A Proposal Model to Incorporate Temporal Aspects to the Relational DBMS Using Active Databases Concept

Eugênio de Oliveira Simonetto
Pontifícia Universidade Católica do Rio Grande do Sul – PUCRS
Campus II - FACI - Departamento de Informática – Uruguaiana – RS – Brazil
E-mail: eugenio@pucrs.campus2.br

Duncan Dubugras Alcoba Ruiz
Pontifícia Universidade Católica do Rio Grande do Sul – PUCRS
Faculdade de Informática - Porto Alegre – RS – Brazil
E-mail: duncan@inf.pucrs.br

Abstract
This work presents a proposal model to incorporate temporal aspects in relational databases taking into account the present functionality offered by the most advanced relational DBMS, e.g. Oracle, Sybase and Ingres, at the active database level (ECA standard). The model takes into consideration two directives. First, the implementation of the temporal aspects of the application information, into the database specification model, must be done automatically. Second, the way to manipulate the application data must be the same as a database application model without the implementation of the temporal aspects by those users who do not need to (or should not) know about the temporal aspects of the database. It has been made a study about (a) which temporal aspects could be implemented following the directives above and (b) which implementation techniques could be employed with the use of triggers and types of temporal data on the data model of Ingres DBMS.

Keywords: Temporal Databases, Active Databases, Data Modelling

1. Introduction

It is not recent the necessity of supporting data temporal characteristics on databases. Many different works have been developed, not only at data modelling level (e.g. [1]), but also at database implementation level (e.g. [2]). There is an effort to standardise it (see TSQL2 [3]). However, the most popular DBMS, e.g. Oracle, Sybase and Ingres, offer very little to support temporal data.

It has fundamental importance the necessary acquaintance of the temporal database implementation model for a given application with a subset of application users who see it as an instantaneous data model with only snapshot relations (according to [4] terminology, “relations of a conventional relational database system incorporating neither valid-time nor transaction-time timestamps”). It becomes more evident whenever we realise the volume of developed application programs that uses existing application databases, and these applications need to support temporal aspects of their data. Since the processes of conversion, migration or even reengineering an information system (IS) is very difficult, the solutions which makes possible the IS evolution, step by step and without significant discontinuity, could be of great value. In [5] Brodie & Stonebraker argue in this sense, on the reengineering context of legacy system.

This work proposes a way of incorporating temporal aspects to relational DBMS taking into consideration the present functionality offered by them. The objective is to turn viable the temporal data treatment without interfering on the users’ work, which does not need to (or should not) know

---

1 The work was partially supported by FAPERGS (Project SIL-TM, Project number: 96/1532.9).
about these temporal aspects on the database. For these reasons the solution automates the update of the temporal parcel of data from the database.

The work is organised as follows. On section 2, we discuss some aspects for time representation on the model. Section 3 defines the different types of users for the proposal model. The data model is presented on section 4 and, on section 5, we present our conclusions.

2. Time Representation on the Model

For the representation of temporal data aspects, we use the concept of database state [7], where each state is defined by their attribute values. Any update done on the value of an attribute creates a new state in the database. Each state has a defined duration that is represented by a time interval in which no update on the attribute value was made. For this reason, each instance of the database is labelled with a temporal interval.

Based on the possible ways of data temporal representation in a database – valid time, transaction time and both of them (see [4] for more details) – we conclude that in the model being proposed the temporal aspects should be represented by transaction time [4]. By using this time line, the temporal data would not need to be manipulated directly by the user, since the DBMS uses the system time. In order to represent the transaction time two special attributes are used: the start of the transaction period (S_TT) and the end of the transaction period (E_TT). In order to validate the proposed model, the time format DD/MM/YY HH:MM:SS has been used. In a real application, the time granularity is defined by the application designer, and may be different of our choice.

On temporal databases, any updating operation should be treated on a different way from the conventional database updates, since they should guarantee that no data, which becomes older, could be lost. All updates should be followed by operations to maintain the old values. For these, updating routines that automatically update the transaction times were developed. These routines became totally transparent to the users of the database.

3. Types of Users

The temporal database implementation using a relational DBMS asks for a specific strategy for its representation in order to turn the managing of the historical data independent from the user intervention or the programmer applications. For being a temporal database, the model should keep all the history of transactions done on the database. These historical data do not concern to the whole contingent of users, but to a part of them who needs to know the temporal aspects of the information. We classify the users of the temporal database into two types: conventional users and temporal users.

The conventional users view the database as an instantaneous database where each update in an attribute value corresponds to a transaction on the database. For this kind of user, in this transaction the stored value is destroyed and only the last value is available. The table 1 shows a possible content of a relation example.

On the other hand, the temporal users have access to all the values stored (previous and current) in the database and, for this reason, to all tables that implement the temporal database. Following the same example showed on table 1, tables 2, 3 and 4 show examples of PERSONS relation, implemented by three different relations: Static_Table, Temporal_Name and Temporal_Address. It is considered that BirthDate is an attribute without temporal variation (static attribute); Name and Address are attributes with temporal variation, and different temporal variations between them. As will be seen in section 4, the Static_Table stores the transaction time of each record, and the temporal tables store the transaction time of each attribute value.
Table 1. Example of data viewed by a conventional user

<table>
<thead>
<tr>
<th>#Person</th>
<th>Name</th>
<th>BirthDate</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Charles</td>
<td>25/01/75</td>
<td>Ipiranga, 24/101</td>
</tr>
<tr>
<td>0002</td>
<td>Antony</td>
<td>30/12/67</td>
<td>Silva, 48</td>
</tr>
</tbody>
</table>

Table 2. Example of Static_Table contents

<table>
<thead>
<tr>
<th>#Person</th>
<th>BirthDate</th>
<th>S_TT</th>
<th>E_TT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>25/JAN/1975</td>
<td>01/08/96 14:00:30</td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>30/DEC/1967</td>
<td>21/07/95 15:30:00</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Example of Temporal_Name contents

<table>
<thead>
<tr>
<th>#Person</th>
<th>Name</th>
<th>S_TT</th>
<th>E_TT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Charles</td>
<td>01/08/96 14:00:30</td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>Antony</td>
<td>21/07/95 15:30:00</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Example of Temporal_Address contents

<table>
<thead>
<tr>
<th>#Person</th>
<th>Address</th>
<th>S_TT</th>
<th>E_TT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Garibaldi, 23/1</td>
<td>01/08/96 14:00:30</td>
<td>10/01/97 12:10:10</td>
</tr>
<tr>
<td>0001</td>
<td>Ipiranga, 24/101</td>
<td>10/01/97 12:10:10</td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>Silva, 48</td>
<td>21/07/95 15:30:00</td>
<td></td>
</tr>
</tbody>
</table>

4. The Data Model

The data model was developed in a way where the temporal aspects are “transparent” to the conventional user of the database. After the design of the conceptual model for a given application, we defined each relation taking into account only the attributes that would be represented in the database, without the presence of the attributes referring to the time aspect (S_TT and E_TT), which was called Instantaneous_Table. In this table, all the attributes not included in primary key are totally dependent on the primary key and all attributes are mutually independent, in terms of functional dependencies. It characterises a table normalised on the third normal form (3NF) [8]. After this table definition, its attributes are classified as static (which does not need to maintain the previous values) and temporal (the ones which is important to maintain the current and the previous values). According to this classification, the attribute will be treated on different ways. For the attributes classified like static it is defined one only table, called Static_Table, which contains the primary key attributes of the Instantaneous_Table, all the static attributes plus the additional attributes which refer to the transaction time (S_TT and E_TT). For each temporal attribute of the Instantaneous_Table, or each set of temporal attributes with the same temporal variation, it is defined a new table, with the primary key attributes of the Instantaneous_Table, the temporal attribute (or set of temporal attributes) plus the additional attributes which refer to the transaction time (S_TT and E_TT). The primary key for the Static_Table, and for each table generated to support each temporal attribute (or each set of temporal attributes), is the attribute, or attributes, from the primary key of Instantaneous_Table plus the special attribute S_TT.
The database users (conventional and temporal) will use only the *Instantaneous_Table* for their updates, without the need to know (nor interfere) that the temporal data will be updated automatically on the tables where they are stored, together with the temporal labelling of the fact on the database (transaction time). The temporal user has access to all the system tables, but only for inquiry purpose. This alternative, modality *Instantaneous_Table*, was validated using the Ingres DBMS. The type of the relational schema resulting from the model is represented on the table 5.

<table>
<thead>
<tr>
<th>Table 5. Type of the relational schema for the proposal model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instantaneous_Table</strong> (&lt;key1,..., keyn&gt;, AtStat1,..., AtStatn, AtTemp1,..., AtTempn);</td>
</tr>
<tr>
<td><strong>Static_Table</strong> (&lt;key1,..., keyn&gt;, AtStat1,..., AtStatn, S_TT, E_TT);</td>
</tr>
<tr>
<td><strong>Temp1_Table</strong> (&lt;key1,..., keyn&gt;, AtTemp1, S_TT, E_TT);</td>
</tr>
<tr>
<td>.....</td>
</tr>
<tr>
<td><strong>Tempn_Table</strong> (&lt;key1,..., keyn&gt;, AtTempn, S_TT, E_TT);</td>
</tr>
<tr>
<td>Note: the underlined attributes compose the primary key of each relation.</td>
</tr>
</tbody>
</table>

Where:
- *key1,..., keyn* - are attributes which make part of the primary key of the *Instantaneous_Table*;
- *AtStat1,..., AtStatn* - are static attributes which do not vary according to the time;
- *AtTemp1,..., AtTempn* - are temporal attributes;
- *S_TT* - is the attribute which represents the start of a temporal interval, e.g. the moment on which the information was included on the database;
- *E_TT* - is the attribute that represents the end of a temporal interval, e.g., the moment when the information becomes not valid to the database.

On the static and temporal tables the primary key is made be the attributes *key1,..., keyn* and *S_TT*.

The structure of the data model on DBMS Ingres, used to validate, presents the form showed on table 6.

<table>
<thead>
<tr>
<th>Table 6. Structure of the data model on Ingres DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>create table <strong>Instantaneous_Table</strong> &lt;Data type&gt;</td>
</tr>
<tr>
<td>(key &lt;Data type&gt;</td>
</tr>
<tr>
<td>AtStat1 &lt;Data type&gt;</td>
</tr>
<tr>
<td>AtTemp1 &lt;Data type&gt;</td>
</tr>
<tr>
<td>Primary key (key));</td>
</tr>
<tr>
<td>create table <strong>Static_Table</strong> &lt;Data type&gt;</td>
</tr>
<tr>
<td>(key &lt;Data type&gt;</td>
</tr>
<tr>
<td>AtStat1 &lt;Data type&gt;</td>
</tr>
<tr>
<td>S_TT date not null with default ‘now’</td>
</tr>
<tr>
<td>E_TT date,</td>
</tr>
<tr>
<td>Primary key (key, S_TT));</td>
</tr>
<tr>
<td>create table <strong>Temp1_Table</strong> &lt;Data type&gt;</td>
</tr>
<tr>
<td>(key &lt;Data type&gt;</td>
</tr>
<tr>
<td>AtTemp1 &lt;Data type&gt;</td>
</tr>
<tr>
<td>S_TT date not null with default ‘now’</td>
</tr>
<tr>
<td>E_TT date,</td>
</tr>
<tr>
<td>Primary key (key, S_TT));</td>
</tr>
</tbody>
</table>
4.1 Data Updating

On the data model, the user works always with the *Instantaneous Table*, e.g. all inserts, updates and deletes are made directly on this table. The routines used to the manipulation of the transaction time (on the static and temporal tables) were done using active rules of the DBMS [9].

4.1.1 The INSERT operation

Each time the user inserts a new fact on the table of the instantaneous data, the routine responsible by the insertion will insert each attribute classified as *temporal* on its respective table, and the *static* attributes will be inserted on the Static_Table. It will also be inserted the time in which the new record was stored on the database, not only on the temporal tables but also on the static one. Time referring to the insertion of fact on the database is represented by S_TT. Insertion routine is exemplified on Ingres on table 7.

Table 7. Structure of the insertion routine on Ingres

<table>
<thead>
<tr>
<th>```sql</th>
</tr>
</thead>
<tbody>
<tr>
<td>create rule insertion after insert into Instantaneous_Table execute procedure insertion (key = new.key, AtStat1 = new.AtStat1, AtTemp1 = new.AtTemp1); create procedure insertion (key &lt;Data Type&gt;, AtStat1 &lt; Data Type &gt;, AtTemp1 &lt; Data Type &gt;) as Begin insert into Static_Table(key,AtStat1) values (:key,:AtStat1); insert into Temp1_Table(key,AtTemp1) values (:key,:AtTemp1); end;`</td>
</tr>
</tbody>
</table>

The generalisation of insertion routine above is obtained by:

1. passing as parameter to the insertion routine (Insertion) all the set of attributes of the table.
2. expanding the reference to key by key1,..., keyn
3. including the command insert of Static_Table to all the static attributes:
   ```sql
   insert into Static_Table (ke1,..., keyn, AtStat1,...,AtStatn) values (:ke1,..., :keyn,:AtStat1,..., :AtStatn);
   ```
4. including commands insert to each temporal attribute:
   ```sql
   insert into Table_Temp1 (ke1,..., keyn, AtTemp1) values (:ke1,..., :keyn, :AtTemp1);
   ...
   insert into Table_Tempn (ke1,..., keyn, AtTempn) values (:ke1,..., :keyn, :AtTempn);
   ```

4.1.2 The UPDATE operation

On temporal database, whenever the user changes the value of an attribute, he/she will be putting an end to its history, and inserting a new value that will give start to a new history on the database. The routine responsible for the *update* operation sets a value ‘NOW’ (the moment value on Ingres DBMS) to the E_TT (end of the interval of the information validity) on the tuple of the temporal table, where the attribute is stored. Thus, generates a new tuple with a new attribute value, assigning the value ‘NOW’ to S_TT (start of the information validity interval).

When there is an update on the *static* attribute, the correspondent routine only changes the value on the Static_Table, without changing the value on S_TT (start of validity). Keeping, then, the same transaction time as the one when the fact was initially inserted on the database. The routines of update are exemplified on table 8.
Table 8. Structure of the update routines on Ingres

```sql
create rule updating
  after update(AtStat1) of Instantaneous_Table
execute procedure updating(key=new.key, AtStat1=new.AtStat1);

create procedure updating
  (key <Data Type>, AtStat1 <Data Type>) as
begin
  update Static_Table set AtStat1=:AtStat1 where key =:key and E_TT is null;
end;

create rule update_AtTemp1
  After update (AtTemp1) of Instantaneous_Table
execute procedure update_AtTemp1 (key = new.key, AtTemp1 =new.AtTemp1) ;

create procedure update_AtTemp1
  (key <Data Type>, AtTemp1 <Data Type> ) as
begin
  update Temp1_Table set E_TT='now'
  where key =:key and E_TT is null;
  insert into Temp1_Table (key,AtTemp1) values (:key, :AtTemp1);
end;
```

A generalisation of the update routines above is the following.
(1) expanding the reference to key through key1,...,keyn
(2) including all of the static attributes on the update statement
    update Static_Table set AtStat1=:AtStat1,..., AtStatn=:AtStatn
    where key1 =:key1 and ... and keyn =:keyn and E_TT is null;
(3) creating a pair of rule-procedure for each temporal attribute: update_AtTemp1 ... update_AtTempn.

4.1.3 The DELETE operation

On temporal database, the delete operation does not remove a record of database, but keeps stored the attribute values of the excluded record associated to the temporal labelling.

On the proposed model, whenever we exclude a record of a database, the routine responsible for the delete operations will attribute value ‘NOW’ to the attributes E_TT on the Static_Table and in all temporal tables where the record up to that moment was valid. On the Instantaneous_Table, the record is really removed. The routine for delete records is showed on table 9.

Table 9. Structure of the delete routines on Ingres

```sql
create rule deletion
after delete of Instantaneous_Table execute procedure deletion(key=old.key);

create procedure deletion (key <Data Type>) as
begin
  update Static_Table set E_TT = 'now' where key= :key and E_TT is null;
  update Temp1_Table set E_TT = 'now' where key= :key and E_TT is null;
end;
```

The generalisation of the deletion routine above is the following.
(1) expanding the reference to key through key1, ... , keyn,
(2) including update commands to each temporal attribute:
    update Temp1_Table set E_TT = ‘now’
    where key1 = :key1 and ... and
    keyn = :keyn and E_TT is null;
    ...
    update Tempn_Table set E_TT = ‘now’
    where key1 = :key1 and ...
    keyn = :keyn and E_TT is null;
5. Concluding Remarks

The merit of this work is to present a way of how to incorporate temporal aspects on the more advanced relational DBMS, following the objectives of: (1) to turn automatic the time maintenance, and (2) not to interfere on the visualisation form of the database implementation model, on the conventional users part (who do not need or should not take into account the database temporal aspects). The solution got on DBMS Ingres, and the generalisation presented, can be accomplished on any DBMS that offer triggers, of similar form to triggers of Oracle and Sybase, or to Ingres rules.

5.1 Performance

Our proposal model may imply on an excessive use of disk space, resulting from the replication of data between the Instantaneous_Table and the others (Static_Table and tables of the temporal attributes). In fact, our original proposal was defined the Instantaneous_Table as a view. But none of the DBMS examined permit updating through views if the view is defined as join of relations. Considering the structure of the data model showed on table 6, the view definition would have the format:

```sql
create view Instantaneous_Table (key, AtStat1, AtTemp1) as
select Static_Table.key, AtStat1, AtTemp1
from  Static_Table, Temp1_Table
where Static_Table.key = Temp1_Table.key and
Static_Table.E_TT is null and
Temp1_Table.E_TT is null;
```

In fact, there wasn’t ambiguity for the propagation of the updates from the view to other tables, because each value from key there would be only one tuple on Static_Table with the same value for key and null value for E_TT. This special situation occurs also on tables that support temporal attributes. We believe the commercial DBMS will permit updates through views in a near future.

5.2 Future Work

We intend to evaluate the real impact of the data replication on the performance of the application databases, using different DBMS. Today, we are using the Oracle DBMS, mainly the clustering options. Also, there is research work being done in PUCRS to represent more semantics of applications, mainly for the dynamic aspects of them, using active rules of databases.

6. References