EXTENDING ATSQL TO SUPPORT TEMPORALLY DEPENDENT INFORMATION

Carme Martín¹, Michael H. Böhlen² and Carlos López¹

1: Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Jordi Girona Salgado 1-3. E-08034 Barcelona, Catalonia
e-mail: martin@lsi.upc.edu, web: http://www.lsi.upc.edu/-tiger/

2: Faculty of Computer Science,
Free University of Bozen-Bolzano
Piazza Domenican 3. I-39100 Bozen-Bolzano, Italy
e-mail: boehlen@inf.unibz.it, web: http://www.unibz.it/inf/

Key words: Temporal databases, Temporally dependent attributes, SQL, ATSQL

Abstract. In temporal databases some attribute values depend on the associated timestamps while other attribute values are independent of them. An attribute is temporally independent if it has a value for a given time period and if this value also holds for periods contained in the original period. In contrast, the value of a temporally dependent attribute holds for the given time period only. Different values hold for its sub-periods. The completed database provides information for each sub-period of a database fact. In this paper we extend ATSQL to support temporally dependent information and report experiences from a prototype implementation

1. INTRODUCTION

Many business applications keep track of the history of business facts. For instance this is the case for a wide-ranging class of real-time measuring and sensor systems. Such systems measure and report data. Some systems report a single value that changes continuously, while other systems report data less frequently and in a summarized form. Ultimately the data is stored in some consolidated, possibly aggregated, form in a database.

Often, the measurement history needs to be analyzed for decision making purposes, and information for non-existing time periods has to be derived. This so-called database completion faces two inherent problems. First, time-varying information exhibits different properties with respect to the associated timestamp. A temporally independent attribute does
not change if it is associated with a sub-period of the original timestamp. A classical example is the name, which usually holds for the entire lifespan of a person as well as for any sub-period of the lifespan. In contrast, a temporally dependent attribute changes if it is associated with a sub-period of the original timestamp. For instance a loyalty bonus is associated with a 10 year period but not with any sub-period thereof.

Besides information for sub-periods of database facts it is also relevant to derive information for time periods with time points for which no information is recorded. We refer to this process as the covering of information gaps. An information gap is the time period for which no information is recorded. Information gaps are covered by considering information from before or after the gap.

Note that the completion and covering must be parametric. For instance if we have a measurement for the morning and a measurement for the evening we have different options for the value for noon. The precise value depends on a number of parameters and in many cases is application dependent. Our goal is a parametric approach with built-in support that allows applications to complete and cover a database as required by the application.

The main contribution of this paper is to extend the temporal query language ATSQL [2] and describe a prototype implementation, using the temporal database system Tiger, that supports temporally dependent information and offers operations to handle completed and covered databases. The completed database is a database, which provides information for all possible time sub-periods. A covered database is the union of the explicit database information and the information for database information gaps. Temporally dependent attributes add two implicit columns with formulas for completing and covering a database.

The paper is organized as follows. The next section presents the basic framework. Section 3 defines the extensions to support temporally dependent attributes. Section 4 formalizes completed database operations in the temporal language ATSQL and explains some implementation issues. Finally, section 5 offers conclusions and directions for future work.

2. FRAMEWORK

In this section, we introduce the temporal language ATSQL and the temporal database system Tiger.

2.1. ATSQL

ATSQL is a temporal extension of the standard language SQL. A main characteristic of ATSQL is to guarantee that replacing the existing non-temporal DBMS with a new temporal DBMS does not affect the functioning of any application code, and a systematic generalization of non-temporal SQL statements to corresponding temporal statements over temporal tables. The main desiderata to facilitate the transition to a temporal DBMS are: upward compatibility, temporal upward compatibility and snapshot reducibility. We briefly introduce ATSQL with an example of an employees relation with attributes Code, Name, Salary and Dpt. This example will be used throughout the paper. We consider only the valid time dimension. Valid time is the time when the fact is true in the modelled reality [6].
EXAMPLE. Creation of the employees SQL-92 table:

```sql
CREATE TABLE employees ( 
    Code INTEGER PRIMARY KEY, 
    Name VARCHAR(32), 
    Salary REAL, 
    Dpt INTEGER);
```

Insertion of the employee Tom in the employees table:

```sql
INSERT INTO employees VALUES (1, 'Tom', 1000, 1);
```

Converting the employees table into a valid time table. The statement is executed on 2005/01/01.

```sql
ALTER TABLE employees ADD VT;
```

Notice that after converting the non-temporal table into a temporal table a new VT column is added. The VT column captures a valid time period with a starting valid time point and an ending valid time point. Starting valid time is 2005/01/01, the time when the insertion is executed. In valid time dimension, where the current time inexorably advances, the concept of Now [3] is used to model the current time. As we do not yet know the ending valid time we use Now.

Deletion is always a logical deletion. The statement is executed on 2005/01/15 and Now is changed to 2005/01/15.

```sql
DELETE FROM employees WHERE Code=1;
```

For a new insertion into the employees valid time table, the SET VALID modifier allows to explicitly specify the valid time period of the inserted fact.

```sql
SET VALID PERIOD '2005/01/15 - 2005/01/30'
```

```sql
INSERT INTO employees VALUES (1, 'Tom', 1000, 2);
```

Standard SQL-92 queries access the current state and yield a non-temporal result:

```sql
SELECT * FROM employees;
```

To exploit temporal information SEQUENCED and NONSEQUENCED statements must be
used. **SEQUENCED** statements provide powerful built-in support for statements that are based on snapshot-reducibility [7]. **NONSEQUENCED** statements account for temporal statements that require the explicit specification of temporal relationships (**PRECEDES**, **DURATION**, ...).

### 2.2. Tiger

Tiger has been designed and developed to promote and further the understanding of temporal databases in general and ATSQL, a temporal extension of SQL, in particular. The main goals were to exploit existing database technology and to achieve substantial support for temporal requests with little syntactic overhead. Tiger is a period-timestamped bitemporal database system running as a front-end, written in SWI Prolog, to the relational database system Oracle. It implements selected aspects of the temporal language ATSQL. Temporal requests in ATSQL are compiled into SQL-commands that are executed by the Oracle database backend. A detailed description of the module structure of Tiger can be found at [http://wwwlsi.upc.edu/~tiger/test/system.php](http://wwwlsi.upc.edu/~tiger/test/system.php).

### 3. TEMPORALLY DEPENDENT ATTRIBUTES

In this section, we propose small ATSQL extension that permits temporally dependent attributes. The extensions preserve the properties of ATSQL.

We assume that time is a linear and discrete set of time points isomorphic to the natural numbers [4]. We assume a time period encoding with periods that are closed at their lower and open at their upper end [8].

A temporally independent attribute has a value for a given time period, and this value also holds for periods contained into the original period. To denote temporally independent attributes the keyword **IND** is used. The value of a temporally dependent attribute holds for the given time period only. The values for the periods contained into the original period must be derived. To denote temporally dependent attributes the keyword **DEP** is used.

**EXAMPLE.** Assume we want to make **Salary** a temporally dependent attribute and **Code**, **Name** and **Dpt** temporally independent attributes. Attributes are temporally independent by default. The new clause to incorporate temporally dependent attributes is

```
ALTER TABLE employees MODIFY Salary ADD DEP;
```

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Salary</th>
<th>FCov(Salary)</th>
<th>FSub(Salary)</th>
<th>Dpt</th>
<th>VT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>fleft</td>
<td>fconst</td>
<td>1</td>
<td>1-15</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>fleft</td>
<td>fconst</td>
<td>2</td>
<td>15-30</td>
</tr>
</tbody>
</table>

The new **ADD DEP** clause adds two implicit columns to the table with the formulas for covering information gaps (**FCov**) and for completing sub-periods (**FSub**). Initially, the values of **FCov(Salary)** and **FSub(Salary)** are the defaults defined by the database system. The internal column names for **FCov** is **Salary$$C** and for **FSub** is **Salary$$S. These internal columns are not directly accessible to the user as indicated by the typesetting.

The definition of the functions **fleft** and **fconst** are stored in the table formula with the following structure:
CREATE TABLE formula (
    Name CHAR(30) PRIMARY KEY,
    Form VARCHAR(300)) ;

<table>
<thead>
<tr>
<th>Name</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>flef</td>
<td>*((valref,dist(vtsref,vteref)),dist(vtsnew,vtenew))</td>
</tr>
<tr>
<td>fcon</td>
<td>*((valref,dist(vtsref,vteref)),dist(vtsnew,vtenew))</td>
</tr>
</tbody>
</table>

The formulas are specified using prefix notation and can include basic arithmetic operations and standard operations such as abs, random, truncate, sin, cos, log, ..., and dist to calculate the duration (length) of a time period. Several predefined expressions are available: vtsref (reference valid time starting period), vteref (reference valid time ending period), valref (value of the reference valid time period), vtensw (new valid time starting period), vtenew (new valid time ending period), vtsnext (next valid time starting period), vtenext (next valid time ending period) and valnext (value in the next valid time period).

Covering formulas can use values from left and/or right tuples. This implies the next tuple from left is the previous tuple from the right case. Figure 1 illustrates predefined expressions from left and right cases.

```
Figure 1. Predefined expressions to make formulas
```

Formula flef uses the current tuple to cover an information gap to the right. Formula fconst generates the values for the sub-periods of the current tuple. The definition of both formulas is the same, the difference is that (see figure 1, from left) in flef: vtsnew is vteref and vtenew is vtensw, and in fconst: vtsnew is vtsref and vtenew is vteref.

**EXAMPLE. Insertion of the new formula fright:**

```sql
    INSERT INTO formula
    VALUES ('fright', '*(valnext,dist(vtsnext,vtenext)),dist(vtsnew,vtenew))') ;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>flef</td>
<td>*((valref,dist(vtsref,vteref)),dist(vtsnew,vtenew))</td>
</tr>
<tr>
<td>fcon</td>
<td>*((valref,dist(vtsref,vteref)),dist(vtsnew,vtenew))</td>
</tr>
<tr>
<td>fright</td>
<td>*((valnext,dist(vtsnext,vtenext)),dist(vtsnew,vtenew))</td>
</tr>
</tbody>
</table>

To change the default formulas for new temporally dependent attributes we can use the CHANGE DEP statement:
CHANGE DEP employees FCOV(Salary)=’fright’;

Now, a new insertion will yield the following result:

```
INSERT INTO employees VALUES (2, ’Matt’, 2000, 1);
```

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Salary</th>
<th>FCOV(Salary)</th>
<th>FSUB(Salary)</th>
<th>Dpt</th>
<th>VT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>left</td>
<td>fconst</td>
<td>1</td>
<td>1-15</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>left</td>
<td>fconst</td>
<td>2</td>
<td>15-30</td>
</tr>
<tr>
<td>2</td>
<td>Matt</td>
<td>2000</td>
<td>fright</td>
<td>fconst</td>
<td>1</td>
<td>15-Now</td>
</tr>
</tbody>
</table>

We can update formulas as follows:

```
DEP UPDATE employees SET FCOV(Salary)=’fright’ WHERE Dpt=2;
```

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Salary</th>
<th>FCOV(Salary)</th>
<th>FSUB(Salary)</th>
<th>Dpt</th>
<th>VT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>left</td>
<td>fconst</td>
<td>1</td>
<td>1-15</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>left</td>
<td>fconst</td>
<td>2</td>
<td>15-30</td>
</tr>
<tr>
<td>2</td>
<td>Matt</td>
<td>2000</td>
<td>fright</td>
<td>fconst</td>
<td>1</td>
<td>15-Now</td>
</tr>
</tbody>
</table>

We can create a valid time table directly, as in ATSQL, but with the explicit specification of temporally independent and dependent attributes:

```
CREATE TABLE departments(
    Code INTEGER IND,
    Benefit FLOAT DEP,
    Name VARCHAR(15) IND) AS VALID;
```

Again, the default formulas for completing and covering can be changed:

```
CHANGE DEP FCOV(Benefit)=’fright’, FSUB(Benefit)=’fconst’;
```

```
INSERT INTO departments VALUES (1, 100.000, ‘Sales’);
```

<table>
<thead>
<tr>
<th>Code</th>
<th>Benefit</th>
<th>FCOV(Benefit)</th>
<th>FSUB(Benefit)</th>
<th>Name</th>
<th>VT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100.000</td>
<td>fright</td>
<td>fconst</td>
<td>Sales</td>
<td>15-Now</td>
</tr>
</tbody>
</table>

The reserved word DEP can be used to display temporally dependent formulas columns. Displaying the formulas is independent of the query type and yields the following results:

```
SELECT *, DEP(Salary) FROM employees;
```

```
<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Salary</th>
<th>FCOV(Salary)</th>
<th>FSUB(Salary)</th>
<th>Dpt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>left</td>
<td>fconst</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Matt</td>
<td>2000</td>
<td>left</td>
<td>fconst</td>
<td>1</td>
</tr>
</tbody>
</table>
```

```
SEQUENCED VALID
```

```
SELECT *, DEP(Salary) FROM employees;
```

```
<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Salary</th>
<th>FCOV(Salary)</th>
<th>FSUB(Salary)</th>
<th>Dpt</th>
<th>VT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>left</td>
<td>fconst</td>
<td>1</td>
<td>1-15</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>left</td>
<td>fconst</td>
<td>2</td>
<td>15-30</td>
</tr>
<tr>
<td>2</td>
<td>Matt</td>
<td>2000</td>
<td>left</td>
<td>fconst</td>
<td>1</td>
<td>15-Now</td>
</tr>
</tbody>
</table>
```

```
NONSEQUENCED VALID
```

```
SELECT*, DEP(Salary) FROM employees
WHERE DIST(VTIME(employees))>5;
```
4. COMPLETED DATABASE OPERATIONS

The incorporation of temporally independent and dependent attributes requires operations to handle this new type of information. The completed database provides information for all possible time sub-periods and is generated with the COMPL operation. Besides deriving information for sub-periods it is also relevant to cover information gaps with the COV operation. Note that we can complete a database after having covered the database to get a result that is covered and completed. In this section, we are going to describe the operations to handle temporally dependent information.

4.1. COV operation

Information gaps are covered using the $FCOV$ formula for temporally dependent attributes and the same value for temporally independent attributes.

EXAMPLE. Consider the table:

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Salary</th>
<th>$FCOV$(Salary)</th>
<th>$FSUB$(Salary)</th>
<th>Dpt</th>
<th>VT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>left</td>
<td>feconst</td>
<td>1</td>
<td>0-4</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>left</td>
<td>feconst</td>
<td>1</td>
<td>0-4</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>1500</td>
<td>left</td>
<td>feconst</td>
<td>1</td>
<td>7-9</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>1500</td>
<td>left</td>
<td>feconst</td>
<td>1</td>
<td>7-9</td>
</tr>
</tbody>
</table>

Covering a table from 0 to SYSDATE (we assume SYSDATE is 9):

SEQUENCED VALID SELECT * FROM COV (L, employeesC) ;

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Salary</th>
<th>Dpt</th>
<th>VT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>1</td>
<td>0-4</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>750</td>
<td>1</td>
<td>4-7</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>1500</td>
<td>1</td>
<td>7-9</td>
</tr>
</tbody>
</table>

The grey background denotes the explicit tuples and the white background the new ones. For the computation of the Salary attribute of the new tuple we can choose between two formulas, one for the derivation from the left tuple and another one for the derivation from the right tuple. Figure 2 illustrates this idea. In COV ($F$, table), $F$ can be: $L$(left) or $R$(right), to indicate which formula we want to use.

![Figure 2. Left or right formulas for the new tuple](image-url)
After covering, the new tuples have the same dependence formulas for temporally dependent attributes as the tuple we have covered. To group tuples to be covered, we proceed in a similar way than Segev and Shoshani [9] to generate the time sequences. More details about grouping tuples to be covered are explained in the chapter seven of Martín Ph.D. thesis [5].

4.2. COMPL operation

The COMPL operation derives the sub-periods of a tuple by applying the function \( FSUB \) to temporally dependent attributes to obtain the values for sub-periods. For temporally independent attributes, COMPL operation also generates the sub-periods but always with the same value.

**EXAMPLE.** Completing a table from \( 0 \) to *SYSDATE*:

```sql
SEQUENCED VALID SELECT * FROM COMPL (employeesC);
```

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Salary</th>
<th>Dpt</th>
<th>VT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tom</td>
<td>1000</td>
<td>1</td>
<td>0-4</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>250</td>
<td>1</td>
<td>0-1</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>250</td>
<td>1</td>
<td>1-2</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>250</td>
<td>1</td>
<td>2-3</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>250</td>
<td>1</td>
<td>3-4</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>500</td>
<td>1</td>
<td>0-2</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>500</td>
<td>1</td>
<td>1-3</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>500</td>
<td>1</td>
<td>2-4</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>750</td>
<td>1</td>
<td>0-3</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>750</td>
<td>1</td>
<td>1-4</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>1500</td>
<td>1</td>
<td>7-9</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>750</td>
<td>1</td>
<td>7-8</td>
</tr>
<tr>
<td>1</td>
<td>Tom</td>
<td>750</td>
<td>1</td>
<td>8-9</td>
</tr>
</tbody>
</table>

The main issue when completing a database is the algorithmic cost. The algorithm cost to generate a completed database for a database with \( n \) facts, where each fact has a distance \( d_i \) (\( 1 \leq d_i \leq k \)) is \( T_A(n,k) \in O(nk^2) \).

For efficiency reasons, completed tables are stored in temporary tables (*table*$_{\text{TMP}}$), that can be used for different statements until *COMMIT* or a new temporal range is selected. Moreover, we do not usually complete an entire table. Instead we generate the relevant sub-periods and this is possible using the **CUT** operation.

4.3. CUT operation

In order to partially complete a database we propose the **CUT** operation. The **CUT** operation finds the relevant sub-periods between two compatible predicates and reduces the cost of completing a database to linear cost.

**EXAMPLE.** Consider the following two tables that provide information from different data sources:
We decide to calculate an average of their values. However, we cannot compare the values of the tables because they are not stored for the same time periods. We require to generate the relevant sub-periods using the CUT operation. Figure 3 graphically shows the time intersection that is produced to find the relevant periods.

![Figure 3. Time intersection between employees1 and employees2](image)

```sql
SEQUENCED VALID SELECT * FROM CUT(employees1, employees2);
```

The idea to consider only relevant information is previously sketched in Böhlen et al. [1]. Furthermore, in Toman [11], SQL/TP is proposed, a language with a point-based approach to temporal extensions of SQL. Our CUT operation is based on the normalization operation of Toman [10].

The use of a temporal range selection SET DEP VALID PERIOD, implies an implicit CUT operation between the period selected and the tuples obtained by the query.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have extended the temporal language ATSQL to support temporally dependent information. Moreover, we have discussed a prototype implementation of these new capabilities of the language ATSQL using the temporal database system Tiger.
In the future, we want to extend the current implementation of temporally dependent attributes. Especially, we are interested to change the isomorphism to natural numbers for timestamps as to use the same date type as Tiger. At this moment, we are implementing a duration function to calculate the distance between all kind of date types. Currently, we are preparing a new web portal for Tiger, where the prototype for temporally dependent information is going to be offered. The new web portal address of TIGER will be: http://www.lsi.upc.edu/~tiger/.

ACKNOWLEDGMENTS

The authors would like to thank Núria Castell for the support she has given to this work, Octavio Torrijos for the implementation of the duration function, Ángel Baltasar for the Tiger web portal development and Javier Fernández for the Tiger download and test creation.

REFERENCES