ChatQPT: Towards Conversing with Relational Query Engines

Hui Li
Xidian University
China
hli@xidian.edu.cn

Sourav S Bhowmick
Nanyang Technological University
Singapore
assourav@ntu.edu.sg

Baochao Xu
Xidian University
China
bcxu_x@stu.xidian.edu.cn

Zhenning Shi
Xidian University
China
znshi@stu.xidian.edu.cn

Xiyue Gao
Xidian University
China
xygao@xidian.edu.cn

Jingjing Li
The Chinese University of Hong Kong
Hong Kong SAR, China
llee.jingjing@gmail.com

ABSTRACT

Imagine a scenario where we could communicate with a relational query processor through messaging platforms like *WeChat* or *WhatsApp*, asking questions related to query processing and optimization. If realized, such a tool could prove invaluable in database education and administration, among others. In this demonstration, we present ChatQPT, a novel system that enables chat-based interactions with a relational query engine (PostgreSQL). It *combines* an LLM-powered interface with a set of specialized external tools tailored to understand various aspects of relational query processing, effectively addressing the shortcomings of both large language models and off-the-shelf RDBMSs in supporting accurate and effective conversations. Our preliminary evaluation through a user study demonstrates the promising capabilities of ChatQPT.

PVLDB Reference Format:
Hui Li, Sourav S Bhowmick, Baochao Xu, Zhenning Shi, Xiyue Gao, and Jingjing Li. ChatQPT: Towards Conversing with Relational Query Engines. PVLDB, 19(12): 4758 - 4761, 2026.
doi:10.14778/3827998.3828115

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://www.dbedu.tech/chatqpt.

1 INTRODUCTION

The increasing adoption of RDBMS and the growing prominence of data science has intensified the demand for professionals with a strong understanding of relational query processing. Yet, even after decades, the internal workings of query processors remain largely opaque to most users. Even though many commercial data platforms provide conversational interfaces for data access and analytics [5, 6, 9, 10], these interfaces primarily help users to formulate queries and inspect results, rather than understand how relational query processors interpret, optimize, and execute SQL. Commercial RDBMSs typically reveal only limited details, *i.e.,* mainly query execution plans (QEPs) [3] via EXPLAIN statement, offering little insight into optimization decisions or the plan space.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828115

Recent research has increasingly focused on developing user-friendly tools to enhance understanding of relational query processing [1]. For example, LANTERN [2] generates natural-language explanations of QEPs; MOCHA [11] offers learner-friendly interaction and visualization of how different physical operator choices affect a selected QEP for a given SQL query; ARENA [13] enables users to visually explore and compare QEPs with alternative query plans; and RETRO [14] aids learners understand cardinality estimation errors associated with a QEP. These systems demonstrate that user-friendly support for different aspects of relational query processing is both feasible and valuable. However, these systems largely exist as standalone platforms with specialized interfaces. Consequently, users must navigate multiple disparate environments, which can be cumbersome. They must switch between multiple tools, learn their distinct interfaces, and often restate the same query across platforms. To date, no existing system offers a *unified, coherent, and user-friendly* interface that supports interaction with the diverse aspects of relational query processing.

Given the popularity and accessibility of chat-based applications, a conversational framework for relational query processing is promising. Specifically, we investigate whether natural language (NL) dialogue can enable users to interactively explore and understand query execution[1]. Such a framework would provide a unified interface for examining QEPs, physical operators, search spaces, alternative plans, cardinality estimation errors, and cost models, thereby supporting both database administration and education.

At first glance, it may seem that general-purpose chatbots (*e.g.,* ChatGPT) built on large language models (LLMs), such as GPT-4o [8], can address this problem. However, as we shall see in Section 2, this is not the case due to the unique challenges inherent in our problem setting.

In this demonstration, we present ChatQPT (Chat with a Query Processor for relaTions), a novel *extensible* system for *multi-turn* conversations with relational query processors. To overcome the limitations of LLMs and off-the-shelf RDBMSs in conversational support, it uses a textbook-based knowledge base and *integrates* a large language model with a suite of task-specific tools [2, 11, 13, 14]. A *dynamic tool-invocation* mechanism coordinates these components, allowing users to interact with the relational query processor in natural language and obtain high-quality responses. A preliminary user study demonstrates the promise of ChatQPT.

[1]We assume queries are written in SQL, not natural language. This setup is particularly relevant to database administration and educational settings where SQL is the standard input.

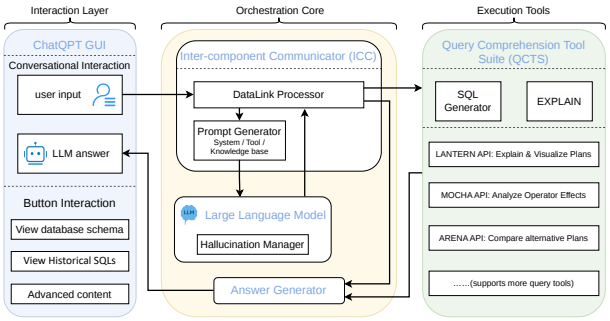

Figure 1: Architecture of ChatQPT.

2 DESIGN PHILOSOPHY

Realizing ChatQPT for conversing with a relational query engine must address the following non-trivial challenges. Firstly, general-purpose LLMs are versatile but not specialized for specific tasks:

> *"General-purpose technology that is good at a lot of different things, not narrowly great and purpose-built for one specific thing."* [7]

Interacting with a relational query engine, however, requires detailed knowledge of its internal query processes—something LLMs lack. Moreover, there is no large–scale public training data focused on question answering for relational query processing.

Secondly, LLMs are prone to hallucinations, generating factually inaccurate responses. This limitation is particularly problematic for conversational systems that must reason about query–specific, internal RDBMS details that are rarely represented in their training data. This makes maintaining accuracy and context a major challenge:

> *"Hallucination should be more likely to occur when dealing with subjects that are obscure or controversial."* [12]

Thirdly, multi-turn conversations are crucial for conversational systems but can mislead LLMs to produce inaccurate responses:

> *"Despite the prompts being completely unrelated, the applications would sometimes use the context of preceding questions to inform subsequent responses."* [12]

For example, during extended interactions, the system may mistakenly return results for a different SQL query–responding about the j-th query instead of the intended i-th one where $i \neq j$.

Lastly, standard RDBMSs provide limited interaction with the relational query processor—beyond the EXPLAIN feature for viewing QEPs—and lack user-friendly interfaces. Therefore, a conversational engine cannot rely solely on their built-in capabilities.

To address the above challenges, the design of ChatQPT is guided by the following four principles: (a) avoid using LLMs for internal, plan-specific query details; (b) exploit LLM strengths in understanding intent, explaining concepts, and generating natural language responses; (c) include a *Hallucination Manager* to manage inaccurate or misleading outputs; and (d) integrate specialized external tools for interpreting query processor features instead of relying solely on standard RDBMS capabilities.

3 SYSTEM OVERVIEW

The architecture of ChatQPT is depicted in Figure 1. The *GUI* facilitates conventional dialog interactions while also providing features for browsing database schemas, reviewing historical SQL queries, and visual exploration of query plans. The *LLM* module is responsible for interpreting user intent, extracting query parameters, and determining whether a back-end tool should be invoked to assist with the response. A critical submodule within it is the *Hallucination Manager*, designed to manage the negative effects of potential hallucinations that might arise in ChatQPT. Complementing this is a modular tool suite, *Query Comprehension Tool Suite (QCTS)*, that supports interactions with various external platforms related to relational query processing. The current version integrates the following tools: LANTERN [2], MOCHA [11], ARENA [13], and RETRO [14], which respectively support QEP explanation, operator–level analysis, alternative plan exploration, and cardinality estimation error diagnosis. Besides, QCTS can easily incorporate other tools through a common MCP-compatible wrapping. The *Answer Generator* module is responsible for presenting the responses of ChatQPT to the end users. Finally, the *Inter-component Communicator (ICC)* module manages the communication between these modules, and serves as a unified coordination layer between the conversational front end and QCTS, so that MCP-compatible tools are invoked and their outputs are processed under a coherent interaction workflow.

ChatQPT GUI. The GUI in Figure 2 is a conversational hub integrating query input, result display, and visual feedback. Users can inspect the database schema via a button (*C1*) to help construct SQL statements. Queries are entered in the bottom input box (*C9*), with submitted input shown on the right (*C3*) and system responses on the left (*C4–C8*). ChatQPT provides two response types: *direct*, generated solely by the LLM, and *derived*, produced via QCTS tools. For derived responses, the system identifies the target platform, generates a clickable link (*C5*), and renders outputs in textual (*C4*), tabular (*C7*), or graphical (*C8*) form. Advanced content, such as execution plans, is initially hidden and can be revealed via buttons. A *historical SQL panel* (*C2*) lets users browse and reuse past SQL queries, mitigating hallucinations in multi-turn conversations by enabling re-execution of previous queries with the same tool.

The GUI uses an *adaptive rendering strategy* powered by the *Answer Generator* to display outputs from various tools in a user–friendly way. It employs a unified rendering mechanism that adapts to different data formats (*e.g.*, text, tree, table) to improve usability and support multi–turn conversations.

Large Language Model (LLM) Module. The *Large Language Model (LLM)* module is the core of ChatQPT. Given the challenges of LLMs (Section 2), we use it judiciously. It interprets user intent, extracts query parameters, and decides whether to invoke specialized backend tools or answer directly. Upon a user request, it first retrieves relevant content from a textbook-based repository to build a retrieval-augmented prompt. This prompt and the conversation history are sent to a decision LLM, which determines whether to use a QCTS tool for higher accuracy. If a tool is used, another LLM rewrites its structured output into natural language. For hallucination-sensitive tasks such as SQL generation, terminology explanation, and platform interfacing, the *Hallucination*

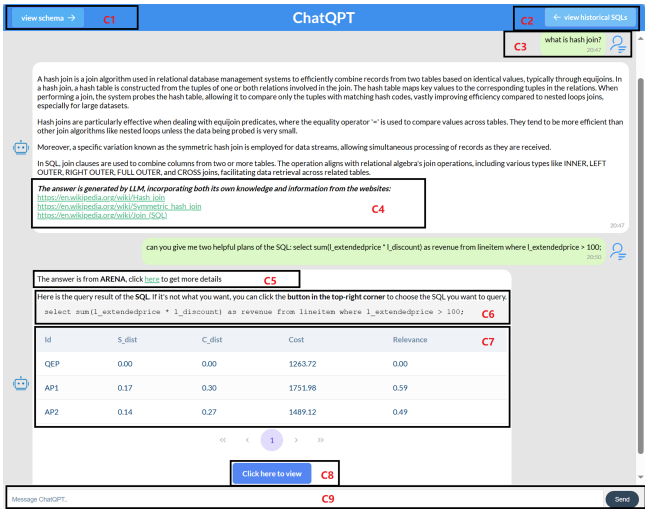

Figure 2: ChatQPT GUI.

Manager applies safeguards to reduce factual errors. Finally, the *Answer Generator* returns a unified response to the user.

Hallucination Manager. ChatQPT organizes its functionalities into three areas corresponding to three common hallucination-sensitive scenarios: (1) *SQL query generation* – Queries are pre-validated using the EXPLAIN command before display, with a cache of verified queries maintained to reduce latency. (2) *Terminology explanation* – Since direct validation is difficult, ChatQPT uses a retrieval-augmented generation (RAG) process over a local knowledge base of RDBMS textbooks and papers. Retrieved references are linked so that users can explore the source material (*C4*). (3) *Platform interfacing* – To handle potential errors in QCTS tool connections or incorrect SQL generation (Section 2), ChatQPT displays the executed SQL for user verification and allows selection from past validated queries with a simple button click (*C2*).

The *Hallucination Manager* was evaluated through experimental and user studies showing clear improvements in accuracy and user trust: (a) In 100 randomly generated SQL queries, 14 hallucination errors were fully corrected by the system, and caching reduced response time by 50 ms~2s. (b) In an ablation study with 132 users, participants were asked to explain database terminology they learned through ChatQPT. 8% of responses from users who interacted with ChatQPT without the *Hallucination Manager* contained errors, whereas those using it were completely accurate. (c) Although no hallucinations occurred in platform interfacing, 65% of users reported higher satisfaction, and all agreed it increased the trustworthiness of ChatQPT's answers.

Query Comprehension Tool Suite (QCTS). ChatQPT integrates an extensible *Query Comprehension Tool Suite* that mediates between user queries and task-specific tools for relational query processing. It integrates: (a) LANTERN for natural language explanations of query execution plans [2], (b) MOCHA for physical operator–based plan exploration [11], (c) ARENA for exploration and comparative visualization of alternative plans [13] and (d) RETRO for diagnosing cardinality-estimation errors through operator-level discrepancies and relevant distributional evidence [14]. As remarked in Section 2, these functionalities are currently beyond the reach of LLMs as well as off-the-shelf RDBMSs.

In the current implementation, tool invocation is primarily handled through *OpenAI* function calling. Newly added tools can be incorporated through a common MCP-compatible wrapping and corresponding processor adaptation, without changing the overall conversational workflow.

Inter-component Communicator (ICC). The ICC module primarily consists of the following two submodules.

Prompt Generator. This module constructs the input to the LLM in two parts: a unified system prompt and a compressed runtime context. The system prompt provides persistent guidance, including descriptions of backend platforms in QCTS and a small number of examples for consistent parameter extraction. The runtime context summarizes the current interaction by combining the user's question, retrieved knowledge snippets, and a compact record of recent tool-related context such as key SQL statements and short result summaries.

DataLink Processor. This module manages the runtime interaction between ChatQPT and the tools in QCTS through a *central processor* and multiple *subsidiary processors*, and serves as a unified coordination layer between the conversational front end and QCTS.

Upon receiving a structured tool call from the decision LLM, the central processor preprocesses its parameters, *e.g.,* validating SQL queries and offering feedback if errors exist, before routing the request to the appropriate *subsidiary processor.* Each subsidiary processor is dedicated to a specific platform, translating between ChatQPT's conversational interface and the platform's unique features. For example, the MOCHA processor resolves challenges such as unsupported operators, typos, and interpretation of operator inclusion/exclusion—issues that arise when transitioning from GUI-based to conversational interaction. After processing, the subsidiary processor returns data to the central processor for unified packaging. This modular design ensures extensibility, allowing ChatQPT to easily integrate new tools by adding corresponding subsidiary processors.

Answer Generator Module. This module enhances data presentation to make information from the *LLM* and *DataLink Processor* more accessible and user-friendly. It processes outputs in three formats: (a) *Text:* Adds line breaks and emphasizes key terms using italics and bold for readability (*C4-C6*). (b) *Tree Diagrams:* Visualizes query plans in standardized tree views for consistent interpretation. (c) *Tables:* Limits displayed entries so users can compare results across dialogue rounds and includes pagination for easier navigation (*C7*).

4 PRELIMINARY USER STUDY

To understand the usefulness of ChatQPT in practice, we undertake a small-scale user study. In this section, we briefly describe it. Note that RETRO [14] was excluded from the user study because of its recent release.

Baselines. No public chatbot exists for relational query engines, so we compare three ChatQPT variants: (a) System A: a baseline chatbot using only an LLM (*i.e.,* the yellow box in Figure 1, implemented with *Qwen-Max-Latest*) and no relational query understanding (no QCTS module). (b) System B: System A extended with basic

query-processing knowledge via an *SQL Generator* that retrieves random cached queries and uses the engine's EXPLAIN feature for QEP details. (c) System C: ChatQPT.

Participants. The study involved 25 unpaid computer science students (ages 21–30) with prior knowledge of relational query processing; some graduate students worked on query optimization. All had chatbot experience, but none had used one for relational query engines. After consent and a brief tutorial with examples, participants held free-form conversations with all three system variants (A, B, C) via a unified interface that concealed system differences. They could ask questions during the study, and any interaction-related inquiries were answered.

Tasks and procedure. Participants were not given predefined tasks but were encouraged to explore the systems freely, simulating real-world use. Typical interactions included asking about query plan explanations, alternative plans, operator effects on cost, and query processing concepts. The study lasted two weeks, during which participants used all systems organically.

Survey Results. After using each system, participants completed a survey evaluating specific functional and experiential aspects of its performance (Figure 3). The questionnaire also collected overall user impressions, focusing on output clarity and dialog responsiveness. Participants rated each system on a 1–10 Likert scale, where 0 represented a very poor experience or satisfaction, and 10 indicated an excellent one.

System C outperformed Systems A and B across most evaluation areas, showing especially strong results in alternative plan exploration (*Q2*), interactivity (*Q6*), and answer formatting (*Q7*). These findings suggest that ChatQPT provides a more effective and satisfying conversational experience.

5 RELATED SYSTEMS

DataChat [6] allows users to build data analysis pipelines through chat or point-and-click interactions using abstracted data functions. *Cortex Analyst* [10] translates plain-English questions into SQL using generative AI, while *Databricks Assistant* [4] acts as an AI pair-programmer for writing, debugging, and optimizing code or workflows. Similarly, *BigQuery* and *Oracle Digital Assistant* [5, 9] use NL-based conversations supported by LLMs to facilitate data preparation, cleaning, and model training. Together, these tools highlight the practical value of NL-driven analytics. However, they primarily address data engineering tasks outside traditional RDBMS, where general-purpose LLMs suffice. None directly targets query processing within RDBMS.

6 DEMONSTRATION PLAN

ChatQPT is implemented using *Python* and *PostgreSQL*. Our demonstration shall use the *TPC-H benchmark* and *IMDb* datasets. The primary goal is to enable the audience to experience conversation with a relational query processor. A **short video** to illustrate the main features of ChatQPT using an example use case in database education is available at https://youtu.be/fh0e8dP3SNQ, and ChatQPT is currently available for free in educational settings through https://dbedu.xidian.edu.cn/chatqpt/.

Unified chat-based conversation with the relational query processor. The audience can explore the database schema used

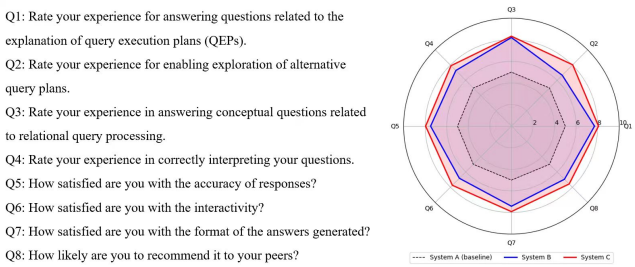

Figure 3: Comparative radar plot of user ratings (Q1–Q8).

by ChatQPT via a button in *C1* (Fig. 2) and either submit their own SQL queries or have ChatQPT generate random ones. Generated SQL statements are verified by the *Hallucination Manager* before execution. Users may subsequently submit follow-up inquiries regarding query processing, with responses generated by the respective platforms, while the LLM manages any questions that fall outside predefined mappings by default. The resulting outputs and responses are presented through the chat interface, enabling users to review explanations and further explore relevant details. Additionally, users can enable or disable the *Hallucination Manager* to observe its effects on predefined examples.

Ad-hoc questioning of relational query processing-related concepts. Users can ask ChatQPT about general, platform-independent concepts in relational query processing. For example, after receiving a natural language explanation of a QEP, they can request clarifications on terms like 'Nested Loop' and 'Index Scan'. The *Hallucination Manager* retrieves accurate information from a trusted local knowledge base and shares it with both the LLM and the user. Hyperlinks allow users to access the original source files, helping them explore and better understand these concepts.

REFERENCES

[1] Sourav S Bhowmick and Hui Li. 2022. Towards Technology-Enabled Learning of Relational Query Processing. *IEEE Data Eng. Bull.* 45, 3 (2022), 29–46.
[2] Peng Chen et al. 2022. LANTERN: Boredom-conscious natural language description generation of query execution plans for database education. In *SIGMOD*. 2413–2416.
[3] Dalibo. 2026. PEV2: PostgreSQL execution plan visualizer. https://explain.dalibo.com/about
[4] Databricks. 2023. Introducing Databricks Assistant, a context-aware AI assistant. https://www.databricks.com/blog/introducing-databricks-assistant
[5] Google. 2025. Gemini in BigQuery overview. https://docs.cloud.google.com/bigquery/docs/gemini-overview
[6] Rogers Jeffrey Leo John et al. 2023. DataChat: An Intuitive and Collaborative Data Analytics Platform. In *SIGMOD*. 203–215.
[7] Logan Kugler. 2025. How Do You Measure AI? *Commun. ACM* 68, 4 (2025), 15–17.
[8] OpenAI. 2024. GPT-4o. https://openai.com/index/gpt-4o-system-card/
[9] Oracle. 2025. Digital Assistant. https://www.oracle.com/chatbots/
[10] Snowflake. 2025. Cortex Analyst. https://docs.snowflake.com/en/user-guide/snowflake-cortex/cortex-analyst
[11] Jess Tan, Desmond Yeo, Rachael Neoh, et al. 2022. MOCHA: A tool for visualizing impact of operator choices in query execution plans for database education. *Proc. VLDB* 15, 12 (2022), 3602–3605.
[12] Jim Waldo and Soline Boussard. 2025. GPTs and Hallucination. *Commun. ACM* 68, 1 (2025), 40–45.
[13] Hu Wang et al. 2023. ARENA: Alternative Relational Query Plan Exploration for Database Education. In *SIGMOD*. 107–110.
[14] Yani Zhang et al. 2026. RETRO: A Visual Tool for Understanding Cardinality Estimation Errors for Database Education. In *SIGMOD*.