International Journal of Networked and Distributed Computing, Vol. 3, No. 2 (April 2015), 109-118

Integrated Usage of Heterogeneous Databases for Novice Users

Ayano Terakawa
Dept. of Information Science, Kyoto Institute of Technology, Matsugasaki Goshokaido-cho, Sakyo-ku
Kyoto, 606-8585, Japan

Teruhisa Hochin, Hiroki Nomiya
Dept. of Information Science, Kyoto Institute of Technology, Matsugasaki Goshokaido-cho, Sakyo-ku
Kyoto, 606-8585, Japan
E-mail: {hochin, nomiya}@kit.ac.jp

Abstract

This paper proposes the system enabling users to use heterogeneous databases in an integrated manner without any conversion and servers. In order to treat a variety of sources in a unified manner, this system is realized by using Java Database Connectivity (JDBC) in accessing databases on the user's computer. It also joins and/or projects them as required. The syntax identifying a kind of database or file is introduced. The system maintains data by using ArrayLists in Java. It is experimentally shown that there is no practical problem in the equijoin of three tables having 100,000 rows in heterogeneous databases.

Keywords: heterogeneous databases; JDBC; unified usage; table name specification

1. Introduction

With the spread of computer technology, users handling their data as electronic data on computers have increased. Archaeologists are included in such users. They also handle their data as electronic data on their computers.

In many cases, archaeologists determine where they store data in their own discretion. Therefore, the data may not be stored in the unified system. Data stored in this way become heterogeneous information sources. Moreover, data are stored in the different databases due to the change of database administrators even in a department. Each department might use various databases for each.

In order to integrally handle these data, a method of converting the data stored in a database to the data in another database is employed. However, data conversion is a time-consuming task. The data conversion is also cumbersome in maintaining consistency between the copy and the original data when the original data are frequently changed.

```
1  Class.forName("com.sqlite.JDBC").newInstance();
2  Connection con = DriverManager.getConnection("jdbc:sqlite:/Users/SQLite/school.db");
3  Statement st = con.createStatement();
4  ResultSet rs = st.executeQuery("SELECT * FROM student");
```

Fig. 1. An example of database connection through JDBC.

The system based on mediators and wrappers[1] can integrate heterogeneous information sources without data conversion. Wrappers provide access to heterogeneous information sources by converting application queries into source-specific commands or queries. Mediators are used to integrate heterogeneous information sources. Mediators integrate and refine the information provided by wrappers. This method is premised on the heterogeneous information sources distributed across a network. For this system, heterogeneous information sources are required to be on server computers. It is, however, often difficult for archaeologists to make their computers servers. Moreover, they may want to handle heterogeneous information sources in a single computer.

Management of distributed and heterogeneous databases has been studied.[2,3] Many types of heterogeneities including heterogeneities due to the differences in DBMSs and semantic heterogeneity have been treated. General solution is the mapping from local schemas to global one. In these studies, large-scale systems are assumed. A database is managed on a server. Several databases on several servers can be used in the integrated manner. The methods proposed in the literatures seem to be too heavy-weight for a small-scale system to adopt. It is considered that the light-weight system is good for a small-scale system. The meanings of the light-weight system include the server-less system. Integration of heterogeneous databases without any servers is required for the small-scale system.

Wang et al. allow users to handle various information sources without copying user's data to a server computer.[4] By making connections to MySQL, PostgreSQL, SQLite, an Excel file, and a CSV file through Java Database Connectivity (JDBC), databases and files can be used in a unified manner. Although heterogeneous information sources are dealt with, their simultaneous treatment is not considered.

This paper proposes the system enabling users to use heterogeneous information sources unifiedly and simultaneously without any data conversion. The proposed system uses JDBC to obtain data from databases and files. It also joins and/or projects them as required. The syntax identifying a kind of database or file is introduced. Table names as well as kinds of databases are required to be put to column names in order to uniquely identify them during query processing. This system maintains data by using ArrayLists in Java. It is experimentally shown that there is no problem in the equijoin of three tables with 100,000 rows in heterogeneous databases.

The remaining of the paper is as follows: Section 2 describes the tools used in the proposed system. Section 3 describes a specific design of the system. Section 4 describes implementation. Section 5 shows an example of the execution of the system. Section 6 experimentally evaluates the system. Lastly, Section 7 gives concluding remarks.

2. Preliminary

2.1. *Java Database Connectivity*

Java Database Connectivity (JDBC)[5] is an API for the connection of a relational database and the program in Java. It has standardized the ability to connect with relational database management system by using the SQL language in Java. In order to use JDBC, it is necessary to prepare a JDBC driver for each database management system. By using JDBC, schema

information and the data stored in a database can easily be obtained.

An example of the usage of JDBC is shown in Fig. 1. This is an example for SQLite. The driver class is loaded at Line 1. A database is connected to at Line 2. An instance object for a statement is created at Line 3. A query is executed and the result is stored in an instance object for a Resultset. We can use various

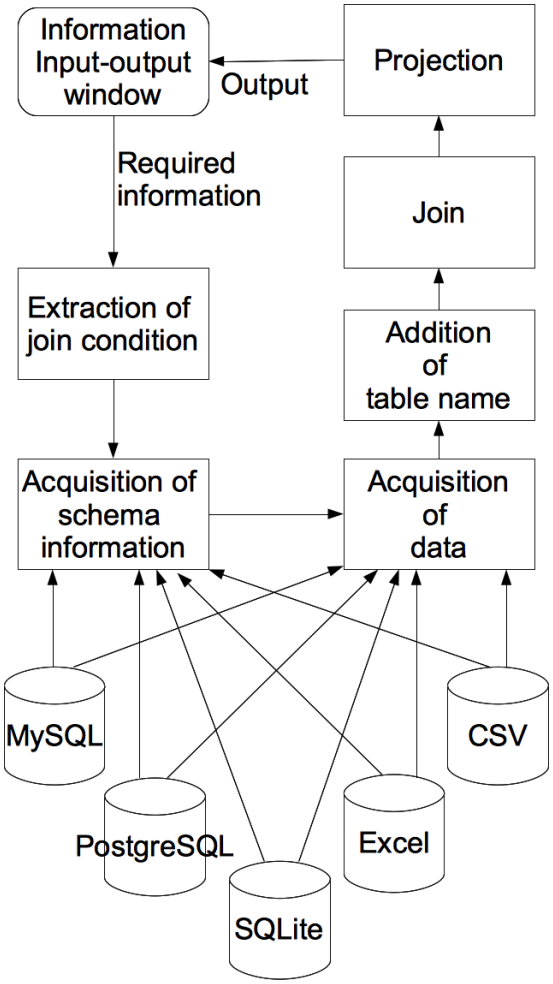

Fig. 2. Flow of join process.

database systems or files only by changing Line 1 and Line 2.

When data are obtained from an Excel file, Apache POI[6] is needed because the JDBC driver for Excel needs it.

2.2. *Schema information*

We get schema information to confirm whether the required table and the columns exist. In case of MySQL, PostgreSQL, and SQLite, we can get the schema information by sending a query using JDBC. If Apache POI is available, we can get the schema information by using Apache POI in Excel. In case of CSV files, we must read the CSV files and get the column names from them.

3. Design

In this system, a user enters the required information through the special window called the "information input-output window." The system extracts the join condition from the information input, and makes a connection to each database. Next, this system obtains the required table and joins them as needed. Finally, the join result is displayed in the information input-output window. Fig. 2 shows the flow of the process of join, which is the most complex process.

3.1. *Information input-output window*

Users specify the information for connecting databases or files. They get the join result through this information input-output window. This window has the following components:

(i) Input fields of the required informations for connecting to databases, and files.
(ii) Input field of projection column names.
(iii) Input field of the join condition.
(iv) Output field of execution result.

By providing (i) to (iv), we can do everything in a single window.

3.2. *Join condition*

The join condition is the condition for the join operation. It is the form of "`<DBKind>delimiter<Table name> <Alias name>delimiter<Column name>`", where <DBKind> is a kind of a database or file, and is one of "MySQL," "PostgreSQL," "SQLite,"

```
MySQL:student:sno = PostgreSQL:student:sno AND PostgreSQL:student:sno = SQLite:student:sno
```

Fig. 3. An example of the join condition.

"Excel," or "CSV." <Table name> is a sheet name in an Excel file. It is a file name in a CSV file. The delimiter is ":". The character ":" cannot be included in the alias name, which is optional. Join conditions are connected with "AND".

An example of the specification of the join condition is shown in Fig. 3. In this example, the table *student* in a MySQL database, the table *student* in a PostgreSQL database, and the table *student* in an SQLite database are joined.

Comparison operators "<=", ">=", "<", ">", "=", "!=", "NOT LIKE", and "LIKE" can be used with numbers or strings in MySQL, PostgreSQL, and SQLite. Only equijoin of columns of each other is supported for Excel and CSV.

3.3. *Use of schema information*

This system uses schema information as follows:

(i) Confirmation of the existence of the column: Before getting the data, the system checks whether the required table and the columns exist. If a table or column does not exist, the system depicts which database or file's table or column is missing.

(ii) Add a kind of database or file, and a table name (and an alias name): Since a table name is not given to the result of the query from each database management system, the system has to put a table name, and a kind of database or file to the query result in order to identify them.

3.4. *Management of data*

This system gets all of data as String data type, and manages them in an ArrayList. Also, the data newly created by join or projection are stored in the order of acquiring the data. Since the number of rows of the result of join and projection cannot be predicted, we use an ArrayList for a variable number of elements. The data structure is shown in Fig. 4. There are (n+1) tables in Fig.4. The table "Table 0" has (m0+1) rows. The first row consists of (c0+1) columns.

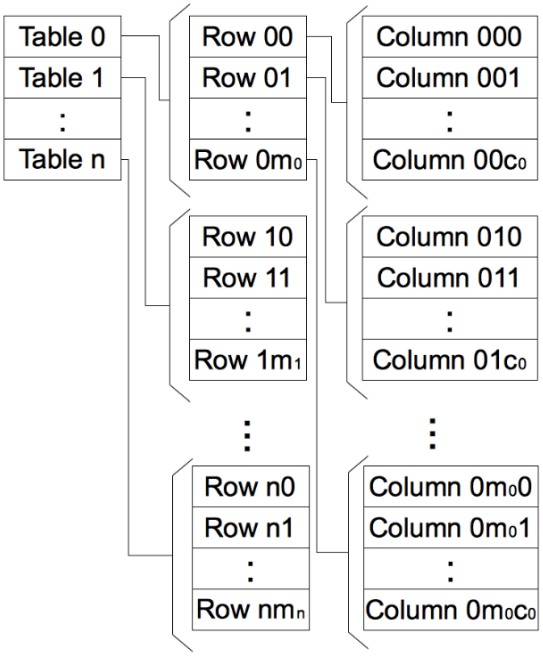

Fig. 4. Data structure.

4. Implementation

4.1. *Extraction of join condition*

From the information input-output window, the system gets the join condition. First, the system partitions the join conditions and creates lists of required tables and required columns. Next, the system creates the query based on the list of tables. The format of the query depends on the kinds of database as follows:

(i) MySQL, PostgreSQL, and SQLite: A query "`SELECT * FROM <From List> WHERE <Condition>`" is created, where <From List> is a list of one or more <Table name> separated by commas.[7]

(ii) Excel and CSV: A query "`SELECT * FROM <Table name>`" is created per table.

4.2. *Join Operation*

Sort merge join[8] and quick sort are used in this system. If rows are sorted, the system does not sort them again.

4.3. *Projection*

The column name projected is obtained from the information input-output window. In the same manner as in the join condition, the column name must be the form of "`<DBKind>:<Table name> <Alias name>:<Column name>`." When the user wants to project more than one column, columns are separated by commas. If blank space or "*" is specified, the system obtains a full set of results.

The information input-output window has a check box to allow the user to select removing duplicates. If this check box is not checked, duplication is not removed and the system outputs a full set of results. If it is checked, the system outputs the projection result without duplication.

5. Execution example

When the system is started, the information input-output window appears. An example of specification is shown in Fig. 5. The information needed to connect to the databases is specified in the text boxes on the left side of the window. In the SELECT section, a column name is specified. In the WHERE section, join condition is specified for MySQL, PostgreSQL, SQLite, Excel, and CSV file.

Next, pressing the Join button invokes the join process. The results appear in the text area of the right bottom of the screen as shown in Fig. 6. The projected table information is displayed under the last binding result. Since the scroll bar is on it, it is possible to see all of the results even if join result is long.

Using a list (JComboBox) under the Join button, the user selects the join process and can see the process selected.

6. Performance evaluation

In order to clarify the performance of the constructed system, we evaluate the performance on the actual machine.

6.1. *Experimental method*

The data acquisition time and the join operation one are measured three times on a personal computer (2.7GHz Intel Corei5, 8GB memory). The results are evaluated in the average. Unit of time is milliseconds.

The tables used in the experiment are as follows:

(i) The numbers of rows are 100, 1,000, 10,000 and 100,000.
(ii) The table has the columns of *id*, *name*, and *price*.
(iii) The column *id* is a random integer ranging from 0 to the number of rows. For example, in the case of 1,000 rows, it is a random integer of 0 to 1,000.
(iv) The column *name* is a random string. Its length is 2.
(v) The column *price* is a random two-digit integer.

All of tables are not manually indexed. Some database management systems may automatically create the primary index to a table. In this case, such an index may be used in the retrieval. Tables are generated by program.

- Experiment1: The data acquisition time of the system is measured.
- Experiment2: Two to five tables, each of which has the same number of rows, are joined. Joining two tables is performed using a table in MySQL and in PostgreSQL. In the case of three tables, the tables in MySQL, PostgreSQL, and SQLite are used. In the case of four, a CSV file is used in addition to the case three. In the case of five, an Excel file is used in addition to the case four. The join order is

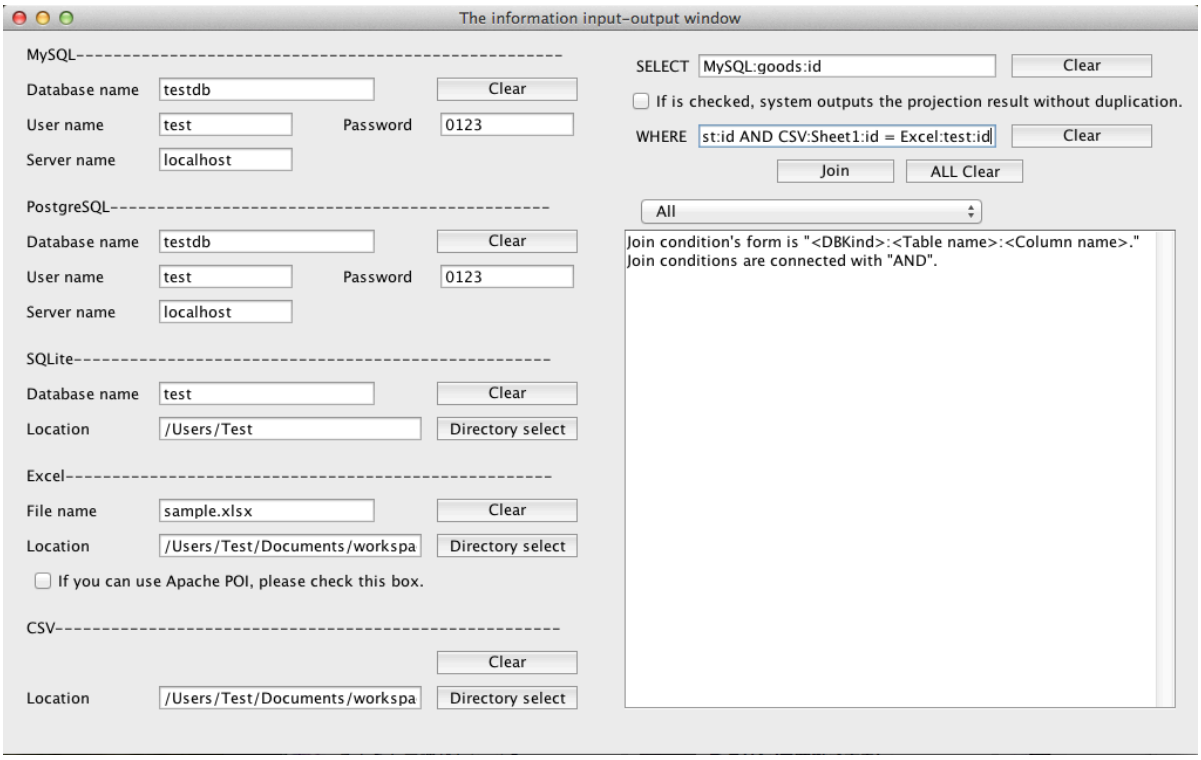

Fig. 5. An example of specification of input information.

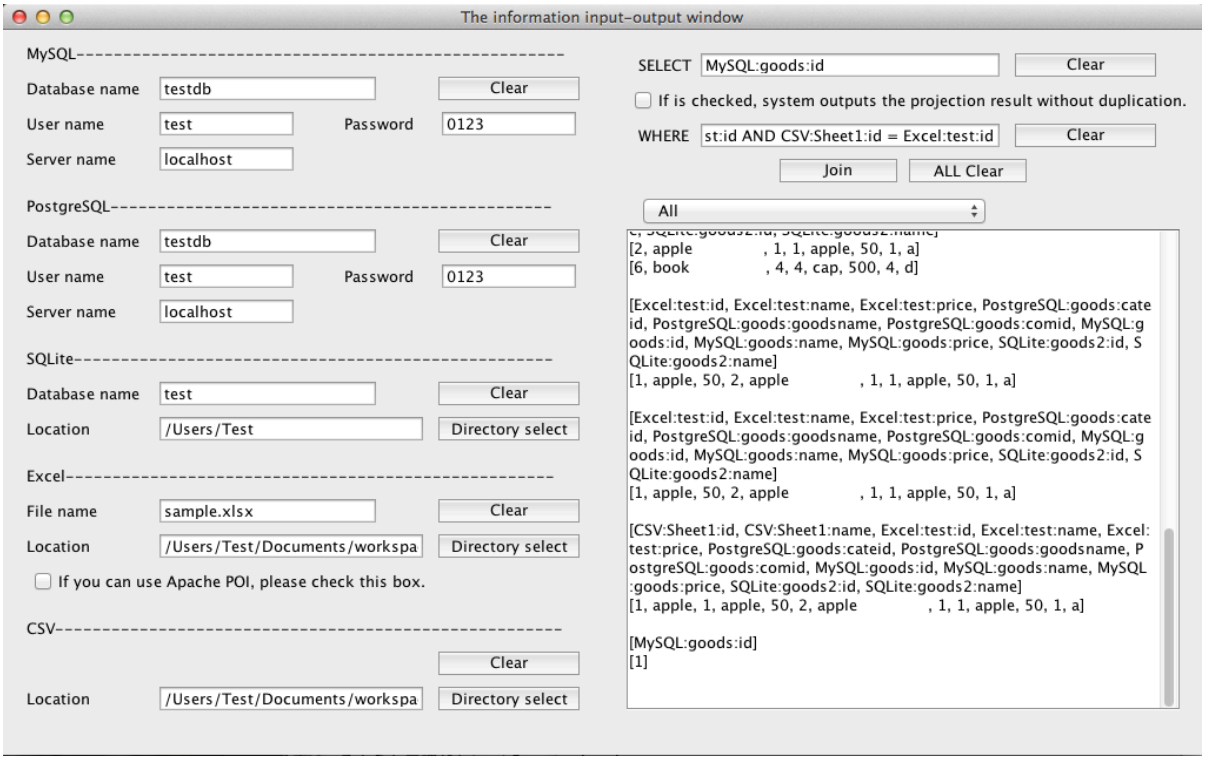

Fig. 6. Result of the join process and projection.

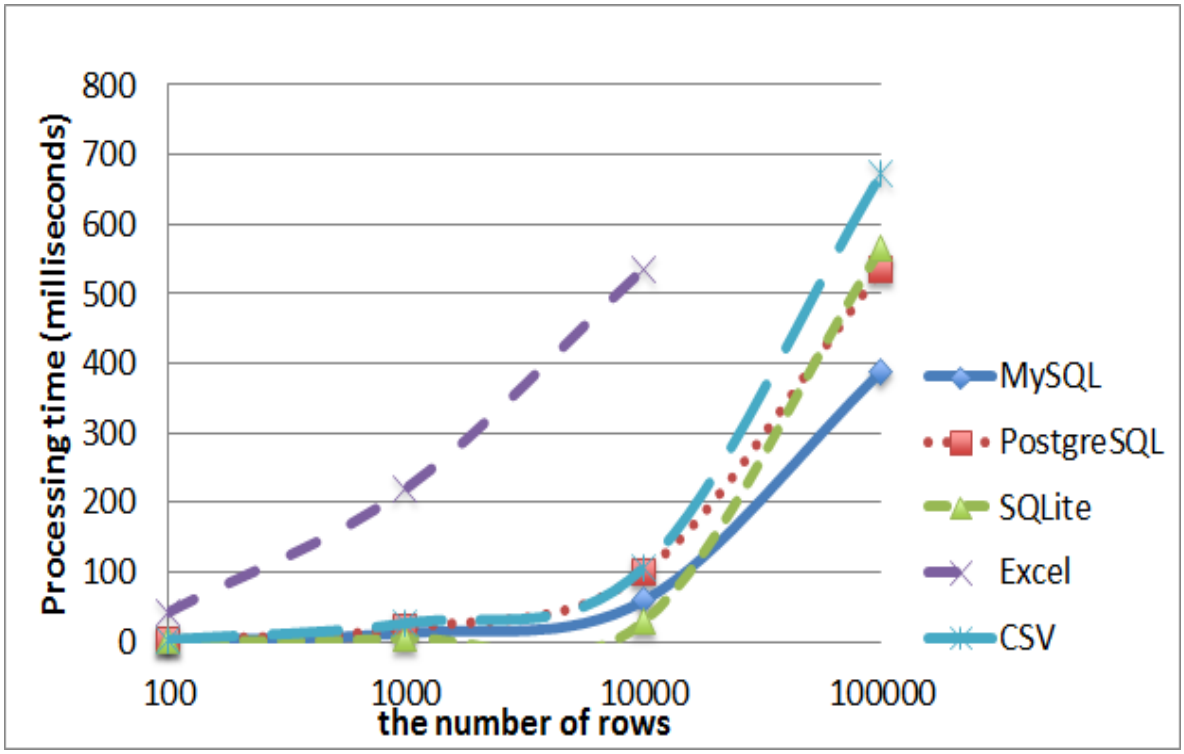

Fig. 7. Data acquisition time.

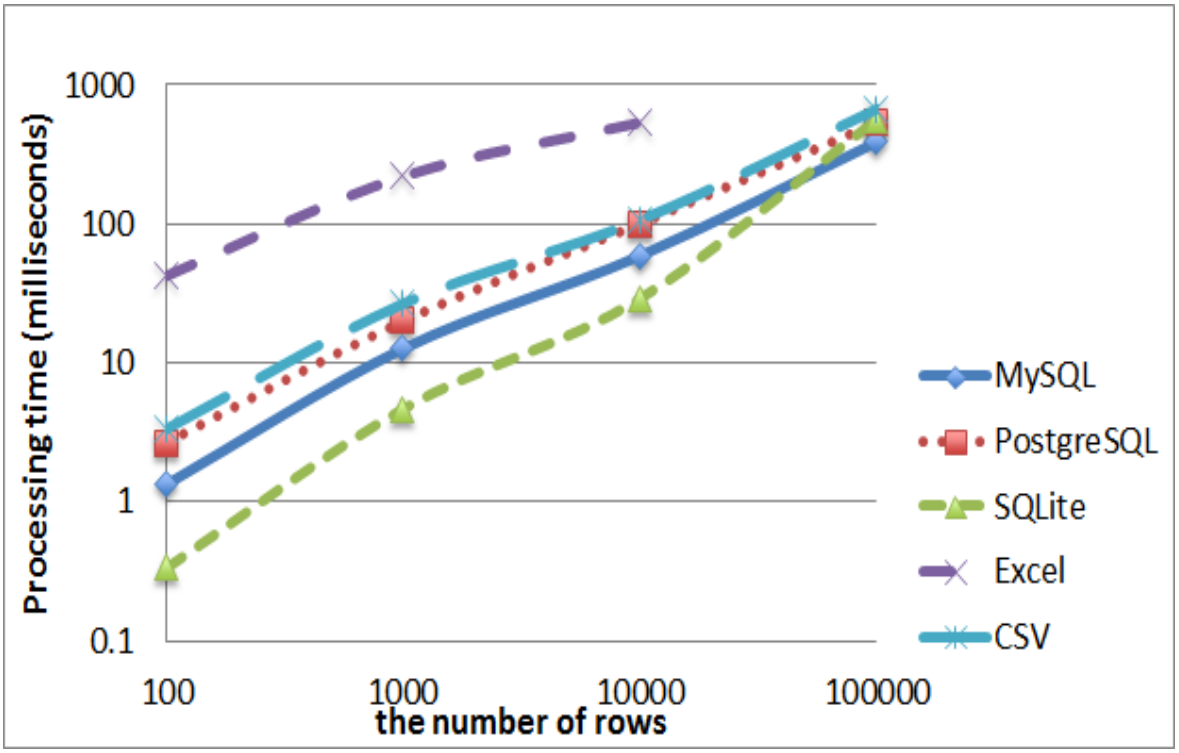

Fig. 8. Data acquisition time (logarithmic graph).

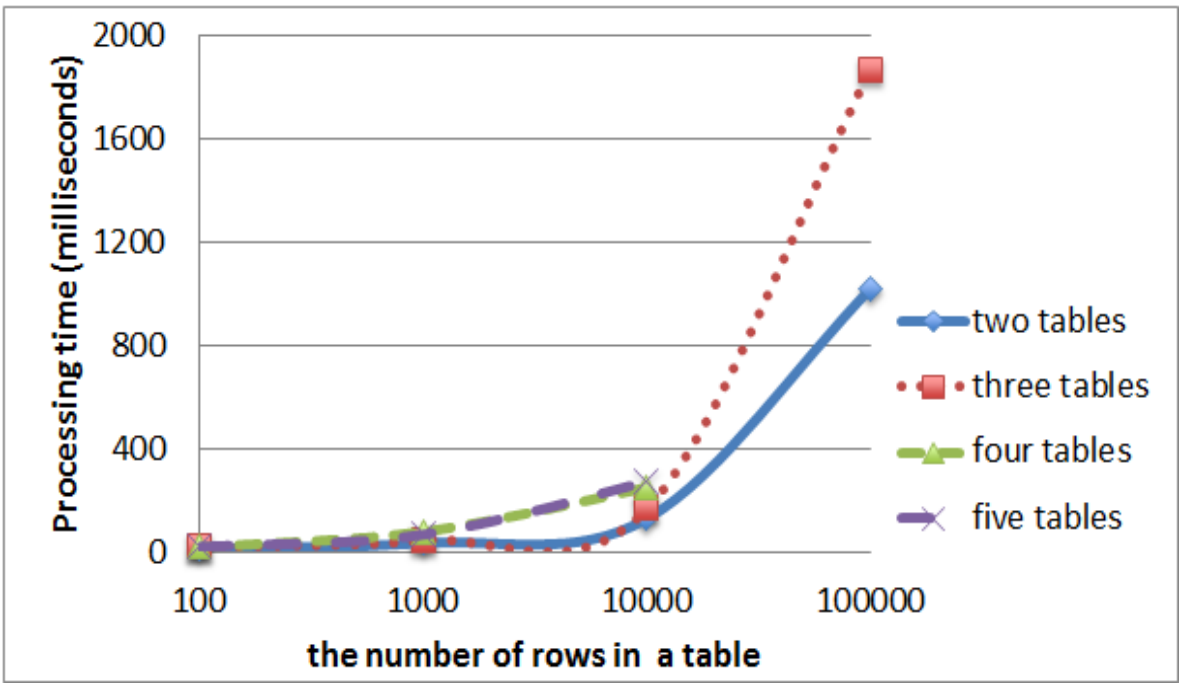

Fig. 9. Time of multi-table join.

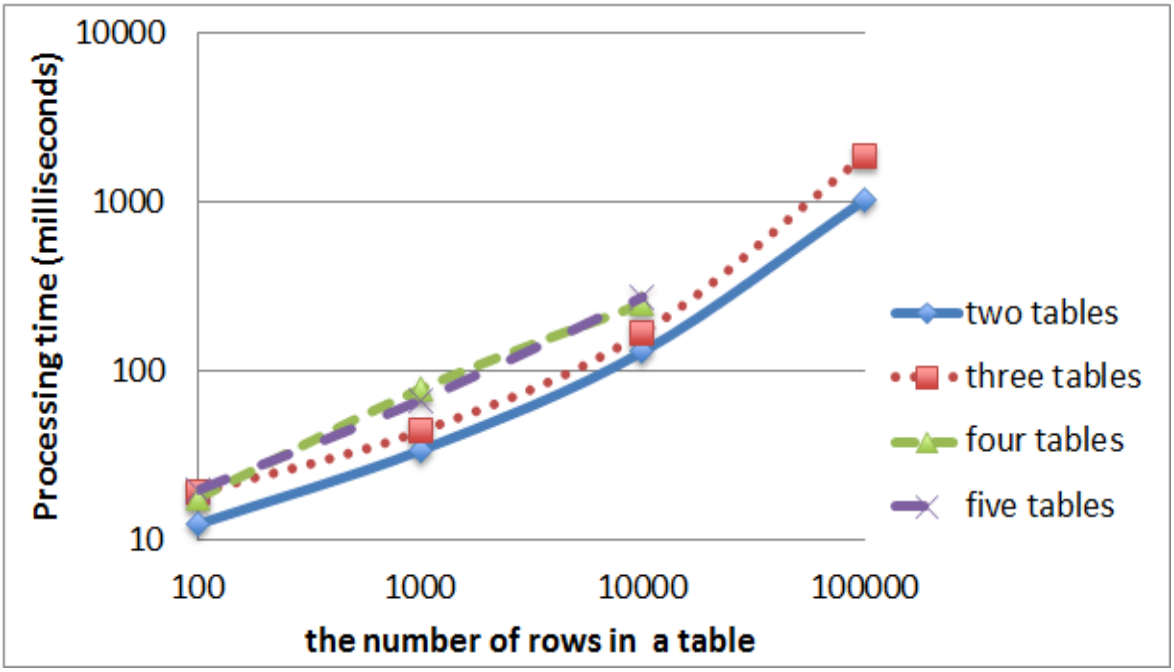

Fig. 10. Time of multi-table join (logarithmic graph).

the one described above from MySQL to an Excel file. Tables are equally joined in the column *id*.

6.2. *Experimental Result*

- Experiment1: Fig. 7 and Fig. 8 show the times of obtaining data from tables. Fig. 8 is a logarithmic graph of Fig. 7. We could not measure the acquisition time of the 100,000 rows for an Excel file because an Excel file (.xls) cannot contain 65,537 or more rows, and because of short of memory. A lot of time is needed to get data for an Excel file, while others take almost equal time shorter than an Excel file.
- Experiment2: Fig. 9 and Fig. 10 show the times of multi-table join. Fig. 10 is a logarithmic graph of Fig. 9. Joining four and five tables of 100,000 rows cannot be measured due to short of memory. The time does not depend on any number of tables up to 10,000 rows, and is not too much. In addition, the processing time increases according to the number of rows in the table over 10,000 rows.

6.3. *Considerations*

- Data acquisition time: It is considered that the acquisition time depends on the kind of database and file because the same algorithm is used to any database and file. It is also considered that there are no practical problem since all kinds of databases other than Excel can get the data from the table of 100,000 rows used in the experiment in less than one second. A lot of time is required to get the data for an Excel file. The Excel driver may affect the performance.
- Time of multi-table join: From Fig. 9 and Fig. 10, we can see that the processing times are unaffected by the number of tables with up to 10,000 rows. Each joining time is almost the same. In 100,000 rows, according to the number of tables, the processing time gets longer. However, it seems there is no practical problem because three tables can be joined in about two seconds.

7. Concluding remarks

In this paper, the data of heterogeneous information sources are made available without any conversion in the integrated manner. We realized the join of tables of heterogeneous information sources. MySQL, PostgreSQL, and SQLite databases as well as Excel and CSV files are connected to at the same time by using JDBC. We have implemented a system capable of performing the join and the project operations of tables in heterogeneous databases. This system gets the schema information of the table, and manages the schema and table information by using ArrayLists. The join algorithm used is the sort merge join. We evaluated retrieval performance on a real machine. It was shown that in 100,000 rows, three tables can be joined in about two seconds. It is considered that there is no practical problem to use this system.

The system, however, does not support the joins besides equijoin. Supporting non-equijoin, handling data of more than 100,000 rows, and implementing a rich user interface are in future work. The memory usage is important for processing the join operation. We are considering the order of join for the purpose of reducing the memory usage. The current implementation reads all of tuples from a database at a time. This wastes memory. Reading a tuple at a time may improve memory usage. Improving memory usage is also in future work. Evaluation of the usability and the performance by real users are also in future work.

References

1. H. Garcia-Molina, Y.Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V Vassalos and J. Widom, *The TSIMMIS Approach to Mediation: Data Models and Languages*, Journal of Intelligent Information Systems, 8 (2), (1997), pp. 117-132.

2. W. Sujansky, *Heterogeneous Database Integration in Biomedicine*, Journal of Biomedical Informatics, 34 (4), (2001), pp. 285-298
3. A. P. Sheth, Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases, ACM Computing Surveys, 22 (3), (1990), pp. 183–236.
4. X. Wang, T. Hochin and H Nomiya, *Feasibility of Unified Usage of Heterogeneous Databases Storing Private Information*, in Proc. IIAI Advanced Applied Informatics 2013 (IIAI AAI 2013), (2013), pp.337–342.
5. JDBC, http://ja.wikipedia.org/wiki/JDBC, (2014/01/25).
6. Apache POI, http://poi.apache.org/, (2014/05/04).
7. H. Garcia-Molina, J. Ullman and J. Widom, *Database Systems The Complete Book*, (Pearson Education, US, 2001), pp.787-791.
8. D. K. Shin and A. C. Meltzer, *A New Join Algorithm*, ACM SIGMOD Record, 23 (4), (1994), pp. 13-20.