Demo of QueryBooster: Supporting Middleware-Based SQL Query Rewriting as a Service

Qiushi Bai
University of California, Irvine
qbai1@uci.edu

Sadeem Alsudais
University of California, Irvine
salsudai@uci.edu

Chen Li
University of California, Irvine
chenli@ics.uci.edu

ABSTRACT

Query rewriting is an important technique to optimize SQL performance in databases. With the prevalent use of business intelligence systems and object-relational mapping frameworks, existing rewriting capabilities inside databases are insufficient to optimize machine-generated queries. In this paper, we propose a novel system called "QueryBooster," to support SQL query rewriting as a cloud service. It provides a powerful and easy-to-use Web interface for users to formulate rewriting rules via a language or express rewriting intentions by providing example query pairs. It allows multiple users to share rewriting knowledge and automatically suggests shared rewriting rules for users. It requires no modifications or plugin installations to applications or databases. In this demonstration, we use real-world applications and datasets to show the user experience of QueryBooster to rewrite their application queries and share rewriting knowledge.

PVLDB Reference Format:
Qiushi Bai, Sadeem Alsudais, and Chen Li. Demo of QueryBooster: Supporting Middleware-Based SQL Query Rewriting as a Service. PVLDB, 16(12): 4038 - 4041, 2023.
doi:10.14778/3611540.3611615

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ISG-ICS/QueryBooster.

1 INTRODUCTION

System performance is critical in database applications where users need answers quickly to make timely decisions. SQL query rewriting [5, 10] is an optimization technique that transforms an original query to a rewritten one that computes the same answers with higher performance. Although query rewriting has been studied extensively as part of the query optimizer inside a database [5, 6], recent works [3, 16] have shown that purely relying on the rewritings inside traditional databases is insufficient to optimize modern query workloads. For instance, with the prevalent use of business intelligence systems (e.g., Tableau and PowerBI) and Web applications developed using object-relational mapping (ORM) frameworks, these machine-generated queries can be difficult for databases to optimize [16] and domain-specific knowledge and human-crafted rewriting rules are necessary to optimize workloads from different applications [2, 11]. We can miss query rewriting opportunities (e.g., rewritings shown in Figure 7) due to various reasons. For instance, both the application and the database layers are black boxes and cannot be modified. Another reason is that existing rewriting plugins of databases have limited expressive power for users to express their rewriting needs.

In this paper, we demonstrate QueryBooster, a middleware-based multi-user system to provide query rewriting between applications and databases as a service. The service intercepts and rewrites an original query from an application before it is sent to a backend database. It provides a web-based interface, using which customers manage their rewritings for different applications and databases. Users formulate rewriting rules using a language or by providing examples. They can see the statistics of different rewritings, such as the number of queries rewritten using a rewriting and the query's performance before and after the rewriting. Since rewriting SQL queries can be hard and time-consuming [17], instead of seeking advice from online forums (e.g., StackOverflow), users can also share and access rewriting rules using QueryBooster. The service requires no plugin installations to the user applications or databases. It also ensures high security as no user data goes through a third-party server. To summarize, QueryBooster has the following benefits. (1) It is easy to use, as users can use the interface to formulate, control, and monitor rewriting rules. (2) It enables users to share their rewriting knowledge and benefit from the wisdom of the crowd. (3) It is non-intrusive, as it requires no plugin installations to the user applications or databases.

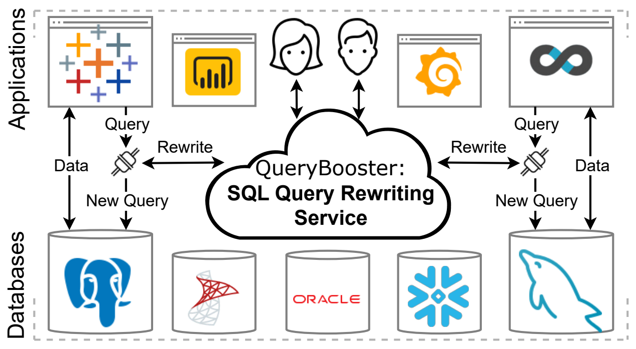


Figure 1: QueryBooster **overview.**

Related Work: Most databases such as AsterixDB, IBM DB2, MS SQL Server, MySQL, Oracle, Postgres, Snowflake, and Teradata do not allow users to customize the rewritings of queries sent to the database. To our best knowledge, only Postgres and MySQL provide a plugin for users to define rules to rewrite queries before sending them to the database. However, their rule-definition

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611615

languages have too limited expressive power, as shown in [11]. Commercial systems also do query rewriting for applications on top of databases. Keebo [8] uses machine learning and approximate query processing (AQP) techniques to accelerate analytical queries. It requires data to go through its server, which may introduce overhead and cause security concerns. EverSQL [4] uses AI techniques to recommend rewriting ideas for queries on MySQL and Postgres. Other systems such as ApexSQL [1] and Toad [13] help database developers analyze query performance bottlenecks. However, none of these solutions allow users to formulate their own rewriting rules to fulfill their rewriting needs. There are also service models such as database-as-a-service [7], function-as-a-service [12], etc. Compared to these systems, QueryBooster is the first system that supports equivalent query rewriting as a service.

2 QUERYBOOSTER SYSTEM OVERVIEW

Figure 2 shows the architecture of QueryBooster. Using the *Web UI*, users can log in to the system through the *User Manager* and manage applications through the *Application Manager*. They can have different sets of rewriting rules for different applications. Through the *Rule Manager* that integrates the technique in [11], QueryBooster provides a powerful interface for users to formulate rewriting rules. It provides an expressive rule language (called VarSQL) for users to define rules. Users can easily express their rewriting needs by specifying the query pattern and its replacement. They can specify additional constraints and actions to express complex rewriting details. In addition, the service allows users to express their rewriting intentions by providing examples, each of which includes an original query and a rewritten one. QueryBooster automatically suggests high-quality rewriting rules based on the examples. The users can choose their desired rules and further modify suggested rules as they want. Rewriting rules are stored in the *Rule Base*.

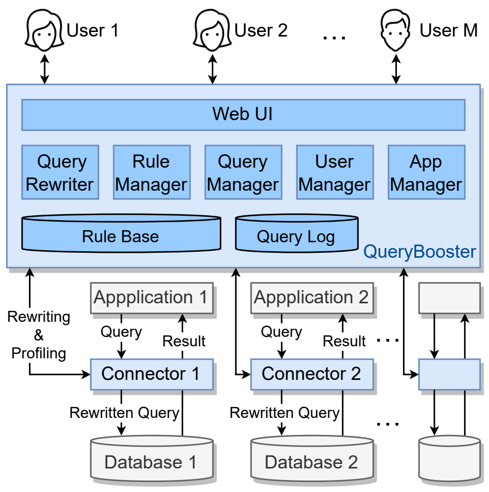


Figure 2: QueryBooster **system architecture.**

QueryBooster provides database *Connectors* for users to download. Without any application or database modifications, a user replaces the original connector with a QueryBooster-provided connector. The connector forwards an original query from the application to the *Query Rewriter*, and sends the rewritten query to the database. When the database returns the result of the new query, the connector sends the query performance information back to the *Query Manager*. Note that the connector does not send query result to QueryBooster. All the query rewriting path and corresponding performance information are stored in the *Query Log*. As a background process, the *Query Manager* periodically runs rewriting rules shared by different users against the workloads in the query log and marks those queries with suggested rules useful to the queries.

Connector License. The connector can be for either a JDBC/ODBC interface or a RESTful interface. Most database vendors provide JDBC/ODBC drivers with an open source license (shown in Table 1). For such drivers, QueryBooster provides a slightly modified version of the driver. For databases with redistribution restrictions on their drivers (e.g., Oracle JDBC driver [9]), QueryBooster provides a software patch for users to compile the customized driver themselves. For applications and databases that communicate through a RESTful interface, QueryBooster provides a proxy web server that intercepts requests between them transparently. We assume the RESTful API endpoint in the application is configurable, i.e., we can switch the target database endpoint to our service.

Table 1: JDBC driver licenses for database vendors.

Database	License	Open source?	Redistribution?
MS SQL Server	MIT	Yes	Yes
MySQL	GNU GPL V2	Yes	Yes
Oracle	Oracle Free	Yes	No
PostgreSQL	BSD-2-Clause	Yes	Yes
Snowflake	Apache-2.0	Yes	Yes

Privacy and security considerations. QueryBooster provides different levels of security guarantees based on the need of users. For organizations that do not want to share their queries with external service providers, QueryBooster can be installed in their intranet behind their network firewall so that no proprietary information leaves their organizations. For organizations that are cost-sensitive and are willing to share their queries and rewritings, they can contribute rewriting rules to the QueryBooster's free service and benefit from the shared rewriting knowledge from other users.

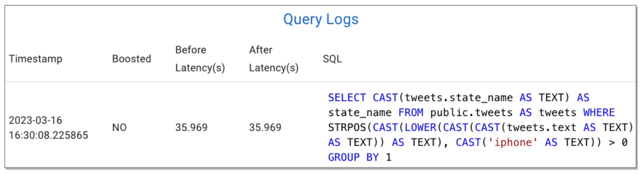
3 DEMONSTRATION SCENARIOS

In this section, we use real-world applications and Twitter datasets to demonstrate the experience of the QueryBooster service for two users to rewrite their application queries and share rewriting knowledge.

Suppose Alice is a database administrator who manages a PostgreSQL database that stores tweet data to support a data analysis team. The team uses Tableau to study the spatial and temporal distributions of keyword-related tweets. After creating a Tableau dashboard on top of PostgreSQL, Alice tries a few visualization queries, and the performance is not satisfying. Therefore, she uses QueryBooster to rewrite the queries for better performance. Through QueryBooster's Web UI, she creates an application and receives an application GUID (Global Unique Identifier) generated from the system. Then, she downloads the provided JDBC driver and places it in the folder of Tableau connectors. Finally, she configures the JDBC driver with the provided application GUID. Alice has now completed setting up QueryBooster.

3.1 Formulating Rules through a Rule Language

To identify the queries with performance bottlenecks, Alice utilizes QueryBooster's query logging feature. She first tries the slow analytics operations on Tableau, and goes to the *Query Logs* page to check the SQL queries sent from Tableau to PostgreSQL. Figure 3 shows the Web UI illustrating the information of one query formulated by Tableau. It shows the query, its timestamp, latency, and whether it has been rewritten or not.



Query Logs

Timestamp	Boosted	Before Latency(s)	After Latency(s)	SQL
2023-03-16 16:30:08.225865	NO	35.969	35.969	SELECT CAST(tweets.state_name AS TEXT) AS state_name FROM public.tweets AS tweets WHERE STRPOS(CAST(LOWER(CAST(CAST(tweets.text AS TEXT) AS TEXT)) AS TEXT), CAST('iphone' AS TEXT)) > 0 GROUP BY 1

Figure 3: The "Query Logs" page of QueryBooster **shows information about queries from the application.**

After investigating those queries, Alice identifies a lot of type-casting expressions (i.e., `CAST(··· AS TEXT)`), as shown in the query in Figure 3. Tableau adds those type-casting expressions to prevent computational overflow or datatype mismatching errors [15]. Based on her knowledge of the tweet data schema, Alice knows that the type-casting expressions are not necessary for the queries. Thus, Alice wants to input a rewriting rule to remove those type-castings from the queries. The VarSQL rule language is easy-to-use, and Alice manually formulates the rule (called "`Remove Cast Text`") as shown in Figure 4.

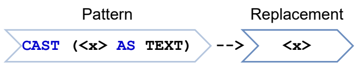
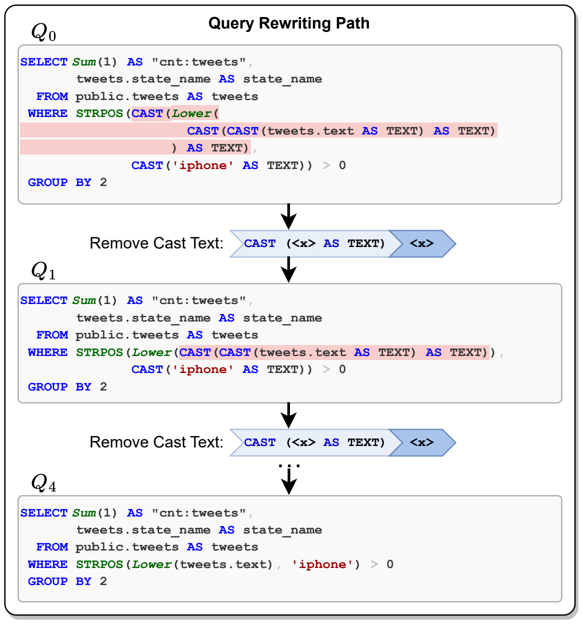


Figure 4: Alice formulates a rule to remove `CAST(··· AS TEXT)` **expressions using the** VarSQL **language [11]. In** VarSQL**'s syntax,** `<x>` **is an element-variable that represents a table, column, value, expression, predicate, or sub-query.**

With the "`Remove Cast Text`" rewriting rule enabled, Alice tries the slow operations on Tableau again, but the performance is not improved. However, the query's rewriting path page (Figure 5) shows that the rule is correctly triggered, and the rewritten query is as expected. Therefore, Alice continues to optimize this query.

3.2 Formulating Rules by Providing Examples

After a deeper investigation into the query and some online search, Alice finds that a trigram index on the "`tweets`"."`text`" attribute in PostgreSQL supports wildcard filtering predicates such as `LIKE` and `ILIKE`. However, PostgreSQL fails to use this index because the wildcard predicate formulated by Tableau is `STRPOS( LOWER(`


Query Rewriting Path
Q_0
Remove Cast Text: CAST (<x> AS TEXT) <x>
Q_1
Remove Cast Text: CAST (<x> AS TEXT) <x>
Q_4

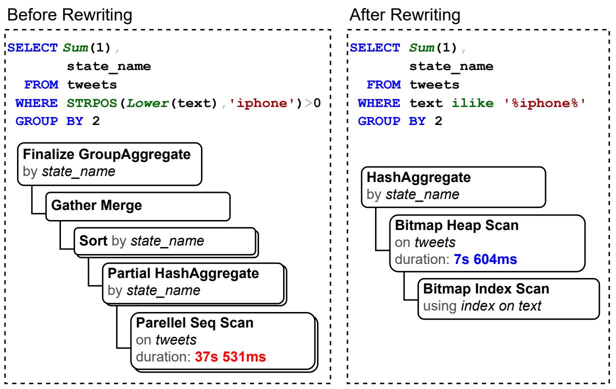
```
SELECT Sum(1) AS "cnt:tweets",
       tweets.state_name AS state_name
  FROM public.tweets AS tweets
 WHERE STRPOS(CAST(Lower(
                      CAST(CAST(tweets.text AS TEXT) AS TEXT)
                     ) AS TEXT),
              CAST('iphone' AS TEXT)) > 0
 GROUP BY 2
```

```
SELECT Sum(1) AS "cnt:tweets",
       tweets.state_name AS state_name
  FROM public.tweets AS tweets
 WHERE STRPOS(Lower(CAST(CAST(tweets.text AS TEXT) AS TEXT)),
              CAST('iphone' AS TEXT)) > 0
 GROUP BY 2
```

```
SELECT Sum(1) AS "cnt:tweets",
       tweets.state_name AS state_name
  FROM public.tweets AS tweets
 WHERE STRPOS(Lower(tweets.text), 'iphone') > 0
 GROUP BY 2
```

Figure 5: QueryBooster **shows the rewriting path of a query.**

"`tweets`"."`text`"), 'iphone') > 0, which is equivalent to "`tweets`"."`text`" `ILIKE '%iphone%'`. Alice identifies that replacing the `STRPOS()>0` predicate with the `ILIKE` predicate in the query can produce a much more efficient plan in PostgreSQL (as shown in Figure 6). Thus, Alice wants to introduce another rewriting rule that achieves this replacing logic. However, for this rewriting, Alice is not sure about how to manually input the rewriting rule, so she uses the QueryBooster's rule-suggestion feature by providing a rewriting example.

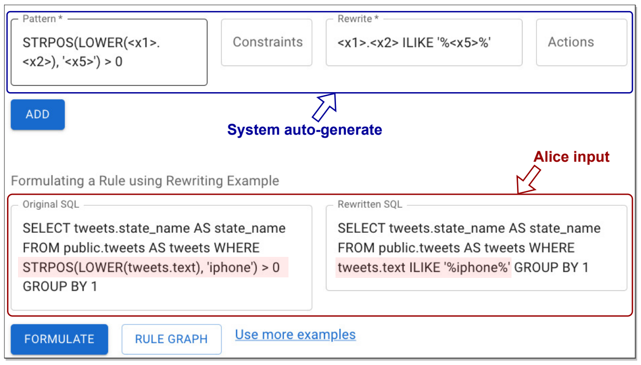


```
SELECT Sum(1),
       state_name
  FROM tweets
 WHERE STRPOS(Lower(text),'iphone')>0
 GROUP BY 2
```

```
SELECT Sum(1),
       state_name
  FROM tweets
 WHERE text ilike '%iphone%'
 GROUP BY 2
```

Figure 6: Queries and plans before and after the rewriting of replacing the `STRPOS()>0` **predicate with the** `ILIKE` **predicate.**

After copying and pasting the original query to both the original query and rewritten query text boxes on the *Rule Formulator* page, Alice modifies the rewritten query to her desired format and clicks the "Formulate" button. Next, QueryBooster automatically generalizes the example pair of queries into a rewriting rule that achieves the rewriting intention of Alice (as shown in Figure 7). Finally, Alice saves the new rewriting rule to the system with the name “`STRPOS To ILIKE`.”



```
STRPOS(LOWER(<x1>.<x2>), '<x5>') > 0
```

```
<x1>.<x2> ILIKE '%<x5>%'
```

```
SELECT tweets.state_name AS state_name
FROM public.tweets AS tweets WHERE
STRPOS(LOWER(tweets.text), 'iphone') > 0
GROUP BY 1
```

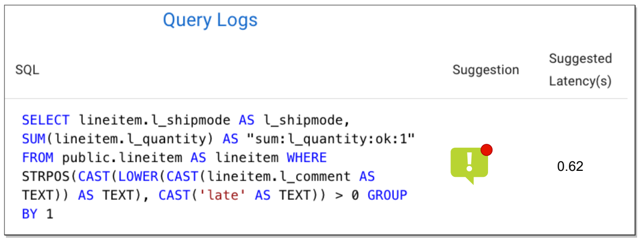
```
SELECT tweets.state_name AS state_name
FROM public.tweets AS tweets WHERE
tweets.text ILIKE '%iphone%' GROUP BY 1
```

Figure 7: QueryBooster **suggests a rule given a rewriting example.**

With the two rewriting rules enabled, the query performance is significantly improved. For example, the original query takes 37.5 seconds, and the rewritten query only takes 7.6 seconds.

3.3 Suggesting Useful Rules from Other Users

Suppose Bob is a database administrator (DBA) and supports a business team who wants to analyze the TPC-H [14] dataset using Tableau on top of PostgreSQL. For one of the analytical queries in Bob’s workload, QueryBooster identifies the potential of boosting the query performance by applying the two rewriting rules provided by Alice. QueryBooster automatically suggests the potential rewriting of the query to Bob, as shown in Figure 8.

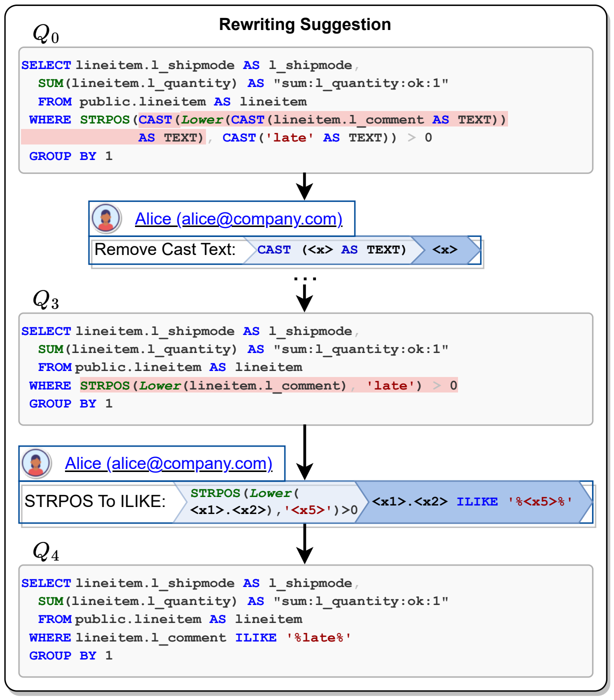


SQL	Suggestion	Suggested Latency(s)
`SELECT lineitem.l_shipmode AS l_shipmode, SUM(lineitem.l_quantity) AS "sum:l_quantity:ok:1" FROM public.lineitem AS lineitem WHERE STRPOS(CAST(LOWER(CAST(lineitem.l_comment AS TEXT)) AS TEXT), CAST('late' AS TEXT)) > 0 GROUP BY 1`	!	0.62

Figure 8: QueryBooster **suggests Bob a rewriting for a query.**

Bob inspects the rewritten query through the “Rewriting Suggestion” page of QueryBooster (Figure 9) and decides to enable these two rules for his application. Bob then returns to Tableau and observes a significant query performance improvement.

To this end, we demonstrate the powerful experience of using the QueryBooster service to rewrite application queries and share knowledge. The source code of the system will be available to the public.



```
SELECT lineitem.l_shipmode AS l_shipmode,
  SUM(lineitem.l_quantity) AS "sum:l_quantity:ok:1"
  FROM public.lineitem AS lineitem
 WHERE STRPOS(CAST(Lower(CAST(lineitem.l_comment AS TEXT))
              AS TEXT), CAST('late' AS TEXT)) > 0
 GROUP BY 1
```

```
CAST (<x> AS TEXT)    <x>
```

```
SELECT lineitem.l_shipmode AS l_shipmode,
  SUM(lineitem.l_quantity) AS "sum:l_quantity:ok:1"
  FROM public.lineitem AS lineitem
 WHERE STRPOS(Lower(lineitem.l_comment), 'late') > 0
 GROUP BY 1
```

```
STRPOS(Lower(
<x1>.<x2>),'<x5>')>0    <x1>.<x2> ILIKE '%<x5>%'
```

```
SELECT lineitem.l_shipmode AS l_shipmode,
  SUM(lineitem.l_quantity) AS "sum:l_quantity:ok:1"
  FROM public.lineitem AS lineitem
 WHERE lineitem.l_comment ILIKE '%late%'
 GROUP BY 1
```

Figure 9: QueryBooster **suggests a rewriting for Bob’s query by applying Alice’s two rules.**

REFERENCES

[1] ApexSQL: SQL execution plan viewing and analysis [n.d.]. https://www.apexsql.com/sql-tools-plan.aspx. last accessed: 2023-01-07.

[2] Qiushi Bai, Sadeem Alsudais, and Chen Li. 2022. Demo of VisBooster: Accelerating Tableau Live Mode Queries Up to 100 Times Faster. In *BigVis Workshop of EDBT 2022*, Vol. 3135. http://ceur-ws.org/Vol-3135/bigvis_short5.pdf

[3] Qiushi Bai, Sadeem Alsudais, Chen Li, and Shuang Zhao. 2023. Maliva: Using Machine Learning to Rewrite Visualization Queries Under Time Constraints. In *EDBT 2023*. 157–170. https://doi.org/10.48786/edbt.2023.13

[4] EverSQL: Automatic SQL Query Optimization for MySQL and PostgreSQL [n.d.]. https://www.eversql.com/. last accessed: 2023-01-07.

[5] Béatrice F. and Georges G. 1991. A Rule-Based Query Rewriter in an Extensible DBMS. In *ICDE 1991*. 248–256. https://doi.org/10.1109/ICDE.1991.131472

[6] Goetz Graefe. 1994. Volcano - An Extensible and Parallel Query Evaluation System. *IEEE Trans. Knowl. Data Eng.* 6, 1 (1994), 120–135.

[7] Hakan H., Balakrishna I., Chen L., and Sharad M. 2002. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD 2002*. ACM, 216–227. https://doi.org/10.1145/564691.564717

[8] Keebo: Data Learning and Warehouse Optimization [n.d.]. https://keebo.ai/. last accessed: 2023-01-07.

[9] Oracle Free Use Terms and Conditions [n.d.]. https://www.oracle.com/downloads/licenses/oracle-free-license.html. last accessed: 2023-01-07.

[10] Hamid P., Joseph H., and Waqar H. 1992. Extensible/Rule Based Query Rewrite Optimization in Starburst. In *SIGMOD 1992*. ACM Press, 39–48. https://doi.org/10.1145/130283.130294

[11] Q. Bai and S. Alsudais and C. Li 2023. QueryBooster: Improving SQL Performance Using Middleware Services for Human-Centered Query Rewriting. In VLDB 2023.

[12] V. Sreekanti, C. Wu, and et al. 2020. Cloudburst: Stateful Functions-as-a-Service. *Proc. VLDB 2020*. 13, 11 (2020), 2438–2452. http://www.vldb.org/pvldb/vol13/p2438-sreekanti.pdf

[13] Toad: Develop, analyze, and administer databases with Toad [n.d.]. https://www.toadworld.com/products. last accessed: 2023-01-07.

[14] TPC-H Website [n.d.]. http://www.tpc.org/tpch/. last accessed: 2023-01-07.

[15] A. Vogelsgesang, M. Haubenschild, and et al. 2018. Get Real: How Benchmarks Fail to Represent the Real World. In *Proc. of DBTest@SIGMOD 2018*. ACM, 1:1–1:6. https://doi.org/10.1145/3209950.3209952

[16] Zhaoguo Wang, Zhou Zhou, and et al. 2022. WeTune: Automatic Discovery and Verification of Query Rewrite Rules. In *Proc. of SIGMOD 2022*. ACM, 94–107. https://doi.org/10.1145/3514221.3526125

[17] X. Zhou, G. Li, C. Chai, and J. Feng. 2021. A Learned Query Rewrite System using Monte Carlo Tree Search. *Proc. VLDB 2021* 15, 1 (2021), 46–58. https://doi.org/10.14778/3485450.3485456