A New Optimized Model to Handle Temporal Data using Open Source Database

Shailender KUMAR[1], Rahul RISHI[2]
[1]*Department of Computer Science & Engineering, Ambedkar Institute of Advanced Communication Technologies and Research, Delhi, 110031, India*
[2]*Department of Computer Science & Engineering, University Institute of Engineering and Technology, Maharshi Dayanand University, Rohtak, 124001, India*
skumar@aiactr.ac.in

***Abstract*—The majority of the database applications now a days deal with temporal data. Temporal records are known to change during the course of time and facilities to manage the multiple snapshots of these records are generally missing in conventional databases. Consequently, different temporal data models have been proposed and implemented as an extension of the temporal less database systems. In the single relation model, the present and past instances are stored in a single relation that makes its handling cumbersome and inefficient. This paper emphasize upon storing the past instances of the records in the multiple historical relations. The current relations will manage the recent snapshot of data. The tuple time stamping approach is used to timestamp the temporal records. This paper proposes a temporal model for the management of time varying data built on the top of conventional open source database. Indexing is used to enhance the performance of the model. The proposed model is also compared with the single relation model.**

***Index Terms*—indexing, object oriented databases, open source software, query processing, runtime.**

I. INTRODUCTION

Time is an important characteristic of the real world. Extensive efforts have been put by the researchers [1-2] over the years to develop new techniques to effectively manage temporal data. Most of the work in this area is focused on the implementation of the time varying features on the top of conventional databases. This study is focused on the implementation of temporal model using open source database.

Temporal databases support three different time variants [3] of relations: system versioned, valid time period relations and bitemporal relations [4]. System versioned relation uses system time to capture changes made to the records. Valid time period relation captures the time during which the record is valid in the real world. The bitemporal relation [4] uses the union of valid and the system time to timestamp its records.

The system time is limited to the present and past instances of the records whereas the valid time includes future instance as well. There are two different approaches to append the timestamp in the relation: tuple time stamping that uses first normal form relations and attribute time stamping that uses non first normal form relations.

In spite of an extensive growth of the research interest in the area of temporal databases [5-8], there is no standard temporal database model till date. As per our knowledge, no work has been done to propose and implement the tuple time stamped [9-11] multiple historical relation model using open source database system. This paper proposes tuple time stamped multiple historical relation model (TTMHR) using valid time period as the time dimension to timestamp the records of the database. The database user is responsible for inserting the start and end times of the time varying records. Integrity constraints are imposed on the temporal data to ensure its accuracy and efficiency. In the proposed approach, multiple temporal relations are formed for all the time oriented attributes of the original temporal relation if their data values are not changing concurrently. Otherwise, the original temporal relation will remain intact.

The rest of this paper is organized as follows: Section II briefly highlights the related work on temporal database. Section III discusses the key issues in relation to the temporal database modeling. Section IV describes the proposed conceptual model using tuple time stamping approach with multiple temporal relations. In this section, the architecture of the proposed model and various data manipulation language operations on time varying data are also discussed. In Section V, the experimental results of the study are included. Query execution time or runtime is used to evaluate and compare the performance of the proposed model with the tuple time stamped single relation model. Finally in Section VI, we conclude by briefing the contributions of this paper.

II. RELATED WORK

There are a number of temporal data models that has been proposed and implemented [10], [12]. Few earlier works offered the analysis of some modern temporal data models [1]. The conceptual design of time varying data model largely focuses on the semantics [13-14], storage, query processing and implementation.

There are numerous temporal extensions that have been proposed and implemented by extending the relational data models [6], [8]. The two different methods to execute these extensions are tuple time stamping and attribute time stamping. Majority of the earlier work [2], [6] and [15-17] in temporal databases employ tuple time stamping approach. The tuple time stamping approach uses first normal form relations whereas attribute time stamping uses non first normal form relations.

Several papers investigated the performance issues of the

Digital Object Identifier 10.4316/AECE.2017.02008

proposed temporal data models. The work in [1] compared the performance of the applications developed using in built logic and the hand coded applications. Atay [4] compared the attribute and tuple time stamping approaches in bitemporal data models. The issues related to the performance of data models using attribute and tuple time stamping methods are also examined.

Another prominent feature considered in the designing of time variant databases is time dimension. The three different alternatives of time used are application or valid time, system time or transaction time and bitemporal [4]. Application time has been the most commonly used time dimension in temporal research papers [13]. It includes the present, past and future instances of records whereas the transaction time [6] includes only the present and past instances. Work is also carried out for the application of temporal logic in the field of artificial intelligence and robotics [18]. An indexing technique for processing queries in bitemporal databases is also proposed in Kaufmann et al. [19]. Data models have also been proposed to address issues related to indeterminacy [20] in temporal relational databases.

The tuple time stamped historical relation (TTHR) model discussed in Halawani et al. [2], [15] uses standard SQL for implementation of the model. In contrast to this, our work uses multiple current and history relations for the static and dynamic attributes. This is done to eradicate the redundancy caused by the tuple time stamping. Further, indexing on the non key attributes is applied to improve the query execution time. Although extensive research work has already been done in temporal data, the use of open source database in the implementation is very less [6]. Our research work uses the most powerful open source database Postgres for the implementation of the proposed model.

III. Preliminaries

A temporal data model is comprised of the conceptual tools for describing temporal schema and its integrity constraints.

A. Temporal Relational Schema

Temporal relation is a collection of time varying and time invariant attributes. The tuples of the temporal relations are time stamped with the defined time dimension. Temporal relational schema is represented mathematically in the formula (1) given below:

$$R^t = (A_1, A_2, A_3, ..., A_n \mid T) \quad (1)$$

Where R^t is a temporal relational schema with finite set of static and dynamic attributes $A_1, A_2, A_3, ..., A_n$ and T denotes the set of valid times. The proper design of the database Schema is vital in the correct retrieval of data.

B. Temporal Relational Algebra

Temporal relational algebraic expression is a combination of temporal algebraic operators and predicate symbols that is used in the logical expression to query temporal database. A temporal relational operator is an operator that yields temporal relation whenever applied to the temporal relations. The definitions of temporal relational operators used in the temporal relational algebra are different from that of the conventional relational algebra as they support time element. The temporal algebra is implemented in such a way that it should not violate any of the integrity constraints.

C. PostgreSQL

PostgreSQL is a powerful open source object relational database management system. The incessant growing feature list of Postgres includes capabilities like extensive backend support, high speed performance, security, good networking support and user friendly interface. It provides a very active community support, extensive research and analytics capabilities that make it an obvious choice for the future research.

Although, major database vendors like Oracle, Teradata and IBM provide temporal support but Postgres provide rich temporal data types support as compared to others. It handles time zones intelligently with a support for ISO 8601 standard. Apart from time zone flexibility, its interval date and time types like tsrange, tstzrange and daterange has a capacity of storing an interval of up to 178 million years with 14 digits precision and measures time at different precisions. These data types are the best temporal data types that can be used to store present, past and future instances of data. It is due to these features of Postgres that it is used as the tool for the implementation of the proposed model.

IV. Proposed Model

The proposed temporal model uses tuple time stamping approach for appending time element to the records. In this method of time stamping, the tuples of the temporal relation are time stamped using any of the three variants of the time dimensions. The different types of time dimension may be valid time, transaction time or bitemporal. We have used valid time as the time element in this paper.

A. Architecture of the Model

The architecture of the proposed temporal model is illustrated in Fig. 1. The model is developed using postgreSQL database and it is capable of effectively managing time varying data.

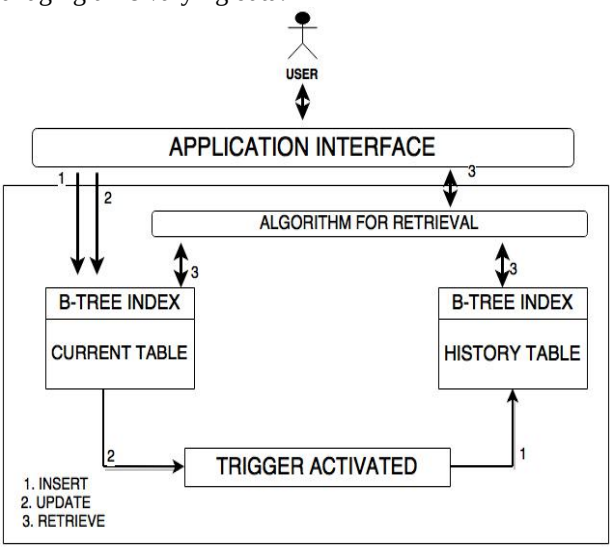

Figure 1. Architecture of the model

The user can interact with the temporal database using application interface developed using Java NetBeans IDE version 8.0.2. The application enables the user to perform

data manipulation operations on both present and historical data. The database is seen as a collection of current table and the history table. The current tables store the current snapshot of data. All the current tables in the database use tuple time stamping approach [2] to store all those tuples which are currently valid for the real world. Insertion of new records is done in current tables only. The history tables stores the past versions of the data. When the update operation is applied on the current table, the old values of the records will be automatically inserted into history table. All the queries related to the past instances will be satisfied by the history tables.

Triggers are generally used whenever operations such as update is being carried out on the present table. In such a case, the old values of the tuple to be updated moves to the history table with upper bound of the time_range attribute set equal to the lower bound of current table updated time_range. Delete operation is not permitted in temporal databases.

Tuple time stamping is also applied to the history table. It uses tsrange data type for the time_range column and upper bound of this column is equal to the lower bound of the time_range of the newly updated tuple. The primary key of the history table is the union of the primary key of the present table and the time_range attribute of the history table. B tree indexing is applied on the non key attribute to minimize the query execution time.

B. Representation of Database

Fig. 2 shows the ER diagram of the proposed database model. The proposed model distributes the time varying attributes in multiple relations and the non temporal attributes are gathered in distinct relation. This is done to remove the redundancy caused by the tuple time stamping of the static and dynamic attributes in the original temporal relation. The user of the database is responsible for inserting the start and end valid time of the time varying records in the current table. Integrity constraints are imposed on the temporal data to ensure its accuracy and efficiency of the database.

The tables of the current database are named by adding front as the suffix whereas all the tables of the history database are named by adding back as the suffix. The employees and the department tables contain all the temporal less attributes which do not change over the time. The primary key of the present tables salaryfront, titlefront, deptempfront is emp_no whereas the primary key of the table deptmangfront is dept_no. The primary key of all the history tables is the combination of the primary key of the corresponding present table and the time_range attribute of the history table. The ER diagram also illustrates the multiplicity of the relationship between the tables.

A trigger functions are used which get activated every time an update operation is being carried out on the tables of the current database. This will result into insertion of updated record in the current table and the movement of earlier snapshot from the current table to the history table. The start valid time of the updated tuple in the current table will be recorded in the history table as the end valid time of the previous instance.

C. Insert Operation

Insertion operation in temporal database is carried out by the user entering the new data values for the static and dynamic attributes. Insertion of new records takes place in the current relations only. The user will set the valid start time and valid end time of the time range attribute time_range.

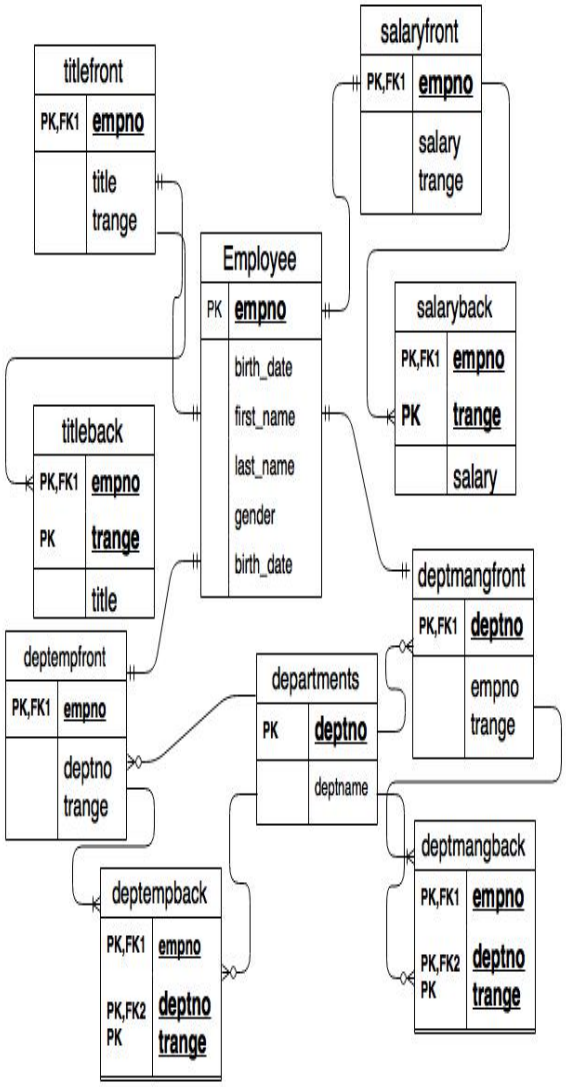

Figure 2. ER diagram of the Database

Algorithm

```
INSERT INTO table_name(column1,column2,….,ColumnN,
time_range)
VALUES
(value1,value2,…….valueN,tsrange(date1,date2))
IF (PRIMARY KEY ALREADY EXISTS IN table_name) then
RETURN unique_key_constraint_violation
Go to step 1.
ELSE
RETURN insertion successful
ENDIF
```

Example:

insert into employees(emp_no, birth_date, first_name, last_ name, gender, hire_date)
values(1,'1994/01/01','axel','blue','M','2014/01/01');
insert into salaryfront(emp_no,salary,time_range)
values (1,10200, TSRANGE ('2014/01/01 00:00:00', '9999/01/01 23:59:00', '[]'));

D. Update Operation

Update operation in temporal database is different from the conventional database. Every time the update operation is carried out on present table, the already defined trigger

will get activated right before that update. This trigger will save update values as new and those which are being updated as old values. The old values of the current table will be inserted into history table with the upper bound of the time_range column set equal to the lower bound of the time_range attribute of the current instance.

Algorithm

```
UPDATE present_table
SET
column1=value1,column2=value2,….columnN=valueN,tim
e_range=tsrange(date1,date2)
WHERE conditions
INVOKE TRIGGER trigger_name
IF(old.values IS DISTINCT FROM new.values)
BEFORE UPDATE ON present_table
EXECUTE PROCEDURE trigger_function
INSERT INTO
history_table(time_range_column,column2,….
columnN)
VALUES(time_range_type(lower(old.time_range),
lower(new.time_range)),old.column2,….old.columnN)
SET old.values IN present_table TO new.values
RETURN UPDATE SUCCESSFUL
ELSEIF(LOWER(NEW.TIME_RANGE)<LOWER(OLD.TIME_RANGE)
) THEN
RETURN input error
ENDIF
Example :
UPDATE salaryfront
SET salary=10499,time_range=TSRANGE('2019/09/27
00:00:00', '9999/01/01 23:59:00', '[]')
WHERE emp_no=10002ANDupper(time_range) =
'9999/01/01 23:59:00';
```

E. Retrieval Operation

There are three different categories of data retrieval in the proposed model. In the first case, the user is interested in the current snapshot of the data. Therefore, the user can directly query the current table and get the latest instance of data. In the second case, time_period defined in the predicate of the query lies in the time_range of history table. Therefore, such query will be satisfied by the history table only. In the third case, the query includes a time period that overlaps the time_range attribute values of both the current and history table. Hence, such queries retrieve data from both the tables.

Algorithm

```
SELECT * FROM
function_name(parameter1,time_period)
IF(time_period &>(SELECT present_table.time_range
from present_table where(condition satisfies
parameter1 )))
RETURN TUPLE FROM present_table
ELSEIF (time_period << (SELECT present_ table.
time_ range from present_table where (condition
satisfies parameter1)))
RETURN TUPLE FROM history_table
ELSE
RETURN TUPLE FROM BOTH present_table AND history_
table
WHERE present_table.time_range && time_period AND
history_table.time_range&&time_period
ENDIF
```

Example :

```
select get_sal.e_no,get_sal.sal,get_sal.t_range,get_ti.tit from
get_sal(10100,'2016/08/03 00:00:00','9999/01/01 23:59:00' )
 INNER JOIN get_ti (10100,'2016/08/03 00:00:00',
'9999/01/01 23:59:00') on (get_sal.t_range) &&
(get_ti.t_range) ORDER BY get_sal.t_range;
```

V. Experiments And Results

The proposed temporal extension of the Postgres comprises of two primary components: temporal data type and the set of operators used in the implementation of temporal functionality. Postgres in built time range data type “tsrange” is used for time stamping the records of the database. It uses a pair of disjoint times such that the tuple is valid during this period.

A. System Requirements

The experiments are carried out on host system with 8GB of RAM and Intel(R) Core (TM) i5 3210M CPU@2.5GHz on Windows operating system. The open source database PostgreSQL version 9.5 is used to implement the proposed model. The application is built using Java NetBeans IDE version 8.0.2.

B. Data Set

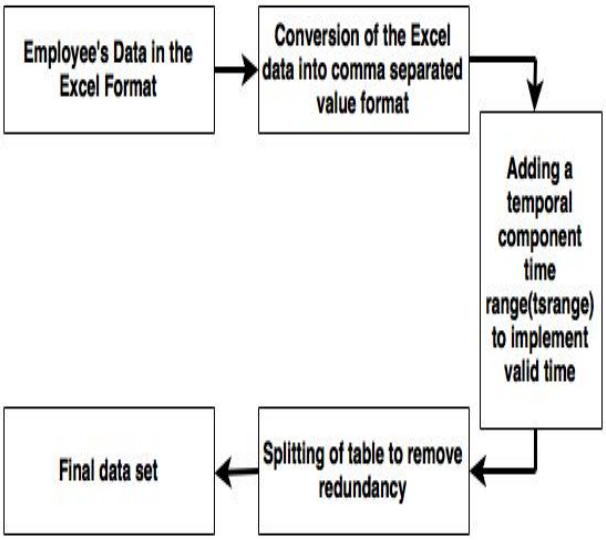

Figure 3. Priory processing of the Data Set

In view of evaluating the proposed model, query tests are performed on the dataset of one lakh employees of a hypothetical company. The original employee dataset has been priory processed as depicted in Fig. 1. The time range data type “tsrange” is used to time stamp the tuples of the temporal relation. The original temporal relations are decomposed into sub relations to enable the extermination of redundancy that has occurred due to presence of static and dynamic attributes together.

C. Test Results

This sub section presents the comparison of the proposed model with the single relation model. The single relation model uses same relation for storing present and past instances of data. The comparison between the two models is done on the basis of the query execution time [1], [4]. The query execution time is the time taken to execute the optimal query execution plan. A total of nine point or range queries are used for the purpose of comparison. Each query is executed ten times and the mean of all these execution times is considered for comparison. Table I summarizes the list of sample point and range queries [16] used for testing the performance of both the models. The query execution time for the sample queries is collected from the pgAdmin tool of Postgres. The tool provides the estimated time to execute the optimal query plan. Therefore, the average of ten execution times of each sample query is considered for comparison of the proposed model with the single relation model.

TABLE I. LIST OF SAMPLE QUERIES

Query number	Explanation
Q1	Select column names from salary table where employee number is 10100 and time period is ('2016/08/03 00:00:00','9999/01/01 23:59:00');
Q2	Select column names from salary table where employee number is 10100 and time period is ('1990/08/03 00:00:00','2000/01/01 23:59:00');
Q3	Select column names from salary table where employee number is 10100 and time period is ('1990/08/03 00:00:00','9999/01/01 23:59:00');
Q4	Inner join between salary and title table with employee number is 10100 and time period is ('2016/08/03 00:00:00','9999/01/01 23:59:00') on salary.time_range && title.time_range ;
Q5	Inner join between salary and title table with employee number is 10100 and time period is ('1990/08/03 00:00:00','2000/01/01 23:59:00')on salary.time_range && title.time_range ;
Q6	Select column names from salary table where employee number is 10100 and date is '2016/08/03 00:00:00'::timestamp;
Q7	Select column names from salary table where employee number is 10100 and date is '1994/08/03 00:00:00'::timestamp;
Q8	Inner join between salary and title table with employee number is 10100 and date is '1994/08/03 00:00:00'::timestamp on salary.time_range && title.time_range ;
Q9	Select column names from salary where salary<4000 and upper(time_range)='9999/01/01 23:59:00';

Table II shows the mean execution times of both the models for the queries listed in Table I. For majority of queries the execution time is better in the proposed tuple time stamped multiple historical relation model. The speedup is calculated by taking the ratio of the mean query execution times for the two models as depicted in formula (2).

$$S_T = \frac{\mu_{SRM}}{\mu_{PM}} \quad (2)$$

Where, S_T is the speedup, μ_{SRM} is the mean query execution time of the single relation model and μ_{PM} is the mean query execution time of the proposed model.

TABLE II. MEAN QUERY EXECUTION TIME AND THE SPEEDUP

Query number	Mean query execution time in Single relation Model	Mean query execution time in proposed Model	Speedup
Q1	0.347	0.287	1.20
Q2	0.396	0.358	1.10
Q3	0.343	0.417	.822
Q4	0.620	0.504	1.23
Q5	0.645	0.594	1.08
Q6	0.328	0.272	1.20
Q7	0.337	0.303	1.11
Q8	0.585	0.503	1.16
Q9	445.403	0.369	1207.05

The queries Q1, Q4, Q6 gives output from the current table and queries Q2, Q5, Q7, Q8 gives output from the history table. On the contrary, Query Q3 returns output combined from the current and history table. It is evident from the results that query execution time of the proposed model is better when output of the query is only from the current or the history table. Whereas, for the queries that needs data values from both present and history table, the single relation model performs better than the proposed model.

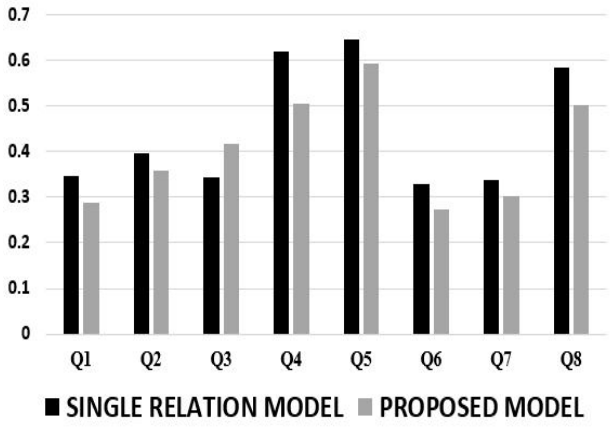

Figure 4. Query execution times of both the models.

B tree index is used on salary attribute of temporal relation salaryfront. Consequently, the query execution time of query Q9 for the proposed model is far better than the single relation model. The graph in Fig. 4 shows the query execution time taken by both the models for the execution of first eight sample queries listed in Table I.

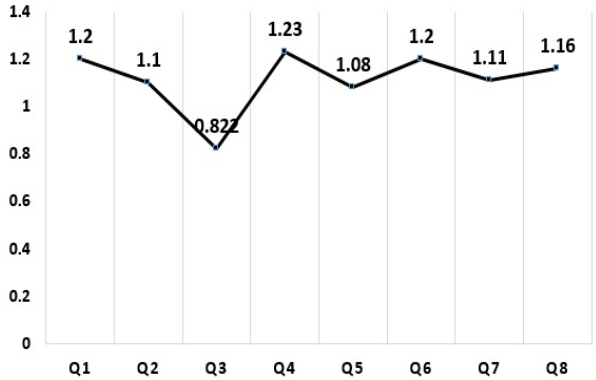

Figure 5. Speed Up of the Proposed Model as compared to Single Relation Model

VI. CONCLUSION

There is an urgent requirement of good temporal data model and powerful query language for the effective management of time varying data. In this article, we have proposed a tuple time stamped multiple historical relation data model which is capable of managing temporal data in an efficient way. The implementation of the model is done using open source database. The PostgreSql version 9.5 in built time range data type "tsrange" is utilized for time stamping the records of the database. The static attributes are grouped in a different relation to avoid redundancy of data. Separate temporal relation is created for all the temporal attributes if their valid time varies.

The query execution time is used to compare the performance of the proposed model with the tuple time stamped single relation model. The results of the comparison reveal that in terms of the query execution time, the proposed model is better than the single relation model except in the case where the query output is generated collectively from current and history tables. Indexing is applied on the frequently accessible non key attributes to improve the query execution time. As a future scope, this

work may be extended to consider other improvements apart from those covered in this article.

REFERENCES

[1] F. Kunzner, D. Petkovic, "A comparison of different forms of temporal data management," Beyond Databases, Architectures and Structures. Springer International Publishing, pp 92-106, 2015. doi:10.1007/978-3-319-18422-7¬_8

[2] S. M. Halawani, N. A. Al-Romema, "Retrieval optimization technique for tuple timestamp historical relation temporal data model," Journal of Computer Science 8.2, pp 243-250, 2012. doi:10.1.1.683.7887

[3] R. D. M. Galante, C. S. D. Santos, N. Edelweiss, A. F. Moreira, "Temporal and versioning model for schema evolution in object-oriented databases," Data & Knowledge Engineering, pp 99-128, May, 2005. doi:10.1016/j.datak.2004.07.001

[4] C. Atay, "An attribute or tuple timestamping in bitemporal relational databases," Turkish Journal of Electrical Engineering & Computer Science, pp 4305-4321, 2016. doi:10.3906/elk-1403-39

[5] M. Kvet, K. Matiasko, M. Vajsova, "Sensor based transaction temporal database architecture," in IEEE World Conference on Factory Communication Systems (WFCS), 2015. doi:10.1109/WFCS.2015.7160547

[6] J. Mate, J. Safarik, "Transformation of relational databases to transaction-time temporal databases," 2nd Eastern European Regional Conference on the Engineering of Computer Based Systems (ECBS-EERC), 201. doi:10.1109/ECBS-EERC.2011.14

[7] I. A. Goralwalla, M. T. Ozsu, D. Szafron. "A framework for temporal data models: exploiting object-oriented technology," Proc. of IEEE TOOLS 23 on Technology of Object-Oriented Languages and Systems, 1997. doi:10.1109/TOOLS.1997.654697

[8] C. S. Jensen, R. T. Snodgrass, "Temporal data management," IEEE Transactions on Knowledge and Data Engineering, pp 36-44, 1999. doi:10.1109/69.755613

[9] N. Edelweiss, P. N. Hubler, M. M. Moro, G. Demartine, "A temporal database management system implemented on top of a conventional database," Proc. XX IEEE International Conference of the Chilean Computer Science Society, 2000. doi:10.1109/SCCC.2000.890392

[10] C. Yang et al., "Standardization on bitemporal information representation in BCDM," IEEE International Conference on Information and Automation, 2015, pp. 2052-2057. doi:10.1109/ICInfA.2015.7279627

[11] A. U. Tansel, "On handling time-varying data in the relational data model," Information and Software Technology, pp 119-126, 2004. doi:10.1016/S0950-5849(03)00114-9

[12] V. T. N. Chau, S. Chittayasothorn, "A temporal compatible object relational database system," in Proc. of IEEE SoutheastCon, 2007. doi:10.1109/SECON.2007.342862

[13] L. Anselma, P. Terenziani, R. T. Snodgrass, "Valid-time indeterminacy in temporal relational databases: Semantics and representations," IEEE Transactions on Knowledge and Data Engineering, pp. 2880-2894, Dec. 2013. doi:10.1109/TKDE.2012.199

[14] C. S. Jensen, R. T. Snodgrass, "Semantics of time-varying information," Information Systems, pp 311-352, June, 1996. doi:10.1016/0306-4379(96)00017-8

[15] S. M. Halawani, N. A. Al-Romema, "Memory storage issues of temporal database applications on relational database management systems," Journal of Computer Science, pp 296-304, 2010. doi:10.1.1.165.759

[16] M. H. Bohlen, R. Busatto, C. S. Jensen. "Point-versus interval-based temporal data models," in Proc. of 14th IEEE International Conference on Data Engineering, 1998. doi:10.1109/ICDE.1998.655777

[17] N. Edelweiss, J. P. M. Oliveira, B. Pernici, "An object-oriented temporal model," International Conference on Advanced Information Systems Engineering, Springer Berlin Heidelberg, 1993, pp. 397-415. doi:10.1007/3-540-56777-1_21

[18] M. Pomarlan, "Visibility-based planners for mobile robots capable to handle path existence queries in temporal logic," Advances In Electrical and Computer Engineering, pp 55-64, Feb. 2014. doi:10.4316/AECE.2014.01009

[19] M. Kaufmann et al., "Bi-temporal Timeline Index: A data structure for processing queries on bi-temporal data," 31st IEEE International Conference on Data Engineering, 2015. doi:10.1109/ICDE.2015.7113307

[20] P. Terenziani, "Irregular indeterminate repeated facts in temporal relational databases," IEEE Transactions on Knowledge and Data Engineering, pp 1075-1079, Apr. 2016. doi:10.1109/TKDE.2015.2509976