Demonstration Proposal D27

TIP: A Temporal Extension to Informix*

Jun Yang Huacheng C. Ying Jennifer Widom
Computer Science Department, Stanford University
{junyang,ying,widom}@db.stanford.edu, http://www-db.stanford.edu/

Abstract

Commercial relational database systems today provide only limited temporal support. To address the needs of applications requiring rich temporal data and queries, we have built TIP (T*emporal* I*nformation* P*rocessor*), a temporal extension to the Informix database system based on its DataBlade technology. Our TIP DataBlade extends Informix with a rich set of datatypes and routines that facilitate temporal modeling and querying. TIP provides both C and Java libraries for client applications to access a TIP-enabled database, and provides end-users with a GUI interface for querying and browsing temporal data.

1 Introduction

Our research in temporal data warehouses [9, 10] has led us to require a relational database system with full SQL as well as rich temporal support, in order to experiment with our temporal view-maintenance techniques. Most commercial relational database systems support only a `DATE` type (or its variants). An attribute of type `DATE` can be used to timestamp a tuple with a specific point in time (e.g., "this patient was taking Prozac as of September 1, 1999"). But for general temporal modeling, a timestamp should be able to consist of multiple time periods (e.g., "this patient was prescribed Prozac from January to April, and then from July to October"). Furthermore, the `DATE` type does not include `NOW`, a special symbol interpreted as the current (transaction) time with many uses in temporal modeling [5]. Although it is possible to encode `NOW` and sets of time periods using `DATE` as the only type of timestamp, doing so can lead to extremely cumbersome queries and inefficient implementations. An alternative is to use one of the available temporal database prototypes [3], but as we will discuss in Section 5, none of them fully meets the needs of our temporal data warehousing application.

In recent years the major relational database vendors have introduced technologies that allow their systems to be extended with software plugins developed by users or third-party vendors. Examples include *Informix DataBlades*, *DB2 Extenders*, and *Oracle Cartridges*. Many extensions have been developed to address different application areas, and some are time-related, such as time-series, spatiotemporal, and temporal index extensions. Unfortunately, these extensions do not provide the general-purpose temporal support that we need, as discussed in Section 5.

Therefore, we have built TIP (T*emporal* I*nformation* P*rocessor*), an extension to the Informix database system based on its DataBlade technology. The TIP DataBlade implements a rich set of new datatypes and routines that support modeling and querying of temporal data. Once the TIP DataBlade is installed in Informix, TIP datatypes and routines become available to users as if they were built into the DBMS. Client applications can connect directly to a TIP-enabled database through a standard API such as ODBC or JDBC, and they can manipulate TIP datatypes using C and Java libraries provided by TIP. We also have developed a Java TIP client with a graphical user interface for querying and browsing temporal data.

*This work was supported by the National Science Foundation under grant IIS-9811947 and by an Informix Software Grant.

Note that TIP does not introduce a new query language for temporal data, such as TSQL2 [7]. Instead, TIP enables temporal queries expressed in SQL by providing an extensive set of built-in datatypes and routines. See Section 5 for further discussion.

2 Temporal Database Features

The TIP DataBlade extends the Informix type system with the following five datatypes related to time:

- *Chronon* represents a specific point in time, just like SQL's built-in `DATE` type. We will use "*year*-*month*-*day*[*hour*:*minute*:*second*]" to notate a `Chronon`. For example, one of the more famous `Chronon`'s is "`2000-01-01 00:00:00`," or "`2000-01-01`" for short. (And yes, TIP is Y2K-compliant!) Of course, TIP internally stores `Chronon`'s (and other datatypes described below) in an efficient binary format.
- *Span* is a duration of time between two `Chronon`'s, which can be either positive or negative. We will use "[+|-]*days*[*hours*:*minutes*:*seconds*]" to denote a `Span` . For example, "`7 12:00:00`" denotes "seven and a half days," while "`-7`" denotes "seven days back."
- *Instant* can be either a `Chronon` or a `NOW`-relative time. A `NOW`-relative `Instant` is represented by an offset of type `Span` from the special symbol `NOW`, whose interpretation changes as time advances. For example, "`NOW-1`" denotes "yesterday."
- *Period* consists of a pair of `Instant`'s. The first `Instant` marks the start of the period and the second one marks the end. For example, "`[1999-01-01, NOW]`" denotes "since 1999," and "`[NOW-7, NOW]`" denotes "during the past week."
- *Element* is a set of `Period`'s. For example, "`{[1999-01,1999-04], [1999-07,1999-10]}`" denotes "from January to April, and then from July to October."

These datatypes allow complex temporal data to be modeled. Consider a table that keeps track of the prescription history for patients. We can store a patient's date of birth as a `Chronon`, dosage frequency as a `Span`, and prescription periods as an `Element`. The following SQL statement is used to create the table:

```
CREATE TABLE Prescription
(doctor CHAR(20), patient CHAR(20), patientDOB Chronon,
 drug CHAR(20), dosage INT, frequency Span, valid Element)
```

The TIP DataBlade also defines an extensive collection of support routines for its datatypes:

- *Casts:* TIP provides casts between TIP datatypes whenever appropriate. For example, a `Chronon` can be converted to a `Period` containing only this `Chronon` (e.g., "`2000-01-01`" becomes "`[2000-01-01, 2000-01-01]`"). A `NOW`-relative `Instant` can be converted to a `Chronon` by substituting the current transaction time for `NOW` (e.g., "`NOW-1`" becomes "`1999-09-30`" if today's date is "`1999-10-01`"). TIP also uses casts to automatically convert SQL strings (in the same format as the strings we have been using in examples) to and from TIP datatypes. For example, the following SQL `INSERT` statement inserts the fact that Dr. Pepper has given Mr. Showbiz a long-term prescription of Diabeta starting from October:

```
INSERT INTO Prescription VALUES
('Dr.Pepper', 'Mr.Showbiz', '1930-01-01',
 'Diabeta', 1, '0 08:00:00', '{[1999-10-01, NOW]}')
```

- *Arithmetic and comparison operators:* TIP overloads built-in arithmetic operators (+, -, *, /) and comparison operators (=, <, >, etc.) to operate on TIP datatypes whenever appropriate. For example, a `Chronon` minus a `Chronon` returns a `Span`, but a `Chronon` plus a `Chronon` returns a type error. Another notable example is that the result of comparing a `Chronon` to a `NOW`-relative `Instant` may change as time advances, because the latter is interpreted based on the value of the current transaction time. Finally, as a general example, the following query finds all patients who were prescribed Tylenol when they were less than w weeks old, for input parameter w. In the query, `start` is a TIP routine that returns the start time of the first period in an `Element`. The "`::`" notation is Informix's syntax for explicit cast, needed in this case to resolve the `*` operator.

```
SELECT patient FROM Prescription
WHERE drug = 'Tylenol'
AND start(valid) - patientDOB < '7 00:00:00'::Span * :w
```

- *TIP-defined routines:* We have already seen a TIP routine `start` in the example above. In addition to `start`, TIP supports numerous routines that are useful in constructing temporal queries. For example, TIP supports Alan's operators [1] for `Period`'s, and for `Element`'s TIP supports `union`, `intersect`, `difference`, `overlaps`, `contains`, `length`, etc., with their expected semantics. Using these routines, we can express many temporal queries succinctly in SQL. For example, suppose we want to know who has taken Diabeta and Aspirin simultaneously, and exactly when they were taken together. This temporal self-join query can be expressed in SQL as follows:

```
SELECT p1.*, p2.*, intersect(p1.valid, p2.valid)
FROM Prescription p1, Prescription p2
WHERE p1.drug = 'Diabeta' AND p2.drug = 'Aspirin'
AND overlaps(p1.valid, p2.valid)
```

- *Aggregates:* TIP provides various aggregate functions for its datatypes. For example, `union_all` computes the union of a collection of `Element`'s and returns a single `Element` as the result. We can use `union_all` to express the temporal *coalesce* operation [4] in the following query, which computes how long each patient has been on prescription medication:

```
SELECT patient, length(union_all(valid))
FROM Prescription
GROUP BY patient
```

  Note that we cannot replace `length(union_all(valid))` with `SUM(length(valid))` in the above query. If a patient has taken multiple prescription medicines during the same period of time, `SUM(length(valid))` will count the length of this period multiple times. Therefore, all prescription periods for a patient must be temporally coalesced into one `Element` before we can compute its total length. With the help of the TIP DataBlade, we are able to express the coalesce operation without adding new language constructs to SQL.

3 Architecture and Implementation

The major components of TIP are shown in Figure 1. TIP C and Java libraries provide the core support for the TIP datatypes described in Section 2. Among the five datatypes, `Element` was the most challenging to implement, because it contains a variable number of periods whose representation could feasibly grow quite

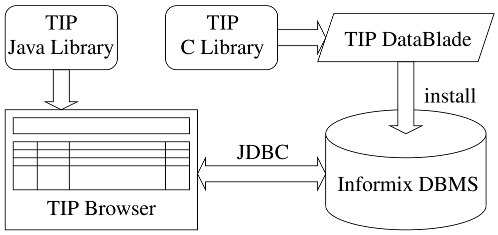

Figure 1: Architecture of TIP.

large. To implement operations on `Element`'s such as `union` and `intersect`, we use efficient algorithms that execute in time linear in the number of periods.

The TIP DataBlade in Figure 1 is the component that actually brings the temporal support into Informix. Once installed, it becomes an integral part of the DBMS; no external modules are necessary. The TIP DataBlade is written mostly in C, and makes heavy use of the TIP C library. It also contains a small amount of code written in SPL (Informix's stored procedure language). The development process was supported by tools from the Informix DataBlade Developer's Kit.

The TIP Browser is a Java client with a Swing-based GUI for querying and browsing data stored in a TIP-enabled Informix database. The TIP Browser connects to the database through JDBC. It uses customized type mapping (a new feature in JDBC 2.0) to retrieve values of TIP datatypes from the database and convert them into Java objects supported by the TIP Java library.

4 Demonstration

Our TIP demonstration (fully functional in October 1999) is based on a synthetic medical database containing various types of temporal data as in the previous examples. We demonstrate a variety of temporal queries over the database, and browse query results using the TIP Browser, as shown in Figure 2. The user may choose to browse TIP database tables and query results according to any attribute of type `Chronon`, `Instant`, `Period`, or `Element`. The value of this attribute specifies when a result tuple is considered to be "valid" for the purpose of browsing. Conceptually, there is a time window of adjustable size and position over the time line. The TIP Browser automatically highlights all result tuples that are valid in the window, and graphically displays their valid periods within the window as segments of the time line (see the rightmost column in Figure 2). A slider interface (beneath the result display area in Figure 2) lets the user move the window along the time line and visualize the distribution of the result tuples over time.

As discussed earlier, the special symbol `NOW` is interpreted as the current transaction time during query evaluation. Therefore, a temporal query may return different results when asked at different times, even if the underlying data remains unchanged. The TIP Browser lets the user enter a different value for `NOW` to override its default interpretation, which provides "what-if" analysis by allowing queries to be evaluated in a temporal context different from the present.

5 Related and Future Work

Other time-related extensions have been made to relational database systems. *Time-series* extensions are designed for capturing single attribute values that change over time. We are interested in more general-purpose temporal support, and we follow the consensus approach of TSQL2 [7] in timestamping tuples rather than attribute values. Some *spatiotemporal* extensions provide period-valued tuple timestamps, but

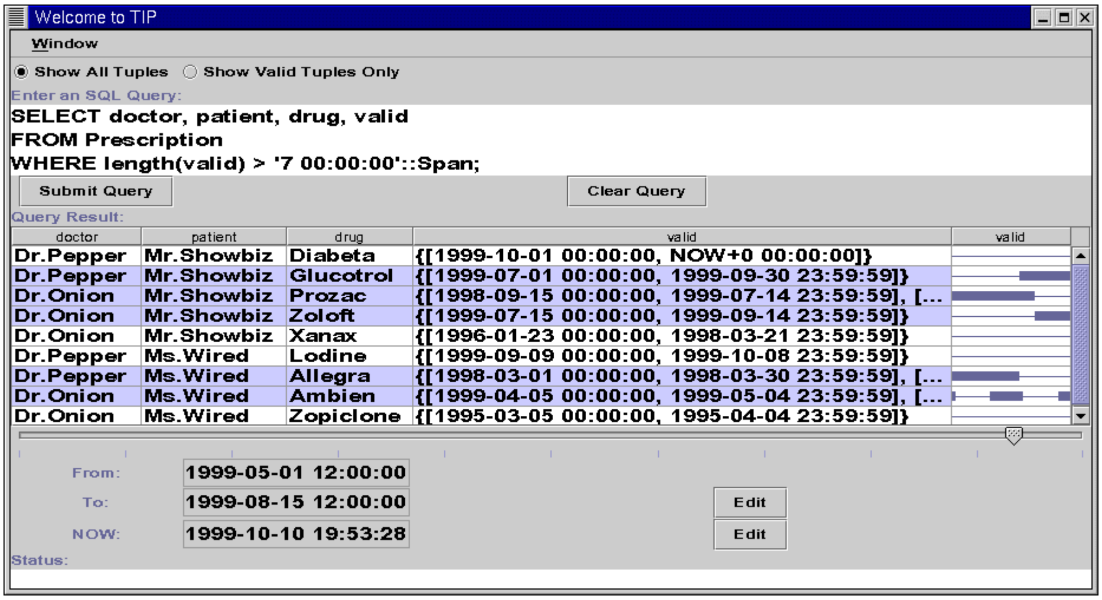

Figure 2: A screenshot of the TIP Browser.

they do not support timestamps containing `NOW` or sets of periods. Reference [2] has implemented a temporal index for period-valued tuple timestamps. Although it handles `NOW` in timestamps, it does not support general `NOW`-relative time instants or sets of periods.

Because of their inherent extensibility, object-oriented database systems provide another possible platform for temporal database extensions, and rich temporal data models such as [6] have been incorporated into these systems. We chose to implement TIP in an extensible relational DBMS instead, because relational technology and SQL are better suited to the temporal data warehousing application we plan to build using TIP [9, 10].

There are many prototype implementations of temporal database systems; see [3] for a survey. Many of them have limited practicality since they are built as small proof-of-concept research prototypes. There are a few exceptions, such as *TimeDB* and *Tiger* [8], which are implemented on top of a commercial relational DBMS. Unlike TIP, however, they use a layered approach: temporal queries are translated by an external module into standard SQL queries, which are then executed in the backend DBMS. Since the backend DBMS has little built-in temporal support, generated queries may become very complex and potentially difficult to optimize for the underlying representation of temporal data. The layered approach also complicates the development of client applications because all client requests must first go through the external module. At this point, TIP is the only temporal database implementation we know of that builds temporal support directly into an extensible commercial relational database system.

Another notable difference between TIP and other temporal database implementations is the way in which temporal queries are supported. Other implementations usually define their own temporal query languages, which differ from SQL to varying degrees. In contrast, TIP leaves SQL intact by allowing temporal queries to be expressed and evaluated using the extensive collection of predefined TIP routines. As we have seen, many temporal queries can be expressed succinctly in TIP without adding new language constructs to SQL. As future work, we will investigate how closely TIP can approach a full-featured temporal query language like TSQL2 in expressive power, while at the same time providing efficient temporal query execution through its implementation as a DBMS extension.

References

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[2] R. Bliujute, S. Saltenis, G. Slivinskas, and C. S. Jensen. Developing a DataBlade for a new index. In *Proc. of the 1999 Intl. Conf. on Data Engineering*, pages 314–323, Sydney, Austrialia, March 1999.

[3] M. H. Böhlen. Temporal database system implementations. *SIGMOD Record*, 24(4):53–60, December 1995.

[4] M. H. Böhlen, R. T. Snodgrass, and M. D. Soo. Coalescing in temporal databases. In *Proc. of the 1996 Intl. Conf. on Very Large Data Bases*, pages 180–191, Bombay, India, September 1996.

[5] J. Clifford, C. E. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the semantics of "NOW" in databases. *ACM Trans. on Database Systems*, 22(2):171–214, June 1997.

[6] I. Goralwalla, M. T. Özsu, and D. Szafron. An object-oriented framework for temporal data models. In O. Etzion, S. Jajodia, and S. M. Sripada, editors, *Temporal Databases: Research and Practice*, pages 1–35. Springer-Verlag, Berlin/Heidelberg, Germany, 1998.

[7] R. T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, Boston, Massachusetts, 1995.

[8] TimeCenter homepage. `http://www.cs.auc.dk/research/DP/tdb/TimeCenter/`.

[9] J. Yang and J. Widom. Maintaining temporal views over non-temporal information sources for data warehousing. In *Proc. of the 1998 Intl. Conf. on Extending Database Technology*, pages 389–403, Valencia, Spain, March 1998.

[10] J. Yang and J. Widom. Temporal view self-maintenance in a warehousing environment. In *Proc. of the 2000 Intl. Conf. on Extending Database Technology*, Konstanz, Germany, March 2000. To appear.