A 1NF Data Model for Representing Time-Varying Data in Relational Framework

Nashwan Alromema
Department of Computer Science,
Faculty of Computing and Information Technology
Rabigh, Saudi Arabia

Fahad Alotaibi
Department of Information System,
Faculty of Computing and Information Technology
Jeddah, Saudi Arabia

***Abstract*—Attaching Date and Time to varying data plays a definite role in representing a dynamic domain and resources on the database systems. The conventional database stores current data and can only represent the knowledge in static sense, whereas Time-varying database represents the knowledge in dynamic sense. This paper focuses on incorporating interval-based timestamping in First Normal Form (1NF) data model. 1NF approach has been chosen for the easily implementation in relational framework as well as to provide the temporal data representation with the modeling and querying power of relational data model. Simulation results revealed that the proposed approach substantially improved the performance of temporal data representation in terms of required memory storage and queries processing time.**

***Keywords*—Time-oriented data model; time-varying data, valid-time data; transaction time data; bitemporal data; data model; N1NF; 1NF**

I. INTRODUCTION

Temporal Database (TDB) is database modeling technique that is considered as repositories of time-dependent data. Several research works have been conducting in this research area starting from the 1970s [1]. Some of these works deal with storage structure and temporal DBMS prototype, while others concentrated on query processing temporal time indexing [2]-[6]. The research work by Snodgrass in [7] treats the problems of temporal databases models, integrity constraints, storage structures, and implementation techniques using different DBMS. A debate within the last three decades was on how to model, implement and query temporal database in efficient way [8]. Since conventional relational database is used to store and process the data that refer to the current time [2], commercial DBMS and standards for the query language do not fully support temporal features [3], [21]. There are two fundamental methods of creating temporal database applications. The first one is an integrated method where the time-varying features of the data are supported by an extended or modified internal model in DBMS. The second method is the stratum method, in which the temporal features of the data are implemented in top of standard DBMS by a layer over DMBS which then changes the outcome into its temporal data [9]. The greatest efficiency is offered by the first method however the second method has greater popularity due to its realism.

A number of temporal data models have been proposed since the early 1980s. These data models are based on schema extension approach of relational data model. There are two common approaches for these extensions, tuple timestamping with First Normal Form (1NF), and attribute timestamping with Non-First Normal Form (N1NF). The study in [17] generalized the models under 1NF approach into Tuple Timestamping Single Relation (TTSR), and Tuple Timestamping Multiple Relations (TTMR) according to the way of data representations. TTSR approach is not efficient since it introduces redundancy, where attribute values that change at different time are repeated in multiple tuples. However, TTMR approach have solved the problem of data redundancy in TTSR, the problem with this approach is that the fact about a real world entity is spread over several tuples in several relations, and combining the information for an object a variation of join known as temporal intersection join would be needed which is generally expensive to be implemented. For N1NF, the problem with this approach, as stated in Jensen [6], there are some difficulties of temporal data models capturing an object in a single tuple such that "the models may not be capable of directly using existing relational storage structures or query evaluation techniques that depend on atomic attribute values". The study in [3] shows an approach of partial implementation of temporal database capabilities in top of widely used commercial DBMS, the model in this study is categorized under TTSR. This study also lacks most of temporal features as well as data redundancy of the proposed representational data model. The study in [10], [19] show an approach of temporal database representation in standard SQL under TTMR approach, the study explains number of examples of temporal data and how temporal manipulations of such data can be effected using standard SQL. A Column Level Temporal System (CLTS) proposed by Kvet in [20] is TTMR approach, the main issue of this model is to keep the duplicity of data minimal. As reducing the duplicities of the data is considered one of the important factors which improve processing speed to get a current snapshot and all data during life cycle of the database object [22]. Atay and Tansel in [18] proposed the Nested Bitemporal Relational Data Model (NBRDM) under N1NF approach [18], NBRDM model attached bitemporal data to attributes and defined a bitemporal relational algebra and a bitemporal relational calculus language for the proposed data model.

In this paper, we describe an approach for implementing temporal database in the framework of relational data model over the most widely used commercial DBMSs (Oracle RDBMS). The proposed approach does not significantly

This Research Project is sponsored by the Deanship of Scientific Research (DSR)-King Abdulaziz University –Jeddah-Saudi Arabia

change the procedures of designing and developing information systems. The major contributions of this research project can be formulated as follows:

- Describe the meaning and use of temporal features in the framework of relational data model.
- The approach is restricted to use the existing technology of designing and implementing databases applications.
- Incorporating temporal aspects need to minor modifications without affecting the performance of the parts of the system that do not use temporal data.
- The proposed implementation approach represents the temporal database in a data model that has expressive power, has less storage memory comparing to other works under TTSR approach, and efficient query processing.
- The implementation is easy, does not cost much, and based on relational database not on XML files as in [11].

This paper utilizes the following concepts on temporal database theory: the representation of real world time as a line. Every point in the line is referred to as an instance, a period is the time separating two instances, and an interval is the duration of loose segment of the time-line. Temporal data types in a temporal database can be identified as an instant of time, period and interval [7]. It is conceivable that time extends infinitely into the past of the future, as such when the relational database model has time introduced to it, it should be limited to delineate a particular time. Time-line chronons is the term for the reading of the time-line clock in the time-line. A time instance is delineated by each tick of a clock. To increase familiarity with temporal description times on a time line clock are expressed though a calendar.

The time line clock chronon is defined as day, month, and year on the Gregorian calendar. The date "22nd of June 2009" is an example. Granules are time points and the dividing scheme that splits the time line into a measurable collection of time segments is referred to as granularity and is an aspect of all temporal information [12]. Temporal databases are depicted by the discrete time model because it is easy and comparatively simple to use [9]. Temporal databases have formulated a taxonomy of time which identifies when a particular event happens or when a given statement can be regarded at factual. User-defined time is one interpretation of the time feature employed in temporal databases. It is expressed in the data that is of the date/time kind (the birth date column for example) and does not suggest anything correlated to the validity of the other columns or temporal time, wherein the column(s) that contain date/time information types are employed to mark the related tuple's time aspects. There are three categories of temporal time. Valid time: where in the related time is employed to determine when a particular statement (event-based) happened or when a particular statement (interval based) is regarded as being factual in the real world [13]. Transaction time: the related time is in reference to the period when the data was actually retained inside the database. Bitemporal-time: the related time is connected to the yield of valid-time and transaction time in the model of bitemporal data. Tuples are regarded as valid at instances of that time by rollback databases [7], [8].

II. METHODOLOGY

Designing any database systems usually goes through three phases, namely, (1) Conceptual design; (2) Logical design; and finally (3) Physical design. The temporal aspects of database schemas are complex and difficult; therefore it is an error-prone to design. In designing temporal database, the same steps as the mentioned above can be followed, in addition to that, defining new features concerning the time aspects because both conventional conceptual model and relational data model do not fully support time-varying aspects. The following steps summarize the proposed methodology for designing temporal database in relational data model.

- Designing the conceptual model for the business logic of the system and map it into conventional relational data model using the mapping methodology described in [5], [7], [10], where all temporal aspects that need to be modeled are ignored at this step. All conventional methods which are used to construct good relational database schema by analyzing the design and applying different forms of normalization should take place in this step.
- Adding the temporal aspect for all the database objects that need to keep the historical changes of the entities' data.

To make this process clear, an example (proof of concept) of the conceptual data model shown in Fig. 1 for EMPLOYEE and DEPARTMENT relations are mapped into relational data model shown in Fig. 2. Adding the temporal aspects to these two relations is shown in Fig. 4. These approaches can apply to any other domain of database technology like biomedical domain, business intelligent, metrological and any other domains.

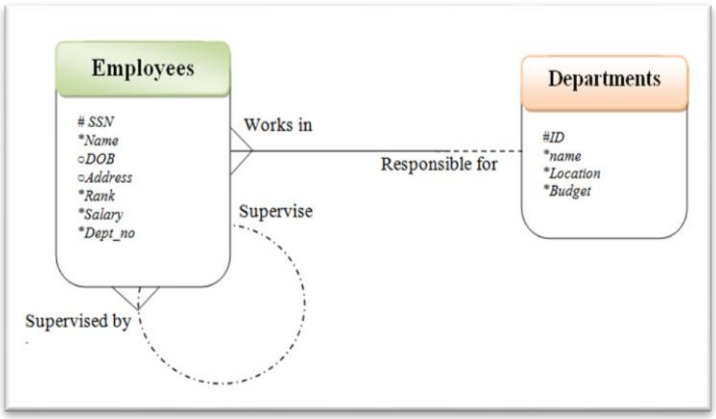


Fig. 1. The conceptual schema of EMPLOYEE and DEPARTMENT database entities.

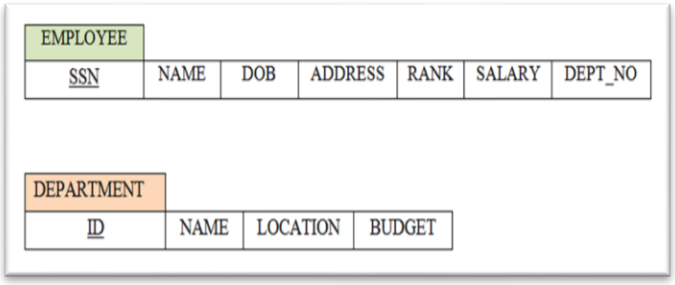


Fig. 2. The relational schema of EMPLOYEE and DEPARTMENT database relations.

III. Representations Methodology

The methodology of representing temporal database in this paper is accomplished by using Tuple Timestamp Historical Relational (TTHR) data model. Fig. 3 shows the conceptual structure of TTHR model. The database applications is directly connected to the main tables which hold the current valid time data, this feature gives the advantages that TTHR can be adapted to any functioning database systems without any changes to the infrastructure. The historical changes of each time-varying attributes in any table are stored in corresponding temporal database table (auxiliary tables) as shown in Fig. 3. The data representation of temporal database in TTHR is accomplished by firstly, defining the database object (entities /relations) for which we want to track the historical changes of the stored data, then we add for each such relations two additional columns Lifespan Start Time (LSST) and Lifespan End Time (LSET), which indicate the beginning and the end of the time interval within which the database object exists in the modeled reality [14], [19]. Secondly, for each such entity /relation, we create an additional relation with the same name as in the basic schema with the suffix VT, we use VT to indicate the valid time model. Example, the relational table EMPLOYEE in Fig. 2, is represented into temporal database (Fig. 4) by adding two additional columns LSST and LSET, after that we create a new table, the schema representation of Table_VT as an example of EMPLOYEE_VT will be: *EMPLOYEE_VT= (SSN, index, Update_A_VST, VET).*

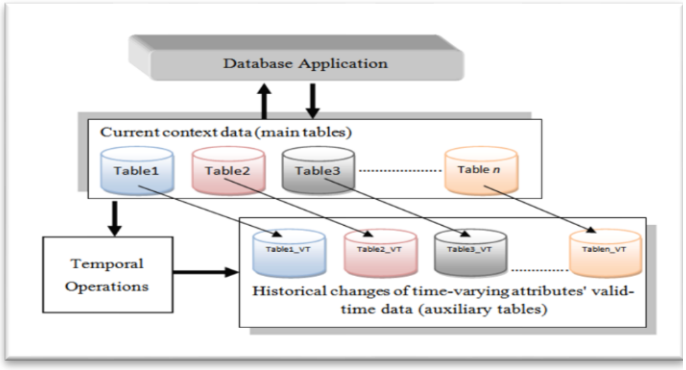
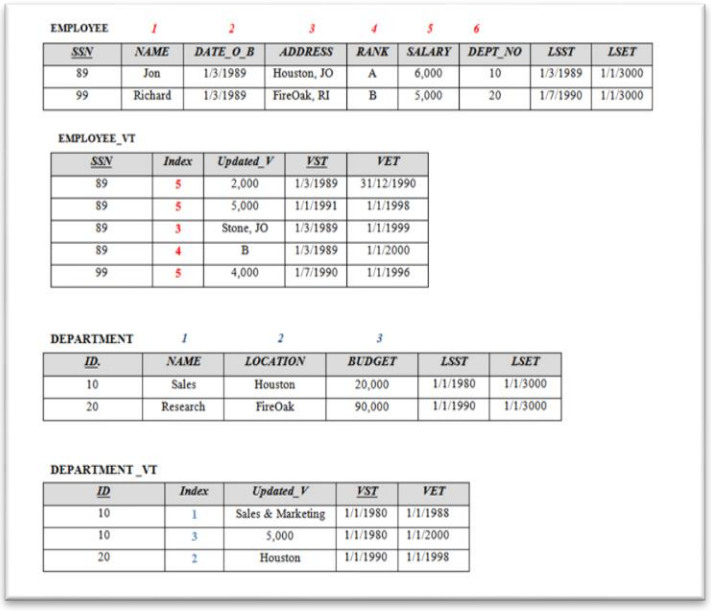


Fig. 3. The conceptual structure of TTSR Model.

The data in the basic table keeps the latest updated data (current data), whereas *Table_VT* stores the historical changes of the validity of the updated attributes in the basic table.



EMPLOYEE

SSN	NAME	DATE_O_B	ADDRESS	RANK	SALARY	DEPT_NO	LSST	LSET
89	Jon	1/3/1989	Houston, JO	A	6,000	10	1/3/1989	1/1/3000
99	Richard	1/3/1989	FireOak, RI	B	5,000	20	1/7/1990	1/1/3000

EMPLOYEE_VT

SSN	Index	Updated_V	VST	VET
89	5	2,000	1/3/1989	31/12/1990
89	5	5,000	1/1/1991	1/1/1998
89	3	Stone, JO	1/3/1989	1/1/1999
89	4	B	1/3/1989	1/1/2000
99	5	4,000	1/7/1990	1/1/1996

DEPARTMENT

ID.	NAME	LOCATION	BUDGET	LSST	LSET
10	Sales	Houston	20,000	1/1/1980	1/1/3000
20	Research	FireOak	90,000	1/1/1990	1/1/3000

DEPARTMENT _VT

ID	Index	Updated_V	VST	VET
10	1	Sales & Marketing	1/1/1980	1/1/1988
10	3	5,000	1/1/1980	1/1/2000
20	2	Houston	1/1/1990	1/1/1998

Fig. 4. The relational schema of EMPLOYEE and DEPARTMENT database relations.

A. Modification Operation

Modify temporal data database is a challenges because of the time dimension [15]. In this representational data model, we consider the insertion, deletion, and update of records in the table of the basic schema, the data in the Table_VT are updated automatically using database triggers or application's function. The following is the rules of data modification operations:

Insertion Operation: Inserting a new record into a table of the basic schema is accomplished as in conventional database, in addition to that the value of LSST field is set to the current date, and the value of the LSST field is set to a very far future time, for example, 1/1/3000. This date is always greater than the current date for the lifespan of the application. Inserting data into Table_VT is accomplished as consequences of updating any attribute in the table of the basic schema as it will be explained in updating operation. Thus, the data in the table of the basic schema always represents the latest current valid data.

Updating Operation: updating a record in a table of the basic schema results into the following actions: 1) If the updated data is an indexed attribute(s) as shown in Fig. 4, then the old value of this attribute and its index with the same value of the primary key and VST and VET fields are inserted into Table_VT, the value of VST and VST can be calculated as follows: (a) if this is the first time to update this attribute (this attribute has not been updated before or no record for this attribute is found in Table_VT), then VST value will have the same value as LSST in the table of the basic schema, and VET will be having the value of the current time. (b) If this attribute has been updated before, then VST will be having the value of VET plus one time granule of the latest update of this attribute. An example of this case is shown in Fig. 3, when the value of the SALARY attribute indexed by 5 has been updated (at time point '1/1/1996') for Richard, and then we look at Table_VT at

that time point, since no record has been found for this attribute and for this object, thus a new record for the updated value of this attribute and corresponding database object has been inserted into Table_VT table with these values:

$$(SSN:99, index:5, Update\_A:4,000, VST:'1/7/1990', VET:'1/1/1996').$$

Another scenario is shown in Fig. 4, for database object Jon when the SALARY attribute indexed by 5 has been updated (at time point '1/1/1998'), then we look for the latest update for SALARY attribute for Jon in Table_VT which is on '31/12/1990', We add one day (assumed in our example the time granularity is one day). The new row is will be having these values:

$$(SSN:89, index:5, Update\_A:5,000, VST:'1/1/1991', VET:'1/1/1998').$$

2) If the updated data is LSET attribute with instance time not equal to 1/1/3000, then this action is considered as logical delete of this record and this record stops to be a life or valid, as if one employee has resigned from the company.

Delete Operation: Delete a record form the basic schema is accomplished by setting the value of LSET to current time as explained in update operation.

In our proposed schema representation Table_VT tables keep the historical changes of the validity of the updated attributes in the basic table. Each record in Table_VT represents the validity of the changed attributes in the basic table during the time interval [VST, VET]. The historical changes of the validity is continuous, the timestamp in VST field coincides with the value of VET field of the preceding record with the same primary key. Fig. 3 shows the schema representation and the update operations on the basic schema tables (EMPLOYEE and DEPARTMENT) and the temporal tables (EMPLOYEE_VT, DEPARTMENT_VT).

The scheme described in Fig. 4 does not address many subtle issues specifically for temporal database [16]. An example of these issues are, constraints on the upper and lower time boundaries of interval-based data model, since the time is discrete, the above schema cannot guarantee that LSST should be less than LSST in the basic schema, and VST should be less than VET in temporal schema. Overlapping of the same fact that belong to the same object, an example the SALARY of an employee is $5000 in the interval [1/1/1990, 1/1/1999] and different salary $7000 is valid in the interval [1/1/1996, 1/1/2005]. Referential integrity constraints, it might have an object in reference relation refers to another object in the referee relation in different time points. As an example, in EMPLOYEE relation (reference) the foreign key Dept_no can have the ID value of DEPARTMENT relation (referee) that is either logically deleted or have interval lifespan time [LSST, LSET] that is not fully cover the interval lifespan time of EMPLOYEE object. Although, these issues can't be verified by conventional DBMS, these problems can be solved by an additional check through triggers of applications functions.

Although the historical changes of data are in temporal schema and the latest current valid data available from the basic schema, our approach is useful for the following reasons:

- Integrity constraints in the basic schema as well as temporal schema can be defined and implemented in DBMS easily without any major update to the existing applications. The purpose of this implementation is to ensure the creation of highly reliable databases.
- The proposed implementation removes data redundancy and satisfied high level of memory storage saving comparing to other implementation techniques discussed in [17], reducing the redundant data will help to facilitate efficient query execution.
- The tables in the temporal schema are updated only by insert operation when specific attribute in the basic schema table updated, thus the growth of this table depends on the frequency of attributes updates.

The current valid data in basic schema table helps in efficient query execution because some queries does not need to have temporal data. Temporal-joins involving data from the temporal schema are less efficient than joins of the tables in the basic schema.

B. Query Operations

Querying temporal databases represented by our approach using standard SQL2 can be classified into current query, sequenced query and non-sequenced query [7], [27]. Current query provide the current valid data which is in the basic schema table, while sequenced query provide the data that were valid during a certain interval of time where this data can be obtained from basic schema, temporal schema, or both depends on the complexity of the query, non-sequenced provide the historical changes of database objects' data. This work presents the following types of queries:

Current Queries: Current query is an ordinary query which provides current values of the data regardless of the time dimension. We project current queries on the basic table schema where the latest current values are stored for example the query that selects the current SALARY and RANK of an employee is

```
SELECT E.SALARY, E.RANK
FROM EMPLOYEE E
WHERE E.SSN = 89;
```

Some current queries involving time predicates for excluding/including valid/not valid lifespan entities, an example, the query that selects the latest SALARY of not valid lifespan employees is

```
SELECT E.SALARY
FROM EMPLOYEE E
WHERE E.LSET <> '1/1/3000';
```

This query selects all employees whom are logically deleted by setting the value of LSET to an instance time not equal to the END_TIME which we consider it in our approach equal to this date '1/1/3000'.

Sequenced Queries: Sequenced query provide the data that were valid during a certain interval of time, and the result of the query is valid-time table unlike current query which returns snapshot state. For example the query that returns the salary of an employee in a certain point of time or in a certain interval of time is

Q1 for point of time t

```
SELECT ES.SSN, ES.upadated_v
FROM EMPLOYEE_VT ES
WHERE ES.index = 5 and
         ES.VST <= t and
         ES.VET >  t and
         ES.SSN = 89;
```

Q2 for interval of time [t_1 ,t_2]

```
SELECT ES.SSN, ES.upadated_v
FROM EMPLOYEE_VT ES
WHERE ES.index = 5 and
      ES.VST < t2 and
      ES.VET >= t1 and
      ES.SSN = 89;
```

Q1 returns exactly one record, whereas Q2 returns one or more records because the time intervals for the salary historical changes of same employee might have an overlap with the input time interval [t1 ,t2]. No duplicated records will be returned for both queries because the data in our model are coalesced [10]. In contrast to other models that need more processing for coalescing function.

Non-sequenced query: provide the historical changes of a database objects' data during their lifespan time, the result of the query is valid-time table like sequenced query. The complexity of Non-sequenced queries depends on number of tables involved because the intervals in which the selected records were valid must be overlap for different tables. For temporal queries we need to define three functions for time interval manipulations as follows:

- Overlap([X,Y], [Z,W]) function takes two time intervals as a parameters, and returns one (1) if the time intervals are overlap and zero (0) otherwise. The following the code in SQL2 for this function.

```
CREATE FUNCTION OVERLAP (X IN NUMBER, Y
   IN NUMBER, Z IN NUMBER, W IN NUMBER)
   RETURN NUMBER    IS
BEGIN
    RETURN
     CASE
      WHEN X < W AND Y >= Z
        THEN 1
        ELSE 0
        END;
  END OVERLAP;
```

- Upper_bound(Y,W) function takes the tow upper boundaries of two time intervals as a parameters, and returns upper boundary of the overlapped time intervals. The following is the code in SQL2 for this function.

```
CREATE FUNCTION UPPER_BONUD (Y IN
   NUMBER, W IN NUMBER) RETURN NUMBER
   IS
   BEGIN
     RETURN
     CASE
      WHEN Y >= W THEN W
      WHEN Y < W THEN Y
      ELSE 0
      END;
  END UPPER_BONUD;
```

- lower_bound(X,Z) function takes the tow lower boundaries of tow time intervals as a parameters, and returns lower boundary of the overlapped time intervals. The following is the code in SQL2 for this function.

```
CREATE FUNCTION LOWER_BONUD (X IN
NUMBER, Z IN NUMBER) RETURN NUMBER
IS
BEGIN
  RETURN
     CASE
    WHEN X >= Z THEN X
    WHEN X <  Z  THEN Z
    ELSE 0
    END;
END LOWER_BONUD;
```

Since the current data are in the basic schema table and the historical changed data are in the temporal schema, then combining these data into one place can be accomplished by database views. We can create view for each time-varying attributes in the basic schema table, for example the SALARY_V view can hold the track log data including the current data for the salaries of all employees. The SALARY_V view is defined as follows:

```
CREATE VIEW SALARY_V AS
   SELECT E.SSN, E.SALARY,
          MAX (CASE
               WHEN ES.VET IS NULL
               THEN E.LSST
               WHEN ES.VET IS NOT NULL
               AND E.LSST > ES.VET
               THEN E.LSST
               WHEN ES.VET IS NOT NULL
               AND E.LSST < ES.VET
               THEN (ES.VET +1 )END)
             AS VST, E.LSET AS VET
FROM EMPLOYEE E LEFT OUTER JOIN
      (SELECT ES.SSN,
      TO_NUMBER(ES.UPADATED_V), ES.VST,
      ES.VET FROM EMPLOYEE _VT ES
      WHERE ES.ATT_INDEX = 5)
      ON E.SSN = ES.SSN
GROUP BY E.SSN, E.SALARY, E.LSET
UNION
```

```
SELECT SSN, TO_NUMBER(UPADATED_V), VST,
VET
FROM EMPLOYEE _VT WHERE INDEX = 5;
```

An example of the query that returns the track log of the salary of an employee for his lifespan time is

```
SELECT * FROM SALARY_V
WHEN SSN =89;
```

Another query that selects the track log information about salary and address (ADDRESS_V is view created by the same way as SALARY_V) of an employee is

```
SELECT S.SSN, AD.ADDRESS, S.SALARY,
  LOWER_BONUD(S.VST, AD.VST)AS VST
  ,UPPER_BONUD(S.VET,AD.VET)AS VET
    FROM ADDRESS_V AD, SALARY_V S
   WHERE SSN =89 AND AD.SSN = S.SSN
  AND OVERLAP (AD.VST,AD.VET, S.VST,
              S.VET) = 1;
```

Above queries can be applied for any other temporal information in employee or department tables. With time, the tracking log query that retains a data for a certain time interval might have a different data in other time interval.

IV. Results and Discussion

The performance evaluation of the proposed model is considered in terms of memory storage efficiency and query processing time. TTHR is compared with the main models in literature namely TTSR and TTMR. The Employees relation in Fig. 4 is represented by the three models, and the size in byte for the attributes in Employees relation is given as in Table I. The cost improvement of the memory storage is considered during one lifespan time and with a frequency of time-varying attributes update equal to 5. The results of memory storage efficiency for the three models are shown in Table I.

Note: Snp Stands for Snapshot and His for History.

TABLE I. Cost Model of Employees Relation Represented by TTSR, TTHR and TTMR

Attribute name	S/ Byte	Cost of data representation where $\delta = 5$								
		TTSR			TTHR			TTMR		
		Snp	His	Total	Snp	His	Total	Snp	His	Total
SSN	9	9	27	36	9	27	36	63	45	108
Name	100	100	300	400	100	0	100	100	0	100
B_date	10	10	30	40	10	0	10	10	0	10
Address	20	20	60	80	20	0	20	9	9	18
Tel_no	9	9	27	36	9	0	9	9	0	9
Spr_SSN	9	9	27	36	9	0	9	9	0	9
Dno	3	3	9	12	3	0	3	3	6	9
Salary	6	6	18	24	6	0	6	6	12	18
Rank	1	1	3	4	1	0	1	1	0	1
VST	10	10	30	40	0	30	30	70	50	120
VET	10	10	30	40	0	30	30	70	50	120
LSST	10	10	30	40	10	0	10	10	0	10
LSET	10	10	30	40	10	0	10	10	0	10
index	1	0	0	0	0	3	3	0	0	0
$\beta = S(\alpha)$	20	0	0	0	0	60	60	0	0	0
Total Cost				1176			371			542

Many parameters affect the cost improvements of TTHR over other models, Fig. 5 shows the cost improvements where all the parameters have been fixed with varying the values of the frequency of time-varying attributes update from 5 to 440 times in a period of time. TTHR has achieved significant saving in storage memory space that ranges between 68%-81% over TTSR approach, and 10%-32% over TTMR that is based on the average change of the time varying attributes. TTHR has achieved some significant saving in storage memory space that is roughly equal or greater than TTMR. The proposed temporal data model is suggested for its simplicity as fewer database objects will be needed to capture the temporal aspects of time-varying data compared to TTMR. Moreover, applying TTHR to an existing database application does not require many changes compared to TTMR. Moreover, the only need is to create the auxiliary relation to capture the historical changes of time-varying attributes but without touching the system itself. This is contrary to TTMR, where the relations need to be decomposed and the integrity constraints need to be redefined.

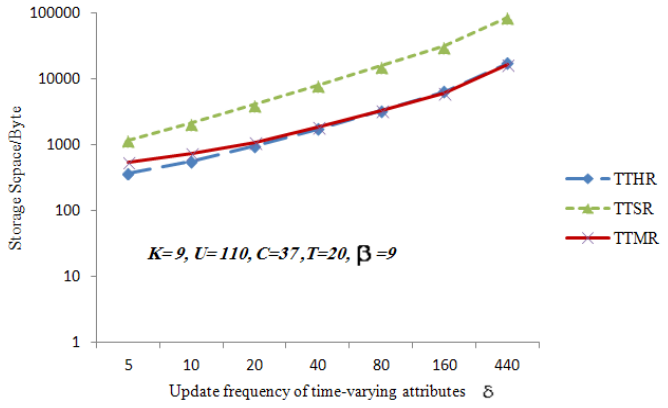


Fig. 5. Cost improvement of Employees relation represented by TTSR, TTHR and TTMR in one lifespan time [0, 10], and variations of δ.

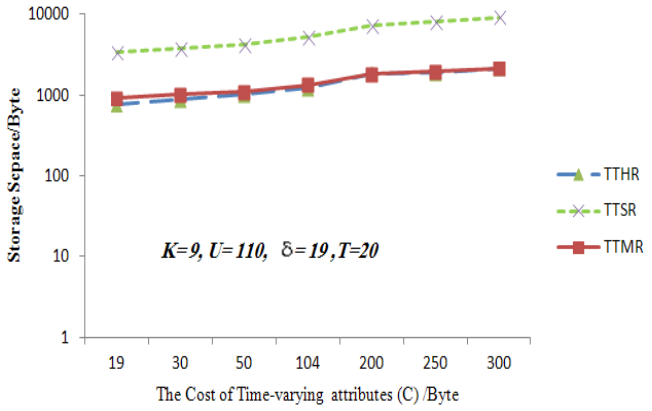


Fig. 6. Cost improvement of Employees relation represented by TTSR, TTHR and TTMR in one lifespan time.

Fig. 6 shows the storage costs of the temporal relational approach after freezing all the parameters and varying the sizes of the time-varying attributes. For these values, TTSR-based approach shows worse storage costs comparing to TTMR-based and TTHR-based approaches. However, the graph shows a positive indication that TTHR can be used as an efficient storage that is better than TTMR-based approach until the value of 150 byte. After this point it seems that both TTHR and TTMR have the same storage efficiency.

Fig. 7 shows the storage efficiency after freezing all the parameters and varying the sizes of key attributes (K) value variations. We increase value from 9 to 300 bytes. As we can see, the TTHR-based approach shows the best storage efficiency than the others. However, it is shown that the difference of storage efficiency is marginal between the TTHR-based approach and the TTMR-based approach.

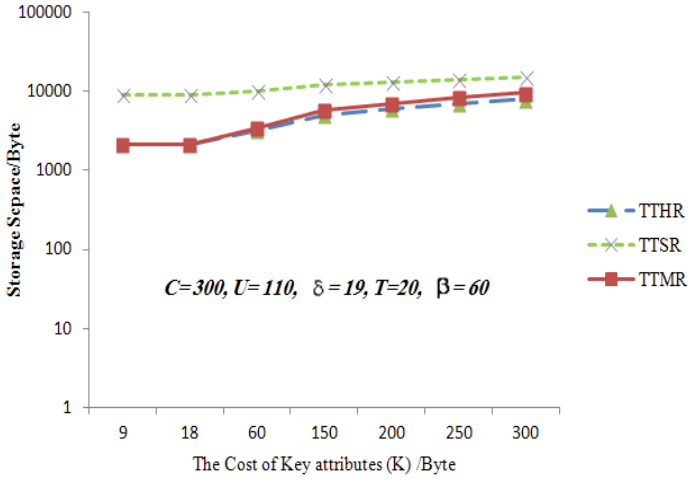


Fig. 7. Cost improvement of Employees relation represented by TTSR, TTHR and TTMR in one lifespan time [0, 10], and variations of Key attributes' size (K).

For query processing time, an experiment has been carried out on the database shown in Fig. 4 with a data set consists of 108,004 instances of Employees. This data set has been randomly generated in the three models to simulate real-world scenarios (the same approach has been taken by Anselma [23]. The SQL Trace facility and TKPROF (Transient Kernel Profiler) are two basic performance diagnostic tools that have been used for queries analysis in the three approaches. TKPROF program outputs the parameters of each query as CPU, Elapsed, Disk, and Query such that:

CPU(C): is time in seconds executing.

Elapsed (E): is the time in seconds executing.

Disk (D): is the number of physical reads of buffers from disk.

Query (Q): is the number of buffers gotten for consistent read.

Queries from 1 to 10 have been run in sequence for each approach. Table II shows the experimental results of executing these queries for each Model.

TABLE II. AN OUTPUT OF QUERY PROCESSING EXPERIMENTAL RESULTS

		TTHR			TTSR			TTMR		
Temporal	**Q**	**C**	**D**	**Q**	**C**	**D**	**Q**	**C**	**D**	**Q**
Current	Q1	0.00	3	3	0.00	1	4	0.00	15	25
	Q2	0.32	1193	8325	0.40	1251	8315	1.10	598	11467
	Q3	0.01	0	1199	0.01	0	1260	0.03	0	356
	Q4	0.00	0	3	0.00	0	7	0.00	0	6
	Q5	0.00	2	5	0.00	2	11	0.00	2	11
Non-sequenced	Q6	0.15	6	8332	2.43	0	351872	0.17	0	7552
	Q7	0.17	0	1206	2.17	0	351896	0.10	0	7672
	Q8	1.31	0	9538	5.84	0	696645	1.70	0	8054
Sequenced	Q9	0.01	12	18	0.01	5	30	0.00	6	12
	Q10	0.29	0	2869	1.03	0	95272	0.28	0	2038

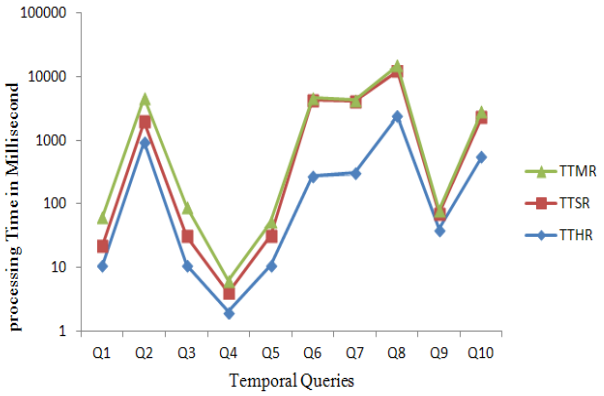


Fig. 8. Query processing time for the 10 queries in the three models.

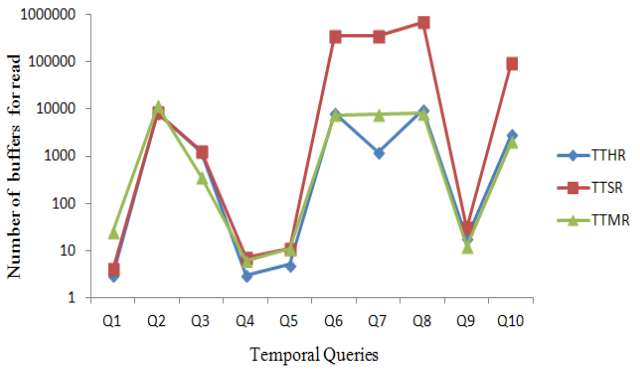


Fig. 9. Number of Buffers read in the three models for the 10 Queries.

From Table II, Fig. 8 and 9 have been plotted to compare the performance of each model in graphical view. It can be shown that TTSR satisfies good query performance in current query (Q1-Q5); the same performance is achieved by TTHR. However, TTMR costs a lot for current queries, but it costs less for both, sequenced (Q6, Q7 and Q8) and non-sequenced (Q9, and Q10) queries and the same performance is achieved by TTHR. TTSR costs a lot for both sequenced and non-sequenced queries due to coalesce function that needs to be applied to the query results to make sure the query result is in snapshot equivalence.

SQL developer suite with TKPROF has been used for these experiments. Measuring the performance of the query by only running the query few times is a pretty bad idea - equivalent to just accepting that the cost of the explanation plan that tells you the best query. Therefore, it is really a need to take into account what resources query is taking up and therefore how it could affect the production system.

V. Conclusion

The 1NF temporal data model proposed in this study uses a novel approach for modeling and implementing interval-based temporal database in relational framework [24]-[27]. In our approach the issues concerning the memory storage and query efficiency, and application development procedures are considered. All of these issues ensure the development of efficient and reliable temporal database over conventional DBMS. In this paper, we proposed an approach for representing temporal data that achieves saving in memory usage range from 68-81% over other temporal representations, and speed up the processing time of current snapshot data. Finally, our approach has better storage representation, reduce query complexities.

Acknowledgment

This paper was supported by the Deanship of Scientific Research (DSR), King Abdulaziz University. The authors, therefore, acknowledge with thanks to DSR's technical and financial support.

References

[1] Findler, N. V., & Chen, D. (1973). On the problems of time retrieval of temporal relations causality, and coexistence. International Journal of Computer & Information Sciences, 2, 3, 161-185.

[2] Date, C. D., Darwen, H., & Lorentzos, N. A. (2003). Temporal data and the relational data model. San Francisco: Morgan Kaufmann.

[3] Novikov, B. A., & Gorshkova, E. A. (2008). Temporal databases: From theory to applications. Programming and Computer Software, 34, 1, 1-6. Pleiades Publishing, Ltd., 2008. Original Russian Text

[4] Tansel, A. U. (2004). On handling time-varying data in the relational data model. Information and Software Technology, 46, 2, 119-126.

[5] Elmasri, R., and Navathe (2000). Fundamentals of Database Systems. 3rd edition. Addison Wesley.

[6] Jensen, C. S., Clifford, J., Gadia, S. K., Segev, A., & Snodgrass, R. T. (1992). A glossary of temporal database concepts. ACM Sigmod Record, 21, 3, 35-43.

[7] Snodgrass, R. T., (2000). Developing Time-Oriented Database Applications in SQL, 1st edition, Morgan Kaufmann Publishers, Inc., San Francisco.

[8] Jensen, C. S., Snodgrass, R. T., & Soo, M. D. (1995). The tsql2 data model (pp. 157-240). Springer US. http://people.cs.aau.dk/~csj/Thesis/pdf/chapter12.pdf

[9] Patel, J. (2003). Temporal Database System Individual Project. Department of Computing, Imperial College, University of London, Individual Project, 18-June-2003, http://www.doc.ic.ac.uk/~pjm/teaching/student_projects/jaymin_patel.pdf

[10] Zimányi, E. (2006). Temporal aggregates and temporal universal quantification in standard SQL. ACM SIGMOD Record, 35, 2, 16-21.

[11] Wang, F., Zhou, X., & Zaniolo, C. (2006, April). Using XML to build efficient transaction-time temporal database systems on relational databases. In Proceedings of the 22nd International Conference on Data Engineering, 2006. ICDE'06 (pp. 131-131). IEEE.

[12] A-Qustaishat, M. (2001). A visual temporal object-oriented model embodied as an expert C++ Library. ADVANCES IN MODELLING AND ANALYSIS-D-,6, 3/4, 3-43.

[13] Bohlen, M. H., Busatto, R., & Jensen, C. S. (1998, February). Point-versus interval-based temporal data models. In Proceedings of 14th International Conference on Data Engineering, (pp. 192-200). IEEE.

[14] Dyreson, C., Grandi, F., Käfer, W., Kline, N., Lorentzos, N., Mitsopoulos, Y., ... & Wiederhold, G. (1994). A consensus glossary of temporal database concepts.ACM Sigmod Record, 23, 1, 52-64.

[15] Tansel, A. U. (2006). Modeling and Querying Temporal Data. Idea Group Inc.

[16] Tansel, A. U. (2004). Temporal data modeling and integrity constraints in relational databases. In Computer and Information Sciences-ISCIS 2004 (pp. 459-469). Springer Berlin Heidelberg.

[17] Halawani, S. M., & Romema, N. A. (2010). Memory storage issues of temporal database applications on relational database management systems. Journal of Computer Science, 6, 3, 296.

[18] Atay, C. (2016). An attribute or tuple timestamping in bitemporal relational databases. Turkish Journal of Electrical Engineering & Computer Sciences. (2016) 24: (pp. 4305 – 4321). doi:10.3906/elk-1403-39.

[19] Noh, S.Y., Gadia, S.K. and Jang, H., (2013). Comparisons of three data storage models in parametric temporal databases. Journal of Central South University, 20(7), pp.1919-1927.

[20] Kvet, M., Matiako, K. and Kvet, M., (2014). Transaction management in fully temporal system. In Computer Modelling and Simulation (UKSim), 2014 UKSim-AMSS 16th International Conference on (pp. 148-153). IEEE.

[21] Snodgrass R, Ahn I. Performance evaluation of a temporal database management system. Commun ACM 1986; 15:96-107.

[22] Arora, S. (2015). A comparative study on temporal database models: A survey. In Advanced Computing and Communication (ISACC), 2015 International Symposium on (pp. 161-167). IEEE.

[23] Anselma, L., Stantic, B., Terenziani, P., and Sattar, A. (2013). Querying now-relative data. Journal of Intelligent Information Systems, 41(2), 285-311.

[24] Halawani, S.M., AlBidewi, I., Ahmad, A.R. and Al-Romema, N.A., 2012. Retrieval optimization technique for tuple timestamp historical relation temporal data model. Journal of Computer Science, 8(2), p.243.

[25] Nashwan Alromema, Mohd Shafry Mohd Rahim and Ibrahim Albidewi, "A Mathematical Model for Comparing Memory Storage of Three Interval-Based Parametric Temporal Database Models" International Journal of Advanced Computer Science and Applications (ijacsa), 8(7), 2017. http://dx.doi.org/10.14569/IJACSA.2017.080741

[26] Alromema, N.A., Rahim, M.S.M. and Albidewi, I., 2016. Temporal Database Models Validation and Verification using Mapping Methodology. VFAST Transactions on Software Engineering, 11(2), pp.15-26.

[27] Ab Rahman Ahmad, N.A., Rahim, M.S.M. and Albidewi, I., 2015. Temporal Database: An Approach for Modeling and Implementation in Relational Data Model. Life Science Journal, 12(3).