

## Project Design Phase-II

### Data Flow Diagram & User Stories

|               |   |
|---------------|---|
| Date          | 19 February 2026  |
| Team ID       | LTVIP2026TMIDS66135   |
| Project Name  | IntelliSQL: Intelligent SQL Querying with LLMs Using Gemini Pro |
| Maximum Marks | 4 Marks   |

#### Data Flow Diagrams:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

The Data Flow Diagram (DFD) for **IntelliSQL** illustrates how information moves from a simple human question to a structured database result, utilizing the **Gemini Flash** engine as the central processing unit.

#### Core Components of the DFD

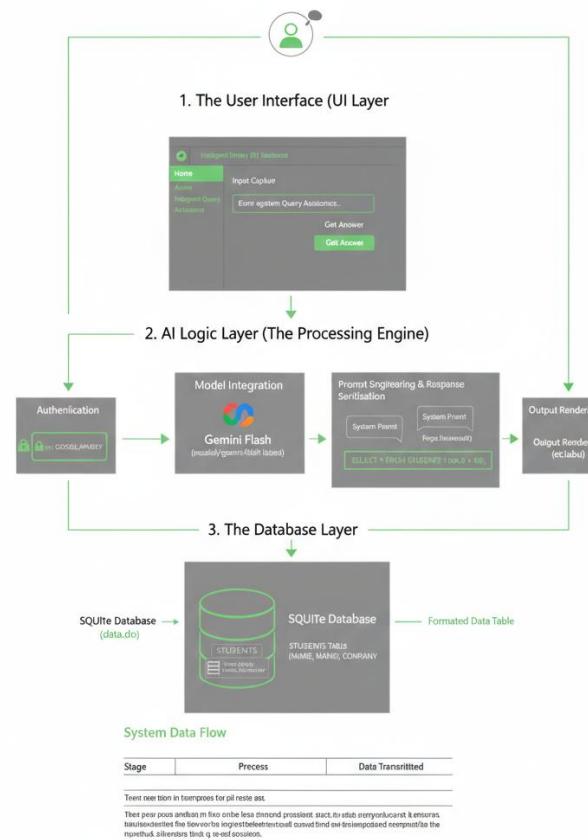
- **External Entity (The User):** The starting point of the flow where the user interacts with the **Streamlit UI** to input a natural language query.
- **The Process (AI Logic Layer):** This is the "brain" of the system that integrates the **Google Generative AI SDK** to communicate with the **Gemini Flash** model.
- **The Data Store (SQLite):** A local database file (data.db) that holds the persistent STUDENTS table records.

---

#### Detailed Data Movement

1. **Input Stream:** The user provides a text-based request (e.g., "Who has more than 90 marks?") via the frontend.
2. **Request Transformation:** The system retrieves the GOOGLE\_API\_KEY from the secure .env file to authenticate the session. It then combines the user's input with a predefined **System Prompt** that explains the table structure (NAME, CLASS, MARKS, COMPANY).

3. **AI Processing:** This combined data packet is sent to the **Gemini Flash** model, which outputs a raw SQL query string.
4. **Data Cleaning:** Before hitting the database, the response passes through a **Regex (Regular Expression)** filter to ensure only a valid SELECT statement is extracted, removing any conversational "noise".
5. **Database Retrieval:** The cleaned SQL is executed against the STUDENTS table in the **SQLite** database via the `read_query` function.
6. **Output Stream:** The database returns the matching rows, which are then rendered into a formatted table on the user's screen.



## 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  |
|---------------------|-------------------------------|-------------------|--|--|----------|----------|
| Administrator       | Environment Setup             | USN-1             | As an admin, I want to securely store my Google API key in a .env file so it is not exposed in source code.    | The app loads the key via os.getenv and connects to the Gemini API.                    | High     | Sprint-1 |
| Administrator       | Database Initialization       | USN-2             | As an admin, I want to initialize a SQLite database with a STUDENTS table to store records.                    | Running sql.py creates data.db with Name, Class, Marks, and Company columns.           | High     | Sprint-1 |
| Customer (Web)      | Home Page                     | USN-3             | As a user, I want a professional landing page with a list of offerings so I understand the app's capabilities. | The Home page displays a "Wide Range of Offerings" list and a welcome title.           | Medium   | Sprint-1 |
| Customer (Web)      | Intelligent Querying          | USN-4             | As a non-technical user, I want to enter natural language questions to generate valid SQL automatically.       | The system translates English into a functional SELECT statement using Gemini Flash.   | High     | Sprint-1 |
| Customer (Web)      | Data Retrieval                | USN-5             | As a user, I want to see query results in a table format so I can analyze student data easily.                 | Results from data.db are rendered in a structured table within the Streamlit UI.       | High     | Sprint-1 |
| Administrator       | Query Extraction              | USN-6             | As an admin, I want the app to use Regex to isolate SQL code from the LLM's conversational text.               | The re.search function extracts only the SELECT query, preventing execution errors.    | High     | Sprint-2 |
| Customer (Web user) | Navigation                    | USN-7             | As a user, I want a sidebar radio menu to navigate between Home, About, and the Query Tool.                    | Clicking sidebar options updates the main page content instantly without page reloads. | Medium   | Sprint-1 |
| Customer (Web user) | User Interface                | USN-8             | As a user, I want a dark-themed UI with green accents to enhance readability and professional feel.            | Custom CSS correctly applies the #2E2E2E background and #4CAF50 green headers.         | Low      | Sprint-2 |
| Administrator       | Data Management               | USN-9             | As an admin, I want to insert sample records for testing, such as Sijo, Lijo, and Dilsha.                      | The database successfully stores records with  | High     | Sprint-1 |

| User Type      | Functional Requirement (Epic) | User Story Number | User Story / Task   | Acceptance criteria   | Priority | Release  |
|----------------|-------------------------------|-------------------|---|---|----------|----------|
|                |                               |                   |   | specific Marks and Company names.   |          |          |
| Customer (Web) | Information Access            | USN-10            | As a user, I want an "About" page to learn how IntelliSQL uses LLM architecture for querying. | The About page displays descriptive text and the project's vision for intuitive querying. | Low      | Sprint-2 |