A Model for Schema Evolution in Temporal Relational Databases

Maria Rita Scalas, Alessandro Cappelli, Cristina De Castro

C.I.O.C. - C.N.R. - Dipartimento di Elettronica, Informatica e Sistemistica
Facolta' di Ingegneria, Viale Risorgimento, 2 , 40136 Bologna - Italy

Abstract: *Recently, attention has been focused on temporal databases, where transaction time and valid time are added to the data. In this context, the modification of the schema which a DBMS can undergo during its lifetime, known as schema changes, must develop into the higher concept of schema evolution. Changes to the schema produce versions of previous schemas along both time axes and these versions must be maintained and managed during the whole database life. In the first part of the paper we provide a short overview on the issues of temporal relational databases and relational schema changes. In the second part we give some basic outline for the integration of these two aspects into a temporal relational database model supporting schema evolution.*

Keywords: *Relational Databases, Temporal Databases, Schema Changes, Schema Evolution.*

1. Introduction

Databases describe a portion of the real world of interest. This portion reflects the *miniworld* of the application. When a new application is created, its structure is recorded in the database *schema*, and the collection of data items are recorded into the database. Non-temporal databases lack the representation and management of time, which is becoming a very important issue. It allows the extension of old applications and is necessary in several new application areas, such as CAD/CAM/CIM and public services. According to the taxonomy in [1,6,8,9,10], at least two different and independent concepts of time are necessary: *transaction* and *valid time*. Transaction time is the instant when an event is recorded in the database, thus it is defined by the system and keeps growing. Valid time is the instant when an event occurred, occurs or is expected to occur in the real world and is defined by the user. Recently, a great deal of work was devoted to the study of new models for data versioning along one or both time axes. This led to the definition of semantically different models: *monotemporal* (*transaction-time* or *valid-time databases), bitemporal databases* and *snapshot*, where no time support is provided [6].

In databases, not only instances, but also the schema can undergo changes, due to modifications of the miniworld or to specific requirements of the application. In literature, the modification that a schema can undergo during its lifetime are known as *schema changes* [5]. In snapshot DBs just one schema at a time describes the structure of the database: this means that schema changes force a new schema to substitute the existing one. A temporal database is modelled in order to maintain the dynamics of the miniworld, therefore schema change must also develop into the broader concept of *schema evolution*. In this paper by schema evolution we indicate the process of creation, maintainance and management of versions of the schema, each pertinent to a distinct interval of transaction and valid time. Some early proposals concerning schema evolution along one time dimension can be found for object-oriented databases and non-first-normal-form models in [2,4,7].

The study of temporal databases which support schema evolution leads to a more complete and powerful description of the miniworld, necessary to avoid the data misinterpretation that a traditional schema change application might produce. As a matter of fact, a schema change not embedded into the schema evolution would cause the previous schema to be overwritten. Therefore, depending on the implementation, information inserted according to such schema are no longer available or are wrongly accessed according to the new schema version, which no longer reflects the data model of such data. For example, if the result of a schema change is to drop an attribute, the schema of the relation after the schema change no longer contains a reference to the dropped attribute. This behaviour is contrary to the main requirement of temporal databases, that is *no data must be lost*. The maintainance of old schemas along both time dimensions leads to a database model capable of maintaining not only the evolution of data pertinent to the different schemas, whose structure is also maintained. Data are retrieved on the basis of the schema version acccording to which they were inserted.

In this paper we consider the traditional schema changes that a snapshot relational database can undergo and propose a solution for the transformation of these single changes into schema evolution within a temporal environment. In particular, in section 2 we describe the basic features of temporal relational databases and schema changes, in order to provide a self-consistent introduction to these issues. In section 3 we go through the aspects connected with the introduction of time coordinates in the schema and describe some basic outlines for the integration. We also describe an overall mechanism for schema evolution which involves the management of storage structures along both time axes.

2. An Overview of Temporal Databases and Schema Changes

Snapshot databases do not support time at all. They store only one version of the data, resulting from the application of the most recent transactions. Transaction-time databases keep all the versions of the data inserted in successive transactions. Valid-time databases record the complete history of the data, updated to the latest transaction. Bitemporal databases support both transaction and valid time and thus are able to store all the histories of the data as seen at each transaction time.

As is well known, in the relational model tuples contain a value for each attribute of the relation they belong to, according to its logical schema. Each tuple is identified by a unique key value within the relation. In the temporal relational model based on an interval logic [3], to each key value corresponds the collection of all the versions of the original tuple. Each version is identified by the relational key value (no longer unique) and by four temporal attributes. IN is the instant when a tuple was stored in the database, OUT the instant when a tuple was modified or deleted from the database (together representing the transaction time interval of the tuple). FROM is the beginning of validity, TO the end of validity; together they represent the valid time interval of the tuple.

A *history* is the collection of all tuples having the same key value but pertinent to different time intervals. After the introduction of the temporal attributes a relational key value (*static key)* identifies a history, and each version within a history is identified by the temporal attribute values. The combination of the static key and the temporal attributes becomes the *temporal key.* Therefore, in a bitemporal relational database, the tuple has the typical structure:

(STATIC KEY, ATTR.1, ... , ATTR.N, IN, OUT, FROM, TO)

The schema of a database is recorded into *catalogues,* which are relations themselves and store general information about the database. One of the catalogues records general information about the relations present in the database, another one stores information about the attributes of each relation. (The other catalogues will not be considered since they are not interesting to our aims). We have seen that the DB schema can undergo changes during the database life. In order to take into account all the operations a user can define on a schema, we can consider the following classes of operations on schemas of single relations:

- **Schema Redefinition:** for each relation whose schema undergoes redefinition, the structure must be completely specified (attributes, names, types, etc).

- **Schema Revision:** For each relation of revision of database the new schema is defined by specifying the changes to apply to the previous structure of the relations, in order to produce the required schema.

Notice that schema redefinition is completely independent from the schema that previously modelled the database, whereas schema revision is performed by applying changes to the most recent schema. In this paper we take into account the schema changes that a relational database can undergo in order to integrate them in a relational temporal database model. They can be classified as:

1) Changes to an attribute:

1.1 Add or drop an attribute
1.2 Rename an attribute
1.3 Modify the domain of an attribute

2) Changes to a relation:

2.1 Add or drop a relation
2.2 Rename a relation

Example: Suppose a relation of a public information system had initially the following schema:.

ID_CARD (name, address, town, country)

In 1985, the fiscal code was introduced and the schema became:

ID_CARD (fiscal code, name, address, town, country)

The two schemas must coexist as a version of a same schema , in order to be able to maintain the ID_CARD information before and after the schema change available as a single relation at the logical level.

3. The model for Schema Evolution

In the following we describe the model we propose for the schema evolution and present a mechanism to manage catalogue information and secondary storage stuctures when a temporal schema change occurs, both in the case of schema redefinition and of schema revision.

A natural way to keep all the schema versions is to add the temporal attributes IN, OUT, FROM, TO also to the catalogues and manage them as temporal relations. Schema evolution involves not only operations on catalogues, but also operations on the storage structures, because many schema changes modify the tuple structure. Let us give some definitions which will be useful afterwards:

Schema Version of a Relation: is one of the schemas that a relation assumes during the database life and which can be modified according to a temporal schema change. In the following, a schema version of a relation will be simply called a *relation version.*

Incompatible Relations: we say that two relations are incompatible if their records have different structures. This fact can arise also within versions of a same relation after a schema change.

Incompatible Temporal Schemas: two temporal schemas are *incompatible* if they contain at least two incompatible relations.

Data Pool: storage structure which contains tuples with the same structure. If two relations are incompatible, it is impossible to record their tuples in the same data pool.

Therefore, in a temporal relational database supporting schema evolution, a single relation at the logical level corresponds to as many data pools as the number of incompatible versions of the relation produced by the schema changes. In non temporal databases there is a one to one correspondence between the relations and their data pools. Due to schema incompatibility, in temporal databases supporting schema evolution, this is no longer necessarily true. We want to make this invisible by means of an appropriate definition and use of the temporal catalogues.

In the following we outline how we propose to describe and divide information of temporal catalogues, with respect to the schema changes taken into account.

Since it is possible to change the name of a relation, in order to identify the relation independently of the various names it assumes, we define a RELation IDentifier (REL_ID) which remains unique and unalterable throughout the database life. In the same way as the temporal key identifies a temporal record, the REL_ID and the temporal attributes identify a relation version. In the same way, since attribute renaming is allowed, also an ATTribute IDentifier (ATTID) is necessary, also unique and unalterable. The REL_ID and ATTID we propose to add can be defined by the system as *surrogates* [3].

The general information about the database schema is divided into four logical catalogues. REL_STAT keeps the general static information about the database relations, that is information which cannot be changed by a schema modification. REL_DYN maintains the dynamic portion of information of the whole schema, that is information which can be modified by one of the schema changes. Similarly, ATT_STAT contains all the general unchangeable information about the attributes of each relation and ATT_DYN keeps the changeable information. For the sake of simplicity in figure 1 we report only the attributes which are interesting to our purposes, intending the typical information of the general catalogues to be also included.

REL_STAT: (REL_ID, CREATOR, DB_NAME, IN, OUT, FROM, TO, TABLE or VIEW, ...)

REL_DYN:(REL_ID,REL_NAME,NUMBER_OF_ATTRIBUTES , IN, OUT, FROM, TO, DATA POOL ADDRESS)

ATT_STAT: (ATTR_ID, CREATOR, REL_ID, IN, OUT, FROM, TO, ...)

ATT_DYN: (ATT_ID, ATTR_NAME,ATTR TYPE, LENGHT OF ATTR., POSITION IN RECORD, IN, OUT, FROM, TO)

The temporal attributes added to the catalogues have a different meaning depending on the particular catalogue. In REL_STAT the transaction time IN determines the transaction which added the relation to the database; OUT records when it was dropped. The validity interval [FROM, TO) determines the validity time-span of the relation, independently from any schema change. In REL_DYN the interval [IN, OUT) determines the transaction life-span of a relation version and [FROM, TO) its validity life-span. In the same way, in REL_STAT, [IN, OUT) and [FROM, TO) indicate the transaction and validity time-spans of an attribute, whereas in REL_DYN they determine the life-span of an attribute version.

The following notation will be used.

- **To** : instant of the first insertion in the database.
- **Tnow** : current instant of transaction time.
- **T∞**: denotes that a temporal item has not been modified yet.
- OUT = T∞ indicates that a temporal item (schema, relation, tuple or data pool) is *actual*, i.e. it has not yet been modified by a successive transaction.

We say that a schema is *marginally (or partially) overlapped* by a schema change if their validity time-spans overlap only partially. We say a schema is *totally overlapped* by a schema change if its validity time-span is completely overlapped by that of the schema change.

3.1 Management of Schema Evolution

In the following paragraphs we describe a mechanism for the management of catalogues and data pools, first for schema redefinition, afterwards for schema revision, detailing the operations to perform on catalogues and data pools when schema incompatibility occurs. The situation is completely different in case of schema redefinition and schema revision. While in the first case the new schema is completely redefined on its validity interval, in case of revision the new schemas are obtained by applying the change to the previous ones. In Fig.1 the effects of schema redefinition and schema revision are shown. In Fig.1a we have the sequence of bitemporal schemas at the instant Tnow of the schema change. All the schemas are still actual with respect to transaction time and each one has its proper validity time-span. As an example, SCHEMA 2 is still actual and is valid on [t1,t2).

The schema change is performed at Tnow and is relative to the validity time-span [t',t").

In Fig.1b the situation after a schema redefinition is shown. SCHEMA 1 and SCHEMA N are unaltered, since their validity life-spans do not overlap that of the schema redefinition. SCHEMA 2 is marginally overlapped by [t',t"), thus the portion [Tnow, T∞) x [t1,t') of such schema is maintained. The region of SCHEMA 2 must be split into two parts; the portion before Tnow is archived, as shown by the vertical line; the portion [Tnow, T∞) x [t1,t') is restored. The same

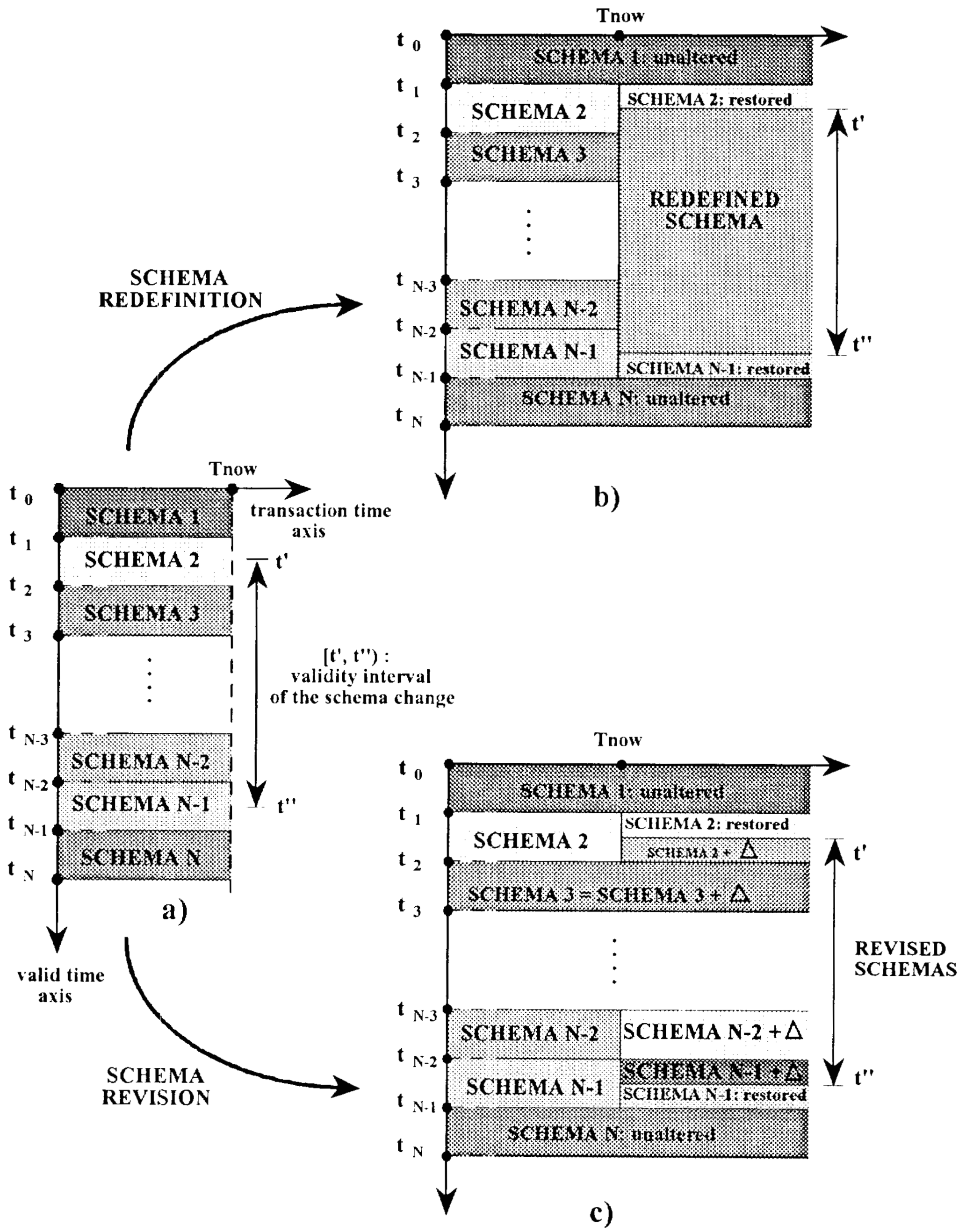

figure 1: (a) situation before schema change
(b) after redefinition or (c) schema revision
at transaction time T_{now} on the validity interval [t',t'')

applies to SCHEMA N-1, which is partially archived and partially restored. All the totally overlapped schemas are archived, as shown by the vertical line, and the new redefined schema is started on the life-span [Tnow, T∞) x [t1,t').

In Fig.1c the situation after a schema revision is shown. SCHEMA 1 and SCHEMA N do not overlap [t',t"), thus they remain unaltered. SCHEMA 2 and SCHEMA N-1 are partially archived and partially restored, exactly as in case of schema redefinition. Consider now the totally overlapped schemas, i.e. SCHEMA 3, ..., SCHEMA N-2. When a revision is applied to these schemas, two situations may occur:

1. The schema change performed on SCHEMA J leaves it unaltered. In figure, we indicate this by: **SCHEMA J = SCHEMA J +** Δ. (e.g. SCHEMA 3 in Fig.1c).
2. The schema change actually affects SCHEMA K. In this case, the existing SCHEMA K is archived, as indicated by the vertical line, and the revised schema, named **SCHEMA K +** Δ, is started. (e.g. SCHEMA N-2 in Fig.1c).

Schema Redefinition mechanism:

In case of redefinition the following operations must be performed for each marginally or totally overlapped relation:

1) The new schema description, the result of the redefinition, is stored in the catalogues. The description of the overlapped schema is archived (OUT = Tnow). If the new schema does not affect any previous schema (insertion of a completely new relation) no archiving is performed. If the schema is marginally overlapped, a schema, whose structure is identical to the one being redefined and with validity interval limited to the non-overlapped portion, is created. An example of this case is given in figure 2.b, by SCHEMA 2 and SCHEMA N-1.

2) If the new schema of the relation causes incompatibility with the previous one or if it is the first definition of a relation in the validity interval [t', t"], it is necessary to proceed as follows: archive the overlapped data pools, if any, create a new data pool, according to the new schema and, finally, restore the unaltered pool portions of the marginally overlapped schemas, if any.

Referring to the logical partition described in figure 1, the operations on catalogues and data pools, in case of schema redefinition, can be summarized as follows:

• ***Archiving***

All schemas, totally or partially overlapped, are identified by scanning REL_STAT and REL_DYN. The attribute OUT of all the schema versions identified is set to Tnow. ATT_STAT and ATT_DYN are managed in a similar way: all the tuples describing the attributes of the overlapped relations are no longer actual, thus OUT = Tnow. All the overlapped data pools are accessed and all records which are still actual are archived (OUT = Tnow).

A new record is added to REL_DYN, describing the new version of the relation schema. Furthermore, all records describing attributes in this new version are added to ATT_STAT and ATT_DYN. The temporal attributes are set to IN = Tnow, OUT = T∞, FROM = t', TO = t". A new data pool is started, according to the new structure. All the records which were still actual when the schema change occurred, are copied from the totally overlapped pools into the new data pool, according to the new structure. Their temporal attributes are reset as follows: IN = Tnow, OUT = T∞, FROM and TO unaltered. As far as the partially overlapped data pools are concerned, the records that are still actual and whose validity interval is totally contained in [t', t"), are copied into the data pool of the new schema, according to the new structure. Their temporal attributes are reset to IN = Tnow, OUT = T∞, FROM and TO unaltered. Records from partially overlapped data pools that are still actual and whose validity interval contains t' (or t") produce a copy which is inserted in the new data pool, according to the new structure with IN = Tnow, OUT = T∞, FROM = t' , TO unaltered. (or FROM unaltered, TO = t").

• ***Restoring marginally overlapped Schemas and data pools***

This case is managed separately because the partially overlapped schemas are only partially modified (see figure 2.b). For example, the portion of SCHEMA 2 relative to the validity interval [t1, t'] is untouched by the schema change. The restoring mechanism works as follows. A new record is added to REL_DYN, describing the structure of SCHEMA 2, with IN = Tnow, OUT = T∞, FROM = t1, TO = t'. ATT_STAT and ATT_DYN are managed in similar way, adding the tuples which describe the attributes of SCHEMA 2 along the two time axes. A new data pool is started, according to the structure of SCHEMA 2. The records belonging to the data pool of the original SCHEMA 2, which were still actual when the schema change occurred and whose validity interval is completely external to [t', t"] are copied into the marginal data pool with IN = Tnow, OUT = T∞, FROM and TO unaltered. Records that are still actual and whose validity interval contains t' (or t") produce a copy which is inserted in the new data pool, according to the old structure with IN = Tnow, OUT = T∞, FROM unaltered, TO = t' (or FROM = t", TO unaltered).

Schema Revision mechanism

In case of revision, each previous and still actual schema, totally or marginally overlapped, may result unaltered or altered by the schema change. Therefore, at least as many final schemas as those overlapped and incompatible are produced. In case of partially overlapped and incompatible pools two additional schemas are also produced. In case a schema is overlapped and is compatible, the revision mechanism concerns only the catalogue information. If the schema is incompatible, the data pools are also concerned and the following operations must be performed:

1) A new schema is generated applying the schema change to the previous schema relative to the same subinterval. Catalogue information is consequently updated.

2) A new data pool is started.

Referring to figure 1, the operations to perform on catalogues and data pools can be summarized as follows:

•***Archiving***

For each totally or marginally overlapped and incompatible Schema, the old description is archived in the catalogues (OUT = Tnow) and the overlapped data pools are also archived like in the case of archiving in schema redefinition.

•***Creation of the new Schema and data pool***

For each overlapped and incompatible schema, a record is inserted in REL_DYN, describing the new schema, which results from the union of the previous schema and the change. The Temporal attributes are set to IN = Tnow, OUT = T∞. For the totally overlapped schemas

FROM and TO maintain the previous values. For the marginally overlapped schemas FROM and TO assume the values of the endpoints of the overlapped portion. The catalogues are revised according to the kind of schema change to apply. For example, if an attribute length is modified, due to a *range change* or a *type change*, in ATT_DYN a new record is inserted, with the new attribute length and the revised records describing the attributes with the old length is archived.

•Restoring marginally overlapped Schemas and data pools

The restoring mechanism is the same as that in schema redefinition, with appropriate FROM and TO values.

4. Conclusions and further issues

In this paper the concept of schema change has been extended into the temporal concept of schema evolution, where all the database schemas, along both time dimensions, are maintained. The schema evolution has been distinguished into revision and redefinition. The former is the natural temporal extension of schema changes, while the latter allows the user to define a new schema disregarding the content of previous schemas of the same relation. In this paper we proposed a model for recording schema evolution and a mechanism to manage catalogues and data pools along both time dimensions when a temporal schema change occurs.

Future work will be devoted to the study of *selective versioning*, as requested by some temporal relational models. Selective versioning leads to the definition of two kinds of typically temporal schema changes: start and stop the versioning of an existing attribute. This is, in our opinion, an interesting issue, but naturally leads to a more complicated management of schema evolution and also involves semantic aspects. Another issue will be the study of the coexistence of different schemas in a single database relative to the same temporal intervals, which is an important topic for design application.

Bibliography:

[1] Dadam P., Lum V., Werner H.D.: Integration of Time Versions into a Relational Database System, Proc. of 10th VLDB, Singapore, August 1984, VLDB Endowment.

[2] Dadam P., Teuhola J.: Managing Schema Versions in a Time-Versioned Non-First-Normal-Form Relational Database, Proc. Datenbanksysteme in buro, technik und wissenschaft GI- Fachtagung, Darmstadt, 1-\.3. , April 1987.

[3] Grandi F., Scalas M.R., Tiberio P.: A History Oriented Data View and Operation Semantics for Temporal Relational Databases, C.I.O.C._C.N.R. report No. 76, Bologna, January 1991.

[4] Kim W., Chou H.T.: Versions of Schema for Object-Oriented Databases, Proc. of 14th VLDB, Los Angeles, California, 1988.

[5] Kim W., Ballou N., Chou H.T., Garza J.F., Woelk D.: Features of the ORION Object Oriented Database System, from *Object Oriented Concepts, Databases and Application*, ACM Press, 1989.

[6] Jensen C., Clifford J., Gadia S.K., Segev A., Snodgrass R.: A Glossary of Temporal Database Concepts, SIGMOD Record, Vol. 21, No. 3, September 1992 (to appear in 10).

[7] Martin N.G., Navathe S.B., Ahmed R.: Dealing with Schema Anomalies in History Databases, Proc. 13th VLDB, Brighton 1987.