Temporal Categorization for Data Organization

Behrooz Seyed-Abbass, University of North Florida, 4567 St. Johns Bluff Rd S, Jacksonville, FL 32224, USA; E-mail: abbassi@unf.edu
Patrick O. Crews, University of North Florida, 4567 St. Johns Bluff Rd S, Jacksonville, FL 32224, USA; E-mail: gleebix@gmail.com

ABSTRACT

Temporal databases are designed to handle records that are time-oriented. While the additional dimension produces data that is rich in meaning, capturing this history results in large volumes of data that are rarely used. Previous research has found that the additional data can impair system performance when attempting to use temporal databases as operational systems. This paper presents work done in an area that the authors have termed, temporal categorization, which involves a method of data organization that groups records according to their unique temporal semantics. The initial testing on temporal categorization indicates the potential to effectively improve system performance as related to time-oriented data.

Keywords: Temporal Databases, Data Organization, Temporal Categorization, Database Performance

1. INTRODUCTION

While there are many ways of associating time with a fact, most temporal research only considers two time dimensions, valid times and transaction times, to be significant [1]. Valid times capture the history of a real-world object. Systems that handle this dimension of time are known as historical databases. Transaction times capture the history of a fact within the database itself and are handled by rollback databases [2]. Previous research has shown valid and transaction times to be orthogonal [3], but it has also been acknowledged that both of these dimensions are necessary to provide complete temporal functionality. Complete temporal systems are known as bitemporal databases due to their use of both time dimensions [4]. The following example illustrates a fictional, bitemporal employee database.

A company hires a new employee named Bob. He begins work on 1/1/2000 as an Engineer and is entered into the system on 12/30/1999, but is mistakenly listed as a Managing Engineer. This mistake is caught and corrected on 1/5/2000. Six months after being hired, Bob is promoted to the rank of Senior Engineer. Finally, Bob will be automatically promoted to the rank of Supervising Engineer one year after becoming a Senior Engineer.

Table 1 demonstrates how this data would be captured in a bitemporal database. Part A is the initial state of the system. Part B shows the table after Bob's title has been corrected. There are now two records in the system, the erroneous record that has been retired and the corrected record that is now active. Part C is the system after Bob's promotion to Senior Engineer. The Bob/Engineer record that was active in Part B has been retired and an updated version with a definitive Valid_To date inserted into the system. The predictive record of Bob's promotion to Supervising Engineer has also been inserted. It has the same Trans_From time

Table 1. Bitemporal employee database

Part A

Emp_Name	Title	Valid_From	Valid_To	Trans_From	Trans_To
Bob	Managing Engineer	1/1/2000	12/31/9999	12/30/1999	12/31/9999

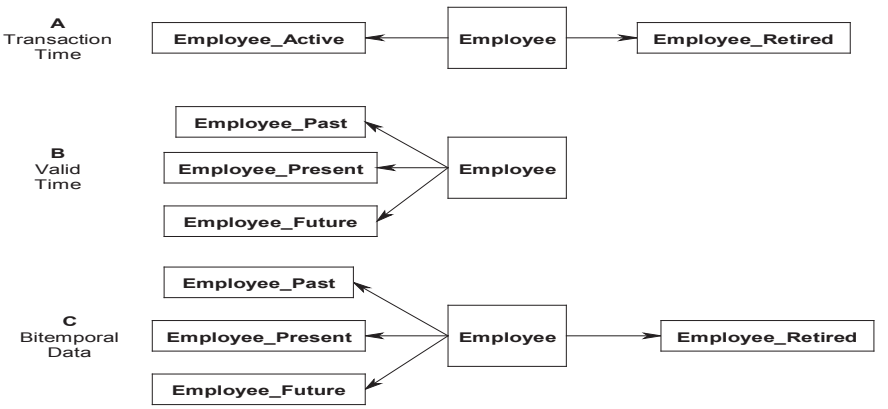
Part B

Emp_Name	Title	Valid_From	Valid_To	Trans_From	Trans_To
Bob	Managing Engineer	1/1/2000	12/31/9999	12/30/1999	1/4/2000
Bob	Engineer	1/1/2000	12/31/9999	1/5/2000	12/31/9999

Part C

Emp_Name	Title	Valid_From	Valid_To	Trans_From	Trans_To
Bob	Managing Engineer	1/1/2000	12/31/9999	12/30/1999	1/4/2000
Bob	Engineer	1/1/2000	12/31/9999	1/5/2000	6/30/2000
Bob	Engineer	1/1/2000	6/30/2000	6/30/2000	12/31/9999
Bob	Senior Engineer	7/1/2000	6/30/2001	7/1/2000	12/31/9999
Bob	Supervising Engineer	7/1/2001	12/31/9999	7/1/2000	12/31/9999

Figure 1. Temporal categorization



as the Bob/Senior Engineer record and both are considered active in the database even though the predictive record has not yet become valid.

The use of bitemporal data clearly adds semantic depth. A user is able to track when a record was entered into the system and when a record was updated or retired, as well as the history of the data in relation to the real world. Unfortunately, this depth comes at a price. The results of the large number of stored records have been shown to negatively impact data retrieval and insertions [5][6][7]. This is particularly true when the database user wishes to focus on records that are currently true in both the real world and the database.

Preceding approaches for dealing with the large number of stored records range from the use of external storage to the implementation of high-performance indexes. One of the first solutions identified the impact of mingling active and retired data in a single system**,** which is comprised of a two-level store to segregate active and retired data [5]. However, the work did not extend to bitemporal systems or to more complicated queries. Another approach focused on using external storage for retired data [8]. While this does allow for the maintenance of active and retired data, it does not permit queries against all data within a system and thus severely limits bitemporal functionality. The use of specialized temporal indexes has also been explored [6]. These high-performance data structures are tailored to the challenges of bitemporal systems, but they also require kernel level changes to the database and are unlikely to be used until they are incorporated into commercial database management systems.

This paper describes a proposed extension to the two-level store where the records are physically separated for categorization according to their temporal semantics by transaction time (active versus retired), valid time (past, present, and future), or a combination of both (sorting active records by valid time and keeping retired records separate from the active ones). The next two sections present an overview of the proposed temporal categorization and considerations related to the database and query processing requirements. Section 4 discusses the initial implementation findings followed by conclusions and recommendations for future work.

2. TEMPORAL CATEGORIZATION

Adding one or more dimensions of time to a database results in increased time for query processing [5][6][7]. This can be attributed to several factors, including the enforcement of temporal constraints and the additional volume of records produced by maintaining historical information in the database [9]. This research work proposes a method, referred to as temporal categorization, of organizing temporal data to alleviate the query processing issues resulting from numerous records. In temporal categorization, the records in a table are physically separated and grouped according to their temporal semantics. The technique involves the creation of separate tables (or storage spaces) for the temporal categories of active and retired in transaction time and valid time.

2.1 Transaction Time

Transaction times are the history of a record within the database. The time values define when a record was entered into the database and when the system stopped regarding it as being true. Therefore, only two semantic categories are defined by transaction times, records that are active and records that are retired. For example, the Employee from Table 1 would be separated into Employee_Active and Employee_Retired. The separation of the records is illustrated in Figure 1 A.

All records are considered true when they are first inserted into the system. This status changes only when a record is updated or deleted. In a transaction time system, a delete operation does not physically remove data from the system. Instead, it is logically deleted [2]. This means that the data remains within the system, but it is marked as inactive and no longer true. The system marks retired data by having a Trans_To value that is less than the current time. Updating a record results in its original form being retired while the newer version of the data becomes what is considered active by the database. An update can be considered to be a combination of a delete transaction and an insert transaction.

2.2 Valid Time

Valid time describes when a fact was true in the real world and creates three possible categories for a record. It was true in the past, it is true in the present, or it will be true in the future [2]. Categorizing records by these semantics is more complicated than using transaction times. This is due to the possibility of present and future records changing their categories. Present records may cease to be true and be moved to the past category. Future records may become true and be moved into the present category.

Figure 1 B presents the categorization of records according to valid times. The records are separated into past, present and future groups. While one cannot truly predict what the future state of an object will be, there are many instances where it is useful to store the predicted state of an object. The inclusion of predictive records is not a requirement for a valid time database, but they do represent a semantic possibility of this time dimension [10]. Therefore, they are included in the proposed valid time categorization.

What makes the valid time changes so challenging is the fact that these recategorizations are not due to user (or system) action alone. They can also result from the passage of time in the real world. The present is always moving and a record that is valid now might not be valid after a few minutes. A database that employed this scheme would need to regularly check future and present records to see if they require a change in category in addition to monitoring the effects of any user updates. There is also the possibility of not having enough records to justify the overhead of categorization. The use of this type of categorization would be decided by how many objects were modeled in the database, how much history (or future) each object had, and how many states were allowed for an object at any point in time.

2.3 Bitemporal Categorization

Figure 1 C presents a possible organization for bitemporal categorization. All active records would be categorized according to their valid time semantics (past, present, or future) and would keep any retired records in a separate group. Further categorization of a bitemporal table's retired records was considered, but it was decided that this might be ineffectual because many queries against a particular state of the database (other than current active data) would fall outside of easily defined temporal categories. Trying to presort the data for every possible temporal query would not be constructive.

3. CONSIDERATIONS FOR CATEGORIZATION

Categorization of temporal data has the potential to improve database performance as related to query processing time and to the complexity of temporal queries. The records in a bitemporal database system throughout time will change by update operations to retired and active records. Over time, the database collects a significant volume of retired records, making the overhead of the temporal categorization process worthwhile. Several possibilities for handling the time-based update and query retrieval have been contemplated.

3.1 Database Categorization

One possibility would be to poll the present and future records at the time of a user query similar to the standard SQL column, shown in Table 2. However, this could severely impact query performance because the system would have to check all of the records for the categorization condition based on the user=s request, particularly when the table grows with retired and valid record over a period of time.

A second possibility would be to maintain the time value of the next shift in categories, such as the earliest Valid_To time in the present group or the earliest Valid_From time in the future group. The system could then reclassify the affected records at the proper time. Unfortunately, any updates, insertions, or deletes would require the system to update its list of update times since these actions could render it inaccurate.

Another option would be for the database to poll the present and future groups at a given time interval, recategorizing as necessary. This approach is highly dependent on the granularity (or level of precision) of the valid time values. If the valid times are only precise to the day, the database could check the tables at the start of each new day. If the valid times were of microsecond granularity, the database would be doing nothing but polling the tables.

3.2 Query Categorization

In practice, if one table is used for holding active and retired data to provide a simple and more optimized query processing, a flag field can be set to mark the retired records. The flag field may be hidden from the users. It is activated and set to retired during the Update and Delete operations according to the semantics. The SQL data manipulation commands would use the flag to access only retired or active records in an optimized approach. To retrieve all the retired records from Employee table as in Table 1 C, a user may execute the following command.

SELECT * FROM Employee WHERE CATEGORY = RETIRED;

CATEGORY may use the options of [RETIRED | ACTIVE] for transaction time to select the correct category of the records from the table.

The active records in the table may be extended into past, present or future categories. In this case, a more comprehensive flag and temporal semantic comparison methodology can be used and the flag can be set to refer to different category domain of past, present or future. To access a particular category domain, the temporal SQL command uses the reserve word CATEGORY with any of the [PAST | PRESENT - CURRENT | FUTURE]. A command could be used to select particular domain as well as search for a conditional semantic value related to date or any other simple or compound condition. Table 2 shows several SQL categorization examples using the fictional Employee table in Table 1 with the query and the SQL that would produce the desired results in a standard system as well as a theoretical database using the proposed temporal categorization.

4. INITIAL IMPLEMENTATION OF TEMPORAL CATEGORIZATION

Physically separating current and historical records to improve system performance across a broad range of temporal queries would be a result of having multiple smaller tables for the system to query rather than one large table. Additionally, having the records separated by temporal semantics eliminates the need for evaluating each record=s timestamps in certain cases. Consider a system where records are categorized by transaction times. If a user wished to search only active records with a Trans_To time of 12/31/9999, the database would not need to test the Trans_To values of each record. It could just run the query against the active data set, ignoring the retired data completely.

To test the possible benefits of this categorization methodology, a simple experimental prototype system with separate tables to support retired and active records was constructed. As records were retired via Update or Delete actions, they were moved into the retired table. The data in these experiments consisted of a single key value coupled with valid and transaction times that tracked an object=s status through a period of time. While this cannot be considered representative of all bitemporal data, it does represent one of the more common applications of temporal databases. Each Insert represents a new state and requires an Update or Delete

Table 2. Comparison of queries across temporal systems

Results	SQL (standard)	SQL (categorized)
All data in the table	SELECT * FROM Employee	SELECT * FROM Employee
All active data in the table	SELECT * FROM Employee WHERE Trans_To = '12/31/9999'	SELECT * FROM Employee WHERE CATEGORY = ACTIVE
All current, active data in the table	SELECT * FROM Employee WHERE Trans_To = '12/31/9999' AND Valid_From < Current Time AND Valid_To > Current Time	SELECT * FROM Employee WHERE CATEGORY = CURRENT AND ACTIVE
All past, active data in the table (interchangeable with future)	Select * FROM Employee WHERE Trans_To = '12/31/9999' AND Valid_To < Current Time	SELECT * FROM Employee WHERE CATEGORY = PAST AND ACTIVE
All active data that was valid within a specified time period (valid for any part of interval)	Select * FROM Employee WHERE Trans_To = '12/31/9999' AND Valid_From < *Start Time* AND Valid_To > *End Time*	SELECT * FROM Employee WHERE CATEGORY = ACTIVE AND Valid_From < *Start Time* AND Valid_To > *End Time*
All data that was valid at a particular point in time for the database	Select * FROM Employee WHERE Trans_From < *Time* AND Trans_To > *Time* AND Valid_From < *Time* AND Valid_To > *Time*	Select * FROM Employee WHERE Trans_From < *Time* AND Trans_To > *Time* AND Valid_From < *Time* AND Valid_To > *Time*

of the current one. The design choice was made to monitor temporal constraint performance as well as query times.

To allow greater flexibility in searching the records, the actual values were also tested in addition to the keyword indicator, such as active or retired, as shown in Figure 1 and Table 2. This provided the capability to test records that were retired after a given date rather than only retired records. The system determines which tables to run against by examining the Trans_To value in the query. If the value matches the placeholder to signify activity, only the active table is searched. If the Trans_To value is something other than that placeholder, it means the record has been retired and only the retired table is used. A single evaluation is performed against the queries Trans_To value, in contrast to the standard system=s need to compare each record=s data against that of the query.

An experimental prototype system was designed to handle the transaction time with active and retired data in combined and separate tables. The performance analysis was focused on transaction time categorization due to its comparative simplicity of result and ease of implementation. Additionally, a combination of Insert/Update/Delete operations were implemented. These operators were built to test the insertion of new versions of real world data and the act of a record's current state becoming real world inactive. A control system was also implemented as the standard system. This system contained all of the functionality of the experimental system minus the categorization. Both systems were built from scratch in an effort to ensure more effective comparisons between systems. They were implemented using the Python programming language.

A collection of queries was designed and run against the standard and experimental systems. The performance data for the queries were collected and averaged. These queries were designed with various business rules and settings for different scenarios that are summarized in following cases. For each case, the queries for the data status involved three levels: all, active after a particular date and active between specified dates.

Case 1: Database records with active status
Case 2: Database records with active status after certain date
Case 3: Database records with active status between date intervals
Case 4: Database records with active and retired status

The performance and results of these queries for different numbers of records were collected and tabulated for both the standard and experimental system, and then averaged. The averaged comparison diagram is shown in Figure 2. It was observed that the categorized experimental system outperformed the standard system in every query category. Figure 2 shows a sample of the average system performance for all different queries used in cases 1-4. The data clearly shows the categorized system's performance to be significantly faster.

Figure 3 shows the insertion times required for various numbers of records in both the standard and experimental system records. The experimental system shows improved performance. It is suspected that this is due to not needing to test whether or not a record is active or retired before testing for any temporal constraints on a table.

Finally, in terms of data storage, there was a constant difference between the systems. The experimental systems showed a difference of approximately 30 additional bytes for any number of records, which appeared to be related entirely to the additional data structure used for holding retired data. The number of records stored did not affect the size difference.

Figure 2. Average query performance comparison

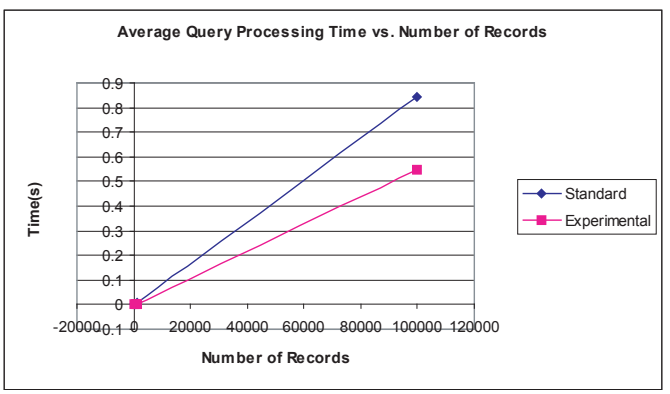


Figure 3. Comparison of record insertion times

5. CONCLUSIONS AND FUTURE WORK

This paper has presented an overview of the authors' current research work on temporal categorization, which was developed as a proposed solution to the problem of how to deal with the large volumes of data that are produced by adding one or more time dimensions to a database. While alternate methods have been suggested, these have typically involved using secondary storage and do not allow for immediate access to the data. By physically separating records according to their temporal semantics, temporal categorization may produce improved query performance due to the smaller volumes of data that must be searched as well as the reduced number of comparisons that must be made in order to find the desired data. This is especially true when dealing with those records that are active and defining the state of objects in the present.

The initial experimental system demonstrated promising performance gains in terms of data retrieval and insertion operations. Even those queries that were expected to show reduced performance were completed more quickly for the categorized system. Additionally, the improved query run times came at the cost of constant storage overhead, which the authors view as a minor cost in comparison to the benefits. In light of these initial findings, temporal categorization can be seen as a viable possibility for temporal data organization. The technique promises improved query processing times and faster data insertion / update transactions for a minor storage cost. The next steps will be to implement this technique in a more robust database system and to explore additional prototypes for temporal semantic categorization.

Future work will consist of evaluating the feasibility and performance of the temporal database categorizations for storage space requirements, insertion and update times, and data retrieval operations for different combinations of temporal criteria. The authors are also interested in seeing how this technique would compare to temporal indexing as a performance enhancement technique. Additionally, it would be worthwhile to examine a wide variety of temporal data sets to better understand how real-world users are storing time-associated records. Finally, the question of valid time categorization must be addressed. This topic will be examined in terms of implementation performance as well as which patterns of database usage will be best suited for these techniques.

6. REFERENCES

[1] Christian S. Jensen and Richard Thomas Snodgrass, "Temporal Data Management," IEEE Transactions on Knowledge and Data Engineering, Volume 11, Issue 1 (January 1999), pp. 36-44.

[2] Christian S. Jensen, "Introduction to Temporal Database Research," Temporal Database Management dr.techn.thesis (April 2000).

[3] Christian S. Jensen, Michael D. Soo, and Richard Thomas Snodgrass, "Unifying Temporal Data Models via a Conceptual One," *Information Systems*, Volume 19, Issue 7 (October 1994), pp 513-547.

[4] Niki Pissinou, Richard Thomas Snodgrass, Ramez Elmasri, Inderpal S. Mumick, Tamer Özsu, Barbara Pernici, Arie Segev, Babis Theodoulidis, and Umeshwar Dayal, "Towards an Infrastructure for Temporal Databases: Report of an Invitational ARPA/NSF Workshop," *ACM SIGMOD Record*, Volume 23, Issue 1, (March 1994), pp 35-51.

[5] Ilsoo Ahn and Richard Snodgrass, "Performance Evaluation of a Temporal Database Management System," *ACM SIGMOD Record*, Volume 15, Issue 2 (June 1986), pp 96-107.

[6] Rasa Bliujute, Christian S. Jensen, Simonas Altenis, and Giedrius Slivinskas, "Light-Weight Indexing of General Bitemporal Data," *Proceedings of the 12th International Conference on Scientific and Statistical Database Management*, (July 2000), pp. 125-138.

[7] Anil Kumar, Vassilis J. Tsotras, and Christos Faloutsos, "Designing Access Methods for Bitemporal Databases," *IEEE Transactions on Knowledge and Data Engineering*, Volume 10, Issue 1, (January 1998), pp 1-20.

[8] Betty Salzberg and Vassilis J. Tsotras, "Comparison of Access Methods for Time-evolving Data," *ACM Computing Survey*, Volume 31, Issue 2 (June 1999), pp. 158-221.

[9] C. J. Date, Hugh Darwen, and Nikos Lorentzos, *Temporal Data and the Relational Model*, Morgan Kaufmann, San Francisco, 2002.

[10] James Clifford, Curtis Dyreson, Tomás Isakowitz, Christian S. Jensen, and Richard Thomas Snodgrass, "On the Semantics of "Now" in Databases," *ACM Transactions on Database Systems,* Volume 22, Issue 2, (June 1997), pp 171-214.