Industrial Paper

Evaluation of Databases for Enterprise Systems Dealing with Images

Tsukasa Kudo†, and Yuki Furukawa†

†Faculty of Informatics, Shizuoka Institute of Science and Technology, Japan
kudo.tsukasa@sist.ac.jp

Abstract -

Nowadays, since various sensors such as surveillance cameras, wearable devices are used, various large amount of data became to be stored in the databases. To manipulate such a data, NoSQL databases have been put to practical use. Especially, MongoDB provides GridFS interface, and it was shown that the performance of such a data manipulation exceeded MySQL which is the conventional relational database. Here, to apply MongoDB to the enterprise systems, it is noted that there is the problem that the join operation is not provided. However, as for the business systems dealing with a large amount of data that takes time, it is expected that the effect of using MongoDB exceeds this problem. In this study, we conduct the comparative evaluation between MongoDB and MySQL for the actual production management system, which is a type of enterprise system. As a result, we show MongoDB is superior to MySQL in the case to manipulate a large amount of data even if the join operations are performed. Furthermore, we show it is possible to construct the configuration that takes each advantage of both by using program language such as Java. In this configuration, MongoDB manipulates a large amount of data; MySQL manipulates the other data including the join operation.

Keywords: MongoDB, database, GridFS, join operation, production management system

1 INTRODUCTION

Nowadays, various devices are spreading rapidly, such as smartphones, surveillance cameras, and various wearable devices. As a result, it is becoming possible to enter various data efficiently into the systems using such an inexpensive entry device [5], [7]. Therefore, even as for the enterprise systems, it is expected that the system can be operated more efficiently by utilizing not only conventional character and numeric data but also various kinds of sensing data such as images and videos.

In order to store and manipulate such a data, various kinds of NoSQL databases have been proposed and put to practical use [15]. Among them, there is MongoDB [3] which is a kind of the document-oriented NoSQL database: it stores the data as documents of semi-structured data model expressed by JSON; particularly, it equips GridFS interface to treat the enormous data efficiently [14]. Here, these documents correspond to the records of the relational database (RDB). And, as for a large amount of data, we had also confirmed that its performance of MongoDB had been superior to MySQL, especially in the insertion [11]. However, to apply MongoDB to the enterprise systems, it is noted that there is the problem that the join operation is not provided.

So, in order to expand its application area in the enterprise systems, it is necessary to solve the problem about the above-mentioned join operation. Here, since the large amount of data manipulation performance of MongoDB is superior to MySQL as above-mentioned, it is expected that the throughput in the entire system can be enhanced by using it, even if the performance deteriorates in the join operation. However, we could not find the study that has evaluated such a performance on the premise of the actual enterprise systems.

In this study, we explore the join operation from the view of MongoDB application. Its aims are as followings. First, we show the application field where MongoDB is superior to RDB, though the join operation is used. Second, we show the program structure, by which the join operation and a large amount of data manipulation can be performed efficiently.

So, our goal in this study is to evaluate such a performance and to show the requirements to apply MongoDB to the enterprise systems. As a target system of this evaluation, we used an actual production management system, which is a type of enterprise system. It was mainly configured to execute SQL statements directly by using batch files, though the complex processing was implemented by using the stored procedure [12]. And, we attempt comparative evaluations between MySQL and MongoDB in two cases.

In the first case, we migrate the above-mentioned SQL statements directly to MongoDB's statements and show the comparative evaluation results on this migration productivity and performance. In addition, we show that the manipulation including the numerous large amount of data deteriorates the performance of MongoDB in this case. In the second case, we use the programing language Java and its database drivers and show the comparative evaluation results of the performance. And, we show that the above-mentioned deterioration can be solved and MongoDB's performance to manipulate a large amount of data is superior to MySQL in this case. Furthermore, we show it is possible to construct the configuration that takes each advantage of both databases: MongoDB manipulates a large amount of data; MySQL manipulates the other data including the join operation.

The remainder of this paper is organized as follows. Section 2 shows the related work, and Section 3 shows the abstract of the target system to clarify the precondition. Section 4 shows the correspondence of data manipulation between MySQL statements used in the system and Mongo shells, which is the data manipulation statements of MongoDB, and we show the implementation of the system by using Mongo

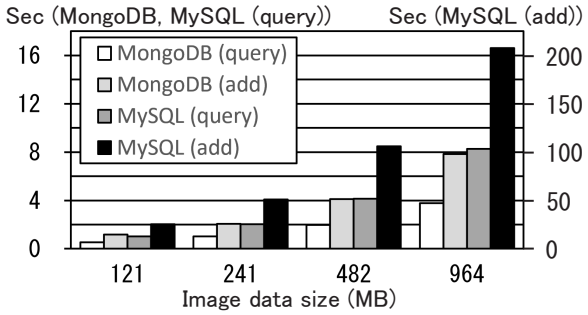

Figure 1: Evaluation result of performance comparison

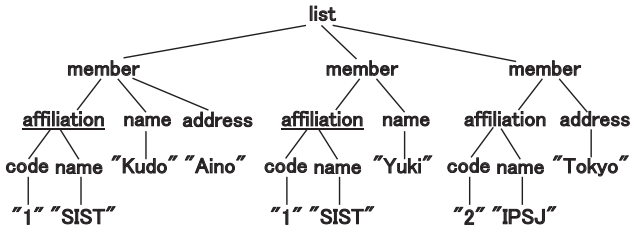

Figure 2: MongoDB structure with embedded document

shells. In Section 5, we show the results of comparative evaluations between MongoDB and MySQL in the case of direct migration from SQL statements to Mongo shells. Similarly, Section 6 shows the comparative performance evaluations in the case of utilizing programing language Java. And, we discuss these results in Section 7. Lastly, Section 8 concludes this paper.

2 RELATED WORK

The comparative evaluations between RDB and NoSQL databases have been performed on such as the data modeling, data manipulation, and performance.

Firstly, for a large amount of data manipulation, MySQL and Oracle, which are the relational database management system (RDBMS), provide BLOB data type [13]; MongoDB, which is a kind of document-oriented NoSQL database, provides GridFS interface [10]. And, it was shown that MongoDB excelled about the performance to manipulate such a data. Moreover, we have already conducted the comparative evaluations on the performance between MongoDB and MySQL for the video data, and we found MongoDB was much more efficient as shown in Fig. 1. Especially, as for the insertion, in this case, MongoDB was 25 times faster than MySQL [11].

Incidentally, there is a method of saving a large amount of data by using the file system without using the databases. However, as for this method, it is pointed out that there are some problems: the authority to access the data cannot be managed; it is difficult to perform the automatic backup such as the replication [18].

Similarly, the comparative performance evaluations of the CRUD operations (insertion, query, update, and deletion) were performed for the data that had been handled by RDB traditionally, such as text and numerical data. As a result, it was shown that the performance of MongoDB is superior to RDB [4], [6].

On the other hand, it has been pointed out that there were two problems to apply it to the enterprise systems: first, it does not maintain the ACID properties of the transaction; second, it does not equip the join operation for the plural collections which correspond to the tables of RDB [16].

As for the first problem, namely the transaction, we had shown a solution in our previous study. Here, the transaction of MongoDB can maintain the ACID properties only on the manipulation of an individual document, which corresponds to the table in RDB. So, firstly, we developed the transaction processing method, by which the ACID properties could be maintained even on the manipulation of plural documents [8]. Next, to evaluate this method, we applied it to the prototype of the actual production management system. As a result, we showed that this method could maintain the ACID properties even on the plural documents manipulation of a large amount of data in addition to the conventional character and numeric data [9].

Furthermore, we showed the application field where MongoDB's large amount of data manipulation is effective through these studies. Concretely, we had been advancing its application study for the production management system utilizing images and videos in order to improve the efficiency of the inventory management work. Here, conventionally, the inventory quantity of various kinds of parts must be counted, and it makes the workload higher. So, we had conceived the method, in which the inventory manager judges visually whether there had been the necessary inventory quantity by using the images and videos [9]. As a result, the manager could perform this business at the office based on the inventory plan calculated beforehand, instead of counting the inventory at the field.

Here, the join operation is not provided by MongoDB. Instead, it is recommended to use the data model of the non-first normal form, called As for the second problem, namely the join operation, it is recommended to use the data model of the non-first normal form, called "embedded documents". Since MongoDB is based on the semi-structured data model shown in Fig. 2, each document of the collection is able to have the individual data structure. So, for example, the attribute "affiliation" in Fig. 2, which is usually saved in the different table by the normalization in the case of RDB, can be saved in the document "member" as the embedded document. So, they can be queried without using the join operation. The comparative performance evaluation between MongoDB with this method and RDB with the join operation was performed, and it has been shown that MongoDB was superior to RDB [6].

However, with this method, it needs to have the same data in the plural documents as the embedded document, and it arises the problem like the update anomaly of RDB. For example, in the case where the name of affiliation in Fig. 2 is changed, many records must be updated. For this problem, by using a programming language, the join operation can be composed even of MongoDB [1]. That is, if the query result of one collection includes the data of foreign key to refer another collection, then it can be utilized to query another collection. Then, the joined data can be composed of both query

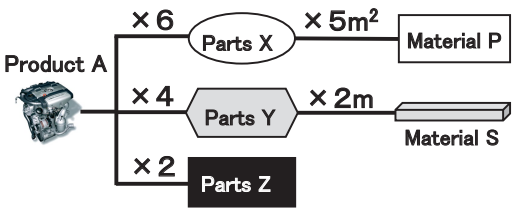

Figure 3: Structure of BOM of target system.

results. Here, in the case of using such a data manipulation, it is expected that its performance will be deteriorated.

On the other hand, as above-mentioned, the performance of the large amount data manipulation of MongoDB is so superior to RDB. So, it is expected that high performance can be obtained by applying MongoDB to the enterprise systems that manipulate a large amount of image data, even in the case where the join operations are implemented by this method.

However, we could not find the study, which evaluated the performance of the image data manipulation being accompanied by the join operation, on the premise of the actual enterprise systems.

3 TARGET SYSTEM

3.1 Target Function of Production Management System

The target enterprise system of this study is an actual production management system of some company which our laboratory is supporting. And, some of their functions have been already in operation; the others are currently under development. We use MySQL for RDBMS, and the calculation processing of each function is executed collectively by batch processing, then the results are stored into the database. We use Excel to entry the source data or to output the processing results as forms. We show the outline of the target system below.

The first is the material requirement calculation function, which has been already in operation and manages the bill of material (BOM) [17] as shown in Fig. 3. In this figure, Product A consists of 6 of part X, 4 of Y, and 5 of Z. And, parts X and Y are manufactured from $5m^2$ board material P and $2m$ stick material S respectively. As for the part Z, the commercial goods are purchased. In this way, by managing the BOM, it is possible to calculate the material cost of A based on the unit price of P, S and the price of Z.

The configuration of this processing is as follows. The data for the calculation consists of the BOM, products, parts, materials as shown in Fig. 3. And, they are stored in the tables of MySQL, and it is changed if necessary by using MySQL for Excel which is a linkage tool between MySQL and Excel. Then, the calculation processing is executed for all the data in a lump sum, and it is not necessary to specify the parameters. So, its process is described only by SQL statements, and they are executed as a batch file for Windows. Lastly, by using the view tables, the calculation results are converted to the various forms to be handled easily. Then, they are output by using the above-mentioned MySQL for Excel.

Spec_id	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
001A				1			1				1	
002A	4	5	4	2	1	4	5	3	2	2	2	
002B						2	1			3		
001A	4	2	2	3	2	7	4	4	4	4	2	1
003H	1		3	1	5	2	2					
109H	4	2	2	3	2	7	4	4	4	4	2	1

Figure 4: Monthly total of each spec

		Mar.								Apr.			
		22	23	24	27	28	29	30	31	1	3	4	5
Order_id	Spec_id	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri	Sat	Mon	Tue	Wed
AP000001-01	001A	▼	●					■			▲		
AP000002-01	002A		▼	●					■			▲	
AP000003-01	003H		▼	●					■			▲	
AP000002-02	002A			▼	●					■			▲
AP000004-01	104A			▼	●					■			▲

Remarks) ▼: Manufactured; ●: Start to prepare shipment; ■: Delivery; ▲: Used

Figure 5: Manufacturing schedule

The second is the production planning function, and the plan is made based on received or expected receipt orders. The contents of the order are designated by the specification (spec) sheet composed of each product type and its quantity, which has spec identifier (ID). And, this system targets the common products specified by the spec ID. That is, though the order includes the custom ordered products which are individually specified in each order, they are not administered by this system. We show the sample of the output documents in Fig. 4 and 5. Figure 4 shows the monthly total number of each spec, and it is used to grasp the long-term order status. Figure 5 shows the monthly work plan which is made per order, and it is used to grasp the daily milestone. We are currently conducting the operational test of this function and preparing the necessary data.

In this function, since the parameter must be specified such as the target month, we composed this in Excel and Excel VBA (Visual Basic for Excel applications). Concretely, parameters are entered from Excel sheet, then by Excel VBA, SQL statements are created and the corresponding batch file is started to execute these statements. Lastly, its results are processed to output forms by Excel VBA similar to the first. Moreover, we composed some MySQL data manipulations by the stored procedure or stored function [12]. For example, to make the production planning, we must estimate the number of business days excluding holidays. So, it is necessary to ensure that the calendar to show the number of each business day from the beginning of the year. And, since such processing includes the iterative processing, it cannot be done by only the simple SQL statements.

The third is the inventory management function that saves the status of each product shelf as the images and videos. Figure 6 shows the inventory images, and these are stored in the database. The aim of this function is to provide the inventory image with its necessary quantity information to the manager at the office to confirm the satisfaction of its inventory. So, it is necessary that the manager can set the inventory status based on the confirmation results, and the images of the spec-

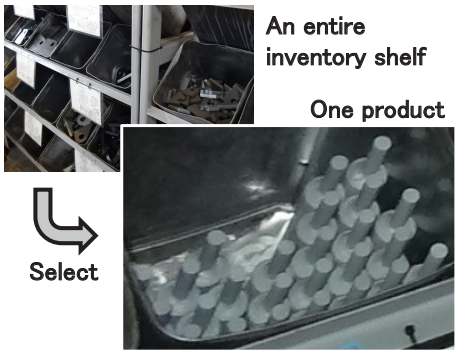

Figure 6: Inventory management utilizing images

ified products must be queried based on the manufacturing schedule shown in Fig. 5. To introduce this function, we are currently conducting the evaluation of its prototype.

Incidentally, the actual system is composed of various functions besides the above: it calculates the MRP (Material Requirement Planning) [17], which is the necessary quantity of the parts and materials, by the linkage of the production plan and BOM; the various management documents are output, and so on. However, since the data manipulation patterns are covered by the above-mentioned cases, we conduct the comparative study on only those in this study.

3.2 Database Structure and Data Manipulations

Figure 7 shows the ER diagram of the target system; below, we indicate the table name and attribute name in the ER diagram in italic. In addition, we show only the main tables and attributes for the sake of simplicity. (1) of Fig. 7 corresponds to the material requirement calculation function, and parts ($parts$) and products ($product$) are associated with BOM (BOM). And, the following price is set to the unit price ($price\_unit$) of parts: $price\_unit$ of $part\_price$ is set in the case where the part is purchased; price per 1kg ($price\_kg$) of $material$ is set in the case where the parts are manufactured from the material associated with material ID (mat_id).

(2) of Fig. 7 corresponds to the production planning function, and the data of $calendar$ such as the number of business days is calculated by the holiday information ($holiday$); the order and product are associated with specification composed of $specification$ and $spec$. The schedule data shown in Fig. 5 is calculated and saved into $manufacture\_plan$, in which each milestone date is included: manufacturing completion date (m_date_num), start date to prepare shipment (c_date_num), delivery date (d_date_num) and used date at the ordering company (u_date_num).

(3) of Fig. 7 corresponds to the inventory management function, and saves the inventory status of products as images and videos. $stock\_shelf$ indicates product shelf, in which the products are stored. And, $stock$ shows the status of the inventory: the image or video name (doc_name) saved in $image$; the correspondence data between them and product shelf is saved in $stock(p\_id, doc\_name)$; the inventory quantity ($quantity$) of $stock$ is set if necessary. Here, since the images and videos are captured at any time, their capture time (chk_time) is included in the primary key of $stock$, and the relationship between $stock$ and $stock\_shelf$ becomes many-to-one. Firstly, the image and video data is saved from the camera into the folder of the PC, then saved into $image$; And, the necessary data for this processing is downloaded from the database to the work folder of the PC when necessary.

We extracted the data manipulation patterns of these tables from the functions mentioned in Section 3.1, and got patterns shown below. Incidentally, the basic CRUD data manipulations of the single table are excluded.

(a) Join operation: in (1), each part is joined with the products which use the part respectively, and the results are saved into $parts\_cost$. So, the join operation between BOM and $parts$ is performed.

(b) Iterative operation: in (2), to set the number of business day ($date\_num$) of $calendar$, the division of the business day and holiday are set to the column $holiday$ of $calendar$ firstly. Then, the numbers of business date are set sequentially from January 1st, that is, it constitutes the iterative operation. Incidentally, it is implemented by the stored procedure in MySQL.

(c) Grouped aggregation operation: in (1), the product of material cost and quantity, which are expressed by $cost$ and $p\_quantity$ of $parts\_cost$ and calculated in (a), are aggregated for each product, and stored into $producrt\_price$.

(d) Selection of record with the self-join operation: in (1), since $part\_price$ has a history on the estimated date (est_ymd), the record having the max estimate date must be queried for each $part\_id$. In the SQL statement, this is expressed by the subquery with the self-join operation as shown in Fig. 8.

(e) Images and videos operation: in (3), the images and videos of the inventory shelves are stored into $image$, so these data must be inserted and queried. In MySQL, this is executed by "load_file" function to insert, and "select into dumpfile" statement to query.

4 IMPLEMENTATION OF DATA MANIPULATION USING MONGODB

4.1 Implementation Policy

In MongoDB, the Mongo shell is provided for methods for the CRUD operations and the interactive data manipulations which have JavaScript interface. And, similar to the SQL statements, JavaScript files can be executed as the batch file, or as a function like the stored procedure in SQL. In this study, we implemented the Mongo shell as a batch file on Windows as shown in Fig. 9. In Fig. 9, "JSfile.js" is the JavaScript file including the Mongo shell methods; and, it is executed by inputting to "mongo" command; then the execution results are output to "out.csv" file by a print statement of JavaScript.

Below, we show the implementation of each operation mentioned in Section 3.2. Incidentally, we describe only the main

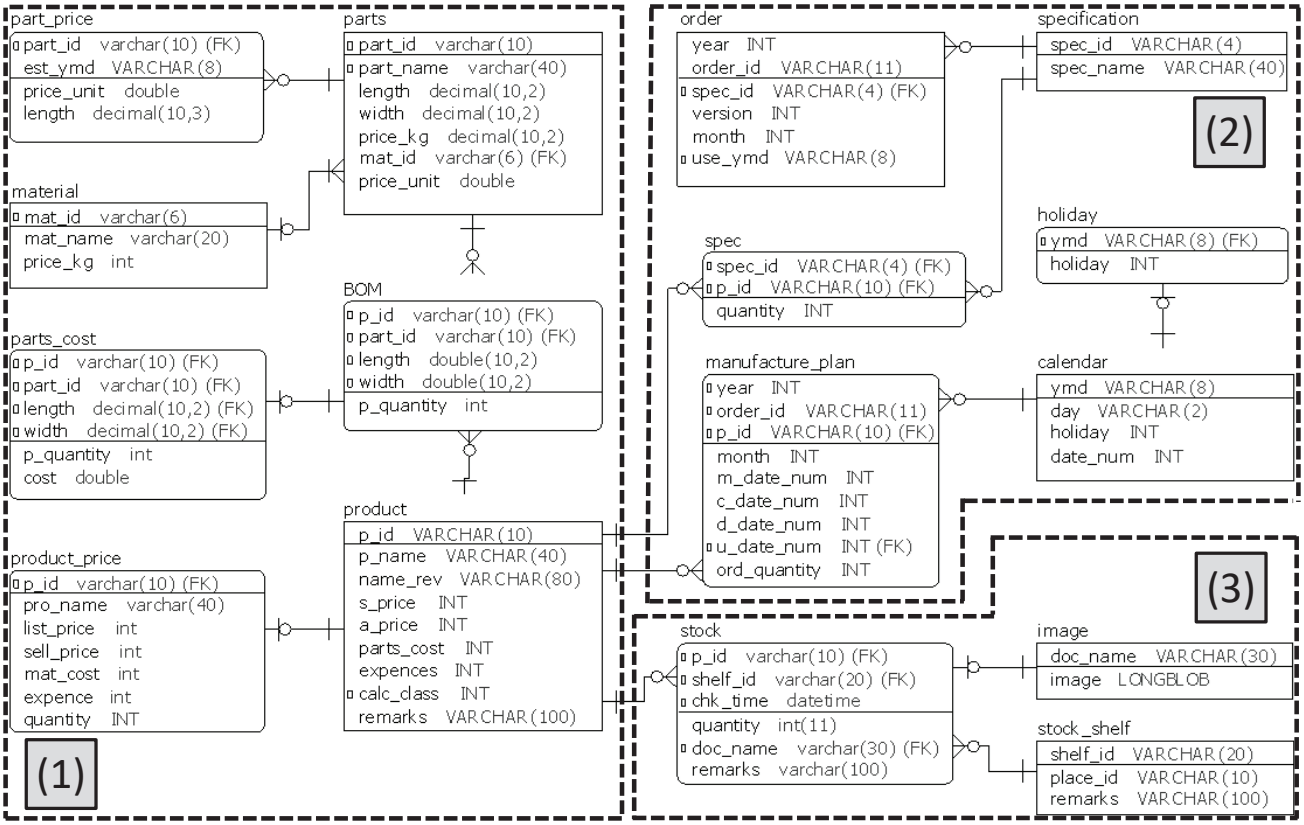

Figure 7: ER diagram of database

```
SELECT * FROM part_price AS a
  WHERE a.est_ymd = (SELECT
    MAX(b.est_ymd) FROM part_price AS b
    WHERE a.part_id = b.part_id);
```

Figure 8: Max value query by self-join

```
> mongo < JSfile.js > out.csv
```

Figure 9: Batch file of Mongo shell

attributes and operations for the sake of simplicity. In the actual system, related attributes and operations are added to the following logic.

4.2 Implementation of Join and Iterative Operation

In the Mongo shell, the join operation is not provided. So, the collections were joined as follows: firstly, we copied the collection corresponding to "many" of many-to-one into the temporary new collection; then, we added the fields of the collection corresponding to "one" to the above collection. In this way, we could create the result collection of the join operation.

In Fig. 10, we show the case of (a) in Section 3.2, in which $parts$ and BOM are joined to create $parts\_cost$. In (1) of Fig. 10, $parts\_cost$ is created by copying BOM, then $cost$

```
// (1) copy BOM to parts_cost
db.BOM.copyTo("parts_cost");
// (2) update parts_cost to join parts
var partRec;  // variable (document of parts)
var part=db.parts.find();  // (3) find method
while(part.hasNext()){
  partRec=part.next();   // get next document
  db.parts_cost.update(  // (4) update method
    {zairyou_id:partRec.zairyou_id,…
     haba:partRec.width}, // query condition
    {$set:{cost:part.price_unit}}, // set cost
    {multi:true});   // update multi documents
}
```

Figure 10: Join operation procedure of MongoDB

field of $parts\_cost$ is set to $price\_unit$ field of $parts$ in (2) as follows. Firstly, by using find method in (3), which corresponds to select statement of SQL, all the documents of $parts$ are queried. Here, documents are sequentially set to $partRec$ as same as the cursor operation of SQL. Next, update method at (4), which corresponds to update statement of SQL, updates the value of $cost$ attribute of all the documents that match the query condition shown by the first parenthesis "{ }" which expresses the pair as of "{field name:field value}". Here, the query condition is such that all these attribute values equal to the specified attribute values. In addition, if $parts\_cost$ collection does not have $cost$ field, then it is inserted.

```
// (1) total cost per p_id
Var mat_cost=db.parts_cost.aggregate
  ({$group:{_id:{p_id:"$p_id"},
   cost:{$sum:"$cost"}}});

// (2) document with latest date
var part_price=db.part_price.aggregate
  ({$match:{part_id:"P0001"},
   {$group:{_id:"$part_id",
      latest:{$max:{est_ymd:"$est_ymd"}}}}});
var partRec=part_price.next();
var laRec=db.part_price.findOne
  ({part_id:"P0001",est_ymd:partRec.latest});
```

Figure 11: Aggregation method of MongoDB

Incidentally, in the case where the join operation is performed for only some of documents matching the specified query condition, only the target documents are inserted at (1) in Fig. 10. For this operation, insert method corresponding to insert statement of SQL is used.

Next, the iterative statement, which is while statement and so on, can be used in the Mongo shell. So, we implemented the iterative operation to create $calendar$ shown in (b) of Section 3.2 by using these statements, like the stored procedure in SQL.

4.3 Implementation of Aggregation and Self-Join Operation

Mongo shell provides the aggregate method, which corresponds to the aggregation operator and group by clause of SQL. So, as for the aggregation operation of material cost for each part in (c) of Section 3.2, it can be executed by this method. In this method, as shown in (1) of Fig. 11, *$group* expression shows the fields to be aggregated, and *$sum* expression shows the aggregation method of summation like the SQL statement. Incidentally, the aggregation results can be got by the cursor operation like Fig. 10.

Similarly, as shown in (2) of Fig. 11, the selection operation of record having the max value shown in (d) of Section 3.2 can be performed by the aggregate method, and the latest estimated date was queried from $part\_price$ in this case. Here, since MongoDB does not provide the join operation, we configured the operation to query the target document again from $part\_price$ using the queried estimated date and $part\_id$ value. In Fig. 11, *$match* expression in aggregate method specifies the query condition. Also, findOne method queries only the single document, and in this figure, it queries as of the query condition that $parts\_id$ is "P0001" and est_ymd is the queried estimated date.

4.4 Image and Video Data Manipulation

The upper limit of the document size of MongoDB is 16 MB, and the GridFS interface is provided for data exceeding this limit. Using this interface, the image and video data is saved into GridFS collection divided from other attributes.

```
REM (1) insert operation of image from file
mongofiles -d iwin2017
    put 2017_5_A-1-3-1.JPG -l A-1-3-1.JPG

REM (2) query operation of image into file
mongofiles -d iwin2017
    get 2017_5_A1-1-1.JPG -l A-1-3-1.JPG
```

Figure 12: Image insertion and query command

Table 1: Comparison of CRUD operation

MySQL	MongoDB	Class
SELECT	find(), findOne()	CRUD
INSERT	insert()	
UPDATE	update()	
DELETE	remove()	
(*Join operation*)		(a)
	[many].copyTo()	
JOIN syntax	[one].find() (use cursor)	
	[many].update()	
Stored procedure	JavaScript	(b)
Stored function	JavaScript	
Group BY clause	aggregate()	(c)
(*Query record with max value)*		(d)
self-JOIN operation	aggregate()	
+ subquery	findOne()	
(*Image and video operation*)		(e)
INSERT	MONGOFILES	
+ LOAD_FILE()	command (put)	
SELECT INTO	MONGOFILES	
DUMPFILE	command (get)	

And, since the data insertion and query are performed by utilizing mongofiles command, not the Mongo shell, we configured to perform this command in batch files which are separated from the JavaScript files.

We show the examples of these commands in Fig. 12. Here, "-d" indicates the database, and "-l" indicates the file name on the disk. That is, we can save the image data into the database with the different name from the name as of disk file. Incidentally, since this command is a utility executed in Windows command line, connection and disconnection with the database is performed at each its execution.

Finally, in Table 1, we show the summary of the implementation method comparison between MySQL and MongoDB. Here, the column "Class" indicates the classification of these data manipulations. "CRUD" shows the basic data manipulation, and others indicate the number in Section 3.2.

5 IMPLEMENTATION OF SYSTEM AND COMPARATIVE EVALUATIONS

In order to demonstrate the target production management system can be constructed by using MongoDB, we implemented the principal part of this system by using MongoDB

according to the correspondence of CRUD operation shown in Table 1. Then, we conducted the comparative evaluations of the program volume and execution performance between MongoDB and MySQL.

5.1 Implementation Using MongoDB

First, as for the material requirement calculation function shown in (1) of Fig. 7, we implemented the process to create $product\_price$. Here, cost data of $parts\_price$ and $material$ is reflected into $parts$, then $product\_price$ is created from $parts$ via $parts\_price$. We implemented this processing using the Mongo shell as the batch file shown in Fig. 9.

Second, as for the production planning function shown in (2) of Fig. 7, we implemented the following processing: one creates $calendar$ from $holiday$; the other makes the csv files for the aggregation and schedule document shown in Fig. 4 and Fig. 5 respectively. We implemented this processing using the above-mentioned batch file, and we embedded the parameters in the JavaScript program directly without linking with Excel for the sake of simplicity.

Third, as for the inventory management functions shown in (3) of Fig. 7, we implemented the following two processing shown in Fig. 12. Incidentally, these implementation methods are same as MySQL except the execution command as shown below.

One is the processing to save the pictures and videos of the product shelves into $image$, and to insert the correspondence data between $stock\_shelf$ and $image$ into $stock$, that is, this creates the correspondence between the shelves and the images or videos. To save the images and videos data, their file name in the camera must be grasped in the insertion program. So, we implemented this processing using Excel VBA to make the insertion batch file, in which insertion is executed by mongofiles command. And, we implemented the correspondence data insertion program by using the Mongo shell, which is executed by the batch file.

The other is processing to query the image and video data. Similar to above, we implemented this processing to be executed by mongofiles command, which was made by using Excel VBA based on the given query condition: specified shelves, or the product shipment date and so on.

5.2 Comparative Evaluations of Program Volume

In order to perform comparative evaluations of productivity between MongoDB and MySQL, we counted the number of source lines of the programs respectively. Table 2 shows these results, and (1) shows the material calculation; (2) shows the production planning function. Incidentally, since the user interface programs of the inventory management system operations were made by using Excel VBA as mentioned in Section 5.1, and they were common to both databases. So, we omitted their evaluation. Here, (2) was divided into the three processings: Plan(C) shows the creation processing of $manufacture\_plan$ in (2) of Fig. 7; Plan(O) shows the processing to output the query results into csv files for the forms shown in Fig. 4 and Fig. 5; Plan(P) shows the output processing of these forms from the csv files. Moreover, since the statement other than the SQL statement and Mongo shell is necessary, we show their individual volume in this table.

Table 2: Comparison of program volume (line)

	MySQL			MongoDB		
	SQL	Else	Total	Shell	Else	Total
(1) Material	15	0	15	24	29	53
(2) Plan(C)	8	64	72	12	91	103
(2) Plan(O)	11	7	18	8	24	32
Total	34	71	105	44	144	188
(2) Plan(P)	0	180	180	0	180	180

(1): Material requirement calculation function
(2): Production planning function
C: create data; O:output to csv file; P: print document

The function indicated by (1) in Table 2 could be configured only by the SQL statements in MySQL. However, in MongoDB, it was necessary to use JavaScript in addition to the Mongo shell. In this case, the number of source lines of the latter was about 3.5 times that of the former. On the contrary, the processing indicated by Plan(C) and Plan(O) in (2) could not be described only by SQL statements in MySQL, so it had to be described with the stored procedures and stored functions; MongoDB was the same as MySQL. In this case, the former number of source lines was about 2 times that of the latter.

The processing indicated by Plan(P) in (2) is the printing of a form, and there was no database access. So, this processing was common to MySQL and MongoDB. That is, the processing to output the form consists of data extraction from the database indicated by Plan(O) and printing as of Plan(P). In this case, the ratio of Plan(P) was 91% in MySQL and 85% in MongoDB.

5.3 Comparative Evaluations of Elapsed Time

We executed each processing mentioned in Section 3.1 on the standalone PC environment to evaluate the elapsed time comparatively. Here, we modified each processing to execute only the database access including the data format operations as the batch file, that is, the following processings were excluded: defining the parameters, printing of forms and so on. The execution environment is as follows. CPU is i7-6700 (3.41GHz); memory is 16GB; the disk is SSD memory of 512GB; OS is Windows 10. We adopted MySQL (Ver. 5.7.12), MongoDB (Ver. 3.4.3) for the database.

We show the evaluation results in Table 3. “Material” shows the elapsed time of the material calculation shown by (1) of Table 2, and it includes the manipulation (a), (c) and (d) mentioned in Section 3.2. The elapsed time of MongoDB was approximately 11 times that of MySQL. “Calendar” and “Plan” are parts of the production planning function: the former creates $calendar$, and as for MySQL, it was executed by stored procedure with the iterative manipulation shown in (b) of Section 3.2; similarly, the latter created $manufacture\_plan$ by

Table 3: Comparison of elapsed time (second)

Processing	MySQL	MongDB	Ratio	Manipulate
Material	1.07	11.90	11.1	(a), (c), (d)
Calendar	9.58	5.10	0.5	(b)
Plan	0.32	26.84	83.0	(a) (3-join)
Image(in)	23.48	260.78	11.1	(e)
Image(out)	0.24	77.76	324.0	(e)

Remarks: Ratio=MongoDB/MySQL

SQL statement to join the 3 tables, $calendar$, $order$ and $spec$. The elapsed time of MongoDB was about 0.5 and 83 times that of MySQL respectively. "Image(in)" shows the case to insert the image data of actual product shelves into database from the disk: the number of images is 340, and the size of each image is from 0.9 MB to 2.9 MB; "Image(out)" shows the opposite case to the previous case, that is, the same data is queried from the database to store into the disk as files.

As a result, in this case, the elapsed time of MongoDB was about 11 and 324 times that of MySQL respectively. That is, MongoDB was degraded despite being more efficient than MySQL in manipulating a large amount of data. The reason for this was as follows. Firstly, since the mongofiles command must be executed for each file as the Windows command individually, the connect operation occurs for each file insertion and database query. So, the delay occurred in this process. On the contrary, MySQL could execute all the data manipulations by connecting once similar to the other operations.

By the way, we separated the fields of the images and videos from the inventory table ($inventory$) and composed the individual table ($image$) as shown in Fig. 7 even in MySQL. This was due to the results of the preliminary study: in the case of gathering all these fields to one table, the extreme delay occurred. That is, to confirm the inventory, firstly we saved the inventory image shot in the order of the shelf ID to this table; then, updated shelf ID ($shelf\_id$) according to the image order. However, this update operation took more than 20 seconds for the above-mentioned 340 data, and it was too long for the operations. Here, it was pointed out that the instances of LONGBLOB should not in the query results if it was not really necessary [13]. However, as a result of this preliminary study, we found that the extreme latency occurred not only in this case but also in the case where the images and video column was not included in the data manipulation.

6 PERFORMANCE EVALUATION USING JAVA

6.1 Implementation Using Program Language

As shown in "Image(in)" and "Image(out)" of Table 3, the performance to manipulate the image data of MongoDB, in this case, is inferior to RDB, though the performance to manipulate the single large amount of video data is better than RDB as shown in Fig. 1. Here, as mentioned in Section 4.4, the connection to, and disconnection from the database are performed for each execution of mongofiles command, and a large number of image data manipulations were performed by this command in this evaluation. As a result, this performance deterioration occurred.

```
// (1) Insert statements of MongoDB
InputStream st = new FileInputStream(new File("fName"));
ObjectId fileId = gridFSBucket.uploadFromStream("doc_name", st);

// (2) Select statements of MongoDB
FileOutputStream st = new FileOutputStream("fName");
gridFSBucket.downloadToStream(fileId, st);

// (3) Insert SQL statement of MySQL
insert into image values ('doc_name', load_file('fName'));

// (4) Select SQL statement of MySQL
select image into dumpfile 'fName' from image
    where doc_name = 'doc_name';
```

Figure 13: Image data manipulation statements

On the other hand, for example, MongoDB Java driver provides GridFSBucket class for GridFS interface, and the multiple image data manipulation can be performed after connecting once. Therefore, in order to prevent the performance degradation shown in Table 3, we implemented this manipulation by using Java. And, we also performed the comparative performance evaluation between MongoDB and MySQL. Here, as for MySQL, this manipulation was implemented by using Java, too.

As for the implementation environment, we used Java Platform Standard edition 8, MongoDB Java driver Ver.3.5.0 and MySQL Connector/J Ver.5.1.1.41. Other environments were the same as in Section 5.3. And, the implementation target was the inventory management function using images shown in (3) of Fig. 7.

Firstly, we evaluated the basic function, that is, the individual performance of the join processing and image manipulation. Next, we evaluated the combination processing, that is, the performance of the case where both of the join operation and image data manipulation are performed. Furthermore, we also evaluated the combination structure of MySQL for the join operation and MongoDB for the image data manipulation. Hereinafter, we indicate this structure by "mix" structure, and its detail is as follows.

The manipulations of a large amount of data such as images are performed by the streaming in both of MongoDB and MySQL. Figure 13 shows the examples of the statement of image data manipulations. Here, "fName" shows the image file name; "doc_name" shows the image name in the database. (1) and (2) show the insert and select statements of MongoDB to manipulate image respectively: In each first line, the streaming of Java is defined; in each second line, upload and download of the image is executed by using GridFSBucket class respectively. Incidentally, "ObjectID" is the identifier of the document in MongoDB, and it can be queried by using the image name "doc_name" in advance the select statement (2). On the other hand, as for SQL statement of MySQL, the image file name to be inserted is specified by using "load_file" function; the destination image file name of the image data to be queried is specified by using "dumpfile" clause.

Therefore, as for the implementation of the image data manipulation in Java, following composition is possible: firstly, the target image name "doc_name" is obtained and saved into the variable of type String; then, the statements to manipulate the image in Fig. 13 are executed. Moreover, in this composition, the image name can be queried by using MySQL and the image data manipulation can be executed by using MongoDB. In this way, by using this mix construction, it was expected that the superior operations of each database could be combined.

6.2 Evaluations of Basic Functions

To evaluate the basic functions, we implemented the following 4 cases by using both of MySQL and MongoDB.

(A) Join operation on three tables (3-Join): this queries the data of three tables that matches the designated query condition of p_id of $product$ by joining these three tables in Fig. 7: $product$, $stock$ and $stock\_shelf$.

(B) Self-join operation (Shelf-Join): this queries only the latest data of $stock$ that matches the designated query condition of p_id. That is, only records having the latest chk_time are queried for the pair $\{p\_id, shelf\_id\}$. Here, it is composed of the subquery with the self-join operation in RDB.

(C) Image insertion: this inserts all the image data existing in the designated folder into the database.

(D) Image download: this downloads all the image data existing in the designated table of the database into the designated folder.

As for (A), though we implemented it by cursor operation in both databases, their structure was different. That is since MySQL has the SQL statement of the join operation, it is possible to query the join results as a cursor. On the other hand, MongoDB has no statement of the join operation. So, we implemented the following processing: firstly, we queried the target data of $product$ including p_id; then, we queried the target data of $stock$ including $shelf\_id$ by this p_id; lastly, we queried the target data of $stock\_shelf$ by this $shelf\_id$, and joined all the query results.

As for (B), we implemented the query by using the subquery and self-join operation shown in Fig. 8 in RDB. On the other hand, as shown in Table 1, MongoDB had the aggregate statement corresponding to the group by clause in RDB. So, we queried the max value of chk_time of $stock$ for the pair p_id, $shelf\_id$, then queried the target data by using these data.

As for (C) and (D), we implemented by the similar structure in both databases. For (C), we queried all the image data of the designated folder and inserted them sequentially into the table. On the contrary, for (D), we queried the image data sequentially by utilizing the cursor and saved into the designated folder. Here, in MySQL, we implemented by utilizing the insert and select statements of SQL. And, in MongoDB, though we implemented by utilizing GridFS interface, the database connection could be maintained by using Java as above-mentioned.

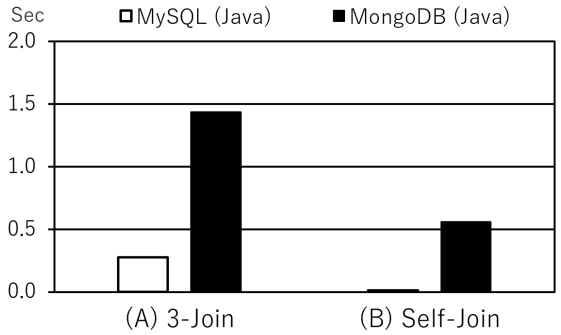

Figure 14: Elapsed time of join operation

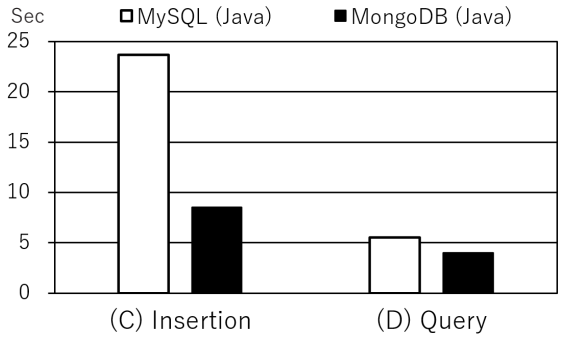

Figure 15: Elapsed time of image manipulation

In Fig. 14, we show the evaluation results of (A) and (B). The number of data of the three tables was 1063, 2000 and 845 respectively, and the numbers of result data of (A) and (B) were 1,770 and 339. As shown in Fig. 14, the elapsed time of MySQL was 0.276 seconds in (A), which is 5 times faster than 1.435 seconds of MongoDB; the one of MySQL was 0.017 seconds in (B), which is 33 times faster than 0.561 seconds of MongoDB.

And, in Fig. 15, we show the evaluation results of (C) and (D). Contrary to the previous results, the elapsed time of MongoDB was 18 times and 9 times faster than the one of MySQL respectively in these cases. The elapsed time of both was 8.5 and 23.7 seconds in (C) respectively, and 4.0 and 5.5 seconds in (D).

6.3 Evaluations of Combination Processing

Next, we performed the comparative performance evaluations for the practical processing by combining the above-mentioned operations. That is, we implemented and evaluated the following processing: firstly, we queried the target image name (doc_name) by the join operation in (A) or (B); then we inserted these image data into the database in (C); lastly, we queried these image data from the database and saved into another folder in (D). Here, we implemented three cases: the first was implemented by only MySQL; in the second, (A) and (B) were implemented by MySQL, and (C) and (D) were implemented by MongoDB; the third was implemented by only MongoDB. Here, (A), (B) and so on is shown in Section 6.2, and hereinafter, it is same.

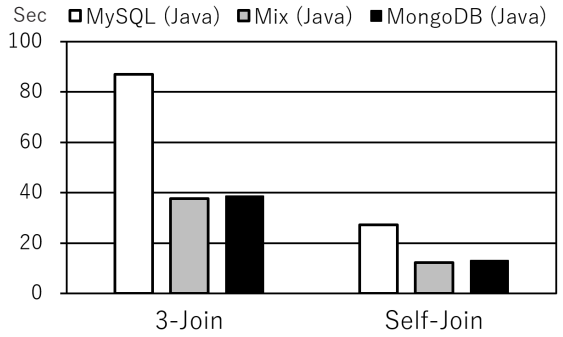

Figure 16: Elapsed time of combination processing

Table 4: Breakdown time of combination processing (second)

Processing	Construct	Join	Insertion	Query
3-Join	MySQL	0.285	68.967	17.658
	Mix	0.280	25.819	11.506
	MongoDB	1.461	25.807	11.419
Self-join	MySQL	0.015	21.758	5.531
	Mix	0.021	8.304	3.894
	MongoDB	0.998	8.196	3.911

We show the evaluation results in Fig. 16, and the breakdown of the elapsed time of each operation in Table 4. Here, since the query results of (A) and (B) were same as those of Section 6.2, the number of image data that was manipulated was also 1,770 and 339 respectively. And, each elapsed time is indicated as follows: "Join" indicates the join operation; "Insertion" indicates the image data insertion; "Query" shows the image data query. As shown in Table 4, since the time to manipulate image data was longer than the time of joining operation, the elapsed time as of MongoDB was shorter than MySQL. Especially, a large difference was observed in the elapsed time of the image insertion.

In addition, as for the mix structure, which is shown by "Mix" in Table 4, it was constructed by using MySQL's join operation and MongoDB's image data manipulation. So, for example, the elapsed time of the join operation is similar to MySQL; the one of image manipulation is the same as MongoDB. That is, we obtained the superior performance for each operation as mentioned in Section 6.1. As a result, the best performance was achieved by the mix structure.

7 DISCUSSIONS

We discuss the evaluation results. First, as for the target production management system, we found that all the functions implemented by using MySQL could be implemented by using MongoDB. As the results of the productivity comparative evaluations, though the number of MongoDB's data manipulation commands increased, the ratio of the description of data manipulation was very small in the actual systems as shown in the Table 2. Therefore, from the viewpoint of the overall system development man-hour, we consider that the importance of the selection concerning the both will be small.

Second, we found some note points to maintain the performance of a large amount of data. As shown in the last paragraph of section 5.3, it was necessary to separate such a data column to the individual table even in MySQL as same as MongoDB. On the other hand, in MongoDB, the connection to the database should be maintained as shown in Table 3 and Fig. 15, that is, in the case to access such a data many times, it should be composed by using programing language and so on.

Lastly, by using programing language, we could use both of MySQL for the join operation and MongoDB for the image data manipulation as shown in the last paragraph of Section 6.1. In the case of manipulating a large amount of data, by using MongoDB, we could obtain better performance than MySQL. However, as shown in Fig. 7, there are many processes that use no image in the enterprise system. On the contrary, they utilize the join operation. So, we currently consider that there is a solution to use both as above-mentioned.

8 CONCLUSIONS

In order to manipulate a large amount of data, the application of NoSQL database is spreading. However, to apply NoSQL databases to the enterprise systems, there is the challenge that the join operation must be implemented efficiently. In this study, we conducted the comparative evaluations between MySQL and MongoDB for the actual enterprise system in two cases: the implementation by using Mongo shells and the one by using programming language Java.

In the first case, we found the functions of general SQL statements could be implemented by using only Mongo shells, though the performance degraded in the case of manipulating many large amounts of data such as images.

In the second case, we found that the above-mentioned deterioration of performance could be solved, and the elapsed time to manipulate a large amount of data was longer than the one of the join operation. That is, better performance was obtained at the whole data manipulations by using MongoDB. Furthermore, we showed it was possible to construct the configuration that took each advantage of both databases: MongoDB manipulated a large amount of data; MySQL manipulated the other data including the join operation.

For the future challenge, we will expand the application area of MongoDB by using sharding and improving the data structure.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 15K00161.

REFERENCES

[1] R. Arora, and R. R. Aggarwal, "Modeling and querying data in MongoDB," International Journal of Scientific and Engineering Research, Vol. 4, No. 7, pp. 141–144 (2013).

[2] M. Bach, and A. Werner, "Document-Oriented Data Stores of Vision Objects," Proc. of Innovative Control

Systems for Tracked Vehicle Platforms, pp. 163–174 (2014).
[3] K. Banker, “MongoDB in Action,” Manning Pubns Co. (2011).
[4] A. Boicea, F. Radulescu, and L.I. Agapin,“MongoDB vs Oracle–database comparison,” Proc. of Third International Conference on Emerging Intelligent Data and Web Technologies, pp. 330–335 (2012).
[5] M. Chen, S. Mao, and Y. Liu, “Big data: A survey,” Mobile Networks and Applications, Vol. 19, No. 2, pp. 171–209 (2014).
[6] C. Győrödi, R. Győrödi, G. Pecherle, and A. Olah, “A comparative study: MongoDB vs. MySQL,” 13th International Conference on EMES, pp. 1–6 (2015).
[7] S. Hiremath, G. Yang, and K. Mankodiya, “Wearable Internet of Things: Concept, architectural components and promises for person-centered healthcare,” EAI 4th International Conference on Wireless Mobile Communication and Healthcare, pp. pp. 304–307 (2014).
[8] T. Kudo, M. Ishino, K. Saotome, and N. Kataoka, “A Proposal of Transaction Processing Method for MongoDB,” Procedia Computer Science, Vol 96, pp. 801–810 (2016).
[9] T. Kudo, Y. Ito, and Y. Serizawa, “An Application of MongoDB to Enterprise System Manipulating Enormous Data,” Proceedings of International Workshop on Informatics (IWIN2016), pp. 277–284 (2016).
[10] MongoDB, Inc., “Welcome to the MongoDB Docs,” https://docs.mongodb.com/ (reffered Oct. 16, 2017).
[11] K. Nagasawa, and T. Kudo, “Development of Mobile Quotation System Utilizing Tablet and MongoDB,” Proc. of 2017 IEICE General conference, D-9-23, p. 113 (2017) (In Japanese).
[12] Oracle Corp., “Chapter 23 Stored Programs and Views,” https://dev.mysql.com/doc/refman/5.7/en/stored-programs-views.html (reffered June 6, 2017).
[13] Oracle Corp., “11.4.3 The BLOB and TEXT Types,” https://dev.mysql.com/doc/refman/5.7/en/blob.html (reffered June 6, 2017).
[14] D.R. Rebecca, and I. E. Shanthi, “A NoSQL Solution to efficient storage and retrieval of Medical Images,” International Journal of Scientific & Engineering Research, Vol. 7, No. 2, pp. 545–549 (2016).
[15] E. Redmond, and J.R. Wilson, “Seven Databases in Seven Weeks: A guide to Modern Databases and the NoSQL Movement,” Pragmatic Bookshelf (2012).
[16] K. Seguin, “The Little MongoDB Book” (2011), http://openmymind.net/mongodb.pdf (reffered May 5, 2017).
[17] S. P. Singh, “Production and Operation Management,” Vikas Publishing House Pvt Ltd (2014).
[18] M.P. Stevic, P., B. Milosavljevic, and B.R. Perisic, “Enhancing the management of unstructured data in e-learning systems using MongoDB,” Program, Vol. 49, No. 1, pp. 91–114 (2015).

(Received October 20, 2017)
(Revised December 26, 2017)

Tsukasa Kudo received the B.S. and M.E. from Hokkaido University in 1978 and 1980, and the Dr.Eng. from Shizuoka University in 2008. In 1980, he joined Mitsubishi Electric Corp. He was a researcher of parallel computer architecture and engineer of business information systems. Since 2010, he is a professor of Shizuoka Institute of Science and Technology. Now, his research interests include database application and software engineering. He is a member of IEIEC and IPSJ.

Yuki Furukawa is currently working toward a B.I. degree at Shizuoka Institute of Science and Technology. Her research interests include production management system and comparative evaluations between MySQL and MongoDB.