SHORT-PAPER

Demonstrating SQLBarber: Leveraging Large Language Models to Generate Customized and Realistic SQL Workloads

JIALE LAO, Cornell University, Ithaca, NY, United States

IMMANUEL TRUMMER, Cornell University, Ithaca, NY, United States

Open Access Support provided by:

Cornell University

Published: 22 June 2025

Citation in BibTeX format

SIGMOD/PODS '25: International Conference on Management of Data
June 22 - 27, 2025
Berlin, Germany

Conference Sponsors:
SIGMOD

Demonstrating SQLBarber: Leveraging Large Language Models to Generate Customized and Realistic SQL Workloads

Jiale Lao
Cornell University
Ithaca, New York, USA
jiale@cs.cornell.edu

Immanuel Trummer
Cornell University
Ithaca, New York, USA
itrummer@cornell.edu

Abstract

Database research and development require a large volume of SQL queries for benchmarking. However, it is difficult to obtain real SQL queries due to privacy issues, and existing SQL generation methods are limited in customization and satisfying realistic constraints. To address this problem, we propose SQLBarber, a novel system leveraging Large Language Models (LLMs) to generate customized and realistic SQL workloads. SQLBarber (a) eliminates the need for users to manually craft SQL templates in advance, while providing the flexibility to accept high-level natural language specifications to constrain the SQL templates, (b) scales efficiently to produce a large number of queries satisfying any user-defined cost distribution (e.g., cardinality, execution plan cost, or execution time), and (c) analyzes execution statistics obtained from Amazon Redshift and Snowflake to derive both the specifications for SQL templates and the cost distribution of SQL queries, ensuring that these constraints reflect real-world workloads. This demonstration allows the audience to experience SQLBarber in action: (1) provide their customized specifications on SQL templates, (2) gain insights on the effect of SQL template and predicate values on SQL costs, and (3) explore real-world specifications of templates as well as cost distributions of queries, and constrain their SQL queries to a desired specification and distribution, with the flexibility to try out different LLM and SQL cost types. A video demonstration is available at https://youtu.be/qOuAKVXXcdM.

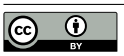
CCS Concepts

• **Information systems** → **Structured Query Language**; **Database performance evaluation**.

Keywords

SQL Generation, Large Language Model, Database Benchmarking

ACM Reference Format:
Jiale Lao and Immanuel Trummer. 2025. Demonstrating SQLBarber: Leveraging Large Language Models to Generate Customized and Realistic SQL Workloads. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion '25), June 22–27, 2025, Berlin, Germany.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3722212.3725101



SIGMOD-Companion '25, Berlin, Germany

ACM ISBN 979-8-4007-1564-8/2025/06
https://doi.org/10.1145/3722212.3725101

1 Introduction

Database research and development demand extensive experiments and benchmarking to prevent performance regressions. However, due to privacy issues, it is difficult to obtain real-world SQL queries to serve as realistic benchmarks. As a result, SQL generation has attracted significant attention from both academia and industry.

Unfortunately, existing SQL generation methods fail to produce customized SQL queries that satisfy realistic constraints. Some approaches rely on predefined rules or finite-state machines to generate SQL templates randomly [1, 10], lacking the flexibility to enforce structural and feature constraints on the templates. Other methods [2] require users to manually craft high-quality SQL templates and subsequently generate queries that meet cardinality or execution time requirements, which is expensive and not scalable.

Thus we develop SQLBarber, a novel Large Language Models (LLMs)-based SQL generation system that: (a) eliminates the need for users to manually craft SQL templates in advance, while providing the flexibility to accept high-level natural language specifications to constrain the SQL templates, (b) scales efficiently to produce a large number of queries that satisfy any user-defined cost distribution (e.g., cardinality, execution plan cost, or execution time), and (c) analyzes execution statistics obtained from Amazon Redshift [8] and Snowflake [9] to derive both the specifications for SQL templates and the cost distribution of SQL queries, ensuring that these constraints reflect real-world workloads.

This demo lets attendees experience SQLBarber in action, using a three-stage process to generate customized, cost-aware, and realistic SQL workloads. Firstly, SQLBarber utilizes an LLM-powered pipeline to design customized SQL templates. This pipeline takes as input the target database and user-specified natural language constraints on SQL templates. Then it uses database schema summary, join path generation, and template generation, along with validation and rewrite to ensure the templates satisfy the constraints. Secondly, SQLBarber employs a profiling stage to evaluate the capability of each SQL template to generate queries of different costs under varying predicate values. This is achieved by substituting placeholders in the previously generated SQL templates by different values to get executable queries, which are subsequently evaluated on the DBMS to get costs (e.g., cardinality or cost). Thirdly, based on the collected profiling statistics, SQLBarber leverages Bayesian Optimization (BO) to explore the predicate value space efficiently, and identifies a set of queries that satisfy the user-specified cost distribution. Finally, SQLBarber allows users to explore real-world specifications of SQL templates as well as realistic cardinality and execution time distributions, and constrain their queries to a desired specification and distribution. In addition, we offer attendees the opportunity to explore the effect of different LLM and cost types.

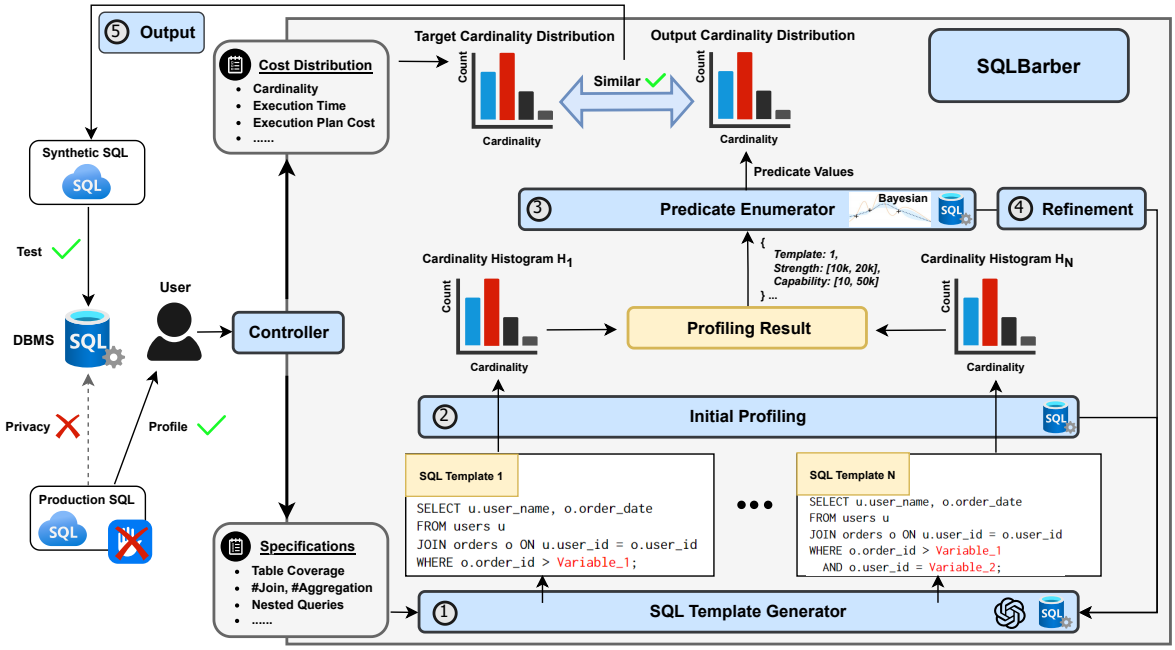


Figure 1: System Overview of SQLBarber

2 System Overview of SQLBarber

Objective. Figure 1 presents an overview of the SQLBarber system, leveraging Large Language Models (LLMs) to generate customized, cost-aware, and realistic SQL queries. **Customization** refers to generating queries based on SQL templates that strictly align with user-provided natural language specifications (e.g., the percentage of table accessed, the number of joins and aggregations, the presence of nested queries). **Cost-awareness** ensures that the generated queries collectively adhere to a user-specified cost distribution, such as execution time or cardinality. **Realism** is achieved by analyzing real-world execution statistics obtained from Amazon Redshift [8] and Snowflake [9]. These statistics are used to derive both the specifications for SQL templates and the cost distribution of SQL queries, ensuring the constraints reflect production workloads.

Scenario. The leftmost part of Figure 1 depicts the scenario. Both industrial database development and academic database research require extensive experiments to prevent performance regression before introducing new features or updating existing components. However, it is rather difficult to obtain real-world SQL queries due to privacy issues. Recently, there has been a trend of publishing production-level query profiles for the database community, led by Amazon Redshift [8] and Snowflake [9]. Therefore, we address the problem of generating customized SQL queries that are constrained by realistic constraints. These constraints restrict both the SQL templates and the SQL costs, derived from real-world query profiles.

Workflow. Figure 1 shows the workflow of the SQLBarber system. SQLBarber takes as input the target database, the natural language specification to customize the SQL templates, and a desired cost distribution (e.g., cardinality) to constrain the generated SQL queries. ❶ SQLBarber uses an SQL Template Generator to create templates based on the user-provided natural language specification. The specification can define both the structure of the SQL template (e.g., the inclusion of nested subqueries) and its features (e.g., the number of tables acessed, the number of joins and aggregations, or the presence of operators such as groupby). First, the Template Generator connects to the target database and produces a text summary of the database schema and the joinable paths. Subsequently, the Template Generator combines the user specification, the database schema, and the joinable paths to construct a prompt to invoke LLM to generate SQL templates. However, due to the hallucination problem in LLM (i.e., returnning a response that is false or entirely made up), it is possible that the generated SQL templates do not satisfy the specification or are invalid. Since LLM possess the capability to evaluate and self-correct their outputs, SQLBarber employs LLM to reason about whether an SQL template satisfies the specification and to identify the reasons behind it. If the template fails, LLM uses the reasoning results along with the error messages from the DBMS (if applicable) to iteratively rewrite this template, ensuring that it becomes both executable and compliant with the user specification. ❷ SQLBarber employs an initial profiling phase to assess the ability of each generated SQL template to produce queries with different costs. The SQL templates generated in the previous step are SQL structures without predicate values. These predicate values are selected from columns in the database. SQLBarber uses a space-filling sampling strategy to evenly sample values from each column. By filling these templates with different predicate values, SQLBarber can produce executable SQL queries with varying costs. These queries are then evaluated on the DBMS to get the desired cost metrics (e.g., estimated cardinality and cost from the execution plan, or actual execution time). After profiling all the SQL templates, SQLBarber gathers information on which templates can produce queries within specific cost ranges, and determines which templates perform most efficiently within those ranges. Note that this phase

only explores a limited number of predicate values for each template (e.g., 50), ensuring that this process is cost-efficient and useful to guide the next exploration stage.

❸ SQLBarber uses a Predicate Enumerator to strategically explore the predicate values for the generated SQL templates, and identify a set of SQL queries satisfying the user-specified cost distribution. Given the desired number of SQL queries and the target cost distribution, SQLBarber calculates the number of missing queries for each cost range after filling the target distribution with queries generated in the profiling stage. Next, SQLBarber ranks the cost ranges based on the number of missing queries in descending order. For each cost range, SQLBarber identifies templates that can produce queries within this range based on the profiling result, and further explores the predicate value space of each template to produce more queries within the range. To improve efficiency, SQLBarber utilizes Bayesian Optimization (BO) as the search algorithm, balancing between the exploitation of well-performing predicate values and the exploration of unknown predicate values.

❹ SQLBarber includes a Template Refinement component that generates new SQL templates by refining existing ones. This component is automatically triggered after ❷ the initial profiling phase and during ❸ the predicate enumeration phase, if existing SQL templates cannot produce queries within a specific target cost range. The refinement process involves identifying SQL templates that produce costs closest to the desired range, and invoking LLM to refine these templates so that the costs fall within the target range. The refinement may change the current join path or modify the SQL structure to make the template more or less complex, depending on whether the goal is to increase or decrease the costs.

❺ Finally, SQLBarber outputs a set of SQL queries that satisfy the user-specified cost distribution. These queries are customized, cost-aware, and realistic, making them suitable for testing and benchmarking purposes.

3 Demonstration Plan of SQLBarber

The demonstration setup includes a table with a laptop connected to a portable projector. The laptop is pre-configured to run SQLBarber to generate the desired SQL queries for databases deployed either locally or on a remote server. While internet access to OpenAI's GPT service is preferred, it is not required, as the demonstration includes cached results for convenience. The demonstration is structured to last five minutes, ensuring clarity and engagement, with additional details available upon request.

Each visitor receives a brief introduction to explain the system and its web interface. The visitors can then proceed to use the SQLBarber system themselves, following the instructions on the user interface. Users begin by either configuring system parameters themselves (Figure 2), or by simply using the default parameters with minimum effort. The interface allows users to select the target database, with popular benchmarks such as TPC-H and TPC-C as options. Users can specify the number of queries to generate, the cost type (e.g., cardinality, execution plan cost, or execution time), and the desired cost range. Users also have the flexibility to provide customized specifications for SQL templates using natural language, and experiment with different LLMs. The demonstration further enables users to adjust the number of profiling samples

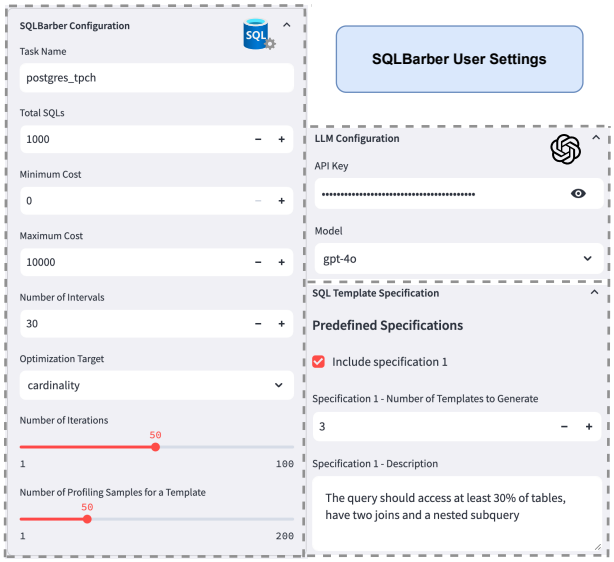


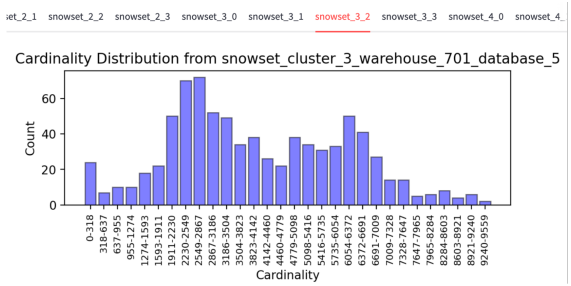
Figure 2: User Interface of SQLBarber: User Settings

and optimization iterations, allowing them to evaluate the system's robustness and observe the impact of these settings on performance.

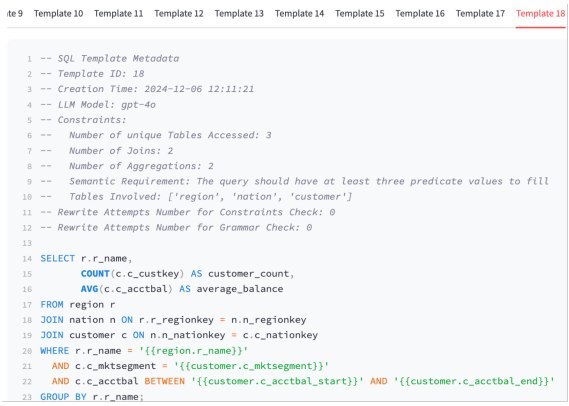
Then users can experience SQLBarber in action (Figure 3). Firstly, users can explore real-world cost distributions from Amazon Redshift and Snowflake, and select a desired distribution as the optimization target (Step 1, Figure 3a). Secondly, they invoke the SQL Template Generator to create SQL templates based on the natural language specifications. The generated SQL templates include placeholders for predicate values, and incorporate meta-information such as the LLM used, as well as constraints and semantic requirements for users to review (Step 2, Figure 3b). Thirdly, users can start the profiling phase for all the generated SQL templates. SQLBarber substitutes the placeholders with different predicate values to produce executable SQL queries and then evaluates these queries on the target database to get their costs. After that, the ability of each SQL template to generate queries spanning different cost ranges with varying probabilities is evaluated. The results are visualized to help users understand the cost profiles of the templates (Step 3, Figure 3c). In the fourth step, users employ the Predicate Enumerator to identify a set of SQL queries that match the user-specified cost distribution. Each iteration is a Bayesian Optimization process strategically exploring predicate values to produce SQL queries to fill the cost ranges that are still short of queries. As iterations continue, the cost distribution of the currently generated queries becomes more and more similar to the target distribution, and we use a Wasserstein Distance to mathematically measure the similarity of the two distributions. The differences between the two distributions and the measured distance are visualized for users in real-time as each iteration completes (Step 4, Figure 3d). Finally, users can review a visual summary of the distance over the iterations, analyze the time cost for each step, and export the generated queries for benchmarking purposes (Step 5, Figure 3e).

4 Related Work

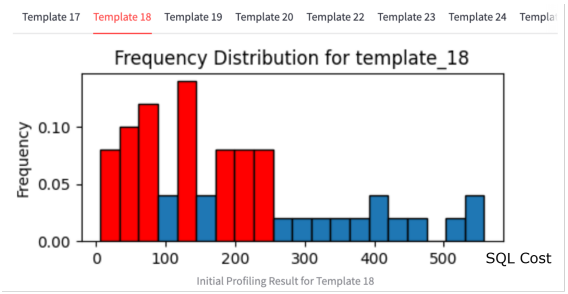
SQLBarber relates to prior works generating SQL queries for DBMS testing and benchmarking [1, 2, 10]. These approaches either rely



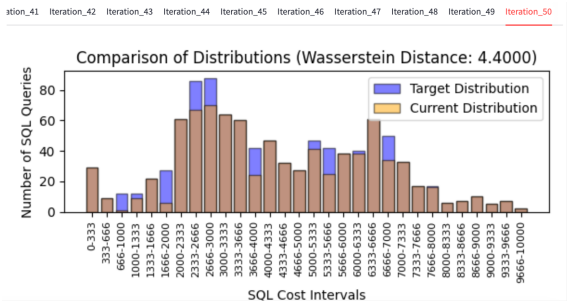
(a) Step 1: Real-World Distributions Exploration



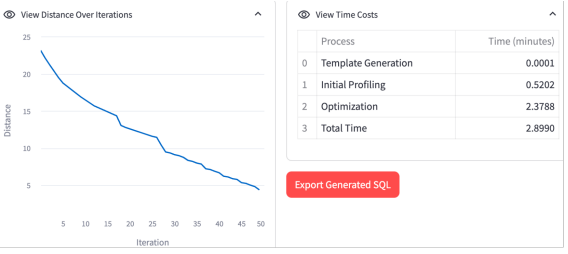
(b) Step 2: SQL Templates Generation



(c) Step 3: Profiling Phase for Generated SQL Templates



(d) Step 4: Predicate Enumeration



(e) Step 5: Result Summary and Export

Figure 3: User Interface of SQLBarber: Workflow

on predefined rules to randomly generate SQL templates and thus lack the flexibility to impose structural and feature constraints on templates [1, 10], or rely on manually crafted high-quality SQL templates and therefore are expensive and not scalable [2]. Differently, SQLBarber utilizes LLMs to generate and refine SQL templates based on natural language specifications, achieving customization and scalability at the same time. Moreover, SQLBarber uniquely provides the ability to produce a set of queries satisfying any given cost distribution, which is achieved by a Bayesian Optimization-based Predicate Enumerator. By aligning with realistic cost distributions, SQLBarber ensures the generated queries are as realistic as possible.

More broadly, SQLBarber relates to prior works on leveraging LLMs to enhance DBMS, including DBMS tuning [3–5], query code generation [7], and natural language query interface [6]. Note that Text-to-SQL [6] focuses on translating natural language questions into SQL queries for retrieving specific answers, without considering the complexity of SQL templates and query costs. In contrast, SQLBarber does not rely on natural language questions as inputs and instead emphasizes generating queries with varying structures, complexities, and costs to support benchmarking objectives.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Award No. 2239326.

References

[1] 2020. A. Seltenreich. Sqlsmith. https://github.com/anse1/sqlsmith
[2] Nicolas Bruno, Surajit Chaudhuri, and Dilys Thomas. 2006. Generating queries with cardinality constraints for dbms testing. *IEEE Transactions on Knowledge and Data Engineering* 18, 12 (2006), 1721–1725.
[3] Victor Giannakouris and Immanuel Trummer. 2024. Demonstrating -Tune: Exploiting Large Language Models for Workload-Adaptive Database System Tuning. In *Companion of the 2024 International Conference on Management of Data* (Santiago AA, Chile) *(SIGMOD/PODS '24)*. Association for Computing Machinery, New York, NY, USA, 508–511. doi:10.1145/3626246.3654751
[4] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2024. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. *Proc. VLDB Endow.* 17, 8 (may 2024), 1939–1952. doi:10.14778/3659437.3659449
[5] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Yuanchun Zhou, Mingjie Tang, and Jianguo Wang. 2024. A Demonstration of GPTuner: A GPT-Based Manual-Reading Database Tuning System. In *Companion of the 2024 International Conference on Management of Data* (Santiago, Chile) *(SIGMOD '24)*. Association for Computing Machinery, New York, NY, USA, 4 pages. doi:10.1145/3626246.3654739
[6] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *Proc. ACM Manag. Data* 2, 3, Article 127 (May 2024), 28 pages. doi:10.1145/3654930
[7] Immanuel Trummer. 2022. CodexDB: synthesizing code for query processing from natural language instructions using GPT-3 codex. *Proc. VLDB Endow.* 15, 11 (July 2022), 2921–2928. doi:10.14778/3551793.3551841
[8] Alexander van Renen, Dominik Horn, Pascal Pfeil, Kapil Vaidya, Wenjian Dong, Murali Narayanaswamy, Zhengchun Liu, Gaurav Saxena, Andreas Kipf, and Tim Kraska. 2024. Why TPC is Not Enough: An Analysis of the Amazon Redshift Fleet. *Proc. VLDB Endow.* 17, 11 (aug 2024), 3694–3706. doi:10.14778/3681954.3682031
[9] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. 2020. Building An Elastic Query Engine on Disaggregated Storage. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 449–462. https://www.usenix.org/conference/nsdi20/presentation/vuppalapati
[10] Lixi Zhang, Chengliang Chai, Xuanhe Zhou, and Guoliang Li. 2022. Learned-SQLGen: Constraint-aware SQL Generation using Reinforcement Learning. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 945–958. doi:10.1145/3514221.3526155