Functionalities and Performance of the Temporal Database

Deepak Kumar, Geetanshu Kansal
Amity Institute of Information Technology
Amity University
Noida, Uttar Pradesh, India

Abstract–All the variable recorded information in the form of raw data which further manipulated and defined in such a way which is meaningful and correlated. These information were recorded at some point in time which may or may not acknowledge the importance of the time at which it has been initiated, processed and terminated. There are uncountable number of databases like medical histories, banking transactions, employee’s database, etc., which are being maintained ever since the mankind evolved and started to keep the record of such information. Time is the key factor to any database created irrespective of its attributes and the values that are being stored in it. This article is consist of the details about the temporal database (TD) and the concepts that shows the functionalities and the importance of the TD in the real world. This article also explains that how the time can further be dissected into smaller versions or granules which has its own significance. Therefore, the main focus is to explore the significance of the TD considering different examples and scenarios.

Keywords: Temporal Database, Bitemporal Relations, TimeER, Application-Time Period Tables, System-Versioned Tables

INTRODUCTION

The database that records the information in regards with change of time, also stated as variations in time when they are recorded, is known as TD.\textsuperscript{[1][3]} As all the information that is recorded was done at some point in time which may or may not be considered as important to record. It makes almost all the applications are similar in nature and tend to be temporal. Some of the examples are: Book-Keeping (Tally) applications for accounting in financial sector; maintaining records for medical-history about patients in a hospitals; reservation systems for tour and travelling agencies, and government authorities for trains, airplanes, buses, hotels etc.; scientific applications like weather-forecast department and/or for astronauts.\textsuperscript{[2][11]}

Conventional database consists tables to store and process the information which refers to be valid in existing data. In this, the object is represented by a tuple and does not reflects any changes to the attributes. Thus, the information stored here can be from the past, present and related to the future with the temporal data types.\textsuperscript{[1][5]} The attributes related to time generally have an impact on the object model. This modelling technique consists properties of the object which distinguishes its uniqueness and differentiate among other similar objects. Therefore, TDB prevents the loss of consistency of the data and maintains the several records occurred in the form of transactions at different instances of the same object.\textsuperscript{[10]}

Such transactions occurring on the same data objects simultaneously tends to conflict among themselves if at least one of them intends to perform ‘write’ operation on the determined data. These transactions have different results, therefore, Concurrency Control (CC) components must interleave operations and execute transactions to reflect results as they occurred in a serial by achieving parallelism. All the transactions goes through three different phases, where, the first is the ‘read phase’. It contains the time when the data is requested from the database and the time when it copying on to the local space where it is inaccessible by the other transactions. The ‘validation time’ is the second phase when auditing of conflicts done by optimization method. At the end, ‘write phase’ comes when manipulation of the data take place on the database.\textsuperscript{[11]}

REVIEW OF LITERATURE

In the study of “Temporal Data Modelling” by Anton Lieskovsky et.al [1], the TD concepts allows to limit the validity of time attributes in the database by adding time duration as an additional information. This dignifies the importance of every tuple consisting similar attributes at various instance. It also states that the same object cannot
occur at the same time and manages the information about complete cycle of the object, ever after its removal from the database.

In the study of “Temporal Data Management (TDM)” by Christian S. Jensen et.al [2], for the better understanding of important concepts of TDM, they summarized the current state-of-art and the challenges. The focus was also brought on the followings: semantics of temporal relational schema (TRS) along with their logical design, acceptance of BCDM as an appropriate model, proposal of different languages TDB queries, time-stamps issues of its representation was resolved, etc. The TSQL2 query language was consolidated and later, it was used in in and as new part of SQL3/Temporal.

As per the study of “A Glossary of TDC” by C. S. Jensen et.al [3], concepts of TDB were named and well defined. It also listed the alternatives along with their respective advantages and disadvantages. An example of an E-mail discussion was presented to explain the theory and justify the implementation in its proper design.

As per the study of “A Consensus Glossary of TDC” which was another paper published by C. S. Jensen et.al [4], two different sets of criteria were proposed to the implementation of TDB and its concepts. First was required to satisfy its relevance in four aspects and the second was to provide the explanation to the naming of its concepts to evaluate the criteria. There were three different concepts: General, Temporal and the Specialized Database interests.

In the study of “Memory Storage Issues if TD Applications on Relational Database Management Systems” by Sami M. Halawani et.al [5], Tuple time stamping TDM performs better and more efficiently which involves time-varying attributes of multiple relations in terms of space used in the memory and the process time. There are two different relations: first is for the snapshot of the current data and the other which holds the aspects of time-varying attributes in auxiliary state. It saves the memory space of about 70-90% over the TDM, where this framework is implemented.

As per the study of “A Logical Temporal Relational Data Model” by NadeemMahmood et.al [6], TDM is an operation to apprehend the nature of entities of changing time; to design temporal query language to process the repossession of temporal data after manipulation. Proposed temporal model is not restricted to remove or lessen the number of temporal attributes from the DB, however, it maintains the records in a relation and prevents unexpected flaws. These flaws can also be identified as redundancy of database and can generate inconsistency because of the presence of excess of temporal attributes.

In the study of “Effective Time TDM” by DimitarPilev et.al [7], data marking effective time (ET) was proposed to model the data field which is simultaneously valid (when the data is in the tuple marked ET) and recorded in the relation. The algorithms were implemented on the university information providing web-based system using ETM.

In the study of “Extraction of a Temporal Conceptual Model from a Relational Database” by Q. Hoang et.al [8], the extraction method was proposed for the TimeER Model based on the set of attributes with their characteristics like, primary key, set of foreign keys of the relational schema in the TRM. This method of extraction was acceptable as any input of RDB, there is a corresponding DB in the TimeER Model.

In the study of “Temporal Features in SQL:2011” by Krishna Kulkarni et.al [9], the approach taken by SQL:2011 in TDB was compared and discussed. The rows and the user-defined columns’ period information of the temporal tables can be accessed by including the start and the end period in the select list if query. It also provides the syntactic extensions set with the specified semantics and its scope. In SQL:2011, statements like insert/update/delete, query expression and constraint definitions are operated by following the semantics. It allows users to specify any start and end column on ATP they desire, where, it does not require any new syntax.

In the study of “Bipolar Fuzzy Querying of TDs” by Christopher Billiet et.al [10], bipolarity was introduced in temporal query specification. The focus was on the representation of valid time in bipolar manner on historical database for a particular interest. Here, the time is imprecise and unknown, therefore, applying bipolar techniques to
such TDBs helps in improvising the research of such historical information with respect to their validity period.

In the study of “A New Concurrency Control Algorithm in TDB” by MirsaeidHosseiniShirvani et.al [11], a new algorithm of an Optimistic Concurrency Control (CC) procedure was applied which proclaims to be serializable, validated, consistent, and detects conflicts in phase of validation and unlock the resources to release by using EOT marker techniques. The time intervals are considered as granules than tuples to enhance the parallelism.

FUNCTIONALITY OF TEMPORAL DATABASE

I. Valid Time

The Valid time is described as a collection of time periods including the past (which is already happened), present (what is happening currently) and/or the future (which may occur in due course) in the modeled reality. There could be uncountable number of facts associated in the form of events and intervals, where the single case is considered as a special and/or important with respect to the need in fulfilment of the required information. Thus, it captures the states of variations in time of the mini-world. Therefore, it is universally said that all the facts are consist of valid time. However, it does not state that it should be recorded in any type of database due to many unknown reasons which restricted the necessity of notation of that time period. Following example can be taken to justify the above statement that, the valid time may be unknown and or the recorded time might be irrelevant to the database of the concerned application. As, there could be different worlds who also has different possible times respectively and each would note/record the different possible valid time, which is different from others’.

The term mini-world was acknowledged as a part of reality for a database to model and record different information. These features of the mini-world are termed as entities for representation in the database for various structures. Another term “fact” was acknowledged for any statement that is assigned a truth value on the basis of their meaning relating to the information, i.e., either true or false. [1][2][3][4][5]

II. Transaction Time

A database fact is stored when the fact is current in order to retrieve at some point in time, once it is stored in a database. Therefore, at the point when a fact is stored in the database is said to be the transition time. Beside facts, it can also be associated with any database’s entity, which makes it different from valid time. Transaction times are consistent as the values can never be recorded after the current time. The recorded time, therefore, serialize the order of the transactions stored in the database. Also, the transaction time cannot be changed once it gets recorded into the database as we cannot change the past. It means, once the transaction is recorded, it remains the same until and unless it is completely removed/deleted from the database. Every database has a transaction time aspect, which may or may not be seen in the database. The visualization of such transaction time entirely depends on the discretion of the design of the database by its designer. These transactions can also be associates of objects and values, which certainly are not the facts. It also has a duration which begins at insertion and end at (logical) deletion [1,2,3,4,5].

III. User-defined Time

As the name suggests, it is a time stated by the user for his or her self-created objects and/or the values used as: Birth date of an employee, hiring date of an employee, date of promotion, termination date, etc. in an organization’s database which maintains the record of each and every employee working in that organization. It can also be used for integer values like monetary information. Salary of an employee, payments, receipts, deposits, withdrawals, loss incurred, etc. are some of the examples of such attributes. [4][5]

IV. Valid-Time Relation

It is a relation (record-table) with exactly one system-supported valid time. With reference to the definition of the valid time, there isn’t any restriction that how the tuples are associated with the valid time. The valid time cannot surpass the current time, the current time is considered as the end of that valid time if it is not specified for any past instance. For example, if we look at the database of the employees of a company, there are number of existing and non-existing employees. The employees who are not the part of the company
has specified valid end time and the employees who are still working has current time as their valid end time. Therefore, in a relation, it varies with different tuples according to their data. [3][4][5]

V. Transaction-time Relation

It is a relation with exactly one system-supported transaction time. As for the valid-time relations, there are no restrictions on how transaction times may be associated with the different tuples on basis of their data stored in that particular relation. In some of the databases, the changes are often made. Therefore, the timestamp (insert, delete or update) of every transaction is recorded that has applied a change simultaneously to the database. The most popular and useful example of such database is banking transaction. In banks, the data is often updated with every transaction done on an account with respect to deposits, withdrawals, payments, receipts, taxes, profits, charges, rollback, etc. [3][4][5]

VI. Bitemporal Relations

This relation is said to be bitemporal as some of the applications require both valid and transaction time in same database. For example, if we consider employees and the department of any company name ‘x’. If an employee from department ‘a’ has been working there for 3 years with salary 30000 has current time as its valid time. The new transaction is made that updates the data of the same tuple which states that the employee is promoted. No the department is changed from ‘a’ to ‘b’ and the salary is update to 35000 from 30000. Therefore, both the date were true, before and after the transaction took place and the valid time changes. For department ‘a’, the new valid end time would be (let suppose) 31/03/2016 for that employee and the valid start time for department ‘b’ would be 01/04/2016 and valid end time would be the current time. Therefore, no attribute was changes in the relation, physically. However, the value of Departments’ valid end time of that employee has changed [3,4,5].

VII. Chronon

A Chronon is termed as a shortest time duration which is non-decomposable in nature (which cannot be further divided or broken into smaller part) is an atomic unit of time. The discrete time model which has types of chronon like valid-time, transaction-time and bitemporalchronons is considered to represent TDB for simple and relative ease of implementation [3,4,5].

VIII. Timestamp

It is a value of time associated with some object. As previously discussed, object is a tuple present in a specific relation defining the object without changing the time-varying attributes along with non-temporal attributes. This concept can be dedicated to valid timestamp, transaction timestamp, bitemporal timestamp, etc[3,4,5].

IMPLEMENTATION OF TEMPORAL DATABASE

In this study, the focus is brought onto the working of the functionalities of the TDB. The major concern have been the time periods in various states with respect to the occurrence of a transaction on an object of a relation. In the following examples, the working and its implementation are discussed in different circumstances.

1. The temporal characteristics of the entities in a relation can either be transaction time (TT), Lifespan (LS), or both. Similarly, the attributes of such entity can either be Valid-time (VT), Transaction-time (TT), or both. Therefore, the design of the relation is defined on the basis of temporal aspects supported which can be seen as an entity type or the attributes.

Components of TimeER model

Entity Type – the representation is done by a rectangle where weak entity is represented by double rectangle. The captured Lifespan or transaction-time are indicated by placing LS, TT, or LT with respect to the requirement of the situation.

Attributes – There are different types of attributes are represented by different symbols. Single-valued attribute is symbolized by oval, while a multi-valued is symbolized by a double oval. Composite attribute is represented by oval directly connected to the ovals for each value of components. Unlike non-temporal attributes, temporal-attribute’s values may vary time to time depending on the type of transaction takes place as per the situation.

Relation – A relation between two entities are represented by a diamond. The designer of the database decides whether the relation is supposed to apprehend the temporal characteristics depending
upon the requirement and the relationship formed between the two.

Following is the example of TimeER Diagram of employees of a company which represents the relation among entities with respect to the non-temporal and temporal aspects supported in the database. [8]

Mapping algorithm from TimeER to Relation Model [8]

Step 1. Entity types are mapped which are not partaking in a superclass/subclass relationship
Step 2. Entity types which are partaking in a superclass/subclass relationship are mapped
Step 3. Mapping of entity’s temporal single-valued attributes
Step 4. Mapping of multi-valued attributes
Step 5. Mapping of Non-temporal relationship types
Step 6. Mapping of temporal binary relationship types which does not have any attribute
Step 7. Mapping of temporal binary relationship types along with their attributes

The literature of TDB is recognized by two dimensions of time for temporal data support, i.e. valid-time and transaction-time (as already discussed in previous section). In SQL:2011, the valid-time support and transaction-time support are provided by Application-time Period (ATP) tables and System-Versioned tables, respectively. ATP’s name can be anything defined by a user, whereas, the name of System-Versioned period is itemized as SYSTEM-TIME as per the standards.

ATP Tables: The tables that contain the definition of period with the name provided by user (user-defined).

For example:

CREATE TABLE Patient(
PNO INTEGER, PStart DATE, PEndDate DATE, PDept STRING, PERIOD FOR Period (PStart, PEnd)
)

With reference to the above example, the user have the liberty to use any name for the period and even the names of attributes that act as start and end period. For both the columns, the data_type must be same which either can be DATE or a timestamp type.

The conventional INSERT statement gives the initial values to support the ATP start and end columns. For Example:

INSERT INTO Patient VALUES (101121, '2016-04-03', '2016-04-09', 'CARD1')

Result:

<table>
<thead>
<tr>
<th>PNo</th>
<th>PStart</th>
<th>PEnd</th>
<th>PDept</th>
</tr>
</thead>
<tbody>
<tr>
<td>101121</td>
<td>2016-04-03</td>
<td>2016-04-09</td>
<td>CARD1</td>
</tr>
</tbody>
</table>

The conventional UPDATE statement is used to modify the tuples of ATP tables including the start and end time of ATP. The DELETE statement is used to remove a tuple from the ATP table. SQL:2011 has the ability to specify alterations. It is delivered by a syntactic extension to specify the period of interest by UPDATE and DELETE statements. For example:

UPDATE Patient FOR PORTION OF PPeriod FROM DATE '2016-04-05' TO DATE '2016-04-07' SET PDept = 'NEURO4' WHERE PNo = 101121

Result:

<table>
<thead>
<tr>
<th>PNo</th>
<th>PStart</th>
<th>PEnd</th>
<th>PDept</th>
</tr>
</thead>
<tbody>
<tr>
<td>101121</td>
<td>2016-04-05</td>
<td>2016-04-07</td>
<td>NEURO4</td>
</tr>
<tr>
<td>101121</td>
<td>2016-04-07</td>
<td>2016-04-09</td>
<td>CARD1</td>
</tr>
</tbody>
</table>

Note that, ATP of the above tuple extends beyond P at both ends which then provide the above mentioned result. The above tuple is considered to be the original as the PDept is CARD1 and then UPDATE trigger fires and change it into second tuple whose PDept is NEURO4. Similarly, third
tuple is also updated where INSERT trigger fires for the second and third tuples.

DELETE Patient FOR PORTION OF PPeriod FROM DATE ’2016-04-05’ TO ‘2016-04-07’ WHERE PNo = 101121

Result:

<table>
<thead>
<tr>
<th>PNo</th>
<th>PStart</th>
<th>PEnd</th>
<th>PDept</th>
</tr>
</thead>
<tbody>
<tr>
<td>101121</td>
<td>2016-04-03</td>
<td>2016-04-05</td>
<td>CARD1</td>
</tr>
<tr>
<td>101121</td>
<td>2016-04-07</td>
<td>2016-04-09</td>
<td>CARD1</td>
</tr>
</tbody>
</table>

Here, the result is deletion of the original tuple where DELETE trigger fires for tuple where PDept = NEURO4 and INSERT trigger fires for the other rest.

System-Versioned Tables: The importance of such application is to preserve the older version of the tuple before performing and UPDATE or DELETE statement. Beside this, another obligation is that the system preserves the start and the end of the period for every tuple in the table and prevents user from modification of any content of historical data along with its associates. Any changes/updates need to be performed, can only be done by the system that too only on the non-temporal attributes of that table. It ensures that the historical date cannot be hampered, which is precarious in nature to meet auditing and compliance regulations. For example:

CREATE TABLE Emp(ENo INTEGER, Sys_start TIMESTAMP(12) GENERATED ALWAYS AS ROW START, Sys_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END, EName VARCHAR(30), PERIOD FOR SYSTEM_TIME (Sys_start, Sys_end) ) WITH SYSTEM VERSIONING

As we have discussed, System-versioned tables are different from ATP tables as users cannot define, assign or alter the values of Sys_start and/or Sys_end columns. Therefore, the definition of Sys_start and Sys_end must include the keyword GENERATED ALWAYS.

In System-versioned table, INSERT INTO spontaneously arrange the value of Sys_start to the transaction timestamp and arrange the value of Sys_end to the uppermost value of the column’s data_type. Let’s consider a following example for the better explanation:

INSERT INTO Emp (ENo, EName) VALUES (11011, ‘Joseph’)

Result:

<table>
<thead>
<tr>
<th>ENo</th>
<th>Sys_start</th>
<th>Sys_end</th>
<th>EName</th>
</tr>
</thead>
<tbody>
<tr>
<td>11011</td>
<td>2016-01-01 09:00:00</td>
<td>9999-12-21 23:59:59</td>
<td>Joseph</td>
</tr>
</tbody>
</table>

Unlike ATP tables, on System-versioned tables only current system tuples can perform UPDATE and DELETE, whereas the historical system tuple cannot be amended by the user. User is also not allowed to modify the start or end period for both current and historical system tuple. Both the statement also result in auto insertion of historical system tuple for current system tuple that is updated or deleted.

Following is the UPDATE statement to change the name of employee from Joseph to David whose ENo is 11011 effective from the transaction timestamp:

UPDATE Emp SET EName = ‘David’ WHERE ENo = 11011

Result:

<table>
<thead>
<tr>
<th>ENo</th>
<th>Sys_start</th>
<th>Sys_end</th>
<th>EName</th>
</tr>
</thead>
<tbody>
<tr>
<td>11011</td>
<td>2016-01-01 09:00:00</td>
<td>9999-12-21 23:59:59</td>
<td>Joseph</td>
</tr>
<tr>
<td>11011</td>
<td>2016-04-01 12:00:00</td>
<td>9999-12-21 23:59:59</td>
<td>David</td>
</tr>
</tbody>
</table>

In this example, there are no INSERT triggers fired and UPDATE trigger is fired for the tuple where the name is updated to DAVID. It also creates the historical system tuple sequence of update in a form of chain without any gaps.

Let’s consider the first table formed where the name of the employee was Joseph. If we perform DELETE statement on it, the execution will be as follows:

DELETE FROM Emp WHERE ENo = 11011

(Let the current time be 2016-03-25 14:00:00)

Result:

<table>
<thead>
<tr>
<th>ENo</th>
<th>Sys_start</th>
<th>Sys_end</th>
<th>EName</th>
</tr>
</thead>
<tbody>
<tr>
<td>11011</td>
<td>2016-01-01 09:00:00</td>
<td>2016-03-25 14:00:00</td>
<td>Joseph</td>
</tr>
</tbody>
</table>
3. Consider a car rental services like “Zoomcar” which contains various properties of cars including colors, types of car, rate (with respect to Kilometers, per hour, and/or per day). Beside the previously mentioned properties, the major and most important property is the starting and the ending time/day of a car rental. The charges are levied as per the total time period calculated. If such attributes values are not inserted into the database of the company by its employees, it may lead to enter several records in multiple tuples of a relation of the same object (which is a particular car rented to one client). These tuple entries would conflict in the relation, each containing different starting and ending rental day/time.

This would reflect as same car is rented by diverse patrons at the same time period, which physically and logically not possible to be practiced. This would result in redundant and inconsistent data. Therefore, TDB implicitly prevents such situations to occur.

Considering the above mentioned example, the client may have its preference of, “How old the car should be, color of the car?” Such preferences can be both negative and positive with respect to the availability of cars. These properties or attributes of the relation decides the satisfactory level of the client depending upon the situation. This further introduces the technique of Bipolar Querying, which entertains both positive and negative preferences of the clients in a relation.

The criteria of satisfaction is modeled by degree of satisfaction ‘s’ ∈ [0, 1], where ‘0’ states the value to be false, which means the client is not equally satisfied if the preferences raised by him/her are not matched. On the other hand, ‘1’ states the value to be true which means the client’s preferences are matched with availability if the car he/she demands.

If, the degree in which the record entered in a tuple of a relation is unsatisfactory compared to the criteria provided by the client, is exactly “inverse of the satisfaction degree, that is, d = 1-s, where d ∈ [0,1].”[10]

Definition: A Bipolar satisfaction degree (BSD) is a pair of

\[(s, d), s, d \in [0, 1]\]

where ‘s’ the satisfaction degree and ‘d’ the dissatisfaction degree.

**CONCLUSION**

In this study, the concentration has been on, Temporal Database and its functionalities. We have shown the performance after implementing temporal database in various scenarios. We have also focused on the importance of Temporal Database and its in day to day business and/or human life. Although we have been working with the limited resources and time constraints, we have highlighted some major terminologies, whereas quite a few have been missed. However, the relative examples and three different scenarios have been explained to provide a broader view over the subject.

**REFERENCES**

[1] Kvet Michal, Lieskovsky A., and Matiasko K., Department of Informatics, Faculty of Management Sciences and Informatics, University of Zilina, Zilina Slovakia, Temporal Data Modelling, 2016


Conference Proceeding of
International Conference on "Latest Concepts in Science, Technology and Management (ICLCSTM-16) at The Institutions of Electronic and Telecommunication Engineers (IETE), Institutional Area, Lodhi Road, New Delhi, India on 19th June 2016
ISBN: 978-81932712-4-7


