SQL Chatbot Documentation

1. **Overview of the Project:**
   This project is a **SQL Chatbot** that allows users to pose natural-language questions (in plain English) about invoice-related data stored in a MySQL database. The chatbot translates those questions into SQL queries (via OpenAI GPT), executes those queries against the database, **normalizes** the retrieved data if required, and returns the cleaned or normalized results back to the user.
2. **Key Capabilities:**
   a. **Natural Language to SQL**: Uses OpenAI to transform user questions into relevant SQL queries.
   b. **Data Normalization**: Identifies anomalies or text-based representations of numbers and dates, and normalizes them for consistent data handling (e.g., "Thirty Thousand" → 30000).
   c. **Dynamic Query Refinement**: If the queries contain subqueries or anomalies (e.g., LIMIT in the wrong place), the system attempts to resolve them before executing.
   d. **All-VARCHAR Columns**: The underlying MySQL database uses only VARCHAR columns. This means the system must carefully handle numeric and date-based comparisons that are stored as text.
3. **Tech Stack:**
   a. **Next.js** (frontend + backend):
      i. The UI is built in Next.js with React components.
      ii. The server-side logic (API) is implemented via Next.js API routes.
   b. **OpenAI** (LLM / GPT):
      i. Uses OpenAI's language models to parse user intent and generate SQL queries.
      ii. Used also to normalize data by applying text-to-numeric or text-to-date conversions.
   c. **MySQL**:
      i. The relational database contains tables for Vendors, Users, Invoices, InvoiceItems, and Payments.
      ii. All columns are stored as VARCHAR (255).
   d. **Node.js**:
      i. The underlying runtime for Next.js to run server-side code.
4. **How the AI part of the application works (Step by Step):**
   4.1 API Route (`route.js`)
   Below is a breakdown of the logic in **`route.js`**:

1. **Imports**
    - **OpenAI** from the `OpenAI` library (for GPT calls).
    - **pool** from `@/utils/db` (for MySQL connection pooling).
2. **system prompt Construction**
   A template string that sets the context for the AI assistant. This includes details about the database schema (tables and columns) and instructs the model on how to process queries. The prompt also clarifies that any unrelated questions should return "Wrong Prompt".
3. **query Prompt**
   Guides the model to generate SQL queries if the user's question is relevant to the database. Explains that all columns are VARCHAR and how to handle text-based numbers or dates in the WHERE clauses.
4. **Second Prompt (Normalization)**
   Creates a new prompt describing the normalization rules. It submits **dbData** to OpenAI with instructions on how to normalize:
    - Convert text-based numbers to numeric (keep both original and normalized if needed).
    - Convert text-based dates to YYYY-MM-DD.
    - Check if strings represent Boolean values.
    - Detect `null` values and replace them with empty JSON data while retrieving.
    - Remove passwords or sensitive info if present.

   Receives the response and parses it into a JSON object of the form {columns, data}.
5. **Create Temporary Table**
   Drops any existing temporary table named **TempNormalizedData**. Dynamically creates a new temporary table with the same columns as returned in the normalized data. All columns remain VARCHAR (255) to stay consistent with the original design. Insert each row of the normalized data into this temporary table.
6. **Third Prompt (Refined Operation Query)**
   Instructs the model to generate a final SQL query **against the temporary normalized data** (i.e., **TempNormalizedData**) for the original user question. Receives the refined SQL query.

7. **Resolve Potential Subquery Issues**
   Uses the **resolveLimitSubquery** function to handle any subqueries that might cause errors (e.g., a `LIMIT` clause inside an `IN` statement).
8. **Execute Refined Query**
   - Runs the refined query on **TempNormalizedData**.
   - Formats the result using **format Response**.
   - Returns the result in JSON to the client.
9. **Error Handling**
   If any step fails, returns a **500** response with a generic error message.

5. **Alternatives & Possible Enhancements:**

1. **Fine-Tuned Model**: If you have a very specific domain, a fine-tuned GPT model (or a local LLM) might yield better or more predictable queries.

2. **Use Pandas AI**:

- Prompt Pandas AI to "Normalize any textual representations of numbers or dates in the following Dataframe."
- Pandas AI (via an LLM) can parse each column and apply transformations, returning a new Dataframe with standardized numeric or date values.

3. **Non-AI approach:**

Maintain a dictionary of textual representations to numeric values. Regex for patterns.

4. **Alternative solution for Temporary Table Creation:**

1. Create an in-memory pandas dataframe and re-use it.

2. If the data is huge, consider gradually **cleaning or normalizing** them at ingestion or in a background job, rather than on demand.

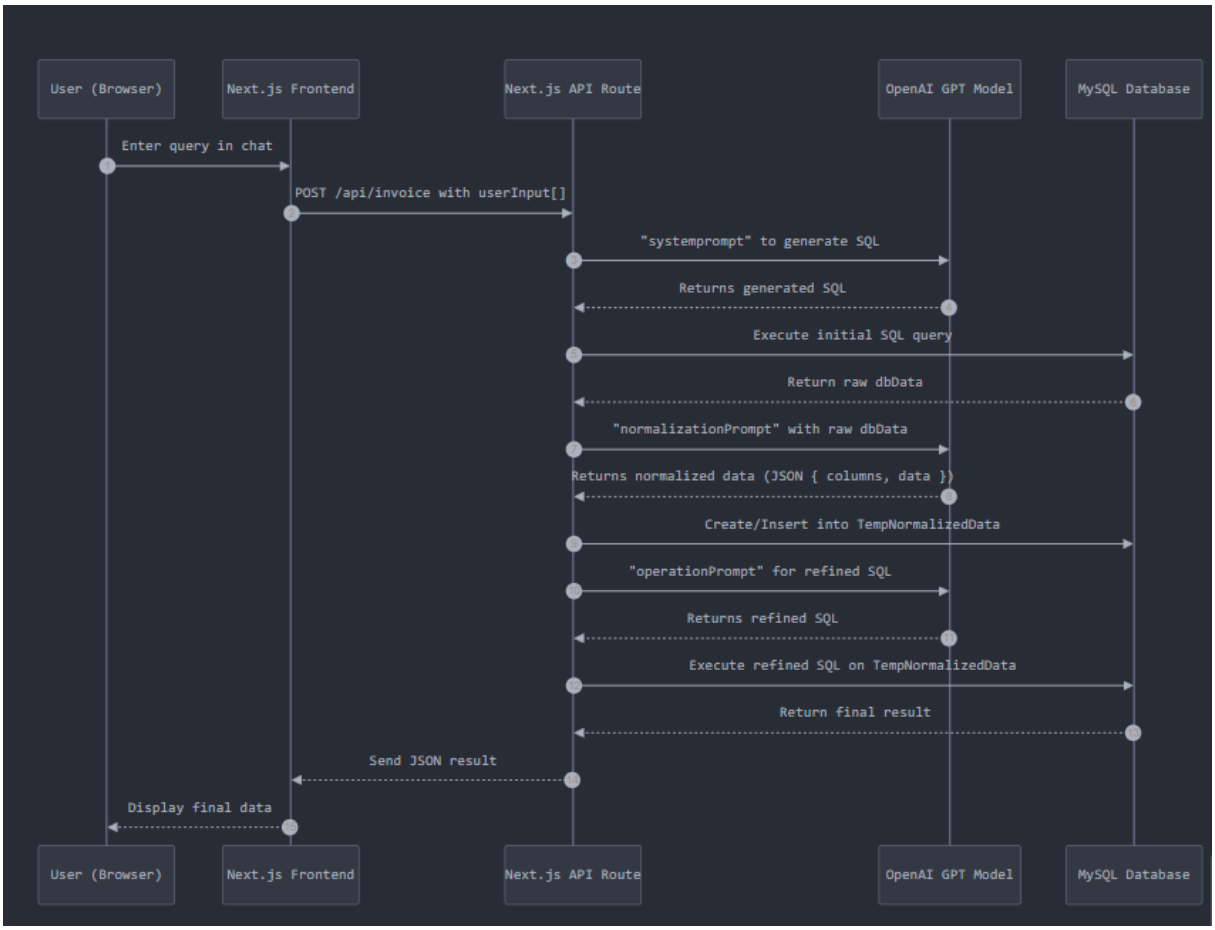
5. **AI-specific considerations:**

1. Chunking Data for LLM.

6. **Alternative solution for Large-Scale Data processing:**

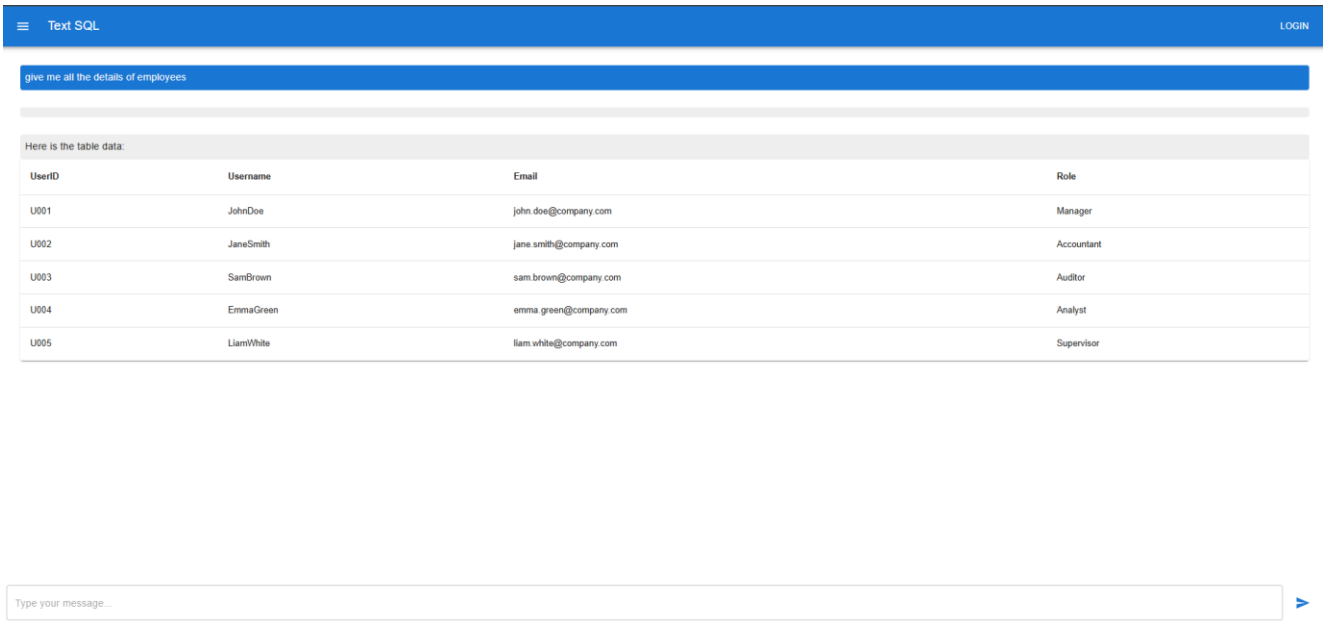
1. Migrate your data to a columnar store (like Snowflake, Big Query, or AWS Redshift).

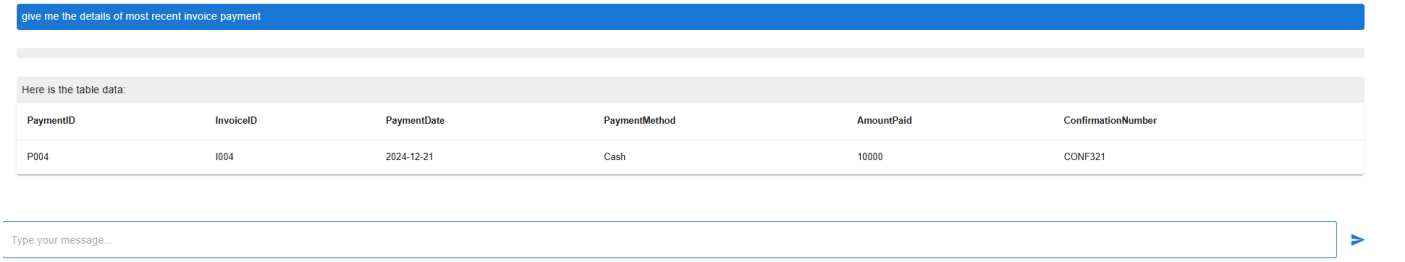
6. **Sequence Diagram:**

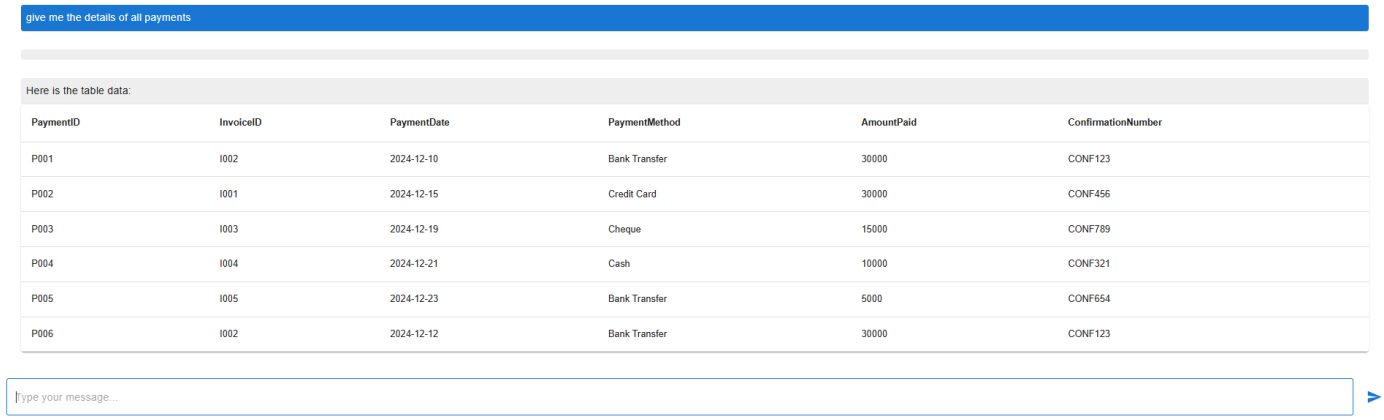


7. **GitHub Repo Link: Github (https://github.com/reysree/sql-chatbot)**

8. **Sample Snippets of the working application:**


Text SQL
LOGIN
give me all the details of employees
Here is the table data:
UserID Username Email Role
U001 JohnDoe john.doe@company.com Manager
U002 JaneSmith jane.smith@company.com Accountant
U003 SamBrown sam.brown@company.com Auditor
U004 EmmaGreen emma.green@company.com Analyst
U005 LiamWhite liam.white@company.com Supervisor
Type your message...


give me the details of most recent invoice payment
Here is the table data:
PaymentID InvoiceID PaymentDate PaymentMethod AmountPaid ConfirmationNumber
P004 I004 2024-12-21 Cash 10000 CONF321
Type your message...


give me the details of all payments
Here is the table data:
PaymentID InvoiceID PaymentDate PaymentMethod AmountPaid ConfirmationNumber
P001 I002 2024-12-10 Bank Transfer 30000 CONF123
P002 I001 2024-12-15 Credit Card 30000 CONF456
P003 I003 2024-12-19 Cheque 15000 CONF789
P004 I004 2024-12-21 Cash 10000 CONF321
P005 I005 2024-12-23 Bank Transfer 5000 CONF654
P006 I002 2024-12-12 Bank Transfer 30000 CONF123
Type your message...

give me the total amount paid till now

Here is the table data:

TotalAmountPaid
120000

Type your message...

what are the invoice whose payment is pending

Here is the table data:

InvoiceID	VendorID	InvoiceDate	DueDate	PaymentDate	TotalAmount	Status	ApprovedBy
I001	V001	2024-12-01	2024-12-15		30000	Pending	U001
I003	V003	2024-12-03	2024-12-18		15000	Pending	U003
I005	V005	2024-12-05	2024-12-22		5000	Pending	U005