Decision Time in Temporal Databases

Mario A. Nascimento[*] and **Margaret H. Eich**
Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas, 75275-0122, USA
{mario, eich}@seas.smu.edu

Abstract

Most of the research found in the Temporal Database literature assumes the use of valid time and transaction time as temporal attributes supported by the DBMS. Some also assume a user-defined time as a temporal attribute which is not supported by the DBMS. We discuss the reasons for the need of such support, in the context of the relational data model. We argue that decision time needs to be supported by the DBMS and we elaborate on a data structure that encompasses both decision time and transaction time. Examples are provided to illustrate our arguments. Finally, based on the three temporal dimensions a new taxonomy for database operations is sketched.

1 Introduction

The need of temporal support in databases has been well discussed [3, 6, 5], and most of the research has assumed the use of valid time and transaction time as those which need to be supported by the database management system (DBMS.) However, we believe that additional support from the DBMS is needed, namely for decision time.

The purpose of this paper is to discuss what is meant by DBMS support regarding temporal data and why, from the perspective of a relational DBMS, such support is necessary for better modelling of real world situations. In addition we discuss why valid time and transaction time are necessary but not sufficient. Hence we introduce the notion of decision time. To support our arguments we use as an example an evolving relation, where to preserve the history associated with one (or more) tuple(s) traditional attributes is (are) not sufficient. We shall see that the need for temporal attributes arise naturally. We also show how such temporal attributes are treated by the DBMS.

This paper is divided as follows. In the remainder of this section we present our assumptions and a glossary of terms we will be using throughout the paper. In Section 2 we discuss when and why DBMS support is needed for valid and transaction times. Additionally we elaborate on the concept of a key throughout that section, basing our discussion on examples. In Section 3 we present one example where DBMS support for decision time is needed, otherwise there is loss of information. This, along with some ideas for the corresponding data structure, presents our contribution towards time representation and reasoning within databases, more specifically decision time. Section 4 presents, rather briefly, a taxonomy for classification of operations in a database, regarding the three discussed temporal dimensions. Finally, in Section 5, we present a summary which encompasses conclusions and outlines future directions.

1.1 Framework and Definitions

We assume the user has available a traditional DBMS. By traditional we mean that at least the following capabilities are present: (*a*) The user is able to create a relation by defining its attributes and defining one (or more) of those attributes as a key for that relation. (*b*) The user has available a query language (perhaps an SQL-dialect [1, Chapter 7]) which allows him/her to retrieve and update the contents of a relation. As we will be discussing several time dimensions shortly, we shall make clear what we mean by each of them. Where possible we make use of a terminology which appears to be widely accepted in the area [2]. It is worthwhile noting that in [2] the notion of decision time is not mentioned at all.

- Object: Any data element in the relational DBMS, for instance, a tuple in a given relation.
- Event: The action an object suffers, being either its creation (and insertion into the database) or an update (deletion included.)
- Valid Time: This is the time range when the object is true in the real world, and includes

[*] On leave of absence from EMBRAPA - CNPTIA, Campinas, Brazil, and supported by CNPq (Process 260088/92.7), Brasilia, Brazil.

start and end time, being thus, a time range. We denote valid time by $V = [V_s, V_e]$.

- Transaction Time: This is the time the object is recorded (which may be an update, insertion or deletion) into the database. The transaction time is punctual, in contrast to valid time. We denote transaction time by T.

- User-defined Time: A temporal attribute which is not valid time nor the transaction time. More than one user-defined temporal attribute may be defined in a relation, however we assume, with no loss in our arguments that only one user-defined time exists per relation. It is treated in the same way as any other non–temporal attribute in a relation. User-defined time is denoted by U.

- Decision Time: This is the time that an event was decided to happen. As transaction time it is punctual and it is denoted by D.

In what follows we argue that without DBMS support for valid and transaction time temporal data cannot be properly handled. We also show that there are situations where the non–existence of another temporal attribute, namely decision time, may cause loss of information. Therefore we conclude that decision time must be supported in the same way that valid time and transaction time are.

Indeed, the reader may argue that most of the problems we are going to point out may be overcome by adding proper user-defined temporal attributes (which do not require DBMS support.) However this is not a good option for the following reasons: (i) the user has to define all temporal attributes beforehand, possibly even before he/she knows what sort of temporal support is needed, or if it is needed at all, otherwise schema modification is required; (ii) the user is responsible for maintaining the relation keys. As we shall see those temporal attributes are to be part of the key, which then makes (i) even more difficult to handle. Therefore we believe that DBMS support for temporal data is very much desired and our discussion is grounded upon such a belief.

As a framework for our discussion we use the `Job` relation to be introduced below and evolve it, i.e. make a series of operations on it as time evolves. To illustrate the concepts that follow, consider the following scenario. There is a relation `Job` which has one attribute called `Position` and another one called `Id` which happens to be the key for this relation. In the following tables the name of a key attribute is shown in boldface. Without loss of generality we look, in the examples given hereafter, at time as a time-line made up of integer values equally spaced.

2 Support for Valid and Transaction Time

Suppose we have one instance of the `Job` relation as presented in Table 1 at time $t = 3$. What in fact we most likely visualize, semantically speaking, from that table is the information in Table 2. Thus, all one can say, while still relying on Table 1, is that either Mary's and John's tuples have $V_s \leq 3$ and $V_e = NOW$. We use NOW to denote the current time, thus it is a dynamic value.

Now suppose that at time 5 one wants to update the attribute `Position` of Mary's tuple in such a way that it reflects the information that from now on Mary is to be a Manager. In this case a standard DBMS would take Mary's tuple to a state very similar to the one shown in Table 1, namely the only difference would be Mary's `Position` attribute, which would be Manager, instead of Clerk. Obviously we lost information, as we are not able to see that Mary had been a Clerk in the past.

Now, let us assume that the DBMS is able to (somehow) perform the following. Once an event occurs on an object it has a valid time range associated to it. If such range is not specified we assume it defaults to $[t, NOW]$, where t is the time of the update. Also, the DBMS is able to record this information in the database as part of the object's attributes. Then, the user would be able to access the relation as it is shown in Table 3. Note that the key did change and we shall discuss this in the following.

As history builds up on a tuple, the original key is not valid any longer. By looking at Table 3 we can see that for a single `Id`, Mary, there are two `Positions`. This calls for a change in the way we see keys in a temporal relation. When history of a tuple is kept it is obvious that each key will be associated not only to a single tuple but to all versions of such tuple. Therefore the new actual key is the union of the previous key(s) with the valid time, specifically only V_s is needed as part of the key. However, this composite key may be neglected by the user. When a valid time is not specified either in a query or update request, the latest version of a tuple is taken into account. Nevertheless the query language should, somehow, also allow access to all versions associated with a given key.

Thus valid time is needed to preserve the history of a tuple. In fact this is the definition of a historical database [5]. Should valid time not be supported by the DBMS, a query such as "Has Mary ever been a Manager before" or "What was Mary's position at time 3" would not be answered, instead only queries such as "Is Mary a Manager" (which implicitly are concerned with the current status of the database) would be possibly answered.

We are still not able to have overlapping valid times for a given tuple. For example, consider if Mary had her position changed from Clerk to Assistant-

Id	`Position`
Mary	Clerk
John	Clerk

Table 1: Snapshot of the `Job` relation at time 3

Id	`Position`	V_s	V_e
Mary	Clerk	3-	NOW
John	Clerk	3-	NOW

Table 2: Implicit information of the `Job` relation at time 3

Manager at time 2, but it is decided at time 5 to be a retroactive promotion from time 2 until time 4. Suppose however that, for some reason, such a transaction committed only at time $T = 7$. The new table is depicted in Table 4. At the same time range [2, 4] Mary would have two Positions. Hence the query "What was Mary's position at time 3" would have two answers, when just one is being sought for. Depending on which snapshot of the database one is looking at there is a different answer for such query.

The answer depends on where the query is positioned in the time-line, i.e., if the query is posed with respect to the knowledge the database had from the $t = 0$ up to (but not including) time 7 (when Mary's promotion was committed into the database) then the answer would be Clerk, otherwise the answer would be Assistant Manager. Therefore the answer depends on whether the update transaction had been committed (recorded in the database) or not. Hence transaction time is also needed to be supported by the DBMS.

Considering this, assume that in addition to attaching valid time to each object once it is updated, the DBMS is also capable of recording the time when the associated transaction committed. Thus, the DBMS would be providing support for transaction time as well. We can then refine the default for the valid time: unless it is defined otherwise, we set $V_s = T$ and $V_e = NOW$, where T is the actual transaction's commit time. In this case the relation `Job` (as seen at $t = 7$) would be, semantically, as shown in Table 5.

We have just introduced the notion of a temporal database as discussed in [5], which is a database providing support for both transaction time and valid time.

Notice that the original key plus the valid time may be not enough to differentiate versions of a tuple. Thus we need to expand on the key again and make it the original key plus valid time plus transaction time, in order to have a proper key for the extended relation.

As we did previously we assume that if no information is given about the query's position in the time-line, its position is the current time, hence the most recent information is to be used.

3 Support for Decision Time

We have discussed why valid time and transaction time is necessary for proper database modeling of reality. Recall that we assume that the use of user-defined temporal attributes for valid time and decision time handling is not an option. If decision time is not supported by the DBMS, modelling the real world cannot be accomplished satisfactorily. We still use the `Job` relation to illustrate our arguments.

As we saw in the previous section queries are to be based on valid and transaction time. However the decision to change Mary's position had been made some time before it was actually committed into the database. Nevertheless queries would be (potentially) misled by the transaction time as it would act as the actual decision time.

Consider Mary's promotion and the fact that although it was decided at time 5, it was actually committed at time 7. In this particular case could Mary be considered a Manager at time 6, considering (*i*) the state of the relation `Job` as of time 5 (or 6) and (*ii*) the current time to be $t \geq 7$. The answer would be negative. Even though at time $t \geq 7$ the transaction had been committed, the `Job` relation as of time 5 (or 6) did not have "officially", so to speak, the necessary knowledge. Hence the incorrect negative answer. Furthermore, as there is no support for decision time there is no way to know when the decision was taken. All we are able to know is that such decision was committed into the database at $t = 7$.

We propose that decision time should be another temporal attribute, i.e., whenever one event happens on one object it must have a decision time associated to it (it defaults to the transaction time if none is assigned.) It would work very much in the same way as valid time, which can also be assigned or left to a default as discussed before, with the exception that valid time is a range and decision time is punctual as

Id	Position	$\mathbf{V_s}$	V_e
Mary	Clerk	0	4
Mary	Manager	5	NOW
John	Clerk	0	NOW

Table 3: Information of the `Job` relation at $t = 5$, using valid time

Id	Position	$\mathbf{V_s}$	V_e
Mary	Clerk	0	4
Mary	Assistant Manager	2	4
Mary	Manager	5	NOW
John	Clerk	0	NOW

Table 4: Two valid times of different tuple versions overlap

transaction time.

Therefore once a decision time is assigned, the DBMS is responsible for recording such information in the database, if none is assigned then it defaults to transaction time. This would lead Table 5 to the semantically richer Table 6. The question is how this is to be accomplished. Although we have not addressed the issue here thus far it is obvious that both valid and transaction times need to be physically recorded and indexed by the DBMS. Suppose that such indices are I_v and I_t. Then we can design the DBMS to handle relations with valid, transaction and decision times, and thus maintain I_v, I_t and a new structure, I_d, the index for decision time.

We propose the following idea to avoid most of the overhead due to the new additional index I_d: to incorporate the decision time inside the transaction time index I_t. Using I_v instead is not a good idea because, in general, it indexes a time range, whereas both I_t and I_d indexes time points. Once a transaction is committed I_v and I_t are updated as before, but in addition the decision time is recorded in I_t as well. Let us call this combined index I_{td}. The difference between the former I_t and the new I_{td} has to do only with the leaf nodes. Notice that we use the terminology assuming a B^+–tree index [1, Chapter 5]. If only a single data type is indexed, as in the I_t for instance, the leaf nodes look like the ones depicted in Figure 1(a). In the I_{td} each leaf node indexing time point t_k has two types of pointers. The first type is the set SP_d where each pointer points to a record that was decided upon at time t_k. The second type is a linked list of pointers P_t where each one points to a record committed at time t_k. Furthermore the leaves are connected by a doubly–linked list. This is shown in Figure 1(b). Note that pointers associated to a decision at time t_d always point to a event committed at time t_t, where $t_d \leq t_t$. Also both types of pointers, SP_d and P_t may be $NULL$ at any of the leaves.

With this structure, and properly manipulation of the new composite index, we are able to know that at a given time, previous to the actual transaction time, decisions may have already been made and thus such knowledge may be retrieved. Also as we made use of an already existing structure, namely I_t, we reduced the overhead in space of a new indexing structure. Therefore, we can now retrieve the fact that at time $t = 5$ Mary was a manager, if such query is posed at $t \geq 7$, and this was done without modifying the `Job` relation schema at all.

For the sake of fairness we should recognize that there is still one problem that this approach does not solve. Until the transaction is committed there is still no way to know whether decisions regarding the time frame previous to the commit on the database had been made or not. It is our opinion that this is a problem which cannot be solved, given that until a fact is recorded it is unknown. Nevertheless we were able to record a decision retroactively (in relation to the actual transaction time) without incurring additional structural overhead. Recall that before introducing decision time, as a temporal attribute, we could only assume that decision time was equal to transaction time, and thus even for $t \geq 7$ Mary would not be considered a Manager at time 5 (or 6.)

Finally, let us discuss the role of the key whenever decision time is taken into account. If we assume that one is not allowed to decide on the very same issue twice then decision time does not need to be part of the actual key. On the other hand if one is not allowed to decide on the same issue twice, we now have the possibility of using only the original key plus decision time as the actual key for a relation.

4 An Enhanced Temporal Taxonomy Classification

As we discussed before, most research have taken into account only valid time and transaction time, and thus a classification of operations would be reduced

Id	`Position`	$\mathbf{V_s}$	V_e	**T**
Mary	Clerk	0	4	0
Mary	Assistant Manager	2	4	7
Mary	Manager	5	NOW	5
John	Clerk	0	NOW	0

Table 5: Implicit information of the `Job` relation with transaction time

Id	`Position`	V_s	V_e	T	**D**
Mary	Clerk	0	4	0	0
Mary	Assistant Manager	2	4	7	5
Mary	Manager	5	NOW	5	5
John	Clerk	0	NOW	0	0

Table 6: Implicit information of the `Job` relation with transaction time and decision time

to those two dimensions. Now that we are convinced that a third temporal dimension is needed, we are able to present Figure 2, which shows how an event may be classified with respect to its decision time, valid time and transaction time. For the sake of illustration, and brevity we explain only a couple of possible events, others can be derived with the aid of Figure 2.

- Retroactive Late Transaction of a Futuristic Decision - As the transaction is retroactive we have $V_e \leq T$, being late implies $D < T$, finally a futuristic decision means that $D \leq V_s$, therefore $D \leq V_s \leq V_e \leq T$ which is feasible.
- Immediate Instantaneous Transaction of a Past due Decision - An immediate transaction implies $V_s = T$, as it is instantaneous we have $D = T$ and the past due decision yields $V_s < D$. Thus we have $T < D$ which is not feasible.

5 Summary and Future Directions

We have presented, by means of a relation evolving in time, why valid time and transaction time are needed to better model reality. We went further and argued why decision time is also needed as well. We have mentioned briefly how to use the transaction time index as an underlying structure to actually implement it. We assumed that relying on user-defined temporal attributes for these three time dimensions would incur too much overhead to the user and therefore those are desired to be supported by the DBMS itself. We discussed how the original relation key needs to be expanded as different temporal attributes enrich the tuple semantics. Using the three temporal dimensions simultaneously we could realize a novel taxonomy for temporal events.

Further research is being conducted in investigating a suitable indexing data structure containing both transaction and decision time. Finally, we plan on investigating on how to combine the three different indices, I_v, I_t and I_d, in order to be able to take advantage of all three temporal dimensions, and also how a query language such as TQuel [4] would need to be extended to allow decision time.

Acknowledgments

The authors thank Ashley C. Peltier and Clenio F. Salviano for comments in previous versions of this paper. Anonymous reviewers' comments and directions to new references are also acknowledged.

References

[1] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems.* Benjamin/Cummings, Redwood City, CA, 2nd edition, 1994.

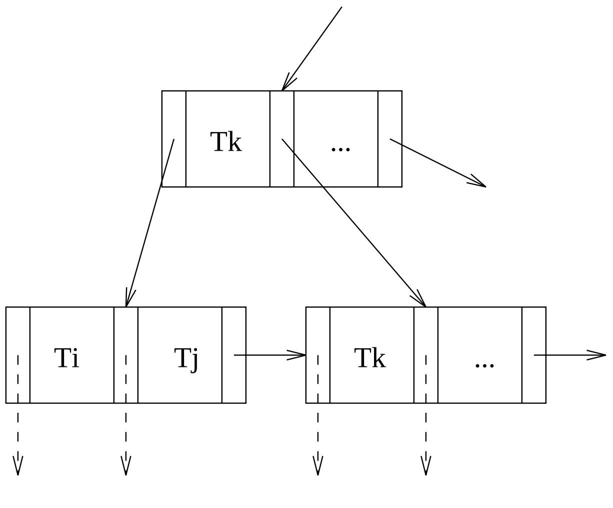
[2] C. S. Jensen et al. Proposed temporal database concepts. In *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, pages A1–A24, Arlington, TX, June 1993.

[3] N. Pissinou et al. Towards an infrastructure for temporal databases - Report of an invitational ARPA/NSF workshop. Technical Report TR 94-01, University of Arizona, March 1994.

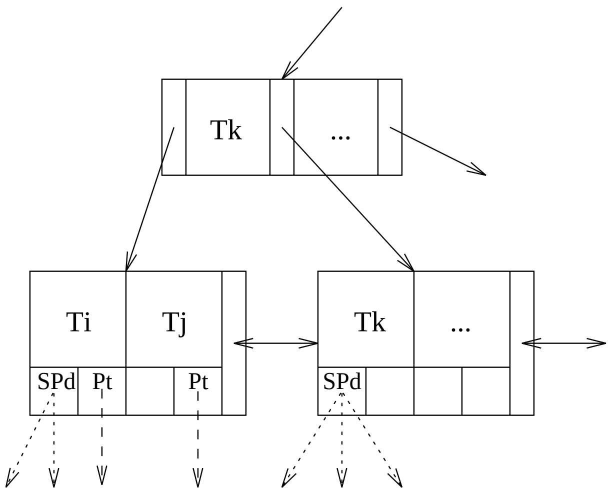
[4] R. T. Snodgrass. The temporal query language TQuel. *Transactions on Database Systems*, 12(2):247–298, June 1987.

[5] R. T. Snodgrass and I. Ahn. Temporal databases. *IEEE Computer*, 19(9):35–42, September 1986.

[6] A. Tansel et al., editors. *Temporal Databases: Theory, Design and Implementation.* Benjamin/Cummings, Redwood City, CA, 1993.

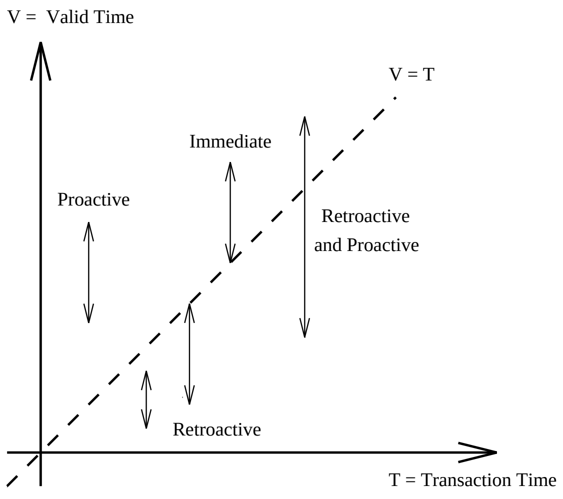


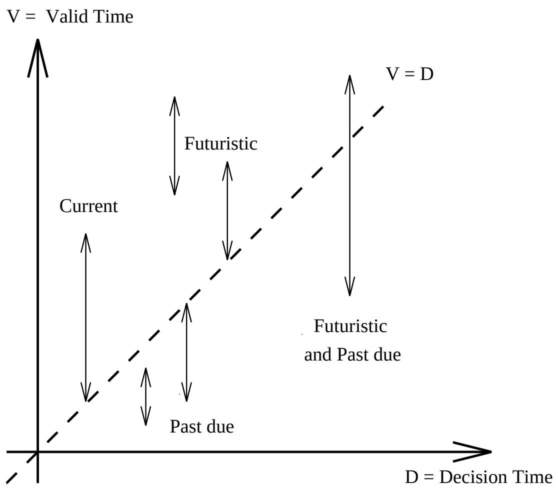
(a) Part of the bottom of a standard B+-tree



(b) Part of the bottom of the proposed indexing tree

Figure 1: Tree shaped indices structures for transaction (and decision) time





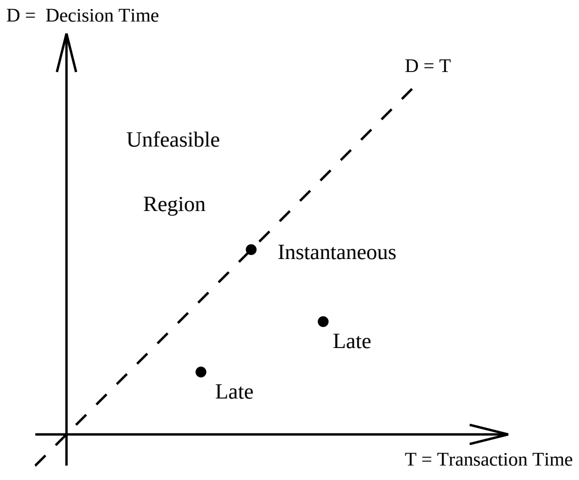


Figure 2: Classification of an event with respect to the three time dimensions