Egyptian Computer Science Journal Vol. 38 No. 2 May 2014 ISSN-1110-2586

Performance Issues Concerning Storage of Time-Variant Data

Dušan Petković
University of Applied Sciences
Hochschulstr. 1, Rosenheim, 83024, Germany
petkovic@fh-rosenheim.de

Abstract

Time-varying data management has been an area of active research within database systems for almost 25 years. Apart from the numerous data models that were investigated and implemented for temporal databases, several other design trade-off decisions have to be considered. One of these concerns how the temporal attributes are stored. This article examines the performance implications for two different forms of storage of time-variant data: attribute and tuple timestamping. We store the same data both using attribute timestamping and using tuple timestamping and perform the same 36 queries on both. The queries vary with respect to the depth of nesting within the data tupels. The results of these experiments show that attribute timestamping performs better for queries with low levels of nesting - the more complex the nesting structure of the query, however, the better the performance of tuple timestamping.

Keywords: *temporal data, attribute timestamping, tuple timestamping, query execution time.*

1. Introduction

A database represents a model of the real world. During the lifetime of an object stored in a database, its properties can change. For such objects it is necessary to consider their time-variant aspects. For this reason, it is an important task of database systems to support management of temporal data, i.e. that current, past and future values of time-variant attributes can be persistently stored.

There are three time dimensions, which are independent to each other: user-defined, valid and transaction time. User-defined time is a time representation designed to meet the specific needs of users. Valid time specifies when certain conditions in the real world are, were or will be valid. Transaction time automatically captures changes made to the state of time-variant data in a database. This time dimension represents the time period during which an instance is recorded in the database. Several temporal data models, which support either valid or transaction time (or both of them) are discussed in [17]. Several issues in terms of valid time and transaction time has been discussed in [3].

There is also another form of time-variant data, called bitemporal data. Bitemporal data are union of valid time and transaction time data. Several data models concerning bitemporal data are known in the literature: The Bitemporal Conceptual Data Model (BCDM), which has been introduced in [9], is a very simple model capturing the essential semantics of time-variant relations. Another example of a bitemporal model is Nested Bitemporal Relation Data Model (NBRDM), introduced and described in [2]. The model, as the name implies, is based upon a nested relational schema.

One of the most important distinctions between existing temporal models is the choice between attribute and tuple timestamping. While tuple timestamping is based upon the first normal form (1NF) to store time-variant data, attribute timestamping uses non-first normal form (NF^2) for the same goal. Each of these alternatives has its advantages and disadvantages. Tuple timestamping, storing only atomic values in a table's column, remains within relational model. Therefore, in this case relational tables are used to represent time-invariant as well as time-variant data. The disadvantage of this approach is increase of data redundancy, because attribute values that change over time are repeated in multiple rows of a table. On the other hand, the advantage of attribute timestamping is that values of time-variant attributes are grouped together and stored as a unit, in one column. Thus, all time varying attributes can be stored in one relation. The disadvantage of attribute timestamping is that it may not be capable of efficiently using existing storage structure. Therefore, all well-known and high effective relational techniques, such as query evaluation, for instance, cannot be used.

The aim of this article is to examine whether attribute-timestamping (i.e. storing time-variant attributes in non-first-normal form) reduces execution time in relation to tuple timestamping (i.e. storing time-variant attributes in first normal form). For this study, we use data of employees from a hypothetical company database with 10 years of past and future data introduced in [1].

1.1 Related Work

As we already stated, numerous temporal models and query languages have been proposed. An annotated bibliography on temporal aspects of data can be found in [6]. The glossary of temporal database concepts is given in [8]. Taxonomy for the classification of temporal databases according to time dimensions has been developed in [15]. According to this taxonomy, Gadia's work, described in [17] is a temporal data model concerning valid time. Ben-Zvi proposed the first data model for bitemporal databases, a temporal query language, storage architecture, indexing, recovery, concurrency, synchronization, and its implementation [5]. Jensen et al. attached four implicit attributes to each time-varying relation, and presented a corresponding temporal query language [9]. The bitemporal data model, BCDM, forms the basis for Temporal Structured Query Language [16,11]. The model is based upon tuple timestamping and only one temporal attribute is allowed in a relation, unless several attributes change synchronously. The review of this model is given in [4].

The ISO specification, which contains the standardized temporal model, is given in [7]. The most important new features in the standardized model are described in [10, 13]. The only existing implementation of the model exists for IBM DB2 and is discussed in [12,14].

Performance issues in relation to temporal data have been investigated in several articles. The first performance tests have been published in [15]. In her work [1], Atay compared attribute and tuple timestamping in relation to the NBRDM data model. Our work is similar to the work of Atay. In contrast to our tests, Atay uses rather small group of queries for testing and her goals are different than ours.

1.2 Roadmap

The rest of this article is organized in the following way: Section 2 describes how the data is stored. First, the creation of the **employee** table with all nested data is shown as well as the way how the data is loaded into the table. Second, the creation and loading of six relational tables, which are logically equivalent to the **employee** table mentioned above is shown. Section 3 discusses all queries used for testing. They are grouped in several different and orthogonal groups, depending on existing temporal dimensions as well as the way how they are stored. Section 4 presents our test results, while the last section gives conclusions and discusses future work.

2. Creation of Temporal Data

In this chapter we show the structure and content of tables, which were used for testing. For this study, we use data of employees from a hypothetical company database with 10 years of past and future data. The data used for the tests of attribute timestamping is equivalent to that used for the tests of tuple timestamping, although the structure of the tables differs. The following subsection shows the creation of the attribute timestamping data.

2.1 Attribute Timestamping Data

We use Oracle's concept of nested tables to create the **employee** table with time-variant columns, whose values are stored as a unit i.e. using the attribute timestamping approach. The **employee** table contains six columns. The columns **emp#** and **birth_date** are time-invariant, while the other four columns (**name**, **address**, **dept_mng** and **salary**) are time-variant. The structure of the **employee** table, with all its columns is given in Table 1. Examples 1 till 4 show the creation of the **employee** table and of all necessary auxiliary (object) types.

Table 1. The structure of the employee table

EMP#	NAME	ADDRESS	BIRTH_DATE	DEPT_MNG		SALARY
	Name-history	Address-history		MANAGER	DEPARTMENT	Salary-history
				Manager-history	Department-history	

Example 1

```
CREATE TYPE BITEMPORAL_VARCHAR AS OBJECT (
    TT_LB DATE,
    TT_UB DATE,
    VT_LB DATE,
    VT_UB DATE,
    VALUE_PART  VARCHAR2(50));

CREATE TYPE BITEMPORAL_NUMBER AS OBJECT (
    TT_LB DATE,
    TT_UB DATE,
    VT_LB DATE,
    VT_UB DATE,
    VALUE_PART  NUMBER);
```

Example 1 defines two object types, **bitemporal_varchar** and **bitemporal_number**, which are used to specify time-variant attributes. (The VT_LB and VT_UB columns specify the lower and upper bound of the valid time data, while TT_LB and TT_UB specify the lower and upper bound of the transaction time data, respectively.)

Example 2

```
CREATE TYPE NAME AS TABLE OF BITEMPORAL_VARCHAR;
CREATE TYPE ADDRESS AS TABLE OF BITEMPORAL_VARCHAR;
CREATE TYPE DEPARTMENT AS TABLE OF BITEMPORAL_VARCHAR;
CREATE TYPE MANAGER AS TABLE OF BITEMPORAL_VARCHAR;
CREATE TYPE SALARY AS TABLE OF BITEMPORAL_NUMBER;
```

Example 2 creates five nested table types (**name**, **address**, **department**, **manager** and **salary**). These types will be subsequently used as a data type of corresponding time-variant columns. The first four types, which contain alphanumerical values, are specified using the BITEMPORAL_VARCHAR object type, while the last one (**salary**) contains numerical values and is defined using the BITEMPORAL_NUMBER type.

Example 3

```
CREATE TYPE TYPE_DEPT_MNG AS OBJECT (
     DEPARTMENT_HISTORY   DEPARTMENT,
     MANAGER_HISTORY   MANAGER );
CREATE TYPE DEPT_MNG AS TABLE OF TYPE_DEPT_MNG;
```

As can be seen from Table 1, two nested columns (of types DEPARTMENT and MANAGER) are grouped together and build a new nested column of type DEPT_MNG. Example 3 creates first the grouping of the two columns and after that the new compound table type, **dept_mng**, which is based on the **type_dept_mng** object type. Example 4 shows the creation of the **employee** table with all its columns, which are based upon object types specified in Examples 1 till 3. In other words, the data types of columns of the **employee** table are object types created in previous examples.

Example 4

```
CREATE TABLE EMPLOYEE (
  EMP# NUMBER primary key,
  NAME NAME,
  ADDRESS  ADDRESS ,
  BIRTH_DATE DATE,
  DEPT_MNG DEPT_MNG,
  SALARY SALARY)
NESTED TABLE NAME STORE AS NAME_TABLE,
NESTED TABLE ADDRESS STORE AS ADDRESS_TABLE,
NESTED TABLE DEPT_MNG STORE AS DEPT_MNG_TABLE
(NESTED TABLE MANAGER_HISTORY STORE AS MANAGER_TABLE,
  NESTED TABLE DEPARTMENT_HISTORY STORE AS DEPARTMENT_TABLE),
NESTED TABLE SALARY STORE AS SALARY_TABLE;
```

Example 5

```
INSERT INTO EMPLOYEE VALUES (101,NAME
(BITEMPORAL_VARCHAR(sysdate,'09.09.9999','01.07.2005','09.09.9
999','Ed Y')), ADDRESS (

BITEMPORAL_VARCHAR(sysdate,'09.09.9999','01.07.2005','09.09.9999',
'BUCA')),
'03.10.1966',
DEPT_MNG(TYPE_DEPT_MNG(DEPARTMENT
(BITEMPORAL_VARCHAR(sysdate,'09.09.9999','01.06.2006','09.09.9999'
,'Im')),

MANAGER(BITEMPORAL_VARCHAR(sysdate,'09.09.9999','18.10.2002','
09.09.9999', 'Tanja X')))),
SALARY(BITEMPORAL_NUMBER(sysdate,'09.09.9999',
sysdate,'09.09.9999',250)));
```

Example 5 shows an INSERT statement, which inserts a tuple in the **employee** table. This statement is here just to show how rows are inserted. For out tests, the **employee** table is loaded with 10,000 tuples, which includes the temporal data of all employees within 10 years.

2.2 Tuple Timestamping Data

Data, which represent the tuple timestamping approach are stored in six tables, whose structure is presented in Table 2. Example 6 shows the creation of all these tables.

Table 2. The structure of six relational tables

EMPNAME	Empid	Name	tt_lb	tt_ub	vt_lb	vt_ub
EMPADDRESS	Empid	Address	tt_lb	tt_ub	vt_lb	vt_ub
EMPBIRTH	Empid	birth_date				
EMPDEPARTMENT	Empid	department	tt_lb	tt_ub	vt_lb	vt_ub
EMPMANAGER	Empid	Manager	tt_lb	tt_ub	vt_lb	vt_ub
EMPSALARY	Empid	Salary	tt_lb	tt_ub	vt_lb	vt_ub

Example 6

```
CREATE TABLE empname
 (empid INT NOT NULL PRIMARY KEY,  name CHAR(20),
  tt_lb DATE,  tt_ub DATE,  vt_lb DATE,  vt_ub DATE);
CREATE TABLE empbirth
(empid INT NOT NULL PRIMARY KEY,  birthdate DATE);
CREATE TABLE empaddress
 (empid INT NOT NULL PRIMARY KEY, address VARCHAR(30),
   tt_lb DATE, tt_ub DATE, vt_lb DATE, vt_ub DATE);
CREATE TABLE empdepartment
 (empid INT NOT NULL PRIMARY KEY,  dname CHAR(20),
    tt_ub DATE, tt_ub DATE, vt_lb DATE, vt_ub DATE);
CREATE TABLE empmanager
 (empid INT NOT NULL PRIMARY KEY, man_name CHAR(20),
     tt_lb DATE, tt_ub DATE, vt_lb DATE, vt_ub DATE);
CREATE TABLE empsalary
 (empid INT NOT NULL PRIMARY KEY, salary DEC (11,2),
  tt_lb DATE,  tt_ub DATE, vt_lb DATE, vt_ub DATE);
```

As can be seen from Table 2, all six tables have the same primary key, **empid**, which corresponds to the **emp#** column in Example 4. The only time-invariant table, **empbirth**, has an additional column (**birth_date**), which corresponds to the column with the same name in the **employee** table from Example 4. All other tables contain time-variant data and are created with four additional columns, two for valid time lower and upper bounds (VT_LB, VT_UB) and two for transaction time lower and upper bounds (TT_LB, TT_UB). The structure of tables in Example 6 is logically equivalent to the structure of the **employee** table in Example 4. These relational tables are loaded with the same content as the **employee** table. The main difference is that the time-variant data in the **employee** table is stored using the attribute timestamping approach, while the data in Example 6 is loaded using the tuple timestamping approach.

3. Time-Variant Queries

This section describes queries used for performance testing. All queries are grouped in two main groups. In the first group, there are three subgroups concerning time dimensions, i.e. one subgroup comprises the queries in relation to valid time, the second one in relation to transaction time and the last one in relation to bitemporal data. The second main group divides the queries in relation to the number of nesting levels, which must be referenced to execute the particular query. In other words, queries concerning the attribute timestamping approach can use only time-variant data from the first level of nesting or first and second level and so on. So, the three subgroups of queries for the second group can be described as follows:

a. The data used in a query for attribute timestamping concern only the first nesting level
b. The data used in a query for attribute timestamping concern the first and second nesting levels. In the case of the second level only one nested table is referenced.
c. The data used in a query for attribute timestamping concern several nested tables at the second level.

For our tests we used 36 queries grouped in nine subgroups. The first subgroup, for instance, comprises six queries concerning valid time and they use data only from the first nesting level (Case a.). Table 3 shows all different combinations used to build the nine groups of queries.

Table 3: Nine different subgroups of queries

Valid time/ Case a.	Transaction time/Case a.	Bitemporal/ Case a.
Valid time/ Case b.	Transaction time/Case b.	Bitemporal/ Case b.
Valid time/ Case c.	Transaction time/Case c.	Bitemporal/ Case c.

Example 7 shows a query, which is related to valid time and concerns Case a., described above. The only difference in Example 8 is that it uses relational tables to solve the same task as Example 7. As can be seen from these examples, only the **salary** table has been concerned, and the nesting level is 1.

Example 7

```
 --Which salary values are stored in the database in the time period between 1.1.2001 and 1.6.2001?
SELECT e.emp#, sal.value_part
FROM EMPLOYEE e,
TABLE (e.salary) sal
WHERE sal.VT_LB  BETWEEN '01.01.2001' AND '01.06.2001';
```

Example 8

```
--Which salary values are stored in the database in the time period between 1.1.2001 and 1.6.2001?
SELECT  nam.empid, sal.salary
FROM EMPNAME name, EMPsalary sal
WHERE  nam.EMPid = sal.EMPid
AND sal.vT_lB BETWEEN '01.01.2001' AND '01.06.2001';
```

Example 9 shows a query, which references bitemporal data and concerns Case b. Example 10 solves the same problem, but uses relational instead of nested tables. As can be seen from Example 9, besides the **dept_mng table**, which is at the first nesting level, query also references the **manager_history** table, which is at the second level of nesting.

Example 9

```
-- Who was the manager of the employee named Can Atay in the time period between 6.6.2003 and 1.1.2007.
-- The data is stored in the database in the time period between 6.6.2003 and 8.8.2007.
SELECT E.EMP#,  MAN.VALUE_PART AS MANAGER,
                MAN.VT_LB, MAN.VT_UB, MAN.TT_LB, MAN.TT_UB
  FROM EMPLOYEE E, TABLE(E.NAME) NAM,
                   TABLE(E.DEPT_MNG) DEP_MAN,
                    TABLE(DEP_MAN.MANAGER_HISTORY) MAN
  WHERE NAM.VALUE_PART = 'CAN ATAY' AND (( MAN.VT_LB > '06.06.2003'
     AND MAN.VT_LB <'01.01.2007')
     OR (MAN.VT_UB > '06.06.2003'
     AND MAN.VT_UB <'01.01.2007')
     OR (MAN.VT_UB > '01.01.2007'
     AND MAN.VT_LB <'06.06.2003'))
     AND ((MAN.TT_LB > '06.06.2003'
     AND MAN.TT_LB <'08.08.2007')
     OR (MAN.TT_UB > '06.06.2003'
     AND MAN.TT_UB < '08.08.2007')
     OR (MAN.TT_UB > '08.08.2007'
     AND MAN.TT_LB <'06.06.2003'));
```

Example 10

```
-- Who was the manager of the employee named Can Atay in the time period between 6.6.2003 and 1.1.2007.
--  The data is stored in the database in the time period between 6.6.2003 and 8.8.2007)
 SELECT e.empid, m.manager,  M.VT_LB, M.VT_UB,   M.TT_LB, M.TT_UB
  FROM EmpNAME e, EMPMANAGER m
  WHERE e.eNAME = 'CAN ATAY' AND M.EMPid = E.EMPid
                   AND (( M.VT_LB > '06.06.2003' AND M.VT_LB <'01.01.2007')
                   OR (M.VT_UB > '06.06.2003' AND M.VT_UB <'01.01.2007')
                   OR (M.VT_UB > '01.01.2007' AND M.VT_LB <'06.06.2003'))
                  AND ((M.TT_LB > '06.06.2003' AND M.TT_LB <'08.08.2007')
                  OR (M.TT_UB > '06.06.2003' AND M.TT_UB < '08.08.2007')
                  OR (M.TT_UB > '08.08.2007' AND M.TT_LB <'06.06.2003'));
```

Example 11 shows a query, which references bitemporal data and concerns Case c. The only difference in Example 12 is that uses relational tables to solve the same task as Example 11. These two queries (as all other tested queries from this subgroup) retrieve data from the second level of nesting *twice*, from the **manager_history** table as well as from the **department_history** table.

Example 11

```
SELECT E.EMP#,NAM.VALUE_PART, MAN.VALUE_PART AS MANAGER,
MAN.VT_LB, MAN.VT_UB,
DEP.TT_LB, DEP.TT_UB
FROM EMPLOYEE E, TABLE(E.NAME) NAM,
TABLE(E.DEPT_MNG) DEP_MAN,
TABLE(DEP_MAN.MANAGER_HISTORY) MAN,
TABLE (DEP_MAN.DEPARTMENT_HISTORY) DEP
WHERE((
MAN.TT_LB  >'01.01.1995' AND MAN.VT_LB <'03.03.1996')
OR (MAN.VT_UB >'01.01.1995'
AND MAN.VT_UB <'03.03.1996')
OR (MAN.VT_UB >'03.03.1996'
AND MAN.VT_LB < '01.01.1995'))
AND DEP.TT_LB = '01.01.1995'
AND DEP.TT_UB > '01.01.1995';
```

Example 12

```
SELECT M.empid, M.MANAGER AS MANAGER,
M.VT_LB, M.VT_UB,
M.TT_LB, M.TT_UB
FROM empmanager m, empname nam, empdepartment d
WHERE (( M.VT_LB >'01.01.1995' AND M.VT_LB<'03.03.1996') OR
(M.VT_UB >'01.01.1995'
AND M.VT_UB<'03.03.1996')OR (M.VT_UB >'03.03.1996'
AND M.VT_LB < '01.01.1995'))
AND d.TT_LB = '01.01.1995'
AND d.TT_UB > '01.01.1995' AND nam.empid= m.empid
AND d.empid = nam.empid;
```

4. Test Results

We used the **autotrace** utility of Oracle to perform tests discussed in this article. Each query is executed five times, and the average of all execution times is calculated. Before executing a query, the following statement has been executed:**alter system flush shared_pool;** This statement empties the buffer cache of the system and, after that, the system uses “cold” buffer for the execution of each query.

Note that we made tests in terms of time and space. We do not publish our tests in relation to space, because they are similar to those published in [1]. Our tests in terms of time showed that there is no significant difference in execution time for queries using nested tables (for attribute timestamping) and relational tables (for tuple timestamping) concerning time dimensions (valid time, transaction time and bitemporal). The only difference exists for the level of nesting in case of the attribute timestamping approach. In other words, the execution time of queries using attribute timestamping is faster only for a group of queries, which use data from the first level of nesting (Case a., above). The more complex nesting structure, the better performance of queries based on tuple timestamping.

Figure 1 shows the average execution time of queries from Case a. As can be seen from the figure, the valid time queries execute faster for nested tables than for equivalent relational tables. In other words, execution time of queries, such as in Example 7, is on average 1.5 times faster than queries, such as in Example 8. (The same is true for transaction time and bitemporal.)

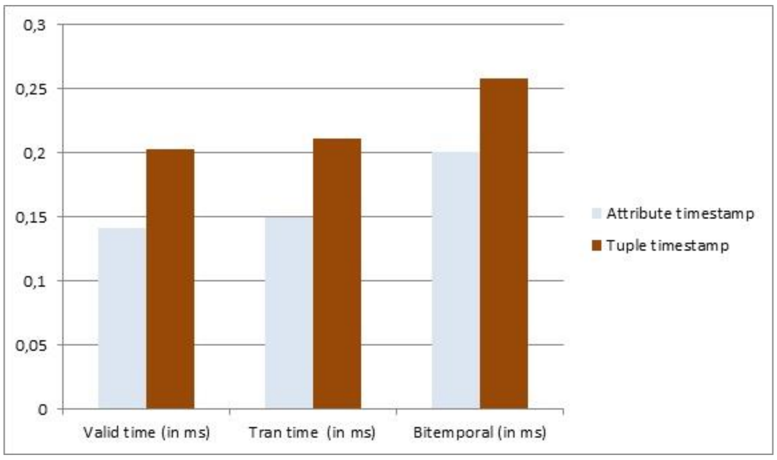
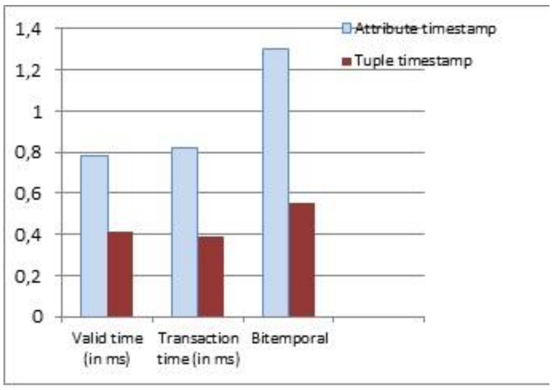


Figure 1: Average execution times of queries from Case a.

Figure 2 shows the average execution time of queries from Case b. As can be seen from the figure, the bitemporal queries execute faster for relational tables than for equivalent nested tables. In other words, execution time of queries, such as in Example 10, is on average two times faster than queries, such as in Example 9. (The same is true for valid and transaction time.)



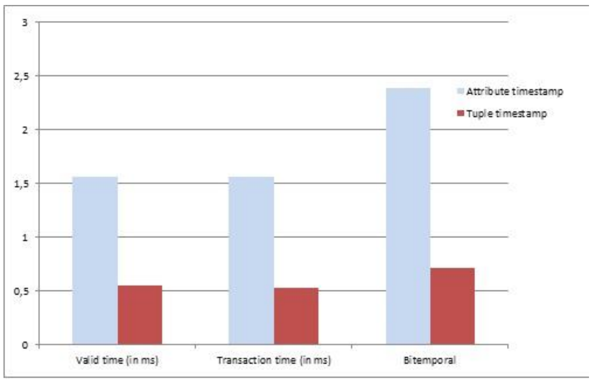


Figure 2: Average execution times: Case b. **Figure 3: Average execution times: Case c**

Figure 3 shows the average execution time of queries from Case c. As can be seen from the figure, the bitemporal queries execute on average three times faster for relational tables than for equivalent nested tables. (The same is true for valid time and transaction time.)

5. Conclusions and Future Work

In this article, we have shown how storage of time-variant data in relation to the attribute and tuple timestamping approaches influences the execution time of corresponding queries. Attribute timestamping stores object's time-variant and time-invariant data in a single tuple rather than splitting it into several tuples. Therefore, the whole information concerning an entity can be stored in a tuple. The advantages of this approach are that it avoids redundancy and is more expressive.

The tuple timestamping approach supports the first normal form and adds four time-variant columns, two for valid time, and two for transaction time. This approach splits the object's history into several tuples that create redundancy for non-variant columns. The more time-variant attributes, the higher the increase in data redundancy. Partitioning time-variant attributes into separate tables reduces data redundancy, which results in many relations.

We have evaluated the execution times of several subgroups of time-variant queries to gain insight into their performance. Our conclusion is that the only difference exists for the level of nesting in a case of the attribute timestamping approach. On the other hand, there is no significant difference in execution times of queries using nested tables (for attribute timestamping) and relational tables (for tuple timestamping) concerning time dimensions (valid time, transaction time and bitemporal).

Concerning future work, there are several possible directions. First, it would be interesting to index nested tables as well as relational ones and perform the same performance tests as in this article. Second, XML provides intrinsic support for attribute timestamping. Therefore, a direction of research could be to represent nested tables in XML and carry out a performance evaluation of them.

References

[1] Atay, C. – A Comparison of Attribute and Tuple Time Stamped Bitemporal Relational Data Models, Proc. of the Int. Conf. on Applied Computer Science, p. 479-489, 2010.

[2] Atay, C.; Tansel, A.U. – Bitemporal Databases: Modeling and Implementations, VDM Verlag, 2009.

[3] Böhlen, M.H., Snodgrass, R.T., Soo, M.D. - Coalescing in Temporal Databases, VLDB, 1996.

[4] Darwen, H,; Date, C.J. – An Overview and Analysis of Proposals Based on the TSQL2 Approach, www.dcs.warwick.ac.uk/~hugh/TTM/OnTSQL2.pdf

[5] Gadia, S. - Ben-Zvi's Pioneering Work in Relational Temporal Databases,. in A. Tansel et al., editors, Temporal Databases (pp. 202 – 207), Benjamin/Cummings, 1993.

[6] Grandi, F. - Introducing an Annotated Bibliography on Temporal and Evolution Aspects in the Semantic Web, SIGMOD Records, Vol. 41, No.4, 2012.

[7] ISO/IEC 9075-2:2011: Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation), 2011.

[8] Jensen, C.S., Dyreson, C.E. (Eds.) - The consensus glossary of temporal database concepts , in Etzion, O et al. - Temporal Databases — Research and Practice, p. 367–405, Springer-Verlag, Heidelberg, 1998.

[9] Jensen, C.S. (ed.) – The TSQL Benchmark, Proc. of the Int. Workshop on an Infrastructure for Temporal Databases, 1993.

[10] Kulkarni, K., Michels, J. - Temporal Features in SQL:2011, SIGMOD Records, Vol. 41, No. 3, 2012.

[11] Lorentzos, N.A. - The Interval-extended Relational Model and Its Applications to Valid Time, Temporal Databases, 1993.

[12] Nicola, M.; Sommerlandt, M. Managing time in DB2 with temporal consistency, http://www.ibm.com/developerworks/data/library/techarticle/dm-1207db2temporalintegrity/ , 2011.

[13] Petković, D. - Was lange währt, wird endlich gut: Temporale Daten im SQL-Standard, Datenbank-Spektrum, 13 (2): p. 131-138, 2013 (in German).

[14] Saracco, C., Nicola, M., Gandhi, L. - A matter of time: Temporal data management in DB2, www.ibm.com/developerworks/data/library/techarticle/dm-1204db2temporaldata/dm-1204db2temporaldata-pdf.pdf , 2012.

[15] Snodgrass, R; Ahn, I. - Performance Evaluation of a Temporal Database Management System, Communications of ACM, 1986.

[16] Snodgrass, R. (Ed.), The TSQL2 Temporal Query Language, Kluwer, Dordrecht, 1995.

[17] Tansel, A.U.; Clifford, J.; Gadia, S.; Jajodia, S. Segev, A.; Snodgrass, R. – Temporal Databases, 1993.