Journal of Computer Science 8 (2): 243-250, 2012
ISSN 1549-3636
© 2012 Science Publications

Retrieval Optimization Technique for Tuple Timestamp Historical Relation Temporal Data Model

Sami M. Halawani, Ibrahim AlBidewi,
Ab Rahman Ahmad and Nashwan A. Al-Romema
Department of Information Systems,
Faculty of Computing and Information Technology in Rabigh,
King Abdulaziz University, Rabigh 21911, Saudi Arabia

Abstract: Problem statement: In the field of database technology, Recoding time related data are referred to as temporal databases. Conventional relational databases are queried using the underlying constructs of SQL which are translated into Relational Algebra (RA). Relational Algebra (RA) is the writing of logical expressions that uses a set of relational operators to perform operations on specific relation(s) and returns results as relation. RA operations are not applied on querying temporal database, because temporal database holds a sequence of snapshot relations. **Approach:** This study focuses on the retrieval of temporal database, by extending RA to TRA. TRA is involved to help querying temporal database that is modeled using Tuple Timestamp Historical Relation (TTHR) Model. **Results:** A technique for querying temporal database modeled by TTHR is the topic of this study. We examine the current issues and problems in temporal database and propose data retrieval optimization technique for temporal database. **Conclusion:** We proposed a data retrieval optimization technique for querying temporal database; this technique applies for the temporal database applications that are modeled by TTHR.

Key words: Relational Algebra (RA), Tuple Timestamp Historical Relation (TTHR), Temporal Relational Algebra (TRA), Temporal Normal Form (TNF)

INTRODUCTION

Temporal database is defined as “a database which supports some aspect of time, not counting user-defined time” or the database system that stores time-varying data (Kostenko, 2007). Several terminologies can be used to define temporal database like time-varying, time-oriented and historical database systems. Traditional database systems are designed to store and process the state of the reality at a sigle point of time, whearas temporal database have the capability to record and process time-varying aspects of the real in past present and feature world Haraty and Bekaii, (2006). Temporal database With growing sophistication of DBMS applications, the lack of temporal support in conventional DBMS raises serious problems when used to develop temporal database applications. Some of the approaches of developing temporal data model suggested extending the standard relational data model so that it supports time varying data and the other approaches are based on extending the snapshot model with time appearing as additional attributes Haraty and Bekaii, (2006). The following approaches which are discussed in Patel (2003) and Torp *et al*. (1999) underline how a temporal database can be created:

- Build a complete TDBMS from scratch that provides a primitive data type and handles the different states/time instances of data being stored (integrated approach)
- Developing a technique that extends non-temporal data model to temporal data model on top of conventional DBMS that acts as stratum (stratum approach)

The first approach involves building a complete TDBMS from bottom up which is a very large and time consuming task, it is also difficult as the underlying principles used by commercial DBMS to optimize operations must be reformed and a lot of theoretical work needs to be carried out to show that the new system is fully complete, the amount of time and manpower required for this approach is similar to that needed by commercial vendors to develop DBMS that

Corresponding Author: Nashwan A. Al-Romema, Faculty of Computing and Information Technology in Rabigh, King Abdulaziz University, Rabigh 21911, Saudi Arabia

we all are familiar with today. While the second approach does not involve any changes to the existing database technology and simply might be developed as we just build new technique for temporal support on top of the existing conventional DBMS that will be used. A Tuple Timestamp Historical Relation (TTHR) is a temporal database model that is proposed by Halawani and Romema (2010), where the relation that needs to capture temporal time aspects decomposed in to two relations, one represents the current state relation and the other recodes the changes in all the time varying attributes. Literature Review

MATERIALS AND METHODS

Temporal Relational Algebra (TRA): TRA is a set of temporal specific operators which helps writing logical expressions for querying temporal database; in addition to that TRA includes all the RA operators. The new temporal operators which are described can be used only for historical database, these operators are known as Until Since (US) logic, US logic operators are sample extension of classical logic that have been shown to be mathematically sound by the research on temporal logic and temporal properties. US logic have proven semantically to be popular operators for querying temporal database which are used in the language of first order temporal logic discussed in (Mariusz, 2007).

Until and since logical operators (US): The conventional relation algebra operators: select (σ), project (Π), product (X), union ($\cup$) and difference (-) together with the new temporal specific operators which are extended from conventional relational algebra are used for querying temporal database. These operators are Until and Since logical operators (Patel, 2003). The semantic of until and since is considered as that, for any two facts A and B in temporal database we can say the following:

A Until B means: A must hold at all times until the time B holds. A holds for all future snapshots up to and including the snapshot when B holds. At time n, the result is false.

A Since B means: A must have held at all times since B held. A holds for all past snapshots back to and including the snapshot when B held. At time 0, the result is false.

N is the latest time/data that the temporal database can be timestamped by and 0 represent the starting time point of the temporal database. Whereas Finger (2000) defined the semantic of Until and Since logic S (Since) and U (Until) as the following:

- S(A,B) reads ‘since A was true in the past, B holds and U(A,B) reads 'until A is true in the future, B holds’
- The operators S and U are assumed to be strict because they inform nothing about the
- Current time events, example B is not forced to hold at the time when A holds

Other unary connective operators which could be derived from Since and Until logical operator are shown in Table 1, where these operators are called modal operators.

These operator are derived from US logic as described in Patel (2003), we reused these operators for querying temporal database which are modeled by TTHR-temporal database model in Halawani and Romema (2010). Interval based model versus point based model in the temporal database involves period and there are a set of predicate and constructors for period temporal data type, where about 13 possible relationship between two periods can be constructed explained in Elmasri and Navathe (2000). We use these constructors as auxiliary relational operators for implementing TRA in conventional RA in Table 2, there is a summary of these functions, where we assumed that [a, b] is a period and [c , d] is another period.

Implementing TRA in RA: Querying temporal database can be accomplished by using the new TRA operators or by using the conventional RA after begin revised in order to manipulate temporal data (keeping in account the time interval) and extract correct information from temporal database. Implementing TRA in terms of RA is discussed in Patel (2003) where the concept of Temporal Normal Form (TNF) is reviewed, which means that TRA works properly on temporal database that is in TNF, These publications are referenced when describing the temporal operators in this study. an example is the relation schema in Fig. 2 that will be used and referred for all examples illustrated in this study, where.

Employee table represents the current state table of the employee and the lifespan time of the employee is considered as part of the modeled reality where we add LSST and LSET.

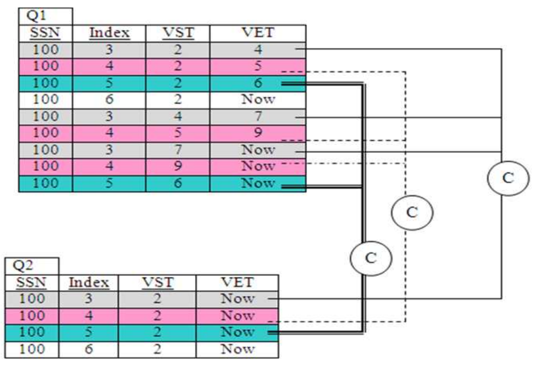
VT-Employee table represents the valid-time aspect of time varying attributes of employee table which are considered and indexed as 3, 4, 5 and 6 for (address, rank, salary and department) respectively as shown below, where we have data for 3 employees and historical valid-time data for their time varying attributes.

Table 1: Derived modal operator for temporal relational algebra

Operator	Meaning	Semantic
♦R	Past	The tuples of R that held at sometime in past or that are in snapshot before the current snapshot (Derived from since operator)
• R	Previous	The tuples of R that held at one tick in the past or that are in snapshot at the previous moment of the current snapshot (derived from Since operator)
■ R	Always in the past	The tuples of R that held at all times in the or that are present all snapshots before the current snapshot (Derived from since operator)
◊ R	Future	The tuples of R that hold at sometime in future or that are in snapshot after the current snapshot (Derived from until operator)
o R	Next	The tuples of R that hold at the next moment or that are in snapshot after the current snapshot. (Derived from until operator)
R	Always	AlwaThe tuples of R that hold at all times in thein future or that are present in all snapshots after the current snapshot (Derived from Until operator)

Table 2: Auxiliary functions for temporal database query

Function	Meaning	Semantic
[a, b] **Overlap** [c, d]	Period [a, b] and [c, d] both has a common shared instance time.	If a<d and b>c then Overlap Else No overlap End if;
Min (a, d)	The minimum value of pair of time instance values	If a< = d then Return d; Else Return d; End if;
Max (a, d)	The maximum value of pair of time instance values	If a> = d then Return a; Else Return d; End if;



Q1

SSN	Index	VST	VET
100	3	2	4
100	4	2	5
100	5	2	6
100	6	2	Now
100	3	4	7
100	4	5	9
100	3	7	Now
100	4	9	Now
100	5	6	Now

Q2

SSN	Index	VST	VET
100	3	2	Now
100	4	2	Now
100	5	2	Now
100	6	2	Now

Fig. 1: Example of Coalesce Function on Employee Relation

Time-relation each tuple in this table contains the time point and the corresponding date that is used in the other relations, we use this table as mapping (look up) table, that represents each time point, e.g., time point 1 represent the time granule 01/01/2000 and time point 2 represents 01/02/2000. We assumed that the time model in this example is considered with time granularity equal to one month. Temporal variable (Now) can be represented as unreachable date like 01/01/3000.

Temporal relational algebra:

Project Π_t:

$$\Pi_{t\ A,B,C}\ R = C(\Pi_{t\ A,B,C}\ R)$$

Project A, B and C attributes from temporal relation R at time point ‘t’ need to be nested in a function known as a coalesce function (we use “C” for denoting coalesce function) which is shown in Fig. 1 and referenced from Patel (2003), this function used for the purpose of storing the result of query in a relation and merge all tuples that have the same non-temporal attributes and overlapping intervals Zimanyi (2006), the goal of this function is to have the result of the query in TNF. To make the idea of coalesce function clear, let us query ssn, index, VST and VET from VT-employee relation for employee with ssn = 100 by:

$$\Pi_{t\ SSN,index,VST,VET}\ \text{VT-employee}$$

The result of this query is in Q1 which violates TNF, we might need this output to be input to another query, so applying the coalesce function is desired to have the result as shown in Q2.

Select σ_t:

$$\sigma_t\, R$$

Select operator is used for retrieve the historical data that are valid at specific time point. As an example is to retrieve the employees’ salary at time point 3, the result is shown bellow:

Q

SSN	Index	α	VST	VET
100	5	200	2	6
102	5	250	1	4

Product X_t:

$$S\ X_t\, R = \Pi_{S,\ R,\ \max(R.VST,\ S.VST),\ \min(R.VET,\ S.VET)}\,(\sigma_{overlap(S,\ R)}\ (SX\ R)$$

Temporal product operator can be implemented by using the project and select operators with addition to the auxiliary functions as shown in the above formula.

Temporal product is different from non-temporal product that binds each tuple from relation S with each tuple from relation R, the result will be a relation of Size(S) * Size(R), the temporal product bind each tuple from S with each tuple from R if and only if their respective time periods overlap. For example we consider that S represents the historical valid-time data of the salary of employee with ssn = 100 and R represents the historical valid-time data of the address of the same employee, by applying the temporal product to these two relations we get the result as the following.

Since-product S_{xt}: Since product Temporal operator can be implemented in conventional RA by using the project and select operators, in addition to the auxiliary functions as shown in the above formula, the output goes through C (Coalesce) function which translates the result of the query into temporal normal form as discussed in (Patel, 2003).

Until-product U_{xt}:

$S\ U_{xt}\ R = C\ (\Pi_{S,\ R,\ S.VST\ -\ 1,\ \min\ (R.VET,\ S.VET)\ -1}$
$\sigma_{overlap\ (S,\ R)}$ (SX R)

Until product Temporal operator can be implemented in conventional RA by using the project and select operators in addition to the auxiliary functions as shown in the above formula, where the output goes through C (Coalesce) function that translate the result of the query into temporal normal form as discussed in (Patel, 2003):

Join $_t \bowtie$:

$S \bowtie_t R =_{S,\ R,\ \max\ (R.VST,\ S.VST),\ \min\ (R.VET,\ S.VET)}$ and
$_{S.common\text{-}att\ =\ R.common\text{-}att}\ \sigma_{overlap(S,\ R)}$ (SX R)

Temporal join is the same as temporal product operator. In addition to that, the common attribute between the two relations should be identical. Temporal natural join is different from non-temporal natural join that would bind only the tuples from relation S with tuples from relation R, where the join is done on the common attributes which is shared by the two relations, this is used to check if two tuples can be joined, for a temporal join, a tuple from the S table is joined to a tuple form R table if their respective time periods overlap as well as obeying the existing conditions to perform a non-temporal join, the since and until join operators are extended from temporal join and defined as the following:
Since Join S $\bowtie_t$:

$S \bowtie S_t\ R = C\ (\Pi_{S,\ R,\ \max\ (R.VST,\ S.VST),\ S.VET\ +\ 1}$ and
$_{S.common\text{-}att\ =\ R.common\text{-}att}\ \sigma_{overlap(S,\ R)}$ (SX R)

Until Joi n U$\bowtie_t$:

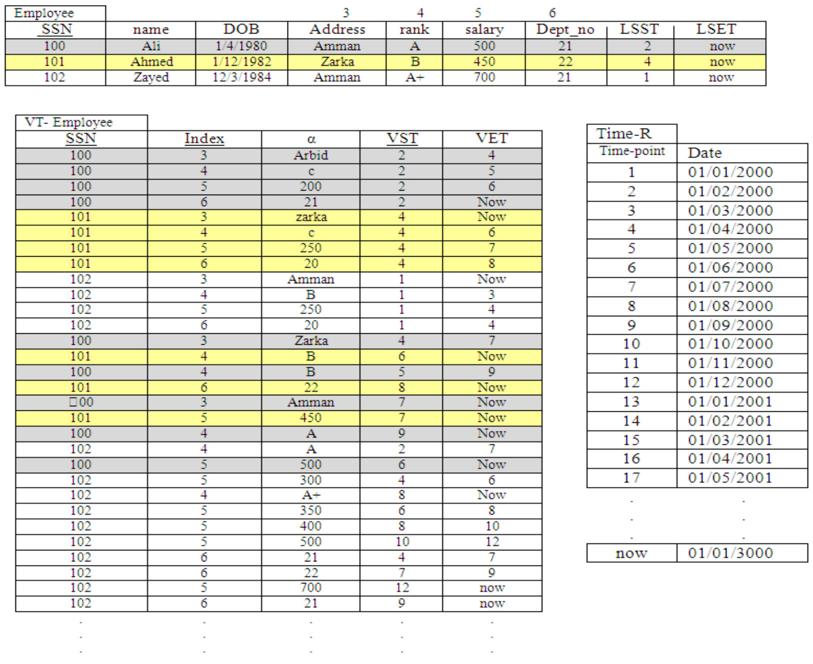
$S \bowtie U_t\ R = C\ (\Pi_{S,\ R,\ S.VST\ -1\ ,\ \min(S.VET\ ,R.VET)\ -1}$ and
$_{S.common\text{-}att\ =\ R.common\text{-}att}\ \sigma_{overlap(S,\ R)}$ (SX R)

The Since join and Until join operators produce a temporal since and until join of the two relations respectively, these two new operators are temporal related operators and can only be applied to temporal databases, where both since and until join are similar to the temporal join except that when integrating time periods of each tuple being used by the since and until join, different time intervals are produced to those found when performing a temporal join, for further reading on these operators and other temporal relational algebra that are not discussed here see this publication (Patel, 2003).

RESULTS AND DISCUSSION

Temporal-SQL: Querying temporal databases which are modeled based on the proposed data model discussed in Halawani and Romema (2010) and implemented using standard SQL can be evaluated according to the supplying time tick to the query. discussed this issue by classifying temporal query into current query, sequenced query and non-sequenced query, we followed the same approach for modeling a technique to query temporal database.

Current queries: In current queries we are asking for the state of the database at the current time. This data might be the current state or the current valid of the database at the current time, these two queries might be the same or different according to the interested data we want to retrieve, if we ask for the data that represent the current state regardless of its validity, we can use these queries, where we project our query to the current state relation as in Q1 bellow. In this case, we do not need any temporal operator and no need to add pure-time(time-slice) condition in the query, since this relation holding the current state of the database, but if we ask for the current state of the database that is valid at the current time (because in valid time model we can store some fact that will be valid in the future and in our proposed temporal data model this fact is stored in the current state relation and the history of data validity stored in the auxiliary relation as introduced before) so if we are asking for the current valid data at the current time the query will be different from the previous one, where data can be gathered from a view that constructed by joining current state relation and the auxiliary historical relation.



Employee

			3	4	5	6		
SSN	name	DOB	Address	rank	salary	Dept_no	LSST	LSET
100	Ali	1/4/1980	Amman	A	500	21	2	now
101	Ahmed	1/12/1982	Zarka	B	450	22	4	now
102	Zayed	12/3/1984	Amman	A+	700	21	1	now

VT- Employee

SSN	Index	α	VST	VET
100	3	Arbid	2	4
100	4	c	2	5
100	5	200	2	6
100	6	21	2	Now
101	3	zarka	4	Now
101	4	c	4	6
101	5	250	4	7
101	6	20	4	8
102	3	Amman	1	Now
102	4	B	1	3
102	5	250	1	4
102	6	20	1	4
100	3	Zarka	4	7
101	4	B	6	Now
100	4	B	5	9
101	6	22	8	Now
□00	3	Amman	7	Now
101	5	450	7	Now
100	4	A	9	Now
102	4	A	2	7
100	5	500	6	Now
102	5	300	4	6
102	4	A+	8	Now
102	5	350	6	8
102	5	400	8	10
102	5	500	10	12
102	6	21	4	7
102	6	22	7	9
102	5	700	12	now
102	6	21	9	now
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Time-R

Time-point	Date
1	01/01/2000
2	01/02/2000
3	01/03/2000
4	01/04/2000
5	01/05/2000
6	01/06/2000
7	01/07/2000
8	01/08/2000
9	01/09/2000
10	01/10/2000
11	01/11/2000
12	01/12/2000
13	01/01/2001
14	01/02/2001
15	01/03/2001
16	01/04/2001
17	01/05/2001
.	.
.	.
.	.
now	01/01/3000

Fig. 2: Example relations for applying temporal on relational schema

```
function coalesce(R)

  R := R order by $a_1,\ldots,a_n$,start
  $R_c$ = empty
  Size = count(R)
  i = 1

  repeat
      $R_i = (a_1,\ldots,a_n)$ [start, end]
      while $R_{i+1} = (a'_1,\ldots,a'_n)$ [start', end'] and $a_1= a'_1,\ldots, a_n= a'_n$ and start'≤end+1
              end := max(end, end')
              i = i+1
       end_while
       insert $(a_1,\ldots,a_n)$ [start, end] into $R_c$
       i := i+1
  while i ≤ size

  return $R_c$
```

Fig. 3: Pseudo Algorithm for the coalesce function

Example: Using the temporal relations in Fig. 2 and 3 and querying the data of the employee using the concept of current query, we assume that we are in time tick 10 and we want to query the data of employee with ssn = 102.

Current state:

Q1

Select * from employee where ssn = 102;

The result of Q1:

```
Q1
SSN Name   DOB     Addess  Rank  Salary     Dept_no   LSST    LSET
102   Zayed           12/3/84      Amman     A+          700          21
        1        Now
```

We can see that this data is valid for all attributes of the employee at current time 10 as we assumed, except the data for salary, where the value of this field will be valid in the future at time tike 12 as shown in VT- Employee relation, but sometimes current state is the right answer when we do not have proactive valid-time data in the temporal data model. In all ways, we still need a mechanism to deal with this issue and to be in high level abstract of the end user, this can be accomplished be constructing a view that joins the current state relation with its auxiliary relation in order to retrieve the valid data where its validity overlap the current time, the view can be built as the following:

Current valid:

Q2:
Select: E.ssn, E.name, E.DOB, Address_VT.α , Rank_VT.α , salary_VT.α , dept_no_VT.α

From: employee E, VT_employee Address_VT , VT_employee Rank_VT :
VT_employee salary_VT, VT_employee dept_no_VT

Where E.ssn = 102 and E.ssn = Address_VT.ssn and Address_VT.index = 3 and
Current_date/time> = Address_VT.VST and Current_date/time < Address_VT.VET and
E.ssn = Rank_VT.ssn and Rank_VT.index = 4 and
Current_date/time>= Rank_VT.VST and Current_date/time < Rank_VT.VET and
E.ssn = Salary_VT.ssn and salary_VT.index = 5 and

Current_date/time >= Salary_VT.VST and Current_date/time < Salary_VT.VET and E.ssn = dept_no_VT.ssn and dept_no_VT.index = 6 and Current_date/time >= dept_no_VT.VST and Current_date/time < dept_no_VT.VET

The result of Q2:

Q2

SSN	Name	DOB	Addess	Rank	Salary	Dept_no	LSST	LSET
102	Zayed	12/3/1984	Amman	A+	500	21	1	Now

Sequenced queries: The current queries take time-varying tables and extract a state at the current point time or at particular point in time.

Once that state is available, it can be manipulated conventionally, but in sequenced queries we consider the result of the query as valid-time table, the query will be over one or more temporal table and produce temporal result in contrast to current query which return snapshot state. Selection, Projection, Sorting and union all queries on temporal table are sequenced queries but applying join operation on two valid-time tables in temporal fashion is more challenging, what is desired here is to combine the history from two tables termed as sequenced join. As an example we consider S to represent the historical data of salary and R to represent the historical data of address as in Fig. 4. To combine the history of salary and address of the employee at each point of time we use sequenced join concept that is discussed in, by using SQL, the query must do case analysis of how the period of validity of each row of S relation overlap the period of validity of each row of R relation. There are four possible cases; case 1 the period associated with S relation is contained in the period of R relation or vice versa for case 4, while case 2 and case 3 identify the overlap between the two periods as shown in Fig. 5. Implementing sequence join for the above example in standard SQL can be accomplished using code fragment 6. Where we use 'union all' operator of the four SQL statements that represent the four cases depicted in Fig. 5. By using SQL case expression, we can rewrite the sequence join using single statement as illustrated in fragment code 7. By using stored function we can rewrite the SQL in simplified version as shown in fragment code Fig. 8 and explained in (Zimanyi, 2006).

First case expression can be implemented as stored function start_instance and the second case expression can be implemented as end_instance function.

S			
SSN	salary	VST	VET
100	200	2	6
100	500	6	Now

R			
SSN	address	VST	VET
100	Arbid	2	4
100	Zarka'a	4	7
100	Amman	7	Now

S X_t R				
SSN	salary	address	VST	VET
100	200	Arbid	2	4
100	200	zarkaa	4	6
100	500	zarkaa	6	7
100	500	Aman	7	Now

Fig. 4: Temporal Product operator between S and R relations

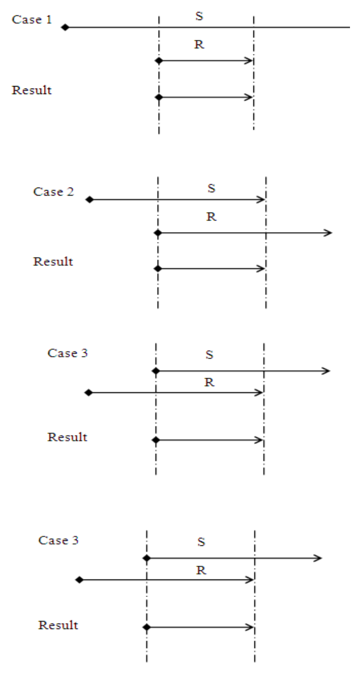
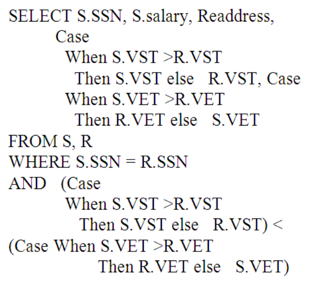


Fig. 5: Sequenced join cases

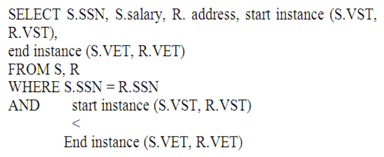
```
SELECT S.SSN, S.salary, Readdress, S.VST, S.VET
FROM S, R
WHERE S.SSN = R.SSN
AND R.VST<= S.VST
AND S.VET <= R.VET
UNION ALL
SELECT S.SSN, S.salary, Readdress, S.VST, R.VET
FROM S, R
WHERE S.SSN = R.SSN
AND S.VST >= R.VST
AND R.VET < S.VET
AND S.VST < R.VET
UNION ALL
SELECT S.SSN, S.salary, Readdress, R.VST, S.VET
FROM S, R
WHERE S.SSN = R.SSN
AND R.VST > S.VST AND S.VET <= R.VET AND R.
VST< S.VET
UNION ALL
SELECT S.SSN, S.salary, Readdress, R.VST, R.VET
FROM S, R
WHERE S.SSN = R.SSN
AND R.VST> S.VST
AND R.VET < S.VET
```

Fig. 6: SQL Fragment Code for Extracting the History of Employee's Salary and Address (sequenced join using union all operator)



```
SELECT S.SSN, S.salary, Readdress,
        Case
          When S.VST >R.VST
            Then S.VST else   R.VST, Case
          When S.VET >R.VET
            Then R.VET else   S.VET
FROM S, R
WHERE S.SSN = R.SSN
AND   (Case
          When S.VST >R.VST
             Then S.VST else   R.VST) <
(Case When S.VET >R.VET
                 Then R.VET else   S.VET)
```

Fig. 7: Fragment Code for Extracting the History of Employee, Salary and Address (Sequenced Join using Case Expression)



```
SELECT S.SSN, S.salary, R. address, start instance (S.VST,
R.VST),
end instance (S.VET, R.VET)
FROM S, R
WHERE S.SSN = R.SSN
AND       start instance (S.VST, R.VST)
               <
          End instance (S.VET, R.VET)
```

Fig. 8: SQL Fragment Code for Extracting the History of Employee, Salary and Address (Sequenced Join using Stored Functions)

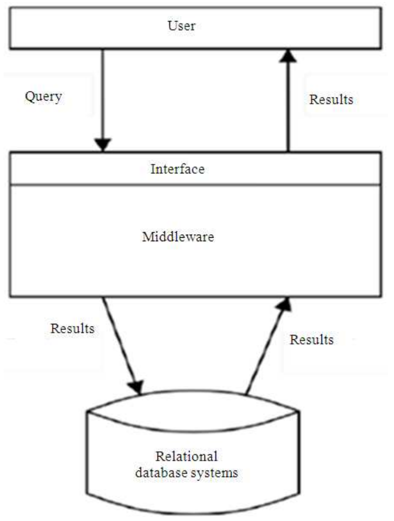


Fig. 9: Temporal Database Management System (TDBMS) (Patel, 2003)

Non-sequenced queries: We have seen current and sequenced queries, the non-sequenced query on temporal tables is straightforward, such queries ignore the time-varying nature of the tables, the phrases of the past, the present and at sometime indicate that the query is a non-sequenced one, some non-sequenced queries do examine the timestamps Fig. 6 and 7. Patel (2003) proposed a technique for developing temporal database on top of conventional relational database management system; the system architecture is in Fig. 9. The middleware is software that gets the query from the end user via the interface and translates the entire temporal query into conventional query to be passed to the conventional DBMS. The results from DBMS are translated into temporal results and send back to the end user. US logic have proven semantically to be popular operators for querying temporal database which are used in the language of first order temporal logic discussed in (Mariusz, 2007). However, they are evaluated all at once, rather than at a specific time (a time-slice query) or at each point of time (a sequenced query). A common example is determined when a change occurred by observing consecutive periods that signify that change.

```
     Create or replace function start_instance(one IN
number, two IN number):
 RETURN  NUMBER IS
 begin
  RETURN CASE WHEN one > two THEN one ELSE
two END;
  END start_instance
Create or replace function end_instance (one IN
number, two IN number)
 RETURN  NUMBER IS
 begin
  RETURN CASE WHEN one > two THEN two ELSE
one END;
  END end_instance;
```

CONCLUSION

We have proposed a technique for temporal database retrieval. This technique applies to temporal database applications that are modeled by TTHR, proposed in Halawani and Alromema (2010). This model is proposed based on the data models which are discussed in (Gregersen and Jensen, 1998; Ahn and Snodgrass, 1986). Our proposed data model is based on tuple time stamping with two relations, one relation is for the current snapshot data and the other one is the auxiliary relation that holds the temporal aspects of whole time-varying attributes, the proposed temporal data model achieves saving in memory usage range from 70-90% over the temporal data model discussed in (Novikov and Gorshkova, 2008), where a framework for temporal database implementation is discussed.

REFERENCES

Ahn, I. and R. Snodgrass, 1986. Performance evaluation of a temporal database management system. Proceedings of the ACM SIGMOD International Conference on Management of Data, (MD’86), ACM, New York, NY, USA., pp: 96-107. DOI: 10.1145/16894.16864

Elmasri, R. and S. Navathe, 2000. Fundamentals of Database Systems. 3rd Edn., Addison-Wesley, Reading, Mass ISBN: 0805317554, pp: 955.

Finger, M., 2000. A logical reconstruction of temporal databases. J. Logic Computat., 10: 847-876. DOI: 10.1093/logcom/10.6.847

Gregersen, H. and C. Jensen, 1998. Conceptual modeling of time-varying information. Heidi Gregersen, Christian S. Jensen.

Halawani, S.M. and N.A.A. Romema, 2010. Memory storage issues of temporal database applications on relational database management systems. J. Comp. Sci., 6: 296-304. DOI: 10.3844/jcssp.2010.296.304

Haraty, R.A. and N. Bekaii, 2006. Towards a Temporal Multilevel Secure Database (TMSDB). J. Comput. Sci., 2: 19-28. DOI: 10.3844/jcssp.2006.19.28

Kostenko, B., 2007. Temporal preprocessor: Towards temporal applications development. Proceedings of the Spring Young Researcher's Colloquium On Database and Information Systems, (SYRCDIS' 07), Moscow State University, Moscow, Russia, pp: 1-3.

Mariusz, G., 2007. Querying temporal database with the language of first-order temporal logic. Studies Logic Grammar Rhetoric, 11: 85-93.

Novikov, B.A. and E.A. Gorshkova, 2008. Temporal databases: From theory to applications. Programm. Comput. Software, 34: 1-6. DOI: 10.1134/S0361768808010015

Patel, J., 2003. Temporal Database System Individual Project. Department of Computing, Imperial College, University of London.

Torp, K., C.S. Jensena and R.T. Snodgrass, 1999. Effective timestamping in databases. VLDB J., 8: 267-288. DOI: 10.1007/s007780050008

Zimanyi, E., 2006. Temporal aggregates and temporal universal quantification in standard SQL. ACM SIGMOD Record. 35: 16-21. DOI: 10.1145/1147376.1147379