# On Schema Versioning in Temporal Databases

Cristina De Castro  Fabio Grandi  Maria Rita Scalas
C.I.O.C.-C.N.R. and D.E.I.S., Università di Bologna
Bologna, Italy

**Abstract**

The support of schema versioning has been considered in the literature on temporal databases only at a limited extent. In particular, solutions for managing schema versions along transaction-time as different interfaces on the same temporal data were proposed so far.

In this paper we investigate the distinct functionalities of new solutions for schema versioning along *valid-* and *transaction-time* in a temporal relational environment. The support of schema versioning implies operations both at intensional and extensional level. Two distinct design solutions (*single-* and *multi-pool*) are presented for the management of extensional data in the presence of schema versioning. Moroever, a further distinction is introduced to define *synchronous* and *asynchronous* versioning of data and schemas.

The proposed solutions differ in the semantics and in the possible operations they support. The mechanisms for the selection of data through a schema version is strictly related to the particular schema versioning soultion, and has also influences on the data definition and manipulation language at user-interface level. We also show how the temporal language TSQL2, originally designed to support transaction-time schema versioning, can accordingly be extended.

## 1 Introduction

Two time dimensions are usually considered in temporal databases [Soo91, TSC$^+$93]: *transaction-time*, which tells when an event is recorded in a database, and *valid-time*, which tells when an event occurs, occurred or is expected to occur in the real world [JCE$^+$94]. According to the temporal dimensions they support, temporal databases can be classified as *monotemporal* (transaction- or valid-time), *bitemporal* or *snapshot*. Transaction-time DBs record all the versions of data inserted, deleted or updated in successive transactions (current and non current versions). Valid-time DBs maintain the most recently inserted versions of data, each relative to a distinct valid-time interval (current versions, forming the present historical state). Bitemporal DBs support both transaction and valid-time and thus maintain all the valid-time versions recorded in successive transactions (present and past historical states). Snapshot DBs do not support time: they maintain only the most recently inserted (current) version. A DB in which relations with more than one temporal format (e.g. snapshot, valid-time and bitemporal relations) coexist can also be called *multitemporal* [DGS94a].

When a *schema change* is applied to a traditional database, the current schema is usually substituted by a brand new version. The data corresponding to the past schema are lost or restarted according to the new schema if the database supports *schema evolution*. In both cases, a portion of intensional information may be no longer available together with the corresponding piece of extensional information. In order to avoid information loss, the concept of *schema versioning* has been introduced [Rodd92b, JCE$^+$94]. In current literature [DT87, McS90, Rodd92a, RS94], several proposals have been made for the maintainance of schema versions along *transaction time* whereas the necessity for the support of schema versioning along *valid time* is still debated. We also studied transaction-time schema versioning in a multitemporal environment in [DGS94b].

In [DGS94c, DGS95], we considered the notion of schema versioning along valid-time which will be further emphasized in this paper. Whereas transaction-time schema versioning is sufficient for any *on-time* schema change, that is schema changes effective when applied, or for applications for which the exact time of application of a schema change is not crucial (this seems to be the case of most CAD/CAM/CIM applications), valid-time schema versioning is made necessary by database applications requiring *retro-* or *pro-active* schema changes. It can be noticed that retroactive changes are quite common in databases, both concerning extensional and intensional data. Like valid-time databases have been introduced to accomodate retroactive changes of extensional data, valid-time schema versioning is necessary to allow also retroactive changes of intensional data. For instance, retroactive changes of intensional data can be enforced by changes in laws with retroactive effects (e.g. new encoding rules requiring more digits can be stated for social security numbers today, but effective from 1/1/1996) or, even more likely, they can be a consequence of *deferred updates* (e.g. the new encoding rules are stated and effective now but the corresponding schema change will be applied to the database only next month). Also proactive changes of intensional data are possible in consequence of particular design choices (i.e. in *what-if* analysis during development or maintainance of database applications), and require valid-time schema versioning to be supported. Advanced database application may also require bitemporal schema versioning, when not only must retro- and pro-active schema changes be managed but it is also necessary to *keep track* of them in the database (e.g. for auditing purposes). As it happens to extensional updates in bitemporal databases, this is only possible by means of schema versioning along both time dimensions.

Moreover, as far as the extensional aspects of schema versioning are concerned, we consider two distinct solutions for the management of data and show how each solution works in reply to schema changes, queries or updates. In the following, by the term *data pool* we denote a repository for extensional data. The solutions for organization and management of data pools are presented and discussed at logical level, without entering physical design details. The two solutions we consider are [DGS94c]:

- **Single-Pool Solution** The data corresponding to all schema versions are maintained into a single data pool.
- **Multi-Pool Solution** Distinct data pools are maintained for distinct schema versions [SCD93, DGS94b].

The single-pool solution is consistent with the solution adopted in [RS94] in [TSQL2] for schema evolution and schema versioning along transaction-time.

Another degree of freedom in data management, which takes place when extensional and intensional data are versioned along the same temporal dimension(s), gives rise to the following distinction [DGS94c]:

- **Synchronous Management** Data are stored, retrieved and updated always through the corresponding schema version, that is the schema version having the same validiy of data along the common temporal dimension(s). Synchronous management implies *synchronous versioning*, where the temporal pertinence of a schema version must include the temporal pertinence of the corresponding data along the common temporal dimensions.
- **Asynchronous Management** Data can be retrieved and updated through any schema version, whose validity is independent of the validity of data also along common temporal dimension(s). Asynchronous management gives rise to *asynchronous versioning*, that is the temporal pertinence of a schema version and the temporal pertinence of the corresponding data are completely independent.

In all the schema versioning proposals published in literature, asynchronous versioning seems to be adopted. As a matter of fact, the schema version to be used for data access can be specified independently from the time pertinence of data and separate linguistic tools are provided to this purpose in query language extensions. For instance, in the TSQL2 [TSQL2, RS94] query:

```
SELECT * FROM REL
   WHERE VALID(REL) OVERLAPS '1/1/80'
   SCHEMA | '1/1/90' |
```

can be used to retrieve (extensional) data valid on 1/1/80 from the relation `REL` using the current schema version as of 1/1/90. Since there is no correlation between the time pertinence of data and schemas, we must be in the presence of "asynchronous" management in order to express (and answer) such a query. In this case (e.g. if `REL` is a valid-time relation), the asynchronous management is guaranteed, in a natural way, by the *orthogonality* of the time dimensions. As a matter of fact, we always have asynchronous management between *different* time dimensions. But assume that `REL` is a transaction-time relation, transaction-time schema versioning is still used, and consider the following query:

```
SELECT * FROM REL
   WHERE TRANSACTION(REL) OVERLAPS '1/1/80'
   SCHEMA | '1/1/90' |
```

Well, this is a legal query only in the case of synchronous management, because different transaction times are to be used for the selection of data and schema (and it seems to be a correct TSQL2 query). In our opinion, synchronous management for transaction-time repesent a very hybrid solution which strains the semantics of transaction-time. It should be kept in mind that reference to transaction-time in the past has the meaning of a *rollback* operation, bringing back the database to a past state of its life. Therefore, the states in which the extensional and the intensional parts of the database can be brought back should be consistent. This is the reason for which synchronous management for transaction-time schema versioning is mandatory in our approach. If an application requires asynchronous management of data and schemas, valid-time schema versioning should be employed. Since no special constraints are enforced by the semantics of valid time, valid-time schema versioning can either be synchronous or asynchronous.

The rest of the paper is organized as follows: in Section 2 we present the single- and the multi-pool solutions, both in the synchronous and asynchronous case and show how they work in valid- and in transaction-time schema versioning. The differences are discussed also on the basis of some examples. Section 3 is devoted to the query language extensions and semantics in the considered environment. The use of different schema versions to manipulate data is illustrated on the basis of the features of each type of extensional management. Examples are provided in order to show how different results can be obtained in the presence of different schema versioning solutions.

## 2 Schema Versioning

In this section we introduce valid-time, transaction-time and bitemporal schema versioning, and describe mechanisms for their support and management. The operations on schema versions and the corresponding data are illustrated by means of figures and examples. More details can be found in [DGS94c].

The data definition language must be extended in order to support all kinds of schema versioning. We show how the TSQL2 temporal extension of SQL-92 [TSQL2, SAA$^+$94] could be upgraded to this purpose. TSQL2 is already designed to support schema transaction-time schema versioning [RS94]. Its data definition language is provided with the CREATE and ALTER statements [SG94], for the creation and modification of a table, respectively.

The statement:

CREATE TABLE $< table\ name > < table\ elements >$
   $< temporal\ definition >$

allows the definition of a new table named $< table\ name >$, where $< table\ elements >$ defines the non temporal attributes of the table and $< temporal\ definition >$ specifies the temporal format of the table (type of data versioning). For instance, the statement:

CREATE TABLE REL$(a_1 : d_1,\ a_2 : d_2,\ a_3 : d_3)$
   AS VALID AND TRANSACTION

defines a bitemporal table with non temporal attributes $a_1,\ a_2,\ a_3$. The domains of such attributes are $d_1,\ d_2,\ d_3$, respectively. For the sake of simplicity, domains will be omitted in the following. The TSQL2 statement ALTER TABLE allows a change to be effected on a relation schema.

In a multitemporal environment, schema changes should also include the change of the table temporal format [SG94, DGS94b, DGS94c]. To this end, ADD or DROP clauses with VALID or TRANSACTION specification must be used. For instance, the statement:

```
ALTER TABLE REL
   DROP TRANSACTION
```

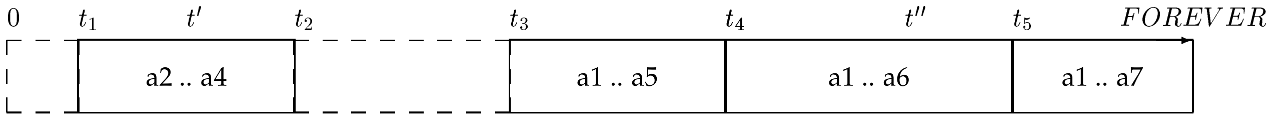


Fig.1a

```
ALTER TABLE REL
DROP COLUMN a5
VALID [t',t'']
```

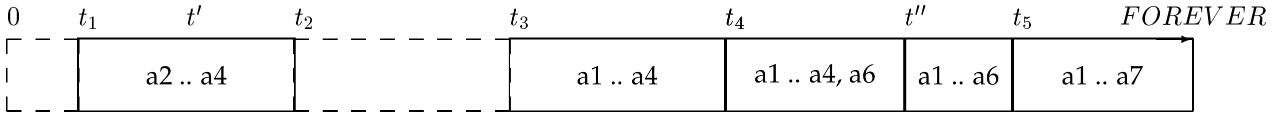


Fig.1b

Figure 1: situation before (1a) and after (1b) the schema change "`DROP a5`" valid in $[t', t'']$

changes the temporal format of table `REL` from bitemporal to valid-time.
In transaction-time schema versioning, a `CREATE` or `ALTER` statement always concerns the current schema version, thus, no time for the new schema (version) can be specified: the implicit transaction-time pertinence of the schema change is always $[NOW, UC]$, and cannot be changed by the user. The symbol $UC$ has the meaning of "until changed" and is used to timestamp only current data (not changed yet). On the contrary, in valid-time or bitemporal schema versioning, a new clause is introduced to allow the user to specify the validity of the schema change. Therefore, in our extension, the `CREATE` and `ALTER` statements are augmented with a `VALID` clause [DGS94c]. For instance, the statement:

```
ALTER TABLE REL
   ADD COLUMN a6 : d6
   VALID [t',t'']
```

requests the creation of a new schema version valid in $[t', t'']$ for relation `REL`, where attribute $a_6$ is added.

## 2.1 Valid-Time Schema Versioning

In all the prospected solutions, management of *intensional data* does not present new or peculiar difficulties. In valid-time schema versioning, system catalogues are implemented by means of valid-time relations, whose tuples correspond to schema versions. The tuple timestamps represent the validity of each schema version. Therefore, the management of intensional data in response to a schema change in valid-time schema verisoning reduces to the update of a valid-time relation. The only difference (and complication) from transaction-time versioning is that more than one schema versions may be interested by a single change. As a matter of fact, whereas in transaction-time versioning only the current schema version is affected by the change [DGS94b], in valid-time versioning all the schema versions totally or even partially overlapped by the validity of the change are affected. An comprehensive example is given in Fig. 1, which illustrates at intensional level the schema change "`DROP COLUMN a5`" with validity $[t', t'']$. The situations before and after the change are shown in Figg. 1a and 1b, respectively. Among all the schema versions, two are partially overlapped by $[t', t'']$. The version relative to $[t1, t2]$ is unaffected, because it does not contain the attribute a5, and so it remains unchanged. The schema version relative to $[t4, t5]$ is partially overlapped and actually affected by the change. This version is thus split into two portions: the non-overlapped portion maintains all its old attributes a1..a6, whereas the overlapped portion loses the attribute a5 becoming (a1 .. a4, a6). The schema version relative to $[t3, t4]$ is also affected and totally overlapped, thus a5 is dropped from this version.

Furthermore, the only difference between single- and multi-pool versioning at intensional level is that, in the latter case, pointers to the new data pool(s) must be stored in the catalogues (see [DGS94c] for details).

As far as extensional data management is concerned, the different strategies qualify the single- versus the multi-pool solution.

### 2.1.1 Single-Pool Solution:

The single-pool consists of a repository where all the extensional data are stored according to a global schema (*completed schema* in [SAA$^+$94]), which includes all the attributes introduced so far by successive schema changes. If a schema change is destructive, such as the drop of an attribute or the restriction of a domain, the change can only be recorded in the catalogues, since no data can be discarded from the single-pool. On the contrary, if a change adds an attribute, a temporal dimension or extends a domain, the whole data pool is converted to the new format. Note that if the change concerns a domain, the attribute must be extended to the largest one defined so far. If the change of a domain produces a new domain incompatible with the old one (e.g. when changing an attribute $CODE$ from numeric to alphabetic), two attributes must be maintained, with the same name as seen by the user, but corresponding to different domains and belonging to different schema versions as recorded in the catalogues. Since the change of temporal format is also allowed, if a schema change adds new temporal dimensions the whole data pool must be converted to the enlarged temporal format, using the temporal conversion maps defined in [DGS94a], as shown in [DGS94b]. Data are thus maintained according to the largest schema and in the largest temporal format so far defined and only the catalogues maintain the history of the changes. This solution does not minimize the data space relative to each schema version.

For example, Tabb.1–3 respectively show the evolution of the single pool of the snapshot table $Employee(EMP\_NAME, ADDRESS)$ before and after the schema changes of conversion to the valid-time format and successive addition of the attribute $PHONE$. The first change is supposed to span from 1985, some update activity concerning the Employee Jones occurs between the two changes and, in 1995, the attribute PHONE is added.

| EMP_NAME | ADDRESS |
|---|---|
| Brown | London |
| Jones | Edimborough |
| Rossi | Rome |
| Matisse | Paris |

Table 1: Single-pool for $Employee$ before the schema changes

| EMP_NAME | ADDRESS | VALID TIME |
|---|---|---|
| Brown | London | $\{1985..FOREVER\}$ |
| Jones | Edimborough | $\{1985..FOREVER\}$ |
| Rossi | Rome | $\{1985..FOREVER\}$ |
| Matisse | Paris | $\{1985..FOREVER\}$ |

Table 2: Single-pool for $Employee$ after the addition of transaction-time

In the single-pool solution, two pieces of information are necessary: the current structure of the single pool (all the attributes defined so far and the largest temporal format) and the structure and temporal format of each schema version. The original structure and temporal format of data in each schema version can thus be reconstructed using the catalogue information.

There are no differences between the synchronous and the asynchronous case as far as schema changes are concerned.

| EMP_NAME | ADDRESS | PHONE | VALID TIME |
|---|---|---|---|
| Brown | London | NULL | $\{95..FOREVER\}$ |
| Jones | Edimborough | NULL | $\{80..87\}$ |
| Jones | New York | 4040404 | $\{89..FOREVER\}$ |
| Rossi | Rome | NULL | $\{95..FOREVER\}$ |
| Matisse | Paris | NULL | $\{95..FOREVER\}$ |

Table 3: Single-pool for $Employee$ after the addition of $PHONE$

#### 2.1.2 Multi-Pool Solution:

The multi-pool solution requires the creation of as many data pools as the number of schema versions.

In valid-time schema versioning, data pools underlying unaffected and totally or partially overlapped schema versions are left untouched. For each affected and totally overlapped schema version a new data pool substitutes the previous one; a new data pool is created for each partially overlapped and affected schema version and a copy of the old pool remains connected to each of the non-overlapped portions of the original schema.

One of the main differences between the single- and the multi-pool solution is the following: a schema change applied to several schema versions produces different results on the different versions. This does not cause substantial differences in the intensional management, but the extensional management must be differentiated on each data pool.

#### 2.1.3 Asynchronous Multi-pool solution for valid-time schema versioning

In this case, each data pool is formatted according to the corresponding schema version. Therefore, when a new data pool is started the tuples are copied from the initial affected pool, according to the change applied to the previous schema.

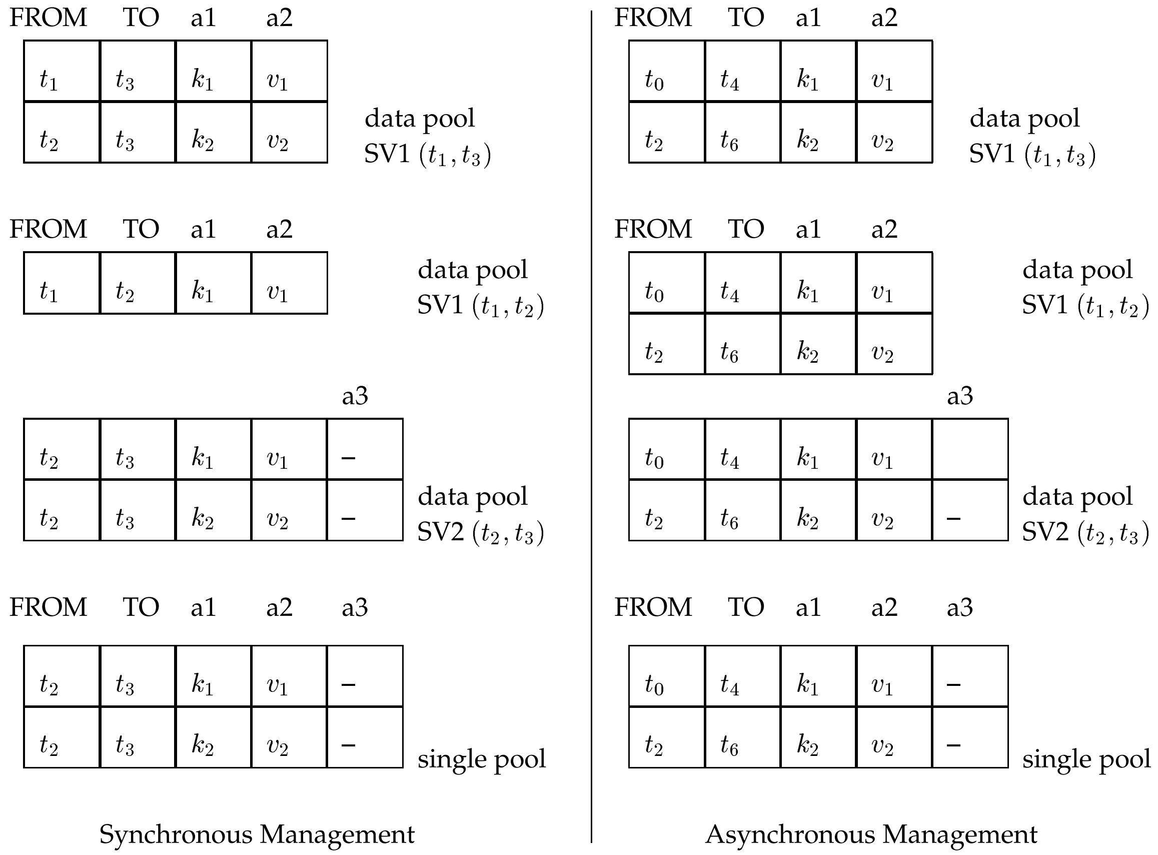
#### 2.1.4 Synchronous Multi-pool solution for valid-time schema versioning

The synchronous management of the multi-pool solution is achieved by restricting the validity of data (if any) in each new pool to its intersection with the schema version validity (due to synchronous versioning, the validity of all data in the pool must be contained in the validity of the pool itself). Note that the above restriction might cause a loss of information on the original validity lifespan of extensional data.

For instance, suppose the schema version of Employee in Tab.4 to be valid in the whole valid-time universe $\{0 \, .. \, FOREVER\}$ and a schema change to occur, which adds the attribute $PHONE$ in the interval $[t', t''] = [90, FOREVER]$. The multi-pool synchronous management produces Tabb. 5–6. It can be noticed that some tuples (e.g. (Jones, New York, $\{89..FOREVER\}$)) may be replicated in both new versions with the valid-time interval split in two parts according to synchronous management. Another example is shown in Fig. 2: the left portion of Fig. 2 illustrates the temporal pertinence and the content of the multi-pool in the presence of valid-time schema versioning. Note that, in the presence of a single schema version, the single- and the multi-pool coincide. According to a change which overlaps SV1 on $[t_2, t_3]$, a further schema version SV2, composed of the attributes a1, a2, a3, is created. In the multi-pool, the insertion of SV2 produces a new data pool and the original tuples are split according to their validity and partitioned between the two pools. In the single-pool, no actual split is effected on the content of the data pool. The right portion of Fig. 2 shows an example for the asynchronous case. If the same schema change is applied in the asynchronous case, no split of data validity must be performed. In this case, the single-pool coincides with the new data-pool and no data duplication is performed.

## 2.2 Transaction-Time Schema Versioning

If transaction-time schema versioning is supported, schema changes concern only the current schema version and the versioning can only be synchronous, since transaction-time does not allow retroactive



| FROM | TO | a1 | a2 |
|---|---|---|---|
| $t_1$ | $t_3$ | $k_1$ | $v_1$ |
| $t_2$ | $t_3$ | $k_2$ | $v_2$ |

data pool SV1 $(t_1, t_3)$

| FROM | TO | a1 | a2 |
|---|---|---|---|
| $t_1$ | $t_2$ | $k_1$ | $v_1$ |

data pool SV1 $(t_1, t_2)$

| | | | | a3 |
|---|---|---|---|---|
| $t_2$ | $t_3$ | $k_1$ | $v_1$ | – |
| $t_2$ | $t_3$ | $k_2$ | $v_2$ | – |

data pool SV2 $(t_2, t_3)$

| FROM | TO | a1 | a2 | a3 |
|---|---|---|---|---|
| $t_2$ | $t_3$ | $k_1$ | $v_1$ | – |
| $t_2$ | $t_3$ | $k_2$ | $v_2$ | – |

single pool

Synchronous Management

| FROM | TO | a1 | a2 |
|---|---|---|---|
| $t_0$ | $t_4$ | $k_1$ | $v_1$ |
| $t_2$ | $t_6$ | $k_2$ | $v_2$ |

data pool SV1 $(t_1, t_3)$

| FROM | TO | a1 | a2 |
|---|---|---|---|
| $t_0$ | $t_4$ | $k_1$ | $v_1$ |
| $t_2$ | $t_6$ | $k_2$ | $v_2$ |

data pool SV1 $(t_1, t_2)$

| | | | | a3 |
|---|---|---|---|---|
| $t_0$ | $t_4$ | $k_1$ | $v_1$ | |
| $t_2$ | $t_6$ | $k_2$ | $v_2$ | – |

data pool SV2 $(t_2, t_3)$

| FROM | TO | a1 | a2 | a3 |
|---|---|---|---|---|
| $t_0$ | $t_4$ | $k_1$ | $v_1$ | – |
| $t_2$ | $t_6$ | $k_2$ | $v_2$ | – |

single pool

Asynchronous Management

Figure 2: synchronous and asynchronous valid-time schema versioning

| EMP_NAME | ADDRESS | VALID TIME |
|---|---|---|
| Brown | London | $\{95..FOREVER\}$ |
| Jones | Edimborough | $\{80..87\}$ |
| Jones | New York | $\{89..FOREVER\}$ |
| Rossi | Rome | $\{95..FOREVER\}$ |
| Matisse | Paris | $\{95..FOREVER\}$ |

Table 4: portion $[0 .. FOREVER]$ of the multi pool of $Employee$ before the addition of $PHONE$

| EMP_NAME | ADDRESS | VALID TIME |
|---|---|---|
| Jones | Edimborough | $\{80..87\}$ |
| Jones | New York | $\{89..90\}$ |

Table 5: portion $[0 .. 90]$ of the multi pool of $Employee$ after the addition of $PHONE$

or proactive changes. In this case, the catalogues are defined and managed as transaction-time tables as shown in [DGS94b].

### 2.2.1 (Synchronous) Single-pool solution for transaction-time schema versioning

As far as the extensional level is concerned, all the tuples of the current data pool must be converted to the enlarged format, as described for valid-time versioning.

### 2.2.2 (Synchronous) Multi-pool solution for transaction-time schema versioning

In this case, a separate data pool, even if archived, must be maintained for each transaction-time schema version. The management of extensional information requires two distinct phases: the start of a new data pool followed by the archiving of the previous one. The format of the new data pool consists of all and only attributes contained in the new schema version. Again, such attributes may be temporal (timestamps). The initialization of the new pool also requires the retrieval of data from the old one. This solution requires only the current tuples of the old pool to be copied into the new one. This choice is done according to the semantics of synchronous versioning, which does not allow modifications of non-current tuples, even if, from a logical point of view, they could also be contained in the data pool of the current schema version.

If the change does not alter the temporal format of data, the selection is effected as follows: if the tuples of the old pool are snapshot or valid-time, they are all copied into the new data pool (all current data). If they contain transaction-time, only the current tuples (whose transaction timestamp includes the present time) are copied into the new data pool.

If the change concerns the temporal format of the table, a copy of the tuples in the old pool is first converted to the desired temporal format; among the resulting tuples, all and only the current ones are actually copied into the new data pool [DGS94c, DGS95].

After the initialization of the new pool, the old one is archived: if data in the source pool are snapshot or valid-time, they remain unaltered, and they are implicitly archived by the archiving of their schema version, i.e. the user knows that such data can be not anymore current since they belong to an archived schema version. If data contain transaction-time, the current tuples are archived by setting the enpoint of their timestamp to the present transaction-time, $NOW$, of the schema change.

For example, consider the transaction-time relation $Dept - Mgr$ in Tab.7 and suppose that the following schema change occurs: the attribute $SALARY$ is dropped at $NOW = 1994$. The after situation in the single pool is unaltered, since no data can be discarded from it. The drop of the attribute can only be recorded in the catalogues. As far as the multi-pool is concerned, the archived pool and the new one are shown in Tabb. 8–9. Note that the current tuples only are copied into the current pool.

| EMP_NAME | ADDRESS | PHONE | VALID TIME |
|---|---|---|---|
| Brown | London | NULL | $\{95..FOREVER\}$ |
| Jones | New York | NULL | $\{90..FOREVER\}$ |
| Rossi | Rome | NULL | $\{95..FOREVER\}$ |
| Matisse | Paris | NULL | $\{95..FOREVER\}$ |

Table 6: portion $[90 .. FOREVER]$ of the multi pool of $Employee$ after the addition of $PHONE$

| MANAGER | DEPT | SALARY | TRANS. TIME |
|---|---|---|---|
| Matisse | Food | 1000 | {80.. 90} |
| Matisse | Toys | 1500 | {91.. 92} |
| Matisse | Clothing | 2000 | {93.. UC} |
| Jones | Food | 900 | {73.. 83} |
| Jones | Clothing | 1800 | {84.. UC} |

Table 7: Transaction-time relation $Dept - Mgr$

## 2.3 Bitemporal Schema Versioning

Bitemporal schema versioning allows the maintainance of all the valid-time schema versions as inserted in successive schema changes. A schema change performed at transaction-time $NOW$ with validity $[t', t'']$ can only concern the current and overlapped bitemporal schema versions. The management of bitemporal schema versioning at intensional level is equal to the update of a bitemporal relation [GST91]. Fig. 3 shows the bitemporal counterpart of the example in Fig. 1.

As far as operations on extensional data are concerned, the management of the single-pool solution is substantially the same as discussed in synchronous or asynchronous valid-time schema versioning and (synchronous) transaction-time schema versioning. In the single-pool, when a schema change is applied, the whole data pool is converted to the "enlarged" format of the new schema version. In the multi-pool, it is necessary to initialize as many data pools as the number of new schema versions obtained by applying the change to the current ones. When a partially or totally overlapped schema version is affected, a data pool is started for each affected portion determined by the valid-time interval of the schema change. The current tuples are copied into the new pool, according to the rules of transaction-time schema versioning. In the case of asynchronous versioning, the valid-time pertinence (if any) of the tuples is not furtherly modified. In the case of synchronous versioning, it is restricted to that of the corresponding schema version. All the data pools corresponding to affected schema versions must be archived.

# 3 Data Manipulation

In this section we show which extensions are required to the query language in a system supporting one of the proposed schema versioning solutions.

## 3.1 Data Retrieval

The TSQL2 data manipulation statements (the `SELECT` statement and the modification statements) are provided with a `SCHEMA` clause in order to support schema version selection [RS94]. Therefore, TSQL2 is designed to work in a system supporting, in our terminology, asynchronous single-pool transaction-time schema versioning. It can be noticed that this particular solution is not permitted in our approach when the table is transaction-time or bitemporal, since transaction-time versioning cannot be asynchronous. The TSQL2 `SELECT` statement has a `SCHEMA` clause devoted to transaction-time specification for schema version selection. The same syntax of TSQL2, but with different semantics, can be used in our proposal only for valid-time schema selection in asynchronous schema versioning. In (synchronous) transaction-time schema versioning, no `SCHEMA` clause is allowed, since the same transaction-time specifications

| MANAGER | DEPT | SALARY | TRANS. TIME |
|---|---|---|---|
| Matisse | Food | 1000 | {80.. 90} |
| Matisse | Toys | 1500 | {91.. 92} |
| Matisse | Clothing | 2000 | {93.. 94} |
| Jones | Food | 900 | {73.. 83} |
| Jones | Clothing | 1800 | {84.. 94} |

Table 8: Archived pool of $Dept - Mgr$ after the change

| MANAGER | DEPT | TRANS. TIME |
|---|---|---|
| Matisse | Clothing | {94.. UC} |
| Jones | Clothing | {94.. UC} |

Table 9: Current pool of $Dept - Mgr$ after the change

used for data selection (contained in the `WHERE` clause) are also used for schema selection. Also in synchronous valid-time schema versioning no `SCHEMA` clause is allowed, since the valid-time specifications in the `WHERE` clause are used for both schema and data selection.

In valid-time schema versioning, synchronous and asynchronous management deeply differ as to schema selection mechanisms. In the synchronous management, the temporal conditions on data valid-time, if any, specified in a `WHERE` clause for extensional data are to be used also for schema selection, whereas any possible schema can be specified in case of asynchronous management. For instance, consider the following query and suppose that the bitemporal table `REL` is subject to asynchronous valid-time schema versioning:

```
SELECT * FROM REL
   WHERE VALID(REL) OVERLAPS '1/1/80'
      AND TRANSACTION(REL) FOLLOWS '1/1/90'
   SCHEMA | [1/ 1/ 75, 1/ 1 / 85] |
```

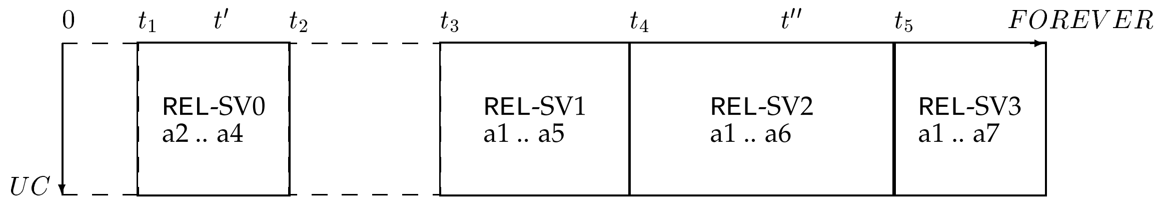
The processing of this query uses all the schema versions of `REL` whose validity overlaps the interval $[1/1/75, 1/1/85]$. Among all the data corresponding to such schemas, the query retrieves the data whose validity overlaps $1/1/80$ and whose transaction-time interval follows $1/1/90$. If more than one schema version are valid in the interval $[1/1/75, 1/1/85]$, as many "copies" of data are retrieved as many schema verions are found.

In case of multi-pool solution, a distinct data pool corresponds to each selected schema version and, thus, data can directly be retrieved in the format they are stored in the pool. In case of single-pool, all the data are stored in the same "enlarged" format and, thus, a filtering phase (including conversion of temporal format) is required in order to adapt retrieved data to the different schema versions involved. If also synchronous verisoning is adopted, the filtering phase must also restrict the time pertinence of retrieved data, along the same temporal dimension(s) used for synchronous versioning, to be completely contained within the temporal pertinence of the corresponding schema version.

Finally, it should be noticed that in the presence of synchronous management, data are always accessed through the schema versions by means of which they were last modified.

### 3.2 Data Modification

The same considerations made for the retrieval statements can be extended to update operations. In transaction-time schema versioning, updates only concern current data and act through the current schema version, thus a `SCHEMA` clause is not allowed. The same applies to synchronous valid-time versioning. In asynchronous valid-time versioning, a `SCHEMA` clause can be used in `INSERT`, `DELETE`, `UPDATE` statements in order to specify the validity of the schema version(s) which must be used for data modification.



```
ALTER TABLE REL
DROP COLUMN a5
VALID [t', t'']
```

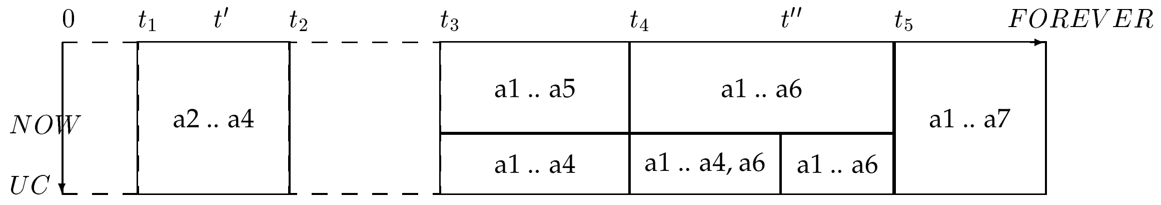


Figure 3: Bitemporal Schema versioning: situation before and after a schema change performed at $NOW$ and valid in the interval $[t', t'']$

In case of synchronous versioning, the temporal pertinence of appended data is always restricted to be contained, along the temporal dimension(s) of synchronous versioning, in the temporal pertinence of the corresponding schema version.

We try now to evaluate by means of some examples how the single- and the multi-pool solutions differently behave in case of valid-time and transaction-time schema versioning.

### 3.2.1 Updates in valid-time schema versioning

Consider Fig. 4a and Fig. 4c which show the multi- and the single-pool of a valid-time relation whose schema is versioned along transaction-time. Suppose that the versioning is asynchronous and that the following updates are applied:

- Update v2 to v10 using SV1.
- Update v2 to v20 using SV2.

The results are shown in Fig. 4b for the multi-pool and in Figg. 4d–4e for the single-pool. Note that in the multi-pool the final result does not depend on the execution order of the updates, since the data pools are actually separated (Fig. 4c). If the same operations are performed on the single-pool, the result depends on the execution order of the updates (see Figg. 4d– 4e).

### 3.2.2 Updates in transaction-time schema versioning

Tabb.7 and 9 show respectively the single- and the current pool of $Dept - Mgr$. In both solutions, the current schema version is $Dept - Mgr(MANAGER, DEPT, TRANS - TIME)$. Suppose Jones becomes manager in department Jewellery and this is recorded at $NOW$ = 1995. The before and after situations are shown in Tab.10 (single-pool) and Tab.11 (multi-pool). Note that in the single-pool, which multi-pool before update

| FROM | TO | a1 | a2 |
|---|---|---|---|
| t0 | t4 | k1 | v1 |
| t2 | t6 | k2 | v2 |

| | | | | a3 |
|---|---|---|---|---|
| t0 | t4 | k1 | v1 | – |
| t2 | t6 | k2 | v2 | – |

Figure 4a

multi-pool after update

| FROM | TO | a1 | a2 |
|---|---|---|---|
| t0 | t4 | k1 | v1 |
| t2 | t6 | k2 | v10 |

| | | | | a3 |
|---|---|---|---|---|
| t0 | t4 | k1 | v1 | – |
| t2 | t6 | k2 | v20 | – |

Figure 4b

single-pool before update:

| FROM | TO | a1 | a2 | a3 |
|---|---|---|---|---|
| t0 | t4 | k1 | v1 | – |
| t2 | t6 | k2 | v2 | – |

Figure 4c

single-pool after update:

$v2 := v20$ (first)
$v2 := v10$ (second)

| FROM | TO | a1 | a2 | a3 |
|---|---|---|---|---|
| t0 | t4 | k1 | v1 | – |
| t2 | t6 | k2 | v10 | – |

Figure 4d

$v2 := v10$ (first)
$v2 := v20$ (second)

| FROM | TO | a1 | a2 | a3 |
|---|---|---|---|---|
| t0 | t4 | k1 | v1 | – |
| t2 | t6 | k2 | v20 | – |

Figure 4e

Figure 4: valid-time asynchronous schema versioning: different results in the multi- and the single-pool

still contains the dropped attribute $SALARY$, the new tuple a NULL $SALARY$ value. Thus, depending on which solution is adopted, the same transaction produces two different results.

| MANAGER | DEPT | SALARY | TRANS. TIME |
|---|---|---|---|
| Matisse | Food | 1000 | {80.. 90} |
| Matisse | Toys | 1500 | {91.. 92} |
| Matisse | Clothing | 2000 | {93.. UC} |
| Jones | Food | 900 | {73.. 83} |
| Jones | Clothing | 1800 | {84.. 95} |
| Jones | Jewellery | NULL | {95.. UC} |

Table 10: single-pool of the transaction-time relation Dept-Mgr after update

| MANAGER | DEPT | TRANS. TIME |
|---|---|---|
| Matisse | Clothing | {94.. UC} |
| Jones | Clothing | {94.. 95} |
| Jones | Jewellery | {95.. UC} |

Table 11: current pool of Dept-Mgr after update

A snapshot table composed by the attributes (a1,a2,a3) is shown in Fig. 5; suppose the following operations are performed: drop of attribute a2, update of tuple (k4,x4) to (k4,x5) and re-add of attribute a2. Fig. 5a1,5a2,5a3 and 5b show the results of such updates in the multi-pool and in the single-pool respectively. The successive re-add of the same attribute does not reconstruct the tuple values according to the original copies, but to the most recent copy of the data pool. In the multi-pool the "new" added attribute is filled up with nulls, whereas the single-pool simple still retrieves the values v1,v2,v3,v4 as proper of the latest schema version, as they were before the drop of a2.

## 4 Conclusions

In this paper we provided a study of valid- and transaction-time schema versioning. Our goal was to introduce different solutions for the management of schema versioning. We proposed two distinct storage solutions: single- and multi-pool. A further distinction has been made between synchronous and asynchronous versioning.

The different design solutions have been principally compared on the basis of their semantic properties, since a detailed analysis of storage, maintainance and query processing costs was beyond the scope of this paper. In conclusion, the single-pool solution avoids data duplication, even if it requires an enlargement of the data format required for storage. On the contrary, the multi-pool requires the duplication of the current portion of data in transaction-time schema versioning, and duplication of the whole affected pool(s) in case of valid-time schema versioning. Anyway, the multi-pool solution allows several independent evolutions of data, each one relative to a specific schema version, whereas a single copy is maintained in the single-pool. Furthermore, the single-pool forces data to be converted according to the desired schema version at query processing time.

Future work will be devoted to a more complete comparison of the proposed solutions in terms of system performance, taking into account distinct application requirements and workloads.

## References

[DGS94a] DE CASTRO C., GRANDI F., SCALAS M.R.: "Semantic Interoperability of Multitemporal Relational Databases", in *Entity-Relationship Approach - ER '93*, Lecture Notes in Computer Science, Vol. 823, Springer-Verlag, 1994.

| a1 | a2 | a3 |
|---|---|---|
| k1 | v1 | – |
| k2 | v2 | – |
| k3 | v3 | – |
| k4 | v4 | x4 |

Figure 5a1

| a1 | a3 |
|---|---|
| k1 | – |
| k2 | – |
| k3 | – |
| k4 | x4 |

Figure 5a2

| a1 | a2 | a3 |
|---|---|---|
| k1 | – | – |
| k2 | – | – |
| k3 | – | – |
| k4 | – | x5 |

Figure 5a3

| a1 | a2 | a3 |
|---|---|---|
| k1 | v1 | – |
| k2 | v2 | – |
| k3 | v3 | – |
| k4 | v4 | x5 |

Figure 5b

Figure 5: transaction-time schema versioning

[DGS94b] DE CASTRO C., GRANDI F., SCALAS M.R.: "Management of Schema Versions in Multitemporal Relational Databases", Second Italian Conference on Advanced Database Systems (SEBD '94), Rimini, Italy, June 1994 (*in Italian*).

[DGS94c] DE CASTRO C., GRANDI F., SCALAS M.R.: "Schema Versioning for Multitemporal Relational Databases", C.I.O.C.-C.N.R. Tech. Rep. No. 100, Bologna, July 1994.

[DGS95] DE CASTRO C., GRANDI F., SCALAS M.R.: "Extensional Data Management in Multitemporal Relational Databases Suporting Schema Versioning", submitted for publication, 1995.

[DT87] DADAM P., TEUHOLA J.: "Managing Schema Versions in a Time-Versioned Non-First-Normal-Form Relational Database", Proc. Datenbanksysteme in Büro, Technik und Wissenschaft, GI-Fachtagung, Darmstadt, April 1987.

[GST91] GRANDI F., SCALAS M.R., TIBERIO P.: "A History-oriented Data View and Operation Semantics in Temporal Relational Databases", C.I.O.C.-C.N.R. Tech. Rep. No. 76, Bologna, January 1991.

[JCE$^+$94] JENSEN C., CLIFFORD J., ELMASRI R., GADIA S.K., HAYES P., JAJODIA S. (editors), DYRESON C., GRANDI F., KAFER W., KLINE N., LORENTZOS N., MITSOPOULOS Y., MONTANARI A., NONEN D., PERESSI E., PERNICI B., RODDICK J.F., SARDA N.L., SCALAS M.R., SEGEV A., SNODGRASS R., SOO M.D., TANSEL A., TIBERIO P., WIEDERHOLD G.: "A Consensus Glossary of Temporal Database Concepts", *ACM SIGMOD RECORD*, Vol. 23, No. 1, March 1994

[JSS93] JENSEN C.S., SOO M.D., SNODGRASS R.T.: "Unification of Temporal Relations", Proc. 9th IEEE International Conference on Data Engineering (ICDE '93), Vienna, Austria, April 1993.

[Kli93] KLINE N.: "An update of the Temporal Database Bibliography", *ACM SIGMOD RECORD*, Vol.22, No. 4, December 1993.

[McS90] MCKENZIE E., SNODGRASS R.: "Schema Evolution and the Relational Algebra", *Information Systems*, Vol. 15, No. 2, 1990.

[Rodd92a] RODDICK J.F.: "SQL/SE - A Query Language Extension for Databases Supporting Schema Evolution", *ACM SIGMOD RECORD*, Vol. 21, No. 3, September 1992

[Rodd92b] RODDICK J.F.: "Schema Evolution in Database Systems - An Annotated Bibliography", *ACM SIGMOD RECORD*, Vol. 21, No. 4, December 1992

[RS94] RODDICK J.F., SNODGRASS T.: "Schema Versioning Support in TSQL2", a TSQL2 Commentary, in [TSQL2].

[SAA$^+$94] SNODGRASS R.T., AHN I., ARIAV G., BATORY D., CLIFFORD J., DYRESON C.E., ELMASRI R., GRANDI F., JENSEN C.J., KAFER W., KLINE N., KULKARNI K., CLIFF LEUNG T.Y, LORENTZOS N., RODDICK J.F., SEGEV A., SOO M.D., SRIPADA S.M.: "TSQL2 Language Specification", *ACM SIGMOD RECORD*, Vol.23, No.1, March 1994.

[SCD93] SCALAS M.R., CAPPELLI A., DE CASTRO C.: "A Model for Schema Evolution in Temporal Relational Databases", Proc. of 7th IEEE European Computer Conference (CompEuro '93), Paris Evry, May 1993.

[SG94] SNODGRASS R.T., GRANDI F.: "Schema Specification in TSQL2", a TSQL2 Commentary, in [TSQL2]

[Soo91] SOO M.: "Bibliography on Temporal Databases", *ACM SIGMOD Record*, March 1991.

[TSC$^+$93] TANSEL A., CLIFFORD J., GADIA V., JAJODIA S., SEGEV A., SNODGRASS R.T. (eds.), *Temporal Databases: Theory, Design and Implementation*, The Benjamin/Cummings Publishing Company, Redwood City, 1993.

[TSQL2] *The TSQL2 Language Design Commitee*: SNODGRASS R.T. (chair), AHN I., ARIAV G., BATORY D., CLIFFORD J., DYRESON C.E., ELMASRI R., GRANDI F., JENSEN C.J., KÄFER W., KLINE N., KULKARNI K., CLIFF LEUNG T.Y., LORENTZOS N., RAMAKRISHNAN R., RODDICK J.F., SEGEV A., SOO M.D., SRIPADA S.M., *The TSQL2 Language Specification*, September 1994 (available by anonymous ftp at site `cs.arizona.edu`).