ISBN 978-952-5726-09-1 (Print)
Proceedings of the Second International Symposium on Networking and Network Security (ISNNS '10)
Jinggangshan, P. R. China, 2-4, April. 2010, pp. 275-281

Database Auditing Design on Historical Data

Narongrit Waraporn
School of Information Technology, King Mongkut's University of Technology Thonburi, Bangkok, Thailand
Email: narongrit@sit.kmutt.ac.th

***Abstract*—Database Auditing is one of the major issues in information security. Lack of data auditing leads the business applications to the lost trail of business processes. To cope with auditing, historical data or temporal database is needed in order to track operations and actors of the operation with the time. Valid and transaction times are two major timestamps in temporal database. In this paper, we demonstrate the techniques to handle database auditing in business transaction by considering operations, valid times, and actors of the operations. These techniques are divided in two sets; using relational databases, and using semi-structure data.**

***Index Terms*— Database Auditing, Historical Data, Temporal Database, Object-Relational Database, XML**

I. INTRODUCTION

Information System, IS, and Information Retrieval, IR, have been defined and adopted into the practices since the beginning of digital revolution. While information has been used for business processes, information security emerged in term of authentication and authorization. Those concepts can protect information but do not provide help in investigation. Then, the log files were proposed to track the trail of access to databases and systems. However, the main purpose of database log is to recovery the transaction. To be able to investigate the transaction, the database audit was introduced.

Many commercial databases suggest database auditing by using the log files. DB2 offers the creation of "audit policy" to apply to certain roles of users. However, they cannot cover every aspect of business processes when considering historical data. We propose techniques of database auditing based on historical data for proprietary systems.

In some transactional systems, database design includes auditing data into operational table. However, the auditing data is not used often. We propose the separation between auditing data and business operation data.

This paper is organized into three main parts. First, the fundamental and reviews of auditing and historical data are in section II and section III respectively. Second, the design of database auditing is presented in various approaches. Using well-established relational databases to implement is suggested with three approaches; row-based, column-based and log-table auditing, in section IV. While semi-structured database serves the complex information, we suggest the use of two alternatives; Object-Relational Database and XML in section V. At last, we conclude and discuss other issues of database auditing in section VI.

II. AUDITING

Auditing a change to a management system such as accounting, performance, and software, has been well defined and proctored to improve the quality. One of the major sources of auditing is data where auditor can find the errors and misuses. Lately, data, itself, is audited for its movement to identify malicious behavior, maintain data quality, and improve system performance in [7], and [13]. [7] proposed a framework for auditing the changes to a database system with the retention policies. The declarative rules for expressing retentions, semantics for audit query answers, and performance of audit query under retention restrictions were demonstrated. To detect anomaly behavior, [13] suggested a grid-based clustering method on a transaction in an audit data stream containing a set of user activities at a particular time.

Since auditing data is likely to be abundant, searching for an auditing answer requires data mining techniques such as classification and regression. [8] presented an auditing data tool based on data mining algorithms. They evaluated their QUality Information System (QUIS) database in different alternatives, instance based classifiers, naive Bayes classifiers, classification rule inducers, and decision trees.

To extend auditing concept into communities, [16] developed a hand-on experience on database security and auditing on relational table containing an audit trail of all operations that change data on the target table. But, the paper did not specify the design of audit trail table while our paper suggests the design alternatives with their pros and cons.

III. HISTORICAL DATA

One key success of auditing is to be able to track the change on the trail of who made the changed, what operation of the modification, and when it happen. The first two trails can be manipulated in relational model while the last trail requires temporal characteristics to handle.

Two different types of time are considered when recording the history of information; valid time and transaction time. [1] suggested that valid and transaction times should guarantee no information loss in any circumstantial transactions. [2] suggested that the transaction time is needed to satisfy the data mining. However, in this paper, we demonstrate only the valid time. Transaction time can be stamped when the "user" column saves the username whom modifies the record.

However, temporal databases have not been widely adopted by commercial databases and IT communities due to the overhead of temporal computing. A different approach on temporal data is to use XML document to manage historical data. [3] fragments transaction information into event data to retrieve some data from XML document. This is due to transmitting large transaction but only the context change based on historical data is sent. An extension of XQuery language is also demonstrated in the paper. Alternatively, implementation techniques based on temporal characteristics to audit data in plain-relational and XML-enabled databases are suggested in this paper.

IV. HISTORICAL DESIGN FOR AUDITING ON RELATIONAL DATABASES

Historical data can be modeled in Relational Database, RDB, in several techniques such as separated tables for history records, transaction logs, and multi-dimension data using XML. Creating a separated table for historical data of each relational table is a straight forward solution with minimum design complexities while the transaction logs must cross-check data from the data dictionary. Both solutions require no change to the original relational tables. Without additional tables, XML columns can be added to the original table with the background of XPATH, XQUERY, and SQL/XML for the data retrieval.

To maintain historical for auditing, four suggested techniques can be implemented; row-based auditing, column-based auditing, log-table auditing, and semi-structure-based auditing which can be implemented by using object-relational types or XML. The first three types are implemented in the relational databases while the semi-structure-based auditing must rely on the extension of database engines on object-relational and XML technologies such as in IBM DB2 9.5, Oracle 10g, and MS SQL Server 2007, but not in MySQL 5.1 where XML data is still maintain in varchar column [10]. We explained the semi-structure-based auditing the section V.

A. Row-based Auditing

This technique creates a separated table for each relational table to maintain historical data. The operational table remains the same as in non-auditing system.

An EMPLOYEE table of operational tables, as shown in Table I maintains only the current value of each employee for business operations. The table, also, includes both static data and historical data. The static data remains unchanged such as data of birth or rare to change such as name. For the historical data, only the last-updated values are maintained in the operational table. The non-historical query, which is regularly requested, still remains unchanged in order to retrieve the information from the EMPLOYEE table.

The auditing table contains every column of the operational table as shown in Table II. To diminish the join-operation query, static data is included in the auditing table. Two timestamps are needed for the valid times; start time and end time to maintain the lifespan of the data. Two additional attributes; operation type and username should be considered. Operation type is recorded to reduce the overhead of comparison among histories of the same data. Username is stored for the reason of responsibility of the sensitive data.

TABLE I.
OPERATIONAL EMPLOYEE TABLE REFERENCED BY EMPLOYEE-HISTORY TABLE FOR ROW-BASED AUDITING

ID	Name	DOB	Address	Hired Date	Salary
101	Tom	1980-01-01	New York	2008-01-01	55000
102	Ann	1985-06-16	London	2009-01-01	60000

Table II shows the history table of the EMPLOYEE table. The history table shows that Jack has worked since 2008/01/01. He moved from New York to Hong Kong on 2008/06/15. On 2009/01/01, his wage was increased to 70000 authorized by Mel. At the end of May 2009, he left the company and it is recorded by Matt.

The records are the same between records in the operational table and the auditing table whose end time is null. The redundancy of these records is for the historical query so that there is no need to retrieve data by using union operation between them.

The row-based auditing has pros and cons.

Pros

- It simplifies the implementation of auditing. When a DML statement such as INSERT, UPDATE, and DELETE, is executed on the operational table, the application can simply copy every value in the record into the auditing table. At the same time, the end time column of the previous history must be updated with the operated time. However, this could be done by database trigger if the system does not handle the temporal issues at the application level. [16], also, used this approach.
- Retrieving static data and historical data are independence. There is no need to union the data from two tables together in any query because the auditing table maintains the static data.

Cons

- Redundancy of current record in both tables. However the historical query would be less complex.
- Retrieving historical data requires the comparison between two records by using recursive query. For example, SQL statement for the auditing of salary of Jack whose ID is 103 is

```
SELECT E1.SALARY, MINS, MAXE,
E1.USER, OPERATION
FROM EMPLOYEE_HISTORY_R E1,
    ( SELECT E2.SALARY, MIN(E2.STARTTIME)
        MINS,   MAX(E2.ENDTIME) MAXE
      FROM EMPLOYEE_HISTORY_R E2
      WHERE ID = 103  GROUP BY SALARY) E3
WHERE E1.SALARY = E3.SALARY
```

TABLE II.
OPERATIONAL EMPLOYEE TABLE REFERENCED BY EMPLOYEE-HISTORY TABLE FOR ROW-BASED AUDITING

PK	ID	Name	DOB	Address	HiredDate	Salary	StartTime	EndTime	Operation	User
1	101	Tom	1980-01-01	New York	2008-01-01	50000	2008-01-01	2009-01-01	I	Mike
2	101	Tom	1980-01-01	New York	2008-01-01	55000	2009-01-01		U	Mel
3	102	Ann	1985-06-16	London	2009-01-01	60000	2009-01-01		I	Mike
4	103	Jack	1975-01-01	New York	2008-01-01	60000	2008-01-01	2008-06-15	I	Mike
5	103	Jack	1975-01-01	Hong Kong	2008-01-01	60000	2008-06-15	2009-01-01	U	Mike
6	103	Jack	1975-01-01	Hong Kong	2008-01-01	70000	2009-01-01	2009-05-31	U	Mel
7	103	Jack	1975-01-01	Hong Kong	2008-01-01	70000	2009-05-31	2008-05-31	D	Matt

B. *Column-Based Auditing*

The column-based auditing solves the redundancy of the row-based auditing. This auditing table does not include the static columns such as date of birth and hired date of an employee. Data in historical column of auditing table are stored only the changed value except the primary key, such as ID, which is used to reference its operational table.

Employee history in Table III stores only the changed data. It is clear to see that data in Table II is less redundant than data in Table I. Under the same ID, Jack moved from New York to Hong Kong on Jun/15/08 and get raise from 60000 to 70000 on Jan/01/09. Selecting not-null value on a particular auditing column in SELECT statement would display only the actual change. For example,

```
SELECT SALARY, STARTTIME, ENDTIME,
USER, OPERATION
FROM EMPLOYEE_HISTORY_C
WHERE ID = 103 AND SALARY IS NOT NULL
```

The query displays the auditing of Jack's salary. Comparing with row-based auditing on the same query, the SELECT statement is much less complex.

Each record in column-based auditing table cannot contain more than one value of historical data because of the uncertainty of end time of each auditing data.

TABLE III.
EMPLOYEE_HISTORY_C TABLE USING COLUMN-BASED AUDITING

PK	ID	Address	Salary	StartTime	EndTime	Opera-tion	User
1	101	New York		2008-01-01		I	Mike
2	101		50000	2008-01-01	2009-01-01	I	Mike
3	101		55000	2009-01-01		U	Mel
4	102	London		2009-01-01		I	Mike
5	102		60000	2009-01-01		I	Mike
6	103	New York		2008-01-01	2008-06-15	I	Mike
7	103		60000	2008-01-01	2009-01-01	I	Mike
8	103	Hong Kong		2008-06-15		U	Mike
9	103		70000	2009-01-01		U	Mel
10	103			2009-05-31		D	Matt

Pros

- The historical query on auditing is less complex than row-based auditing. So, it is likely to execute faster.
- Disk space would be used less than the row-based auditing if the historical column is varchar.

Cons

- Maintaining many NULL values in the table would lead to other problems especially when writing a query without considering the semantic of NULL cautiously [15].
- If most operation on the table is INSERT, but not UPDATE, the number of records would be n times bigger where n is the number of historical columns; two times in our examples because of ADDRESS, and SALARY.
- SELECT statement requires a recursive join in order to query two or more auditing columns. Therefore, it is higher complexity than the case of row-based auditing for this query.

C. *Log-Table Auditing*

Log tables have been used for transaction management in the relational database for a long time. Due to the nature of transaction that needs to know operation, data, and time of execution, log tables can be utilized for the auditing purpose too. For example, DB2 audit facility allows DBA to maintain an audit trail for a series of predefined database events and saved it in an audit log file [6]. Most major databases such as Oracle and SQL Server use a similar method [11] and [9] while MySQL is partially support [10]. Postgres recommends the use of trigger for audit trail [12].

However, the log tables of commercial databases are not intended for business process but rather for database administrator purpose. Additionally, log tables do not serve some auditing aspects such as end time to application. We suggest two developing approaches to handle auditing data using log mechanism.

1) Multiple Audit Log Column Tables for Transaction Logs

To separate auditing log data from the operational data, we create extra table for each auditing column. For example, if ADDRESS and SALARY columns in the EMPLOYEE table are auditing columns, we create ADDRESS and SALARY tables for auditing purposes as shown in Table IV and Table V.

Pros

- It reduces the size of an auditing record.
- It simplifies the design of auditing tables.

Con

- The number of tables which is depending on the number of auditing columns for each table could be high.

TABLE IV.
AUDIT LOG TABLE FOR ADDRESS

PK	ID	Address	StartTime	EndTime	O	User
1	101	New York	2008-01-01		I	Mike
2	103	New York	2008-01-01	2008-06-15	I	Mike
3	103	Hong Kong	2008-06-15	2009-05-31	U	Mike
4	102	London	2009-01-01		I	Mike
5	103		2009-05-31		D	Mel

TABLE V.
AUDIT LOG TABLE FOR SALARY

PK	ID	Value	StartTime	EndTime	O	User
1	101	50000	2008-01-01	2009-01-01	I	Mike
2	103	60000	2008-01-01	2009-01-01	I	Mike
3	101	55000	2009-01-01		U	Mel
4	103	70000	2009-01-01	2009-05-31	U	Mel
5	102	60000	2009-01-01		I	Mike
6	103		2009-05-31		D	Mel

2) Single Audit Log Table for Transaction Logs

To combine audit data into one place, we integrate every auditing column from all operational tables into one single auditing log table. The audit log table composes of name of table and column, primary key of the record in the operational table, new value, valid start time, operation that causes the change and name of user who manipulates this information.

Example of single audit log table of the database containing EMPLOYEE and DEPARTMENT tables is shown in the Table VI. Every transaction occurring to the operational table on the sensitive column is written into the single audit log table. A single insertion of employee number 101 into EMPLOYEE table activates the insertion into audit log table two times; one log record for ADDRESS and another for SALARY if EMPLOYEE table has two auditing columns. Updating on an auditing attribute will insert an auditing record into the log table. Similar to insertion, deletion of a record will be logged twice into audit log table in case of two auditing columns such as deletion of employee 103 in Table VI.

TABLE VI.
SINGLE AUDIT LOG TABLE FOR EVERY TABLE; EMPLOYEE AND DEPARTMENT, IN DATABASE

PK	Table	Column	ID	Value	StartTime	EndTime	O	User
1	Employee	Address	101	New York	2008-01-01		I	Mike
2	Employee	Salary	101	50000	2008-01-01	2009-01-01	I	Mike
3	Employee	Address	103	New York	2008-01-01	2008-06-15	I	Mike
4	Employee	Salary	103	60000	2008-01-01	2009-01-01	I	Mike
5	Employee	Address	103	Hong Kong	2008-06-15	2009-05-31	U	Mike
6	Employee	Salary	101	55000	2009-01-01		U	Mel
7	Employee	Salary	103	70000	2009-01-01	2009-05-31	U	Mel
8	Employee	Address	102	London	2009-01-01		I	Mike
9	Employee	Salary	102	60000	2009-01-01		I	Mike
10	Employee	Address	103		2009-05-31		D	Mel
11	Employee	Salary	103		2009-05-31		D	Mel
12	Department	Manager	D1	103	2008-01-01		I	Mel
13	Department	Manager	D1	101	2009-05-31		U	Mel

Pro

• It minimizes the number of tables. There is no need to scatter the auditing data into many tables.

Cons

• Query requires extra predicates to check the names of table and column.

• The data type of VALUE column must be general type such as VARCHAR due to the various possibility of data type of auditing column e.g. CHAR for ADDRESS, but DECIMAL for SALARY. Data type must be casted back to the original type before comparing it with the original column.

For example, an SQL statement to find name of last user who made a change (UPDATE or INSERT) the SALARY column is

```
SELECT SL.USER, E.name
FROM EMPLOYEE E, SINGLELOG SL
WHERE E.ID = SL.ID AND
   SL.TABLE = ‘Employee’  AND SL.COLUMN =
    'Salary' AND E.SALARY = INT(SL.VALUE)
```

• Audit log table is very large if there are many auditing columns from different tables, especially, on the system with lot of transactions. Dividing data into subsystems and having a single audit log table for each subsystem are recommended.

Both approaches require extra processing for every transaction occurring to the databases, especially, the auditing data. By default, database engines have already manipulated log tables. With this extra processing, the overall system will be degraded. If the auditing is not the main purpose or major use of the application, the system log handling by the database engine is recommended. The supports from database administrators and extra learning of programmers are required.

V. SEMI-STRUCTURED-BASED AUDITING

Since the emerging of objection-oriented paradigm, structured data has been redefined into complex information. Objected Oriented Databases, OODB, had been researched and proposed to communities for many years. However, the IT industry, especially in database software such IBM DB2, Oracle DB and others, has not completely adopted OODB. In fact, Object-Relational database, ORDB, was added into relational database for semi-structured information. Oracle DB includes user-defined types, array of a type, nested tables, and object methods to serve the inheritances, polymorphism, and encapsulation [5].

In contrast, IBM DB2 has proposed the semi-structured information into another technology, XML which claims to be pure XML [4]. It means that data is neither stored as Character Large Object, CLOB, nor shredded into multiple relational tables. It stores data in tree-structure as the nature of XML.

Therefore, we suggest two semi-structured types to manage auditing data; object-relational and XML types.

A. *Object-relational Type*

To audit data in ORDB, we create new types for auditing such as StringAudit type to store auditing data of character columns, and NumberAudit of numeric columns.

```
CREATE TYPE  StringAudit AS(
     VALUE VARCHAR(20),
     STARTDATE DATE,
     ENDDATE DATE,
     OPERATION CHAR(1),
     USER CHAR(10));

CREATE TYPE  NumberAudit AS(
     VALUE DECIMAL(10,2),
     STARTDATE DATE,
     ENDDATE DATE,
     OPERATION CHAR(1),
     USER CHAR(10));
```

Auditing data such as start and end dates of such information, operation and user that manipulate the information are included in the new auditing types. However, the auditing type is used to store only one single record of transaction. To handle the historical data as temporal database, we create an array type for each auditing type.

```
CREATE TYPE StringAuditArray as
        VARRAY(100) OF StringAudit;

CREATE TYPE NumberAuditArray as
        ARRAY(100) OF NumberAudit;
```

However, DB2 array is based only on one of the built-in data type. It means creating array of user-defined type is currently not allowed in DB2. In contrast, Oracle allows array of user-defined type in a special type called VARRAY.

Each element in the array represents the single record of history. Therefore, when creating a table for the auditing purposing, we declare the auditing array type for columns needed to be audited.

```
CREATE TABLE EMPLOYEE_AUDIT_ORDB
 ( ID CHAR(10) NOT NULL,
    NAME CHAR(15),
   DOB DATE,
   ADDRESS VARCHAR(20),
   ADDRESS_AUDIT STRINGAUDITARRAY,
   HIREDDATE DATE,
   SALARY DECIMAL(10,2),
   SALARY_AUDIT NUMBERAUDITARRAY,
  PRIMARY KEY (ID) );
```

Since most transactions need the current value instead of its history, the current-valued columns such as ADDRESS, and SALARY of auditing columns are created separately from the auditing array for faster transaction of most query.

B. XML Type

Due to the semi-structure of XML, auditing data can be defined in the nested elements of XML without creating a new data type as in ORDB. XML also allows repeating elements under the same parent node. So, the historical records of an auditing column such as ADDRESS can be maintained as many as records.

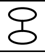
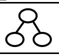
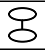
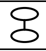
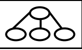
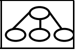
Most commercial databases enable XML data type to maintain semi-structure infomation as it is. We suggest two auditing approaches using XML;

- column-based XML to audit each field of the record
- row-based XML to audit each record

1) Column-based XML Auditing

Column-based XML auditing maintains history of each column of the record separately. The XML tree is at minimum. For example, Employee table in Table I is added with two additional XML columns, ADDRESS_AUDIT and SALARY_AUDIT. XML tree of SALARY_AUDIT stores only of salary history of one employee only. Tom's salary has increased once. His auditing salaries in XML tree are in two child-nodes as shown in Table VII. XML schema for the salary auditing is shown in Fig. 1.

TABLE VII.
COLUMN-BASED XML TABLE TO AUDIT ADDRESS AND SALARY

ID	Name	DOB	Address	Address _Audit	Hired Date	Salary	Salary _Audit
101	Tom	1980-01-01	New York		2008-01-01	55000	
102	Ann	1985-06-16	London		2009-01-01	60000	
103	Jack	1975-01-01			2008-01-01		

Pros

- There is no extra auditing table.
- Auditing data is separated in different columns but under the same record. This is the same with ORDB approach.
- Size of table is as small as the number of records except the deleted records which is maintained for the auditing purpose. We can create view to retrieve only current employee by using XML/SQL to choose only XML node whose "operationType" element is "D". Setting the salary of deleted employee to an invalid value such as -1 would simplify the query to SQL only.

Cons

- It requires extra knowledge of XQUERY of developers.
- Joining two or more XML tree may be needed for a query that needs to access two or more auditing columns.

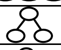
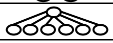
```
<schema >
 <element name="auditing">
   <complexType>
    <sequence>
      <element minOccurs="1" name="history" type="historyType"/>
    </sequence>
   </complexType>
</element>
 <complexType name="salaryHistoryType">
    <sequence>
      <element minOccurs="1" name="value" type="salaryType"/>
      <element minOccurs="1" name="startdate" type="Date"/>
      <element minOccurs="0" name="enddate" type="Date"/>
     <element minOccurs="1" name="operation" type="operationType"/>
     <element minOccurs="1" name="user" type="string"/>
    </sequence>
    <attribute name='id' type='string'/>
 </complexType>
 <simpleType name="operationType">
     <restriction base="string">
        <enumeration value="I"/>
        <enumeration value="U"/>
        <enumeration value="D"/>
      </restriction>
   </simpleType>
 <simpleType name="salaryType">
     <union memberTypes="salary0  salaryRange"/>
 </simpleType>
  <simpleType name="salary0">
      <restriction base="decimal">
         <enumeration value="0"/>
      </restriction>
   </simpleType>
    <simpleType name="salaryRange">
      <restriction base="decimal">
          <minInclusive value="10000"/>
          <maxInclusive value="1000000"/>
      </restriction>
   </simpleType>
</schema>
```

Figure 1. XML Schema for column-based XML auditing of Salary_Audit column in Employee table

2) *Row-based XML Auditing*

To combine auditing information of entity into a single XML tree, row-based XML auditing merges child nodes of every XML trees of the same record of column-based XML auditing into one tree under the same column as shown in Table VIII. An attribute named “column” of “history” element in XML Schema in Fig. 2 identifies the auditing column.

TABLE VIII.
ROW-BASED XML TABLE FOR AUDITING

ID	Name	DOB	Address	Hired Date	Salary	Audit
101	Tom	1980-01-01	New York	2008-01-01	55000	
102	Ann	1985-06-16	London	2009-01-01	60000	
103	Jack	1975-01-01		2008-01-01		

Pros

• Auditing data is not retrieved often. Separating it in a column can reduce the overhead of business operational functions.

• XML data still requires larger spaces comparing to relational data. Commercial data engines tend to store them separately from the relational data. Therefore, it does not increase the number of data blocks that contain only relational data.

• Physically, combining all XML data from different logical columns into a single column would simplify data retrieval. Another word, an Xquery expression can retrieve information from a single source.

Cons

• Design of auditing data does not divide information logically.

• Xquery on row-based XML auditing will be more complex than of column-based.

```
<schema >
 <element name="auditing">
   <complexType>
    <sequence>
      <element minOccurs="1" name="history" type="historyType"/>
    </sequence>
   </complexType>
  </element>
   <complexType name="historyType">
     <sequence>
       <element minOccurs="1" name="value" type="string"/>
       <element minOccurs="1" name="startdate" type="Date"/>
       <element minOccurs="0" name="enddate" type="Date"/>
       <element minOccurs="1" name="operation" type="operationType"/>
        <element minOccurs="1" name="user" type="string"/>
      </sequence>
      <attribute name='id' type='string'/>
      <attribute name='column' type='columnType'/>
   </complexType>
   <simpleType name="operationType">
       <restriction base="string">
           <enumeration value="I"/>
           <enumeration value="U"/>
           <enumeration value="D"/>
       </restriction>
   </simpleType>
    <simpleType name="columnType">
      <restriction base="string">
          <enumeration value="Address"/>
          <enumeration value="Salary"/>
        </restriction>
     </simpleType>
</schema>
```

Figure 2. XML Schema for row-based XML auditing of Employee table

Our XML approach on temporal data suggests the use of XML elements to demonstrate the auditing data primarily. On the other hand, [3] suggests the use of XML attributes for temporal data. Their data stream management framework for historical XML data proposed XCQL, which is a XQuery Language for Continuous Queries for the temporal extension. Our approach could be simply adapted to their scheme when it is adopted by commercial databases.

VI. CONCLUSION AND DISCUSSIONS

Table space (or disk space) of auditing table should be separated from the operational table. Database engine could run an auditing query faster than running it against one large table containing both operational and auditing data. Even though, we partition one large table into two table spaces. Overhead of checking which partition will be used against the query is added to execution time. Also, DBA would manage the DBMS easier.

To retrieve auditing information, XQuery such as FLWER expression, in SQL/XML provides us access to elements and attributes in XML data.

XML indexing is adding to commercial databases while semantic searching on XML is on horizon [14]. These issues and others will provide support to semi-structured approaches.

We suggest a various number of solutions of database auditing. Some solutions are suitable for databases that do not efficiently support semi-structured data. While semi-structure data are emerging into some commercial databases.

Database auditing is a major concern in some systems. Choosing the right design would prevent the performance deterioration, reduce data redundancy, save the storage, and simplify auditing queries.

REFERENCES

[1] G. Bhargava, S. K. Gadia, "Relational Database Systems with Zero Information Loss" IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 1, pp. 76-87, February 1993

[2] X. Chen, I. Petrounias, "A Framework for Temporal Data Mining" Database and Expert Systems Applications, Springer Berlin / Heidelberg pp. 796-805 Vol. 1460/1998

[3] S. Bose, L. Fegaras, "Data Stream Management for Historical XML Data", SIGMOD 2004 Paris, France, pp. 239-250, June 13-18, 2004

[4] R. F. Chong, X. Wang, M. Dang, D. R. Snow, "Understanding DB2 Learning Visually with Examples", 2nd edition, 2007

[5] "Oracle VARRAY Examples" http://www.dba-oracle.com/tips_oracle_varray.htm

[6] "Introduction to the DB2 Universal Database (DB2 UDB) audit facility" http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/admin/ c0005483.htm

[7] W. Lu, G. Miklau, "Auditing a Database Under Retention Restrictions", IEEE Inter. Conf. on Data Eng., ICDE 2009, pp. 42-53

[8] D. Luebbers, U. Grimmer, M. Jarke1, "Systematic Development of Data Mining-Based Data Quality Tools", proc. of the 29th VLDB Conference, pp. 548 - 559, Berlin, Germany, 2003

[9] "Understanding SQL Server Audit" http://msdn.microsoft.com/en-us/library/cc280386.aspx

[10] "Simple Data Auditing" http://forge.mysql.com/wiki/SimpleDataAuditing

[11] "Logging, Auditing, and Monitoring the Directory" http://download.oracle.com/docs/cd/B14099_19/idmanage.1012/b14082/logging.htm#i126963

[12] The PostgreSQL Global Development Group, "Postgre SQL 8.4.0 Documentation" http://www.postgresql.org/files/documentation/pdf/8.4/postgresql-8.4.0-A4.pdf

[13] N. H. Park, W. S. Lee, "Anomaly Detection over Clustering Multi-dimensional Transactional Audit Streams", IEEE International Workshop on Semantic Computing and Applications IWSCE 2008, pp. 78-80

[14] U. Supasitthimethee, T. Shimizu, M. Yoshikawa, K. Porkaew, "Meaningful Interrelated Object Tree for XML Keyword Search", 2nd International Conference on Computer and Automation Engineering, Singapore, pp. 339-344, January 26 – 28, 2010

[15] N. Waraporn, K. Porkaew, "Null Semantics for Subqueries and Atomic Predicates" IAENG International Journal of Computer Science, Vol. 35, Issue 3, pp. 305-313, September 2008

[16] Li Yang, "Teaching Database Security and Auditing", SIGCSE'09, pp. 241-245, March 4–7, 2009