IMPLEMENTING A MODERN TEMPORAL DATA MANAGEMENT SYSTEM

Ramon Mata-Toledo[1]
Morgan Monger[2]

Abstract

Temporal data management is a concept that has been around for many years. A temporal data management system (TDMS) manages data that is tracked over time. In this paper, the authors present an Oracle-based implementation of a TDMS that provides access to temporal data. The design and implementation presented in this paper are presented at a high level, with the significant features such as reference intervals and temporal relationships. The most notable TDMS benefits are a semi-portable solution and an implementation that maximizes on native database features. The paper finally presents an evaluation of the TDMS implementation with a feature comparison and benchmarking.

Introduction

Temporal data is quite simply data that is tracked over time. A temporal data management system (TDMS) manages temporal data analogous to a relational database management system (RDBMS) managing snapshots of data; for the purposes of this paper, the RDBMS will be referenced as a snapshot database or snapshot DBMS. Various approaches have been proposed over the years regarding temporal data management, including middleware, query language extensions, and a native TDMS. Rather than build on the diverse and non-standardized implementations of a TDMS, an Oracle-based implementation has been created that encompasses the chosen best features of these existing solutions [3] [5] [8]. In addition, the TDMS presented in this paper incorporates temporal relationship operators and reference intervals to enforce data validity [1]. Such an implementation based on native features can provide automated temporal data management functions without complex application development, while providing acceptable performance. In the next sections, the design and implementation considerations are discussed.

Design And Implementation

From a design perspective, there are certain objectives that must be achieved by the TDMS: full DBMS compatibility, maximization of native features, simplification of implementation, and acceptable performance [9]. Compatibility is achieved by isolating TDMS functionality from the Oracle functionality. Maximizing use of native features is achieved by utilizing key Oracle features: functions, procedures, and triggers. Simplifying the implementation requires managing the complexity of the TDMS [9]. Acceptable performance is achieved by minimizing the overhead of temporal operations.

[1] Ramon Mata-Toledo, Professor at James Madison University, email: matatora@jmu.edu

[2] Lead Developer/Designer, Datatel, Inc., email: mdm@datatel.com

With respect to the design objectives, there are also design constraints for the TDMS. Reference intervals and temporal relationship operators are essential to the TDMS temporal constraints. Temporal relationship operators are implemented as functions and can be utilized like any other SQL function. Temporal data is presented as temporal views, where a temporal view is defined as a merge of an original snapshot table and a temporal table. This has the benefit of separating the presentation of the temporal data from the actual values, permitting custom views to be built from the temporal table. The primary key of the snapshot table has been limited to a single numeric value. This limitation is necessary because tracking different data types or multi-part keys increases the complexity of the TDMS design. Finally, automating the TDMS has been implemented using the trigger functionality of Oracle [9].

Architecture

At the design level, the architecture can be viewed in a layered model as shown in Figure 1. The TDMS exists in the RDBMS at the same level as the Database. The right side of Figure 1 shows the component architecture of the TDMS. The use of native features is shown with the procedures, functions, and triggers. The Temporal Catalog centralizes the mapping of the snapshot table(s) and column(s) to a temporal table. The Temporal Data Table provides a temporal table pattern that allows consistent automated functionality. The Reference Interval Metadata contains information regarding the temporal constraints which are composed of reference intervals and temporal relationship operators.

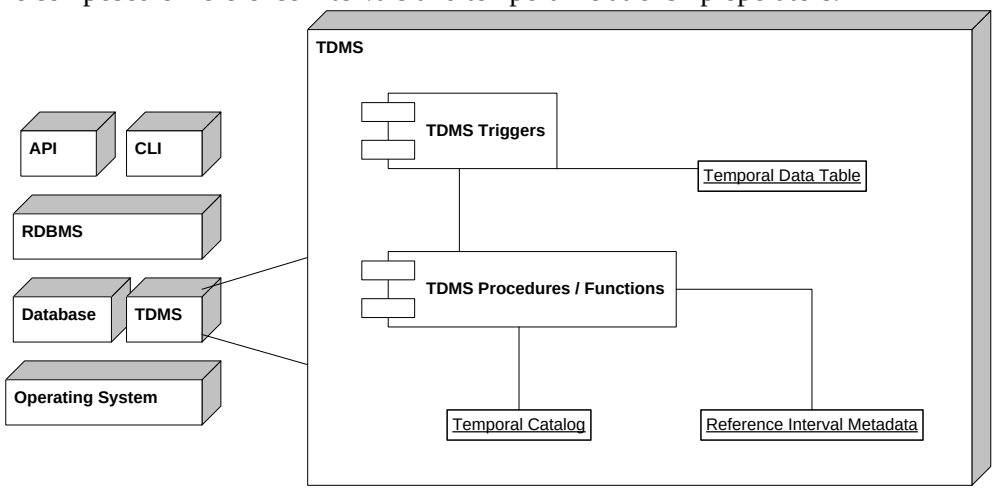

Figure 1: TDMS Architecture

Deployment Method

The TDMS is deployed as an isolated subsystem of Oracle and is constructed as a SQL script that contains the PL/SQL code and table metadata. This provides flexibility in deployment because the SQL script runs in many different client interfaces; however, some constraints were necessary in the implementation of this TDMS to provide the aforementioned flexibility. Notable deployment constraints involve tables currently accessed while being modified (mutating tables) and dynamic tables (pipelined functions) [4].

Implementation Constraints

Though these constraints are implementation specific, the implementation may provide an example solution to other developers encountering similar issues. A mutating table is a problem because the triggers may need to query the snapshot table to create a copy for temporal data. A resolution to the problem is to use a state package to track the modified rows for the various triggers [7]. To manipulate interval information, dynamic tables pass back modified rows of data to create new tables, which can be collapsed or expanded accordingly.

Evaluation

Implementation Comparison

Temporal and functional comparisons of other TDMS implementations that are based on Oracle provide a good implementation evaluation. ChronoLog is a front-end that compiles temporal requests into SQL commands for Oracle. T-squared DBMS is a temporal relational database integrated into Oracle. Böhlen provides a very comprehensive comparison of these systems from temporal and functional perspectives [2]. From the comparisons in Table 1, the presented TDMS appears to exhibit common features of the other implementations.

	Tuple Timestamps	**Attribute Timestamps**	**Temporal Queries**	**Temporal Updates**	**Temporal Constraints**	**Temporal Views**
ChronoLog	X		X	X	X	X
T-squared DBMS	X		X	X		X
TDMS		X	X	X	X	X

Table 1: Functional Comparison

Benchmarking

Benchmarking provides a different means of comparison between implementations than the implementation comparison in the previous section. A semantic benchmark provides a means of verifying that a query against the temporal data returns the correct results [6]. Figure 2 shows an example where the valid start and end dates for marriage status when individuals lived with their parents. This query provides a good example of using the TDMS to discover when values were valid. A performance benchmark can be used to compare snapshot query performance to TDMS query performance [6].

The performance benchmarking is based on various size random data sets: 1,000, 10,000, and 100,000 records. The tests for each data set include INSERT and UPDATE operations with the TDMS enabled (Temporal) and disabled (Snapshot). A notable observation is that the TDMS takes longer to handle INSERT and UPDATE operations on a temporal column. For the INSERT and UPDATE tests, the TDMS performance degrades considerably for larger data sets. The authors believe this performance can be improved by using native optimizations such as indexing and caching. The performance

degradation appeared to be the result of the temporal trigger scheme in this implementation. Despite the large data set performance problem, the TDMS seems to handle data sets of 10,000 records or less quite well.

Question:
What were the marital statuses of those who lived with their parents?

Query:

```
select v.name,
'[    '||to_char(to_date(t.v_start),'MM/YY')||',    '||to_char(to_date(t.v_end),'MM/YY')||'    )
'||v.mstatus as mstatus
from employ_view2_mstatus_tview v, employ_view1_residence_tview t
where v.id = t.id AND t.residence = 'With Parents' AND tdms_overlap(t.v_start,t.v_end,
v.v_start, v.v_end) = 1;
```

Result:

```
NAME          MSTATUS
------------- ------------------------
Jenny Aris    [ 12/77, 05/80 ) Single
Bill Nee      [ 01/74, 11/76 ) Single
Ken Witts     [ 01/80, 01/82 ) Single
Ken Witts     [ 04/86, 06/86 ) Divorced
```

Figure 2: Example Semantic Test

Future Improvements

With regards to the current implementation, there are some improvements that can be addressed. First, the performance can be improved with a new trigger scheme or native optimizations. The next change to the TDMS involves adding removal procedures. This functionality can benefit the TDMS by providing consistent removal methods, similar to the TDMS creation procedures. Another beneficial change involves allowing multi-part keys; the current solution restricts primary keys to a single numeric column. This change can allow more complex snapshot tables to be tracked by the TDMS. Enforcing temporal relationship changes can also benefit the TDMS. For instance, if the temporal relationship operator is altered in the Temporal Catalog, then a trigger condition can verify that the existing temporal data does not violate this change before writing the new temporal relationship operator.

Conclusions

The benefits of the TDMS are that it provides an operating system independent solution and maximizes on native features. The TDMS provides a semi-portable solution that works in any Oracle 10 RDBMS. The TDMS utilizes many native features of the Oracle RDBMS. The use of triggers, procedures, and functions in the TDMS provides a unique system that makes the best use of native features for performance and stability. The

implementation comparison results demonstrated that the TDMS matched up to existing implementations. For benchmarking, there are two types for this paper: semantic and performance. For each temporal query, the correct data is returned in all instances. The performance benchmarking demonstrated that the TDMS performed quite well for data sets of 10,000 records and smaller. Finally, there are several suggested improvements that can improve performance and allow for more complex temporal operator functionality.

References

[1] J.F. Allen, "Maintaining knowledge about temporal intervals," Communications ACM,
vol. 26, pp. 832-843, 1983.

[2] M.H. Böhlen, "Temporal database system implementations," SIGMOD Rec., vol. 24, pp. 53-60, 1995.

[3] C.J. Date and H. Darwen, Temporal Data and the Relational Model, Morgan Kaufmann Publishers Inc, 2002.

[4] S. Feuerstein and A. Odewahn, Oracle PL/SQL Developer's Workbook, Sebastapol, CA: O'Reilly & Associates, 2000.

[5] C. Jensen, "Temporal Database Management," pp. i-1328, 2000.

[6] P. Kalua and E. Robertson, "Benchmark queries for temporal databases," "Computer Science Department, Indiana University, Bloomington, Indiana 47405, USA"., March, 1993.

[7] T. Kyte, "Avoiding Mutating Tables," vol. 2005, pp. 7, 8/9/1999. 1999.

[8] R.T. Snodgrass, Developing time-oriented database applications in SQL, Morgan Kaufmann Publishers Inc, 2000.

[9] C. Vassilakis, P. Georgiadis and A. Sotiropoulou, "A Comparative Study of Temporal DBMS Architectures," in "DEXA Workshop", pp. 153-164, 1996.