# Unified Access Layer with PostgreSQL FDW for Heterogeneous Databases

Xuefei Wang, Ruohang Feng, Wei Dong, Xiaoqian Zhu, Wenke Wang

# Unified Access Layer with PostgreSQL FDW for Heterogeneous Databases

Xuefei Wang[1], Ruohang Feng[2], Wei Dong[1],
Xiaoqian Zhu[1], and Wenke Wang[1]

[1] College of Computer, National University of Defense Technology,
Changsha, China
{wishfay,wdong,zhu_xiaoqian,wangwenke}@nudt.edu.cn
[2] Umeng+ Company of Alibaba Group, Beijing, China
ruohang.frh@alibaba-inc.com

**Abstract.** Large-scale application systems usually consist of various databases for different purposes. However, the increasing use of different databases, especially NoSQL databases, makes it increasingly challenging to use and maintain such systems. In this paper, we demonstrate a framework for designing a foreign data wrapper (FDW) for external data sources. We propose a novel method to access heterogeneous databases, including SQL and NoSQL databases, by using a unified access layer. This method was applied in some real business applications of Alibaba, in which we were able to do various operations on Redis, MongoDB, HBase, and MySQL by using a simple SQL statement. In addition, the information exchange and data migration between these databases can be done by using unified SQL statements. The experiments show that our method can maintain good database performance and provide users with a lot more convenience and efficiency.

**Keywords:** Unified access layer, Heterogeneous databases, Foreign data wrapper.

## 1 Introduction

In the big data era, a wide variety of non-relational NoSQL databases have been developed to meet massive data processing and analysis requirements. Different NoSQL databases are designed and built according to different feature orientations. In a practical project, in order to fully combine the advantages of using a variety of database capabilities, a large-scale system will often integrate these databases, including SQL and NoSQL, to support it [1]; this also brings some considerable challenges for deployment and maintenance. Typically, users must interact with these databases at the programming level with customized APIs. This reduces portability and requires system-specific codes. Some commercial companies have combined an SQL relational processor with a MapReduce query processor [2]. However, many of the most popular NoSQL databases, such as MongoDB and HBase, do not have SQL interfaces for their systems.

adfa, p. 1, 2011.

## 2 Related Work

Generally, data integration methods aim to integrate data arising from different SQL systems. However, NoSQL systems play an important role in many domains [3]. In [4], the Save Our System (SOS) 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. In [5], a relational layer supporting SQL queries and joins was added on top of Amazon SimpleDB. However, this applies only to SimpleDB, and other NoSQL databases are not applicable. ISO/IEC 9075-9:2008 defined the SQL/MED, or management of external data, an extension to the SQL standard [6]. During information retrieval from heterogeneous source systems, there are three main challenges: (i) resolving the semantic heterogeneity [7] of data including resolving the structural (data model) heterogeneity of data, (ii) bridging differences in data querying syntaxes, and (iii) data integration method may decrease the performance of the system.

## 3 Design FDW for External Databases

Based on the SQL/MED method, we used PostgreSQL FDW as a platform to manage external data sources. As Fig. 1 shows, to design an FDW, we first need to analyze the target API set of external databases; that is, we must take care of the syntactic heterogeneity problem. Second, we need to design different SQL syntax for different data storage methods that correspond to the semantic heterogeneity problem. Finally, we need to design condition pushdown to execute a complex query without much performance loss. Following this framework, we designed and applied FDWs for HBase, MongoDB, MySQL and Redis in the business system of Alibaba. As an example, we will give a detailed overview of FDW for HBase, called HBase_FDW[1].

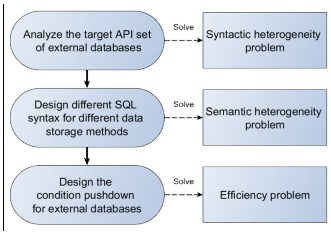


**Fig. 1.** Design framework of FDW

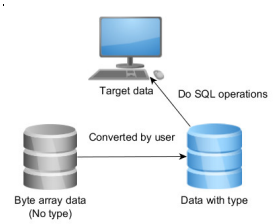


**Fig. 2.** The process of using byte array data

**Data Model and API.** HBase is a sparse long-term storage, multi-dimensional, sorted mapping table [8]. The HBase API consists of: get, scan, put, and delete.

**SQL Grammar.** FDWs manage foreign data as 'relations'. We establish an abstraction for HBase according to the physical data model of HBase with the following quintuple: (rowkey family, qualifier, timestamp, value) to index a unique value. Then

[1] This work can be accessed on GitHub: https://github.com/Vonng/hbase_fdw.

we set the type of these fields as byte array. For string users need to follow the encoding and decoding as Fig. 2 shows, and for other data types, they need to process data according to different serialization programs in the query statement.

**Condition Pushdown.** Pushdown means translating the 'where' clause in an SQL query statement to the corresponding external database API. By adding condition pushdown, we could minimize performance overheads in the system as much as possible. As Table. 1 shows, we have done all the basic operations of HBase. Any complex operations can be split into a combination of these basic operations.

**Table 1.** Condition pushdown for HBase

| Type of query | SQL statement | Original operation in HBase |
|---|---|---|
| Unconditional query | SELECT * FROM hbase.table | $\sigma$ = scan |
| Query with a row-key | SELECT * FROM hbase.table WHERE rowkey=k | $\sigma_{key=k}$ = get(k) |
| Comparison | SELECT * FROM hbase.table WHERE rowkey BETWEEN k1 AND k2; | $\sigma_{k1<key<k2}$ = scan(startRow = k1, stopRow = k2) |
| 'in' expression | SELECT * FROM hbase.table WHERE rowkey in (k1,k2,…, kn) | $\sigma_{key\ belongs\ to\ \{k1,k2,...,kn\}}$ = getMultiple([k1,k2,…,kn]) |
| Regex | SELECT * FROM hbase.table WHERE name REGEXP regexp | $\sigma_{key \sim regexp}$ = scan(filter = "RowFilter(='regexstring : regexp')") |
| List of columns | SELECT * FROM hbase.table WHERE column in (c1,c2,…, cn) | $\sigma_{column\ belongs\ to\ \{c1,c2,...,cn\}}$ = scan(column = [c1,c2,…,cn]) |

## 4 Unified Access Layer Via PostgreSQL

We carried out our research work on a business system of Alibaba, which needs to provide external queries of indicators. Each project team needs to write complex, repetitive code to handle business data queries. From project development testing to acceptance may take a group one or two months, which is very costly. So we built a unified access layer for public data, which greatly facilitated the R&D personnel who need to maintain the business logic; we called this new system EasyDB.

The architecture of EasyDB is shown in Fig. 3. Based on PostgreSQL, a unified access layer is built and makes all the other databases transparent for users. In this way, users can use SQL to operate all these databases conveniently.

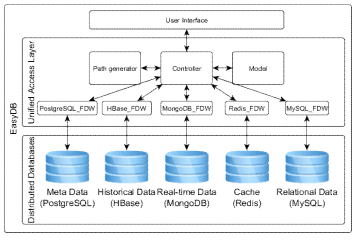


**Fig. 3.** Architecture of EasyDB

```
INSERT INTO mysql.app_daily_report
  SELECT
    a.rowkey,
    meta.app_id, meta.app_name, meta.app_owner,
    json_object_agg(date, active)  AS active_agg,
    json_object_agg(date, install) AS install_agg,
    json_object_agg(date, launch)  AS launch_agg
  FROM
    (SELECT
        rowkey,CURRENT_DATE AS date,
        active,install,launch
      FROM mongo.app_user_stat WHERE rowkey = 'test_app_id'
      UNION ALL
      SELECT
        rowkey,date,
        active,install,launch
      FROM hbase.app_user_stat
      WHERE rowkey = 'test_app_id' AND date BETWEEN '20160101' AND '20170101'
    ) a, LATERAL (SELECT * FROM psql.app_meta m WHERE a.rowkey = m.app_id) meta;
```

**Fig. 4**. A complex example in EasyDB

In EasyDB, after a request is made by the user, it will be parsed by the controller in EasyDB. By matching different schemas, the request could be responded by the corresponding data model and DBMS. An SQL statement will be translated into the API of different databases by FDWs. By condition pushdown, all these native operations will be executed in external databases instead of dealing with the data fetched from external data. As Fig. 4 shows, a complex query which involves many different databases can be done by only using SQL.

## 5 Evaluation

In this section, we test our system in two aspects: efficiency and actual use value.

First, to evaluate the performance of the developed system, we executed runtime tests on EasyDB. Experiments were performed using a cluster with 25 nodes: each node has an Intel(R) dual-core 2.5 GHz processor, 4 GB memory, and HDD storage.

**Dataset.** Tests were performed on five databases, including PostgreSQL, HBase, MongoDB, Redis and MySQL. The number of data rows of each query in this test is 1,000,000.

**Results.** The results of the performance tests are shown in Fig. 5. **Entire running time** began when the user sent the request to the EasyDB, and the measurement ended when the operation was completed in the EasyDB. **Data retrieval time** began when the user request was sent from the unified access layer to the data source and ended when the operation was completed in the EasyDB. **Native query time** presents the running time of native database queries. The time overhead caused by unified access layer can be measured by the difference between the entire running time and data retrieval time, which only takes up 6.5% at most and 4.0% on average. In fact, for large-scale distribution heterogeneous databases, the main cost involves the round trip time (RTT), and the FDW only takes up a small part of the overhead. Thus, our method does not have much impact on the efficiency of the system, which makes the entire running time very close to data retrieval time.

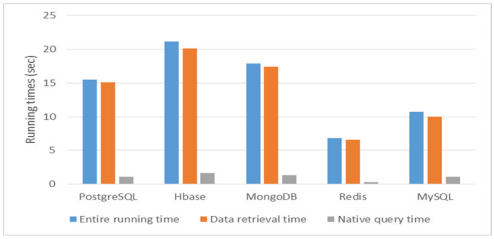


**Fig. 5.** Running times of Queries of different databases

Second, we conducted statistical studies on the development of multiple projects[2]. By using the unified access layer, a few weeks' work can be done in just a few days.

[2] Statistical information of projects is obtained from software management system of Umeng+ of Alibaba Group.

At the same time, users can readily use various databases, even without much information about the data model or API of these databases.

## 6 Conclusion

In this paper, based on a practical system, we designed a unified access layer for heterogeneous databases. We applied FDW technology to our production practice in order to solve practical problems, and we proposed HBase FDW to fill gaps in related fields. With only SQL, we can perform operations on all the databases in this system, and the experimental results show that the efficiency of our system is satisfactory.

**Acknowledgment.** The authors would like to thank the anonymous reviewers for their detailed and thoughtful feedback which improved the quality of this paper significantly. This work is funded by National Natural Science Foundation of China under Grant No. 61690203, No. 61379146 and No. 61272483.

## References

1. Sellami, R., Bhiri, S., Defude, B.: Supporting multi data stores applications in cloud environments. IEEE Trans. Serv. Comput. **9**(1), 59-71 (2016)
2. Yang, H. C., Dasdan, A., Hsiao, R. L., Parker, D. S.: Map-reduce-merge: simplified relational data processing on large clusters. In: ACM SIGMOD International Conference on Management of Data, pp. 1029-1040. ACM (2007)
3. Botta, A., Donato, W. D., Persico, V.: Integration of cloud computing and internet of things. Future Gen. Comput. Sys. **56**(C), 684-700 (2016)
4. Atzeni, P., Bugiotti, F., Rossi, L.: Uniform access to non-relational database systems: the SOS platform. In: International Conference on Advanced Information Systems Engineering, pp. 160-174 (2012)
5. Calil, A., Ronaldo, D. S. M.: SimpleSQL: a relational layer for SimpleDB. In: East European Conference on Advances in Databases and Information Systems, pp. 99-110 (2012)
6. Lawrence, R.: Integration and Virtualization of Relational SQL and NoSQL Systems Including MySQL and MongoDB. In: International Conference on Computational Science and Computational Intelligence, pp. 285-290. IEEE (2014)
7. Ceruti, M. G., Kamel, M. N.: Semantic Heterogeneity in Database and Data Dictionary Integration for Command and Control Systems. In: Department of Defense Database Colloquium, pp. 27 (1994)
8. George, L.: HBase - The Definitive Guide: Random Access to Your Planet-Size Data. DBLP. 2011