Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2014) 22: 479 – 498
© TÜBİTAK
doi:10.3906/elk-1207-100

BtSQL: nested bitemporal relational database query language

Canan Eren ATAY[1,*], **Abdullah Uz TANSEL**[2]
[1]Department of Computer Engineering, Dokuz Eylül University, İzmir, Turkey
[2]Department of Computer Information Systems, Baruch College, City University of New York, New York City, New York, USA

Received: 25.07.2012 • **Accepted:** 23.11.2012 • **Published Online:** 17.01.2014 • **Printed:** 14.02.2014

Abstract: A nested bitemporal relational data model and its query language are implemented. The bitemporal atom (BTA) is the fundamental construct to represent temporal data and it contains 5 components: a value, the lower and upper bounds of valid time, and the lower and upper bounds of the recoding time. We consider 2 types of data structures for storing BTAs: 1) string representation and 2) abstract data-type representation. We also develop a preprocessor for translating a bitemporal structured query language (BtSQL) statement into standard SQL statements. The BtSQL includes the select, insert, delete, and update statements of the SQL, extended for bitemporal relational databases. It supports bitemporal, historical, and current context. Bitemporal context is for auditing purposes, historical context is for querying past states of a bitemporal database, and current context is for querying the snapshot state of a bitemporal database. We also evaluate the performance of the 2 alternative implementation methods by considering retrieval, insertion, and update queries.

Key words: Bitemporal database, nested bitemporal relational model, bitemporal atom type, bitemporal query, BtSQL

1. Introduction

It is difficult, if not impossible, to identify a substantial computer application that does not change as time progresses. Consider student data, which may include past, present, and future data on enrollments, grades, degree programs, and degrees awarded. As another example, employee histories typically include past, present, and future data on salary, department, and title; all of these attributes change over time. In the financial markets, businesses must track the cash flow or account balances over time for each customer. Other examples of ever-changing time-related data include patient medical records, with diagnoses, X-rays, and lab tests; stock market data; reservation systems for airlines, car rentals, and hotels; spatial databases; and data warehousing records. Databases, in general, maintain the recent state of the domain modeled, whereas built-in time-management support can greatly increase the functionality of a database application. A temporal database maintains an object's past, present, and (if available) future values of data.

A temporal database stores the history of the objects (valid time) or the history of the database activity (transaction time). Valid time captures the history of an object but does not preserve the history of retroactive and postactive changes. Transaction time records the changes in the database; nevertheless, it does not carry historical or future data. Bitemporal database systems maintain both transaction and valid time.

The tuple time-stamping approach splits the object's history into several tuples that create redundancy.

*Correspondence: canan@cs.deu.edu.tr

In case there is more than one temporal attribute, new values on each attribute significantly increase the redundancy. Time-stamped attributes, on the other hand, store the attribute value together with its timestamp in an attribute time-stamping approach. Each attribute stores the history of the values and each tuple has the object's whole history. Only values in a tuple that are updated have to be changed; the others remain the same.

In this paper, we intend to discuss our previously proposed Nested Bitemporal Relational Model (NBRM) [1], in which the valid and transaction timestamps are attached to attributes and more than one level of nesting is allowed to represent the histories of the entities and their relationships. The NBRM is built on top of an object-relational database management system (RDBMS) that supports abstract data types and nested relations [2,3]. Using nested relations overcomes the problems discussed above in tuple time-stamping. Naturally nested relations are more complex; however, we believe that is worthwhile in managing temporal data.

We implement an attribute time-stamping approach on a conventionally available database and show that the proposed model is utilized successfully with bitemporal, current, and historical contexts. We consider alternative implementation approaches, a bitemporal atom (BTA) represented as a string (BTA_String) or as an abstract data type (BTA_ADT) stored in a collection type, and use this prototype for the performance evaluation of these approaches. The bitemporal relational algebraic operators slice and rollback are implemented, which are peculiar for the temporal data. The slice operator is implemented for the first time for nested bitemporal relational databases using the attribute time-stamping approach. The tests show that both new operators functionally perform well. We also develop a preprocessor for the bitemporal structured query language (BtSQL), designed for translating BtSQL statements into standard SQL statements. We successfully manage to hide tedious bitemporal query specifications from the user. For portability issues, the prototype is implemented in the Java programming language.

Section 2 discusses some related work. In Section 3, a NBRM is described. Section 4 discusses the implementation of different BTA types and describes how these types are stored in nested tables, how the nested bitemporal relational algebra is implemented, and how the preprocessor works. Section 5 provides the evaluation of each implementation method, with a set of updates and queries. Section 6 gives the performance evaluation and Section 7 has the conclusions and future work plans.

2. Related work

Information systems have been researched in many aspects for decades and the time-related area is not an exception [4]. Tuple time-stamping and attribute time-stamping are 2 common approaches widely followed by temporal database researchers. The tuple time-stamping approach adds 2 special time attributes (i.e. BEGIN and END) to first normal form (1NF) relations [5,6]. This approach has all of the advantages of traditional relational databases. However, there is undue data redundancy [7]. The attribute time-stamping approach with N1NF [8] relations prevents data redundancy and is more expressive. Thus, it avoids the horizontal and vertical data redundancy that occurs in tuple time-stamping [7].

Ben-Zvi proposed the first data model for bitemporal databases, indexing, storage architecture, concurrency, recovery, and a temporal query language and its implementation in [5]. Snodgrass proposed a temporal model that supports valid and transaction times, where tuples are time-stamped with either time instants or time intervals [6]. Bhargava and Gadia attached transaction and valid timestamps to attribute values [9]. They gave a relational algebra for their model and defined new operators to capture and update the data. Their model allows the environment of updates and queries to be restructured, and it can be used as an auditing database system. The bitemporal conceptual data model forms the basis for the temporal SQL (TSQL), proposed by Jensen et al. [10]. TSQL2 is based on a tuple time-stamping data model [11] and 3 time dimensions are supported: user-defined time, valid time, and transaction time; valid time and transaction time are recorded in implicit attributes. The semantics of arithmetic operations that involve time spans and time instants are not explicitly supported in TSQL2. They are left to the calendar as calendar-specific operations. Because TSQL2 treats all instants as indeterminate at finer granularities, time durations that have mixed granularities cannot be represented. SpyTime is another bitemporal database based on tuple time-stamping that reports the movement of spies in cities, and it also has a set of benchmark temporal queries on this database [12]. The T4SQL was proposed, based on the tuple time-stamping approach, as a new query language in [13], which operates on multidimensional temporal relations. It allows one to query temporal relations provided with (a subset of) the temporal dimensions of valid, transaction, availability, and event time, according to di?erent semantics. Although any T4SQL query can be translated into an equivalent SQL query, the corresponding SQL queries are more complex, their size is bigger, and their execution is often quite ine?cient.

There are other implementations of temporal databases on top of relational or object relational databases, some of which can be found in [14]. Most of these implementations use tuple time-stamping, but that in [15] is a model that builds valid time support directly into an extensible commercial object-relational database system. There is another model that uses attribute time-stamping and temporal elements and supports 4 different types of users, which is similar to our concept of a context, except that it only supports valid time [16].

XML is also a new database model serving as a powerful tool for approaching semistructured data. The hierarchical structure of XML provides a natural environment for the use of temporally grouped [17] or attribute time-stamping approaches. The authors in [18] showed that transaction-time, valid-time, and bitemporal database histories can be represented in XML and queried using XQuery without requiring any extensions of the current standards. The study in [19] presented the ArchIS system, which uses XML to support the attribute time-stamping approach, XQuery to express powerful temporal queries, temporal clustering, indexing techniques for managing the actual historical data in a RDBMS, and SQL/XML for executing the queries on the XML views as equivalent queries on the relational database. The study in [20] has a comparison of various temporal XML data models that occur in the literature.

3. The Nested Bitemporal Relational Model

3.1. Preliminaries

Atom is the basic undefined term that takes its values from the universe U. Let T be a subset of U and represent the set of time points 0, 1, ..., *now*, where 0 is the relative origin of time. The *now* denotes the present time instant, and its value increases as time advances. A *time unit* is user-defined and can be any combination of seconds, minutes, hours, days, etc. A *time interval* is a set of consecutive time points. The closed interval $[l,\ u]$ represents all of the values between l and u, inclusively, whereas the half-open interval $[l,\ u)$ does not include u. A *temporal set* is a set of time points that can be grouped into disjoint time intervals. Although set operations such as intersection, union, and difference can be defined on intervals, intervals are not closed under set operations. A temporal set that is represented by the maximal intervals having consecutive time points is defined as a *temporal element* [7]. Examples of a time point, time interval, and temporal element are shown in Figure 1.

A *BTA* is defined as a triplet, <transaction time, valid time, value>, where the transaction and valid time components can be applied as a time point, a time interval, or a temporal element. A BTA in the form of $<[\mathrm{TT}_l,\ \mathrm{TT}_u),\ [\mathrm{VT}_l,\ \mathrm{VT}_u),\ \mathrm{V}>$ represents:

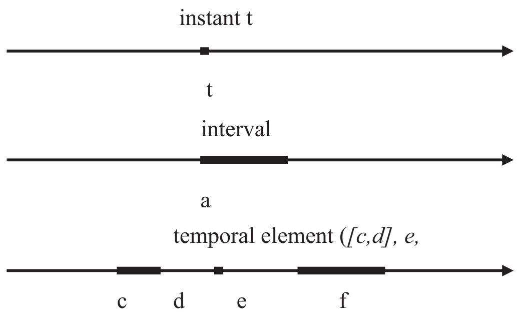


Figure 1. Time point, time interval, and temporal element on the time axis.

TT_l: Transaction time lower bound,

TT_u: Transaction time upper bound,

VT_l : Valid time lower bound,

VT_u: Valid time upper bound,

V: Data value.

Because the value of *now* changes as time progresses, the [TT_l, *now*] or [VT_l, *now*] interval is closed and expanding.

Example 1. The BTA {<[28, 39), [31, 42), 24K>} states that value 24K is written to the database at transaction time 28, effective starting from valid time 31. At 11 time points later, when the transaction time is 39, the database updates that at valid time 42, value 24K is no longer valid.

3.2. Nested bitemporal relation schemes

A tuple scheme and a nested bitemporal relation scheme are defined inductively. While a tuple scheme is a finite sequence of schemes, a nested bitemporal relation scheme is a tuple scheme with previously delineated components. The nesting depth of a scheme is called its *order.* An atom's and a BTA's order are equal to zero. The order of a nested bitemporal relation scheme is one more than the order of its tuple scheme. The inductive definition of bitemporal tuple and nested bitemporal relation schemes are given in [1].

The defined NBRM is based on attribute time-stamped (temporally grouped) [17] nested bitemporal relations. The EMPLOYEE table in Table 1 is an example of a nested table, where time intervals are attached to attributes. The nesting level, the order of a bitemporal relation, of relation EMPLOYEE is 3. The atomic attributes EMP# and BIRTH-DATE are at nesting level 1; the bitemporal attributes NAME, ADDRESS, DEPARTMENT, and SALARY are at nesting level 2; and the bitemporal attributes DNAME and MANAGER are at nesting level 3 of relation EMPLOYEE. For most applications, a few levels of nesting would be sufficient to model temporal data.

Example 2. The definition of the nested bitemporal relation scheme EMPLOYEE depicted in Table 1 is shown below.

EMPLOYEE: = relation <e>

e: = tuple: <EMP#, ENAME-B, ADDRESS-B, BIRTH-DATE, DEPARTMENT, SALARY-B>

ENAME-B: = relation: <NAME>

ADDRESS-B: = relation: <ADDRESS>

DEPARTMENT: = relation: <DNAME-B, MANAGER-B>

Table 1. A nested bitemporal relation, EMPLOYEE (note that DEPARTMENT and SALARY-B are attributes of EMPLOYEE and are displaced on the next line to save space).

EMP#	ENAME-B	ADDRESS-B				
	NAME	ADDRESS	BIRTH DATE	DNAME-B	MANAGER-B	SALARY
				DNAME	MANAGER	
E_1	<[1, now], [1, now], Bob Brown>	<[1, now], [1, now], a_1>	1975	{<[1, 8), [1, 10), Sales>, <[9, now], [11, now], Planning>}	<[1, now], [1, now], Bob Brown >	{<[1, 13), [1, 15), 25K>, <[14, 32), [16, 34), 32K> <[33, now], [35, now), 40K>}
E_2	{<[12, 27), [15, 27), Carol Ken >, <[28, 40), [28, 40), Carol Brown >, <[41, 45), [41, 45), Carol Ken >, <[53, now], [55, now], Carol Ken >}	{< [12, 27), [15, 27), a $_2$>, <[28, 40), [28, 40), a $_1$>, <[41, 45), [41, 45), a $_2$>, <[53,now], [55, now], a_2>}	1990	{<[14, 45), [15, 45), TechSup>, <[53, now], [55, now], TechSup >}	{<[14, 45), [15, 45), Bob Brown>, <[53, now], [55, now], Amy Angel >}	{<[14, 21), [15, 25), 20K>, <[22, 45), [26, 45), 22K>, <[53, now], [55, now], 25K>}
E_3	<[15, now], [15, now], Liz White >	{<[15, 56), [18, 56), a $_3$>, <[57, now], [57, now], a_5>}	1982	<[15, now], [18, now], Sales>	{<[15, 25), [15, now], Bob Brown >, <[26, now], [15, now], Amy Angel>}	{<[15, 27), [18, 30), 22K>, <[28, 39), [31, 42), 24K>, <[40, now], [43, now], 26K>}
E_4	<[10, now], [10, now], Amy Angel>	<[10, now], [10, now], a_4>	1985	<[10, now], [10, now], Sales>	<[10, now], [10, now], Bob Brown>	{<[10, 30), [10, 34), 25K>, <[31, 44), [35, 48), 28K>, <[45, now], [49, now], 30K>}

DNAME-B: = relation: <DNAME>
MANAGER-B: = relation: <MANAGER>
SALARY-B: = relation: <SALARY>
EMP#, BIRTH-DATE: = tuple <atom>
NAME, ADDRESS, DNAME, MANAGER, SALARY: = tuple <BTA>

Notice that E_2 has nonoverlapping time intervals from valid time 45 to 55 and transaction time 45 to 53, when she left the company and rejoined. E_2's coming back to the company is recorded at transaction time 53 and joined at valid time 55.

3.3. Nested bitemporal relation algebra and calculus

There are 3 commonly used contexts to query bitemporal databases: bitemporal context, current context, and historical context. Bitemporal context refers to the entire bitemporal history, which is useful for auditing queries. In the current context, we refer to only currently valid tuples of a bitemporal relation. While bitemporal context is to investigate the history of corrected errors, current context is for querying the snapshot state of a bitemporal database. A bitemporal relation is restricted to its state at a given time point or time interval in the rollback context. The nested bitemporal relational algebra operations for the bitemporal, historical, and current context are defined in [1]. The nested bitemporal relational calculus (well-formed formulas for bitemporal, historical, and current context) for the NBRM is given in [21].

4. Implementation of the NBRM

This section outlines how the NBRM presented in the previous section can be implemented. The architecture of the NBRM is shown in Figure 2. This model provides database users with different types of support related to context requirements. One of the advantages of the attribute time-stamping approach is that all of the temporal attributes can be included in one relation [22]. This relation may have nontemporal attributes along with the temporal attributes. Any unique nontemporal attribute is chosen as a primary key for this relation. For the experiments, we use a hypothetical company database with over 10 years of past and possible future data.

4.1. BTA type

We defined a BTA in Section 3 in the form of $<[TT_l, TT_u), [VT_l, VT_u), V>$. The BTA contains the built-in data type DATE for the lower and upper bounds of the transaction and valid times. Time intervals might be in any granularity, i.e. DATE and TIME-STAMP, depending on the application. The value part may be CHARACTER, CHARACTER VARYING, CHARACTER LARGE OBJECT, NUMERIC, DECIMAL, INTEGER, SMALLINT, BIGINT, or BOOLEAN.

There are 2 possibilities to represent BTA types: the first approach implements it as a string and the second defines it as an abstract data type. By representing a BTA as a string, a logically coherent unit is not decomposed over several attributes. These representations hide the complexity of the abstract structures from end users and application programmers. Figure 3 shows the 5 components stored as a string, BTA_String. Figure 4 depicts the BTA as an abstract data type, BTA_ADT.

The type system facilities of object-relational databases allow us to define a BTA as a string (BTA_String) and as a structured abstract data type (BTA_ADT). Removing or retrieving a component, such as the transaction time lower and/or upper bound as a substring, is allowed and used in the query expressions. Once the BTA is defined, it can be used in SQL statements where other built-in types are used.

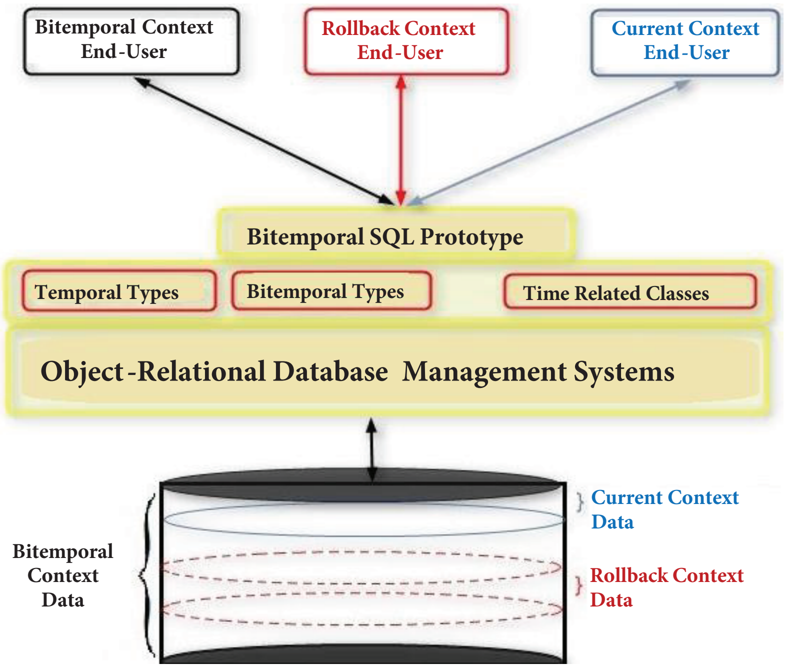


Figure 2. Architecture diagram of the proposed bitemporal object relational database system.

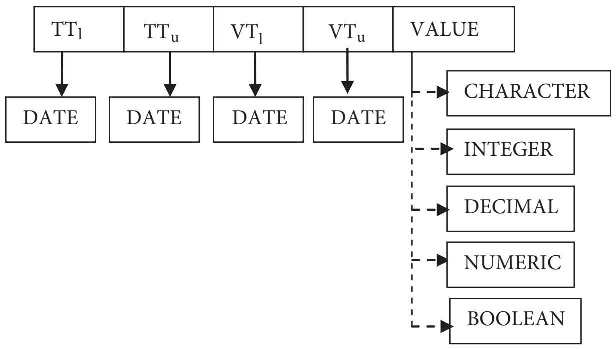


Figure 3. Representation of the BTA as a string, BTA_String.

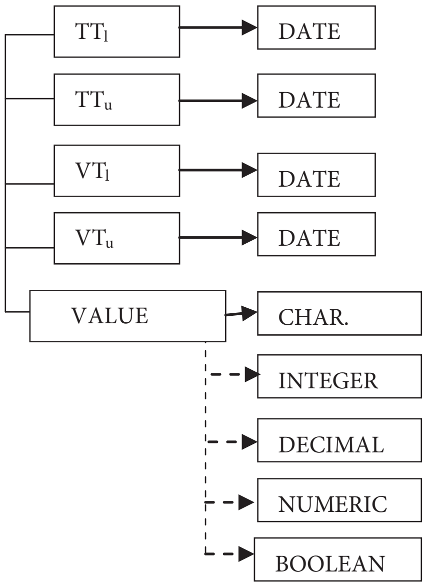


Figure 4. Representation of the BTA as an abstract data type, BTA_ADT.

Abstract data types can be declared to be the 'data type' of an entire table so that the table's attributes are defined by the abstract data type. By in-lining the repeated objects in the table, the reliance on creating another table with its own structure and indices is removed in collection type tables. Data manipulation operations such as select, insert, and delete can be applied similarly to ordinary tables.

4.2. Nested bitemporal relation

A tuple in a nested bitemporal relation is an instance of the structured type on which the table is defined. It gives the instance a unique identity. Having a set of identical abstract data types in a single tuple actually simulates the attribute time-stamping approach with a single-attribute table for each object's time related attributes. These are temporally grouped relations. Figure 5 gives the definition of the Employee table introduced in Section 3 (in Table 1) with a NESTED TABLE collection type. Note that the DEPARTMENT attribute (named as DEPT_MNG) consists of 2 bitemporal tables: MNG _HISTORY and DEPT_HISTORY.

```
CREATE TABLE EMPLOYEE (
  EMP#     NUMBER Primary Key,
  NAME     BTA_NAME,
  ADDRESS  BTA_ADDRESS,
  BIRTH_DATE DATE,
  DEPT_MNG BTA_DEPT_MNG,
  SALARY   BTA_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 MNG_TABLE,
     NESTED TABLE DEPARTMENT_HISTORY STORE AS DEPT_TABLE),
NESTED TABLE SALARY STORE AS SALARY_TABLE;
```

Figure 5. Definition of a nested bitemporal relational table, EMPLOYEE.

4.3. Implementation of the nested bitemporal relational algebraic operations

Select, project, Cartesian product, and set theoretic operations are handled by the query processor of the object relational database system. The bitemporal_atom_decomposition and bitemporal_atom_formation operations are also managed by the query processor. We briefly comment on the implementation methodology explained by Atay and Tansel in [21] for the slice and AS_OF (rollback) operation that are included in BtSQL.

4.3.1. Slice operation

The slice operation works on the 2 bitemporal attributes, and it returns the first attributes' value part, along with the common time intervals that they have, followed by the second attributes' value part. The first finds if the given 2 intervals intersect or not, by comparing the lower and upper bounds. If that is the case, then it finds the starting and ending points of their common interval. Finally, it returns the corresponding value part of the 2 bitemporal attributes' value parts, along with the common new intervals. We implement the slice operation as a function and embed it in the database system. It handles both string and ADT representations of BTAs. Table 2 depicts the result of the (E_2, SLICE $_{(\cap, SALARY, MANAGER)}$) operation on the EMPLOYEE table given in Table 1.

Table 2. Result of (E_2, SLICE $_{(\cap, SALARY, DEPARTMENT)}$).

EMP#	MANAGER	VTlb	VTub	SALARY
E_2	Bob Brown	15	25	20K
E_2	Bob Brown	26	45	22K
E_2	Amy Angel	55	*now*	25K

4.3.2. AS_OF operation

NBRM answers queries about past states by rolling the database back to a state sometime in the past, through the AS_OF clause that is added to the SQL syntax, which rolls back a relation to some earlier time. We implement the 'AS OF' clause as a function that is embedded in the database system. It receives an attribute name along with the transaction time interval (or point).

Table 3. Result of (EMP#, AS_OF $_{(30,\,SALARY)}$) of the EMPLOYEE table.

EMP#	SALARY(TTlb, TTub)	SALARY(VTlb, VTub)	SALARY
E_1	[1, 13)	[1, 15)	25K
E_1	[14, 30)	[16, 30)	32K
E_2	[14, 21)	[15, 25)	20K
E_2	[22, 30)	[26, 30)	22K
E_3	[15, 27)	[18, 30)	22K
E_3	[28, 30)	[30, 30)	24K
E_4	[10, 30)	[10, 30)	25K

It first finds, for every tuple k, the set of BTAs in that attribute. If the given interval (or point) intersects the transaction time of the BTA, that BTA and the BTAs with earlier transaction times are returned. Table 3 displays the result of (EMP#, AS_OF $_{(30,\,SALARY)}$) operation on the EMPLOYEE tables' SALARY attribute, which is rolled back to a state where the transaction time is equal to 30.

5. BtSQL

In order to demonstrate the feasibility of the NBRM proposed in [1], we design a graphical user interface for the application programmers and end-users. The preprocessor converts bitemporal queries into statements in standard SQL and passes them to the DBMS. BtSQL supports the SELECT, INSERT, DELETE, and UPDATE statements of SQL, extended for bitemporal relational databases. We provide an example where the bitemporal join is restricted by the time slice operation in Section 5.4. The NBRM allows the formulation of useful queries by joining 2 bitemporal nested tables. More example queries on bitemporal joins can be found in [21].

end_value and **sysdate**: end_value is a special constant for representing the infinite upper limit and/or 'now' (we use '09.09.9999'). This is common practice in other implementations, as well. sysdate is a SQL function that returns the current time (now). In BtSQL specifications, both the valid time and transaction time upper bounds default to 'now' if a specific time is not specified.

5.1. Insert in BtSQL

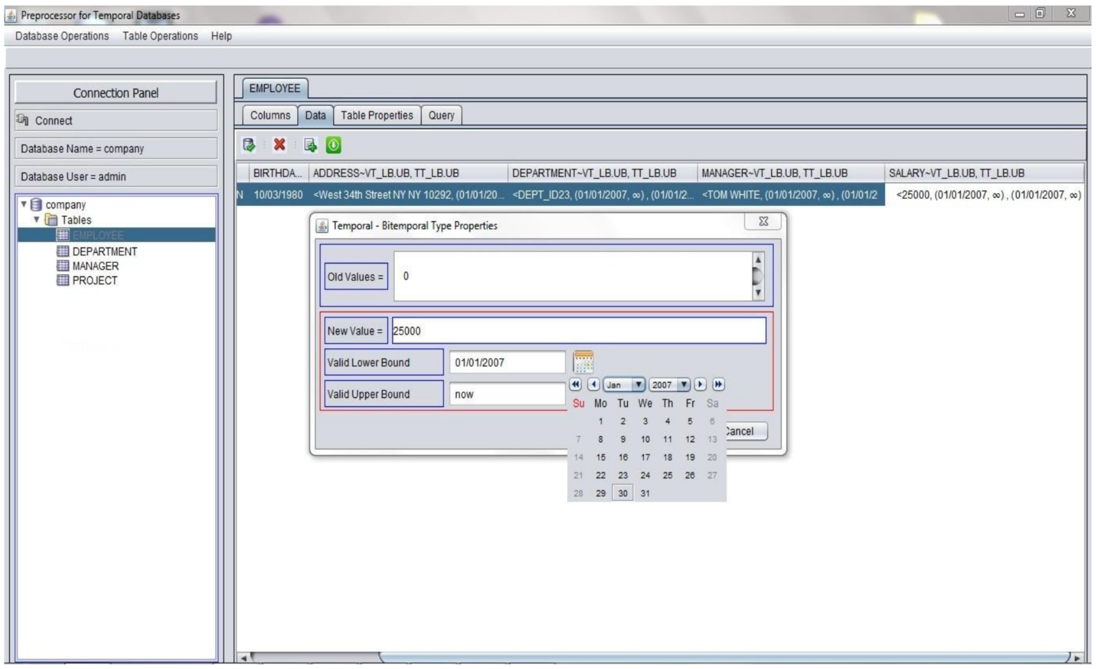
The insert specification in BtSQL has the following semantics in the SQL:

Insert_BtSQL(*Relation_Name, Values, VT*) $\rightarrow$ **Insert_SQL**(*Bt_Values*)

where Insert_BtSQL is the insert specification in BtSQL and Insert_SQL is the insert statement in SQL. *Values* are pairs <att_name, value> and *VT* is the valid time lower bound. For each <att_name, value> pair, the corresponding *Bt_Values* is in the form of a BTA along with the attribute name: <att_name, [sysdate, end_value], [valid_time, end_value], att_value>.

Figure 6 illustrates the process of inserting 'MIKE BROWN' with EMP# 20001, birth date '10/3/1980', address 'West 34th Street NY NY 10292', into the DEP_ID23 department into the EMPLOYEE table. 'TOM WHITE' is assigned as his manager, and his salary is 25,000 starting on 1 January 2007.

Figure 7 is the actual SQL code that inserts the tuple for BTA_ADT. Note that this is an example of inserting a new employee in a hypothetical company. Since the user provides the valid time, the truth of values starts from 1/1/07 for the system.



	BIRTHDA...	ADDRESS~VT_LB.UB, TT_LB.UB	DEPARTMENT~VT_LB.UB, TT_LB.UB	MANAGER~VT_LB.UB, TT_LB.UB	SALARY~VT_LB.UB, TT_LB.UB
N	10/03/1980	<West 34th Street NY NY 10292, (01/01/20...	<DEPT_ID23, (01/01/2007, ∞) , (01/01/2...	<TOM WHITE, (01/01/2007, ∞) , (01/01/2	<25000, (01/01/2007, ∞) , (01/01/2007, ∞)

Old Values = 0

New Value = 25000

Valid Lower Bound 01/01/2007

Valid Upper Bound now

Figure 6. Inserting a tuple with BtSQL.

```
INSERT INTO EMPLOYEE VALUES
(20001,
NAME(BTA_ADT(sysdate, end_value, '01.01.2007',end_value,
'MIKE BROWN')),
ADDRESS(BTA_ADT(sysdate, end_value,'01.01.2007', end_value,
'West 34th Street NY NY 10292')),
'10.03.1980',
DEPT_MNG(TYPE_DEPT_MNG(
DEPARTMENT(BTA_ADT(sysdate, end_value, '01.01.2007',end_value,
'DEPID_23')),
MANAGER(BTA_ADT(sysdate, end_value, '01.01.2007',end_value,
'TOM WHITE')))),
SALARY(BTA_ADT(sysdate, end_value, '01.01.2007', end_value,
25000)));
```

Figure 7. Inserting a tuple with BTA_ADT type in SQL.

5.2. Update in BtSQL

An update operation 'inserts' a new BTA while preserving the old version. The update specification in BtSQL has the following semantics in SQL:

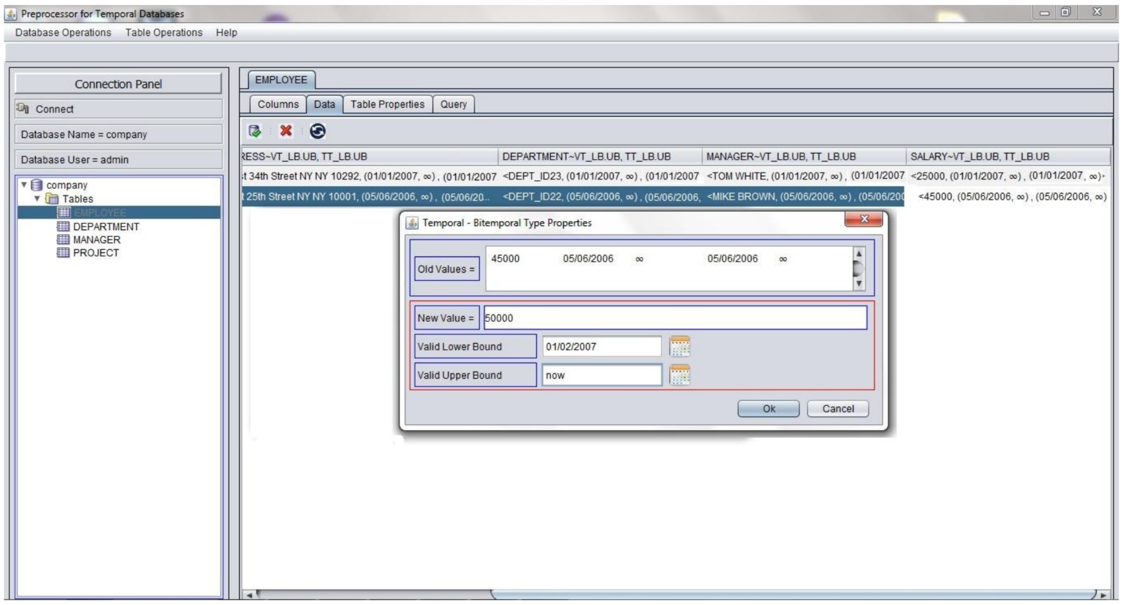
Update_BtSQL(*Relation_Name, condition, att_value, VT*) $\rightarrow$

Update_SQL(*SQL_closeBTA, SQL_insertBTA*)

where Update_BtSQL is the update specification in BtSQL and Update_SQL has 2 SQL statements. *SQL_closeBTA* is an update statement that sets the transaction time upper bound to sysdate and the valid time upper bound to valid_time to VT for the tuple identified by the condition in att_value. *SQL_insertBTA* is an insert statement that inserts a new BTA <[sysdate, end_value], [valid_time, end_value], att_value> into the attributes specified in the condition statement.

BtSQL asks for table name(s) to be updated, a condition, and the new values, as well as the valid time. The system finds the last bitemporal variable where the valid time upper bound is 'end_value' and replaces the valid time upper bound with the user-provided new VT valid time for tuples, which satisfies the condition. The valid time upper bound of the existing tuple cannot be greater than the 'end_value' ('09.09.9999') since it is user-provided. It next inserts the new BTA type into the database for satisfying the tuple(s). Its valid time lower bound gets the valid time when the change was/is/will be effective. Its transaction time lower bound gets the sysdate, and both intervals' upper bounds get the end_value. The BTA's value part gets the user-provided new value. Since the transaction and valid time upper bounds are set to end_value, this last inserted BTA is valid until a new update or delete query is performed.

For example, Figure 8 shows how the employee's salary data is updated with EMP#=12345 to 50,000, effective 1 February 2007. Figure 9 shows the actual SQL code needed to make this update possible for BTA_ADT. If a data error is discovered, a compensating update operation has to be performed to correct the error. The erroneous data are kept; the correct value part and valid time are updated using the correct date.



Database Name = company

Database User = admin

RESS~VT_LB.UB, TT_LB.UB	DEPARTMENT~VT_LB.UB, TT_LB.UB	MANAGER~VT_LB.UB, TT_LB.UB	SALARY~VT_LB.UB, TT_LB.UB
t 34th Street NY NY 10292, (01/01/2007, ∞) , (01/01/2007	<DEPT_ID23, (01/01/2007, ∞) , (01/01/2007	<TOM WHITE, (01/01/2007, ∞) , (01/01/2007	<25000, (01/01/2007, ∞) , (01/01/2007, ∞)>
t 25th Street NY NY 10001, (05/06/2006, ∞) , (05/06/20...	<DEPT_ID22, (05/06/2006, ∞) , (05/06/2006,	<MIKE BROWN, (05/06/2006, ∞) , (05/06/200	<45000, (05/06/2006, ∞) , (05/06/2006, ∞)

Old Values = 45000 05/06/2006 ∞ 05/06/2006 ∞

New Value = 50000

Valid Lower Bound 01/02/2007

Valid Upper Bound now

Figure 8. Update for the salary attribute with the preprocessor.

```
UPDATE TABLE (     SELECT E.SALARY
                   FROM EMPLOYEE E
                   WHERE E.EMP#= 12345) SAL
SET SAL.VALID_TIME_UPPER_BOUND = '02.01.2007',
      SAL.TRAN_TIME_UPPER_BOUND = sysdate
WHERE SAL.VALID_TIME_UPPER_BOUND = end_value;

INSERT INTO THE ( SELECT E.SALARY
                         FROM EMPLOYEE E
                         WHERE E.SSN = 12345)
VALUES(BTA_ADT(sysdate, end_value,
                   '02.01.2007', end_value, 50000));
```

Figure 9. SQL update code with the BTA_ADT type for the salary attribute.

5.3. Delete in BtSQL

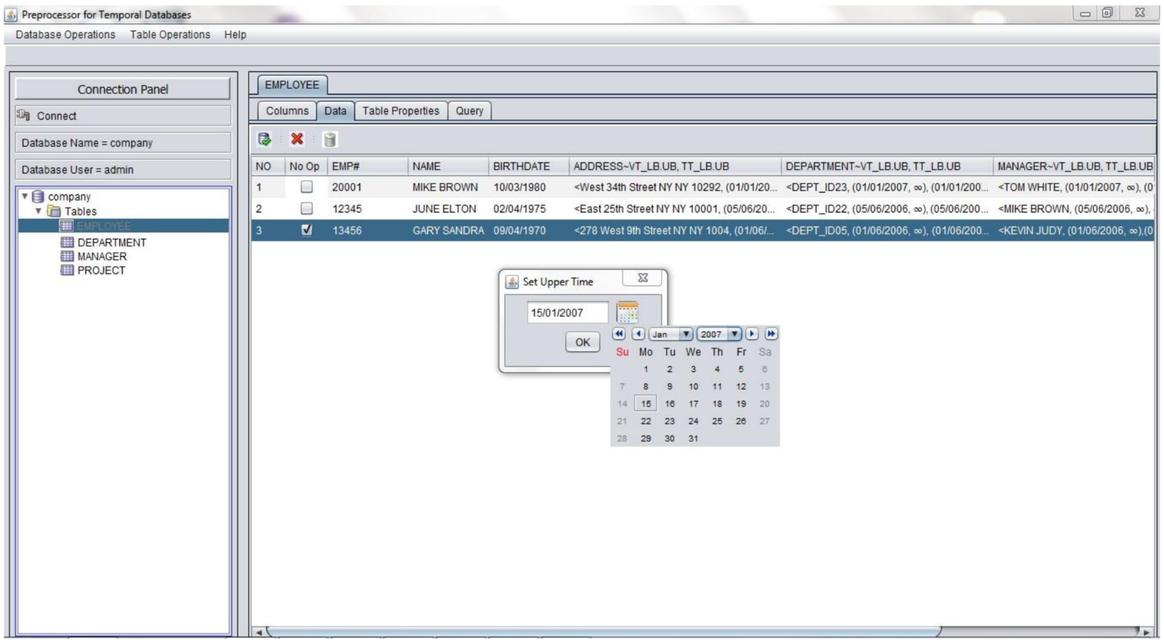
The delete specification in BtSQL has the following semantics:

Delete_BtSQL (*Relation_Name, condition, VT*) $\rightarrow$

Delete_SQL($SQL\_closeBTA$)

where Delete_BtSQL is the specification in BtSQL and Delete_SQL is the corresponding update_statement in SQL, which includes a sequence of update statements that sets the transaction time upper bound to sysdate and valid time upper bound to VT for each time-dependent attribute. Condition is a simple SQL condition that appears in the WHERE clause.

Tuples are never physically deleted from temporal/bitemporal databases for several reasons. Since an implementation example is on company databases in this paper, if an employee leaves a company, his/her information is typically never deleted from the database. The bitemporal attributes' valid time upper bound is replaced with the provided valid time, and the sysdate is recorded as the transaction time's upper bound. The function receives the primary key, EMP# of the employee, and the valid time when the employee leaves the company. For every bitemporal attribute, the function finds the last BTA where the valid time upper bound is end_value, and then replaces it with the valid time when the employee was/is/will no longer be employed. The BtSQL's DELETE page indicates that EMP# 13456 will not be working beginning on 15 January 2007, as shown in Figure 10. Figure 11 depicts the actual SQL code as to how the delete is done for BTA_ADT.



NO	No Op	EMP#	NAME	BIRTHDATE	ADDRESS~VT_LB.UB, TT_LB.UB	DEPARTMENT~VT_LB.UB, TT_LB.UB	MANAGER~VT_LB.UB, TT_LB.UB
1	☐	20001	MIKE BROWN	10/03/1980	<West 34th Street NY NY 10292, (01/01/20...	<DEPT_ID23, (01/01/2007, ∞), (01/01/200...	<TOM WHITE, (01/01/2007, ∞), (0
2	☐	12345	JUNE ELTON	02/04/1975	<East 25th Street NY NY 10001, (05/06/20...	<DEPT_ID22, (05/06/2006, ∞), (05/06/200...	<MIKE BROWN, (05/06/2006, ∞),
3	✓	13456	GARY SANDRA	09/04/1970	<278 West 9th Street NY NY 1004, (01/06/...	<DEPT_ID05, (01/06/2006, ∞), (01/06/200...	<KEVIN JUDY, (01/06/2006, ∞),(0

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Figure 10. Delete a tuple with the preprocessor.

5.4. Retrieval in BtSQL

Retrieval in BtSQL has the following semantics in SQL:

Select_BtSQL (Result, Source, Condition, AS_OF, [Valid Time], [Transaction Time]) $\rightarrow$ **Select_SQL** (SELECT Result [Valid Time], [Transaction Time]

FROM Source
WHERE Condition AND [AS_OF] AND [Time_Slice])

```
UPDATE TABLE (SELECT E.NAME
              FROM EMPLOYEE
              WHERE E.EMP#= 13456) NAM
SET NAM.VALID_TIME_UPPER_BOUND = '01.15.2007',
    NAM.TRAN_TIME_UPPER_BOUND = sysdate
WHERE NAM.VALID_TIME_UPPER_BOUND = end_value;

UPDATE TABLE (SELECT E.ADDRESS
              FROM EMPLOYEE
              WHERE E.EMP#= 13456) ADR
SET ADR.VALID_TIME_UPPER_BOUND = '01.15.2007',
    ADR.TRAN_TIME_UPPER_BOUND = sysdate
WHERE ADR.VALID_TIME_UPPER_BOUND = end_value;

UPDATE TABLE (SELECT DEPARTMENT_HISTORY
              FROM TABLE (SELECT E.DEPT_MNG
                          FROM EMPLOYEE
                          WHERE E.EMP#= 13456)) DEP
SET DEP.VALID_TIME_UPPER_BOUND = '01.15.2007',
    DEP.TRAN_TIME_UPPER_BOUND = sysdate
WHERE DEP.VALID_TIME_UPPER_BOUND = end_value;

UPDATE TABLE (SELECT MANAGER_HISTORY
              FROM TABLE (SELECT E.DEPT_MNG
                          FROM EMPLOYEE
                          WHERE E.EMP#= 13456)) MAN
SET MAN.VALID_TIME_UPPER_BOUND = '01.15.2007',
    MAN.TRAN_TIME_UPPER_BOUND = sysdate
WHERE MAN.VALID_TIME_UPPER_BOUND = end_value;

UPDATE TABLE (SELECT E.SALARY
              FROM EMPLOYEE
              WHERE E.EMP#= 13456) SAL
SET SAL.VALID_TIME_UPPER_BOUND = '01.15.2007',
    SAL.TRAN_TIME_UPPER_BOUND = sysdate
WHERE SAL.VALID_TIME_UPPER_BOUND = end_value;
```

Figure 11. Delete a tuple with the BTA_ADT type in SQL.

where Select_BtSQL is the retrieval specification in BtSQL and Select_SQL is the corresponding select statement in SQL. A specification that is enclosed within square brackets is optional.

Result: list of attributes

Source: list of relations

Condition: list of SQL conditions connected by AND, OR, NOT, etc.

AS_OF: rollback (transaction) time for rolling back the relation to the specified time. This is embedded in the WHERE clause as a function call.

Valid time: valid time in the result.

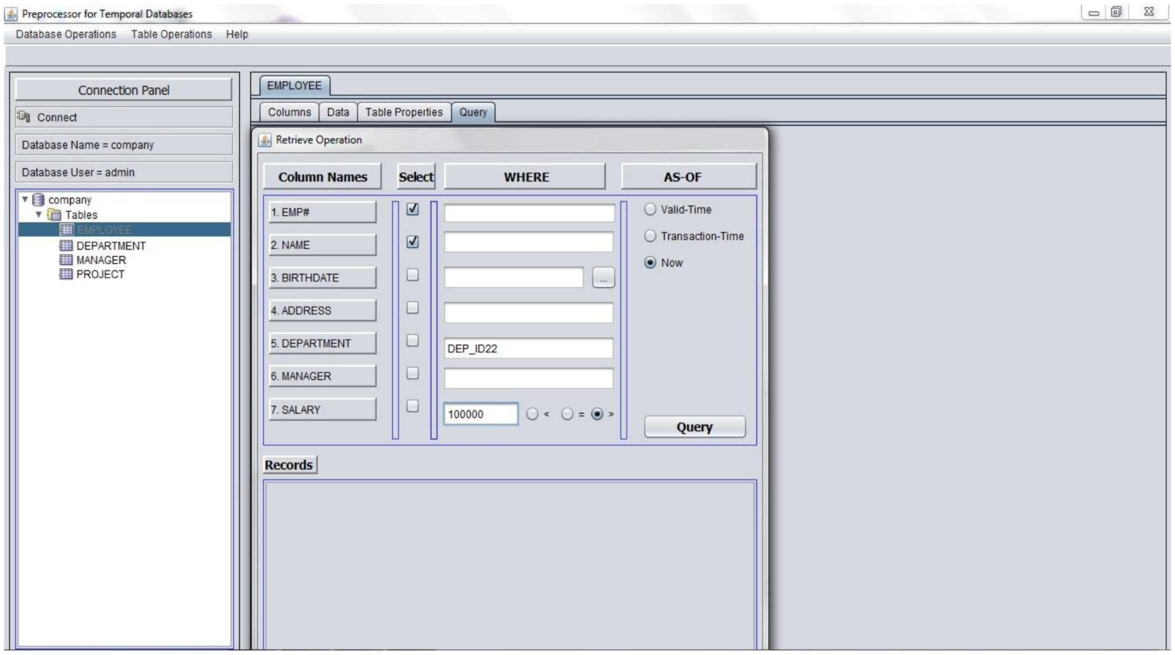
Transaction time: transaction time in the result.

Time slice: time slice operation on the specified attributes. This is embedded in the WHERE clause as a function call.

BtSQL accepts queries in a bitemporal context, in a current context, or in a historical context. Queries of course involve the relation name listed in the FROM clause. The query selects tuples that satisfy the condition(s) of the WHERE clause, and then projects the result to the attributes listed in the SELECT clause. All of the options and flavors of the SELECT statement in SQL can also be used in BtSQL.

If the query is in a historical context, then the transaction time point or interval is specified in the AS_OF clause, in which all other restrictions, as well as the capabilities for the bitemporal context, apply as well. If 2 bitemporal attributes' common time intervals, or 'when', need to be queried, then the Slice operation (its operation or clause) should be chosen. Slice is used in queries as any other clauses independent of the bitemporal context, current context, or historical context.

As an example, Figure 12 displays a query that lists employee numbers and names that currently work in Department 22 and earn more than 100K. This current context query selects the employee numbers and names that satisfy the conditions DEPARTMENT = 'DEP_ID22' and SALARY > 100000, and whose BTA's valid times are equal to end_value, which is used for *now*. It then passes the result to the EMP# and NAME attributes listed in the SELECT clause. Figure 13 shows how this query is written with BTA_ADT.



Database Name = company
Database User = admin

company
Tables
EMPLOYEE
DEPARTMENT
MANAGER
PROJECT

Column Names	Select	WHERE
1. EMP#	✓	
2. NAME	✓	
3. BIRTHDATE		
4. ADDRESS		
5. DEPARTMENT		DEP_ID22
6. MANAGER		
7. SALARY		100000 < = [✓ >]

AS-OF: Valid-Time Transaction-Time [✓ Now]

Figure 12. Current context query with BtSQL.

```
SELECT E.EMP#,NAM.VALUE AS NAME,
FROM EMPLOYEE E,
      TABLE(E.NAME) NAM,
      TABLE(E.DEPT_MNG) DEP_MAN,
            TABLE(DEP_MAN.DEPARTMENT_HISTORY) DEP,
      TABLE(E.SALARY) SAL
WHERE DEP.VALUE = 'DEP_ID22' AND SAL.VALUE >100000
AND NAM.VALID_TIME_UPPER_BOUND = end_value
AND SAL.VALID_TIME_UPPER_BOUND = end_value
AND DEP.VALID_TIME_UPPER_BOUND = end_value
```

Figure 13. Current context query with the BTA_ADT type.

6. Performance evaluation

We conduct experiments to measure the performances of the 2 implementation methods. The experiments are intended to compare the performances of the bitemporal tables stored in the nested table collection type with 2 different implementations: BTAs defined as a string (BTA_String) and as an abstract data type (BTA_ADT). The performance of a bitemporal relational model is measured by examining the processing time for queries and updates on an already populated database.

Each database contains a bitemporal table that has 6 explicit attributes: 2 nontemporal attributes are EMP#, the primary key of the table, of type INTEGER, and Birthday, of type DATE. The other bitemporal attributes, NAME, ADDRESS, and SALARY bitemporal attributes, use a nested table collection type. DEPARTMENT is a nested table with 2 columns, DNAME and MANAGER. DNAME records the department with which the employee is affiliated, and MANAGER records the employee's manager; each stores BTAs in the nested table collection type.

In comparing the relative performances, the following question is considered: which implementation method, BTA_String or BTA_ADT, performs faster in terms of database modifications and queries? The answer to this question is important, because it should significantly affect the bitemporal DBMS design and implementation decisions.

6.1. System configuration

For the experiment, an object RDBMS, Oracle9i, is used. It is run on a Pentium IV 3.0-GHz PC with 1 GB of memory and 1500–3000 MB of system-controlled swap space. During the study, the system was used exclusively for our experiments. The server and client processes ran on the same machine.

6.2. Data generation

In this step, a set of bitemporal data objects are generated. Since bitemporal data in real-world applications could not be obtained, objects containing bitemporal data are generated synthetically, objects whose bitemporal attributes are random variables drawn from normal distributions between 01.01.1995 and 01.01.2007. The granularity of the DATE values is 'MM.DD.YYYY'. All of the methods presented here can be utilized for any granularity in the application.

Unique employee numbers between 10,001 and 20,000 are used, and they increase by 1 sequentially. Each employee is assigned a birth date in the MM.DD.YYYY format. A total of 10,000 distinct names and addresses are generated for the testing. There are 30 departments and 30 managers from DEP_ID1 through DEP_ID30 and from MANAGER_ID1 through MANAGER_ID30, respectively. Each employee is assigned to 1 department and 1 manager at a time. Employees change their departments and managers 5 times on average, and receive an additional 5% salary increase when their department changes. Every employee has a 3% salary increase each new year. For these updates, it is assumed that the transaction time bounds are within 1 to 10 days, less or more, from the valid time bounds.

6.3. Update operations and queries

We conduct 3 experiments. In the first experiment, we insert 10,000 tuples into both tables created with BTA_String and BTA_ADT. We run the other 2 experiments starting with 10 years of data. Each table thus contains approximately 10,000 tuples, 50,000 nested bitemporal relations that represent sets of BTAs, and 300,000 BTAs.

In the second experiment, the tests are performed by executing modifications as a series of updates for each bitemporal attribute. The first 3 update operations modify only a single tuple in the table. The second update modifies a group of tuples, depending on a condition such as updating DNAME or changing the department's MANAGER name. The last update operation modifies all of the tuples in a NBRM table.

In the third experiment, the main goal is to show that the NBRM allows the formulation of useful queries. To demonstrate the NBRM's functionality, we illustrate this point with 2 sets of queries. The queries are designed and run, and the required time is measured over both databases.

Insert 10,000 initial data: The insert time is the same for BTA_String and BTA_ADT. Both tables insert 10,000 data within 21 s. This is possibly because of the sequential disk writes for inserting the tuples in both cases.

Update 1: Change the name to 'KAMERON JUANA_ONCE' for the employee whose EMP# is 19955, valid from 07.07.2007.

Update 2: For the employee with EMP# 19955, change his/her department to 'DEP_ID12', valid from 07.07.2007 to now.

Update 3: For the employee with EMP# 19955, change his/her salary to 65,000 during the validity period (07.07.2007, now].

Updates 1 to update 3 involve 1 tuple that requires 1 disk access for both implementation methods. Because the execution times for the BTA_ADT type are shorter than the BTA_String execution times, the BTA_ADT type performs better than the BTA_String type. This result is presented in Figure 14.

Update 4: Change MANAGER_ ID13 to MANAGER_ID05 for all employees, valid from 07.07.2007.

Update_4 updates a set of tuples resulting from a selection condition applied to a table. The BTA_ADT performs slightly better than the BTA_String since updating the string type requires more time than the abstract data type. Figure 15 depicts the results of this update. Unlike updates 1 to 3, update 4 accesses many tuples that require more disk access.

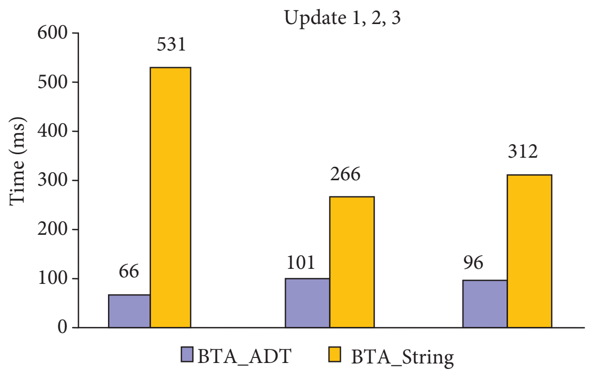


Figure 14. Updating times of a single tuple.

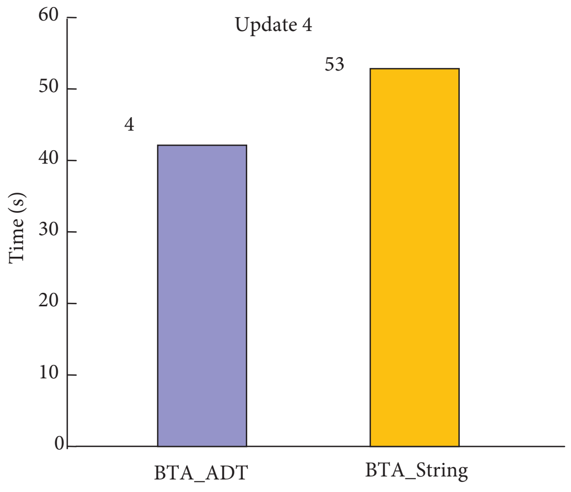


Figure 15. Updating times of a group of tuples.

Update 5: Give a 5% salary increase to all employees, valid from 07.07.2007.

Figure 16 shows the results of updating the time-related attributes for all of the tuples. Clearly, the BTA_String performs poorly compared to the BTA_ADT because updating the string type requires more time than the abstract data type.

BTA_String requires more processing in updating because the SUBSTR string function reads the whole BTA_String to find tuples that are valid. On the other hand, only the transaction and valid time upper bound fields are read in the BTA_ADT type. Therefore, it is expected that the BTA_ADT type would perform better than the BTA_String type in updating the tuple(s).

Query 1: List the salary values in the database that are stored between the times 01.01.2001 and 01.01.2006.

This is a bitemporal context query, and it uses a valid time interval. The selection operation picks tuples where the valid time components are between 01.01.2001 and 01.01.2006. The projection operation retains the EMP# as the first attribute, and the value and other 4 components from the SALARY bitemporal attribute.

Both methods return the selected tuples almost at the same run time for Query 1, as shown in Figure 17. The BTA_ADT and BTA_String time components are extracted and successfully used in the expression in bitemporal context.

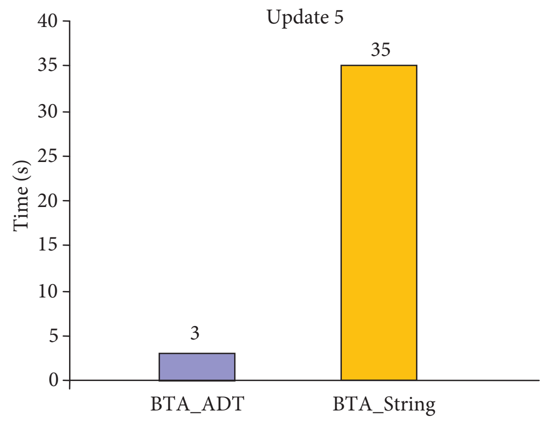


Figure 16. Updating times of all of the tuples.

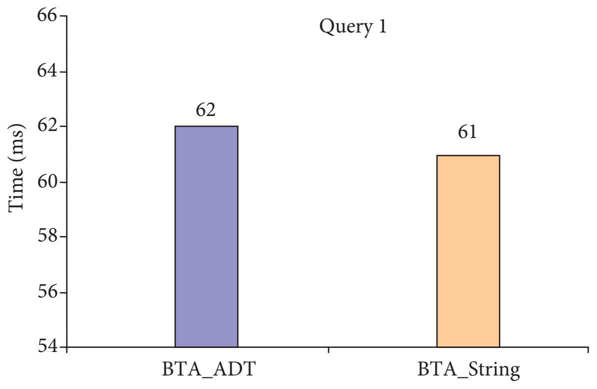


Figure 17. Run time for Query 1.

Query 2: Find the names of employees that have shared the same address. When was it?

This is also a bitemporal context query. This query joins the table with itself, and then uses the time slice operation. The selection operation picks tuples where the ADDRESS attributes' value components are equal. The time slice operation synchronizes the valid time component of the ADDRESS with respect to the ADDRESS_A valid time component, and hence implements 'when'. Finally, the projection operation retains ENAME's and the ADDRESS bitemporal attributes' value components, and the common valid time lower and upper bounds. The main goal of this query is to show that the NBRM allows the formulation of useful queries by joining 2 bitemporal nested tables.

BTA_String outperforms BTA_ADT for Query 2 as shown in Figure 18. The join operation requires more disk reads in the case of BTA_ADT, since the abstract data type implementation involves subtables, which require more tuples. However, BTA_String requires fewer disk accesses since BTAs are stored as a set within a tuple.

Query 3: Get records for all of the departments in which the employee CANAN ATAY has worked in the database as of ['01.01.2004', '12.12.2006'].

This is a historical context query with a time interval. The AS_OF operation rolls back the department attribute to time value interval '01.01.2004', '12.12.2006'. The selection operation picks tuples from the name attribute where the value is 'CANAN ATAY', and then the projection operation displays the department attribute value and valid time components.

Query 4: As of 01.01.2006, who was working in the DEP_ID22 department?

This is a historical context query retrieving the state of a table as of '01.01.2006' in the past. The selection operation picks tuples where value component is equal to 'DEP_ID22'. The projection operation retains the EMP#, name attribute value part, department attribute's name, and valid time components.

Query 3 selects one tuple, namely the department of a particular employee, from a rolled-back attribute. A rolled-back bitemporal attribute is on 2 levels of nesting. BTA_String and BTA_ADT perform almost alike. Query 4 first rolls back the 2 level-nested 'DEPARTMENT' bitemporal attributes. Next, it goes through every tuple and returns the names of employees who are affiliated with the given department ID. In this query type, BTA_ADT and BTA_String perform closely, as shown in Figure 19. It is also interesting to observe that Queries 3 and 4 resemble the performance patterns of the update queries.

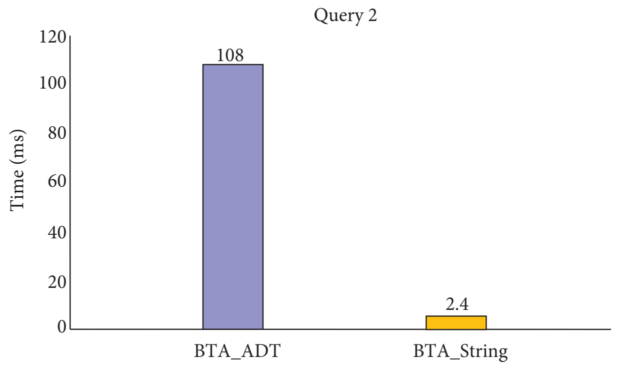


Figure 18. Run time for Query 2.

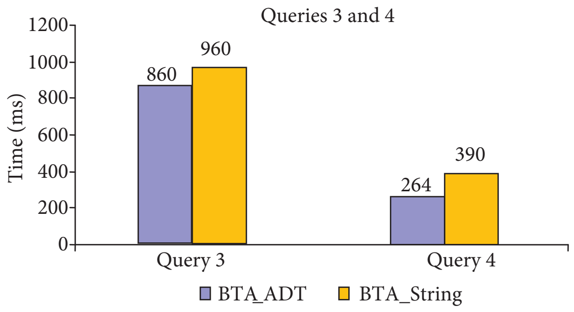


Figure 19. Run times for Queries 3 and 4.

7. Conclusion

This paper demonstrates the feasibility of implementing a bitemporal database on a commercially available object-relational database system. Time stamps are attached to the attributes (temporally grouped) where N1NF (nested) relations are used. We implemented a bitemporal database as a test bed by considering 2 alternative methods and evaluated their performances. The database can be viewed within a bitemporal, historical, or current context. It was shown that the proposed model can be used successfully while implemented with bitemporal, historical, or current context.

We constructed nested bitemporal relational databases with 2 types of BTA, BTA_ADT and BTA_String. We showed that since the BTA's 5 components are stored in BTA_String and in BTA_ADT, it is possible to extract and manipulate any one of them in the expressions. BTAs are stored in a nested table collection type. The performance tests showed that while a BTA_ADT BTA is better with updates, a BTA_String is slightly better with querying. The bitemporal relational algebraic operators, time slice and rollback performed equally well for the 2 representations.

Object-relational database systems have richer semantics and data types, such as abstract data types, than RDBMSs. Moreover, they have the capability to define temporal semantics through these abstract data types. The standard query language SQL3 includes object-relational features that can serve as built-in temporal semantics, and which, therefore, provide a robust platform for implementing temporal databases. Commercial object-relational database systems implement some features of SQL3 and provide readily available platforms to test the concepts developed in [1].

We extend the SQL with bitemporal querying constructs and developed a graphical user interface. A preprocessor translates temporal statements into standard SQL statements. A core set of statements such as insert, update, delete, and select are available in BtSQL. Modification and simple selects on bitemporal relations are supported, including slicing in both the transaction-time and valid-time dimensions. The user interface hides tedious bitemporal query specifications for the user.

The experimental results showed that implementation of the NBRM on an object-relational database is quite attainable. Our bitemporal database system can be used as a test bed to demonstrate the feasibility of bitemporal databases in many application domains, since it supports the essential constructs needed in bitemporal databases. The main conclusion of this work is that a user-friendly graphical query language can be designed and implemented for attribute time-stamped (temporally grouped) bitemporal databases within the framework of an object-relational database. It is our hope that our work will lay the foundation for the widespread implementation of bitemporal relational databases.

TSQL2 is based on homogeneous tuples, but BtSQL also supports heterogeneous tuples. While TSQL2 uses a special operator, coalescing, to collapse all value-equivalent tuples into a single tuple, the result is also coalesced in our model. The relational bitemporal algebra is defined for both languages. Although an equivalent relational bitemporal calculus is provided in the NBRM, a corresponding calculus is not defined in TSQL2. While SpyTime does not have any current queries, we have an example of a 'now' query. Both the SpyTime and NBRM queries have examples of valid/transaction time points and valid time interval-related bitemporal queries. While SpyTime does not support transaction time interval-type queries, the NBRM queries do. The NBRM queries query given time points or time intervals in the past (historical context), but SpyTime does not have such an example. Both the SpyTime and NBRM queries have auditing purpose-type bitemporal context queries. Because BtSQL is powerful enough to support all of the semantics of the queries listed in [12], the NBRM queries satisfy more than the requirements of the SpyTime benchmark queries.

We are working on a comparison of the NBRM and various tuple time-stamped bitemporal models. We will use the same tests on the same data to carry out a performance evaluation of our proposed model against the tuple time-stamped bitemporal models. We specifically plan to use the SpyTime database and its set of benchmark queries to evaluate the performance of the BTA_String and BTA_ADT representations. We will also have an opportunity to compare the performance of tuple time-stamped and attribute time-stamped bitemporal data models, since our model is capable of supporting both approaches. We plan to extend the bitemporal data model by data definition and data management capabilities. Data warehouses store historical data and therefore could clearly benefit from the research on temporal databases. We are working on a project that incorporates the NBRM into a data warehouse. Implementing bitemporal data types as a built-in type into an open source DBMS, such as PostgreSQL, is another possible future work. We also plan to incorporate spatial data into the NBRM, which would effectively create a spatio-bitemporal database. Such spatio-bitemporal databases would have built-in support for both space and time(s) and, consequently, could enable new database applications.

Acknowledgments

We would like to express our deep gratitude for the valuable comments of the anonymous reviewers, which improved the manuscript significantly.

References

[1] A.U. Tansel, C.E. Atay, "Nested bitemporal relational algebra", International Symposium on Computer and Information Sciences, pp. 622–633, 2006.

[2] M. Stonebraker, D. Moore, Object-Relational DBMSs: Tracking the Next Great Wave, San Francisco, The Morgan Kaufmann Series in Data Management Systems, 1999.

[3] J. Melton, Understanding Object-Relational and Other Advanced Features, San Francisco, Morgan Kaufmann Publishers, 2003.

[4] A.U. Tansel, J. Clifford, S.K. Gadia, S. Jajodia, A. Segev, R.T. Snodgrass. Temporal Databases: Theory, Design, and Implementation, San Francisco, Benjamin/Cummings, 1993.

[5] J. Ben-Zvi, The Time Relational Model, PhD, University of California, 1982.

[6] R.T. Snodgrass, "The temporal query language TQuel", ACM Transactions on Database Systems, Vol. 12, pp. 247–298, 1987.

[7] S.K. Gadia, "A homogeneous relational model and query languages for temporal databases", ACM Transactions on Database Systems, Vol. 13, pp. 418–448, 1988.

[8] G. Özsoyoğlu, M.Z. Özsoyoğlu, V. Matos, "Extending relational algebra and relational calculus with set-valued attributes and aggregate functions", ACM Transactions on Database Systems, Vol. 12, pp. 566–592, 1987.

[9] G. Bhargava, S.K. Gadia, "Relational database systems with zero information loss", IEEE Transactions on Knowledge and Data engineering, Vol. 5, pp. 76–87, 1993.

[10] C.S. Jensen, M.D. Soo, R.T. Snodgrass, "Unifying temporal data models via a conceptual model", Information Systems, Vol. 19, pp. 513–547, 1994.

[11] R.T. Snodgrass, I. Ahn, G. Ariav, D.S. Batory, J. Clifford, C.E. Dyreson, C.S. Jensen, R. Elmasri, F. Grandi, W. Käfer, N. Kline, K.G. Kulkarni, T.Y.C. Leung, N.A. Lorentzos, J.F. Roddick, A. Segev, M.D. Soo, S.M. Sripada, The TSQL2 Temporal Query Language, Dordrecht, Kluwer, 1995.

[12] D. Shasha, Y. Zhu, "SpyTime – a performance benchmark for bitemporal database", www.cs.nyu.edu/shasha/spytime/spytime.html, last accessed 1 December 2009.

[13] C. Combi, A. Montanari, G. Pozzi, "The T4SQL temporal query language", ACM International Conference on Information and Knowledge Management, pp. 193–202, 2007.

[14] M. Dumas, M.C. Fauvet, P.C. Scholl, "TEMPOS: a platform for developing temporal applications on top of object DBMS", IEEE Transactions on Knowledge and Data Engineering, Vol. 16, pp. 354–374, 2004.

[15] J. Yang, H. Ying, J. Widom, "TIP: a temporal extension to informix" Proceedings of the Special Interest Group on the Management of Data, pp. 596–671, 2000.

[16] V.T. Chau, S. Chittayasothorn, "A temporal compatible object relational database system", Proceedings of the IEEE Southeast Conference, pp. 93–98, 2007.

[17] J. Clifford, A. Croker, "The historical relational data model (HRDM) and algebra based on lifespans", Proceedings of the 3rd International Conference on Data Engineering, pp. 528–537, 1987.

[18] F. Wang, C. Zaniolo, "XBiT: an XML-based bitemporal data model", Proceedings of the 23rd International Conference on Conceptual Modeling, pp. 810–824, 2004.

[19] F. Wang, X. Zhou, C. Zaniolo, "Using XML to build efficient transaction-time temporal database systems on relational databases", Proceedings of the 22nd International Conference on Data Engineering, pp. 131–135, 2006.

[20] K.A. Ali, J. Pokorny, "A comparison of XML-based temporal models", Advanced Internet Based Systems and Applications, pp. 339–350, 2009.

[21] C.E. Atay, A.U. Tansel, Bitemporal Databases: Modeling and Implementation, Saarbrücken, VDM Verlag, 2009.

[22] A.U. Tansel, "Temporal relational data model", IEEE Transactions on Knowledge and Data Engineering, Vol. 3, pp. 464–479, 1997.