Temporal Preprocessor: Towards Temporal Applications Development

© Boris Kostenko

Moscow State University
bkostenko@acm.org

Ph.D. advisor: Sergey D. Kuznetsov

Abstract

Effective querying and managing of temporal databases represent an unanswered challenge to the modern research community. In this paper, we introduce a temporal preprocessor that can be used to aid in creation of temporal applications and for testing various methods and approaches in temporal databases implementations. We outline our proposal, current results and directions for further research and development.

1 Introduction

Most database applications that are widely used in our everyday life manage time-related data. We can name nearly any of reservation, schedule, planning, monitoring and managing applications as an example. Databases that store time-varying data are called temporal databases.

During our research in the area of temporal databases, we decided to create a temporal extension that would meet the following requirements. First, it must provide an effective solution for developing temporal applications implemented over commercial database management systems (DBMS) with minimum additional effort from a developer. Second, this solution must be flexible, so we can easily modify, test and compare different approaches and methods of temporal queries implementation with each other.

Temporal databases have been the focus of much research work and many solutions have been proposed so far. Most temporal database system prototypes were based on layered approach, when a special layer or stratum between temporal user application and ordinary database management system was established. Such layer converts temporal queries received from a user into statements in SQL (or any other) query language and passes them to the DBMS. After that, the corresponding result from the database is translated back into the temporal one and returned to the user. We also selected layered approach, but we tried to integrate our layer into a source code of application and/or database system. Thanks to such integration, one can be sure that a resulting application is working as effective as if a developer coded it himself. This integration can be done by using temporal preprocessor or compiler on an application source code. We decided to start with the PHP programming language, because it has clear syntax and semantics and is powerful enough to manipulate with arrays. Moreover, it has dynamic declarations, thus we can create functions and execute them in runtime.

Proceedings of the Spring Young Researcher's Colloquium On Database and Information Systems SYRCoDIS, Moscow, Russia, 2007

2 Related works

During the last two decades, many researches were conducted in the temporal databases area of research. Some of them dealt with design of temporal databases. Others propose methods of effective indexing, implementation of join statements and so on. One of the first temporal query language TQuel and its partial implementation for the Ingres DBMS was introduced in [2]. Later many papers introduce temporal query languages that usually extend SQL query language [3, 6]. However, there were always two approaches to temporal database management system implementation: either build it from scratch or create a temporal middleware. In [7, 8] the second (layered) approach concepts are discussed in detail. In [1] author gives direct solutions how to express temporal queries in a SQL language. However, these "SQL" solutions are not always good because of many nested selects, joins, unions and dynamic tables manipulations.

Unlike [2, 3, 6] we do not start with temporal query language definition, but we provide query functions for most useful temporal requests, so a developer just needs to provide a temporal clause in addition to ordinary SQL query. On the other hand, we have additional opportunities for particular queries optimizations. Another distinguishing feature of the proposal is the tremendous flexibility both for developer who can alter predefined algorithm any time and for researcher who can easily add new methods and check them instantly.

A public available working temporal DBMS – TimeDB [4, 5] is implemented on top of IBM

Cloudscape 10 or Oracle 10g as a layer between user application and underlying DBMS. Thus, it needs additional resources to keep it alive in compare with compilation time solution, proposed in this paper.

We do not want to oppose our proposal to the temporal DBMS as our solution complement to them in the areas where developers need to create their own temporal applications over relational DBMS because there exists no adequate product on a market or a temporal DBMS cannot be used for any reason. Moreover our preprocessor can be used with temporal DBMS to help to reduce errors in a source code (and in runtime if desired) and allow developers to formulate query statements clearly in a way they prefer.

3 Architecture and implementation issues

The proposed extension has 3-modules architecture (see Figure 1). Temporal queries are processed in the following sequence: query > parser >> parsed query > optimizer >> script (plan) > code generator >> source code. Parser receives a query from a user and converts it to the internal representation format (parsed query). Next optimizer analyzes parsed query and selects processing method according to the database tables definitions. Optimizer generates a script (plan) that describes how to get temporal query result from the current DBMS. After that, the code generator takes the script and produces a source code. The returned source code is the code that performs the specified temporal query on top of an existing relational database. This source code is stored as a function and developer just needs to replace an initial query execution with the call of this function. We use placeholders in temporal queries so each function call can be parametrized according to the current program state.

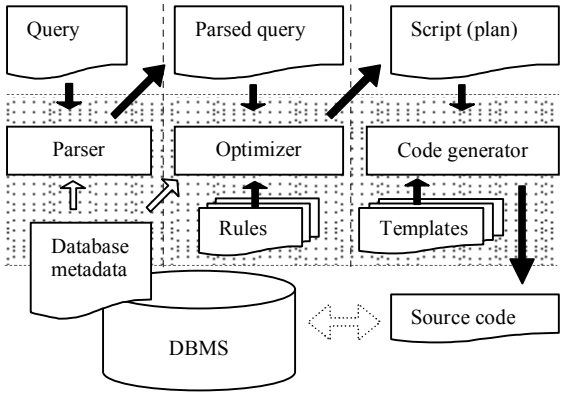

Figure 1. Preprocessor architecture

In the introduced architecture, each module performs single step regardless of other modules and steps. Thus, we can easily extend, modify or replace any module until we follow input/output format conventions. Fox example, we can extend code generator, and get a temporal processor for another programming language, including DBMS languages. Changes to optimizer lead us to different methods of temporal queries handling as well.

Here we need to note that considerable part of logic is encoded in "Rules" for optimizer and "Templates" for code generator. Because we selected PHP as our first target language and it is possible to dynamically create a function from its source code and call it after that then we can perform all needed temporal tasks in runtime. Thus, we can create a ready to use temporal database extension for PHP. A developer just needs to include a library file, provide temporal processor with table definitions that include field names and their types (with temporal ones), and call required temporal queries with temporal processor runtime proxy.

As noted above, many researches of temporal database extensions started with introduction of a query language extension and then demonstrated how different temporal requests could be expressed with this query language. Unlike them, we chose another approach. We assume that a developer knows what information he wants to retrieve from a database. Actually, he does not even need a database query result. The main aim of database data retrieval is to populate some structures or arrays in a program with certain data. Moreover, it is often more natural for a developer to formulate his data requirements in natural language than to translate them into a query language and then retranslate query execution results back. We also suggest that optimizations that are more specific are better than common ones. Therefore, we introduce not query language but query functions.

To understand better differences between them consider the next two "queries": query("select * from employees where company_id = 17")', and 'query_with_company_id("select * from employees", 17)'. The results of these two calls will be identical (function names are self-explanatory), but in general with the second approach we will get more stable and errorless solution. Moreover we can use next query 'query_employees_table_with_company_id(17)' that is even more specific. In most cases later variants are better to use, because they are more obvious to developer, there are more ways of optimizations and there is less chance of a wrong usage. So a developer can create such parameterized query functions for often used queries. In any case one can always use the most general query function to express his query.

In case of temporal databases, we use several predefined query functions to formulate statements that use time and time relations. Thus, we selected most useful and high-usage queries to create convenient functions for them. Almost all "single state" requests can be performed with query_on_time() function, which limits query to a specified point of time. In order to use relations and aggregate functions on timeline we need to add functions like query_following_events() and query_timeline_aggregates(). To successfully execute queries with projections and database modifications we needed a set of functions for time intervals set operations. These functions provide us with an ability to fold time intervals and merge them to archive

normalized state of valid-time interval in database if needed.

There are two components in querying with functions: one can use only simple query function but formulate all queries fully, or formulate query only once and create a query function for it and after that call this function with actual parameters. But in case of temporal queries we know how to process temporal columns, that's why we provide some predefined query functions, get less information about particular queries from a developer, and provide some optimizations.

The proposed temporal preprocessor is responsible for implicit query functions generation based on user queries. So a developer can provide more specific "Rules" and "Templates" and achieve better performance, or just use standard ones.

4 Results achieved and future work

The research and implementation are not finished yet, thus we cannot provide performance graphs, and comparison charts between different methods. However, current results show that we have already achieved simplicity of use and flexibility for further research and experiments. We also noted that temporal applications development process is speed up, because less time required formulating correct queries and testing them.

Parser facilities are limited now and we plan to extend them in order to parse rather complex temporal queries. Another goal is implementation of code generators for more programming languages so we will have the ability to compare performance issues of various methods. We also need to add reliable performance counters and find adequate formula to estimate and to compare effectiveness of different approaches.

Further research of how to retrieve information grouped by timeline fields and how to aggregate them better is desired as well.

5 Conclusion

In this paper, we introduced a solution for effective and easy development of temporal applications on top of existing relational DBMSs. The proposed solution also provided us with an opportunity to test and compare different models and approaches in temporal databases implementation. The achieved results gave us new ideas and showed possibilities of success in further research and development.

References

[1] Richard T. Snodgrass. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann Publishers, Inc., San Francisco, July, 1999, 504+xxiii pages.

[2] Richard T. Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems* 12(2), June 1987, pp. 247–298.

[3] R.T. Snodgrass, M.H. Boehlen, C.S. Jensen, and A. Steiner. Adding Valid Time to SQL/Temporal. *Change proposal, ANSI X3H2-96-501r2, ISO/IEC JTC 1/SC 21/WG 3 DBL-MAD-146r2*, November 1996.

[4] TimeDB - A Bitemporal Relational DBMS. Web site, May 2005. http://timeconsult.com/Software/Software.html

[5] TimeDB - A Bitemporal Relational DBMS. TimeDB release version 2.2. (Zip-archive, Java and JDBC), May 2005. http://timeconsult.com/Software/TimeDB 2.2.zip

[6] David Toman. Point-Based Temporal Extension of Temporal SQL. *In Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases*, pages 103-121, 1997.

[7] Kristian Torp, Christian S. Jensen, and Michael Böhlen. Layered Temporal DBMS: Concepts and Techniques. In *Database Systems for Advanced Applications '97, Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 371-380, 1997.

[8] K. Torp, C. S. Jensen, and R. T. Snodgrass. Stratum Approaches to Temporal DBMS Implementation. In *Proceedings of IDEAS, Cardiff, Wales*, pages 4-13, 1998.