Implementation of Integrity Maintenance Method of Query Result by Bitemporal Database

Tsukasa Kudou†, Masahiko Ishino‡, Kenji Saotome*, Nobuhiro Kataoka** and Tadanori Mizuno***

†Mitsubishi Electric Information Systems Corporation, Japan
‡Mitsubishi Electric Information Technology Corporation, Japan
* Hosei Business School of Innovation Management, Japan
** School of Information Science and Technology, Tokai University, Japan
*** Faculty of Information, Shizuoka University, Japan
kudou-tsukasa@mdis.co.jp

Abstract - Generally, databases of mission-critical systems are updated with entry data by transaction processing, and are queried to make statistics and so on by batch processing. So, it is necessary that both processing can be executed simultaneously for the efficient system operation. In principle, a bitemporal database system can not only maintain the integrity of the query results even under simultaneous data entry, but also query the corrected data to provide valid query results. However, in the actual operation of a mission-critical system, various problems occur such as data-entry mistakes and data-entry backlogs. Therefore, the bitemporal database has to be able to support these problems. In this paper, we show an application case of a bitemporal database into a mission-critical system, to investigate an implementation of a method for maintaining the integrity of the query results under real-world conditions. As a result, we have confirmed that integrity was maintained even while the database was being updated, and that various kinds of corrections done by the actual system operations were reflected in the query results. Furthermore, we confirmed that the method is effective in not only the data corrections for a short period confirmation work but also the correction management of data for a long period of real-world use, and that data correction can be managed in both internal processes and business procedures individually.

Keywords: temporal database, bitemporal database, query, integrity, mission-critical system

1 INTRODUCTION

In many mission-critical systems such as in retail, finance and manufacturing, data entered from online-terminals are committed to the database in discrete transactions (hereinafter "online entry"). Also, at regular intervals or as needed, batch query jobs are performed, often accessing a massive amount of data, to make statistical documents, analysis documents and so on. For example, in the case of a retail system, the sales data are entered from terminals in each store, and reflected in the central corporate database. Then, settlement of accounts processing is performed on a daily or monthly basis by executing queries involving a great deal of data from all of the stores for the given period. Here, if we divide the time zone of the online entry and the batch processing, the batch processing has to be performed in night, and moreover its time may pass away by the extension of the online entry time. Therefore, it is necessary that online entry can continue concurrently with the various queries being processed to produce the reports.

In this situation, a snapshot database [2], which stores only the latest state of the data, has the problem that the integrity of the query results isn't guaranteed when data are changed by the online entry system, between or while individual queries in the batch job. Database systems, which employ transaction processing [4] equipped with lock control to manage concurrency, maintain the integrity of their data even when simultaneous accesses by many users arise. And, even if a large amount of data is queried, the method to divide a long time transaction of large batch into many mini-batches [4] is used. However, a long time is necessary for querying a great deal of data, so, even if the each query processing avoids conflict with the online entry, some portions of the query results may come after the online entry although other portions come before.

On the other hand, the temporal databases manage the data that changes in chronological order, and a lot of researches have been performed about them [3], [6], [7], [11], [13], [14]. In temporal databases, time can be captured along two distinct time lines: the valid time and the transaction time [7]. The valid time denotes the time a fact was true in the real-world; the transaction time is the time during which the fact was present in the database as stored data. Temporal databases are divided into three types: valid time databases, transaction time databases and bitemporal databases. Valid time databases manage only the valid time, transaction time databases manage only the transaction time, and bitemporal databases manage both the valid time and the transaction time [12].

It has been shown that a transaction time database can express prior states of the database at designated past transaction times as a snapshot [5], unaffected by continuing online entry. However, in actual system operations, when errors in the data are detected during batch processing, we have to correct the data and restart the batch job from the beginning. In this case, the problem arises that the corrected data is not reflected in the query results of the redone batch job, because the transaction time of the corrected data comes after the designated time of the query.

The version-control data model manages not only the times

when data is added or deleted, but also derived relations between versions [8], [9]. As a result, it can manage both version sets that are derived from the designated transaction times: one set is created in chronological order by normal data entry; the other set is created out of order by data correction. So, corrected data can be reflected in the query result. This model is important in software development, CAD, and other systems which rely on strict version control. But, in applying it to mission-critical systems that require a high data input frequency, processes for detecting corrected data and deriving new versions have to be executed so frequently that performance suffers.

Using a bitemporal database, we can obtain query results that have verifiable data integrity and reflect the corrected data even while online entry continues concurrently. Here, to apply this to mission-critical systems, we have to confirm that it can support a wide range of corrections which occur during actual system operations. On the other hand, this database must manage records with two kinds of time attributes, which increases the difficulty of applying it to information systems by complicating the query procedures and increasing of the amount of data. Therefore, there are very few cases where it has been implemented, and we could not find a case in which a bitemporal database had been used and evaluated in an actual mission-critical system.

In this paper, we show that a bitemporal database can maintain the integrity of the query results in real-world conditions. That is to say, various kinds of corrections which were applied to the data during actual system operations are properly reflected in the query results, and batch queries were run concurrently with live online entry. Furthermore, we applied it to a mission-critical system and evaluated its effect on actual system operations. As a result, we confirmed that the above-mentioned integrity was maintained. Moreover, we confirmed the following effects: first, data correction could be managed over a long period of real-world use in addition to the use in confirmation work in a short period; second, the data corrections of internal processes and business procedures could be managed individually.

To handle the implementation problems, we improved the following points. First, we created work files for batch processing jobs by constructing a temporary table of the database (hereinafter "work table"). We then processed the data step-by-step using this work table in order to simplify each individual query procedure. Second, we implemented the system such that each record is defined to be valid until it is superseded. Thus, because we could maintain complete bitemporal data by only storing one data for a change of real-world, the overall increase in the amount of data was minimized.

In section 2, we expound on the problems that arise when a transaction time database is queried concurrently with online entry, and in section 3, we show that this problem is solved by utilizing a bitemporal database. In section 4, we show an implementation case of a bitemporal database in a mission-critical system and show the operation of the system. Finally, in section 5, we evaluate and consider the implementation of the bitemporal database.

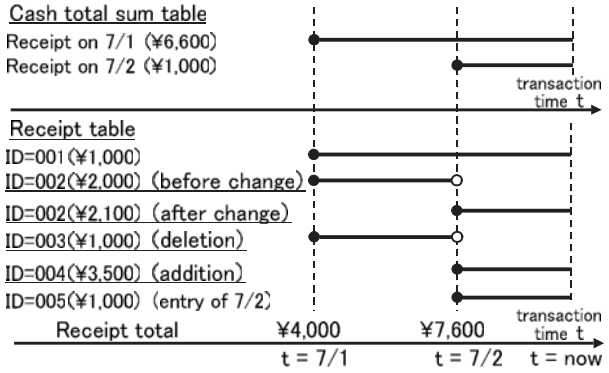


Figure 1: Snapshot of transaction time database.

2 PROBLEM OF TRANSACTION TIME DATABASE

2.1 Query Processing to Deal with

In this paper, we deal with the following query processing.

i. **Concurrent Execution of Query and Online Entry**
Query of batch processing is executed concurrently with the online entry from many online-terminals that update the database with high frequency.

ii. **Query Large Amounts of Data in a Single Batch**
Massive amounts of data are queried in a lump, so the processing takes time.

iii. **Redoing Batch Queries after Data Correction**
When data errors are detected in the query result, we have to correct the data such by making changes, deletions or additions. Then, the queries have to be executed again on the same data without updates from further online entry.

Such query processing is standard in mission-critical systems like the settlement account processing of retail systems, in which queries are run on massive amounts of data and executed by batch processing. Transaction processing of a database system equips various kinds of integrity constraints [4], and a method that maintains the integrity of the database by the cooperation even when it was updated by a group of people has been proposed [10]. Therefore, the integrity of the database is maintained in many cases. However, the confirmation of the integrity, which has to query large amount of data, has to be executed by batch processing. For example, the comparison of several tables, the calculation of the receipt totals to compare with the actual cash total sum, and so on. Therefore, the batch processing has to be redone after correcting data when data errors are detected.

2.2 Query Method during Online Entry

In the transaction time database, the transaction time is expressed by $[t_a, t_d)$. In this expression, t_a shows the time that the data was added to the database, and t_d shows the time that

the data was logically deleted from the database. As long as the data hasn't been deleted yet, t_d is expressed as "now", which shows the time when querying is executed [1], [12]. When we change some data, the time t_d of the data is set to the time of the change. Thereby it is logically deleted. Then, new data is added to the database to replace the old data logically. By this way, the data once added is left in the database without being deleted physically. Therefore, we can get the snapshot at transaction time t by querying the data in the condition of $t_a \leq t < t_d$, and the integrity of the snapshot is kept even if the database is updated by the online entry while we are querying.

Figure 1 shows the application example of querying snapshot in a retail system. By comparing the receipt table of July 1st with the cash total sum table of the same day, three data errors were detected: an entry mistake of $ID = 002$, an overlap entry of $ID = 003$ and entry leakage of $ID = 004$. And, corrections of data by change, deletion and addition were done on July 2nd. Moreover, a new receipt data of $ID = 005$ was added on July 2nd. In the snapshot of July 1st, these update on July 2nd were not reflected. By this characteristic, even if the database were updated by the online entry during the query, the integrity of the query result can be maintained.

2.3 Problem about Query of Correction data

In the actual business system, the right query result of the receipt table of July 1st, which total is adjusted with the cash total sum table, is necessary. However, there is a problem that it is impossible to query current corrected state of July 1st. Because, by the snapshot on July 1st, the data before correction is queried in the example of Figure 1; by the snapshot on July 2nd, the receipt data on July 2nd $ID = 005$ is queried, too.

Here, in the following examples, we express the transaction time by making its unit a day. In the implementation, its unit is determined according to requirements for the system such as frequency of the online entry.

3 QUERYING BITEMPORAL DATABASE

We show that the problem in section 2.3 can be solved by the bitemporal database.

3.1 Composition of Bitemporal Database

The relation of the bitemporal database R is expressed as follows.

$$R(K, T, V, A) \tag{1}$$

We show each attribute as follows.

- $K = \{K_1, ..., K_m\}$
  This expresses the set of attributes constituting the primary key of the snapshot queried by designating both time attributes: the transaction time and the valid time.
- $T = \{T_a, T_d\}$
  This expresses the time period attribute of the transaction time, which is generated by system and isn't made public to the users. Here, T_a shows the time that the data was added to the database (hereinafter "addition time"), and T_d shows the time that the data was logically deleted from the database (hereinafter "deletion time"). As long as the data hasn't been deleted yet, the instance of attribute T_d is expressed as "now".
- $V = \{V_a, V_d\}$
  This expresses the time period attribute of the valid time, i.e. the corresponding fact was true in the real-world. Here, V_a shows the beginning time of the time period, and T_d shows the ending time. In the case that the data is still true when we query it, the instance of attribute V_d is expressed as "now" like T_d. Regarding the valid time, the data once added is left in the database without being deleted physically like the transaction time, too. And, we can get the snapshot of a designated valid time by querying the database.
- $A = \{A_1, ..., A_n\}$
  This expresses the other attributes.

3.2 The Method for Querying Bitemporal Database during Online Entry

In the bitemporal database, the integrity of the snapshot is also kept even while it was being updated by the online entry, like the transaction time databases shown in section 2.2, because it manages the transaction time. Moreover, we can query the state of the real-world of any designated valid time, because it manages the valid time.

The snapshot of R, transaction time of which is t_1 and valid time t_2, consists of the data that satisfy the both following conditions: its instance of transaction time period $T = \{T_a, T_d\}$ includes t_1; its instance of valid time period $V = \{V_a, V_d\}$ includes t_2; Therefore, its relation $R_1(t_1, t_2)$ is expressed as follows.

$$R_1(t_1, t_2) = \{r | r \in R \wedge r[T_a] \leq t_1 \wedge t_1 < r[T_d] \wedge r[V_a] \leq t_2 \wedge t_2 < r[V_d]\} \tag{2}$$

Here, $r[T_a]$, $r[T_d]$, $r[V_a]$ and $r[V_d]$ show the respective instance of attributes T_a, T_d, V_a and V_d of r, the data included in R. Therefore, in the case that an error data was detected in the query result and corrected at transaction time t, we can query the corrected data by the snapshot designating times as follows: the condition of transaction time t_1 is $t_1 > t$; valid time t_2 is the same with last query.

3.3 Effect to Data Corrections

Figure 2 shows the query results, which is executed by designating transaction time t_1 and valid time t_2, in the case of Figure 1. Regarding the bitemporal database, the receipt data in real-world conditions can be queried. That is, as shown in

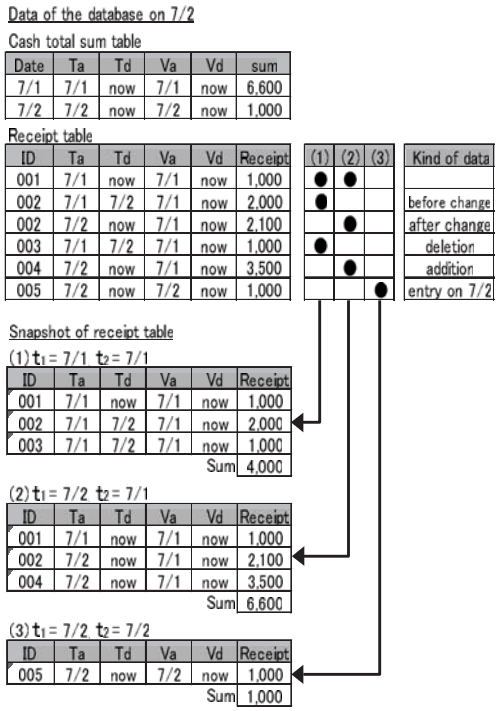


Figure 2: Snapshot of bitemporal database.

item (2) of Figure 2, we can query the corrected state of July 1st by designating t_1 July 2nd and t_2 July 1st in the condition of Equation (2): the change, deletion and addition on July 2nd are reflected; the receipt on July 2nd is not reflected. Incidentally, item (1) of figure 2 shows the query result of data before correction, the same as on July 1st in figure 1, and item (3) shows the query result of data entered on July 2nd.

Moreover, even while data are being corrected, we can get a snapshot that has integrity. The data corrections are also done by the usual online entry. Therefore, as shown in section 3.2, when we query the database by designating transaction time t_1, we can get the corrected result, which was performed by t_1. And, this query result isn't influenced by the online entry, including data corrections, ongoing at that time.

4 APPLICATION TO A MISSION-CRITICAL SYSTEM

We applied the bitemporal database to a local government system. In this section, we show the overview of this system, the implementation of the bitemporal database and the operation of its batch processing.

4.1 Overview of Local Government System

4.1.1 Composition of the system

The local government system is a mission-critical system for the public administration business of local government like a city hall. And, as shown in Figure 3, it consisted of various kinds of subsystems to assist the local government business.

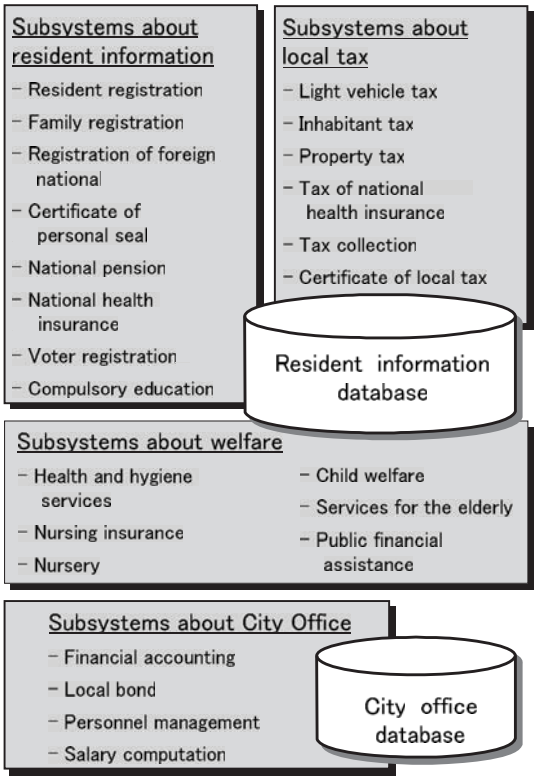


Figure 3: Composition of local government system.

They were classified by business contents as follows.

- **Subsystems about Resident information**
  They were used for the business, such as management and certificate of the residents who live in the city.
- **Subsystems about Local Tax**
  They were used for the business of the local tax, such as levy and certificate about tax.
- **Subsystems about Welfare**
  They were used for the business of welfare, such as qualification management, levy and grant.
- **Subsystems about City Office**
  They were used for the business of the office work of local government, such as personnel management, salary computation and financial accounting.

4.1.2 Characteristics of the Database

Each business needed the record data management in chronological order. We show the examples of the record data as follows.

- **Transfer of Resident**
  Each resident has his or her transfer records: they begin by birth or transfer into the city; via change of address, marriage and so on, they end by death or transfer to other city.

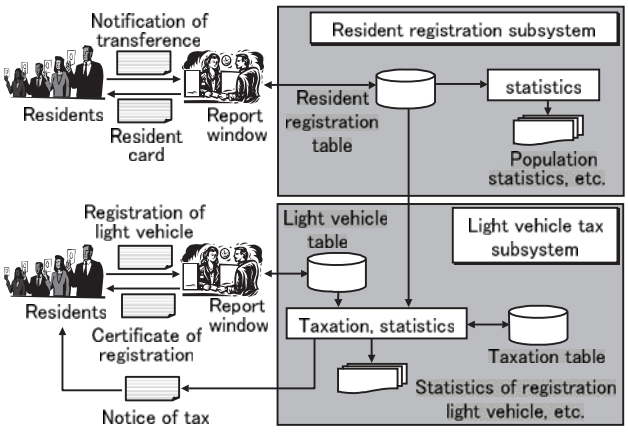


Figure 4: Dataflow example of local government system.

- **Taxable Article**
  The taxable article such as a light vehicle has the transfer records: its registration, transfer, disuse and so on.
- **Qualification of Welfare**
  The qualification of welfare has the acquisition and loss records, which are managed for the premium payment, the certificate of issuance, the insurance payment and so on.
- **Transfer and Diligence of Staff**
  The records of each staff of the local government were managed: his or her transfer, diligence, paid salary and so on.

Figure 4 shows the dataflow of the resident registration subsystem and the light vehicle tax subsystem, as the example of the data flow of the local government system. Notifications were accepted with the report window of the city hall, and its data were entered by the online entry and accumulated in the database to be queried by various processing of the system. And, the processing to query a great deal of data is processed by batch processing, such as making statistics, tax calculation, and so on.

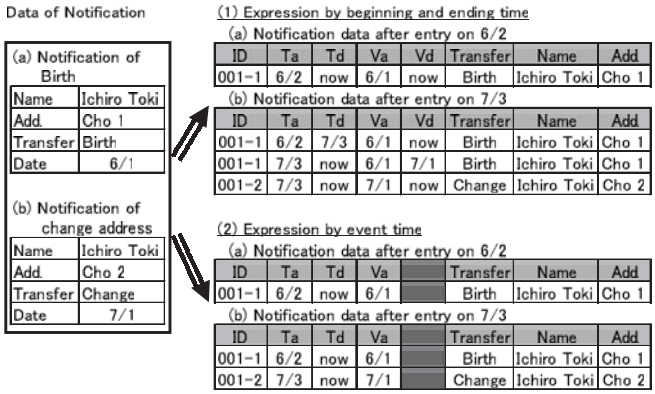
4.2 Implementation of Bitemporal Database

4.2.1 Policy of Implementation

We used the commercial relational database and added attributes of the transaction time and the valid time to each table, to compose a bitemporal database.

4.2.2 Implementation of Transaction Time

Since transaction time is used as one of primary key attributes of the database, the unit of transaction time had to be decided based on the frequency of data entry. In this system, data were entered from terminals, and the data entry took several seconds at least. So, we made the unit of transaction time 1 second.


Data of Notification

(a) Notification of Birth

Name	Ichiro Toki
Add.	Cho 1
Transfer	Birth
Date	6/1

(b) Notification of change address

Name	Ichiro Toki
Add.	Cho 2
Transfer	Change
Date	7/1

(1) Expression by beginning and ending time

(a) Notification data after entry on 6/2

ID	Ta	Td	Va	Vd	Transfer	Name	Add.
001-1	6/2	now	6/1	now	Birth	Ichiro Toki	Cho 1

(b) Notification data after entry on 7/3

ID	Ta	Td	Va	Vd	Transfer	Name	Add.
001-1	6/2	7/3	6/1	now	Birth	Ichiro Toki	Cho 1
001-1	7/3	now	6/1	7/1	Birth	Ichiro Toki	Cho 1
001-2	7/3	now	7/1	now	Change	Ichiro Toki	Cho 2

(2) Expression by event time

(a) Notification data after entry on 6/2

ID	Ta	Td	Va		Transfer	Name	Add.
001-1	6/2	now	6/1		Birth	Ichiro Toki	Cho 1

(b) Notification data after entry on 7/3

ID	Ta	Td	Va		Transfer	Name	Add.
001-1	6/2	now	6/1		Birth	Ichiro Toki	Cho 1
001-2	7/3	now	7/1		Change	Ichiro Toki	Cho 2

Figure 5: Expression of valid time.

4.2.3 Implementation of Valid Time

As for the valid time, because it depended on the business, we decided its unit from the necessity about the business. We show the examples of the unit of the valid time as follows.

- **A Minute:** the diligence of the office staffs.
- **A Day:** the transfer of residents, the transfer of taxable articles, the acquisition and loss of the qualifications of the welfare, the period of bank transfer and the transfer of the office staffs.
- **A Month:** the payment information of salary of the office staffs.
- **A Year:** the amounts of tax such as the light vehicle tax.

In the implementing of the valid time attribute, we used both expressions shown as follows.

- **Expression by Beginning and Ending Time**
  We expressed the valid time of some tables by the beginning and ending time: the table that needed to subscribe the time becoming valid or invalid beforehand such as the bank transfer period; the tables that needed to manage beginning and ending time by the set such as the acquisition and loss of the qualification.
- **Expression by Event Time**
  Data were changed or added by an event and maintained until the next event occurrence. For example, the state of a resident was changed by the event of transfer such as his or her birth or change of address, and the state was maintained until the next transfer.

When we changed a data expressed by the beginning and ending time, the following procedure was executed and two records are added as shown in item (1) of Figure 5. In Figure 5, T_a, T_d, V_a, V_d are the same as the notation in section 3.1.

- $ID = 001 - 1(T_a = 6/2)$: deletion time T_d is set to the original data.

- $ID = 001 - 1(T_a = 7/3)$: the data after change of the valid time is added, the ending time V_d of which is set to July 1st that is the beginning time of the next record.
- $ID = 001 - 2(T_a = 7/3)$: the data, address of which has been changed into 2-chome, is added.

Here, the data $ID = 001 - 1(T_a = 6/2)$ shows the record between June 2nd and July 2nd of the transaction time, and the data $ID = 001 - 1(T_a = 7/3)$ shows the record after July 3rd.

On the other hand, when we change a data expressed by the event time, only a record $ID = 001 - 2(T_a = 7/3)$ is added as shown in item (2) of Figure 5. Therefore, data increase in the case of the expression by the event time is less than the expression by the beginning and ending time. Incidentally, in this case, the state in the real-world of the designated valid time is expressed by the data that event time is eve of the designated time, if there is no data that event time agrees with it.

4.2.4 Support for Behind Entry Data

There was business that had to create documents like statistics at the end of business hours of the designated date, though the data of the real-world were behind in their notifications to enter the system. Therefore, we created them based on the notification date to the local government. For example, some kinds of resident transfers should be notified within 14 days from the actual transfer date: birth, change of address, transfer into the city and so on. However, statistics such as population statistics or transfer statistics of residents had to be created after the business end of the designated date to be reported to the next day. Therefore, we managed the notification date in addition to the transfer date that is a valid time in the table of the resident registration subsystem. The notification date is a user-defined time [7], [11], which is the time attribute defined by user in the temporal database.

Figure 6 shows the example of the snapshot by the notification date, in which N_a, N_d show the beginning and ending time of the notification date. Here, the ending time becomes the notification date of the next notification. That is, the time period attribute of the notification date is the same with the valid time, and we can query the database by the notification date in the same way with the valid time. Figure 6 shows the example of the data, notification date of which is June 2nd and was corrected on June 3rd. Item (1) of Figure 6 shows the query result designating both the notification date and the transaction time June 2nd, and its data are before the correction. And, Item (2) shows the query result designating notification date June 2nd like item (1) and the transaction time June 3rd, which reflected all kinds of corrections: the change of $ID = 002 - 1$, the deletion of $ID = 003 - 1$ and the addition of $ID = 004 - 1$.

4.3 Implementation of Online Entry

The notifications of the residents were accepted with the report window of the city hall, and its data were entered from the business screen of the online-terminals. Regarding business, this entry had the following characteristic.

- The simultaneous entry of the identical resident from more than one terminal could not happen in the general business, because the entry was done by the notification of the resident.
- The entry time at the report window was comparatively long to confirm the notification contents.

Therefore, we used the optimistic lock to reduce lock period, which used the addition time of the transaction time and executed by the following procedure.

i. The corresponding data is read from the table without acquiring a lock.

ii. The data is changed with the business screen.

iii. Once again, the data is read from the table with the record locking by the same condition as the last time.

iv. If it is being locked or its addition time is updated, the data is judged that has been updated by the others, so that the table is not updated. In the other case, the table is updated by the changed data.

4.4 Implementation of Data Correction

We added the data correction feature, which is shown as follows, to the business screen in addition to the data entry feature of the notification.

- **Correction as Internal Process**
  The corrections of the data entry error, in the entry data confirmation works of system operations, were done as the internal process of the local government.
- **Correction as Business Procedure**
  The other corrections were done as usual business procedure of the local government.

Data of the database

Data on 6/2 [Na, Nd]:User-defined time(Notification date)

ID	Ta	Td	Va	Vd	Na	Nd	Transfer	Name	Add
001-1	6/2	now	6/1	now	6/2	now	Birth	Taro Jisei	Machi 1
002-1	6/2	now	5/31	now	6/2	now	Into City	Jiro Henko	Machi 2
003-1	6/2	now	5/21	now	6/2	now	Birth	Sabro Sakujo	Machi 3

Data on 6/3

ID	Ta	Td	Va	Vd	Na	Nd	Transfer	Name	Add.	(1)	(2)
001-1	6/2	now	6/1	now	6/2	now	Birth	Taro Jisei	Machi 1	●	●
002-1	6/2	6/3	5/31	now	6/2	now	Into City	Jiro Henko	Machi 2	●	
002-1	6/3	now	5/31	now	6/2	now	Into City	Jiro Henko	Machi 5		●
003-1	6/2	6/3	5/21	now	6/2	now	Birth	Sabro Sakujo	Machi 3	●	
004-1	6/3	now	5/21	now	6/2	now	Into City	Shiro Tsuika	Machi 4		●

Snapshot of the database

(1) Transaction time=6/2, Notification date=6/2

ID	Ta	Td	Va	Vd	Na	Nd	Transfer	Name	Add
001-1	6/2	now	6/1	now	6/2	now	Birth	Taro Jisei	Machi 1
002-1	6/2	6/3	5/31	now	6/2	now	Into City	Jiro Henko	Machi 2
003-1	6/2	6/3	5/21	now	6/2	now	Birth	Sabro Sakujo	Machi 3

(2) Transaction time=6/3, Notification date=6/2

ID	Ta	Td	Va	Vd	Na	Nd	Transfer	Name	Add
001-1	6/2	now	6/1	now	6/2	now	Birth	Taro Jisei	Machi 1
002-1	6/3	now	5/31	now	6/2	now	Into City	Jiro Henko	Machi 5
004-1	6/3	now	5/21	now	6/2	now	Into City	Shiro Tsuika	Machi 4

Figure 6: Snapshot by user-defined time: notification date.

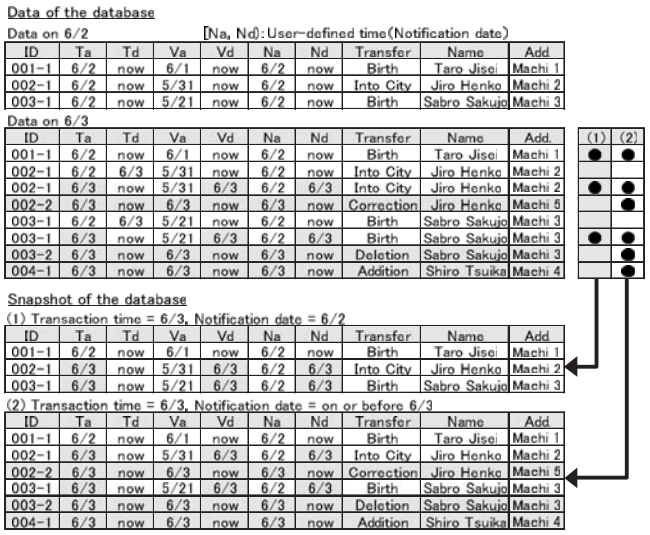


Figure 7: Correction as business procedure.

The correction by the internal process was executed with using the transaction time as shown in Figure 6. Item (2) of Figure 6 shows the query result of the notification that was notified on June 2nd and corrected on June 3rd, which was queried by designating the notification date June 2nd and the transaction time June 3rd. This correction was done as the internal process, so that it was not shown on the official documents such as the transfer record of the resident card.

On the other hand, the corrections as the business procedure have to be recorded on the official documents. For example, regarding the resident card, there were three kinds of corrections of business procedure: the official authority correction for the change, the official authority deletion for the deletion and the official authority mention for the addition. And, these corrections were recorded on the resident card.

Figure 7 shows the examples of the corrections done as the business procedure. Here, the examples are the same corrections that were done as the internal process in Figure 6. Item (1) of Figure 7 shows the snapshot queried by the same designating time as item (2) of Figure 6, which transaction time is June 3rd and notification date is June 2nd. For the corrections were done as the business procedure, the notification dates of them were June 3rd, and the correction result did not reflect in the query result. On the other hand, for the corrections were done with the notification date and their records were accumulated, both before and after correction records were queried by designating the both times as follows: the notification date was June 3rd, or before, and the transaction time was June 3rd. The relation of this snapshot is generally expressed as follows.

$$R_2(t_1, t_3) = \{r | r \in R \wedge r[T_a] \leq t_1 \wedge t_1 < r[T_d] \wedge r[N_a] \leq t_3\} \quad (3)$$

Here, t_1 and t_3 are the designated transaction time and notification date respectively, and $r[T_a]$, $r[T_d]$ and $r[N_a]$ are same as equation (2).

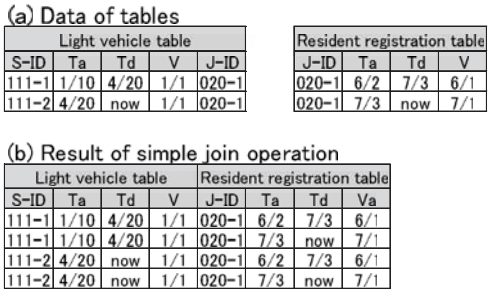


Figure 8: Problem of join operation on a bitemporal database

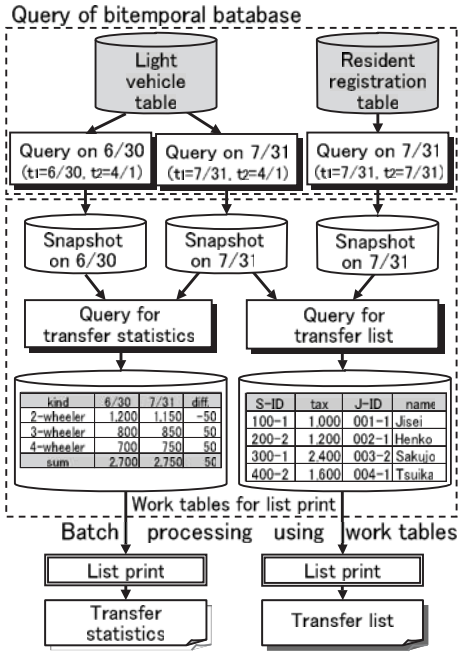


Figure 9: The composition of batch processing

4.5 Implementation of Batch Processing

Since the bitemporal database managed the records of the both time, the transaction time and the valid time, its query procedure became complicated. For example, as shown in Figure 8, the light vehicle table had the foreign key $J-ID$, which was the main key of the resident registration table. However, because each table had chronological records individually, the query result of join operation between them by $J-ID$ became the direct product about the time attributes. That is, the time attributes of these tables were not synchronous each other, so they could not be used as the key of join operation.

To solve this problem, we composed temporary files by the work tables in the application system, which are usually composed by the sequential access method (SAM) file. And, we processed data by the query function of the database, in the whole batch processing, to simplify each individual query procedure and maintain its performance. Specifically, as shown in Figure 9, the snapshot of each table was queried at

the beginning of the batch processing to output the result into the corresponding work table, and the following processing was performed by querying this work table.

Here, Figure 9 shows the example of the batch processing, which outputs were the transfer statistics of the light vehicle between June and July and the transfer list in July. Since the base date of the light vehicle tax was April 1st, each time of the snapshot was designated as follows: the transaction time $t_1 = 6/30$ and $7/31$, the last day of each month, and the valid time $t_2 = 4/1$, the base date. And, the snapshot of resident information for the transfer list was queried with designating both of the times on July 31st, which was its latest information. As a result, the query of this snapshot did not become complicated, because it was performed about each table individually. And, the join operations about the work tables did not become complicated, too, because it could be performed by the same way as a snapshot database.

4.6 Operation of Batch Processing

The data entered to the system was queried by various kinds of batch processing as shown in Figure 4. Regarding the designating time, the queries were divided into the following two kinds.

4.6.1 Query of Data at End of Business Hours

Various statistics such as the population statistics were created daily or monthly at the end of the business hours of the day, and their results were reported to the next day. In the conventional system operation, they had been made by the night batch processing after the business hours of the report window. In this application system, we could execute the batch processing during the business hours of the next day and so on, because we queried the data of the end of business hours by designating the transaction time. As a result, we could reduce the night batch processing.

In the batch processing, query results were checked by the first step. And, when error data were detected, the batch was redone after the data correction. Regarding the batch processing, all kinds of the corrections reflected in the query result, as shown in Figure 6: the change, deletion and addition. Here, as for the data that took time until its notification to the local government, they were queried by using its notification date as shown in section 4.2.4.

4.6.2 Query by Designating Valid Time

In the taxation processing, the base date of the taxation was designated and the taxation was done after this date. However, corrections of the taxation about the delay of the notification often occurred, because the transfer notifications of taxable articles to the local government usually took time.

In the case of the light vehicle tax, the taxation processing had been usually executed with the base date of April 1st, and since then, its correction processing was regularly performed.

Data of the database

ID	Ta	Td	Va	Vd	Tax	(1)	(2)	(3)	Correct
001-1	1/10	now	1/1	now	1,000	●	●	●	
002-1	1/10	4/20	1/1	now	1,200	●			
002-1	4/20	now	1/1	3/31	0				
002-2	4/20	now	4/1	now	1,600		●	●	Change
003-1	1/10	5/20	1/1	now	5,200	●	●		
003-1	5/20	now	1/1	3/31	0				
003-2	5/20	now	4/1	now	7,200			●	Change
004-1	1/10	4/20	1/1	now	2,500	●			
004-1	4/20	now	1/1	3/31	0				Deletion
005-1	1/10	5/20	1/1	now	4,000	●	●		
005-1	5/20	now	1/1	3/31	0				Deletion
006-1	4/20	now	1/1	now	7,200		●	●	Addition
007-1	5/20	now	1/1	now	3,000			●	Addition

Snapshot of the database

(1) Transaction time = 4/15, Valid time = 4/1

ID	Ta	Td	Va	Vd	Tax
001-1	1/10	now	1/1	now	1,000
002-1	1/10	4/20	1/1	now	1,200
003-1	1/10	5/20	1/1	now	5,200
004-1	1/10	4/20	1/1	now	2,500
005-1	1/10	5/20	1/1	now	4,000
				Sum	13,900

(2) Transaction time = 5/15, Valid time = 4/1

ID	Ta	Td	Va	Vd	Tax
001-1	1/10	now	1/1	now	1,000
002-2	4/20	now	4/1	now	1,600
003-1	1/10	5/20	1/1	now	5,200
005-1	1/10	5/20	1/1	now	4,000
006-1	4/20	now	1/1	now	7,200
				Sum	19,000

(3) Transaction time = 6/15, Valid time = 4/1

ID	Ta	Td	Va	Vd	Tax
001-1	1/10	now	1/1	now	1,000
002-2	4/20	now	4/1	now	1,600
003-2	5/20	now	4/1	now	7,200
006-1	4/20	now	1/1	now	7,200
007-1	5/20	now	1/1	now	3,000
				Sum	20,000

Figure 10: Query Results by designating valid time.

Figure 10 shows the example, in which the taxation processing had been executed on April 15th and its correction processing was performed every month since then. We queried by designating the valid time the base date of April 1st and the transaction time the taxation processing date or the correction processing date. By this condition, we could query the result, in which the corrections done until each processing date were reflected.

5 EVALUATION AND CONSIDERATION

5.1 Evaluation about Query of Correction Data

5.1.1 Query of Data at End of Business Hours

Even if error data were detected in the batch processing, it became possible to redo the processing after correcting these error data, during the online entry of the usual business data. In the online entry, the correction of entry data could be performed as the internal process by using the transaction time, in addition to the usual correction done as the business procedure. The integrity of the query result was maintained about both of these data corrections.

Since the queries in batch processing became to be able to execute during the online entry, the night batch processing

and its attendant work such as the waiting for the online entry beyond business hours was reduced. Therefore, the system operation load as the overtime work could be reduced. Moreover, when several batch processing overlaps on the same day, it was enabled to make the schedule of batch processing flexibly, in which, for example, the low processing of priority was executed later such as the next day.

Here, the fact in the real-world wasn't always reflected in the system instantly like the resident's transfer as shown in section 4.2.4. So, as for the statistics, which were created by the data of the end of business hours on the designated date to be reported to the next day, we queried by using not their valid time but their notification date that is the user-defined time. In this way, we could query the state of the data of the end of business hours on the designated date even if here are notification backlogs.

5.1.2 Query of Correction for a Long Period

Like the taxation processing as shown in Figure 10, some business needed to manage its correction for a long period and grasp the status of the correction in the transfer statistics. Regarding the business like this, we could also query the result that reflected all kinds of corrections, which included the change, deletion and addition, done until the designating date. Even in this case, we could easily create the transfer statistics by querying the total in the last time and this time, and the transfer total between the both times. Moreover, it could be executed during the online entry, because this query procedure used the snapshot, too.

5.1.3 Management of Correction Data

The online entry data were confirmed and corrected before they were used by the business procedure. Regarding the business that managed the records such as the resident registration, the correction records in this step did not be needed by the business procedure. So, these corrections were performed as the internal process of the system operations. On the other hand, after the data were used by the business procedure, their corrections also had to be done as the business procedure. Both of these corrections, as the internal process and as business procedure, could be managed individually by the bitemporal database.

5.2 Evaluation of Implementation of Database

In the record management of the bitemporal database, the problems about the increase of the amount of data and the complication of the query procedure occurred.

About the amount of data, two records needed to add the database for each change in the real-world as shown in item (1) of Figure 5: one of them was the record of the before, of which ending time of the valid time was updated, and another was the record of the after. We could reduce these two records to one record by the method that used the expression by the event time as shown in item (2) of Figure 5. Therefore, we adopted the expression by the event time to the tables except the tables that needed to manage the ending time of the valid time in particular.

As a result, the increase of records to the amount records needed in the original business became almost only the correction records as the internal process. Regarding this application system, this increase was 20% in a year even if maximum and the amount of data was less than twice using for 5 years, the life cycle of the system. Incidentally, in recent years, the price of the unit capacity of the storage media was falling and the increase of the amount of data didn't become a problem in the aspect of the system building cost.

Next, to simplify each query procedure, we used the work tables as temporary files of the batch processing, and processed data step-by-step. As a result, we achieved the necessary query performance for the business, in the function range of the commercial relational database. Incidentally, since inserting a great deal of data into the work table with indexes degrades the performance, we generated indexes after inserting or importing of data in this case.

5.3 Consideration

5.3.1 Query of Data at End of Business Hours

We confirmed that the corrections of data were reflected in the query result and its integrity was maintained by the bitemporal database, even while the online entry was performed. Moreover, we confirmed that this is effective for the system operation, because the night batch processing to query the data of the end of business hours on the designated date could be reduced by executing it in the business hours of the next day and so on.

For example, the batch processing and its attendant work of the overtime of the local government, having about 40,000 populations, could be reduced about 1.5 hours a day from the conventional system. In addition, for the system operation in the overtime was ridded of, the batch processing could be transferred from the system department to the control department of the business. As a result, the work of contact and adjustment, such as the request of the batch processing or the receipt of the documents of it, could be reduced. In recent years, the nonstop services are expanding with the internet application such as the electronic government, the electronic commerce. Therefore, we consider that the operation to query by batch processing without stopping the online entry is effective in such a field.

Moreover, we found that the notification date, which is user-defined time, should be managed apart from the valid time in some kinds of business, because, in the actual business operation, all the fact of the real-world may not reflect in the system instantly. For example, in the resident registration subsystem, both times were used according to the business contents: the age calculation of the resident used the birthday that is the valid time; the statistics of the residents used the notification date of the transfer that is the user-defined time.

5.3.2 Query of Correction for a Long Period

The data, which were correcting for a long period, had to be managed by its records of correction from the base time of the business until the query time. The bitemporal database manages both the times: the valid time that expresses the base time and the transaction time expresses the query time. We confirmed that this is effective such as to grasp the records of data for a long period and to create the transfer statistics.

5.3.3 Management of Correction Data

The bitemporal databases could manage correction records about the both of the correction: as the internal process and as the business procedure. The correction as the internal process didn't accumulate records for business procedure, but these correction records were accumulated inside the database by using the transaction time. Therefore, we considered that these records are effective for the management of the change process of data in the system, especially in the mission-critical systems that need to adapt inspection of a high level.

6 CONCLUSION

For the efficient operations of mission-critical systems, it is necessary both batch processing and online entry can be executed simultaneously even under the actual system operation such as data-entry mistakes and data-entry backlogs. We applied a bitemporal database into a mission-critical system, and throughout its actual system operations, we confirmed that the data corrections were reflected in the query results, and that the integrity of the query results was maintained even while the database was being updated by online entry. Furthermore, we confirmed that it is effective about not only the data corrections for a short period confirmation work but also the correction management of data for a long period of real-world use, and that data correction can be managed in both internal processes and business procedures individually. In recent years, the nonstop services are expanding with the development of internet applications. Therefore, we consider that the system operation being able to query by batch processing without stopping the online entry become effective in such a field.

REFERENCES

[1] L. Bækgaard, and L. Mark, Incremental Computation of Time-Varying Query Expressions, IEEE Trans. knowledge and Data Eng., Vol. 7, No. 4, pp. 583–590 (1995).
[2] G. Bhargava, and S. K. Gadia, Relational Database Systems with Zero Information Loss, IEEE Trans. knowledge and Data Eng., Vol. 5, No. 1, pp. 76–87 (1993).
[3] N. Edelweiss, P. N. Hübler, M. M. Moro, and G. Demartini, A Temporal Database Management System Implemented on top of a Conventional Database, Proc. XX International Conference of the Chilean Computer Science Society, pp. 58–67 (2000).
[4] J. Gray, and A. Reuter, Transaction Processing: Concept and Techniques, Morgan Kaufmann (1992)
[5] C. S. Jensen, L. Mark, and N. Roussopoulos, Incremental Implementation Model for Relational Database with Transaction Time, IEEE Trans. knowledge and Data Eng., Vol. 3, No. 4, pp. 461–473 (1991).
[6] C. S. Jensen, and R. T. Snodgrass, Temporal Data Management, IEEE Trans. knowledge and Data Eng., Vol. 11, No. 1, pp. 36–44 (1999).
[7] G. Özsoyoğlu and R. T. Snodgrass, Temporal and Real-Time Databases: A Suevey, IEEE Trans. knowledge and Data Eng., Vol. 7, No. 4, pp. 513–532 (1995).
[8] H. -J. Park, and S. I. Yoo, Implementation of Check-out/Checkin Mechanism on object-Oriented Database System, Proc. 7th International workshop on Database and Expert System Applications, pp. 298–303 (1996).
[9] L. Shrira, and H. Xu, SNAP: Efficient Snapshots for Back-in-Time Execution, Proc. 21st International Conference on Data Engineering, pp. 434–445 (2005).
[10] H. Skaf, F. Charoy, and C. Godart, Flexible Integrity Control of Cooperative Applications, 9th International Workshop on Database and Expert Systems Applications, pp. 901–906 (1998).
[11] R. Snodgrass and I. Ahn, Temporal Databases, IEEE COMPUTER, Vol. 19, No. 9, pp. 35–42 (1986).
[12] B. Stantic, J. Thornton, and A. Sattar, A Novel Approach to Model NOW in Temporal Databases, Proc. 10th International Symposium on Temporal Representation and Reasoning and Fourth International Conference on Temporal Logic, pp. 174–180 (2003).
[13] A. U. Tansel, Integrity Constraints in Temporal Relational Databases Extended Abstract, Proc. Int. Conf. on Information Technology:Coding and Computing, pp. 460–464 (2004).
[14] P. Terenziani, Symbolic User-Defined Periodicity in Temporal Relational Databases, IEEE Trans. Knowledge and Data Eng., Vol. 15, No. 2, pp. 489–509 (2003).

(Received September 9, 2007)
(Revised November 3, 2008)

Tsukasa Kudou Tsukasa Kudou received the master's degree in Engineering from Hokkaido University in 1980 and received the Dr. degree in industrial science and engineering from Shizuoka University, Japan, in 2008. In 1980, he joined Mitsubishi Electric Corp. He was the researcher of parallel computer architecture, the engineer of application packaged software and business information system. Since 2005 he is the engineer of Mitsubishi Electric Information Systems Corp. Now, his research interests include database application and software engineering. He is a member of IEIEC and Information Processing Society of Japan.

Masahiko Ishino received the master's degree in science and technology from Keio University in 1979 and received the Ph.D. degree in industrial science and engineering from graduate school of Science and technology of Shizuoka University, Japan, in 2007. In 1979, he joined Mitsubishi Electric Corp. Since 2005, he is the system engineer of Mitsubishi Electric Technology Corp. Now, His research interests include management information system, industrial engineering, application system of data-mining, and information security system. He is a member of Information Processing Society of Japan, Japan Industrial Management Association and Japan Society for Management Information.

Kenji Saotome received a BE from the Osaka University, Japan in 1979, and a Dr.Eng in Information Engineering from the Shizuoka University, Japan in 2008. From 1979 to 2007, he was with Mitsubishi Electric, Japan. Since 2004, he has been a professor of Hosei business school of innovation management. His current research areas include LDAP directory applications and single sign-on system. He is a member of the Information Processing Society of Japan.

Nobuhiro Kataoka received the master's degree in Engineering from Osaka University in 1968 and received the Ph.D. degree in Information Science from graduate school of Information Science Tohoku University, Japan, in 2000. In 1968, he joined Mitsubishi Electric Corp. He was the engineer of basic software, middle software and application packaged software. From 1992, he was transferred to corporate information management department and managed to expand ERP (SAP) all over the corporation. Since 2000 he is a Professor of Tokai University. Now, His research interests include management information system and business process modeling. He is a member of IEEE, IEIEC, and Information Processing Society of Japan.

Tadanori Mizuno received the B.E. degree in industrial engineering from the Nagoya Institute of Technology in 1968 and received the Ph.D. degree in computer science from Kyushu University, Japan, in 1987. In 1968, he joined Mitsubishi Electric Corp. Since 1993, he is a Professor of Shizuoka University, Japan. Now, he is a Professor of graduate school of Science and technology of Shizuoka University. His research interests include mobile computing, distributed computing, computer networks, broadcast communication and computing, and protocol engineering. He is a member of Information Processing Society of Japan, the institute of electronics, information and Communication Engineers, the IEEE Computer Society, ACM and Informatics Society.