Implementation of a Temporal Database on a Conventional Database

Nain Kanwal kaur, Navneet Chaudhary, Mandeep Singh

Abstract - Researches concerning Temporal Databases are being developed for more than 20 years. However, very few implemented systems are available. Several temporal data models were proposed, extending traditional data models in a way to capture also the temporal features. A feasible way of using a temporal database is to implement the temporal database on the conventional database mapping temporal database model to it. This paper describes a temporal management system. Data and rules related to a temporal data model are managed by this system, implemented on a conventional DBMS.

Keywords - Transactions, Time, Mapping.

1. INTRODUCTION

A temporal database is a database that deals with the aspect of ‘Time’, which is an important real-world phenomenon. A wide range of real-world database applications use time-dimensions to represent their varying data. The need of representing temporal features can be identified in several application domains, like Academic, Accounting, Data warehousing, Financial, Geographical Information Systems, Insurance, Banking, etc. Temporal databases allow the storage and the recovery of all the states assumed by a object during its lifetime, thus recording its evolution with time. To provide this feature, temporal timestamps are added to data, representing the transaction time (TT – moment when the information is stored in the database) and/or the validity time (VT – time when the information is valid in the application). When both timestamps are used the result is a bitemporal database. Temporal information may be added to data in three different levels: to each attribute’s value, to a tuple, or to the whole database. Usually attributes are times tamped, but mixed representations may be used, for instance, to timestamp an object and represent it’s lifetime, the whole object (tuple) may be time stamped, besides the time stamping of it’s attributes. The implementation of a temporal database may be achieved mapping the temporal data model to a conventional database, as related in recent researches. However, little attention has been given to the temporal evolution rules of data during updates. A special strategy is required to provide the management of historic data, keeping this as independent as possible from the user. The operations managing temporal databases are the same used to manage conventional databases (insert, delete, update). However, additional operations are needed to maintain correctly historic information, usually controlled by temporal logic rules.

2. DATA MANAGEMENT TEMPORAL DATABASE SYSTEM

Data management in conventional databases is performed using the insert, update, delete and select operations. All the commercial databases implement these operations. Update and delete operations may be executed on any tuple when the user is allowed to do it. Integrity constraints maintain the database integrity. In temporal databases the same management operations (insert, update, delete and select) are used. However, their implementation is not exactly the same. In update operations the past value is not simply substituted by the new one, because the whole past of data shall be maintained. Both values keep stored in the database, time stamped with proper time values. A delete operation in temporal databases is executed as a logical exclusion. As all the past values shall be preserved, the delete operation is represented through the timestamp, ending its lifetime. Temporal databases also present integrity constraints, represented as rules that insure the temporal data consistency. These rules are different according to the type of implemented temporal database: Transaction Time Database (TTDB), Valid Time Database (VTDB) or Bitemporal Database (BTDB).

2.1 CASE STUDY : A CONVENTIONAL DATABASE

www.ijcsce.org

228
Suppose Employee is a table having personnel information in a database:

- Employee(Name, Salary, Title, BirthDate DATE)

It is easy to know the salary of an employee:

- SELECT Salary FROM Employee WHERE Name = 'John'; It is also easy to know the date of birth of an employee:
- SELECT BirthDate FROM Employee WHERE Name = 'John';

### 2.2 CONVERSION INTO A TEMPORAL DATABASE

We want to keep the employment history

- Employee(Name, Salary, Title, BirthDate, FromDate DATE, ToDate DATE)

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Title</th>
<th>BirthDate</th>
<th>From Date</th>
<th>ToDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>16000.000</td>
<td>Assistant</td>
<td>9/9/86</td>
<td>1/1/11</td>
<td>1/6/11</td>
</tr>
<tr>
<td>John</td>
<td>17000.000</td>
<td>Assistant</td>
<td>9/9/86</td>
<td>1/6/11</td>
<td>1/10/11</td>
</tr>
<tr>
<td>John</td>
<td>17000.000</td>
<td>Lecturer</td>
<td>9/9/86</td>
<td>1/10/11</td>
<td>1/2/12</td>
</tr>
<tr>
<td>John</td>
<td>17000.000</td>
<td>Professor</td>
<td>9/9/86</td>
<td>1/2/12</td>
<td>1/1/13</td>
</tr>
</tbody>
</table>

For the data model, new columns are identical to attribute BirthDate

Determine the Salary: To know the employee’s current salary, things are more difficult.

- SELECT Salary FROM Employee WHERE Name = 'John' AND FromDate <= CURRENT TIMESTAMP AND CURRENT TIMESTAMP <= ToDate;

Determine the salary history: For each person, the maximal intervals of each salary.

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>FromDate</th>
<th>ToDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>16000.000</td>
<td>1/1/11</td>
<td>1/6/11</td>
</tr>
<tr>
<td>John</td>
<td>17000.000</td>
<td>1/6/11</td>
<td>1/1/13</td>
</tr>
</tbody>
</table>

An employee could have arbitrarily many title changes between salary changes

**SQL CODE:**

a. For Updating:
CREATE TABLE Temp(Salary, FromDate, ToDate) AS SELECT Salary, FromDate, ToDate FROM Employee WHERE Name = 'John';

repeat
UPDATE Temp T1 SET (T1.ToDate) = (SELECT MAX(T2.ToDate) FROM Temp AS T2 WHERE T1.Salary = T2.Salary AND T1.FromDate < T2.FromDate AND T1.ToDate >= T2.FromDate AND T1.ToDate < T2.ToDate) WHERE EXISTS (SELECT * FROM Temp AS T2 WHERE T1.Salary = T2.Salary AND T1.FromDate < T2.FromDate AND T1.ToDate >= T2.FromDate AND T1.ToDate < T2.ToDate) until no tuples updated.

b. For deleting: DELETE FROM Temp T1 WHERE EXISTS (SELECT * FROM Temp AS T2 WHERE T1.Salary = T2.Salary AND (T1.FromDate > T2.FromDate AND T1.ToDate <= T2.ToDate) OR (T1.FromDate >= T2.FromDate AND T1.ToDate < T2.ToDate))

### 3. IMPLEMENTING TEMPORAL DATABASES ON TOP OF CONVENTIONAL DBMS

Due to the fact that a completely temporal DBMS is not yet available, the implementation of a temporal database may be done on top of a conventional database, providing a suitable mapping between the two data models. As the conventional DBMS does not behave as a temporal DBMS, the operation management shall be adapted. Operation implementation depends on the underlying database resources, and on the possibility of representing temporal evolution rules to control data integrity.

According to the temporal data model to be implemented and to the DBMS available resources, three different implementation levels are identified. The first level (Level 1) DBMS do not support rules to control data integrity, which must be implemented by application programs. In Level 2 the DBMS support rules, but there are still some
features of the data model, such as the messages in object-oriented models, must be implemented through applications programs. And in Level 3 the DBMS support every feature of the data model to be implemented. Several experiences mapping temporal data models to conventional databases have been developed, showing the feasibility of this approach. However, analyzing the realized experiences, it can be realized that not all the features of the temporal data model are conveniently represented. Relational databases are the most popular commercial databases nowadays, and therefore they are the most often used to implement temporal databases. Special attributes are used to present the timestamps (TT, VT, or bitemporal). Object-oriented databases have also been used to implement temporal databases. All these implementations focus only on the mapping of the data model and of the stamps to the underlying database. The Temporal Management System presented in this paper implements a temporal data model on top of a conventional database. A relational database was selected due to the fact of being largely used in practice.

4. THE TF-ORM DATA MODEL

TF-ORM (Temporal Functionality in Objects with Roles Model) is a temporal object-oriented data model. This model differs from other temporal object-oriented data models by the use of the role concept. Several roles may be defined for each class, expressing different behaviors that an object of this class can present. Instances of these roles are dynamically instantiated during the existence of an object. As an example, a person in a hospital (represented as an instance of a class Person) may play the roles of patient, doctor, nurse, etc. An object is still an instance of only one class, but it can play different roles during its lifetime (the same person may evolve from the nurse role to the doctor role). The same object may also present two or more roles at the same time (a doctor may also be a patient of the hospital). In addition, an object can have more than one instance of the same role at the same time (as when a doctor works in two different hospitals). Three different class types can be defined: (i) resource classes, modeling information and resources; (ii) process classes, representing the processes to be executed with this information and the resources; and (iii) agent classes, representing the persons that carry out the processes.

The three class types are modeled in a similar way. Each role is defined by:

- A name, unique in the corresponding class.
- Static properties (having the same value all over the instances lifetime) and dynamic properties (when they may assume different values with time), the last being timestamped with transaction and validity time.
- A set of abstract states the role can assume while playing this role.
- The messages the role can receive or send. Human decisions are represented in agent classes as incoming messages.
- State transition rules and constraint rules that define the dynamic evolution of an object.

These rules model the temporal behavior of an object playing a specific rule, and must be analyzed by the DBMS during data definition.

5. MAPPING AND MANAGING TF-ORM ON ORACLE

The need of implementing a DBMS supporting TF-ORM led to the solution of using a conventional database, mapping the temporal model to it, and managing the data definition according to the TF-ORM rules. This section describes how a TF-ORM manager was implemented on the conventional database Oracle. The choice of Oracle was defined by the possibility of supporting the TF-ORM rules, due to the existence of special mechanisms like triggers, stored procedures, stored functions and views. Besides this, Oracle is a very popular commercial database. The use of a relational database to implement an object-oriented database is not a new solution, requiring only a suitable mapping between the two different data models. Another important characteristic of Oracle is the already available manipulation of dates, necessary to implement a temporal data model. As TF-ORM uses both the transaction and the validity times as timestamps, a bitemporal database is implemented. To simplify the query evaluation, temporal intervals were used as timestamps in place time points. No restriction of temporal definition was made (past, present and future times may be defined). The operation management is executed by specific procedures that update the database according to the manipulated tables, using triggers to prevent user actions.

CONCLUSION
Researches in temporal database implementation showed that such databases might be implemented on top of conventional databases, mapping the temporal features to explicit attributes representing time. Some experiences were done implementing the TF-ORM data model on different databases. This paper presents the implementation of a TF-ORM manager, on top of the conventional database Oracle. Only the relational features of Oracle were used. This decision was based on the fact that relational databases are largely used, leading thus to a system with a great flexibility, which can be adapted to other relational databases with very little effort.

The presented TF-ORM manager is part of an integrated environment implementing TF-ORM. This environment includes not only the already mentioned specification tool, but also a query tool, that translates queries expressed in the TF-ORM query language to SQL.

REFERENCES


[8] Georgia Garani’s “Temporal Databases”.

[9] Hatem Halaoui’s “Modeling Spatio-Temporal Databases”.


www.ijcsce.org