International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

QueryAI: A Conversational Interface for SQL DatabaseQuerying Using Natural Language Processing

P. Kartik Naidu[1], Prachi Shrivas[2], Priyal Porwal[3], Priyanka Arya[4], Reeshita Nigam[5]

[1,2,3,4,5]Department of Computer Science and Engineering Acropolis Institute of Technology and Research Indore, India

Abstract
QueryAI is an interface leveraging natural language for SQL database querying. By incorporating large language models (LLMs) and natural language processing (NLP) techniques, QueryAI enables users to query a MySQL database using English-based commands. This interface translates user inputs into SQL statements and presents the outcomes, offering access to those without SQL proficiency. This paper explores QueryAI's design, implementation, and potential applications in data analysis and business intelligence.

Keywords: Language Processing, SQL Database Querying, Large Language Models (LLMs), MySQL Integration, Conversational Database Interface.

1. INTRODUCTION

Accessing data within relational databases generally requires SQL expertise, which can limit access for non-technical users. This reliance on SQL-trained personnel can cause delays in gaining insights, thus slowing down effective decision-making in organizations.
QueryAI tackles this challenge by enabling users to interact with MySQL databases through natural language. Utilizing NLP and large language models (LLMs), QueryAI converts simple English queries into SQL commands, allowing users to retrieve and explore data without needing specialized skills. Through a conversational interface, QueryAI broadens data access, offering a practical and user-friendly way for people in various roles to obtain information directly from databases.

2. BACKGROUND

Relational databases play a crucial role in data storage and management, yet accessing data within them often demands SQL expertise. This skill requirement poses a barrier for non-technical users, restricting their direct access to insights and increasing dependence on technical teams. As organizations become more data-driven, there is a growing need to make data access more inclusive, allowing users from various backgrounds to interact with databases efficiently.
Recent progress in natural language processing (NLP) and large language models (LLMs) presents promi-

sing solutions to overcome this barrier. With NLP, systems can interpret natural language inputs from users and transform them into SQL queries, making database interaction simpler and more conversational. QueryAI capitalizes on these advancements by offering a natural language interface tailored for MySQL databases, enabling non-technical users to access and retrieve data without needing SQL expertise. This innovation fosters a more flexible, data-informed decision-making environment within organizations.

3. PROPOSED METHODOLOGY

QueryAI employs a blend of natural language processing (NLP) and SQL query generation to create a conversational interface for engaging with databases. The system starts by processing user inputs in plain English, where NLP techniques analyze the intent and identify key terms. An LLM model then translates this information into a structured SQL query designed to fit the specific database schema.

The main components of this methodology include:

A. NLP for Intent Recognition: Utilizing libraries such as spaCy, QueryAI detects entities, intent, and context in user input, accurately aligning it with database queries.

B. SQL Query Generation: Based on the identified intent, the system generates SQL code that fulfills the user's request.

C. Real-Time Database Execution: The constructed SQL query is executed on a connected MySQL database, with results immediately retrieved.

D. Contextual Query Support: QueryAI retains a history of interactions, allowing users to pose follow-up questions and build on previous queries.

This methodology enables users to interact with databases in an intuitive, efficient way, without needing SQL expertise. The integration of NLP and LLMs ensures accurate query translation, making database access seamless for a wide range of users.

4. SYSTEM DESIGN AND ARCHITECTURE

A. Natural Language Interface: Users can input queries in simple English, such as "Show all customers from New York." QueryAI interprets these inputs, identifies the user's intent, and translates them into SQL queries to interact with the database.

B. Streamlit Interface: The frontend is developed using Streamlit, offering a user-friendly web-based interface. Streamlit enables real-time interactions, presenting query results in a structured, organized format.

C. SQL Query Generation: Core functionality relies on NLP libraries like spaCy and Transformers to analyze user queries in natural language and generate accurate SQL queries. This translation process is essential to ensure both accuracy and user satisfaction.

D. Conversation Context: QueryAI maintains the flow of conversation by preserving context, allowing users to ask follow-up questions without restating previous information. For example, after requesting "top sales this month," a user can ask for "customer details" without losing the context of the prior query, enhancing fluidity.

E. Simple Setup and Configuration: QueryAI is easy to set up, needing only basic MySQL database connection details (e.g., host, port, username, password). A clear setup guide and intuitive configuration panel help users connect to their database with minimal technical requirements.

F. Real-Time Query Execution: QueryAI executes queries and displays results in real-time, providing users with immediate feedback. This instant response improves efficiency, allowing users to refine queries

iteratively for more detailed insights.

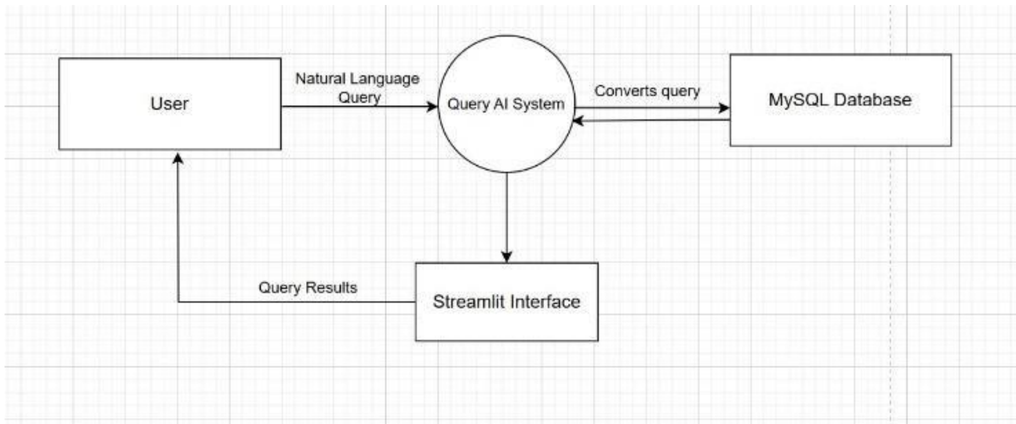

Fig. 1 General Architecture of the System

5. EASE OF USE

A. User-Friendly Interface

QueryAI features an easy-to-use, chat-based interface built with Streamlit, allowing users to engage with databases in a natural way without needing SQL knowledge. The simplicity of the interface makes it accessible for individuals with varying levels of technical expertise.

B. Natural Language Processing

With the integration of NLP, QueryAI allows users to submit queries in plain English. This removes the need to understand complex SQL syntax or memorize database structures, making data querying more accessible and user-friendly.

C. Ongoing Conversations

The system supports conversation history, allowing users to maintain context between multiple queries. Users can ask follow-up questions without repeating previous details, ensuring a smooth and uninterrupted interaction.

D. Easy Setup and Configuration

Setting up QueryAI is simple, requiring just basic information for MySQL database connection (e.g., host, port, user, password). The system provides clear instructions and a straightforward configuration panel, enabling users to connect with minimal technical effort.

E. Instant Query Execution

QueryAI executes queries in real-time and displays results instantly, giving users immediate feedback. This quick response feature boosts productivity and enables users to fine-tune their queries for deeper insights.

6. IMPLEMENTATION

A. Technologies Used

- **Python**: Serves as the primary programming language for implementing backend logic.
- **Streamlit**: Facilitates the creation of a dynamic and responsive web interface.
- **MySQL**: A relational database management system used for data storage and retrieval.
- **spaCy and Transformers**: Natural Language Processing (NLP) libraries utilized for parsing and understanding user queries.

- **SQLAlchemy and pymysql**: Tools for managing database connections and executing SQL queries efficiently.

B. Workflow

- **User Input**: The user submits a query in natural language.
- **NLP Processing**: QueryAI analyzes the query and converts it into an appropriate SQL command.
- **Database Interaction**: The SQL command is executed within MySQL, and the resulting data is retrieved.
- **Display Results**: The results are displayed on the interface, supporting continuous interaction

7. EVALUATION AND RESULTS

To assess the performance of QueryAI, we conducted tests across various MySQL databases with different schema structures and data complexities. The evaluation focused on three key areas: query accuracy, response time, and user satisfaction.

A. Query Accuracy:

The effectiveness of QueryAI in interpreting natural language inputs and generating accurate SQL queries was evaluated using a set of predefined queries from different databases. Initial tests showed that QueryAI achieved about 85% accuracy for simple and direct queries (e.g., “List all customers from New York”). For more intricate, multi-step queries, the accuracy dropped to approximately 70%, particularly when dealing with ambiguous or highly complex queries. This indicates that QueryAI performs well with straightforward queries but may need further NLP improvements for handling more complex or multi-faceted questions.

B. Response Time:

QueryAI's response time was evaluated by measuring the duration between receiving user input and displaying the query results. For simple queries, the average response time was under 2 seconds, while more complex queries took around 3-5 seconds. The response time remained stable across tests, suggesting that the system is suitable for real-time interactions. However, optimization may be needed for larger databases or more complex queries.

C. User Satisfaction:

A diverse group of users, including both technical and non-technical participants, tested QueryAI for usability. The feedback was generally positive, with users appreciating the intuitive interface and conversational querying approach. Non-technical users found it particularly helpful for retrieving data without SQL knowledge, although some expressed frustration when handling more complicated requests. Overall, users indicated that QueryAI meets its goal of accessibility and ease of use, but there is room for improvement in handling complex queries more accurately.

In conclusion, QueryAI demonstrates strong performance in handling basic database queries with good accuracy and fast response times. While it excels with simple queries, additional refinement is needed to enhance its ability to manage complex or ambiguous queries.

8. CONCLUSION

QueryAI marks a significant advancement in simplifying database querying for non-technical users. By utilizing natural language processing (NLP) and large language models (LLMs), QueryAI enables users to interact with MySQL databases using plain English commands instead of writing complex SQL queries. This conversational method bridges the gap between technical and non-technical users, allowing a wider

range of professionals to independently access, analyze, and derive insights from data.

The project showcases the potential of NLP-powered tools to improve data accessibility and boost productivity in data-driven environments. With features like an intuitive interface, real-time query execution, and conversational context support, QueryAI proves to be a valuable asset in business intelligence and decision-making. Early evaluations indicate that it is effective in converting natural language into SQL, delivering accurate and timely data retrieval that meets the needs of diverse user groups.

Despite its success, there are areas for further enhancement. Future improvements could focus on expanding QueryAI's compatibility with other database systems beyond MySQL and refining its NLP capabilities to better handle more intricate, multi-step queries. Additionally, adding advanced error handling and user feedback features could improve QueryAI's reliability and overall user experience.

In conclusion, QueryAI makes database querying more accessible and user-friendly by removing technical barriers. As NLP and LLM technologies continue to advance, tools like QueryAI will likely become essential for organizations aiming to provide all employees with direct access to data, fostering a more agile and data-driven decision-making culture.

REFERENCES

1. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space.
2. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.
3. Sun, Y., Zhang, J., & Han, J. (2018). "Natural Language to SQL: Where Are We Now and Where Are We Going?" Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 3774–3783.
4. Li, F., & Jagadish, H. V. (2014). "Constructing an Interactive Natural Language Interface for Relational Databases." Proceedings of the VLDB Endowment, 8(1), pp. 73–84.
5. Finegan-Dollak, C., et al. (2018). "Improving Text-to-SQL Evaluation Methodology." Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 351–360.
6. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), pp. 4171–4186.

10. SOURCE CODE

```
from dotenv import load_dotenv
fromlangchain_core.messages import AIMessage,HumanMessage
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnablePassthrough
import streamlit as st
from langchain_core.output_parsers import StrOutputParser
from langchain_openai import ChatOpenAI
from langchain_community.utilities import SQLDatabase
from langchain_groq import ChatGroq
```

```
def init_database(user:str, password:str, host:str, port:str, database:str)->SQLDatabase:
db_uri=f"mysql+mysqlconnector://{user}:{password}@{host}:{port}/{database}"
    return SQLDatabase.from_uri(db_uri)

def get_sql_chain(db):
    template = """
    You are a data analyst at a company. You are interacting with a user who is asking you questions about the company's database.
    Based on the table schema below, write a SQL query that would answer the user's question. Take the conversation history into  account.

    <SCHEMA>{schema}</SCHEMA>

    Conversation History: {chat_history}

    Write only the SQL query and nothing else. Do not wrap the SQL query in any other text, not even backticks.

    For example:
     Question: which 3 artists have the most tracks?
     SQL Query: SELECT ArtistId, COUNT(*) as track_count FROM Track
     GROUP BY ArtistId ORDER BY track_count DESC LIMIT 3;
     Question: Name 10 artists
     SQL Query: SELECT Name FROM Artist LIMIT 10;

     Your turn:

     Question: {question}
     SQL Query:
     """
     prompt = ChatPromptTemplate.from_template(template)

    llm = ChatGroq(model="mixtral-8x7b-32768", temperature=0)

    def get_schema(_):
    return db.get_table_info()

    return (
     RunnablePassthrough.assign(schema=get_schema)
            | prompt
            | llm
            | StrOutputParser()
    )
```

```
def get_response(user_query:str, db:SQLDatabase, chat_history: list):
sql_chain = get_sql_chain(db)

template = """
You are a data analyst at a company. You are interacting with a user who is asking you questions about the company's database.
Based on the table schema below, question, sql query, and sql response, write a natural language response.
    <SCHEMA>{schema}</SCHEMA>

    Conversation History: {chat_history}
    SQL Query: <SQL>{query}</SQL>
    User question: {question}
    SQL Response: {response}"""

    prompt = ChatPromptTemplate.from_template(template)


    llm = ChatGroq(model="mixtral-8x7b-32768", temperature=0)

    chain = (
        RunnablePassthrough.assign(query=sql_chain).assign(
        schema=lambda _: db.get_table_info(),
        response=lambda vars: db.run(vars["query"]),
        )
        | prompt
        | llm
        | StrOutputParser()
    )

    return chain.invoke({
    "question": user_query,
    "chat_history": chat_history,
    })

    if "chat_history" not in st.session_state:
    st.session_state.chat_history = [
  AIMessage(content="Hello! I'm a SQL assistant. Ask me anything about
your database."),
    ]

load_dotenv()

st.set_page_config(page_title="Chat with MySQL", page_icon=":speech_balloon:")
```

```
st.title("Chat with MySQL")
with st.sidebar:
st.subheader("Settings")
st.write("This is a simple chat application using MySQL. Connect to the database and start chatting")

st.text_input("Host", value="localhost", key="Host")
        st.text_input("Port", value="3306", key="Port")
        st.text_input("User", value="root", key="User")
        st.text_input("Password", type="password", value="root1234",     key="Password")
        st.text_input("Database", value="college", key="Database")

        if st.button("Connect"):
        with st.spinner("Connecting to Database..."):
        db = init_database(
        st.session_state["User"],
        st.session_state["Password"],
        st.session_state["Host"],
        st.session_state["Port"],
        st.session_state["Database"]
        )
        st.session_state.db = db
        st.success("Connected to database!")

        for message in st.session_state.chat_history:
        if isinstance(message, AIMessage):
        with st.chat_message("AI"):
        st.markdown(message.content)
        elif isinstance(message, HumanMessage):
        with st.chat_message("Human"):
        st.markdown(message.content)
        user_query = st.chat_input("Type a message...")
        if user_query is not None and user_query.strip() != "":
st.session_state.chat_history.append(HumanMessage(content=user_query))

        with st.chat_message("Human"):
        st.markdown(user_query)

        with st.chat_message("AI"):
response=get_response(user_query,st.session_state.db,st.session_state.chat_history)
        st.markdown(response)

st.session_state.chat_history.append(AIMessage(content=response)
```