Unified SQL Query Middleware for Heterogeneous Databases

Huiran Zhang¹, Cheng Zhang¹, Rui Hu¹, Xi Liu¹, and Dongbo Dai*¹

¹School of Computer Engineering and Science, Shanghai University, Shanghai, 200444, China
*Corresponding author’s e-mail: dbdai@shu.edu.cn

Abstract. With the development of modern computer technology, the data which needs to be stored and processed by modern application systems becomes more and more varied. While traditional relational databases still dominate, NoSQL databases are developing rapidly and occupy a place in the entire industry. Therefore, more and more applications need to connect to multiple database systems at the same time. However, the different characteristics supported by different database systems and the differences in query syntax rules make multi-database application development very complicated. This paper talks about a unified query middleware system that supports federated analytical queries with standard SQL syntax for various data sources and can perform joins, aggregates and filter on data from different data sources in a single query.

1. Introduction
With the rapid development of computer technology, the data that needs to be stored and processed by modern application systems begins to show a large amount and diversity. Based on the characteristics of the data structure, the data can be divided into two types[1]. One is structured data with fixed model, and the corresponding storage solution is traditional relational database. The other type is unstructured data, with no predefined data model, and the corresponding storage solution is the emerging non-relational database. Previously, almost all applications chose relational databases as the database management system because of their technical completeness and stable performance. However, with the advent of the Big Data era, the large amount of unstructured data has led to a bottleneck in the performance of relational databases[2]. Therefore, NoSQL databases have become a more popular solution with their flexible data model, high scalability and Big Data capabilities[3]. Despite the rapid development of NoSQL databases, they were not created to replace relational databases. Both are used in different scenarios and play an indispensable role in their respective data storage fields.

Due to the large scale of modern application systems, it is often necessary to store and process data of different structures and sizes, which needs to be stored in different types of databases, so the application systems must be required to have the ability to connect to different databases[4]. However, this is not an easy task. Due to the different features supported by different database systems and the difference in query syntax rules, the development of multi-database application becomes very complicated[5]. Developers need to spend a lot of time and energy to learn and maintain multiple independent database interfaces, which leads to confusing structure, increasing difficulty and inefficiency.

Therefore, the research on unified query of heterogeneous databases has been gradually developed. In[6], the SOS (Save Our Systems) was proposed, which defined a common API for Redis, MongoDB, and HBase. SOS makes it easy to access through different NoSQL databases, but it cannot handle
SQL-based access well. Wang et al.[7] proposed a new method to access heterogeneous databases by using a unified access layer, but it did not optimize for complex queries. In existing business applications, the new database called F1 announced by Google Research is a hybrid database of SQL and NoSQL, which mainly combines the high scalability of BigTable and the availability and functionality of SQL databases[8]. However, since F1 is completely designed in accordance with Google's needs, it is not widely applicable to general scenarios. SkySQL[9] integrates relational database MariaDB and NoSQL database Cassandra, allowing MariaDB engine to access the data in Cassandra cluster, making Cassandra a table in MariaDB, and users can write data in this table. However, this solution also has disadvantages due to data update and migration issues.

The purpose of this paper is to implement a unified federated query middleware that allows users to perform federated analytical queries on various data sources using standard SQL syntax. In this way, a unified data processing paradigm for multiple data sources can be built, shielding the underlying physical storage and computing layer, and maximizing the efficiency of business processing data. At the same time, it can provide pluggable interface for developers to connect new data sources by themselves.

2. Architecture
The main architecture of the middleware proposed in this paper is shown in figure 1. It consists of three layers: syntax parsing layer, which is responsible for parsing, validation, optimization of SQL statements, splitting of mixed SQL queries and finally generating query plan[10]; computing engine layer, which is responsible for routing query plan to specific execution plan, and then interpreting it into executable code for specific storage or engine; data storage layer, which is responsible for data prepared extraction and storage.

![System architecture](image)

**Figure 1. System architecture.**

2.1. Syntax parsing layer
This layer receives SQL statements from the user interface and outputs the abstract syntax tree after lexical and syntactic analysis. For example, when the user inputs: SELECT id, name FROM users WHERE age > 20; first, the Lexer converts the character sequence into a word sequence, then the Parser parses the meaning of the combined statement according to the syntax rules, generating an abstract syntax tree to represent the syntactic structure of the program statement in the form of a tree data structure, as in figure 2.
Then, the generated abstract syntax tree needs to be verified to check whether it is legal, such as verifying the existence of SQL scheme, field, function, etc. and whether the SQL statement conforms to the specification, and generating the RelNode tree after completion. Finally, optimize the RelNode tree and split the mixed query to generate a query plan.

2.2. Computing engine layer

This layer receives the query plan generated from the upper layer, and resolves which databases the tables to query are in. If the query is from a single data source, then the underlying database can be queried and results returned via JDBC connection; if a federated query from heterogeneous databases is required, then the Spark data processing engine needs to be started, the execution plan interpreted into executable code for specific storage or engine, and then queried and results returned separately.

To elaborate federated query from heterogeneous databases with an example, consider the following query which exhibits federated join of a table named ‘log’ from Elasticsearch and another named ‘info’ from Hive. The specific processing flow is shown in figure 3.

```
SELECT * FROM
(SELECT * FROM index.log AS log WHERE log.content IS NOT NULL) AS es_log
INNER JOIN
(SELECT * FROM database.info AS info WHERE info.content IS NOT NULL) AS info
ON es_log.id = info.id;
```

![Figure 3. Query processing flow.](image-url)
2.3. Data storage layer
This layer is the actual physical storage of the application system data, and the user queries and processes the data from this layer through the middleware system proposed in this paper. When the user performs the query operation, the connection information of the database contained in this layer as well as the table and field information need to be collected into the metadatabase. In this way, the information about the tables can be obtained from the metadatabase when parsing SQL statements in the syntax parsing layer to generate query plans. The structure of the TABLES table in the metadatabase is designed as shown in table 1. The JSON structure of the data source collection is shown in figure 4.

| Field Name | Type     | Comment                                |
|------------|----------|----------------------------------------|
| Tbl_id     | INT      | Table id                               |
| Create_time| TEXT     | Table creation time                    |
| Db_id      | INT      | The id of the database to which the table belongs |
| Tbl_name   | VARCHAR(64) | Table name                          |

Figure 4. Sample JSON structure for MySQL.

3. Database operations and results
In the following sections we will query the underlying test dataset using SQL statements through the user interface. The SQL queries will fetch data from the databases mentioned in the architecture, i.e. MySQL, Elasticsearch and Hive. These queries are similar to the query schemes used in solving real-world problems. With the middleware system proposed in this paper, operations such as join, filter, aggregate and nested queries between databases can be performed to query the data.

3.1. Single data source query
Users can simply query the underlying data using standard SQL statements, without having to care about the specific storage engine, so that they can focus on business data processing. For example, when the data to be queried is stored in Elasticsearch, the user can simply use the following query statement.

```
SELECT firstname, age, email, city, balance FROM type2 WHERE age > 30 LIMIT 5;
```

The query process and results are shown in figure 5.

Figure 5. Single data source query example.
3.2. Inter-database join
To demonstrate inter-database join, we use following query to find out the basic information about the relevant users and their logs. In this query, an equijoin is performed between a type named info from Elasticsearch and a table named users in MySQL database.

```
SELECT u.username, u.sex, u.address, l.content
FROM users AS u, log AS l
WHERE u.username=l.username
LIMIT 5;
```

The query process and results are shown in figure 6.

![Figure 6. Inter-database join example.](image)

3.3. Aggregates
Aggregate functions such as SUM, AVG, COUNT, MIN, MAX are supported by the system. In the following example we try to find out the AVG of salary grouped by address. For this example, we use the combination of Elasticsearch and MySQL database.

```
SELECT l.address, AVG(u.salary) as avgSalary
FROM users AS u, log AS l
WHERE u.username=l.username
GROUP BY l.address;
```

The query process and results are shown in figure 7.

![Figure 7. Aggregates example.](image)

3.4. Nested queries
The nested queried are supported as shown below. In this example we try to find out the information of users whose salary is greater than 15000.

```
SELECT t.username, t.salary, m.address, m.content
FROM
(SELECT username, salary FROM users WHERE salary >= 15000) AS t,
(SELECT username, address, content FROM log WHERE content IS NOT NULL) AS m
WHERE t.username=m.username;
```
The query process and results are shown in figure 8.

Figure 8. Nested queries example.

4. Conclusion
This paper implements a unified SQL federal query middleware, which treats multiple existing heterogeneous databases as a whole and provides a unified user interface to the outside, hiding many details of the underlying databases. Developers only need to understand how to call the unified interface to realize query operations on multiple sources of data, so as to maximize the efficiency of business processing.

Considering the above data operations using SQL queries, it has been demonstrated that the developed unified query middleware system is capable of fetching data from multiple heterogeneous database systems. Moreover, with this system, database operations such as filtering, aggregates, joins and nested queries can be easily and seamlessly implemented across database systems.

In addition, there are many shortcomings in the subject, which need further research and improvement. There is a scope for analysis and research on the performance aspect of the proposed system, and a comparative study of individual database systems is needed.

References
[1] Bjeladinovic, S. (2018) A fresh approach for hybrid SQL/NoSQL database design based on data structuredness. Enterprise Information Systems, 12: 1202–1220.
[2] Ramya, M., Thirumahal, M.R. (2020) QUERY SYSTEM FOR MYSQL AND MONGODB. International Journal of Multidisciplinary Educational Research, 9: 115-120.
[3] Gupta, A., Tyagi, S., Panwar, N., et al. (2017) NoSQL Databases: Critical Analysis and Comparison. In: 2017 International Conference on Computing and Communication Technologies for Smart Nation. Gurgaon. pp. 293-299.
[4] Vathy-Fogarassy, Á., Hugyák, T. (2017) Uniform data access platform for SQL and NoSQL database systems. Information Systems, 69: 93-105.
[5] Ali, W., Shafique, M.U., Majeed, M.A., et al. (2019) Comparison between SQL and NoSQL databases and their relationship with big data analytics. Asian Journal of Research in Computer Science, 4: 1-10.
[6] Atzeni, P., Bugiotti, F., Rossi, L. (2012) Uniform access to non-relational database systems: the SOS platform. In: International Conference on Advanced Information Systems Engineering. Gdansk. pp. 160-174.
[7] Wang, X.F., Feng, R.H., Dong W., Zhu, X.Q, Wang, W.K. (2017) Unified Access Layer with PostgreSQL FDW for Heterogeneous Databases. In: 14th IFIP International Conference on Network and Parallel Computing. Hefei. pp. 131-135.

[8] Shute, J., Oancea, M., Ellner, S., et al. (2012) F1-the fault-tolerant distributed rdbms supporting google's ad business. In: International Conference on Management of Data. Arizona. pp. 777-778

[9] Daniel Bartholomew. (2012) Mariadb vs. mysql. http://rozero.webcindario.com/disciplinas/fbmg/abd3/MariaDB_vs_MySQL.pdf

[10] Begoli, E., Camacho-Rodríguez, J., Hyde, J., et al. (2018) Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In: Proceedings of the 2018 International Conference on Management of Data. Houston. pp. 221-230.