American Journal of Database Theory and Application 2012, 1(2): 13-25
DOI: 10.5923/j.database.20120102.02

Virtual Database Technology for Distributed Database in Ubiquitous Computing Environment

Yuji Wada[1,*], Yuta Watanabe[1], Keisuke Syoubu[1], Hiroshi Miida[1], Jun Sawamoto[2]

[1]Department of Information Environment, Tokyo Denki University, Inzai, Chiba, 247-1382, Japan
[2]Faculty of Software and Information Science, Iwate Prefectural University, Takizawa, Iwate, 020-0193, Japan

Abstract In this paper, our research objective is to develop a database virtualization technique in order to let data analysts or other users who apply data mining methods to their jobs use all ubiquitous databases on the Internet as if they were recognized as a single database, thereby helping to reduce their workloads such as data collection from the Internet databases and data cleansing works. In this study, firstly we examine XML schema advantages and propose a database virtualization method by which such ubiquitous databases as relational databases, object-oriented databases, and XML databases are accessed as if they all behave as a single database. Then, we show the method of virtualization of ubiquitous databases that can describe ubiquitous database schema in a unified fashion using the XML schema. It consists of a high-level concept of distributed database management of the same type and of different types, and also of a location transparency feature. Next, we discuss about the development of a common schema generation method and propose the virtual database query language for use in the virtualized ubiquitous database environment. Finally, we propose a database incompatibility trouble-recovery technique for use in a virtualized ubiquitous database environment.

Keywords Database Virtualization, Data Mining, XML Schema, Ubiquitous Databases, Database Integration, Database Query, Database Recovery

1. Introduction

Nowadays, massive amounts of data are collected daily in ubiquitous sensor network environments. With such data available and elaborately structured, it is more important than ever to locate and extract knowledge and trends from them using data mining techniques. For example those data are valuable to support analyses and decision-making in businesses. Such data normally exist in databases of various types—called ubiquitous databases hereinafter—that might usually be distributed and placed anywhere. A salient problem, however, is that a person who engages in data mining using ubiquitous databases would have to spend much time for database selection and data collection which would be merely a preparatory step to the actual data mining tasks. What a person really should want must be instead to concentrate on the work of analysis and rule extraction.

In our study, the primary objective is therefore to develop a virtualization technique in order that the data analyst or other users can use all ubiquitous databases as if they were recognized as a single database, thereby helping to reduce the user's workload. As we proceed with our study, we must also find a way to troubleshoot and recover from any ubiquitous database troubles. It is necessary that a data recovery technique be provided for use when a ubiquitous database encounters a trouble. However, an important shortcoming of database virtualization techniques is that when such a system encounters a trouble, some incompatibility might occur between the virtual database and the associated real databases. Recovering the real database alone would not be sufficient. It would only recover the data in it, leaving the unresolved incompatibility between the virtual database and the associated real databases.

For this reason, the secondary objective of our study is to develop a database incompatibility trouble-recovery technique in a virtualized ubiquitous database environment.

2. Related Research

Some earlier reports in[1],[2] and[3] have described the study of database virtualization technology.

The report[1] proposed development of a system to disseminate information actively to all users in a mobile computing environment without fail, as sourced from various types of database groups connected by a wide-area network. By image-copying of the data of the local database group to a meta-database through the basic search and build

* Corresponding author:
yujiwada@sie.dendai.ac.jp (Yuji Wada)
Published online at http://journal.sapub.org/xxx

operations, for example, it is intended to combine data and include different types of local database group.

The data integration technique, teiid, which is described in[2], enables virtualization of various types of databases; through such virtual databases, one can access such data sources as relational databases, web databases, and application software such as ERP and CRM, etc. in real time. They can all be integrated for use. In fact, teiid has a unique query engine. Furthermore, the real-time data integration is accomplished by connecting business application software through the JDBC/SOAP access layer with data sources which are accessed through the connector framework.

Reference[3] similarly describes a module known as a wrapper that allows accessing and integrating data from various sources such as RDBs, the Web, and Excel files.

In our study, we considered the metadata, UML, E–R model, and the XML schema as candidates for use to accomplish database virtualization. Thereby, ubiquitous databases can be used as if they were a single database. We then compared the advantages and disadvantages of each to analyse them as follows.

(1) The use of metadata presents many advantages for creation that are irrelevant to what database model the metadata are based on. On the other hand, an important disadvantage is that they require a great workload to create them in their initial stage. Moreover, no definition and manipulation language to manage metadata has been standardized yet.

(2) The UML and the E–R model have similar fundamental characteristics; each has an advantage that its database design concept structure is irrelevant to what data model it is based on and with what DBMS product it is associated. However, those are only a few design techniques. No specific DBMS and definition and manipulation language are provided.

The matters described in (1) and (2) above subsume a structured static schema for their use. Therefore, they have a difficulty in use with databases of various kinds that are available on the Internet in a flexible fashion in[4, 5].

(3) The XML schema is now widely used to exchange information in the Internet environment. Additionally, now that more studies and further developments of XML database management systems are made than ever before, it is advantageous to use it because its definitions and manipulations are well standardized in[6]. However, from a data model perspective, it presents the problem that it does not go well with any object-oriented data model that is now associated with multimedia. Even given that fact, now that the object-oriented concept is incorporated into the extended standardization of the SQL language, it has been improved in its affinity level with the associated XML schema being converted to a relational database schema. In addition to that, because the XML schema is a semi-structured dynamic one, it is still advantageous because of the fact that it can be used in a flexible fashion with various databases available on the Internet.

In the relevant literature, several studies[7–11] of the XML schema conversion have been reported. One of those reports[7] proposes an XML-to-relational mapping framework and system that provides the first comprehensive and end-to-end solution to the relational storage of XML data. Another report[8] offers a flexible mechanism for modifying and querying database contents using only valid XML documents, which are validated over the XML-Schema file's rules. Report[9] proposes cost-based XML storage mapping engine and explores the space of possible XML-to-relational mappings and selects the best mapping for a given application. Report[10] describes the integration of XML with a relational database system to enable the storage, retrieval, and update of XML documents. A common data model based on XML has been introduced and schema mapping based on that approach has been presented in[11].

In our previous studies[12-15], we examined XML schema advantages and proposed a virtualization method by which such ubiquitous databases as relational databases, object-oriented databases, and XML databases are usable, as if they all behaved as a single database.

On the other hand, studies of recovery techniques from database trouble are now well underway at a practical level with respect to centralized databases or distributed databases, and are widely used in the field of Online Transaction Processing (OLTP)[4],[5]. However, few practical studies have been undertaken in environments associated with database virtualization.

Therefore, we propose a means to recover the associated databases by allowing users to examine the virtual environment only for ubiquitous databases without having to examine real databases, and to ensure the integrity between a virtual database and an associated real database.

In summary, the method of virtualization of ubiquitous databases proposed in our study describes ubiquitous database schema in a unified fashion using the XML schema. Moreover, it consists of a high-level concept of distributed database management of the same type and of different types, and also of a location transparency feature, so that it has the important capability of recovering from database trouble, even if solely within a virtual environment.

In the following Chapter 3, we propose the virtualization of ubiquitous databases. In Chapter 4, we describe the virtual database definition, such as common schema generation and schema conversion. In Chapter5, we show the virtual database query for use in a virtualized ubiquitous database use environment. In Chapter 6, we discuss the verification for data mining application. Finally, in Chapter 7, we address the feature of recovery from ubiquitous database trouble while we are in a virtual environment.

3. Database Virtualization

Databases of many kinds exist in terms of their associated data model differences and vendor differences. Regarding differences among data models, each has different data representation, and unique associated manipulation. Some

typical examples include the table type of relational databases (RDB), XML-representation type of XML databases (XMLDB), and object-oriented databases (OODB). Even the same model database might have different features among vendors. Regarding RDB for example, there might be some differences in SQL and/or data type representation. The typical example is that we have MySQL, PostgreSQL, and SQLServer from different vendors.

These differences according to the model and vendor bring some undesired results. For example, we might end up spending more time and labour during application system development because of the different data models that must be confronted. For example, we might need to acquire the right API to handle data of every different type of database. Virtualization of such different types of modelled databases to unify the procedures for all of them would probably impart less of workload and cost, and facilitate their management in a more flexible manner. Consequently, virtualization of databases, if it could be done, would facilitate application system design and database management as well.

To have a virtualization feature, we will consider the inclusion of features to manage distributed databases of similar types, the distributed databases of different types, and provide location transparency for users, such that they notice no differences of database structure or location and become able to use databases of all kinds in a flexible fashion. Figure 1 portrays a comprehensive view of the database virtualization technique.

For virtualization of ubiquitous databases in our study, we will describe the schema information of the real databases, of which more than one always happens to exist, by creating and using one common XML schema. We also provide functionality of data search and update with the XML-based common data manipulation API.

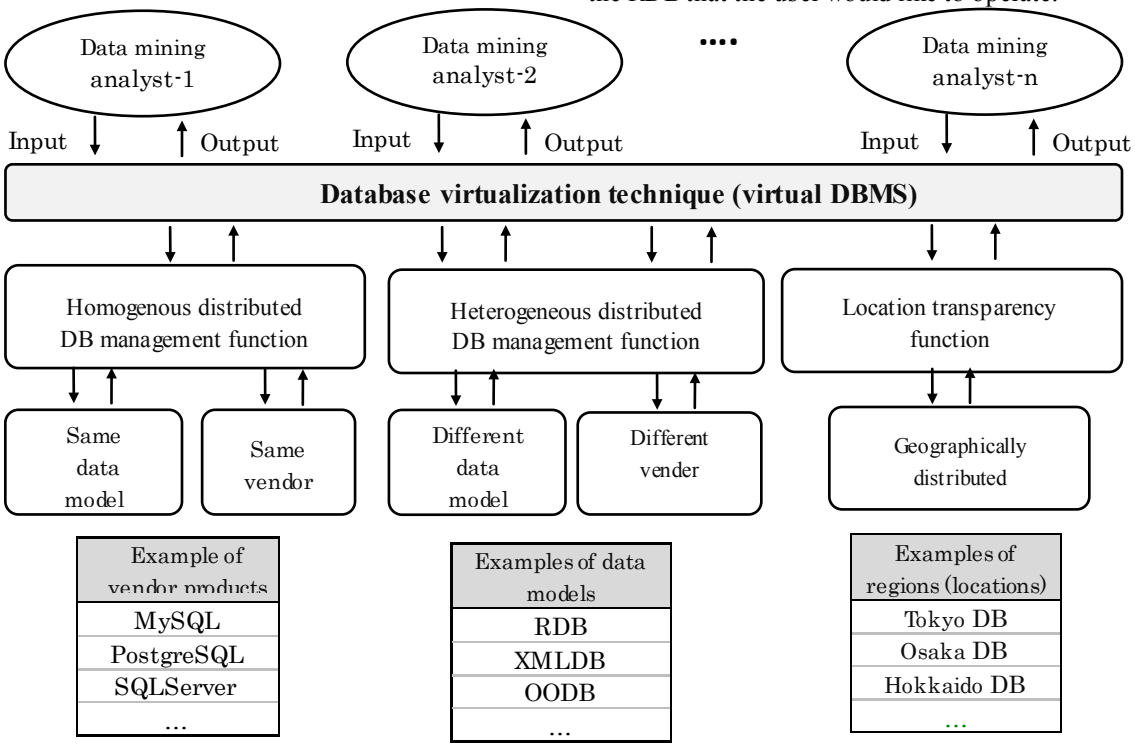
3.1. Visualization of Homogeneously Distributed Databases

Described next, as the first step of database virtualization, is a method of building a virtual database management system for RDBs provided by different vendors.

3.1.1. XML Conversion Program

Considering virtualization not only for the RDB, but also for the different data models such as XML DB and object-oriented DB, which will be required in the next phase, we will use an XML schema that provides a flexible representation capability and a high transparency capability.

To do so, we will produce such a virtualization concept in which the user would feel as if he or she were locally manipulating the remote-site RDB from a local RDB process environment. That can be accomplished by converting the schema information and data information of the local RDB into the XML schema, and then storing that information into the RDB that the user would like to operate.



MySQL: Oracle(Sun Microsystems)
PostgreSQL:PostgreSQL Global Development Group
DQLServer:Microsoft

Figure 1. Database virtualization technique

We developed an XML conversion program, XMLExport/Import, as depicted in Figure 2. We then used such different vendor RDBs as MySQL, PostgreSQL, and SQLServer2005 because they are available in the RDB virtualization system creation environment. We used PHP as the programming language to develop an XML Export/Import system. We have to rebuild the XML tree with our XML conversion program when the distributed database is redefined.

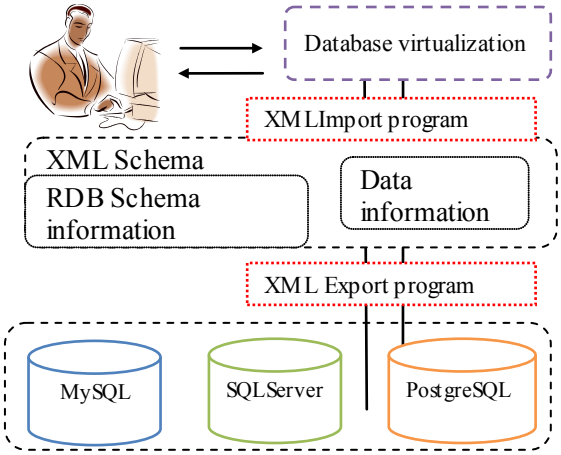


Figure 2. Virtualization technique for homogeneous distributed databases

3.1.2. RDB Schema Conversion into XML

```
<?xml version="1.0" encoding"UTF-8" standalone="yes"
 <root>
    <rdb Name="mysql">
       <database Name="questionnaire" >
       <table_structure Name="member">
             <field Field="samplenum"
              Type="integer" Null="FALSE"   Default="   />
    <field Field="answerday" Type="text"
     Null="FALSE"   Default="   />
       ….
    </table_structure>
 <schema>
      <constraint Type="PRIMARY KEY"
       Table="member"   Column="samplenum"   />
….
      <constraint Type="FOREIN KEY" Table="questionnaire"
      Column="samplenum" Retable="member"
      ReColumn="samplenum" />
….
  </schema>
</database>
</rdb>
</root>
```

Figure 3. Example of RDB schema information conversion into XML

The following describes how the RDB schema is converted into XML.

Figure 3 presents results of reading the schema information from the RDB and converting it into XML. The RDB schema information that is converted into an XML format includes "table names", "field names (associated data types and default values)", and "constraints (primary key constraint, unique constraint, check constraint, NOT NULL constraint, and foreign key constraint)". capability.

Regarding the XML tree structure, we described the table information in the table_structure node with its elements of Field="column name", Type="data type", Null="TRUE or FALSE" (NOT NULL constraint), as shown in Figure 3. We described the schema information in the schema node with its elements of TYPE="constraint name", Table="table name", Column="column name", ReTable="referenced table name", ReColumn="referenced column name", and Check="rule".

3.1.3. RDB Data Conversion into XML

The manner in which the RDB data are converted into XML is described next. Figure 4 portrays results of reading the data information from the RDB and conversion into XML. Because of the XML tree structure, we had dbname="database name", tblname="table name", and the succeeding column name="actual data".

```
<root>
  <dataset dbname="mysql">
     <data tblname="member" samplenum="10001"
       answerday=""2007/7/6""
       answertime="" 13:07:19:499"" />
    <data tblname="member" samplenum="10002"
       answerday=""2007/7/6""
       answertime="" 13:10:33:507"" />
        ….
 </dataset>
</root>
```

Figure 4. Example of actual RDB data conversion into XML

3.2. Virtualization of Heterogeneously Distributed Databases

The second step we discuss is the virtualization of modelled DBs of different types. For virtualization of different types of modelled DB, we describe the schema information of each model using a single common schema. The common schema we will use is an XML Schema. Around it, we will perform virtualization. Figure 5 presents a virtualization method for different database types.

To accomplish schema conversion from a different modelled database, we first get the schema information from an RDB to work on. Then we convert it into the correct XML schema for that RDB. We currently have to rebuild the XML tree with our schema conversion module in Figure 5 when the distributed database is redefined.

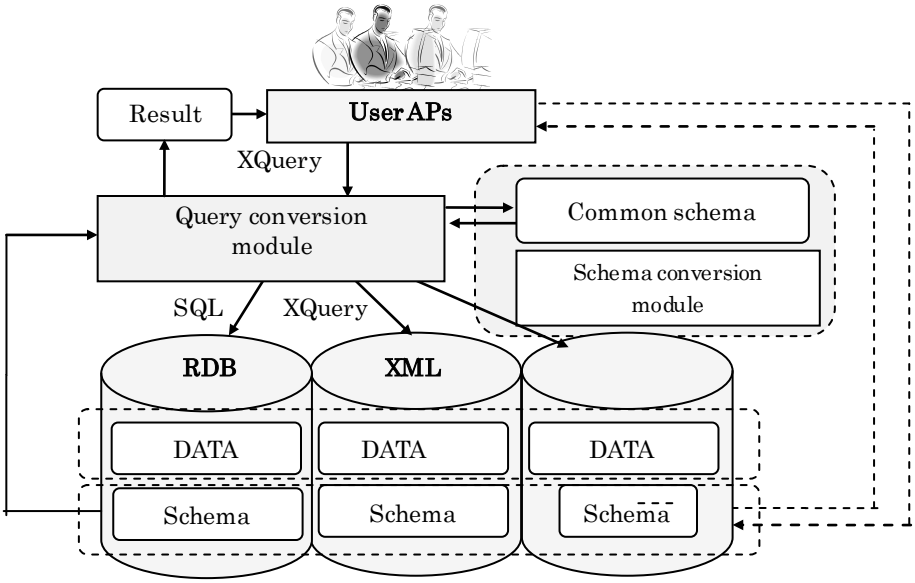


Figure 5. Virtualization technology for heterogeneous distributed databases

Table 1 presents schema conversion correspondences between the two. Because any XML DB is already described in the XML format, we extract the schema information without conversion. For an OODB, we will use the object-oriented functionality that is available because it is fundamentally an extension of RDB.

On the other hand, when the data are manipulated, our query conversion module in Figure 5 automatically transfers the access results to the application program.

Table 1. SQL and associated XML

	SQL	XML
Table definition	CREATE TABLE table name…	<xsd: element name=”table name”…
Column definition	CREATE TABLE… column name...	<xsd: element name=”column name”…
Data type definition	CREATE TABLE… data type..	<xsd: element… type=”data type”…
Default values	CREATE TABLE… column name DEFAULT value	<xsd: element… default=”value”…
Primary key constraint	PRIMARY KEY	<xsd: key…
Unique constraint	UNIQUE	<xsd:unique …
Foreign key constraint	FOREIGN KEY	<xsd: keyref … refer =…
NOT NULL	NOT NULL	<xsd:… nillable=”false”...
Method	CREATE METHOD	
Inheritance	CREATE TABLE… UNDER upper level table name	<xsd: complexType …

4. Virtual Database Schema Definition

4.1. Common Schema Structure

A common schema provides the user with information about the structure of the virtual database, and is also used for checking query statements and constraints. Users see only a single common schema of all distributed databases. As an example, Figure 6 shows the structure expressed by a common schema for a RDB and XMLDB.

The root element is located at the top, and the level below contains elements indicating various types of databases, for example, DB1 and DB2. The elements of the next lower level contain, in the case of a RDB, the database name, and in the case of a XMLDB, the collection path. Then, stored in the next lower level, in the case of a RDB, is an element indicating the table structure, and in the case of a XMLDB, the document structure. The RDB table structure first has an element indicating the table name, followed by the “row” elements listed below. The next lower level contains an element indicating the column name.

Figure 7 shows the structure, expressed in XML, of DB1 in Figure 6. The table name “Employee information” is used as the root element, and below, data is entered into the columns of “Employee ID”, “Name”, “DOB” (Date of Birth) and “Sex” enclosed by “row” elements.

The locations enclosed within this “row” form one row of the table. Similarly, data enclosed by “row” elements of the second and subsequent rows are arranged as sub-elements of the element indicating the table name. Constraint information is provided using <xsd:annotation> and <xsd:appInfo> as information-oriented comments. The main details regarding the outputting of constraints are as follows. (Primary key (unique) constraint: < r:index index-key=”column name” primary(unique)=”yes”>, external key constraint: <f:key fkey-column=”column name” ref-column=”reference column” ref-table=”reference table” rule-delete="operation at delete” rule-update="operation at update" />,check constraint: <r:check check-column=” column name” rule=”rule”>)

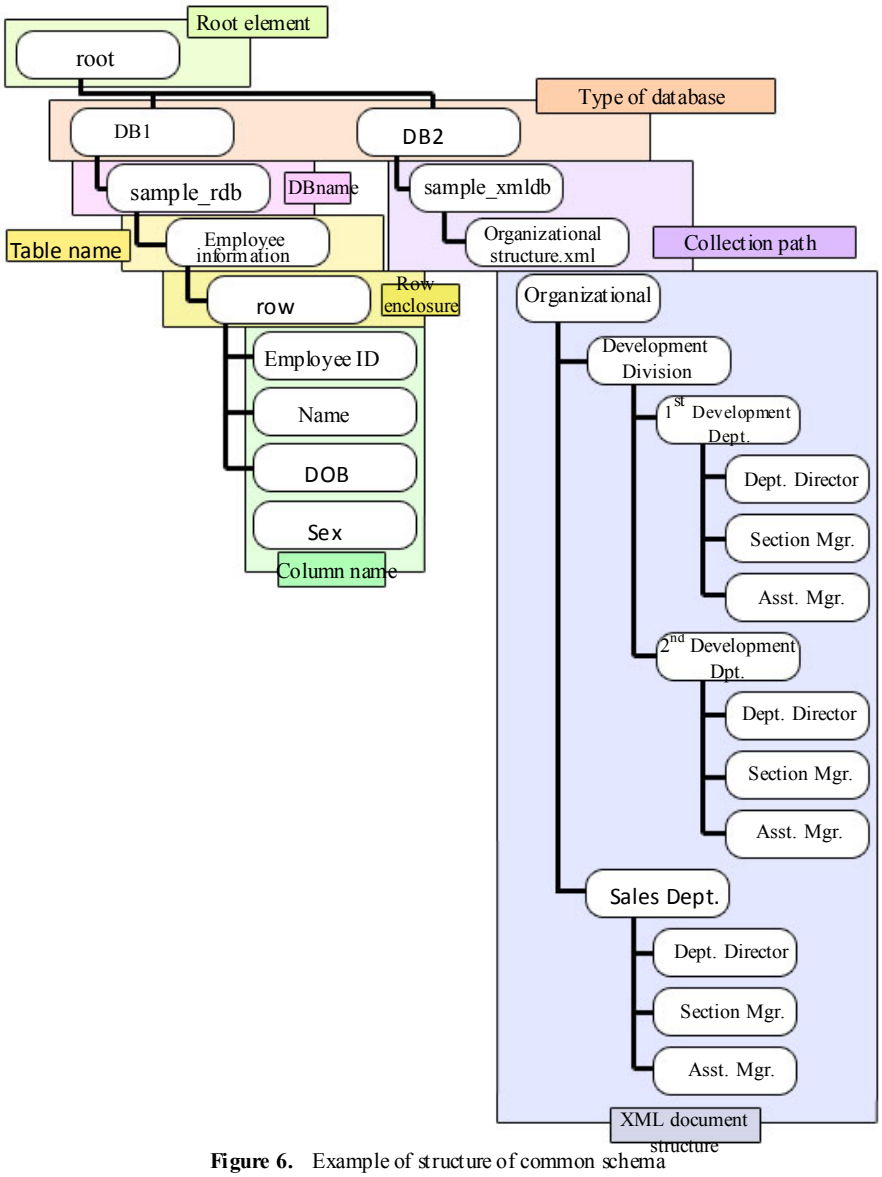


Figure 6. Example of structure of common schema

4.2. Common Schema Creation

Here, as a sample schema used in this operation example, a database containing the following information is created and an XML schema is then constructed using PostgreSQL. Similarly, to verify the basic constraints, in a column, the primary key constraint is assigned to "Employee ID, Affiliation ID", the unique constraint is assigned to "Affiliation", the external key constraint is assigned to "Affiliation ID", the check constraint is assigned to "salary", and the "NotNull" constraint is assigned to the "name".

- CREATEDB EmployeeDB;
- CREATE TABLE Employee Table (
  Employee ID int PRIMARY KEY,
  Name varchar(50) NOT NULL,
  Salary int CHECK(0 < salary),
  Affiliation ID int REFERENCES
  Affiliation table (Affiliation ID)
  ON UPDATE CASCADE ON DELETE CASCADE);
- CREATE TABLE Affiliation table (
  Affiliation ID int Primary Key,
  Affiliation varchar(5) UNIQUE);

```
<Employee information>
    <row>
      <Employee ID>1</Employee ID>
      <Name>Taro Yamada</Name>
      <DOB>1970-03-04</DOB>
      <Sex>Male</Sex>
    </row>
  <row>
    <Employee ID>2</ Employee ID>
      <Name>Shinji Kaneko</Name>
      <DOB>1970-03-04</DOB>
      <Sex>Male</Sex>
  </row
```

Figure 7. Example of XML schema

Figure 8 shows the XML schema that is output based on the above RDB schema. It can be seen that the column name and constraint information are output. In the current version of our schema conversion module, users (such as DBAs) have to rename the duplicate database names or the duplicate table names, manually.

```
<!—Omitted-->
      <xs:element name=" Affiliation table"
type="Affiliation table Type"/>
      <xs:element name=" employee table "
type="Employee table Type"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="Affiliation table Type">
   <xs:annotation>
      <xs:appinfo>
         <r:index index-key=" Affiliation id"
primary="yes"/>
         <r:index index-key=" Affiliation "
unique="yes"/>
      </xs:appinfo>
<!— Omitted -->
      <xs:complexType>
         <xs:sequence>
            <xs:element minOccurs="1" name="
Affiliation id" r:nullable="false"
            <xs:element minOccurs="0" name="
Affiliation " r:sqltype="varchar"
<!—Omitted-->
<xs:complexType name=" Employee table Type">
   <xs:annotation>
      <xs:appinfo>
         <r:index index-key="Employee id"
primary="yes"/>
  <r:check check-column="Salary" rule="0 &lt;
&quot;salary&quot;"
  <r:fkey fkey-column=" Affiliation id" ref-column="
Affiliation id" ref-table="
```

Figure 8. Example of common schema

5. Virtual Database Query

A query input by a user is a single query to a virtual database rather than an individual query to each database. Possible candidates for user-issued queries are SQL, XQuery[16], and virtual database original queries. In this study, the common schema expressing the entire virtual database structure is described in XML format, and since this is presented to the user as the structure of the entire virtual database, XQuery was chosen as the type of query to be issued by a user to the virtual database. This is a first step, and in the future, when OODBs as well as RDBs and XMLDBs are supported, if the required query cannot be expressed with XQuery or is complicated and difficult, defining an original query for a virtual database may also be considered.

5.1. Query Conversion

When both a RDB and XMLDB are used at the same time, differences will exist in the query language used for each database, e.g., SQL for the RDB and XQuery for the XMLDB. Moreover, differences will exist in the data model since the RDB data is managed in a table-like structure, and the XMLDB is managed in a tree structure. Consider, for example, the case in which data as in Table 2 is stored in the RDB and data as in Figure 9 is stored in the XMLDB.

Table 2. Example of employee information table

Table name: Employee information

Employee ID	Name	DOB	Sex
1	Taro Yamada	1970-3-4	M
2	Shinji Kaneko	1975-4-2	M
3	Yasuko Oota	1974-1-9	F
4	Reika Yasuda	1980-6-25	F
5	Isamu Kirishima	1948-10-16	M
6	Koki Oonishi	1958-8-2	M
7	Kazuo Yamaguchi	1970-3-4	M
8	Kaoru Saitoh	1983-2-12	F
9	Yasushi Yamagishi	1986-3-5	F
10	Kenji Kimura	1986-6-2	M
11	Hiroki Yuuki	1986-9-30	M

```
        Organizational Structure.xml
< Organizational Structure >
   <Development Div.>
      <1st Development Dept.>
         <Dept. Director>3</ Dept. Director >
         <Section Mgr.>2</ Section Mgr.>
         <Asst. Mgr.>7</ Asst. Mgr.>
      </1st Development Dept.>
      <2nd Development Dept.>
         < Dept. Director >5</ Dept. Director >
         < Section Mgr.>6</ Section Mgr.>
         < Asst. Mgr.>10</ Asst. Mgr.>
      </2nd Development Dept.>
      <Sales>
         < Dept. Director >3</ Dept. Director >
         < Section Mgr.>4</ Section Mgr.>
         < Asst. Mgr.>8</ Asst. Mgr.>
      </Sales>
   </Development Div.>
</Organizational Structure>
```

Figure 9. Example of XMLDB data

Then, consider the case in which, based on these two data sources, it is desired to acquire the Dept. Director of the 1st Development Dept. In this case, a query such as the following will be issued.

• To acquire the name of the Dept. Director of the 1st Development Dept.

・Obtain employee ID and name list from RDB

```
SELECT Employee ID, Name
FROM EmployeeInformation;
```

・Obtain targeted employee ID from XMLDB

```
let $a := document("Organizational Structure.xml")
          /Organizational Structure/Development
Dept./1st Development Dept./Dept. Director
    return $a
```

The execution results from this query are as shown in Table 3 and Figure 10.

Table 3. Example of employee ID and name list

Employee ID	Name
1	Taro Yamada
2	Shinji Kaneko
3	Yasuko Oota
4	Reika Yasuda
5	Isamu Kirishima
6	Koki Oonishi
7	Kazuo Yamaguchi
8	Kaoru Saitoh
9	Yasushi Yamagishi
10	Kenji Kimura
11	Hiroki Yuuki

```
<Dept. Director>3</Dept. Director>
```

Figure 10. Employee ID of Dept. Director of 1st Development Dept

Results similar to the above are output in response to each query; however, the results that the user ultimately desires cannot be output based on the results from the RDB and XMLDB, and the user must personally compare those results. Moreover, because each query language is different, the user must be aware that multiple queries (SQL and XQuery in this case) are issued according to the DB used. Additionally, the user must also ascertain which data is stored in which database. These requirements place a large burden on the user, and as a result, certain tasks become very time consuming.

When implementing this example with a virtual database, the user will issue the following type of query.

• To acquire name of Dept. Director of 1st Development Dept.

```
for $employee in common-schema()/RDB
      /sample_rdb/Employee information/row
   for $manager in common-schema()/XMLDB
      /sample_xmldb/Organizational structure.xml
        where $employee/Employee ID
   = $manager/Organizational structure/Development
Div./1st Development Dept./Dept. Director
   return $employee/name
```

○RDB

○XMLDB

The execution results obtained in response to this query are as shown in Figure 11.

```
<name>Yasuko Oota</name>
```

Figure 11. Execution results

When the actual data was checked, the content of the /Organizational structure/Development Div./1st Development Dept./Dept. Director element from the XMLDB data was found to be "3." Then, by looking up the name of the individual having an employee ID of "3" in the data of the "Employee information table" in the RDB, the name "Yasuko Oota" can be verified. Thus, it is found that the required information was obtained correctly in response to the query. Queries can extract data from different databases. Users do not need to write two different queries (one for every database). The final results are combined.

5.2 Common Schema Query

The common-schema() statement in the query is a newly defined function for the virtual database, and is used to indicate the root of the common schema. By specifying, after the common-schema() statement, the path for obtaining the required values, the user is able to obtain RDB data or XMLDB data depending on the format that the data is stored in. These values are converted to object format, and set as the QueryArg object (as shown in Figure 12), which represents a variable. In this example, RDB data is set as the object representing $employee and XMLDB data is set as the object representing $manager. Further, by specifying the path and using a "where" clause to specify a condition for the value stored in this variable, RDB and RDB, RDB and XMLDB, and XMLDB and XMLDB values can be compared, and only that content which satisfies the condition can be returned. When attempting to obtain this result, the user sees only a structure indicating a common schema, and is not aware of the actual databases existing behind the virtual database. Of course, the individual queries, such as SQL and XQuery, for the actual databases are not generated. Thus, the values acquired from the actual DBs can be compared with one another, without the user being aware of their differences.

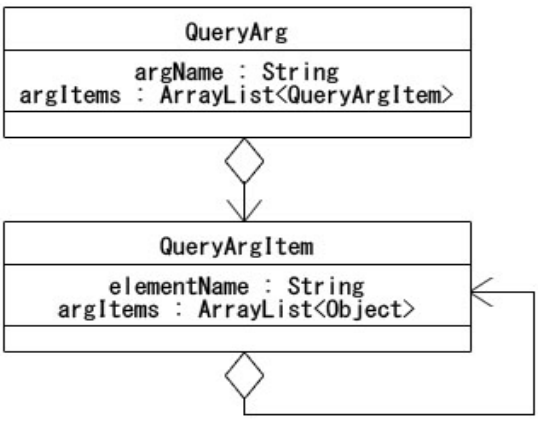


Figure 12. Variable objects in a query shown

In this manner, the use of a virtual database enables RDB and XMLDB values to be acquired and utilized without the user being aware of the difference among the databases.

6. Verification Preparation for Data Mining Application Function

6.1. Verification from User Interface

The effectiveness of a virtual database that uses data mining is verified. Here, the tool described in reference[17] is treated as a data mining support tool, which enables the results of the mined data to be visualized, and helps analysts obtain a true understanding of the situation.

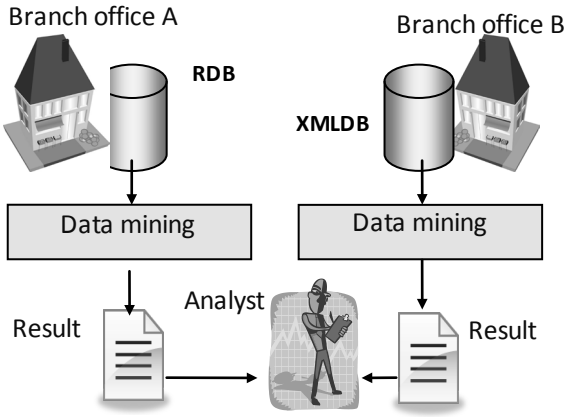


Figure 13. Analysis from multiple sources

First, for the purpose of verification, transaction data containing the purchase history of a convenience store is used as the data to be analysed. Consider the case in which the data is stored in a RDB at branch office A and in a XMLDB at branch office B, as shown in Figure 13. Usually, when performing the analysis, data is output from both databases and the outcome results must be compared using a data mining tool.

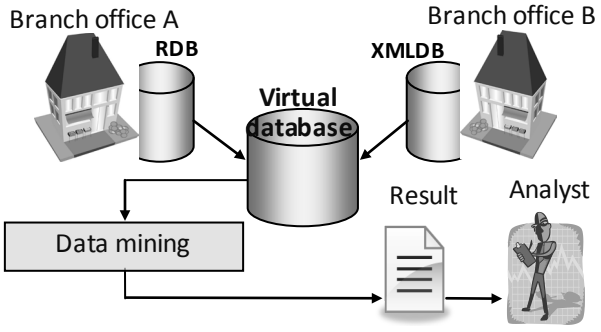


Figure 14. Analysis when using a virtual database

In a similar scenario, Figure 14 shows the way in which data mining is performed using a virtual database. As can been seen in the figure, since data is output from a virtual database formed from multiple databases, there is only one data output, rather than multiple outputs. Thus, a single result can be extracted by mining this data. In other words, there is no need to perform multiple data mining operations, and there is no need to compare the results. For example, even when using the tool described in reference[17], since the result can be output as a single comprehensive result rather than as multiple results, the outcome can be discovered easily by analysing and visualizing data from the two branch offices.

6.2. Verification from Application Interface

The API for a virtual database is described below. In this study, a virtual database using XQuery was developed as a first step. However, the effectiveness of several other patterns should be compared and reviewed. The proposals are as follows.

- **Proposal 1. XQuery + extended functionality**

This proposal is to support XQuery plus extended functions not provided by XQuery, as shown in Figure 15. Although XQuery allows flexible queries, there is a problem in that the statement length will increase in the case of a complex query. Conceivable extensions include a control or update function for constraints associated with the RDB.

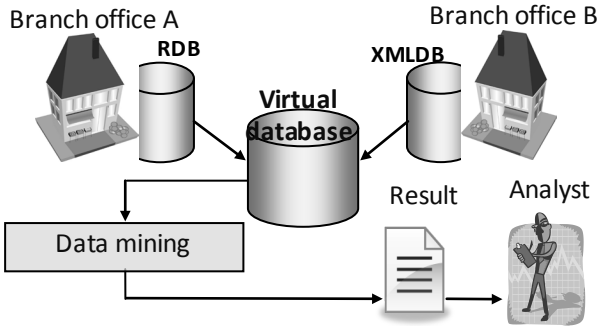


Figure 15. API (XQuery + extension)

- **Proposal 2. SQL + extended functionality**

This proposal is to support SQL plus functions not provided by SQL, as shown in Figure 16. SQL can be written simply with short statements, but it lacks flexibility. Therefore, as extensions, the addition of statements such as a "For" statement, a function for issuing queries to such elements as a XMLDB, and a function for discriminating between elements having the same name are considered essential.

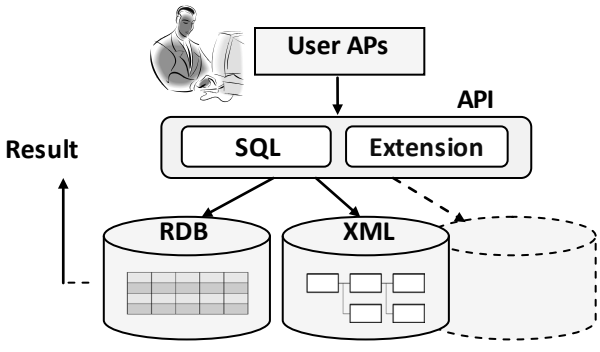


Figure 16. API (SQL + extension)

- **Proposal 3. New query language**

The two types of queries based on the above proposals have advantages and disadvantages. Therefore, as shown in Figure 17, this proposal is to develop a completely new query language, rather than use an existing language. It is thought that by considering various patterns, an appropriate query can be developed. The new query language is one of the future development themes. So, we have not yet completed the development.

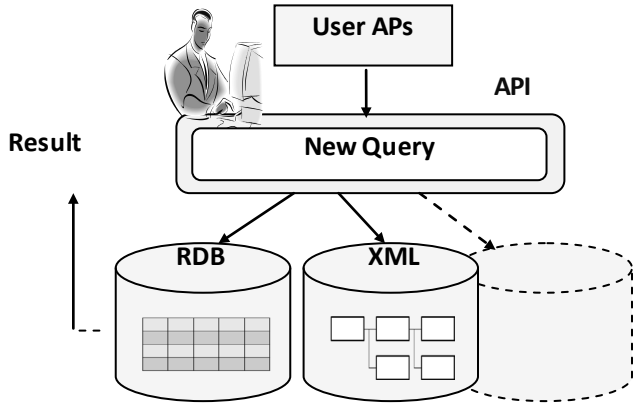


Figure 17. API (new query language)

7. Recovery for Virtual Databases

As described in this section, we will propose a trouble recovery method in the virtual database environment in case a trouble occurs when using a ubiquitous database. We will describe it along with its associated experiment results.

7.1. Example of Virtual Database Construction

First, we describe the virtual database we prepared for the experiment. For database virtualization, we use a common schema described in the XML for all actual associated database structures. Figure 18 presents an example of a real database on which employee data are described on its RDB side, and the department data on the XMLDB side. On the RDB side, we have a database named database1; in it, we have tables named employee and salary with respectively associated columns of id and name, and id and salary. On the XMLDB side, we registered an XMLDB with the key named post in the collection of /db/sampledb/.

Describing this environment above using a common schema, we can devise the scheme depicted in Figure 19, where in node db, we have nodes rdb and xmldb. The RDB structure is stored in rdb, with the XMLDB structure in xmldb. The child node of rdb has the node, mass, showing the name of the database; the child node of xmldb has the node, collection, showing the collection. The RDB is described with the name, table, as the child of mass, and the name, column, as the child of table. On XMLDB, a node named mass is prepared as the child of collection, and the key is described to its name. For the children of mass, the entire XMLDB structure is described.

The user is expected to use and issue query statements while regarding the data structure based on such a common schema described as above. The data structure that is given to the user is transparent as to which DB it is associated with—RDB or XMLDB. Furthermore, the common schema is not given to the user. The virtualized DBMS, when it receives the query statements from the user, finds and locates the right DB to operate based on the common schema, and converts the query into what is appropriate to each associated DBMS to get it executed by that DBMS.

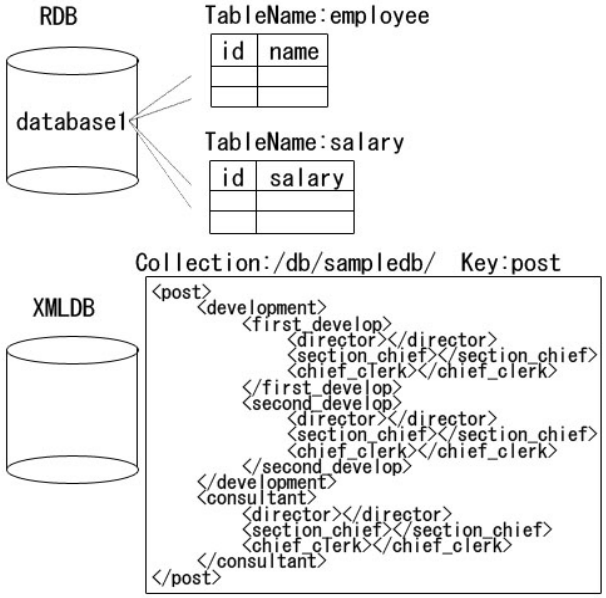


Figure 18. Example of a real database

7.2. Trouble Recovery

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<db>
    <rdb>
        <mass name="database1">
            <table name="employee">
                <column>
                    <id></id>
                    <name></name>
                </column>
            </table>
            <table name="salary">
                <column>
                    <id></id>
                    <salary></salary>
                </column>
            </table>
        </mass>
    </rdb>
    <xmldb>
        <collection name="/db/sampledb">
            <mass name="post">
                <station>
                    <development>
                        <first_develop>
                            <director></director>
                            <section_chief></section_chief>
                            <chief_clerk></chief_clerk>
                        </first_develop>
                        <second_develop>
                            <director></director>
                                <section_chief></section_chief>
                            <chief_clerk></chief_clerk>
                        </second_develop>
                    </development>
                    <consultant>
                        <director></director>
                        <section_chief></section_chief>
                            <chief_clerk></chief_clerk>
                    </consultant>
                </station>
            </mass>
        </collection>
    </xmldb>
</db>
```

Figure 19. Example of common schema

In this section, we describe trouble recovery when required to make to the virtual database in addition to the

information of the problems known when the recovery is requested on the real DBMS side and of the necessary operations on the virtual DBMS side to avoid the problems. Here we presume that some hardware trouble has occurred. We therefore describe how to recover from it based on the log available with the backed-up data are restored.

7.2.1. Incompatibilities with Real DBMS

Here we describe possible problems that would result if we had each real DBMS get the log individually, for example, without implementing any functionality of backup recovery with the virtual DBMS.

We presume that the process shown in Figure 20 was done with the virtual DBMS. In that example, data for the employee of id=1 is given as[salary: 200,000y→250,000]; for the employee of id=2[salary : 450,000→520,000], and the update operation of[post: post.post.consultant.director (8→2)],[post.post.consultant.section_chief(2→9)] is executed. Additionally, DB_A is presumed to be an RDB, and DB_B an XML. In addition, for simplicity, serial processing is assumed.

In this example case, presuming that trouble occurred at the start of the No. 5 process, we would know that the DB_A processing aborted, so that no REDO would be required. On the other hand, we know that the DB_B processing has not started, so no recovery would have to be made. Consequently, the processing for DB_A and DB_B can be integrated.

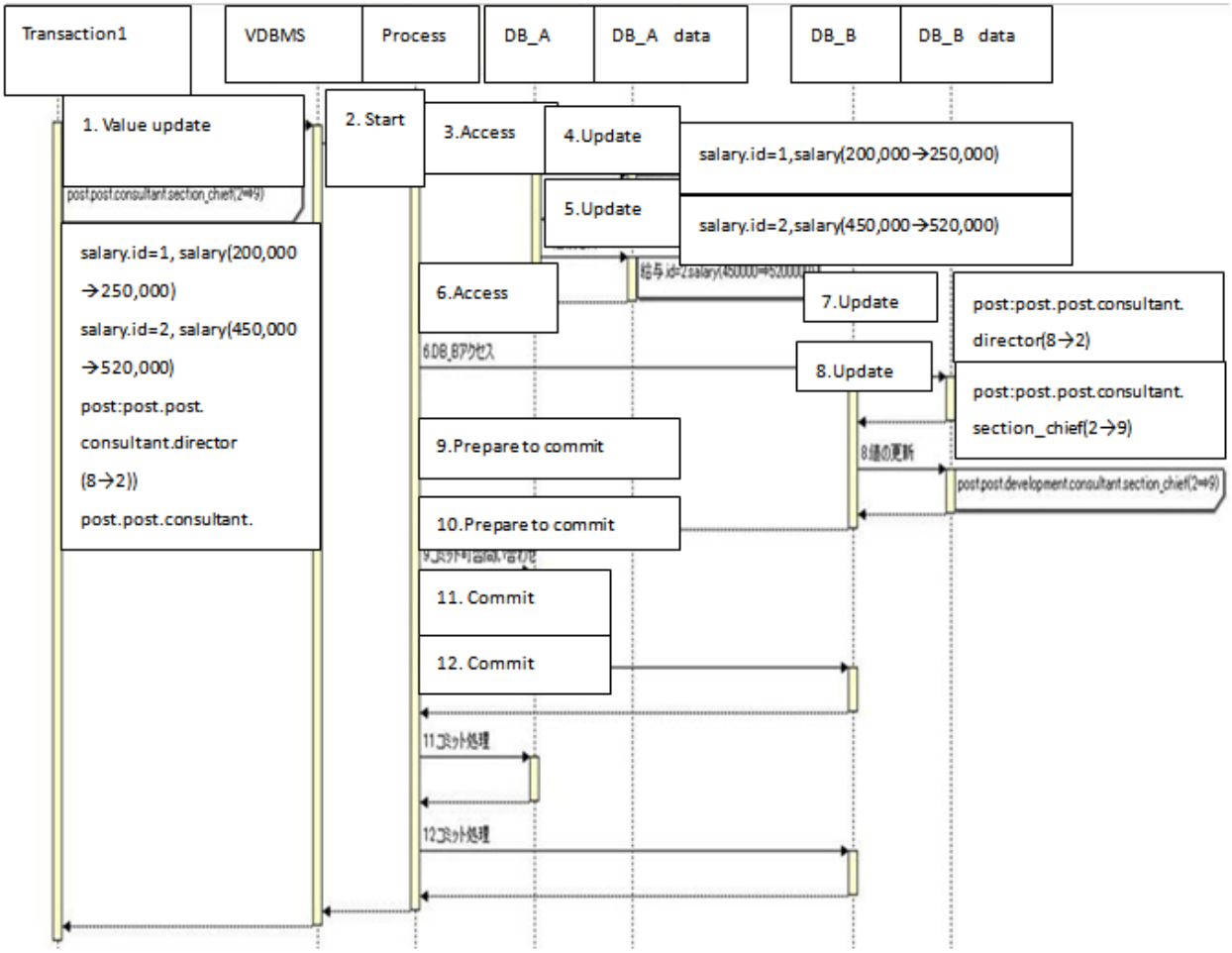


Figure 20. Example of incompatibility

```
001 Transaction Start
001 Value update (salary.id=1, salary(200,000→250,000))
001 Value update (salary.id=2, salary(450,000→520,000))
001 Value update (post:post.post.consultant.director(8→2))
001 Value update (post.post.consultant.section_chief(2→9))
001 Transaction End
```

Figure 21. Example of log of virtual DBMS

7.2.2. Trouble Recovery with Virtual DBMS

As described earlier, when recovering from trouble at each DBMS, an incompatibility might arise between the virtual database and the real database depending on the timing with which the trouble occurs.

To avoid that problem, we manage the database operations log not to be taken at each real DBMS, but on the virtual DBMS side. Thereby, we manage to recover from trouble. We have the log taken on the virtual DBMS side with no database differences to be recognized by the user, as shown in Figure 21. For that, we assign and attach the transaction serial number to the top of each log element. Using such serial numbers, we identify the information related to every log taken during parallel processing.

If hardware trouble were to occur, for instance, then the data might be destroyed. As with the ordinary DBMS, such is the case with the virtual DBMS. For that reason, the entirety of every database should be backed up. Along with the virtual DBMS, we would advise that common schema information be backed up as well. The retained common schema would fundamentally not be available for the user to view, but it might be used to help operate the virtual DBMS to restore every associated database data from information left on the log.

First, we find the "Transaction start", and find the number put at its top of that line. If the "Transaction end" that has the same number as that one were found, it would indicate that a process had been committed when the trouble occurred, which means that the process would have to be completed to achieve restoration. However, if no transaction end were found, then no process was committed. Consequently, no restoration would be necessary.

For example, regarding the log portrayed in Figure 21, we find the log of "001 Transaction start". Checking further down the log, we also find "001 Transaction end". Then, meaning that the transaction of 001is recognized as the process that must be restored.

As a "Value update" series, processing is executed in steps, as shown, to restore them. We use the common schema retained through the backup process. For "001 Value update (salary.id=1,salary(200,000→250,000))"—because the user would issue query statements without noticing any database difference even if it were to exist—the hierarchy levels above "mass" are ignored in favor of finding out whether <table name="salary"> or <salary> exists as the child of "mass". In our example, because we find <table name="salary"> as the child of <rdb><mass name="database1">, we know that we need to access the RDB database, database1.

In the following step, we move on and prepare for the query statement for the RDB to be used to update the salary amount for 200,000–250,000 for the row having id=1 in the table, salary, in the database, database1.For "001 Value update (post.post.consultant.director (8→2))", we check whether there exists <table name="post"> or <post> as the child of "mass". In our example, we find <post> as the child of <xmldb><collection name="/db/sampledb"><mass key="post">, we access to the XMLDB that is registered with the key, post, in the /db/sampledb/ collection in it.Then we move on to prepare for the query statement for the XMLDB to update the <director> value, which is found as the child of <post><consultant>, from 8 to 2, and issue it to begin the restoration process.

Presuming that the log were continued with "002 Transaction start", "003 Transaction start", and "003 Transaction end", then the log of 002 would be ignored; only the log of 003 would be processed to REDO. Through this series of processes described above, we can accommodate cases where, if some trouble occurred even during the No. 12 process, for example, that DB_A receives a commit statement to send to DB_B.

8. Conclusions

Firstly, we proposed the method of virtualization of ubiquitous databases and confirmed that we can describe ubiquitous database schema in a unified fashion using the XML schema.

Then, we developed the common schema conversion program for RDB schema into XML schema. Especially, we showed the schema constraints (such as PRIMARY KEY, CHECK, NOT NULL, UPDATE CASCADE ON DELETE, UNIQUE) can be converted.

Next, we developed the integration program of XML DB schema into the common schema. In addition, we implemented the common data manipulation API (for example, extension of the existing XQuery modules) to access the virtual databases and we incorporated location transparency functions to this API.

Conventionally, when acquiring data from multiple types of DBs, a user has to issue a query corresponding to the DBMS, and then use the results from that query to infer the desired information. The newly developed virtual database enables the user to acquire and compare information as if only a single database were being accessed, without the user being aware of the differences among the databases. As part of this development work, an API for a virtual database was designed. In the future, we plan to compare and review this API, to consider which patterns are most suitable, and then implement and verify those API patterns.

Finally, we described aspects related to recovery from virtual database trouble. We proposed a procedure for the virtual DBMS to restore data after some hardware trouble in cases not only with database data but also with common schema information retained in addition to the associated log information that is used.

However, for system trouble in which the data are left undeleted, the same trouble recovery operation, if done as for hardware trouble, would require much more time than expected, meaning that it would be much more costly to use. To avoid this situation, such a recovery procedure that could best use checkpoint functionality, for example, would be

required.

We are planning to examine that procedure in detail in future studies. Another important issue that we must work to solve is the confirmation of compatibility of a log taken for a virtual database and for associated real databases. In addition, the new query language, which is discussed in Section 6.2, is also one of the future development themes.

ACKNOWLEDGEMENTS

This research was supported by a Grant-in-Aid for Scientific Research C (Subject No. 20500095: Study of Data utilization with Ubiquitous Database Virtualization Technology).

REFERENCES

[1] K. Mori, S. Kurabayashi, N. Ishibashi, and Y. Shimizu, "Method of Sending Information Actively Reducing User Information Load Dynamically in a Mobile Computing Environment", DEWS2004 , 2004.

[2] teiid, http://www.jboss.org/teiid, Red Hat

[3] DB2:Information Integrator V8.1, http://www.jpgrid.org/documents/pdf/WORK4/sugawara_ws4.pdf

[4] H. Garcia-Molina, J.D. Ullman, and J. Wisdom, Database Systems: The Complete Book Second Edition, Pearson Education Inc. 2009.

[5] R. Elmasri and S.B. Navathe, Fundamentals of Database Systems, 5th Edition, Pearson Education Inc. 2007.

[6] S. Abiteboul, P. Buneman, and D. Suciu, "Data on the Web: From Relations to Semistructured Data and XML", Morgan Kaufmann Series in Data Management Systems (1999).

[7] S. Amer-Yahia, F. Du, and J. Freire, "A comprehensive solution to the XML-to-relational mapping problem", Proc. 6th Annual ACM International Workshop on Web Information and Data Management, pp.31-38, 2004.

[8] I. Varlamis and M. Vazirgiannis, "Bridging XML-schema and relational databases: a system for generating and manipulating relational databases using valid XML documents", Proc. 2001 ACM Symposium on Document engineering, pp.105-114, 2001.

[9] P. Bohannon, J. Freire, P. Roy, and J. Simeon, "From XML Schema to Relations: A Cost-Based Approach to XML Storage", Proc. 18th International Conference on Data Engineering, pp. 64-75, 2002.

[10] G. Kappel, E. Kapsammer, and W. Retschitzegger, "Integrating XML and Relational Database Systems", World Wide Web: Internet and Web Information Systems, 7, pp. 343-384, 2004.

[11] R. Li, Z. Lu, W. Xiao, B. Li, and W. Wu, "Schema Mapping for Interoperability in XML-Based Multidatabase Systems", Proc. 14th International Workshop on Database and Expert Systems Applications, 2003.

[12] Y. Wada, Y. Watanabe, J. Sawamoto, and T. Katoh," Database Virtualization Technology in Ubiquitous Computing", Proc. 6th Innovations in Information Technology (Innovations'09), pp.170-174, 2009.

[13] Y. Wada, Y. Watanabe, K. Syoubu, J. Sawamoto, and T. Katoh,"Virtualization Technology for Ubiquitous Databases", Proc. 4th Workshop on Engineering Complex Distributed Systems (ECDS 2010), pp. 555-560, 2010.

[14] Y. Wada, Y. Watanabe, K. Syoubu, J. Sawamoto, and T. Katoh, "Virtual Database Technology for Distributed Database", Proc. IEEE 24th International Conference on Advanced Information Networking and Applications Workshops(FINA2010), pp.214-219, 2010.

[15] Y. Wada, Yuta Watanabe, Keisuke Syoubu, Hiroshi Miida, Jun Sawamoto and Takashi Katoh," Technology for Multi-database Virtualization in a Ubiquitous Computing Environment", International Workshop on Informatics (IWIN2010),pp. 89-96, 2010.

[16] S. Buxton, J. Milton, K. Shibano, K. Yamahira, T. Kodera. and M. Tsuchida, XQuery Querying XML, XQuery, XPath,, and SQL/XML in Context (Programmer's SELECTION), SHOEISYA, pp.669, 2008.

[17] H. Miida, and Y. Wada, Development of Comparison and Analysis Supporting Tool of Data Mining, FIT 2009, D-030, pp. 195-198, 2009.