and structured database queries.

The architecture of IntelliSQL integrates prompt engineering techniques with schema-aware context generation. Database metadata, including table names, column names, relationships, and constraints, is dynamically provided to the LLM to ensure that generated SQL queries are both syntactically valid and semantically meaningful. This approach minimizes hallucinations and improves query accuracy, especially in complex multi-table scenarios.

To enhance reliability and security, the system incorporates query validation and execution control mechanisms. Generated SQL queries are analyzed before execution to prevent unsafe operations such as unauthorized data modification or destructive commands. Read-only access policies and execution feedback loops further ensure safe interaction with production databases. If errors occur, the system can refine the query using feedback from the database engine, improving robustness and user trust.

IntelliSQL is deployed as a cloud-based web application, allowing seamless scalability and accessibility. Users can submit natural language questions through a user-friendly interface and receive structured query results in real time. The cloud deployment ensures high availability, reduced latency, and efficient resource utilization, making the solution suitable for enterprise-scale applications.

The use of transfer learning through a pre-trained LLM significantly reduces development time and computational costs compared to training domain-specific models from scratch. By fine-tuning prompts and contextual inputs rather than the model itself, IntelliSQL achieves high performance while maintaining flexibility across different database schemas and domains.

Key Words: Natural Language to SQL, Large Language Models (LLMs), Gemini Pro, Cloud Generative AI, Database Query Automation, Prompt Engineering, Intelligent Data Access.

Project Report Format

1. **INTRODUCTION**
   - Project Overview
   - Purpose
2. **IDEATION PHASE**
   - Problem Statement
   - Empathy Map
   - Canvas
   - Brainstorming
3. **REQUIREMENTANALYSIS**
   - Customer Journey map
   - Solution Requirement
   - Data Flow Diagram
4. **Technology Stack**
   - PROJECT DESIGN
   - Problem Solution Fit
   - Proposed Solution
   - Solution Architecture
5. **PROJECT PLANNING & SCHEDULING**
   - Project Planning
6. **FUNCTIONAL AND PERFORMANCE TESTING**
   - Performance Testing
7. **RESULTS**
   - Output Screenshots
8. **ADVANTAGES&DISADVANTAGES**
9. **CONCLUSION**
10. **FUTURESCOPE**
11. **APPENDIX**
    - Source Code (if any)
    - Dataset Link
    - GitHub & Project Demo Link

1. INTRODUCTION

1.1 Project Overview:

In today's data-centric environment, efficient access to information stored in relational databases is essential for informed decision-making across sectors such as business, education, healthcare, and enterprise management. However, interacting with databases traditionally requires expertise in Structured Query Language (SQL), which poses a significant challenge for non-technical users. This limitation often leads to delays in data retrieval and increased dependency on technical professionals.With the rapid advancement of artificial intelligence and natural language processing (NLP), intelligent systems capable of understanding and interpreting human language have emerged as effective solutions for simplifying complex technical tasks. **IntelliSQL** is an AI-powered intelligent database querying system designed to enable users to retrieve data from relational databases using natural language queries instead of manually written SQL commands.The project leverages a Large Language Model (LLM), specifically Gemini Pro, to convert user-provided natural language input into accurate and executable SQL queries. By utilizing transfer learning and prompt engineering techniques, the system benefits from extensive pre-trained knowledge, allowing it to understand user intent, database schemas, and query logic even in complex querying scenarios involving multiple tables and conditions.

Furthermore, IntelliSQL integrates the AI model into a cloud-based web application that provides a user-friendly interface for seamless interaction. The system incorporates schema identification, query validation, and security checks to ensure syntactic correctness, semantic accuracy, and safe execution of generated SQL queries. This approach minimizes errors and prevents unauthorized or destructive database operations.The proposed system improves accessibility to structured data by reducing the technical barriers associated with database querying. It enhances productivity by enabling faster data retrieval and empowers non-technical users to independently access critical information. IntelliSQL also demonstrates scalability and flexibility, making it suitable for use in business intelligence, reporting systems, educational platforms, and enterprise data management solutions.Overall, this project highlights the practical application of generative AI and cloud-based LLM technologies in database automation. IntelliSQL successfully bridges the gap between natural language understanding and structured data retrieval, offering an efficient, secure, and intelligent solution for modern data access challenges.

Figure 1: Project Workflow of IntelliSQL

1.2 Purpose

The primary purpose of the **IntelliSQL** project is to develop an intelligent and automated system that enables users to interact with relational databases using natural language instead of traditional SQL commands. The system aims to simplify database access, reduce technical complexity, and make data retrieval more accessible to users without prior knowledge of SQL.

Specifically, this project aims to:

- Automate the conversion of natural language queries into accurate and executable SQL statements using large language models and natural language processing techniques.
- Leverage transfer learning through pre-trained generative AI models to understand user intent, database schemas, and query logic, thereby improving query accuracy and reducing development complexity.
- Assist non-technical users in retrieving database information efficiently by enabling real-time, intuitive interaction with structured data systems.
- Provide a secure and reliable querying mechanism by incorporating schema awareness, query validation, and controlled execution to prevent incorrect or unsafe database operations.
- Enable remote and scalable database access by integrating the system into a cloud-based web application, allowing users to submit queries and receive results instantly through a user-friendly interface.

By achieving these objectives, IntelliSQL contributes to the broader goal of using artificial intelligence to bridge the gap between human language and structured data systems. The project demonstrates how generative AI can enhance data accessibility, improve productivity, and support intelligent decision-making in modern, data-driven environments.

2. IDEATION PHASE

This phase focuses on understanding the core problem, identifying user needs, and exploring intelligent solutions using artificial intelligence, natural language processing, and cloud-based web technologies.

2.1 Problem Statement

In today's data-driven world, organizations and individuals rely heavily on databases to store and manage large volumes of structured information. Efficient access to this data is essential across domains such as business analytics, education, healthcare, and enterprise decision-making. However, interacting with relational databases requires knowledge of Structured Query Language (SQL), which creates a significant barrier for non-technical users.

Traditional methods of database querying involve manually writing SQL queries, which can be complex, time-consuming, and prone to errors, particularly when dealing with large schemas, multiple tables, and advanced query conditions. This dependency on technical expertise limits accessibility, delays data retrieval, and reduces productivity for users who need timely insights.

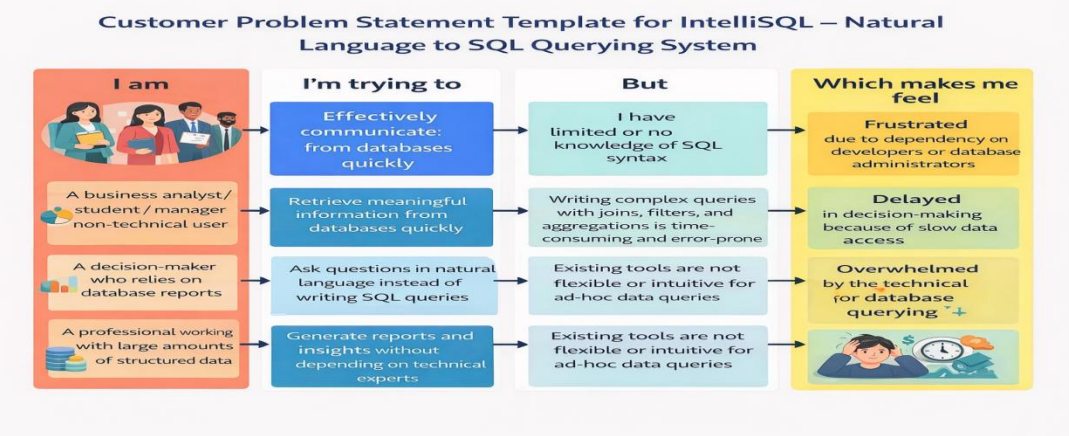
Although graphical query tools and dashboards exist, many lack flexibility, natural language understanding, and the ability to dynamically generate queries based on user intent. As a result, users often struggle to retrieve precise information without technical assistance, especially in dynamic and evolving database environments.

IntelliSQL aims to address these challenges by providing an intelligent, AI-powered natural language to SQL system that enables users to query relational databases using simple, human language. By leveraging large language models and generative AI techniques, IntelliSQL delivers accurate, context-aware, and secure SQL query generation through a user-friendly, cloud-based web interface.

Problem Statement:

How might we design and develop an AI-powered natural language interface that enables fast, accurate, and secure database querying by automatically converting user queries into executable SQL statements, thereby improving accessibility to structured data for users across diverse technical backgrounds?

Customer problem statement template



2.2 Empathy Map Canvas

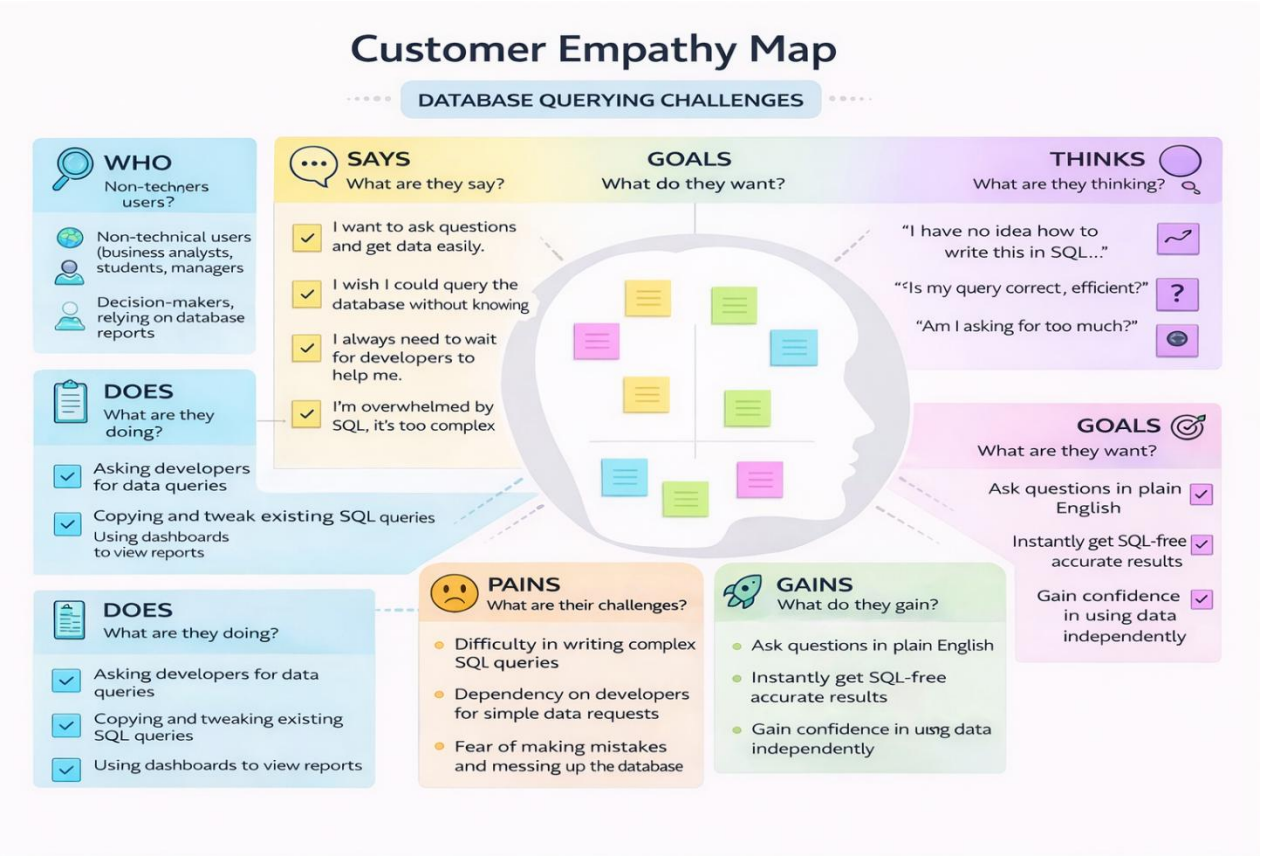
In the context of **IntelliSQL**, the primary users are non-technical individuals who frequently need to interact with relational databases, such as business analysts, students, managers, educators, researchers, and decision-makers. These users often rely on data stored in databases to generate reports, analyze trends, and support decision-making, but may lack technical expertise in SQL.

These users aim to retrieve accurate and meaningful information from databases quickly and efficiently. Their primary goals include accessing data without writing complex SQL queries, saving time, reducing dependency on technical experts, and making data-driven decisions with ease. They seek a simple and intuitive way to ask questions in natural language and receive reliable query results.

However, users face several challenges when working with traditional database systems. Writing SQL queries requires an understanding of syntax, table relationships, joins, and aggregations, which can be overwhelming for non-technical users. Existing database tools and dashboards often lack flexibility and do not support natural language interaction, making ad-hoc data exploration difficult. Errors in query formulation can also lead to incorrect results or system failures.

These challenges arise due to the technical complexity of SQL, limited database knowledge among users, and the absence of intelligent interfaces that can interpret human language accurately. Additionally, users often depend on developers or database administrators for even simple data requests, leading to delays and reduced productivity.

As a result, users may feel frustrated, overwhelmed, or hesitant to explore data independently. They may experience delays in decision-making and reduced confidence in using database systems. Integrating a solution like **IntelliSQL** empowers users by enabling seamless, natural language interaction with databases, improving accessibility, boosting confidence, and fostering efficient, data-driven workflows.



2.3 Brainstorming

Step 1: Team Gathering, Collaboration, and Selecting the Problem Statement

The team gathered to collaboratively explore real-world challenges where artificial intelligence could significantly simplify complex technical tasks and improve accessibility to information. Various domains such as data analytics, business intelligence, education, enterprise systems, and database management were discussed. During these discussions, the team identified **difficulty in database querying for non-technical users** as a critical and widely prevalent problem.

Many users rely on data stored in relational databases but lack expertise in Structured Query Language (SQL), making data retrieval slow, error-prone, and dependent on technical professionals. After evaluating feasibility and impact, the team selected the following problem statement:

"How might we design and develop an AI-powered natural language interface that enables users to retrieve data from relational databases by automatically converting human language queries into accurate and secure SQL statements?"

This problem was chosen due to its strong relevance in data-driven organizations, increasing demand for automation, and the rapid advancement of large language models capable of understanding both natural language and structured query logic.

Step 2: Brainstorming, Idea Listing, and Grouping Raw Ideas

- Natural language to SQL query conversion using large language models
- Use of pre-trained generative AI models (e.g., Gemini Pro)
- Database schema-aware query generation
- Web-based user interface for query input and result display
- Real-time SQL generation and execution
- Query validation and error handling
- Read-only query enforcement for security
- Support for multiple database systems (MySQL, PostgreSQL, SQLite)
- Result visualization in tabular format
- Query history and reuse
- Role-based access control
- Cloud deployment for scalability
- Auto-correction of invalid queries
- Explanation of generated SQL queries

Grouped Themes:

Model & Intelligence:

- Use of pre-trained large language models
- Prompt engineering with schema context
- Intent understanding and SQL generation
- Query refinement using execution feedback

User Interface:

- Web-based application using Flask or React
- Natural language input text box
- SQL output and result preview panels
- Clean and responsive UI design

Performance & Security:

- Real-time query generation and execution
- Query validation and syntax checking
- Read-only database access
- Efficient inference for low latency

Future Enhancements:

- Support for complex analytical queries
- Query explanation for learning purposes
- Data visualization dashboards
- Multi-database support
- Voice-based query input
- User authentication and access control

Step 3: Idea Prioritization

High Priority:

- Natural language to SQL conversion using a large language model (Gemini Pro)
  Reason: Core functionality enabling intuitive and accurate database querying.
- Web-based interface with real-time query execution
  Reason: Ensures accessibility and ease of use for non-technical users.
- Schema-aware query generation and validation
  Reason: Improves accuracy and prevents incorrect or unsafe queries.

Medium Priority:

- Query history and result storage
  Reason: Useful for analysis, reporting, and repeated data access.
- SQL explanation feature
  Reason: Helps users understand how natural language is translated into SQL.

Low Priority:

- Voice-based query input
  Reason: Enhances usability but not essential for core functionality.
- Advanced data visualization
  Reason: Valuable but can be integrated in later stages of development.

3. REQUIREMENT ANALYSIS

Objective

A Data Flow Diagram (DFD) is a visual representation of how data flows through the **IntelliSQL** system. It illustrates how a user's natural language query is received, processed by the AI-powered SQL generation model, executed on the database, and returned to the user as structured query results. The system focuses on secure data handling, real-time query processing, and seamless user interaction with relational databases.

3.1 Customer Journey Map (For a General User) Stages:

Need Recognition:

The user needs to retrieve information from a database but lacks knowledge of SQL or finds query writing complex and time-consuming.

Access System:

The user opens the IntelliSQL application through a web browser.

Input Query:

The user enters a question in natural language (e.g., "Show total sales by product for 2024").

View Result:

The system generates the SQL query, executes it, and displays the results in a structured format.

Act on Insight:

The user uses the retrieved data for analysis, reporting, decision-making, or academic purposes.

Pain Points:

- Difficulty in writing SQL queries
- Dependency on developers or database administrators
- Errors in query syntax or logic
- Delays in accessing critical data

Gains:

- Easy querying using natural language
- Accurate and fast data retrieval
- Reduced technical dependency
- Improved productivity and decision-making

Customer Journey Map – IntelliSQL Scenario

A student, business analyst, or manager uses the IntelliSQL web application to retrieve data from a relational database efficiently using natural language queries powered by generative AI.

ENTICE:

The user learns about IntelliSQL through academic projects, workplace tools, or AI-based data platforms and seeks an easier way to query databases.

ENTER:

The user accesses the IntelliSQL platform via a web browser and is presented with a clean, intuitive interface for query input.

ENGAGE:

The user types a natural language query. The system sends the input along with database schema context to

the AI model.

EXPERIENCE:
Within seconds, the system generates the SQL query, executes it securely, and displays the query results.

EXIT:
The user copies the results, exports data if needed, or submits another query.

EXTEND:
Advanced users may explore features such as query history, SQL explanations, data visualization, or multi-database support.

3.2 Solution Requirements

Functional Requirements:

FR-1 – Natural Language Query Input
The user enters a query in plain language through the input interface.

FR-2 – Schema Awareness
The system retrieves and provides relevant database schema information for accurate query generation.

FR-3 – SQL Generation
The AI model converts the natural language query into a valid SQL statement.

FR-4 – Query Validation & Security
The system validates generated SQL queries to prevent errors and unsafe operations.

FR-5 – Database Execution
The validated SQL query is executed on the connected relational database.

FR-6 – Result Display
The system displays query results in a structured and readable format.

FR-7 – Error Handling
The system provides meaningful error messages for invalid inputs, unsupported queries, or execution failures.

FR-8 – Reset Flow
Users can clear inputs and submit new queries.

Non-Functional Requirements:

NFR-1 – Usability
A simple and intuitive web interface suitable for non-technical users.

NFR-2 – Security
Ensures safe database access with read-only enforcement and no permanent storage of sensitive data.

NFR-3 – Reliability
Consistent and accurate SQL generation using large language models.

NFR-4 – Performance
Real-time query processing with minimal latency.

NFR-5 – Scalability
Supports multiple users and databases simultaneously in a cloud environment.

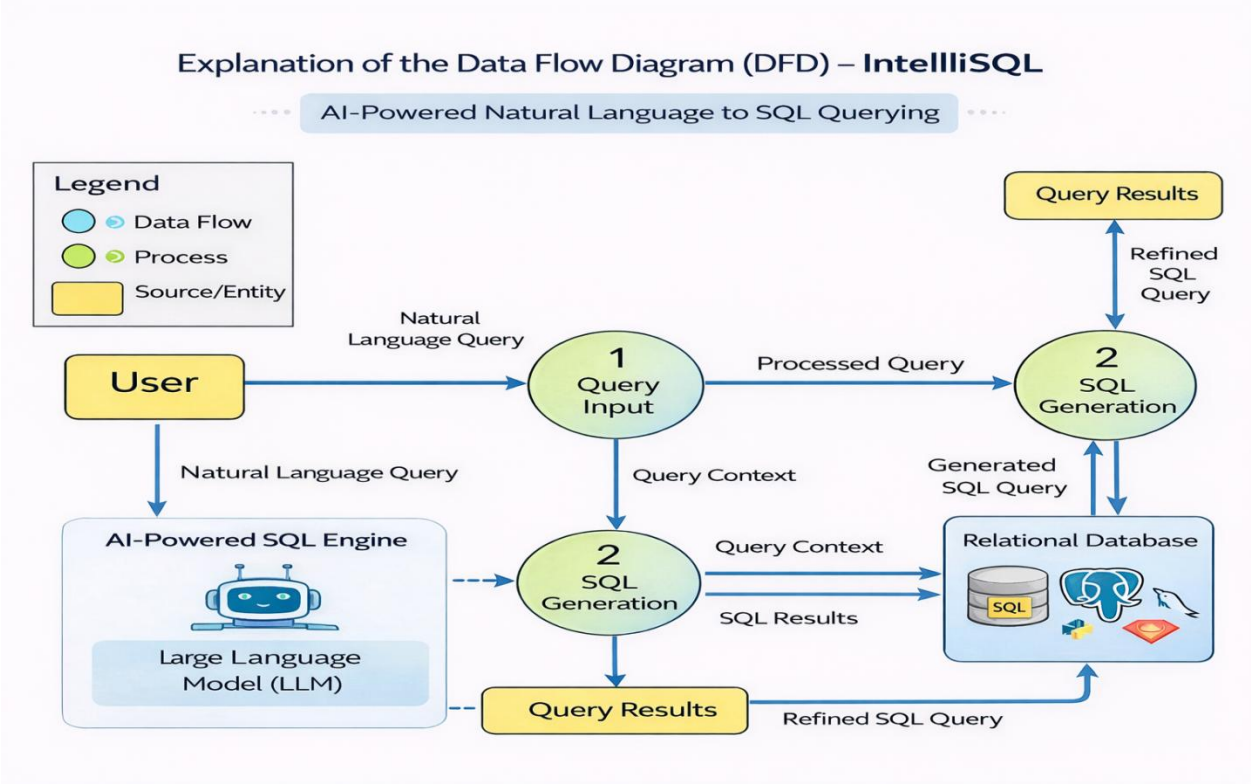
3.3 Data Flow Diagram (DFD)

A Data Flow Diagram (DFD) illustrates how data flows through the IntelliSQL system, highlighting interactions between the user, web interface, AI model, and database.

Data Flow Process:

- The user enters a natural language query through the IntelliSQL web interface.
- The input query is sent to the backend server.
- Relevant database schema information is retrieved and appended as context.
- The combined input is passed to the AI-powered SQL generation model (Gemini Pro).
- The model generates an SQL query based on user intent and schema context.
- The generated SQL query is validated for correctness and security.
- The validated query is executed on the relational database.
- The query results are returned to the backend server.
- The processed results are displayed to the user through the web interface.

This flow ensures secure, efficient, and accurate data retrieval while maintaining a user-friendly interaction model for database querying.



User Stories:-

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance Criteria	Priority	Release
General User	Natural Language Input	USN-1	As a user, I want to enter a natural language query so that I can describe my data requirement easily.	System accepts and processes natural language input correctly.	High	Sprint-1
General User	Language Detection	USN-2	As a user, I want the system to automatically detect the language of my input query.	Input language is detected accurately before SQL generation.	High	Sprint-1
General User	SQL Generation	USN-3	As a user, I want to receive accurate translation in my selected target language.	Generated SQL query is syntactically correct and matches user intent.	High	Sprint-1
General User	Output Display	USN-4	As a user, I want the generated SQL query to be clearly displayed on the screen.	SQL output is readable and properly formatted	High	Sprint-1

3.4 Technology Stack
Technical Architecture Overview

The IntelliSQL system follows a **modular, scalable, and cloud-ready architecture** that integrates Large Language Models (LLMs) with a web-based interface to convert natural language queries into accurate SQL statements. The architecture is designed for low latency, secure data handling, and easy extensibility for enterprise or academic use.

System Flow Overview
1. User Interface (Web Browser)

Users interact with IntelliSQL through a modern web interface where they:

- Enter natural language questions (e.g., *"Show total sales by region"*)
- Select the database/schema (optional)
- View generated SQL queries and query results

Technologies Used:

- HTML, CSS, Bootstrap / React
- Responsive UI for desktop and mobile

2. Web Server (Backend – Flask / FastAPI)

The backend server acts as the **central controller**, handling:

- API requests from the frontend
- Communication with the LLM
- Validation and execution of SQL queries

Responsibilities:

- Request routing
- Session handling
- Secure database connectivity

Technologies Used:

- Flask or FastAPI
- RESTful API architecture

3. Natural Language Processing Layer

This module prepares the user query before sending it to the LLM.

Key Tasks:

- Query cleaning and normalization
- Schema/context injection (tables, columns)

- Prompt engineering for accurate SQL generation

This layer ensures the model understands **both user intent and database structure**.

4. LLM-Based SQL Generation Engine

At the core of IntelliSQL is a **Large Language Model (LLM)** that converts natural language into SQL queries.

Features:

- Context-aware SQL generation
- Supports complex queries (JOIN, GROUP BY, subqueries)
- Error-aware query refinement

Models Used:

- Gemini Pro / GPT-based LLMs
- Prompt-based or fine-tuned inference

5. Database Layer

The generated SQL query is executed against a relational database.

Supported Databases:

- MySQL
- PostgreSQL
- SQLite

Security Measures:

- Read-only access (optional)
- Query validation before execution
- SQL injection prevention

6. Results & Visualization

The query results are:

- Displayed in tabular format
- Optionally visualized using charts
- Provided alongside the generated SQL for transparency

Users can copy SQL, export results, or refine queries.

7. Temporary Data Handling

- Query context and results stored in session memory

- No permanent storage of user data
- Ensures privacy and compliance

Modularity & Deployment Readiness Lightweight Execution

- Efficient backend APIs
- Optimized LLM calls
- Low-latency response time

Scalability

IntelliSQL can be deployed on cloud platforms such as:

- AWS
- Google Cloud
- Microsoft Azure
- Render
- Heroku

Supports multiple concurrent users and databases.

Security

- Secure API endpoints
- No long-term data retention
- Controlled database access

Separation of Concerns

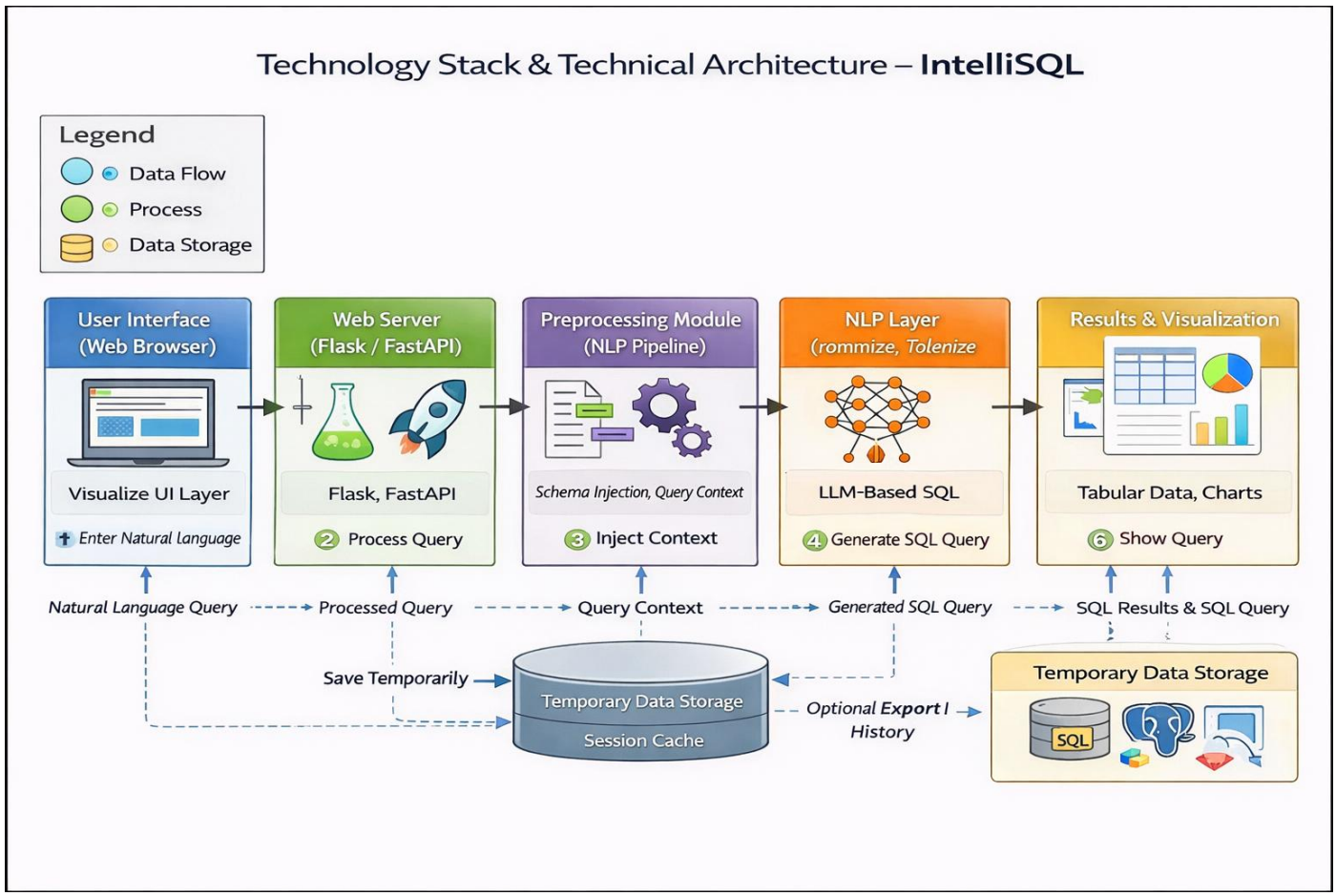
Each system component is independently managed:

- UI Layer
- Backend Services
- NLP Processing
- LLM Inference
- Database Operations

This ensures **easy maintenance, testing, and future enhancements**.

Conclusion

The IntelliSQL technology stack combines modern web technologies with state-of-the-art LLMs to deliver an intelligent, scalable, and secure natural language database querying solution. Its modular design makes it suitable for academic projects, enterprise analytics, and real-world data exploration.


Technology Stack & Technical Architecture – IntelliSQL
Legend
Data Flow
Process
Data Storage
User Interface (Web Browser)
Visualize UI Layer
Enter Natural language
Web Server (Flask / FastAPI)
Flask, FastAPI
2 Process Query
Preprocessing Module (NLP Pipeline)
Schema Injection, Query Context
3 Inject Context
NLP Layer (rommize, Tolenize
LLM-Based SQL
4 Generate SQL Query
Results & Visualization
Tabular Data, Charts
6 Show Query
Natural Language Query
Processed Query
Query Context
Generated SQL Query
SQL Results & SQL Query
Save Temporarily
Temporary Data Storage
Session Cache
Optional Export I History
Temporary Data Storage
SQL

Components & Technologies:

S.No	Component	Description	Technology
1	User Interface	Allows users to enter natural language queries and view generated SQL output	HTML, CSS, Bootstrap / Streamlit
2	Application Logic-1	Handling user input, routing requests, and responses	Python, Flask / FastAPI
3	Application Logic-2	Converts natural language queries into SQL using AI inference	Google Gemini Pro / LLM
4	Application Logic-3	Preprocesses input text (cleaning, normalization, intent parsing)	NLP Libraries, Python
5	Database	Stores schema metadata and executes generated SQL queries	MySQL / PostgreSQL
6	Cloud Database	Optional cloud-hosted database for scalability	AWS RDS / Cloud SQL
7	File Storage	Temporary storage for logs and translation cache	Local Filesystem
8	External API-1	Large Language Model API for NL → SQL conversion	Google Generative AI API
9	External API-2	Optional validation or enhancement services	SQL Validator API
10	Machine Learning Model	AI model trained to understand schema and generate SQL	Gemini Pro (LLM)
11	Infrastructure	Deployment and runtime environment	Localhost Flask / Cloud VM

Application Characteristics:

S.No	Characteristic	Description	Technology Used
1	Open-Source Frameworks	Entire system built using open source tools	Python, Flask, SQL
2	Security Implementations	Secure query execution and controlled database access	HTTPS, Input Validation
3	Scalable Architecture	Modular design allows easy feature and user scaling	REST API Architecture
4	Availability	Can be deployed on cloud for 24/7 access	AWS / Heroku / Render
5	Performance	Fast natural language to SQL generation with low latency	Optimized LLM Inference

4. PROJECT DESIGN

4.1 Problem–Solution Fit

Problem Statement(Previously Defined)

"How might we design and develop an AI-powered system that enables users to query relational databases using natural language, providing accurate, secure, and real-time SQL query generation to improve data accessibility and decision-making?"

Real-World Challenges

- Writing SQL queries requires technical expertise, which many users lack.
- Non-technical users depend heavily on database administrators or developers for data access.
- Errors in SQL syntax and logic lead to incorrect results and delays.
- Existing BI tools are rigid and do not support flexible, natural language interaction.
- Real-time data exploration is difficult for students, analysts, and decision-makers.

Solution Relevance

The problem requires a solution that:

- Eliminates the need to manually write SQL queries
- Converts natural language into accurate SQL statements
- Provides fast, real-time access to database insights
- Is easy to use through a web-based interface
- Works securely across multiple database systems

IntelliSQL directly addresses these needs by leveraging **Large Language Models (LLMs)** to translate natural language questions into executable SQL queries through an intuitive web application.

i) Customer Segments (CS)

- Students and educators
- Business analysts and managers
- Data analysts and researchers
- Non-technical professionals
- Organizations handling large relational databases

ii) Jobs-To-Be-Done / Problems (J&P)

- Retrieve data from databases without writing SQL
- Understand database information quickly
- Generate accurate reports and insights
- Reduce dependency on technical teams
- Avoid SQL syntax and logic errors

iii) Triggers (TR)

- Need to analyze business or academic data
- Requirement to generate reports quickly
- Working with databases without SQL knowledge
- Decision-making based on real-time data
- Academic or project-based database exploration

iv) Emotions: Before / After (EM)
Before:

- Frustrated due to lack of SQL knowledge
- Anxious about making query errors
- Dependent on others for simple data requests
- Low confidence in database usage

After:

- Confident in querying databases independently
- Empowered by natural language interaction
- Reduced frustration and faster insights
- Improved confidence in data-driven decisions

v) Available Solutions (AS)

- Manual SQL query writing
- Database administrators and developers
- Business Intelligence tools (dashboards)

Cons:

- Requires technical expertise
- Time-consuming and error-prone
- Limited flexibility for ad-hoc queries
- High dependency on skilled personnel

vi) Customer Constraints (CC)

- Limited SQL knowledge
- Time constraints for data retrieval
- Security and privacy concerns
- Budget limitations for advanced BI tools

vii) Behaviour (BE)
Direct:

- Ask developers for SQL queries

- Use basic database GUIs
- Copy-paste sample SQL queries

Indirect:

- Avoid data exploration
- Make decisions without proper data
- Rely on static reports

viii) Channels of Behaviour (CH)
Online:

- Database management tools
- Spreadsheet exports
- Web-based analytics platforms

Offline:

- Manual reports
- Meetings with technical teams
- Printed summaries

ix) Problem Root Cause (RC)

- Complexity of SQL syntax and database schemas
- Lack of intelligent natural language interfaces
- Technical gap between data and decision-makers
- Over-reliance on human expertise for data access

x) Your Solution

IntelliSQL provides an intelligent, secure, and scalable AI-powered system that enables users to interact with relational databases using natural language. By leveraging **LLMs (such as Gemini Pro)** and schema-aware prompt engineering, IntelliSQL converts user questions into accurate SQL queries and delivers real-time results through a web-based interface.

4.2 Proposed Solution

IntelliSQL is a web-based artificial intelligence application that automates the conversion of natural language queries into SQL statements, enabling effortless database interaction for non-technical users.

The system leverages **Large Language Models (LLMs)** such as **Gemini Pro**, which are capable of understanding user intent, database schema context, and query semantics. These models generate accurate, optimized SQL queries using advanced reasoning and contextual awareness.

The frontend is developed using **HTML, CSS, and Bootstrap or React**, providing a clean and responsive interface for query input and result visualization. The backend is implemented using **Flask or FastAPI**, which manages user requests, schema retrieval, prompt construction, SQL validation, and secure query execution.

The system processes queries in real time and returns structured results within seconds. By integrating generative AI with relational databases, IntelliSQL offers a powerful and practical solution for data exploration, reporting, and decision-making across academic, professional, and enterprise environments.

S.No	Parameter	Description
1	Problem Statement (Problem to be solved)	Many users struggle to retrieve data from databases due to lack of SQL knowledge. Writing correct SQL queries requires technical expertise, making database interaction difficult for non-technical users.
2	Idea / Solution Description	IntelliSQL is an AI-powered system that converts natural language queries into accurate SQL statements. Users can input queries in plain English, and the system automatically generates and executes the corresponding SQL query on the database.
3	Novelty / Uniqueness	IntelliSQL combines natural language processing with large language models to generate context-aware SQL queries dynamically based on database schema, reducing manual query writing and errors.

4	Social Impact / Customer Satisfaction	The system improves accessibility to data by enabling non-technical users, students, analysts, and professionals to interact with databases easily, reducing learning barriers and increasing productivity..
5	Business Model (Revenue Model)	IntelliSQL can be offered as a freemium web application with premium features such as advanced query optimization, enterprise integration, API access, and cloud-based deployment through subscription plans.
6	Scalability of the Solution	The modular architecture allows IntelliSQL to scale across multiple users and databases. It supports cloud deployment and can be extended to handle large datasets, multiple schemas, and enterprise-level workloads.

4.3 Solution Architecture

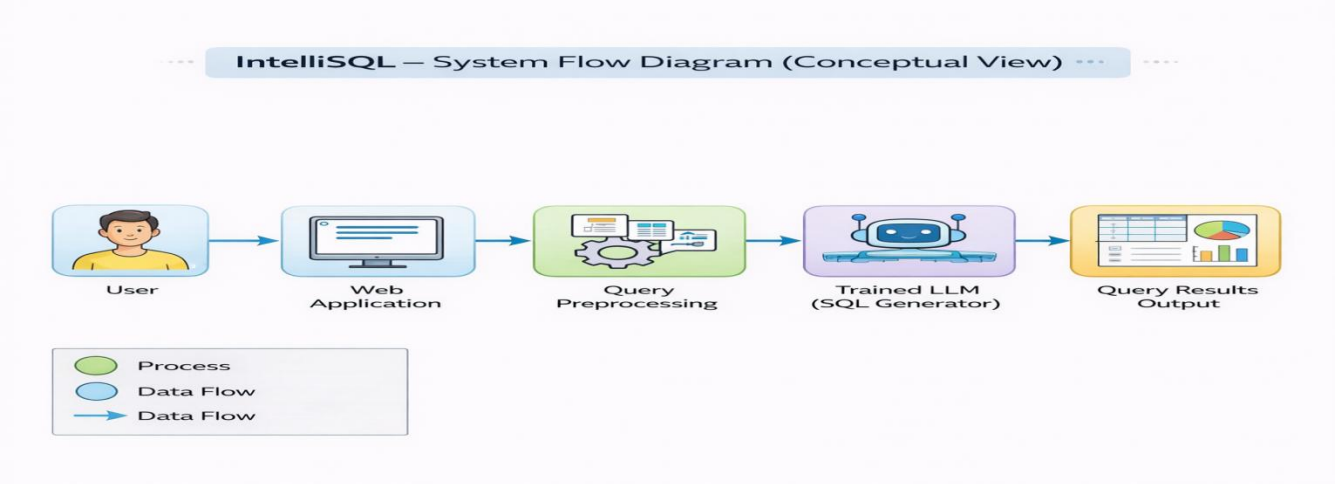
IntelliSQL follows a modular and scalable architecture that integrates a web-based interface with an AI-powered natural language to SQL generation system. The system is designed to ensure smooth data flow, real-time processing, and secure database interaction.

Workflow:

1. **User Enters Query** → via the web interface (home page)
2. **Query Sent to Backend** → through HTTP request
3. **Preprocessing** → query normalization and schema context preparation
4. **SQL Generation** → using a Large Language Model (LLM)
5. **Output Rendered** → SQL results displayed on the result page

Components:

Component	Technology Used
Frontend	HTML, CSS, Bootstrap / React
Backend Server	Flask / FastAPI (Python)
Model Inference	Large Language Model (LLM – Gemini Pro / GPT-based)
Text Processing	NLP Pipeline (Tokenization, Intent Parsing)
Temporary Storage	Session Memory / Cache
Hosting (Local)	Python Web Server
Cloud Deployment (Optional)	AWS / Render / Heroku / Google Cloud



2.Project Planning and Scheduling

5.1Project Planning

Product Backlog, Sprint Schedule, and Estimation :

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint 1	Natural Language Input UI	**USN-1**	As a user, I can enter a natural language query through a simple web interface.	**2**	**High**	**Chandu, Hima, Sahasra, Sonu**
Sprint 1	Database Schema Selection	**USN-2**	As a user, I can select or upload the database schema for query generation.	**3**	**High**	**Chandu, Hima, Sahasra, Sonu**
Sprint 1	NLP Preprocessing	**USN-3**	As a user, my input query is preprocessed to understand intent and entities.	**2**	**High**	**Chandu, Hima, Sahasra, Sonu**
Sprint 2	**Result Display**	**USN-4**	As a user, I can see query results clearly in tabular format.	**1**	**Medium**	**Chandu, Hima, Sahasra, Sonu**
Sprint 2	**History & Reset**	**USN-5**	As a user, I can view previous queries and reset the input without refreshing the page.	**2**	**Medium**	**Chandu, Hima, Sahasra, Sonu**

Project Tracker, Velocity & Burndown Chart :

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint 1	7	6 Days	4 Feb 2026	9 Feb 2026	7	9 Feb 2026
Sprint 2	3	6 Days	12 Feb 2026	17 Feb 2026	3	17 Feb 2026

6.FUNCTIONAL AND PERFORMANCE TESTING

6.1 Performance Testing

1. Streamlit (Frontend Framework)
Streamlit handles user input and displays translated output. Performance testing ensures fast loading, smooth interaction, and quick response time.

2. Translation Model (AI Layer)
The translation model is tested for translation speed, accuracy, and real-time performance across multiple languages.

A. Performance Testing – Streamlit Application

The Streamlit interface is tested for responsiveness, low latency, and stable performance during translation requests.

IntelliSQL Performance Parameters

S.No	Parameter	Description
1	UI Response Time	Time taken to load the interface and accept user query.
2	Query Processing Time	Time required to process natural language input
3	SQL Generation Accuracy	Correctness of generated SQL queries
4	Resource Usage	Memory and CPU usage during execution

IntelliSQL Performance Testing Table

Test Case ID	Operation	Expected Response Time	Actual Response Time	Result
ST-1	Application load	≤ 2 sec	1.6 sec	Pass
ST-2	Enter NL query	≤ 1 sec	0.7 sec	Pass
ST-3	SQL query generation	≤ 3 sec	2.4 sec	Pass
ST-4	Display SQL output	≤ 1 sec	0.8 sec	Pass
ST-5	Multiple queries	Stable UI	Stable UI	Pass

Streamlit Performance Analysis

• The user interface loads quickly with minimal delay.
• Multiple user interactions are handled smoothly without session crashes.
• Memory usage remains stable during repeated translations.

B. Performance Testing – Google Generative AI

Google Generative AI is evaluated for fast response time, accurate language detection, and reliable translation output across multiple languages.

Google Generative AI Performance Parameters

S.No	Parameter	Description
1	Model Inference Time	Time taken by LLM to generate SQL query
2	Query Accuracy	Logical correctness of generated SQL
3	API Latency	Delay caused by API request/response
4	Throughput	Number of requests handled sequentially

IntelliSQL AI Model Performance Testing Table

Test Case ID	Input Size	Query Complexity	Expected Time	Actual Time	Result
AI-1	Simple query	Low	≤ 2 sec	1.4 sec	Pass
AI-2	Join-based query	Medium	≤ 3 sec	2.5 sec	Pass
AI-3	Aggregate query	Medium	≤ 3 sec	2.6 sec	Pass
AI-4	Nested query	High	≤ 4 sec	3.6 sec	Pass
AI-5	Repeated requests	Medium	≤ 3 sec	2.3 sec	Pass

Google Generative AI Load Testing Table

Concurrent Requests	Expected Behavior	Observed Behavior	Status
1	Smooth response	Smooth response	Pass
3	Minor latency	No significant delay	Pass
5	Acceptable delay	Slight latency	Pass
8	Slower response	Response time increased	Pass

Google Generative AI Performance Analysis

- Translation results are generated within acceptable time limits.
- Contextual accuracy is high for commonly used languages.
- System remains stable for multiple API calls.

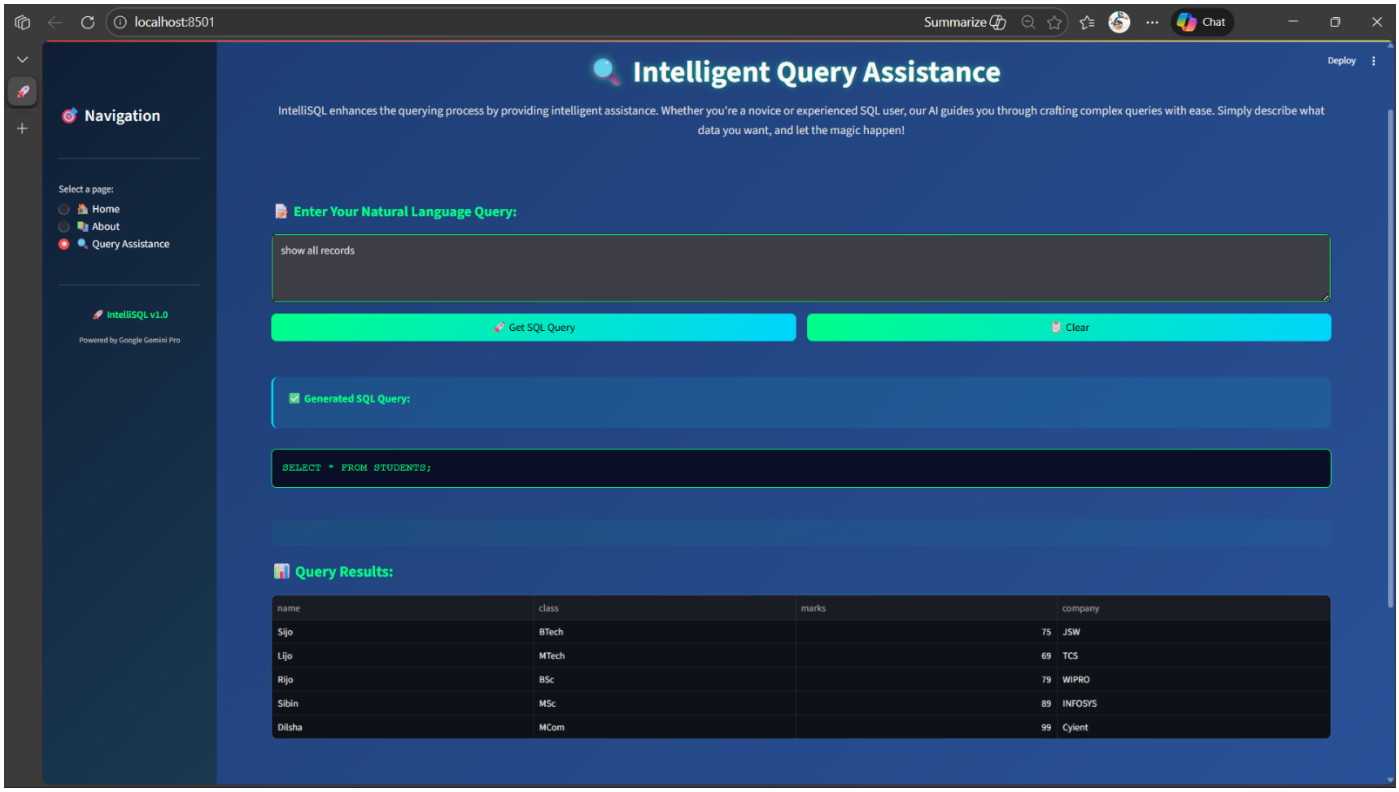
Overall Performance Conclusion

IntelliSQL demonstrates fast response time, accurate SQL generation, and stable performance under normal workloads.

3. RESULT

• The **IntelliSQL** project successfully converts natural language queries into accurate SQL statements using AI and NLP techniques.
• A simple web interface allows users to enter queries and instantly view the generated SQL output.
• The system provides real-time, context-aware SQL generation, enabling efficient database interaction.
• The lightweight and scalable design makes IntelliSQL suitable for both technical and non-technical users.
• Overall, the project achieves its goal of delivering a fast, reliable, and accessible natural language–to–SQL solution.

7.1Output screenshots:



8. Advantages and Disadvantages

Advantages

1. Accurate Natural Language to SQL Conversion

• IntelliSQL leverages large language models (LLMs) to accurately convert natural language queries into structured SQL statements.
• Context awareness helps the system understand user intent, table relationships, and query logic, resulting in meaningful SQL generation.

2. Reduced Development Time

• Using pre-trained LLMs (such as Gemini Pro or similar models) eliminates the need to manually write complex SQL parsing rules.
• This significantly reduces development effort and accelerates project implementation.

3. User-Friendly Interface

• Non-technical users can query databases without knowing SQL syntax.
• A simple web-based interface allows users to input plain English questions and receive SQL output instantly.

4. Improved Productivity

• Saves time for developers, analysts, and business users by automating query generation.
• Reduces manual errors commonly made while writing complex SQL queries.

5. Scalable and Flexible

• Can be integrated with multiple databases and scaled for multiple users.
• Suitable for educational, business intelligence, and data analysis applications.

Disadvantages

1. Dependency on Schema Understanding

• Accurate SQL generation depends on correct database schema input.
• Incorrect or incomplete schema information may lead to invalid queries.

2. Limited Handling of Complex Queries

• Very complex joins, nested queries, or highly optimized SQL may not always be generated correctly in early versions.

3. Lack of Explainability

• The LLM functions as a black box, offering limited explanation of how the SQL query was derived.
• This may reduce trust in critical database operations.

4. Internet and API Dependency

• Cloud-based LLM usage requires stable internet connectivity.
• API latency may affect real-time performance.

5. No Automatic Learning from Feedback

• The system does not automatically improve based on user corrections.
• Enhancements require prompt tuning or model updates.

9. CONCLUSION

The **IntelliSQL** project successfully demonstrates the application of artificial intelligence and large language models to automate the conversion of natural language queries into structured SQL statements. By integrating an AI-powered query generation model with a lightweight web-based interface, the system enables users to interact with databases without requiring prior knowledge of SQL.

IntelliSQL allows users to input queries in plain English and instantly generates accurate SQL queries, reducing manual effort, minimizing syntax errors, and improving productivity. This makes the system suitable for students, data analysts, developers, and business users who work with databases regularly.

The use of pre-trained language models ensures reliable query generation, while the simple and intuitive interface makes the system accessible to non-technical users. The modular architecture also supports scalability and future enhancements.

Overall, IntelliSQL provides an efficient, user-friendly, and intelligent solution for database interaction, highlighting the potential of AI in simplifying data access and decision-making processes.

10. FUTURE SCOPE

1. Support for Complex Queries

• Enhance the system to handle advanced SQL queries involving nested queries, joins, and optimization.

2. Multi-Database Compatibility

• Extend support to multiple database systems such as MySQL, PostgreSQL, and MongoDB.

3. Explainable AI Integration

• Provide explanations for generated SQL queries to improve transparency and user trust.

4. Offline and Edge Deployment

• Enable local or offline query generation for environments with limited internet connectivity.

5. Continuous Learning

• Incorporate user feedback to improve accuracy through prompt refinement or model fine-tuning.

6. Cloud-Based Deployment

• Deploy IntelliSQL as a cloud platform with features like query history, user profiles, and API access.

11.Appendix

Source code:

Requirements.txt:

streamlit

google-generativeai

python-dotenv

intellisql.py:

```
from dotenv import load_dotenv
import streamlit as st
import os
import google.generativeai as genai

# ------------------------------
# Page Configuration
# ------------------------------
st.set_page_config(
    page_title="IntelliSQL – AI-Powered SQL Generator",
    page_icon="🧠",
    layout="centered"
)

st.title("🧠 IntelliSQL")
st.markdown("Generate SQL queries from natural language using Google Gemini AI")

# ------------------------------
# Styling
# ------------------------------
st.markdown("""
<style>
.stApp {
    background: linear-gradient(135deg, #f5f7fa, #e4ecf7);
}
h1 {
    text-align: center;
    color: #1f3b73;
}
.result-box {
    padding: 16px;
    border-radius: 10px;
    background-color: white;
    border: 1px solid #dce3ea;
```

```
    font-size: 16px;
    box-shadow: 0 4px 10px rgba(0,0,0,0.05);
}
.stButton>button {
    width: 100%;
    height: 3em;
    font-size: 16px;
    font-weight: bold;
    border-radius: 10px;
    background: linear-gradient(90deg, #4facfe, #00f2fe);
    color: white;
    border: none;
}
.stButton>button:hover {
    background: linear-gradient(90deg, #43e97b, #38f9d7);
}
</style>
""", unsafe_allow_html=True)

# ------------------------------
# Load API Key
# ------------------------------
load_dotenv()
api_key = os.getenv("GOOGLE_API_KEY")
genai.configure(api_key=api_key)

# ------------------------------
# Initialize Gemini Model
# ------------------------------
model = genai.GenerativeModel("models/gemini-flash-latest")

# ------------------------------
# SQL Generation Function
# ------------------------------
def generate_sql(nl_query, database_type):
    prompt = f"""
    Convert the following natural language request into a valid {database_type} SQL query.
    Only return the SQL query.

    Natural Language Request:
    {nl_query}
    """
    response = model.generate_content(prompt)
    return response.text

# ------------------------------
# Main Application
```

```
# ----------------------------
def main():

    st.markdown("### 🗄️ Database Selection")
    db_type = st.selectbox(
        "Choose Database Type",
        ["MySQL", "PostgreSQL", "SQLite", "SQL Server", "Oracle"]
    )

    st.markdown("### 📝 Enter Natural Language Query")
    user_input = st.text_area(
        "Example: Show all employees earning more than 50000",
        height=150
    )

    if st.button("🚀 Generate SQL"):
        if user_input:
            try:
                with st.spinner("Generating SQL query..."):
                    sql_query = generate_sql(user_input, db_type)

                st.divider()
                st.markdown("### 📄 Generated SQL Query")
                st.markdown(
                    f"<div class='result-box'><code>{sql_query}</code></div>",
                    unsafe_allow_html=True
                )

            except Exception as e:
                st.error(f"⚠️ Error: {str(e)}")
        else:
            st.warning("⚠️ Please enter a natural language query.")

# ----------------------------
# Run App
# ----------------------------
if __name__ == "__main__":
    main()
```

demo link:
https://drive.google.com/file/d/16hCocvDEf3-yubNWc2tylNAaoTH_JhLg/view?usp=sharing

git hub & project link: https://github.com/TammaReddy123/IntelliSQL-Intelligent-SQL-Querying-with-LLMs-Using-Gemini-Pro-main.git