Project Report Format

1. INTRODUCTION:-

1.1 Project Overview

The *Intelligent SQL Querying with LLMs Using Gemini* project is designed to simplify database interactions by enabling users to retrieve information using natural language queries. Instead of manually writing SQL statements, users can enter questions in plain English, and the system automatically converts them into valid SQLite SQL queries using the Gemini Large Language Model (LLM).The application integrates a user-friendly interface, a language model for SQL generation, and a SQLite database for query execution. This approach improves accessibility for non-technical users while increasing efficiency and reducing query errors.

1.2 Purpose

The primary purpose of this project is to provide an intelligent and intuitive mechanism for database querying. The system aims to:

• Eliminate the need for advanced SQL knowledge
• Enable natural language-based data retrieval
• Improve query accuracy through AI assistance
• Reduce time required for database operations
• Enhance user productivity and experience

By leveraging the Gemini LLM, the solution bridges the gap between human language and structured database queries, making data exploration easier and more efficient.

2. IDEATION PHASE:-

2.1 Problem Statement:-

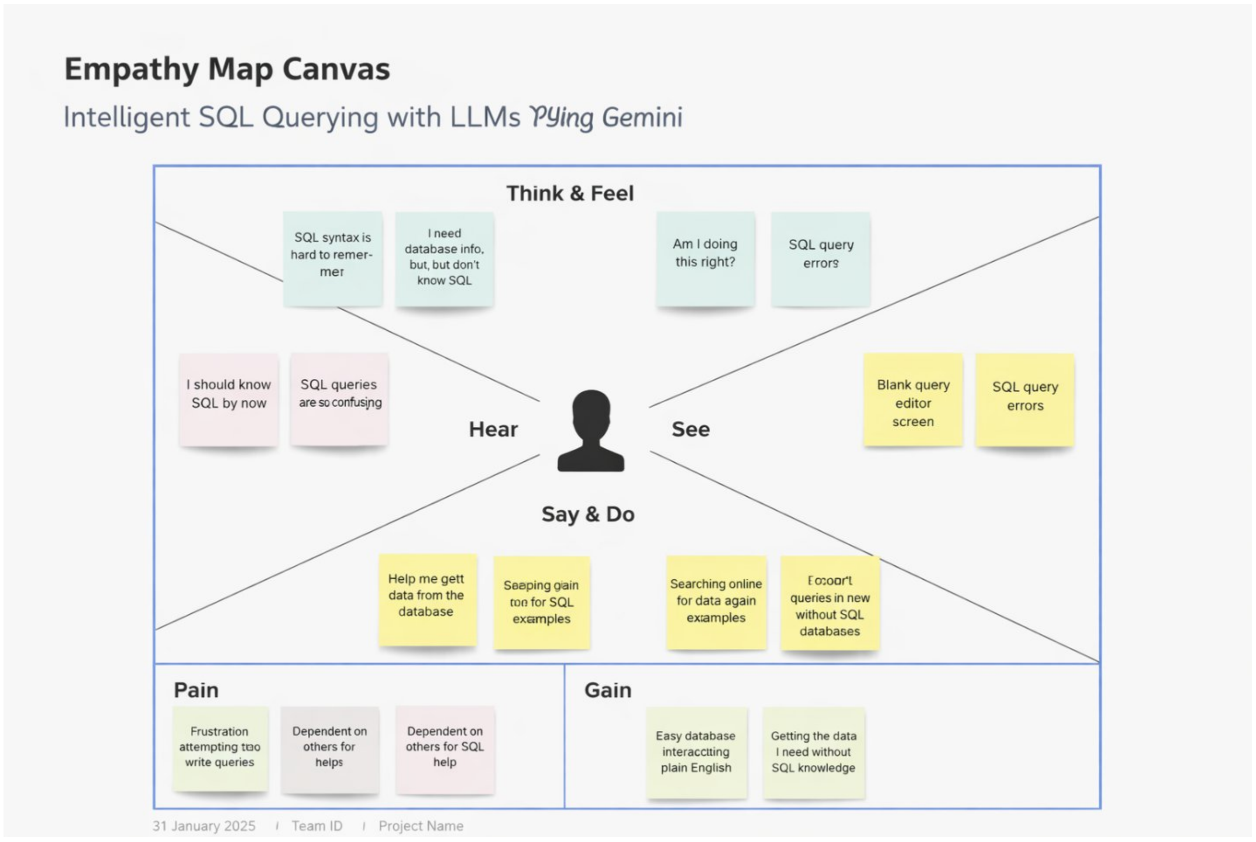
Problem Statement (PS)	**I am (Customer)**	**I'm trying to**	**But**	**Because**	**Which makes me feel**
PS-1	**I am** a student / beginner learning databases.	**I'm trying to** retrieve information from a SQL database.	**But I** find it difficult to write correct SQL queries.	**Because** SQL syntax and query structure are complex and easy to forget.	**Which makes me feel** frustrated and less confident while working with databases.

PS-2	**I am** a user who understands what data **I** need.	**I'm trying to** ask questions from the database quickly.	**But I** cannot always translate my questions into SQL queries.	**Because** writing queries requires technical knowledge.	**Which makes me feel** dependent on others or slowed down in my work.

2.2 Empathy Map Canvas:-

Empathy Map Canvas:
An empathy map is a visual tool used to understand users, their needs, and their challenges while interacting with a system.For the *Intelligent SQL Querying with LLMs Using Gemini* project, this canvas helps identify how users think, feel, and behave when working with databases and SQL queries. Many users struggle with SQL syntax, query errors, and data retrieval despite knowing what information they need.

By mapping user frustrations, expectations, and goals, the team gains deeper insight into real user problems. This understanding guides the design of a solution that allows users to interact with databases using simple natural language instead of complex SQL commands.


Empathy Map Canvas
Intelligent SQL Querying with LLMs Pying Gemini
Think & Feel
SQL syntax is hard to remer-mer
I need database info, but, but don't know SQL
Am I doing this right?
SQL query errors
Hear
I should know SQL by now
SQL queries are so confusing
See
Blank query editor screen
SQL query errors
Say & Do
Help me gett data from the database
Seaping giain ton for SQL examples
Searching online for data again examples
queries in new without SQL databases
Pain
Frustration attempting tao write queries
Dependent on others for helps
Dependent on others for SQL help
Gain
Easy database interaccting plain English
Getting the data I need without SQL knowledge
31 January 2025
Team ID
Project Name

2.3 Brainstorming:-

Step-1: Problem Statement Selection

During the ideation phase, our team discussed challenges faced by users while interacting with SQL databases. We observed that writing SQL queries requires knowledge of syntax and structure, which can be difficult for beginners and non-technical users.

Based on this discussion, the team identified the following problem:

Problem Statement:
Users often struggle to write accurate SQL queries to retrieve information from databases due to limited SQL knowledge and syntax complexity.

Step-2: Brainstorming & Idea Listing

The team generated multiple solution ideas to address this problem:

- Develop a system to convert natural language into SQL queries
- Create an intelligent SQL query assistant
- Build an AI-based database interaction tool
- Implement a chatbot-style database interface
- Use a Large Language Model for query generation

These ideas were centered around simplifying database querying and improving user accessibility.

Step-3: Idea Prioritization

After evaluating feasibility, innovation, and practical usefulness, the team selected the following idea:

Selected Idea:
Intelligent SQL Querying with LLMs Using Gemini

Justification:

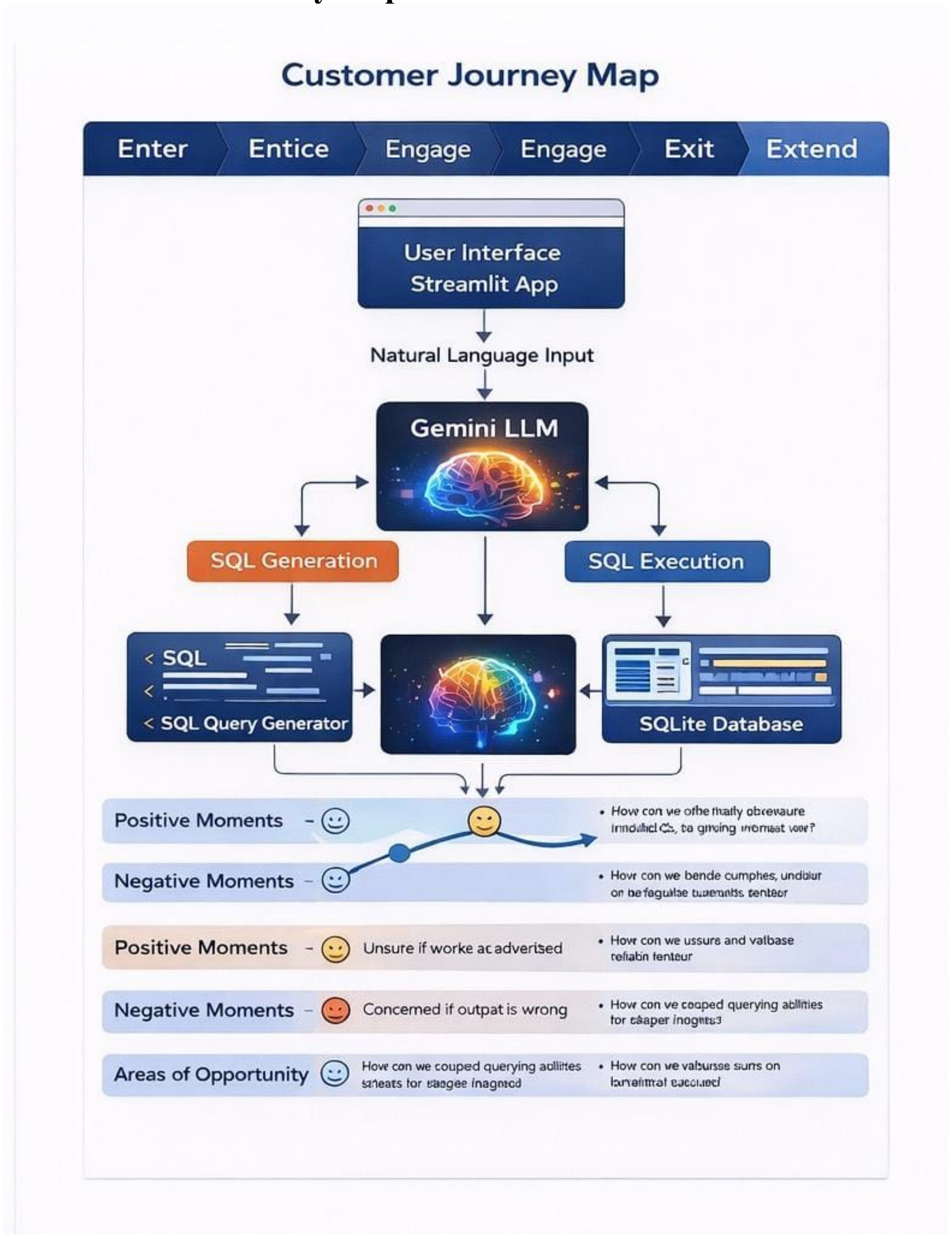
- Enables users to query databases using natural language
- Reduces dependency on SQL syntax knowledge
- Leverages modern LLM capabilities for query generation
- Technically feasible and highly practical

Proposed Solution Overview

The proposed system allows users to enter questions in natural language. The Gemini Large Language Model interprets the input and generates corresponding SQL queries, which are executed on a SQLite database. The results are then displayed through a Streamlit interface.

3. REQUIREMENT ANALYSIS:-

3.1 Customer Journey map:-


Customer Journey Map
Enter
Entice
Engage
Engage
Exit
Extend
User Interface Streamlit App
Natural Language Input
Gemini LLM
SQL Generation
SQL Execution
SQL
SQL Query Generator
SQLite Database
Positive Moments
Negative Moments
Positive Moments
Unsure if worke ac advertised
How con we ussure and valbase reliabin tenteur
Negative Moments
Concerned if outpat is wrong
How con ve cooped querying ablities for eaaper inognts
Areas of Opportunity
How con we couped querying aollites
How con we valbursse surrs on

3.2 Solution Requirement:-

Functional Requirements:

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Question Handling	• User can enter questions in plain English • System accepts flexible sentence formats • Input validation before processing
FR-2	LLM-based SQL Generation	• Convert user question into SQL query • Use Gemini model for query generation • Ensure SQLite-compatible syntax • Return only SQL without explanations
FR-3	Database Interaction	• Execute generated SQL query • Connect to SQLite database (data.db) • Fetch results from STUDENTS table • Handle invalid queries safely
FR-4	Output Visualization	• Display query results in UI • Show data in readable format (table/list) • Display empty results gracefully • Show execution errors clearly
FR-5	Transparency & Learning	• Show generated SQL query to user • Allow user to verify query • Help users understand SQL logic
FR-6	System Stability	• Handle API errors (quota/model issues) • Handle SQL execution errors • Display user-friendly error messages

Non-functional Requirements:

Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	System should provide a simple and intuitive interface where users can easily input questions and view results without SQL knowledge.
NFR-2	**Security**	API keys and database access must be handled securely. Sensitive data such as credentials should not be exposed in the UI or logs.
NFR-3	**Reliability**	System should consistently generate valid SQL queries and execute them correctly under normal operating conditions.
NFR-4	**Performance**	Query generation and execution should occur with minimal delay to ensure smooth user experience.
NFR-5	**Availability**	System should remain accessible whenever users interact with the application, subject to API service availability.
NFR-6	**Scalability**	System design should allow future extension to larger databases, additional tables, or alternative LLM models.

3.3 Data Flow Diagram:-

Data Flow Diagrams:

A Data Flow Diagram (DFD) illustrates how data moves through the *Intelligent SQL Querying with LLMs Using Gemini* system.It represents how user questions are processed, how the LLM generates SQL queries, how the database executes them, and how results are returned to the user. The diagram highlights the interaction between the user interface, Gemini model, SQL generator logic, and database.

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
End User	Ask Questions in Natural Language	USN-1	As a user, I can enter my question in plain English instead of writing SQL queries.	User can type questions in an input box.	High	Sprint-1
		USN-2	As a user, I can generate an SQL queryautomatically from my question.	System converts text into valid SQL.	High	Sprint-1
		USN-3	As a user, I can execute the generated SQL query on the database.	Clicking button runs query without errors.	High	Sprint-1
		USN-4	As a user, I can view query results in a readable format.	Results displayed in table/list format.	High	Sprint-1
		USN-5	As a user, I can see the generated SQL query for learning purposes.	Generated SQL visible on screen.	Medium	Sprint-2
Administrator		USN-6	As an admin, I can manage database records.	Database updates reflect in queries.	Medium	Sprint-2
		USN-7	As an admin, I can ensure correct schema for LLM interaction.	System works with defined table structure.	Medium	Sprint-2

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
System	Query Intelligence	USN-8	System should validate SQL queries before execution.	Invalid queries handled safely.	High	Sprint-1
		USN-9	System should handle LLM response errors gracefully.	User receives clear error messages.	High	Sprint-1

3.4 Technology Stack:-

Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	Interface where user enters natural language questions and views SQL/results	**Streamlit (Python Web UI)**
2.	Application Logic-1	Handles user input, prompt creation, and response processing	**Pytho n**
3.	Application Logic-2	Generates SQL queries from natural language questions	**Google Gemini API (LLM)**
4.	Application Logic-3	Executes generated SQL queries and fetches results	**SQLite3 (Python DB Connector)**
5.	Databas e	Stores structured student data used for querying	**SQLite (data.db)**
6.	File Storage	Stores database file locally	**Local File System**
7.	External API-1	Converts English questions into SQL queries	**Google Gemini API**
8.	Infrastructure (Server / Cloud)	Runs application locally via browser	**Local System (VS Code + Streamlit Server)**

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Frameworks used to build application UI and logic	**Streamlit, Python, SQLite**
2.	Security Implementations	Protects API keys and prevents direct exposure	**Environment Variables (.env), API Key Handling**
3.	Scalable Architecture	System can be extended to larger databases / models	**Modular Python Design**
4.	Availabilit y	App accessible whenever local server is running	**Streamlit Local Server**
5.	Performanc e	Lightweight queries and fast response for small datasets	**SQLite + Gemini Flash Model**

4. PROJECT DESIGN:-

4.1 Problem Solution Fit:-

Problem Statement

Interacting with databases requires knowledge of Structured Query Language (SQL), which creates a significant barrier for non-technical users. Many users understand the information they need but are unable to construct correct SQL queries. This results in errors, delays, and dependency on technical personnel.

Customer / Target Users

• Non-technical users
• Students and beginners learning databases
• Business users and analysts
• Users requiring quick access to data insights

Current Challenges

• Lack of SQL knowledge
• Difficulty in understanding query syntax
• High probability of query errors
• Time-consuming data retrieval process
• Dependency on developers or database experts

Existing Situation

Users must manually write SQL queries while understanding database structure and syntax rules. Even simple data requests require technical effort, reducing efficiency and slowing decision-making processes.

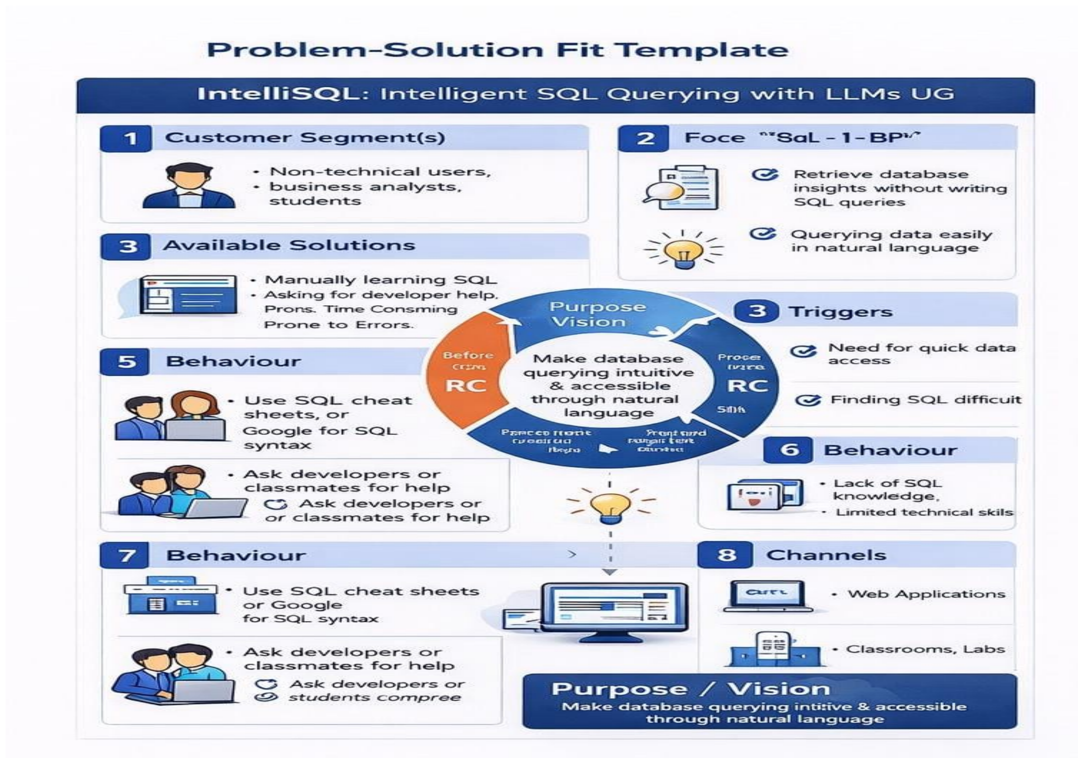
Proposed Solution

The proposed solution, **IntelliSQL**, leverages a Large Language Model (Gemini) to translate natural language questions into valid SQL queries. Users can interact with the database using plain English instead of writing SQL commands.Example interactions:

• "Show all students"
• "Display highest marks"
• "Students working at Infosys"

The system automatically generates and executes the corresponding SQL query and presents the results.

TEMPLATE:-


Problem-Solution Fit Template
IntelliSQL: Intelligent SQL Querying with LLMs UG
1 Customer Segment(s)
• Non-technical users,
• business analysts, students
2 Foce "SqL - 1 - BP"
Retrieve database insights without writing SQL queries
Querying data easily in natural language
3 Available Solutions
• Manually learning SQL
• Asking for developer help. Prons. Time Consming Prone to Errors.
Purpose Vision
Before RC
Make database querying intuitive & accessible through natural language
RC
3 Triggers
Need for quick data access
Finding SQL difficult
5 Behaviour
• Use SQL cheat sheets, or Google for SQL syntax
• Ask developers or classmates for help
Ask developers or or classmates for help
6 Behaviour
• Lack of SQL knowledge,
• Limited technical skils
7 Behaviour
• Use SQL cheat sheets or Google for SQL syntax
• Ask developers or classmates for help
Ask developers or students compree
8 Channels
• Web Applications
• Classrooms, Labs
Purpose / Vision
Make database querying intitive & accessible through natural language

4.2 Proposed Solution:-

Proposed Solution Template:

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Users find SQL difficult and require technical knowledge to query databases.
2.	Idea / Solution description	System converts natural language questions into SQL queries using Gemini LLM and executes them on a SQLite database.
3.	Novelty / Uniqueness	Eliminates need for SQL expertise by enabling human-like interaction with databases.
4.	Social Impact / Customer Satisfaction	Improves accessibility, saves time, and increases productivity for non-technical users.
5.	Business Model (Revenue Model)	Can be offered as a subscription-based SaaS tool or enterprise solution.
6.	Scalability of the Solution	Extendable to multiple databases, domains, and cloud environments.

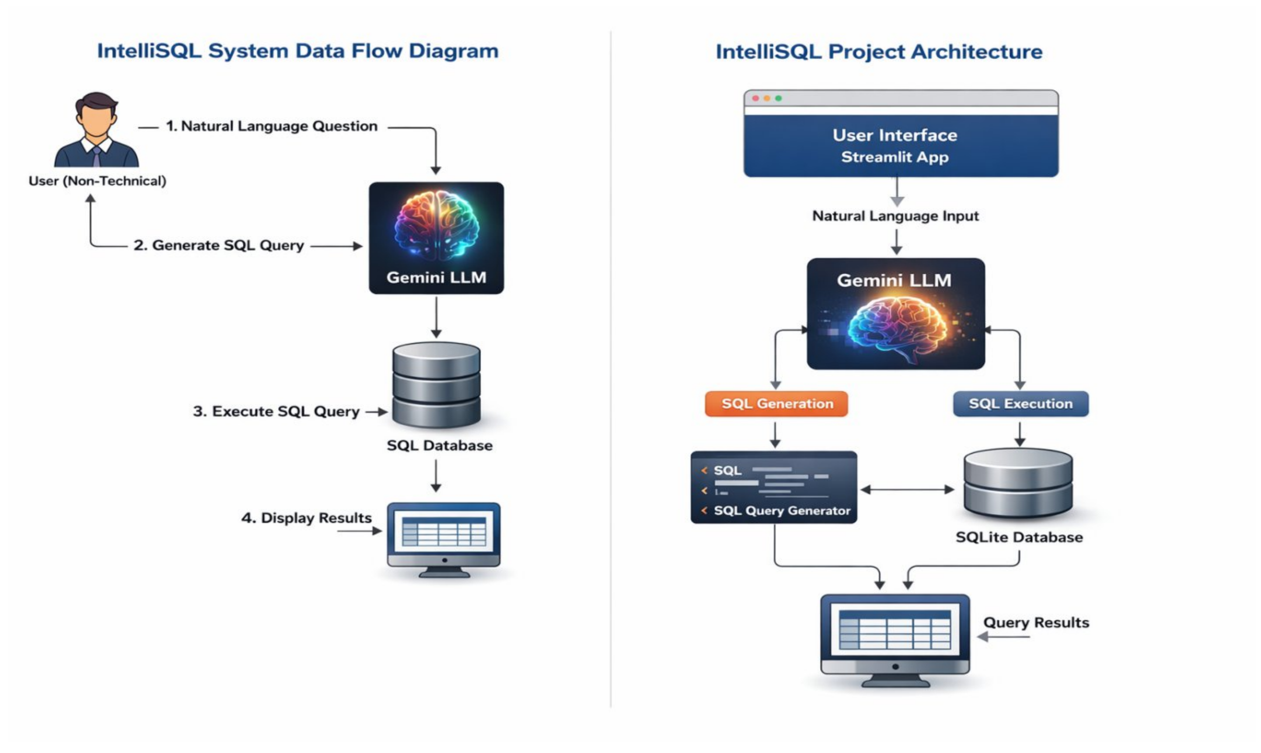
4.3 Solution Architecture:-

Solution Architecture:

The solution architecture defines how the system transforms user questions into executable SQL queries using a Large Language Model. Its goals are to:

• Provide an intuitive interface for users to ask questions in natural language.
• Use the Gemini LLM to translate user input into valid SQLite SQL queries.
• Execute generated queries securely on the database and return results.
• Ensure accuracy, performance, and reliability of query generation.
• Define clear interaction between UI, LLM engine, and database layer.
• Support extensibility for additional databases and advanced query logic.

Example - Solution Architecture Diagram:



5. PROJECT PLANNING & SCHEDULING:-

5.1 Project Planning:-

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Database Setup	USN-1	As a developer, I can create the SQLite database for storing student data	2	High	Team
Sprint-1	Database Setup	USN-2	As a developer, I can design the STUDENTS table schema	1	High	Team
Sprint-2	Database Setup	USN-3	As a developer, I can insert sample records into the database	2	High	Team
Sprint-1	Gemini Integration	USN-4	As a developer, I can configure the Gemini API key and environment	3	High	Team
Sprint-1	Gemini Integration	USN-5	As a developer, I can design a prompt for SQL generation	3	High	Team
Sprint-2	SQL Generation	USN-6	As a user, I can enter a question in natural language	2	High	Team
Sprint-2	SQL Generation	USN-7	As a system, I can generate SQL queries using Gemini	5	High	Team
Sprint-2	SQL Execution	USN-8	As a system, I can execute generated SQL on SQLite database	3	High	Team
Sprint-2	User Interface	USN-9	As a user, I can view the generated SQL query	1	Medium	Team

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-2	Error Handling	USN-10	As a user, I can view the generated SQL query	2	High	Team

Project Tracker, Velocity & Burndown Chart: (4 Marks)

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	11	10 Days	29 Jan2025	07 Feb 2025	11	07 Feb 2025
Sprint-2	13	10 Days	8 Feb 2025	17 Feb 2025	13	17 Feb 2025

Velocity

Velocity = Total Story Points Completed / Number of Sprints

Total Story Points = 11 + 13 = **24**
Number of Sprints = **2**

Velocity = 24 / 2
Velocity = **12 Story Points per Sprint**

6. FUNCTIONAL AND PERFORMANCE TESTING:-

6.1 Performance Testing:-

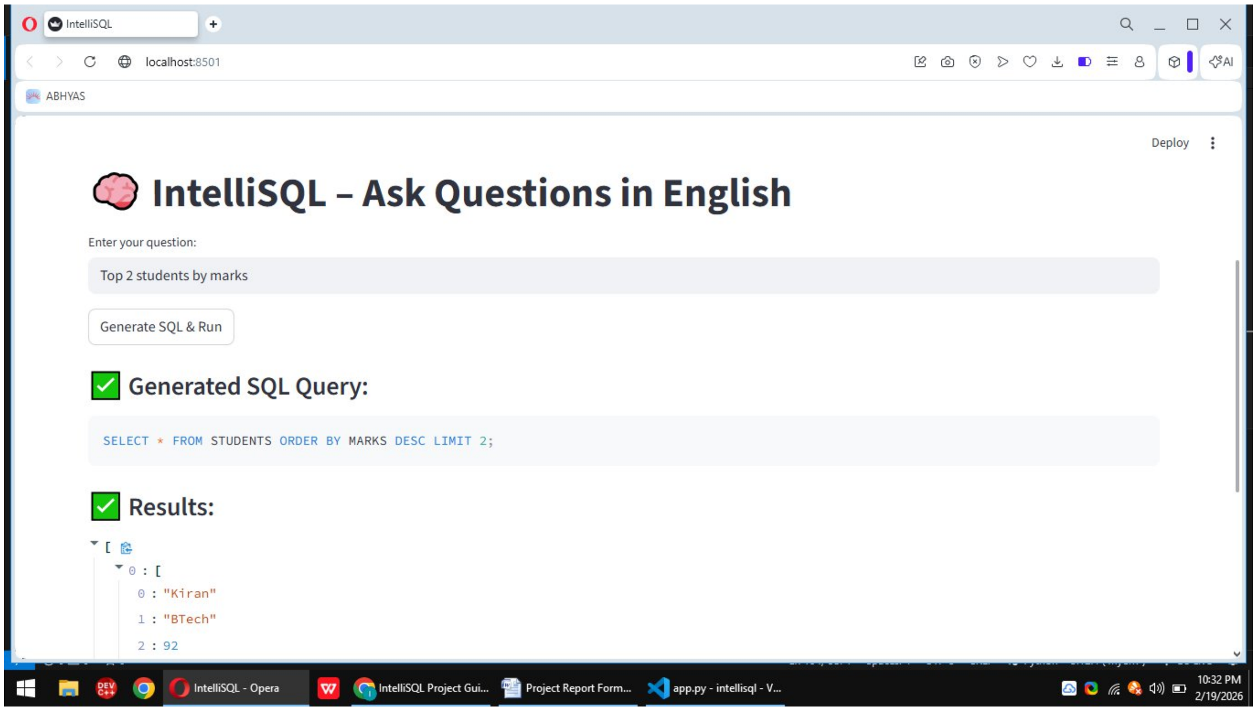
Test Scenarios & Results

Test Case ID	Scenario (What to test)	Test Steps (How to test)	Expected Result	Actual Result	Pass/Fail
FT-01	Text Input Validation (e.g., topic, job title)	Enter valid and invalid text in input fields	Valid inputs accepted, errors for invalid inputs	System accepted valid queries and showed warning for empty input	Pass
FT-02	SQL Query Generation	Enter natural language question and click "Generate SQL &	Correct SQL query should be generated	SQL queries generated correctly based on	Pass

		Run"		user input	
FT-03	SQL Execution on Database	Run generated SQL query on SQLite database	Query should execute without errors	Queries executed successfully and results displayed	Pass
FT-04	API Connection Check	Check if API key is correct and model responds	API responds successfully	API connected and model responded properly	Pass
PT-01	Response Time Test	Measure time taken for SQL generation	System should handle without crash	Application handled queries smoothly	Pass
PT-02	Multiple Query Handling	Execute multiple queries sequentially	System should handle without crash	Application handled queries smoothly	Pass
PT-03	Error Handling Test	Provide invalid or incorrect questions	Proper error message should appear	Errors displayed correctly without system failure	Pass

7. RESULTS:-

7.1 Output Screenshots:-



🧠 IntelliSQL – Ask Questions in English

Enter your question:

Top 2 students by marks

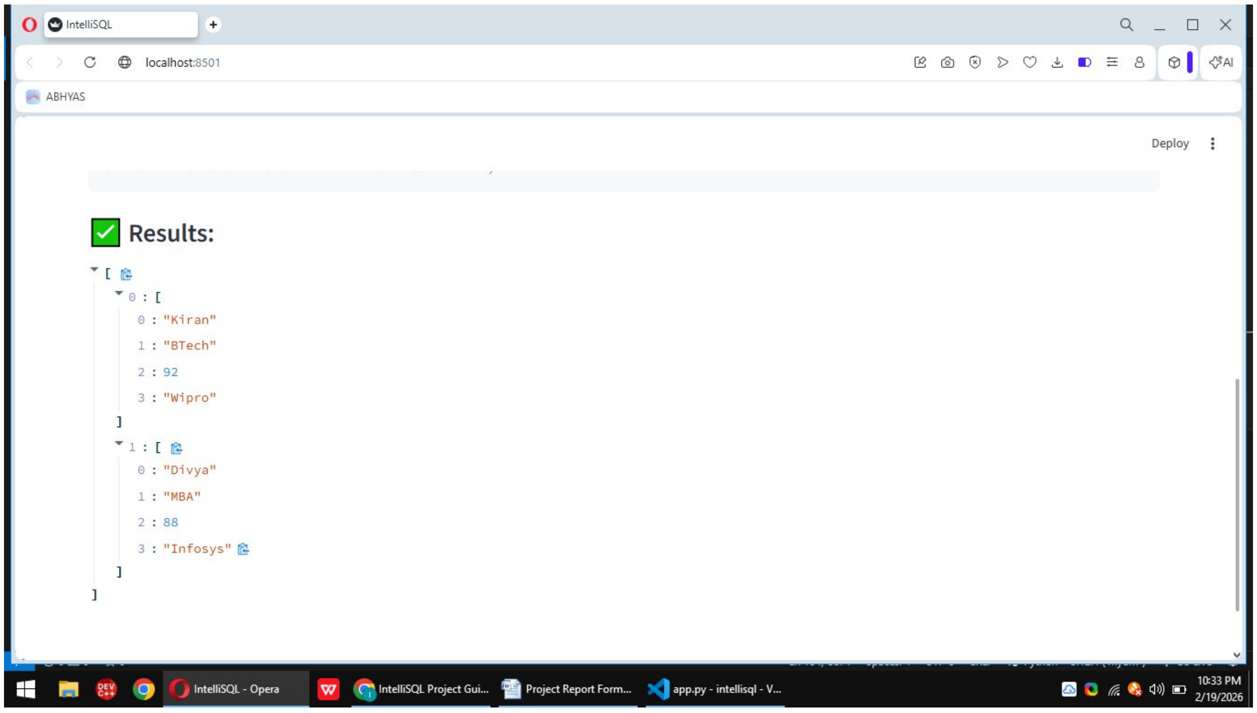
Generate SQL & Run

✅ Generated SQL Query:

```
SELECT * FROM STUDENTS ORDER BY MARKS DESC LIMIT 2;
```

✅ Results:

```
[
  0 : [
    0 : "Kiran"
    1 : "BTech"
    2 : 92
```



✅ Results:

```
[
  0 : [
    0 : "Kiran"
    1 : "BTech"
    2 : 92
    3 : "Wipro"
  ]
  1 : [
    0 : "Divya"
    1 : "MBA"
    2 : 88
    3 : "Infosys"
  ]
]
```

8. ADVANTAGES & DISADVANTAGES:-

Advantages

• Allows users to query databases using natural language instead of SQL
• Reduces dependency on technical SQL expertise
• Saves time by automatically generating queries
• Improves productivity and ease of data retrieval
• Minimizes syntax errors in SQL queries
• Provides a user-friendly and interactive experience
• Enables faster data exploration and analysis
• Easily extendable to other databases and use cases

Disadvantages

• Accuracy depends on the quality of the language model response
• Requires a valid API key and internet connectivity
• May generate incorrect queries for ambiguous questions
• Performance affected by API latency or quota limits
• Limited by predefined database schema and prompt design
• Dependency on third-party AI service availability.

9. CONCLUSION:-

The *Intelligent SQL Querying with LLMs Using Gemini* project demonstrates how Large Language Models can simplify and enhance database interactions. By enabling users to submit queries in natural language and automatically converting them into valid SQL statements, the system reduces the need for technical expertise and minimizes human errors.

The integration of the Gemini LLM with a SQLite database and a user-friendly interface provides an efficient, accessible, and scalable solution for data retrieval. The project highlights the practical application of AI in improving usability, productivity, and automation within data-driven systems.

Overall, the solution successfully bridges the gap between human language and structured query systems, making database operations more intuitive and efficient. Future enhancements can further improve accuracy, performance, and support for complex queries and multiple database types.

10. FUTURE SCOPE:-

The *Intelligent SQL Querying with LLMs Using Gemini* project can be further enhanced to expand its capabilities and real-world applicability. Possible future improvements include:

• Support for multiple databases such as MySQL, PostgreSQL, and SQL Server
• Improved handling of complex queries, joins, and nested conditions
• Enhanced prompt engineering for higher SQL accuracy
• Integration of user authentication and role-based access control

• Addition of query history and saved queries feature
• Implementation of offline or hybrid model options
• Performance optimization and caching mechanisms
• Advanced error detection and query correction suggestions
• Visualization of query results through charts and dashboards
• Deployment as a web service or enterprise-level application

These enhancements can significantly improve scalability, usability, and practical adoption of the system.

11. APPENDIX:-

SOURCE CODE:-

App.py:-

```
import streamlit as st
import sqlite3
import google.generativeai as genai
import os
import re
from dotenv import load_dotenv

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

# Create Model
model = genai.GenerativeModel("gemini-2.5-flash")

# Better Prompt
BASE_PROMPT = """
You are an expert SQL generator.

Database: SQLite
Table Name: STUDENTS

Columns:
- NAME (TEXT)
- CLASS (TEXT)
- SECTION (TEXT)
- MARKS (INTEGER)
- COMPANY (TEXT)

Rules:
```

```
- Return ONLY valid SQL query
- Use SQLite syntax
- No explanations
- No markdown
- Table name must be STUDENTS

Supported Operations:
- SELECT
- WHERE
- COUNT
- AVG
- MAX
- MIN
- ORDER BY
- LIMIT

Examples:

Question: Show all students
SQL: SELECT * FROM STUDENTS;

Question: Students working at Infosys
SQL: SELECT * FROM STUDENTS WHERE COMPANY='Infosys';

Question: Highest marks
SQL: SELECT MAX(MARKS) FROM STUDENTS;

Question: Average marks
SQL: SELECT AVG(MARKS) FROM STUDENTS;

Question: Count students
SQL: SELECT COUNT(*) FROM STUDENTS;

Question: Top 3 students by marks
SQL: SELECT * FROM STUDENTS ORDER BY MARKS DESC LIMIT 3;
"""

# ✅ Clean Gemini Output (VERY IMPORTANT)
def clean_sql_response(text):
    text = text.replace("```sql", "").replace("```", "")
    text = text.strip()

    # Remove accidental newlines / spaces
    return text
```

```python
def get_gemini_response(question):
    response = model.generate_content(BASE_PROMPT + "\nQuestion: " + question)

    raw_text = response.text
    return clean_sql_response(raw_text)

def run_sql_query(query):
    conn = sqlite3.connect("data.db")
    cursor = conn.cursor()
    cursor.execute(query)
    rows = cursor.fetchall()
    conn.close()
    return rows

# UI
st.set_page_config(page_title="IntelliSQL", layout="wide")

st.write("Example Questions You Can Try:")
st.write("""
• Show all students
• Students working at Infosys
• Highest marks
• Lowest marks
• Average marks
• Count students
• Students with marks above 80
• Students from BTech
• Top 2 students by marks
""")
st.title("🧠 IntelliSQL – Ask Questions in English")



question = st.text_input("Enter your question:")

if st.button("Generate SQL & Run"):
    if question:
        sql_query = get_gemini_response(question)

        st.subheader("✅ Generated SQL Query:")
        st.code(sql_query, language="sql")

        try:
            results = run_sql_query(sql_query)
            st.subheader("✅ Results:")
            st.write(results)

        except Exception as e:
            st.error(f"SQL Error: {e}")
```

sql.py:-

```
import sqlite3

conn = sqlite3.connect("data.db")
cursor = conn.cursor()

cursor.execute("DROP TABLE IF EXISTS STUDENTS")

cursor.execute("""
CREATE TABLE STUDENTS (
    NAME TEXT,
    CLASS TEXT,
    SECTION TEXT,
    MARKS INTEGER,
    COMPANY TEXT
)
""")

cursor.execute("INSERT INTO STUDENTS VALUES ('Ravi', 'BTech', 'A', 85, 'Infosys')")
cursor.execute("INSERT INTO STUDENTS VALUES ('Anu', 'MCom', 'B', 78, 'TCS')")
cursor.execute("INSERT INTO STUDENTS VALUES ('Kiran', 'BTech', 'A', 92, 'Wipro')")
cursor.execute("INSERT INTO STUDENTS VALUES ('Divya', 'MBA', 'C', 88, 'Infosys')")

conn.commit()
conn.close()

print("Database Ready ✅")
```

Requirments.txt:-

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

API KEY:-

```
GOOGLE_API_KEY=AIzaSyAxb-kS-lKCpQDGS2aohbPmSvvCUrysOEk
```

GitHub & Project Demo Link:-

GitHub Link:-

https://github.com/Indu-varshini/SmartBridge-project-.git

Project Demo Link:-

https://github.com/Indu-varshini/SmartBridge-project-/blob/main/SmartBridge%20project.mp4