Agentic AI System for Data-Driven Question Answering Using SQL Generation with Large Language Models

Atacan Durmusoglu
Principal Product Analyst
OLX Group
Berlin, Germany
atacan.durmusoglu@olx.com

Ming Zhang
Principal Data Scientist
OLX Group
Amsterdam, Netherlands
ming.zhang@olx.com

Jessy Neves
Senior Data Engineer
OLX Group
Lisbon, Portugal
jessy.neves@olx.com

Miguel Chin
Data Engineering Manager
OLX Group
Lisbon, Portugal
miguel.chin@olx.com

Andreas Merentitis
Chief Data Officer
OLX Group
Berlin, Germany
andreas.merentitis@olx.com

***Abstract*—In the realm of data analytics, translating natural language queries into executable SQL statements remains a complex challenge, particularly within domains rich in context-specific terminology and intricate data schemata. This paper introduces an agentic system designed to interpret user questions, extract analytical entities, generate and execute SQL queries to retrieve the desired data insights. Developed in the context of OLX, a global online classifieds marketplace with diverse platforms and categories, the system addresses ambiguities inherent in natural language and domain-specific jargon. By decomposing user queries into refined analytical entities, leveraging deterministic methods alongside Large Language Models (LLMs), and employing a modular agent-based architecture, the system achieves efficient and reliable translation of user intent into data retrieval operations. In testing, the system dynamically queries 20 tables to answer questions about 94 metrics and achieved a 94% satisfaction rate from users. We discuss the architecture, components, implementation details, and lessons learned offering insights for future research in data-driven conversational agents.**

***Keywords*—Agentic systems, SQL generation, large language models, data analytics, conversational agents.**

I. INTRODUCTION

The surge in data-driven decision-making has intensified the need for tools that bridge the gap between natural language and data retrieval languages like SQL. Users often possess domain expertise but lack proficiency in formal query languages, creating a demand for systems that can accurately interpret and execute data queries expressed in everyday language.

OLX, a leading global online classifieds marketplace, operates multiple platforms across various countries, encompassing a multitude of categories such as goods, services, real estate, and motors. The complexity of OLX's data ecosystem, compounded by nuanced terminologies and overlapping concepts, poses significant challenges in automating the translation of user questions into SQL queries.

This paper presents an agentic system designed to address these challenges by decomposing user queries into analytical entities, resolving ambiguities through user interaction, and leveraging both deterministic methods and LLMs for query generation. Our contributions include:

- A modular architecture that separates concerns and allows for extensibility.
- Techniques for handling domain-specific ambiguities and terminologies.
- Integration of deterministic parsing and LLMs to optimize performance and reliability.
- Methods for dynamic retrieval and augmentation of knowledge bases within the query generation process.

II. RELATED WORK

Translating natural language into SQL has been a focus of research for years [1][2], with approaches ranging from rule-based systems [3] to more recent work that leverages transformer architectures [4].

Agent-based AI has gained traction, offering modularity and the ability to handle complex tasks

through the interaction of specialized agents [5][6]. The integration of LLM-based agents for SQL query generation has been explored by a variety of approaches [7].

Our work differentiates itself by combining agent-based architecture with structured outputs, retrieval methods with LLMs, and deterministic methods [8], tailored for the complexities of OLX's data environment.

III. System Architecture

The system is designed as a collection of interacting agents, each responsible for specific tasks in the process of translating user questions into SQL queries. The main components are:

- User Interaction Agent: Manages the dialogue with the user, handles clarification, and orchestrates the overall process.
- Analytical Entities Extraction Agent: Parses user queries to extract metrics, dimensions, filters, and other relevant entities.
- Similar Questions Retrieval Agent: Searches a knowledge base of previous queries and SQL statements to find relevant examples.
- SQL Generation Agent: Generates SQL queries based on refined user intent, examples, and documentation.
- Execution and Error Handling Module: Executes the SQL queries and manages errors during execution.

These agents are interconnected through a workflow that relies on tool-invoking mechanisms. Each agent has the discretion to call the appropriate tools as needed, allowing for dynamic interaction based on the context of the task.

For instance, after the User Interaction Agent gathers initial input, it may invoke the Analytical Entities Extraction Agent to parse the query. Subsequently, the Similar Questions Retrieval Agent may be called to fetch relevant examples that inform the SQL Generation Agent when the User Interaction Agent decides we have enough information to proceed with the data analysis. SQL Generation Agent fixes the query by looking at the error returned from the database. This modular approach enables agents to iterate on their tasks, refining their outputs until the SQL query is successfully generated and executed.

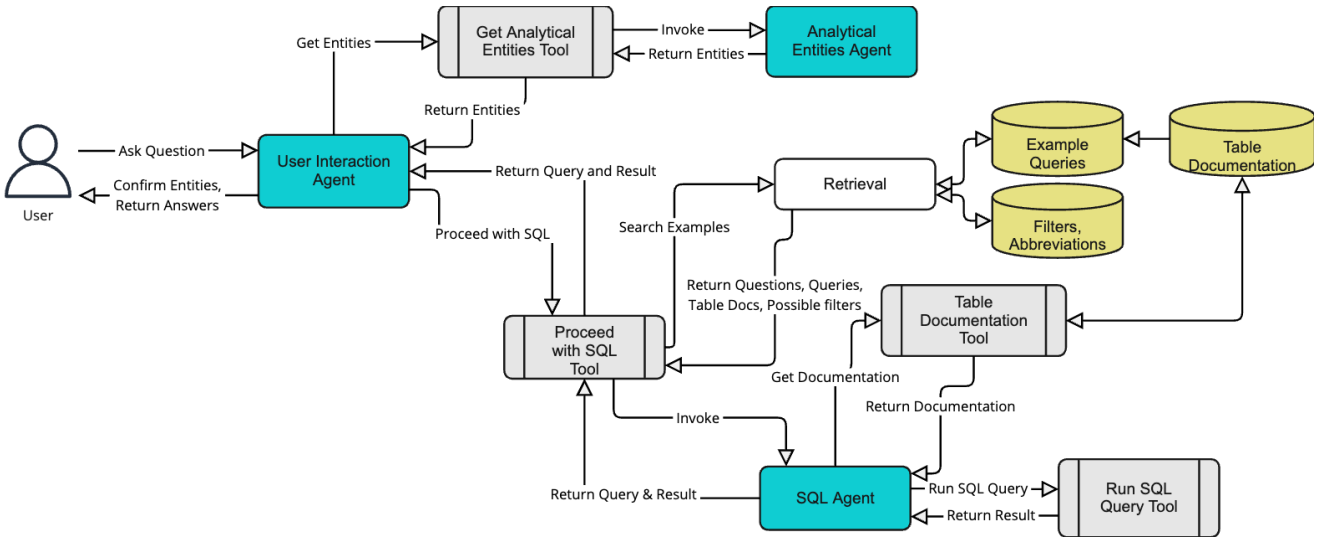

Fig. 1. System Architecture

A. User Interaction Agent

The User Interaction Agent serves as the coordinator, guiding the user through the process: 1) Receives the user's natural language query. 2) Calls the Analytical Entities Extraction Agent to parse the query. 3) Presents the extracted entities to the user for confirmation. 4) Handles user feedback, possibly iterating multiple times to refine the entities. 5) Initiates the SQL generation process upon confirmation.

The agent maintains context across multiple turns, allowing users to modify or append information incrementally.

B. Analytical Entities Extraction

User queries often contain implicit assumptions and domain-specific language that can lead to misinterpretation. To address this, we decompose queries into analytical entities.

- Metrics: The quantitative measures or KPIs to be calculated.
- Dimensions: Attributes to break down the metrics (e.g., by category or time).
- Filters: Constraints applied to the data (e.g., specific countries or websites).
- Additional Information: Any extra details or clarifications needed.

For example, when a user asks “How many door-to-door orders did we have in Poland, in the Car Parts category for B2C sellers, in 2025 by calendar month, and how much revenue did we make from these orders?” We analyze it as

- Metrics: door-to-door orders, revenue
- Sites: olx.pl
- Dimensions to break down by: month
- Time period filter: between 2025-01-01 and 2025-12-31
- Offer category filter: Car Parts
- Other filters: seller type is B2C

C. Retrieval of Similar Questions and Metrics

The decomposition of analytical entities enables scoped knowledge retrieval, allowing each component to be processed and refined separately.

To augment the SQL generation process, the system harnesses a knowledge base that consists of user questions with the specific metrics involved and the corresponding SQL query that extracts the required data. The primary retrieval strategy leverages the metric names—extracted from the user question in the previous step—to perform focused searches within this knowledge base.

The system employs several search methodologies: a) Embedding Search: Utilizes vector embeddings to associate metric names with historically analogous queries. b) Keyword Search: Implements direct textual matching to capture exact term occurrences. c) BM25 Algorithm: Applies a probabilistic model based on term frequency to rank the relevance of documents. These methods may occasionally overlook domain-specific nuances. For example, while the terms "order" and "transaction" might appear semantically similar, they denote distinct concepts within the OLX context. To mitigate such ambiguities, an auxiliary agent further refines the candidate results by incorporating company-specific terminologies.

Complementary to metric-based retrieval, the system maintains a dedicated repository for previously extracted dimensional filters, such as site identifiers and offer categories. This approach enables efficient query augmentation by selectively incorporating only the relevant filters when specific categories or site constraints are detected in the user query. By maintaining these filters separately, the system achieves greater modularity and substantially reduces prompt complexity. This selective augmentation strategy is particularly beneficial given the potentially unlimited variations of filter combinations across the marketplace ecosystem.

D. SQL Generation Agent

With refined analytical entities and relevant example queries, the SQL Generation Agent constructs the SQL query: It uses retrieved queries as templates, adapting them to the current context. It works with detailed information about database tables and columns that is automatically extracted from the retrieved example queries. It adjusts the query based on specific dimensions and filters provided by the user.

When the agent identifies the need to include additional tables not present in the example queries (particularly for implementing user-requested

dimensional breakdowns or filters), it has access to a tool that can fetch documentation for these additional tables on demand. This allows the agent to explore join paths and related tables as required.

E. Execution and Error Handling

The generated SQL query is executed against the database, with careful handling of potential errors: Any database errors are captured and returned to the SQL Generation Agent. The agent iteratively modifies the query based on error messages, re-executing until successful or a maximum number of attempts is reached. Upon successful execution, only the results along with the final SQL query are presented to the User Interaction Agent, deliberately excluding the extensive table documentation and knowledge retrieval artifacts used during generation. This approach significantly reduces token length in the conversation, which not only improves system performance but also enhances accuracy by maintaining focus on the essential information when presenting insights to the user.

IV. System Evolution and Iterations

The current system architecture emerged through several iterations, each addressing limitations discovered during development and testing.

Topic-based Single Agent Approach: Our initial implementation used a single agent where users selected a topic before asking questions. This agent then loaded pre-defined sample queries and table documentation for that topic. This proved ineffective due to: a) Limitations on the number of sample queries that could be included in context, b) Inability to handle cross-topic questions, c) Overwhelming the context with irrelevant table documentation.

Retrieval by Embedding the whole User Question: We evolved to a RAG-based approach that dynamically fetched relevant sample SQL queries based on the user question. However, embedding-based retrieval introduced new challenges. When embedding entire user questions (containing dimensions like country, site, category), these dimensions disproportionately influenced similarity matching. For example, the user question "the number of listings in Poland in January 2025 in the fashion category" incorrectly matched with "the number of orders in Poland in January 2025 in the fashion category" with a high cosine similarity of 0.87, despite the key metrics being different (listings vs. orders). Meanwhile, a question containing the same metric type - "number of orders in Poland in 2023" - achieved only 0.67 similarity. This phenomenon, which we term "dimension dominance", occurs when contextual elements in the embedding space overpower the semantic significance of key analytical concepts like metrics and KPIs. The standard cosine similarity measure between normalized embedding vectors fails to prioritize these business-critical distinctions when they represent only a small portion of the overall token set in the query.

Metric-Focused Retrieval: We found that focusing retrieval on the extracted metrics rather than the entire question produced significantly better results. This approach prioritizes the metric (calculating listings vs. orders) over circumstantial dimensions, reduces noise from contextual details that should be handled separately as filters, and improves the relevance of retrieved examples for SQL generation.

This iterative refinement process led to the current architecture that separates analytical entity extraction from knowledge retrieval, improving both precision and flexibility. Enabled by the customized RAG-based approach, the current system can look up data in 20 different database tables, and answer questions about 94 distinct metrics.

V. Evaluation and Lessons Learned

Given the innovative nature of the system, the absence of established public benchmarks necessitated a tailored approach to evaluate its performance. To address this, we developed a custom Slack bot designed to respond to data-related queries within a dedicated Slack channel. The channel comprised 128 testers representing diverse roles, including Data Analysts, Product Managers, and members of the leadership team. Participants were instructed to pose real-world, data-driven questions encountered in their daily workflows. The quality of the bot’s responses was assessed through user feedback via thumbs-up or thumbs-down reactions to its replies.

Over a six-month testing period, the bot answered 852 questions, of which only 51 were rated negatively. The overall satisfaction rate is 94%.

The development of this agentic system has yielded valuable insights that can inform future implementations in natural language data retrieval. Combining deterministic methods with large language models has proven to be a superior approach, as deterministic processes reduce unpredictability, cost, and execution time while enhancing reliability—particularly evident in tasks like SQL parsing for table name extraction, or handling domain-specific terminologies through automated abbreviation replacement and maintained glossaries has been crucial for reducing confusion and improving accuracy. The system's modular architecture, separating concerns across distinct agents, significantly improves maintainability and makes it easier to identify where an improvement is needed.

Furthermore, the implementation of user confirmation loops ensures alignment between system understanding and user intent, substantially reducing the likelihood of erroneous data retrieval. Breaking down complex queries into smaller, manageable components has simplified processing and enhanced the system's ability to handle complexity. The structured extraction of analytical entities, combined with similarity-based retrieval methods, has improved SQL query precision, while enabling the SQL Generation Agent to iterate based on database error messages has enhanced query generation robustness. This balanced integration of AI flexibility with deterministic precision demonstrates the potential for creating efficient and accurate natural language interfaces for complex data retrieval tasks across specialized domains.

VI. CONCLUSION

We have presented an agentic system that effectively translates natural language queries into SQL statements, enabling users with little or no SQL skills to run complex data queries for insights. Our multi-agent architecture handles domain-specific ambiguities through interactive user confirmation, enabling precise data retrieval while democratizing data access for users with varying levels of technical expertise. This modular approach demonstrates significant advantages in complex data environments like OLX, while remaining adaptable to other domain-specific challenges. The system was tested with over 850 real-world queries during a six-month evaluation period, achieving 94% positive user ratings—validating its practical utility in democratizing data access while providing a scalable framework for domain-specific optimization. Up until now, the system has already replied to more than 9000 user messages. Future research will explore the scalability enhancements and the integration with broader data ecosystems to further augment the utility of this approach.

REFERENCES

[1] V. Zhong, C. Xiong, and R. Socher, "Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning," arXiv:1709.00103 [cs.CL], 2017.

[2] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch, "Natural language interfaces to databases—an introduction," *Journal of Natural Language Engineering*, vol. 1, no. 1, pp. 29–81, 1995.

[3] F. Li and H. V. Jagadish, "Constructing an interactive natural language interface for relational databases," *Proceedings of the VLDB Endowment*, vol. 8, no. 1, pp. 73-84, 2014.

[4] Y. Mellah, A. Rhouati, E. H. Ettifouri, and B. Toumi, "SQL generation from natural language: A sequence-to-sequence model powered by the transformers architecture and association rules," *Journal of Computer Science*, vol. 17, no. 5, pp. 480-489, 2021.

[5] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," arXiv:2304.03442 [cs.HC], 2023.

[6] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J. Wen, "A survey on large language model based autonomous agents," Front. Comput. Sci., vol. 18, no. 6, Mar. 2024, doi: 10.1007/s11704-024-40231-1.

[7] X. Zhu, Q. Li, L. Cui, and Y. Liu, "Large language model enhanced text-to-SQL generation: A survey," arXiv:2410.06011 [cs.DB], 2024.

[8] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," arXiv:2005.11401 [cs.CL], 2021.