Efficient Management of Schema Versioning in Multi-Temporal Databases

Zouhaier Brahmia[1], Mohamed Mkaouar[2], Salem Chakhar[3], and Rafik Bouaziz[1]
[1]Faculty of Economic Sciences and Management, University of Sfax, Tunisia
[2]Faculty of Science, University of Tunis-El Manar, Tunisia
[3] LAMSADE Laboratory, University of Paris Dauphine, France

Abstract: *To guarantee a complete data history in temporal databases, database management systems have to manage both evolution of schema over time, through their versioning, and evolution of data defined under different schema versions. This paper proposes a new approach for schema versioning in multi-temporal databases. It allows an efficient management of schema versions and their underlying data, through a smooth conversion of the temporal database. When creating a new schema version, the basic idea consists in forbidding 1) any automatic transfer of data defined under previous schema versions to this new version, in order to avoid data loss and ambiguousness in the interpretation of temporal intervals of data, and 2) any change in the structures of previous schema versions, in order to permit to the legacy applications to remain operational after this schema evolution.*

Keywords: *Schema evolution, schema versioning, temporal databases, multi-temporal databases, application time, database conversion.*

Received July 31, 2010; accepted October 24, 2010

1. Introduction

1.1. Temporal Databases

Many computational applications like banking, auditing, flight reservations, and weather monitoring have to store evolution of data over time. Each of these applications requires a temporal DataBase (DB) which records time-referenced data [9, 14, 18, 22], so, each entity is represented by a set of tuples. Temporal DBs use one or two time dimensions to timestamp data: transaction time, which indicates when an event is recorded in the DB, and valid time, which represents the time when an event occurred, occurs or is expected to occur in the real world. Temporal DBs are classified, according to these time dimensions, into four categories [9]: Snapshot (SN), Transaction Time (TT), Valid Time (VT), and Bi-Temporal (BT) DBs.

- Snapshot DBs, which contain only snapshot relations, are traditional DBs.
- Transaction time DBs, which contain only transaction time relations, timestamp tuples with their transaction times: transaction start time (TST) and transaction end time (TET). Changes can only apply to the current tuples; past data can not be modified and future tuples can not be introduced. The domain of TET includes the value "UC" (Until Change) [22]; a tuple that has UC as the value of its TET represents the current tuple of an entity.
- Valid time DBs, which contain only valid time relations, timestamp tuples with their valid times: Validity Start Time (VST) and Validity End Time (VET). So, it is possible to store past, current and future data. But after any data correction, the erroneous value will not be retained. The domain of VET includes the value "Now" [22], a tuple that has now as the value of its VET represents the current tuple of an entity until some changes occur.
- Bi-temporal DBs, which contain only bi-temporal relations, use both transaction time and valid time to timestamp tuples.

A DB which contains relations of different formats (SN, TT time, VT, and BT) is called a multi-temporal DB [7].

In this paper, we consider multi-temporal relational DBs. This is justified as follows:

- The choice of the multi-temporal environment is due to our aim to study all cases of schema evolution, including format changes (e.g., migration from SN to BT format).
- The choice of the relational environment is due to our aim to establish foundations of schema versioning for temporal DBs through a well-founded data model (i.e., relational model). We will proceed later to the revision and to the extension of our approach to make it applicable in an object-oriented or an XML environment.

1.2. Schema Versioning

1.2.1. Schema Evolution Versus Schema Versioning

In DBs, schemas are also subject to change. There are

two techniques [19, 20] to manage this change:

- *Schema Evolution:* Which is provided by a system that allows recovery of previous extensional data after each schema change, except data of dropped columns. Only the last schema version is kept.
- *Schema Versioning:* Which is provided by a system that allows both maintenance of extensional data and management of previous schema after any schema changes. It offers facilities which subsume those of "Schema evolution".

Before going farther, we need to define the concept of schema version of a relation.

- *Definition 1:* A schema version of a relation corresponds to the result of either the creation of this relation or the application of change operations on its schema.

1.2.2. Motivations

Schema of DBs can evolve over time due to many reasons such as changes in user requirements, new functionalities, compliance to new regulations, correction of deficiencies in the current schema, and migration to a new platform.

Schema versioning is highly required for several reasons like avoiding loss of data after schema changes, maintenance of legacy data formatted according to past schemata, reuse of legacy applications, and auditing purposes.

Several studies of real-world cases in seven DB applications from different fields [21], in a business system [8], or in the DB of wikipedia [5] show that we need generic solutions for schema versioning.

1.2.3. Discussed Issues

If a DataBase Management System (DBMS) does not record all schemas and their evolutions over time, then: 1) Manipulation and querying of temporal data defined under these schemas become insufficient and inefficient. 2) History of data can not be complete. Take for example the case when the DB administrator drops an attribute "A" from a schema "S". Values of "A" can not be retrieved after doing this operation if the DBMS does not keep track of the schema "S" and all tuples defined under "S". So, the DBMS has to keep track of both schema and data evolution, through schema and data versions, respectively.

In this paper, we deal with schema versioning in multi-temporal relational DBs (schema change operations and their effect on the underlying data). We propose an approach for schema versioning that avoids the three following problems which arise when the DBMS does not support schema versioning.

- *Problem 1:* Existing applications written before a schema change can not work after this change when the schema is being updated in a destructive manner. To be operational, they must be rewritten and recompiled after such a schema change (e.g., drop of columns or relations), since these applications were compiled against this schema.
- *Problem 2:* Possible ambiguity can occur for the interpretation of temporal dimensions of tuples after a schema change that modifies the format: "How to interpret the valid times of tuples defined under a snapshot relation when the DB administrator changes its format to be a bi-temporal one?".
- *Problem 3:* Loss of data can occur after dropping a relation or after a schema change which drops or modifies domains of some attributes of a relation.

1.3. Structure of the Paper

The rest of the paper is organized as follows. Section 2 introduces basic features of our approach for schema versioning in multi-temporal relational DBs. Section 3 presents our technique for multi-temporal DBs conversion when their schemas evolve. Section 4 illustrates our approach by an example. Section 5 is devoted to a brief review of related works and to the discussion of our approach in such a context. Finally, conclusions can be found in section 6.

2. Basic Features of the Proposed Approach for Schema Versioning

2.1. VERSIOS: The Multi-Temporal Data Model Supporting Schema Versioning

The model that we propose for schema versioning in multi-temporal DBs is named VERSIOS. It is temporally ungrouped [9]. We think that temporally ungrouped models are better advisable to relational DBs, at the logical and the physical levels. They provide a simple extension of relational DBMS to the temporal dimension. We do not adopt temporally grouped models because they are better advisable for object-oriented or XML DBs which do not respect the first normal form.

Our model [1, 2, 3] is briefly described below. Let us take: R as a relation; V_k_R as the table that stores tuples of R defined under its schema version number k; KA_i as a key attribute of R and NKA_j as a non-key attribute of R. According to the format of the schema version number k of R, V_k_R will be defined as follows:

- $V_k\_R(\underline{KA_1}, \underline{KA_2}, \ldots, \underline{KA_n}, NKA_1, NKA_2, \ldots, NKA_m)$ if its format is snapshot; this table stores only current tuples.
- $V_k\_R(\underline{KA_1}, \underline{KA_2}, \ldots, \underline{KA_n}, \underline{TST}, TET, NKA_1, NKA_2, \ldots, NKA_m)$ if its format is transaction time; this table stores past and current tuples.
- $V_k\_R(\underline{KA_1}, \underline{KA_2}, \ldots, \underline{KA_n}, \underline{VST}, VET, NKA_1, NKA_2, \ldots, NKA_m)$ if its format is valid time; this table stores past, current and future tuples.

- $V_k\_R(\underline{KA_1}, \underline{KA_2}, \ldots, \underline{KA_n}, \underline{VST}, VET, \underline{TST}, TET, NKA_1, NKA_2, \ldots, NKA_m)$ if its format is bi-temporal; it stores past, current and future tuples.

Notice that the DBMS creates V_k_R automatically and without any tuple, at the application time of the schema version number k of R.

To store information about schema of relations and to be able to maintain history of schema changes, we propose the catalogues presented below. Among all the relational system catalogues, we present only the two catalogues describing the relations and their attributes, since they are those mostly involved in schema versioning.

Relation (Relation_Name, Schema_Version_Number, Schema_Version_Format, Application_Start_Time, Application_End_Time, Schema_Version_State)

This catalogue keeps for each relation its name, and for each schema version of this relation its number, its format, its application start time (i.e., the time from which this schema version is applied), its application end time (i.e., the time from which this schema version is considered as a past one), and its state (Past, Erroneous, or Current).

Attribute (Relation_Name, Schema_Version_Number, Attribute_Name, Domain, Key, Attribute_Order_Number)

This catalogue stores information about attributes of relations under their different schema versions. It keeps, for each attribute, the name of the relation and the number of the schema version to which it belongs, its name, its domain (string, integer, real…), if it is part of the primary key, and its order number in this schema version.

The synoptic diagram of VERSIOS is depicted in Figure 1. When receiving an order from the DB administrator about changing the schema of a relation, VERSIOS creates a new schema version of this relation and updates the catalogues.

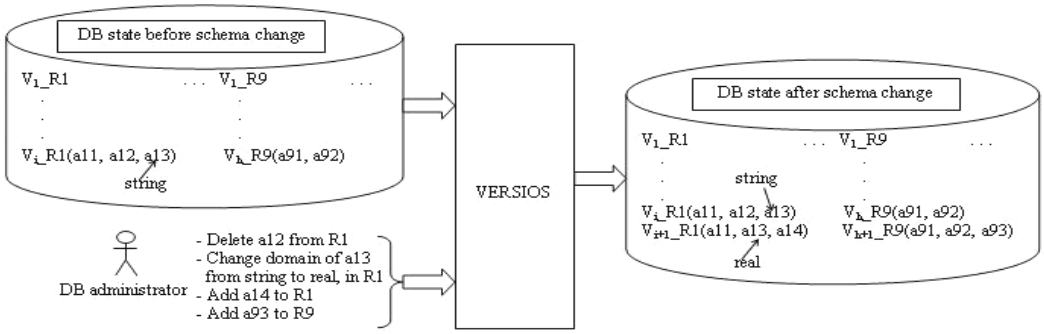

Figure 1. Synoptic diagram of VERSIOS.

2.2. Schema Change Operations

We have defined three categories of schema change operations [3] that we consider the most useful for applications using multi-temporal relational DBs.

- *Operations Acting on a Relation:* Creation, deletion, renaming, or change of the format of a relation; activating a deleted relation.
- *Operations Acting on an Attribute of a Relation:* Addition, deletion, renaming, or change of the domain of an attribute; transformation of an attribute into a relation (i.e., reification).
- *Operations Acting on More than one Relation at a Time:* Fusion of many relations into a single new relation; splitting a relation into several relations.

Since our purpose, in this paper, is to describe logical solutions for schema versioning, we limit the discussion to a sub-set of changes applied to a single relation and their attributes, changes that we judge the most significant for applications. We study in particular all the cases which imply an increase show in Table 1 or a decrease show in Table 2 of the temporal dimension of a relation when its format changes.

The schema evolution of a relation from a version V_i to a version V_{i+1} is the result of a set of schema change operations performed on V_i. Notice that our approach does not lead to proliferation of schema versions. The application of a new schema version of a relation is anyway a seldom task during the DB lifetime, which can only be performed by a DB administrator. This task may consist of dozens of schema change operations which are grouped together in the same single transaction.

Table 1. Cases of increase of the temporal dimension.

Format of the Last Schema Version	Format of the New Schema Version
SN	TT, VT, BT
TT	VT, BT
VT	TT, BT

Table 2. Cases of decrease of the temporal dimension.

Format of the Last Schema Version	Format of the New Schema Version
BT	SN, TT, VT
TT	SN
VT	SN

2.3. A New Time Dimension for Schema Versioning: Application Time

Three approaches for schema versioning are described in [7]: transaction time, valid time, and bi-temporal schema versioning. Among these approaches, we see that valid time schema versioning is more appropriate than the other approaches for the two reasons presented below.

1. Valid time schema versioning provides more facilities than transaction time schema versioning since it supports retroactive and proactive changes which are important tasks in information systems. Consider for example the promulgation of a new law enforcing a change of domain in some administrative attribute with retroactive effect, or a schema change which is designed and approved but could not be implemented on time because of some delay (e.g., retirement or illness of the DB administrator, crash of or contractual dispute with the software house in charge). These cases require retroactive schema changes to be effected with the stated validity.
2. Valid time schema versioning is simpler than bi-temporal schema versioning which requires also the management of transaction time of schema versions, the management of inconsistent states of the DB during some periods, etc. We think that schema versioning in information systems does not need these extensions.

However, we see that schema versioning is not a simple management of meta-data along transaction time and/or valid time axis, in a similar way to the management of data along these axes. We believe that schema versioning must be managed with a lot of precaution, and requires suitable techniques which are different of those used to manage temporal data. Moreover, schema versioning must be piloted by the DB administrator who creates, changes, and deletes schema.

So, we propose a new approach [1, 3] which takes into account all these aspects. It is based on schema versioning along a constrained valid time axis that we call "application time". Each schema version is timestamped with an application start time and an application end time which are defined below:

- *Definition 2:* The application start time of a schema version of a relation R is the instant fixed by the DB administrator to apply this schema version. From that instant, this schema version is considered to be the current schema version for R.
- *Definition 3:* The application end time of a schema version of a relation R is the instant fixed by the DB administrator to make end to the application of this schema version. From that instant, this schema version is no longer the current one and is considered to be a past schema version of R.

The constraints that we have put on valid time axis to have the application time for schema versioning are described in the following:

- The application of a new schema version of a relation with proactive effect is not authorized.
- The application of a new schema version of a relation with retroactive effect is authorized only and only if it is done with these tasks:
  - Previous schema versions which are completely overlapped by the application time interval of the new schema version are not deleted but they are kept with their underlying data without any change. Only states of these schema versions are changed from "Past" to "Erroneous", in the catalogues.
  - Previous schema versions which are partially overlapped by the application time interval of the new schema version have their application time intervals accordingly restricted.
  - All programs that were executed using erroneous schema versions (and thus they have produced erroneous results) must be compiled and performed again with the new schema version. To do this, the DBMS must keep track of all program executions and data that have been manipulated by these programs.

Thus, in our approach, schema versions of a relation are successive in time. There are neither parallel nor merging schema versions.

3. A Smooth Conversion of Multi-Temporal DBs when Schema Evolve

In order to guarantee a correct evolution of schema and their underlying data, we define five rules [3] that allow a smooth conversion (i.e., conversion without problems and errors) of the DB when schema evolves. We will illustrate these rules by an example of schema evolution.

- *Rule 1:* At the application time of the new schema version V_{i+1} of a relation R, the DBMS must not change structures of previous schema versions of R. Such a change has to be forbidden in order to guarantee that existing applications continue to be operational after every schema evolution.
- *Rule 2:* At the application time of the new schema version V_{i+1} of a relation R, the DBMS must not do any automatic migration of data from tables of previous schema versions of R to the table(s) of V_{i+1}. Such a transfer of data has to be forbidden since it can lead to loss of data when the new schema version does not contain some attributes belonging to the previous schema version.
- *Rule 3:* At the application time of the new schema version V_{i+1} of a relation R, if this version changes

the format of the previous schema version V_i with an increase of the temporal dimension see Table 1, the DBMS has to convert the format of every previous schema version $V_k(1<=k<=i)$ of R, which is lower than the format of V_{i+1}, to a new format as mentioned in Table 3.

Table 3. New formats of previous schema versions after increasing the temporal dimension.

Format of Version V_{i+1}	Format of Version V_k (1<=k<=i)	
	Before Conversion	After Conversion
TT	SN	TT_{SN}: Transaction time originating from snapshot
VT	SN	VT_{SN}: Valid time originating from snapshot
BT	SN	BT_{SN}: Bi-temporal originating from snapshot
VT, BT	TT_{SN}	
TT, BT	VT_{SN}	
VT, BT	TT	BT_{TT}: Bi-temporal originating from transaction time
TT, BT	VT	BT_{VT}: Bi-temporal originating from valid time

This rule allows avoiding ambiguity in the interpretation of temporal dimensions of tuples defined under previous schema versions when the format of the new schema version is richer. To manipulate data correctly, the new format of each previous schema version keeps track of the original format.

For illustration, suppose that we have a TT schema version V_i of a relation R. Tuples defined under this version were timestamped only through their transaction times. Suppose that the DB administrator applies a new BT schema version V_{i+1} of R. If the DBMS does not convert V_i to become a BT_{TT} schema version (by adding a valid time interval to every tuple defined under V_i; see Rule 3.1 presented below), we will have problems since we have no information about valid times of tuples that were defined under V_i.

In the same way, every previous schema version of R that has a format which is lower than the BT format must be converted. With such conversions, all involved previous schema versions of R become in adequacy with the format of the new schema version, but without loss of data. The next two rules, Rule 3.1 and Rule 3.2, show how Rule 3 is applied.

- *Rule 3.1:* At the application time of the new schema version V_{i+1} of a relation R with increase of the temporal dimension, such that V_{i+1} has a VT or a BT format, the DBMS must add a valid time interval to the table of each previous schema version of R having a TT, a TT_{SN} or a SN format. Then, the DBMS must assign VST and VET values to each tuple defined under these previous schema versions, as presented below:

1. For each current tuple, the valid time interval is defined as follows:
   a. The VST takes one of the two following values:
      - The value of the application start time of the new schema version V_{i+1}, when this tuple is defined under a previous SN (converted to VT_{SN} or to BT_{SN}) schema version.
      - The value of the transaction start time of this tuple, when this tuple was defined under a TT (converted to BT_{TT}) or a TT_{SN} (converted to BT_{SN}) schema version.
   b. The VET takes the value "Now" [22], meaning that it increases with time until some changes occur.
2. For each past tuple, which was necessarily defined under a TT or a TT_{SN} schema version, VST and VET attributes take respectively the value of the TST attribute and the value of the TET attribute of this tuple.

- *Rule 3.2:* At the application time of the new schema version V_{i+1} of a relation R with increase of the temporal dimension such that V_{i+1} has a TT or a BT format, the DBMS must add a transaction time interval to the table of each previous schema version of R having a VT, a VT_{SN}, or a SN format. Then, the DBMS must assign TST and TET values to each tuple defined under these previous schema versions, as follows:

1. For each current tuple, the TET attribute takes the value "UC" [22] and the TST attribute takes one of the two following values:
   - The value of the application start time of the new schema version V_{i+1}, when this tuple is defined under any previous SN (converted to TT_{SN} or to BT_{SN}) schema version V_k.
   - The value of the validity start time of this tuple, when this tuple was defined under any VT (converted to BT_{VT}) or any VT_{SN} (converted to BT_{SN}) schema version.
2. For each past tuple, which was necessarily defined under a VT or a VT_{SN} schema version, TST and TET attributes take respectively the value of the VST attribute and the value of the VET attribute of this tuple.

Rule 3.1 (respectively Rule 3.2) allows avoiding overlap of valid time (respectively transaction time) intervals of tuples defined under schema versions which have been converted. As for the assignment of approximate values to VST and VET attributes (respectively to TST and TET attributes) for these tuples, the DBMS is always able to interpret suitably these values thanks to formats of these converted schema versions (i.e., VT_{SN}, TT_{SN}, BT_{SN}, BT_{VT}, and BT_{TT}) and to inform users that the exact values of VST and VET (respectively of TST and TET) are unknown and generally precede the present values. The DBMS can answer user queries with expressions like "valid before the date *d*" or "having an approximate valid time interval *I*" (respectively "inserted in the DB

before the date *t*" or "having an approximate transaction time interval *T*"). Notice that:

- End users will be able to provide exact values for VST and VET attributes, if possible.
- The addition of VST and VET attributes (respectively of TST and TET attributes) to tables of previous schema versions does not raise any problem for existing applications using these tables.

4. An Illustrative Example

Assume that we have a DB that contains a snapshot relation SALESMAN created on 2007/12/01 with the attributes salesman ID, NAME, CITY and SALARY. Table 4 shows the state of this relation on 2008/03/09. Table 5 shows the state of the DB catalogues at the same date.

Table 4. The state of the SALESMAN relation on 2008/03/09.

V_1_SALESMAN			
ID	**NAME**	**CITY**	**SALARY**
1	Ahmed	Sfax	1000
2	Fares	Sfax	1200

Assume that the following two schema changes are applied to the SALESMAN relation:

- SC1: On 2008/03/10, the attribute CITY is dropped from SALESMAN, a new attribute PHONE is added to SALESMAN, and the format of SALESMAN is changed to become transaction time.
- SC2: On 2009/04/15, a new attribute BONUS is added to SALESMAN and the format of SALESMAN is changed to become bi-temporal.

Tables 6 and 7 show respectively the state of the SALESMAN relation and the DB catalogues on 2008/03/10, after execution of SC1.

Suppose that we have manipulated some tuples under the table V2_SALESMAN. Table 8 shows the state of the SALESMAN relation before and after execution of SC2. Table 9 shows the state of the DB catalogues after execution of SC2.

Table 6. The state of the SALESMAN relation on 2008/03/10, after SC1.

V_1_SALESMAN					
ID	**NAME**	**CITY**	**SALARY**	**TST**	**TET**
1	Ahmed	Sfax	1000	2008/03/10	UC
2	Fares	Sfax	1200	2008/03/10	UC
V_2_SALESMAN					
ID	**NAME**	**PHONE**	**SALARY**	**TST**	**TET**
empty					

Table 5. The state of the catalogues on 2008/03/09.

RELATION					
Relation_Name	**Schema_Version_Number**	**Schema_Version_Format**	**Application_Start_Time**	**Application_End_Time**	**Schema_Version_State**
SALESMAN	1	SN	2007/12/01	null	Current
ATTRIBUTE					
Relation_Name	**Schema_Version_Number**	**Attribute_Name**	**Domain**	**Key**	**Attribute_Order_Number**
SALESMAN	1	ID	string	Yes	1
SALESMAN	1	NAME	string	No	2
SALESMAN	1	CITY	string	No	3
SALESMAN	1	SALARY	real	No	4

Table 7. The state of the catalogues on 2008/03/10, after SC1.

RELATION					
Relation_Name	**Schema_Version_Number**	**Schema_Version_Format**	**Application_Start_Time**	**Application_End_Time**	**Schema_Version_State**
SALESMAN	1	TT_{SN}	2007/12/01	2008/03/09	Past
SALESMAN	2	TT	2008/03/10	null	Current
ATTRIBUTE					
Relation_Name	**Schema_Version_Number**	**Attribute_Name**	**Domain**	**Key**	**Attribute_Order_Number**
SALESMAN	1	ID	string	Yes	1
SALESMAN	1	NAME	string	No	2
SALESMAN	1	CITY	string	No	3
SALESMAN	1	SALARY	real	No	4
SALESMAN	2	ID	string	Yes	1
SALESMAN	2	NAME	string	No	2
SALESMAN	2	PHONE	string	No	3
SALESMAN	2	SALARY	real	No	4

Table 8. The state of the SALESMAN relation before SC2 (Part a) and after SC2 (Part b).

Part a					
V_1_SALESMAN					
ID	**NAME**	**CITY**	**SALARY**	**TST**	**TET**
1	Ahmed	Sfax	1000	2008/03/10	2009/03/26
2	Fares	Sfax	1200	2008/03/10	UC
V_2_SALESMAN					
ID	**NAME**	**PHONE**	**SALARY**	**TST**	**TET**
3	Khadija	9633445	1200	2008/04/12	UC
4	Aicha	9755667	1000	2008/06/22	UC
1	Ahmed	9877889	1100	2009/03/27	UC

Part b							
V_1_SALESMAN							
ID	**NAME**	**CITY**	**SALARY**	**TST**	**TET**	**VST**	**VET**
1	Ahmed	Sfax	1000	2008/03/10	2009/03/26	2008/03/10	2009/03/26
2	Fares	Sfax	1200	2008/03/10	UC	2008/03/10	Now
V_2_SALESMAN							
ID	**NAME**	**PHONE**	**SALARY**	**TST**	**TET**	**VST**	**VET**
3	Khadija	9633445	1200	2008/04/12	UC	2008/04/12	Now
4	Aicha	9755667	1000	2008/06/22	UC	2008/06/22	Now
1	Ahmed	9877889	1100	2009/03/27	UC	2009/03/27	Now
V_3_SALESMAN							
ID	**NAME**	**PHONE**	**SALARY**	**BONUS**	**VST**	**VET**	**TST**	**TET**
empty								

Table 9. The state of the catalogues after SC2.

RELATION					
Relation_Name	**Schema_Version_Number**	**Schema_Version_Format**	**Application_Start_Time**	**Application_End_Time**	**Schema_Version_State**
SALESMAN	1	BT_{SN}	2007/12/01	2008/03/09	Past
SALESMAN	2	BT_{TT}	2008/03/10	2009/04/14	Past
SALESMAN	3	BT	2009/04/15	null	Current

ATTRIBUTE					
Relation_Name	**Schema_Version_Number**	**Attribute_Name**	**Domain**	**Key**	**Attribute_Order_Number**
SALESMAN	1	ID	string	Yes	1
SALESMAN	1	NAME	string	No	2
SALESMAN	1	CITY	string	No	3
SALESMAN	1	SALARY	real	No	4
SALESMAN	2	ID	string	Yes	1
SALESMAN	2	NAME	string	No	2
SALESMAN	2	PHONE	string	No	3
SALESMAN	2	SALARY	real	No	4
SALESMAN	3	ID	string	Yes	1
SALESMAN	3	NAME	string	No	2
SALESMAN	3	PHONE	string	No	3
SALESMAN	3	SALARY	real	No	4
SALESMAN	3	BONUS	real	No	5

5. Related Work and Discussion

A good survey of different schema versioning issues in temporal DBs was presented in [19].

The paper [7] presents three approaches for schema versioning: transaction time schema versioning, valid time schema versioning and bi-temporal schema versioning. Non-temporal schema versioning had been proposed before for example in [15]. It is equivalent to transaction time schema versioning without explicit management of timestamps. We propose a new constrained valid time axis for schema versioning. We think that it is more suitable and more flexible for schema versioning in enterprise information systems than transaction time and valid time.

In [23], the author studies schema evolution and schema versioning supports in SQL-99 language and some commercial relational and object-relational DBMS (like Oracle8i Server and Ingres II). These systems support only schema evolution since they retain always the last version of any schema. After each schema modification, they kept only the updated schema and they import and adapt data defined under the past schema to the new schema. Also, most existing object-oriented DBMS like O_2 [10] and GemStone [4] support only schema evolution. Grandi presents in [13] a tool prototype, named SVMgr, that supports non-temporal schema versioning in snapshot relational DBs.

Many works on schema versioning in object-oriented DBs e.g., [15, 16] introduced a non-temporal schema versioning and dealt with alternative schema versions. Our approach does not provide alternative schema versions since our environment is an enterprise information system. In such an environment, schema versions of relations are generally successive in time, and applications use always one schema version of the same relation at the same time.

Grandi and Mandreoli propose in [12] a formal model called OODM-SV for temporal versioning of schema and instances in a temporal object-oriented DB. In [11], the authors introduce a temporal and versioning model (called TVSE) to manage schema evolution in object-oriented DBs. These two approaches adopt temporal schema versioning and manage simultaneously data and schema versioning. Our approach is similar to these approaches by supporting both data and schema versioning. But, unlike them, our approach does not consider schema versioning as a simple management of intentional data (i.e., schema) along transaction time and/or valid time axis, in a similar way to the management of extensional data along these axes. In our approach,

schema versioning is performed through suitable techniques and rules, on the one hand, and it is piloted by the DB administrator and not by the DBMS, on the other hand.

In [3], we present basic principles of our approach for schema versioning in multi-temporal DBs. Our previous works [1, 2] prepare the present work by showing the feasibility of our approach for the management of multi-temporal DBs that support schema versioning. In [1], we propose an approach and a prototype (named VERSIOD) to manipulate (i.e., insert, update and delete) data under different schema versions of relations in a multi-temporal DB. This approach keeps track of data history, guarantees the consistency of data and optimizes spaces of data storage. In [2], we present a flexible solution and a prototype to query multi-temporal data in a multi-version environment; non-expert users are released from syntactical difficulties of textual temporal query languages like TSQL2, and they can easily express multi-schemas temporal queries by means of interaction with the system.

Recently, a lot of work has been done on schema evolution in temporal XML and semi-structured DBs e.g., [6, 17], often applying concepts and techniques developed by temporal DB research.

6. Conclusions

In this paper, we propose a new approach for schema versioning that ensures a smooth DB evolution and conversion when creating a new schema version of a relation, through:

1. Avoiding any automatic transfer of data defined under previous schema versions to the new one.
2. Forbidding any change in structures of previous schema versions.
3. Conversion of previous schema versions if the new one leads to increase of the temporal dimension.

Our approach allows resolving three problems that may hold when a schema evolves:

1. Applications that become non-operational after a schema change that modifies the structure of the current schema.
2. Possible confusion in the interpretation of temporal intervals of tuples.
3. Data loss after dropping or changing the domain of some attributes.

Moreover, we propose a new time axis, called application time, which is a restriction of valid time. It is more suitable for schema versioning in enterprise information systems than the other temporal axes.

Our future work will aim at studying transaction versioning within multi-temporal DBs. This versioning is necessary to keep trace of the dynamic behaviour of information systems, to re-execute transactions that had used erroneous data for example, and to redress the effects of using these erroneous data.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions on the earlier version of the paper, which helped to improve the quality of the paper.

References

[1] Bouaziz R. and Brahmia Z., "Gestion Des Données Temporelles Dans Un Environnement Multiversion De Schémas," *Technique et Science Informatiques*, vol. 28, no. 1, pp. 39-74, 2009.

[2] Brahmia Z. and Bouaziz R., "Interaction Visuelle Pour L'interrogation Des Bases De Données Temporelles Multi-Versions," *in Proceedings of 1st International Conference on Information Systems and Economic Intelligence*, Tunisia, pp. 663-680, 2008.

[3] Brahmia Z., Bouaziz R., and Chakhar S., "Gestion De L'évolution Des Schémas Dans Les Bases De Données Multi-Temporelles," *in Proceedings of 14th International Business Information Management Association Conference*, Turkey, pp. 1862-1874, 2010.

[4] Bretl R., Maier D., Otis A., Penney D., Schuchardt B., Stein J., Williams E., and Williams M., "The Gemstone Data Management System," *in Proceedings of Object-Oriented Concepts, Databases, and Applications*, pp. 283-308, 1989.

[5] Curino C., Moon H., Tanca L., and Zaniolo C., "Schema Evolution in Wikipedia: Toward a Web Information System Benchmark," *in Proceedings of 10th International Conference on Enterprise Information Systems*, Spain, pp. 290-297, 2008.

[6] Curino C., Moon H., Ham M., and Zaniolo C., "The PRISM Workwench: Database Schema Evolution without Tears," *in Proceedings of 25th International Conference on Data Engineering*, China, pp. 1523-1526, 2009.

[7] De-Castro C., Grandi F., and Scalas M., "Schema Versioning for Multitemporal Relational Databases," *Information Systems*, vol. 22, no. 5, pp. 249-290, 1997.

[8] De-Vries D. and Roddick J., "The Case for Mesodata: An Empirical Investigation of an Evolving Database System," *Information and Software Technology*, vol. 49, no. 9-10, pp. 1061-1072, 2007.

[9] Etzion O., Jajodia S., and Sripada S., *Temporal Databases: Research and Practice*, Springer-Verlag, 1998.

[10] Ferrandina F., Meyer T., Zicari R., Ferran G., and Madec J., "Database Evolution in the O2 Object

Database System," *in Proceedings of 21th International Conference on Very Large Data Bases*, Switzerland, pp. 170-181, 1995.

[11] Galante R., Dos-Santos C., Edelweiss N., and Moreira A., "Temporal and Versioning Model for Schema Evolution in Object-Oriented Databases," *Data and Knowledge Engineering*, vol. 53, no. 2, pp. 99-128, 2005.

[12] Grandi F. and Mandreoli F., "A Formal Model for Temporal Schema Versioning in Object-Oriented Databases," *Data and Knowledge Engineering*, vol. 46, no. 2, pp. 123-167, 2003.

[13] Grandi F., "SVMgr: A Tool for the Management of Schema Versioning," *in Proceedings of 23rd International Conference on Conceptual Modeling*, China, pp. 860-861, 2004.

[14] Jensen C. and Snodgrass R., "Temporal Database," *in Proceedings of Encyclopedia of Database Systems*, Copenhagen, pp. 2957-2960, 2009.

[15] Kim W. and Chou H., "Versions of Schema for Object-Oriented Databases," *in Proceedings of 14th International Conference on Very Large Data Bases*, California, pp. 148-159, 1988.

[16] Lautemann S., "Schema Versioning in Object-Oriented Database Systems," *in Proceedings of 5th International Conference on Database Systems for Advanced Applications*, Australia, pp. 323-332, 1997.

[17] Moon H., Curino C., and Zaniolo C., "Scalable Architecture and Query Optimization for Transaction-Time DBs with Evolving Schemas," *in Proceedings of ACM SIGMOD International Conference on Management of Data*, USA, pp. 207-218, 2010.

[18] Rahim M., Kurmin N., Wahid M., and Daman D., "Equi-Join Table Optimization Technique for Temporal Data," *The International Arab Journal of Information Technology*, vol. 4, no. 3, pp. 272-280, 2007.

[19] Roddick J., "A Survey of Schema Versioning Issues for Database Systems," *Information and Software Technology*, vol. 37, no. 7, pp. 383-393, 1995.

[20] Roddick J., "Schema Versioning," *in Proceedings of Encyclopedia of Database Systems*, pp. 2499-2502, 2009.

[21] Sjoberg D., "Quantifying Schema Evolution," *Information and Software Technology*, vol. 35, no. 1, pp. 35-44, 1993.

[22] Torp K., Jensen C., and Snodgrass R., "Effective Timestamping in Databases," *The VLDB Journal*, vol. 8, no. 3-4, pp. 267-288, 2000.

[23] Türker C., "Schema Evolution in SQL-99 and Commercial Object-Relational DBMS," *in Proceedings of 9th International Workshop on Foundations of Models and Languages for Data and Objects*, Germany, pp. 1-32, 2000.

Zouhaier Brahmia received his MSc in computer science from the Faculty of Economics and Management of Sfax, University of Sfax, Tunisia, in July 2005, and a PhD in computer science from the same Faculty, in December 2011. He joined the Department of Computer Science in the Faculty of Economics and Management of Sfax as an assistant professor, in September 2012. He is a member of Multimedia, Information systems and Advanced Computing Laboratory, since 2004. His scientific interests include temporal databases, native XML databases, schema versioning, data management, and World Wide Web extensions.

Mohamed Mkaouar received his MSc in computer science from the Faculty of Science of Tunis, University of Tunis-Elmanar, Tunisia, in October 1999, and a PhD in computer science from the same Faculty, in 2012. His research interests include temporal databases and information system modelling.

Salem Chakhar is with the Centre for Research in Regional Planning and Development, ESAD, Laval University, Québec City, Canada. He received his MSc in computer science and modeling from the High School of Management of Tunis, Tunisia and a PhD in computer science from the University of Paris-Dauphine, France. His research interests include geographical information science and systems, spatial modeling and analysis, database and information systems, fuzzy theory and applications and decision support systems. He has published in journals such as International Journal of Geographical Information Science, Computers, Environment and Urban Systems, Information Sciences, Information and Software Technology, European Journal of Operational Research, and Environment and planning B: Planning and Design.

Rafik Bouaziz is a doctor on computer science. Currently, he is the vice-dean of the Faculty of Economics and Management of Sfax, Tunisia. He was a consulting engineer in the Organization and Computer Science and a head of the Department of Computer Science at CEGOS-TUNISIA between 1979 and 1986. His main research topics of interest are temporal databases, real-time databases, information systems, data warehousing and workflows.