Towards Executing Sloppy SQL Queries Over Tabular Data Lakes

Jan-Micha Bodensohn*
Technical University of Darmstadt
jan-micha.bodensohn@tu-darmstadt.de
Darmstadt, Germany

Jakob Steinke*
Technical University of Darmstadt
jakob.m.steinke@gmail.com
Darmstadt, Germany

Carsten Binnig
Technical University of Darmstadt & hessian.ai & DFKI
carsten.binnig@tu-darmstadt.de
Darmstadt, Germany

***Abstract*—Data lakes store raw tabular data without enforcing a consistent schema, making data ingestion easy but exploration and querying difficult. Users must often combine multiple tools to find the required tables and prepare them for each query. To address this gap, we tackle a new problem setting where users directly query data lakes by writing so-called sloppy SQL queries, which simply assume an idealized schema for each query. We further present SQLake, a multi-agent framework which executes such sloppy queries by semantically rewriting them for the data lake, thus eliminating any data preparation overhead. SQLake uses a set of Data Lake Actions that enable the agents to interact with the data lake, including specialized retrievers, a Join Path Finder, and a UDF Builder to handle noise. Our initial evaluation shows the promise of this approach, bringing us closer to our goal of making data lakes as easy to query as relational databases.**

***Index Terms*—table, data lake, sql, query, llm, agent**

I. Introduction

Data lakes are hard to query. Large organizations often rely on data lakes to store vast amounts of tabular data. Unlike relational databases, data lakes store loose tables in their original form without enforcing a consistent schema. While this flexibility simplifies data ingestion, it comes at a cost: querying and analysis become much more challenging.

In relational databases, a user can formulate precise SQL queries against a known schema with clearly defined foreign-key relationships and consistent data types. In data lakes, however, there are no schemas with foreign-key relationships connecting individual tables, and the data is often noisy, with formatting that varies between tables. What data exists in the data lake is often not known to the user, let alone how it is organized. Since SQL queries need to make assumptions about the data and schema, the user cannot easily write queries that operate directly on the data lake tables.

A motivating example. Take for example the SQL query shown in Fig. 1. Whereas this query is easy to execute on a relational database with a matching schema, it may fail on a data lake because the attribute *currency* is actually called *currency_code*, the revenue is spread across multiple *sales* tables for each site, and there are no explicit foreign-key relationships specifying how these tables should be joined. Moreover, the underlying data may be noisy, using different representations such as EUR and € for the currency.

*Authors with equal contribution.

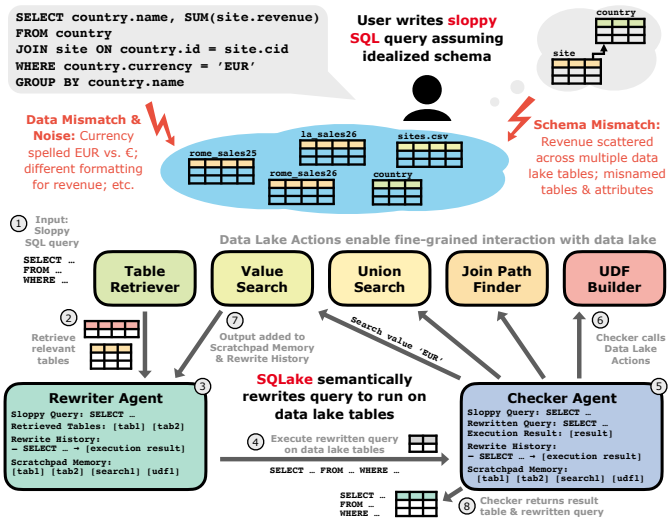

Fig. 1. Sloppy SQL enables users to directly query data lakes with SQL by simply assuming an idealized schema for their query. SQLake uses a feedback loop of two LLM agents, the Rewriter and the Checker, to semantically rewrite the user-written sloppy query to make it executable on the data lake tables.

To answer the query, the user would have to combine multiple approaches to first retrieve the required tables [1], determine how to connect them [2], map between schemas [3], and clean up noisy entries before executing the query. This data preparation step is both time-consuming and error-prone, often requiring repeated exploration and substantial knowledge about the tools and data to obtain a correct result.

Querying data lakes with sloppy SQL. To address these challenges, we propose a new paradigm that enables the user to directly query data lakes by writing so-called *sloppy* SQL queries. Each sloppy query simply assumes an idealized schema to express the query according to the user's domain understanding, freeing them from manually searching through the data lake and preparing the required data [4].

At the same time, sloppy SQL enables the user to specify precisely which information to compute by building on the well-defined execution logic of SQL. For example, while the meaning of a slightly ambiguous question like *What is the site revenue for Euro countries?* may become clear when consulting the schema of a relational database [5], a data lake does not provide such supporting structure.

Executing sloppy SQL with SQLake. As our main contribution, we present *SQLake*, a novel approach which executes such sloppy SQL queries over tabular data lakes. The core idea behind SQLake is to semantically rewrite the user-written sloppy query to make it executable on the available data lake tables while still capturing the user's intent.

Rather than preparing the data for the query, SQLake adapts the query to the data. The output of the system is thus not just the intended result table of the user-written sloppy query, but also includes the rewritten query which produces this table from the data lake, serving as a traceable explanation of the query result. Moreover, data integration and cleaning are limited to the steps necessary to execute each query.

II. A Multi-Agent Query Rewriting Framework

To execute a sloppy SQL query over a tabular data lake, SQLake must first identify the required tables in the data lake and then adapt the query to their data and schema. To accomplish this, it combines two central ideas: *Semantic Query Rewriting* using a feedback loop of two LLM agents to rewrite the query, and *Data Lake Actions* that enable the agents to interact with the data lake. In the following, we briefly explain the two ideas based on the example in Fig. 1.

(1) Semantic Query Rewriting. Given the sloppy query shown in Fig. 1,① SQLake first retrieves potentially relevant tables from the data lake.② It then provides small previews (first three rows) of the top 10 tables together with the sloppy query to the *Rewriter* agent, which generates a rewritten version of the query to adapt it to the retrieved tables.③ SQLake then tries to execute the rewritten query on the data lake tables and forwards the execution result to the *Checker* agent.④ The Checker inspects the rewritten query and its execution result to determine if it accurately captures the user's intent.⑤ Moreover, it can call several Data Lake Actions to provide additional context for the following iterations.⑥ The outputs of the Data Lake Actions and the previews of the tables used by the rewritten query are stored in a scratchpad memory, and the rewritten query, its execution result, and the Checker's feedback are added to a rewrite history.⑦ Both the scratchpad memory and the rewrite history are provided to the Rewriter and the Checker in the following iterations, enabling SQLake to carefully manage the context of the two LLM agents. Finally, the entire process, starting with the retrieval of potential additional relevant tables, is repeated until the Checker decides to return the result to the user.⑧

Through the Semantic Query Rewriting, SQLake iteratively adapts the user-written sloppy query to the tables in the data lake. This feedback loop enables the two LLM agents to reason about the data and query and to react to difficulties such as execution failures, schema mismatches, and even noisy data through the use of appropriate Data Lake Actions.

(2) Data Lake Actions. The Data Lake Actions provide the interface between SQLake and the data lake. In contrast to the broad retrieval methods used by existing approaches [6], they enable much more fine-grained interactions with the data.

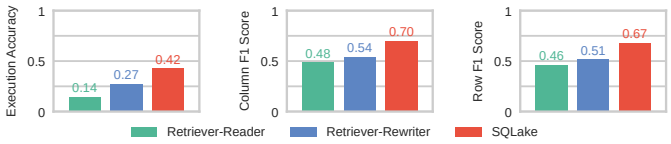

Fig. 2. SQLake outperforms both baselines. We measure execution accuracy as well as column- and row-wise F1 scores to consider partially-correct tables.

For example, in Fig. 1, SQLake needs to determine the exact string value for the filter condition, so it calls a specialized retrieval action to perform a fuzzy search for individual cell values. Other specialized retrieval actions include searching for unionable tables [7] and for tables that provide specific missing columns. Moreover, SQLake includes a Join Path Finder to deal with the missing foreign-key relationships in data lakes and an action to build custom Python UDFs to handle noisy values such as EUR and €.

The Data Lake Actions are called by the Checker agent, and their outputs are fed back into the Rewriter agent to consider in the following query rewriting step. Notably, the actions themselves do not become part of the rewritten query, but instead support the Rewriter in adapting the query to the data lake. This ensures that the rewritten query is executable on its own without requiring further involvement of SQLake.

III. Initial Evaluation

In our initial evaluation, we compare SQLake against two baselines: a RAG-style *Retriever-Reader* which retrieves relevant tables and then uses an LLM to generate the result table [6], and a *Retriever-Rewriter* which retrieves tables and then rewrites the sloppy query in a single step. All approaches use GPT-5.4 with reasoning effort set to medium [8].

Since sloppy SQL queries over tabular data lakes are a novel problem setting, there are no adequate evaluation datasets for this task. Therefore, we evaluate on a small hand-crafted dataset of 97 queries over a data lake with 83 tables. We measure the table-level execution accuracy as well as column- and row-wise F1 scores to consider partially-correct tables.

As shown in Fig. 2, SQLake outperforms both baselines, with F1 scores reaching as high as 0.70. Nevertheless, executing sloppy SQL queries over tabular data lakes remains a challenging problem, as apparent in the low table-level execution accuracy. We find that many failures are caused by the large overlap between the tables in the data lake, making it hard to decide which to use as input for the rewritten query.

IV. Outlook

In this extended abstract, we proposed the idea of sloppy SQL to directly query tabular data lakes and briefly showcased SQLake, a novel approach to execute such queries by rewriting them for the data lake tables. Although our initial evaluation shows promising results, executing SQL queries over data lakes remains a challenging problem. In future work, we plan to extend our set of Data Lake Actions into a full algebra of data operations that enable LLMs to reason through the messy data in large data lakes and compose complex query plans.

ACKNOWLEDGMENT

This research was partially funded by the German Federal Ministry of Research, Technology and Space (BMFTR) as part of the Software Campus research program (16IS23067) within the Project Management Agency for Artificial Intelligence. It was also supported by the BMFTR and the state of Hesse as part of the NHR program and the HMWK cluster project 3AI, by the LOEWE Spitzenprofessur of the state of Hesse (III 5-519/05.00.003-(0005)), and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy (EXC3057/1 "Reasonable Artificial Intelligence", Project No. 533677015). We also thank DFKI Darmstadt and hessian.AI for their support.

REFERENCES

[1] S. Zhang and K. Balog, "Ad Hoc Table Retrieval using Semantic Similarity," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*. Lyon, France: ACM Press, 2018, pp. 1553–1562. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3178876.3186067

[2] Y. Dong, C. Xiao, T. Nozawa, M. Enomoto, and M. Oyamada, "Deepjoin: Joinable table discovery with pre-trained language models," *Proc. VLDB Endow.*, vol. 16, no. 10, p. 2458–2470, Jun. 2023. [Online]. Available: https://doi.org/10.14778/3603581.3603587

[3] Y. Liu, E. H. M. Pena, A. Santos, E. Wu, and J. Freire, "Magneto: Combining small and large language models for schema matching," *Proc. VLDB Endow.*, vol. 18, no. 8, p. 2681–2694, Apr. 2025. [Online]. Available: https://doi.org/10.14778/3742728.3742757

[4] B. Hättasch and C. Binnig, "Databases: From Data Storage Towards Partners for Information Access and Discovery," in *AutomationXP25: Hybrid Automation Experiences*, 2025. [Online]. Available: https://ceur-ws.org/Vol-4101/paper11.pdf

[5] D. Gomm, C. Wolff, and M. Hulsebos, "Are We Asking the Right Questions? On Ambiguity in Natural Language Queries for Tabular Data Analysis," in *EurIPS 2025 Workshop: AI for Tabular Data*, Nov. 2025. [Online]. Available: https://openreview.net/forum

[6] J. Herzig, T. Müller, S. Krichene, and J. Eisenschlos, "Open Domain Question Answering over Tables via Dense Retrieval," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, Jun. 2021, pp. 512–519. [Online]. Available: https://aclanthology.org/2021.naacl-main.43

[7] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data," *Proceedings of the VLDB Endowment*, vol. 11, no. 7, pp. 813–825, Mar. 2018. [Online]. Available: https://dl.acm.org/doi/10.14778/3192965.3192973

[8] A. Singh *et al.*, "OpenAI GPT-5 System Card," Dec. 2025. [Online]. Available: http://arxiv.org/abs/2601.03267