PMTV: A Schema Versioning Approach for Bi-temporal Databases

Han-Chieh Wei and Ramez Elmasri
Department of Computer Science and Engineering
The University of Texas at Arlington
{wei, elmasri}@cse.uta.edu

Abstract

Most of the existing schema versioning approaches are developed for single time-dimension temporal databases. For bi-temporal databases, which allow both retroactive and proactive updates to the schema and database, these approaches introduce many problems and perform inefficiently. In this paper, we first review two schema versioning approaches, single table versioning (STV) and multiple table versioning (MTV) approaches, and also the proposed partial multiple table versioning (PMTV) approach. We prove the correctness of PMTV by showing it producing equivalent results to STV, and then compare these three approaches in database conversion and temporal query processing. The comparison shows that the proposed PMTV approach not only solves the problems but also performs more efficiently than the other approaches when applied to bi-temporal databases.

1. Introduction

The aim of temporal databases is to accommodate time dimensions to capture the dynamic character of real world objects. Not only database objects undergo change, but also the database schema changes due to the dynamic changes of application requirements or design specifications. *Schema versioning*[5,16], and its weaker companion, *schema evolution*[1,15,18], are two techniques dealing with changes made to the schema while retaining the consistency between the database and its schema. Basically, the difference between schema evolution and schema versioning is that schema evolution keeps only one (current) version of schema and corresponding data, whereas schema versioning preserves versions of schema and data during the evolution.

There are differences when these two approaches are implemented in snapshot databases versus in temporal databases. In snapshot databases, schema evolution only keeps the current schema and corresponding data. The old schema and legacy data are obsolete and will not be kept. However, temporal databases not only store the current data but also the historical data and even future information expected to occur. Therefore, if there is only one schema version, it needs to retain the full schema information. In temporal relational databases, this is recognized as *single table versioning* (or *complete table*) [3,4,29] (**STV**) approach.

For schema versioning, whenever the schema is changed, a new schema is created and defined as a new version of the schema. Accordingly, the new version of data is converted from the old data to be consistent with the new schema. Both the legacy schema and data are still stored in the catalog and database system as old versions. As a result, the legacy data is preserved and also the applications defined on the old schema need not be rewritten nor recompiled. This is true both in snapshot and temporal databases. However, in snapshot schema versioning, the concepts of time and history still cannot be included. That is, the schema changes can only apply to the current version. Changes to the past or plans for the future schema still cannot be captured. On the other hand, in temporal databases, especially bi-temporal databases [9,22,23], incorporating both transaction time and valid time, can fulfill the requirements of schema versioning not only because they allow the users or application programs to access temporal information but also because they allow retroactive and proactive updates. Therefore, if the database schema is stored in a bi-temporal catalog, it can provide the most flexibility for database schema evolution. However, there is a cost tradeoff between the flexibility of retroactive and proactive schema changes and the cost of implementing these mechanisms. The complexity is high because changes not only affect the current versions of data but also the past and even the future versions which makes the database conversion much more complicated than for conventional snapshot databases. The schema versioning approach is know as multiple table version approach (**MTV**) [4,29] in temporal databases.

Most previous research [5] discusses STV and MTV in transaction-time databases only. In our previous paper [29], we fully explored the effect of schema changes involving both retroactive and proactive updates in *bi-temporal* relational databases by comparing these two approaches. Algorithms for schema update and database conversion were defined for these two operations. The problems and complexity of the two approaches were also discussed. We then proposed a compromise approach, *partial multiple table versioning* (**PMTV**). In this paper, we will prove the correctness of the PMTV approach, and

0-7695-0756-5/00 $10.00 © 2000 IEEE

compare the time and space complexity of the three approaches. Query processing is the most critical problem for the PMTV approach because it requires EVENT-JOIN [19,20], which includes Temporal-NATURALJOIN and Temporal-OUTERJOIN and these are the most expensive operations in temporal databases. We propose the index join approach to solve this problem.

In the next section, we briefly review these three approaches and their problems and advantages by an example. In section 3, we prove the correctness of the PMTV approach. Section 4 compares the time and space complexity of the approaches for database conversion. The problem of bi-temporal query processing is discussed in section 5. Section 6 concludes this paper.

2. Schema versioning approaches

The following example reviews the three versioning techniques implemented in bi-temporal databases and discuss their problems. For simplicity, the example only shows the versions of one entity in the relation. More schema change examples and algorithms can be found in [29].

Example Assume that the bi-temporal relation Employee is created at time 10 with valid-time interval [10, *now*] and with the attributes employee ID, name, salary, and position. Figure 1 shows the current state of the relation at time 40, for the versions of Employee 'John'.

Employee

ID	Name	Salary	Position	VS	VE	TS	TE	
1	John	30k	P1	10	30	10	20	
1	John	30k	P1	10	20	20	UC	*
1	John	35k	P2	20	50	20	35	
1	John	35k	P2	20	30	35	UC	*
1	John	40k	P3	30	65	35	40	
1	John	40k	P3	30	60	40	UC	*
1	John	45k	P4	60	80	40	UC	*

Figure 1. The state of the Employee relation at time 40. The tuples with '*' are the current versions.

Assume that the following schema change is applied to the relation Employee:

SC: At time 50, a new time-varying attribute Bonus is added to Employee, which is valid from time 25 to 65, and John's bonus is recorded as 5% and is valid during [25,65].

2.1 Single Table Version (STV) approach

In single table version, each table has only one version throughout the lifetime of the database. As mentioned earlier, the schema needs to include the history of all changes. This idea is proposed in [14] as *complete schemata*, which follows the idea of *complete table* in [4]. A complete schema consists of tables defined over the union of attributes that have ever been defined for them. If attributes are dropped, the dropped attribute will still be retained in the database since in append-only temporal databases data will never be deleted. Therefore the record size, and hence the table size for this approach will only grow but never shrinks. Figure 2 shows the results of schema changes *SC* for STV.

Employee

ID	Name	Salary	Position	Bonus	VS	VE	TS	TE
1	John	30k	P1	null	10	30	10	20
1	John	30k	P1	null	10	20	20	UC
1	John	35k	P2	null	20	50	20	35
1	John	35k	P2	null	20	30	35	~~UC~~ 50
1	John	40k	P3	null	30	65	35	40
1	John	40k	P3	null	30	60	40	~~UC~~ 50
1	John	45k	P4	null	60	80	40	~~UC~~ 50

Part (a)

1	John	35k	P2	null	20	25	50	UC
1	John	35k	P2	5%	25	30	50	UC
1	John	40k	P3	5%	30	60	50	UC
1	John	45k	P4	5%	60	65	50	UC
1	John	45k	P4	null	65	80	50	UC

Part (b)

Figure 2. Relation Employee in STV after *SC*.

Three problems are identified in this approach: **space overhead**, **search overhead**, and **database availability**. As we can see from Figure 2, the problem of space overhead results from excessive *data duplication* and *null values* whenever the database is converted to conform to the schema change. In the single table version approach, every time an attribute is added to or dropped from a relation, all current versions of entities need to be checked to see if their valid time interval overlaps with the schema change. For temporal databases, which usually contain a large amount of data, the search time will be a large overhead. For the problem of database availability, when a new attribute is added or the type domain of an attribute is generalized, part of (or even the whole) database will not be available for a period of time due to the process of database conversion, which requires augmentation and reorganization of the storage space.

2.2 Multiple Table Version (MTV) approach

In multiple table version, every time a relation schema is changed, it creates a new table version. The current entity versions in the source table whose valid-time interval overlaps with interval of the schema change need to be copied into the newly created table along with the value of the new attribute (if the schema change is attribute addition).

Figures 3 shows the table versions and data converted from the source versions after *SC* is executed: version V_1 is created, the data are converted from the current entity versions marked with '*' in table version V_0.

(a) **Employee_V$_0$** (Valid lifespan: I_{v0} = [10,*now*])

ID	Name	Salary	Position	VS	VE	TS	TE	
1	John	30k	P1	10	30	10	20	
1	John	30k	P1	10	20	20	UC	
1	John	35k	P2	20	50	20	35	
1	John	35k	P2	20	30	35	UC	*
1	John	40k	P3	30	65	35	40	
1	John	40k	P3	30	60	40	UC	*
1	John	45k	P4	60	80	40	UC	*

(b) **Employee_V$_1$** (Valid lifespan: I_{v1} = [25,65])

ID	Name	Salary	Position	Bonus	VS	VE	TS	TE
1	John	35k	P2	5%	25	30	50	UC
1	John	40k	P3	5%	30	60	50	UC
1	John	45k	P4	5%	60	65	50	UC

Figure 3. (a) the original table. (b) after SC.

Three problems are found in the MTV approach [29]: **data duplication**, **multischema queries** [5], and **mandatory version creation**.

2.3 Partial Multiple Table Version (PMTV)

PMTV [29] uses the concept of temporal normalization [13]. For the schema change of attribute addition, since the new added attribute will not be synchronized with the existing attributes in the relation, the PMTV approach creates a bi-temporal relation with only the new attribute plus the key attribute (ID) of the relation being modified. The complete relation can be later reconstructed by applying the *Entity-Join* [19] operation. The result of the schema change *SC* is show in Figure 4.

Employee

ID	Name	Salary	Position	VS	VE	TS	TE
1	John	30k	P1	10	30	10	20
1	John	30k	P1	10	20	20	UC
1	John	35k	P2	20	50	20	35
1	John	35k	P2	20	30	35	UC
1	John	40k	P3	30	65	35	40
1	John	40k	P3	30	60	40	UC
1	John	45k	P4	60	80	40	UC

(a) Original Employee relation

Emp_Bonus

ID	Bonus	VS	VE	TS	TE
1	5%	25	65	50	UC

(b) New created relation for attribute Bonus after *SC*

Figure 4. Applying *SC* using partial multiple table version.

The PMTV approach solves all the problems in the STV and MTV approaches as presented in [29]. However, there are several issues that need to be investigated:

1. The correctness of the PMTV approach.
2. Does PMTV perform better than STV and MTV?
3. Reducing the cost of Entity-Join performed in PMTV to reconstruct the complete object information. Entity-Join [19] includes *temporal equal join* and *temporal outer join* which are the most expensive operations in temporal query processing.

3. Correctness of PMTV

In this section, we will prove the correctness of the PMTV approach by showing that the table state after a schema change applying STV approach can be losslessly composed by taking Temporal EntityJoin of the table versions generated by PMTV approach (The equivalence of STV and MTV are shown in [5]). Because of the space limit, we only present the proof of attribute addition.

Definitions:

R = < $ID, A_1, A_2, \ldots, A_n, VS, VE, TS, TE$ > is a bi-temporal relation schema.

$r_o = \{ t_i \mid t_i = <ID, VA_1, VA_2, \ldots VA_n, VS_t, VE_t, TS_t, TE_t> \}$ The set of instances of relation R before the schema change *SC*.

SC: add_attr(A_x, val_x, VS_x, VE_x, TT_x) The schema change *SC* add an attribute A_x with default value val_x at time TT_x with valid-time interval [VS_x, VE_x].

R_{sc}=<$ID, A_1, A_2, \ldots, A_n, \mathbf{A_x}, VS, VE, TS, TE$> The relation schema after *SC*.

$r_{sc} = \{ t_i \mid t_i = <ID, VA_1, VA_2, \ldots VA_n, \mathbf{VA_x}=null, VS_i, VE_i, TS_i, TE_i> \}$ The state of relation instance when attribute A_x is added at time TT_x. Figure 2(a) shows an example of r_{sc} where A_x is new added attribute Bonus.

$r_{cur} = \{ t_i \mid t_i \in r_{sc}, TE_i = UC \}$ The set of current versions of entities at time TT_x, e.g. the tuples with * mark in Figure 1 are in r_{cur}.

$r_{sc\_cur} = \{ t_i \mid t_i \in r_{cur}$ AND $[VS_i,VE_i]$ OVERLAP $[VS_x,VE_x] \}$ Tuples in r_{cur} whose valid-time intervals overlaps with [VS_x,VE_x].

$r_m = \{ t_i \mid t_i \in r_{sc}$ AND $(TE_i < TT_x) \}$

For **STV** approach, as shown in [29], we consider two sets of tuples in r_{sc_cur}: r_I and r_P.

r_I: the set of tuples in r_{sc_cur} whose valid-time interval is included in [VS_x, VE_x],

$r_I = \{ t_i \mid t_i \in r_{sc\_cur}, [VS_i,VE_i] \subseteq [VS_x, VE_x] \}$

r_P: the set of tuples in r_{sc_cur} whose valid-time interval is partially overlaped with [VS_x, VE_x],

$r_P = r_{sc\_cur} - r_I$

After schema change *SC*, we have

$r_I \Rightarrow r'_I \cup r_{new\_I}$, where

$r'_I = \{ t_i \mid t_i \in r_I, TE_i \leftarrow TT_x \}$

$r_{new\_I} = \{ t_i \mid t_i \in r_I, VA_x \leftarrow val_x, TS_i \leftarrow TT_x, TE_i \leftarrow UC \}$

, and

$r_P \Rightarrow r'_P \cup r_{new\_P1} \cup r_{new\_P2}$, where

$r'_P = \{ t_i \mid t_i \in r_p, TE_i \leftarrow TT_x \}$

$r_{new\_P1} = \{ t_i \mid t_i \in r_P, [VS_i,VE_i] \leftarrow ([VS_i,VE_i] - [VS_x,VE_x]), TS_i \leftarrow TT_x, TE_i \leftarrow UC \}$

$r_{new\_P2} = \{ t_i \mid t_i \in r_P, VA_x \leftarrow val_x, [VS_i,VE_i] \leftarrow ([VS_i,VE_i] \cap [VS_x,VE_x]), TS_i \leftarrow TT_x, TE_i \leftarrow UC \}$

Hence, applying the schema change *SC* to relation *R* with current set of instances r_{sc} using STV approach, gives the result:

$$R \xrightarrow{SC} R_{sc}$$
$$r_o \xrightarrow{SC} r_{STV}$$
$$r_{STV} = r_m \cup (r'_I \cup r_{new\_I}) \cup (r'_P \cup r_{new\_P1} \cup r_{new\_P2})$$

For the PMTV approach, after the schema change *SC*, a new bi-temporal table, R_A_x is created, with the set of instances r_{Ax}, in addition to the old table *R* and its instances r_o.

$R\_A_x = <ID, A_x, VS, VE, TS, TE>$

$r_{Ax} = \{ t_i \mid t_i = <ID_t, VA_x=val_x, VS_x, VE_x, TT_x, UC>\}$

To prove the equivalence between STV and PMTV, we need to show, that:

$r_{STV} \equiv r_{Ax}$ Entity-Join r_o

Entity-Join [19] (referred as Event-Join in [20]) joins together a temporal object that has been vertically partitioned into several relations via temporal normalization [13]. Several algorithms and implementation methods have been proposed for single time dimension entity-join [19,20,25]. The Entity-Join here is actually a bi-temporal entity-join. We are not going to investigate how these algorithms and implementations can be extended for bi-temporal Entity-Join. We are only interested in the join result. Here we adopt and extend the algorithm of sort-merge entity join from [20] because it is easy to present and understand.

The entity-join (r_o Entity-Join r_{Ax}) is done as: (r_o Temporal Natural-Join r_{Ax} on *ID*) $\cup$ (r_o Temporal Outer-Join r_{Ax}).

In the following algorithm, we assume that both relations are sorted on ID as the primary order and on TS as the second order. At each iteration, two tuples, $x_i \in r_o$ and $y_i \in r_{Ax}$, are compared to each other and one or more result tuples will be produced based on the relationship between the tuples on their ID values and time intervals (both valid-time and transaction-time intervals).

Algorithm:

```
(1) Read $x_i$, $y_j$ for each $x_i \in r_o$, $y_j \in r_{Ax}$, do the following steps:
(2) if $x_i$(ID) > $y_j$(ID) then read next y.
(3) else if $x_i$(ID) = $y_j$(ID), consider the following cases:
(3.1)   if $x_i$(TE) < $y_j$(TS) then
            -- generate an outer-join result tuple $t_x$ for $x_i$:
            $t_x$=<$x_i$(ID),$x_i$(A$_1$,..A$_n$),null,$x_i$([VS,VE]),$x_i$([TS,TE])>   (1)
(3.2)   else if $x_i$([TS,TE]) $\supset$ $y_j$([TS,TE])
(3.2.1)     if $x_i$([VS,VE]) $\subseteq$ $y_j$([VS,VE]) then
                -- generate an intersection tuple $t_{xy}$ for $x_i$,
                $t_{xy}$=<$x_i$(ID),$x_i$(A$_1$,..,A$_n$),$y_j$(A$_x$),$x_i$([VS,VE]),
                    $y_j$([TS,TE])>   (2)
                -- generate an outer join result tuple $t_x$ for $x_i$,
                $t_x$=<$x_i$(ID),$x_i$(A$_1$,..,A$_n$),null,$x_i$([VS,VE]),
                    $x_i$(TS), $y_j$(TS)>   (3)
(3.2.2)     elseif $x_i$([VS,VE])$\cap$$y_j$([VS,VE]) $\neq \emptyset$ then
                -- generate an outer join result tuple $t_{x1}$ for $x_i$,
                $t_{x1}$=<$x_i$(ID),$x_i$(A$_1$,..A$_n$),null,$x_i$([VS,VE]),
                    $x_i$(TS), $y_j$(TS)>   (4)
                -- generate an outer join result tuple $t_{x2}$ for $x_i$,
                $t_{x2}$=<$x_i$(ID),$x_i$(A1,..,An),null, $x_i$([VS,VE]) –
                    $y_j$([VS,VE]), $y_j$([TS,TE])>   (5)
                -- generate an intersection tuple $t_{x3}$ for $x_i$,
                $t_{x3}$=<$x_i$(ID), $x_i$(A$_1$,..,A$_n$), $y_j$(Ax), $x_i$([VS,VE]) $\cap$
                    $y_j$([VS,VE], $y_j$([TS,TE])>   (6)
(4) Read next tuple(s).
```

When applying this algorithm to implement r_o Entity-Join r_{Ax}, after schema change *SC*, we have the following input values for y_j:

For all $y_i \in r_{Ax}$, y_i([VS,VE]) = [VS$_x$,VE$_x$],
y_i(TS) = TT_x, y_i(TE) = *UC*,
$y_i(A_x) = val_{Ax}$

we'll get the following results:

For the set of tuples (1):
$\{t_x \mid t_x = <x_i(ID), x_i(A_1,..A_n), \textbf{null}, x_i([VS,VE]), x_i([TS,TE])>\}$
$\equiv r_m$

The set of tuples (2):
$\{t_{xy} \mid t_{xy} = <x_i(ID), x_i(A_1,..,A_n), val_{Ax}, x_i([VS,VE]), TT_x, UC>\}$
$\equiv r_{new\_I}$

Set of tuples (3):
$\{t_x \mid t_x = <x_i(ID), x_i(A_1,..,A_n), \textbf{null}, x_i([VS,VE]), x_i(TS), TT_x>\}$
$\equiv r'_I$

Set of tuples (4):
$\{t_{x1} \mid t_{x1} = <x_i(ID), x_i(A_1,..A_n), \textbf{null}, x_i([VS,VE]), x_i(TS), TT_x>\}$
$\equiv r'_P$

Set of tuples (5):
$\{t_{x2} \mid t_{x2} = < x_i(ID), x_i(A_1, .., An), \textbf{null}, x_i([VS,VE]) – y_j([VS,VE]), TT_x, UC > \}$
$\equiv r_{new\_P1}$

Set of tuples (6):
$\{t_{x3} \mid t_{x3} = < x_i(ID), x_i(A_1, .., A_n), y_j(Ax), x_i([VS,VE]) \cap y_j([VS,VE], TT_x, UC > \}$
$\equiv r_{new\_P2}$

r_{Ax} Entity-Join r_o
$= (1) \cup (2) \cup (3) \cup (4) \cup (5) \cup (6)$
$= r_m \cup (r_{new\_I} \cup r'_I) \cup (r'_P \cup r_{new\_P1} \cup r_{new\_P2})$
$= r_{STV}$

Therefore, the table state after schema change *SC* is equivalent for the STV and PMTV approaches. The equivalence of the afterwards update operations to the new tables version(s) for these two approaches has been recognized and can be found in [20].

4. Comparison of three approaches

The three approaches presented in this paper for schema change and database conversion can be characterized by the following costs:

1. *time*, the time needed to search for the current object versions before the database conversion,
2. *space*, the space required for the new object versions,
3. *query processing*, the time needed to process a temporal query.

In this section, we compare the space and time costs based on our previous research [29]. The time cost is measured as the number of block accesses. The space is measured in terms of the additional space used for the null values in the STV approach and the duplicated data in both STV and MTV approaches. For the PMTV approach, the additional space is the space of the new created tables for the added attributes.

The following are the definitions of the parameters:

Parameters

S_a :	average attribute size (byte).
B :	block size.
I_{vsc} :	a temporal element which is the valid lifespan of the schema change.
N_a :	average number of attributes in original table *R*.
N_{tp} :	number of tuples in table *R* before any schema changes.
$N_{ver,T}$:	average number of current versions of table *R* at time *T*.
N_e :	average number of distinct entities in table *R*.
N_O :	average number of current versions of each entity whose valid-time intervals overlap with I_{vsc}. where $N_O = N_P + N_F$
N_P :	average number of current versions of each entity whose valid-time intervals partially overlap with I_{vsc}.
N_F :	average number of current versions of each entity whose valid-time intervals included in I_{vsc}.
N_{add}:	the number of attribute addition operations.
N_{OT}:	the number of table versions in MTV whose valid-time intervals overlap with I_{vsc}.

4.1 Search time overhead

For both STV and MTV approaches, the first step of data conversion after a schema change is searching for the current versions of each object, and then selecting the versions whose valid-time interval intersects with the schema change. If there is no temporal index, for the STV approach, even though the effective time of the schema change is short, a complete scan of the table is necessary to find all current versions. Number of block accesses for STV approach:

$$(S_a \cdot N_a) N_{tp} / B$$

For the MTV approach, since the table versions represent a sort of temporal index, the tables that need to be scanned are limited to the table versions whose valid lifespans intersect with the schema change. However, if a schema change is applied with a long time interval, the search overhead may be as high as the STV approach.

Number of block accesses for MTV approach:

$$(S_a \cdot N_a)\ (N_{tp} / N_{ver,T}) \cdot N_{OT} / B$$

For the PMTV approach, there is no search needed since it is independent from the current table state. Therefore, the number of block accesses for PMTV approach is always zero.

Consider the following example:
Assume for some table *R*, S_a=8, N_a=8, and the block size B= 512 bytes. For the MTV approach, we assume $N_{ver,T}$=5 and the tuples are evenly distributed into each of the table versions. To compare the search time of the three approaches for data conversion, we assume that N_{tp} ranges from 1000 to 5000, and the number of table versions whose valid-time span overlaps with the schema change (N_{OT}) ranges from 1 to 5. The results are shown in Figure 5.

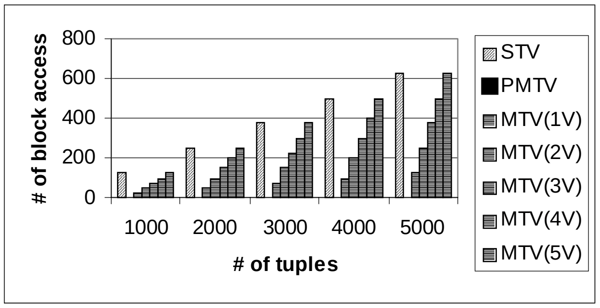

Figure 5. Search time for the current versions

We can see that the PMTV approach does not need any search where as the STV approach requires 125 to 625 block accesses as N_{tp} increased from 1000 to 5000. For the MTV approach, as the number of the overlapped table versions increases, the number of the block accesses also increases and eventually equals the STV approach when all the table versions overlap with the schema change.

4.2 Storage space cost

To compare the space complexity, we use the formulas defined in our earlier research [29]. The space required for attribute addition by STV is the space for the null values S_{null} and for the duplicated attributes S_{dup}.

$$S_{STV} = S_{null} + S_{dup}$$

$$= S_a * (N_{add} * N_{tp} + (2N_P + N_F) * N_e * \sum_{i=0}^{N_{add}-1} i) +$$

$$Sa * \left[N_{add} * N_a + \sum_{i=0}^{Nadd-1} i \right] * \left[(2 N_P + N_F) * N_e \right]$$

The space required for attribute addition by MTV is for the overlapped data version duplicated in the new table versions.

$$S_{MTV} = S_a * \sum_{i}^{Nadd} \left[\left(\sum_{j=1}^{i} (N_a + j) \right) * \frac{N_O}{i} * N_e \right]$$

The space required for attribute addition by PMTV is only the size of two attributes for each entity.

$$S_{PMTV} = N_{add} * [(S_a * 2) * N_e]$$

The main parameters of the analysis are N_{tp} and N_{add}, i.e., the number of tuples in the original table before the schema change and the number of attribute additions.

We now present the results of the analysis. Figure 6 compares the additional space required for the three approaches using fixed parameters N_a=8, S_a=8, N_e=50, N_{tp}=1000 as the number of attribute addition (N_{add}) increased from 1 to 8.

Figure 7 compares the space cost using fixed parameters N_a=8, S_a=8, N_e=20, N_{add}=8 as the number of tuples in the original tables (N_{tp}) ranges from 400 to 1400.

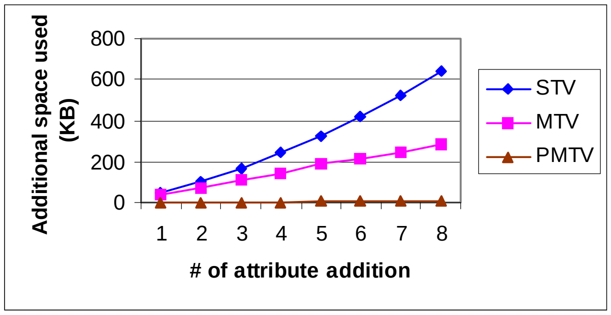

Figure 6. The Space cost for database conversion with fixed N_{tp} where N_{add} increased from 1 to 8.

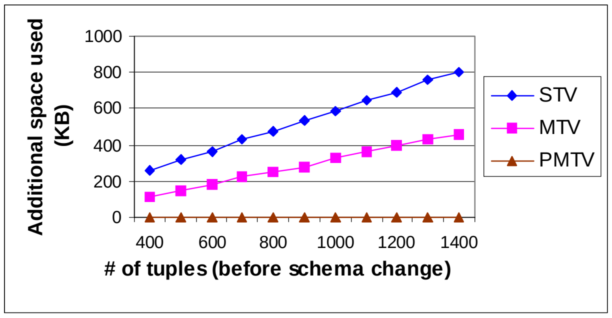

Figure 7. The Space cost for database conversion with fixed N_{add} where N_{tp} ranges from 400 to 1400.

These two figures show that the approach of PMTV saves a lot of space compared to the other approaches for database conversion.

5. Temporal query processing

Different types of temporal queries have been defined in [27]. In this section, we consider the most general bi-temporal query to compare these three approaches. This kind of query has the format "*range*//*range*/*point*" as defined in [27], which means "Retrieve a list of objects with keys in range *K* which are valid during valid-time range I_v as of time *T*".

Consider the following query, "List employee John's history during time 25 to 65 as of time 60". For this query, if there is no temporal index, a complete scan of the whole Employee table is required for the STV approach but no join operation is needed. MTV approach provides sort of a temporal index; consequently the number of tuples to be scanned is limited. However, MTV has the multischema problem, therefore joins may be required between different versions of the queried table [29]. For the PMTV approach, if no index exists, not only is the complete table scan required but also join operations between the original table and the tables created for the added attributes must be implemented, making the query very expensive. However, as we will show, the PMTV approach performs no worse than the other approaches if a bi-temporal index is used.

For STV and MTV approaches, while processing bi-temporal queries, with the form of "*//*/*point*", we can expect STV performs better than MTV because the MTV approach needs not only the complete scan of all the table versions but also the join operations. Therefore, we only compare PMTV with the STV approach.

Many join techniques have been well developed for snapshot relational databases [12], and some were extended to implement temporal joins [9,19,24,25]. Entity-Join [19] (referred as Event-Join in [20]) joins together a temporal object that is vertically partitioned into several relations via time normalization [13]. This is the case in our application. In [24,25], partition join is used to process the Entity-join operation and both give good results. However, the proposed partition techniques only applied to valid-time databases assuming that an object cannot have more than one value of a temporal attribute at any time point, which is not true in bi-temporal databases. Other Entity-Join techniques have also been proposed [21,26,28]. However, it is difficult to apply these techniques in bi-temporal databases.

Instead of adopting additional join mechanism or building the extra join index table and its index [21,26], we present the *Bi-temporal Entity Join* method by applying a single bi-temporal indexing structure for both purposes of indexing and Entity Join.

There are many temporal index techniques proposed, e.g., Time Index [6] and its derivatives, TP index, AP-tree, R-tree and its variants. A complete survey can be found in [17]. Most of the work focuses on the indexing of one single time dimension. One approach of bi-temporal index is to view bi-temporal data as a special case of spatial data and to adopt spatial indices to bi-temporal data. Recently, several bi-temporal indexing techniques have been proposed [10,11] based on the spatial indices R-tree [8] or R*-tree [2].

Here we use R-tree as our bi-temporal index. Because of the space limit, we only introduce how to incorporate indexing and Entity-Join in one index structure. The detailed implementation of Entity-Join and the efficiency of different index structure in bi-temporal databases is out side the scope of this paper and will be included in our another work.

The R-tree index structure is a direct extension of B^+-trees in *n* dimensions. The data structure is a height-balanced tree which consists of intermediate and leaf nodes. A leaf node contains index record entries of the form

$$(I, \textit{tuple-ID})$$

where *tuple-ID* refers a tuple in the database and *I* is an n-dimensional rectangle which is the bounding box of the object indexed

$$I = (I_1, I_2, \ldots, I_n)$$

where *n* is the number of dimensions and I_i is a closed bounded interval describing the extent of the object along dimension *i*. For bi-temporal databases, the dimension is 2.

The intermediate nodes contain entries of the form

$$(I , \textit{child-ptr})$$

where *child-ptr* is the address of a lower level node of the tree and *I* covers all rectangles in the lower node's entries.

For the STV approach, a R-tree, $\boldsymbol{RT_{STV}}$ is built upon the data set r_{STV} of schema R_{sc}. The total number of entries in the nodes of the leaf level, N_{STV}, is the number of tuples in r_{STV}, where

$$N_{STV} = N_{tp} + (N_F + 2*N_P) * N_e$$ [1]

For the PMTV approach, because the information of all the added attributes are part of the objects in relation *R*, therefore, the instances in the added attribute relation R_A_x, together with the instances in relation *R*, can be indexed by the same R-tree, $\boldsymbol{RT_{PMTV}}$. The total number of entries in all the nodes of the leaf level is N_{PMTV},

$$N_{PMTV} = N_{tp} + N_e$$

We compare the space overhead of $\boldsymbol{RT_{STV}}$ and $\boldsymbol{RT_{PMTV}}$ in terms of the total number of tuples to be indexed , i.e., the total number of entries in the leaf nodes of both trees , $n(RT_{STV})$ and $n(RT_{PMTV})$, after the schema change. We assume (continuing from section 4):

V_e:	the average number of versions for each entity *e*.
N_{cur}:	the average number of current versions for each entity.
α :	the percentage of V_e which is the number of current versions for each entity.
β :	the percentage of tuples in the N_{cur} whose valid-time intervals overlap with I_{vsc}.
γ:	the percentage of tuples in N_o whose valid-time intervals partially overlap with I_{vsc}.

[1] This equation and the next one were introduced in [29].

$N_{tp} = N_e * V_e$
$N_{cur} = \alpha\, V_e$
$N_o = \beta\, N_{cur}$
$N_p = \gamma N_o$

Then we have
$N_{STV} = N_eV_e + \alpha\beta\,(\gamma+1)\, V_e\, N_e$
$N_{PMTV} = N_eV_e+ N_e$

The following two figures compare $n(RT_{STV})$ and $n(RT_{PMTV})$ assuming V_e=10, α = 0.8, γ = 0.5, and the number of distinct entities N_e ranges from 100 to 1000. In Figure 8, with β = 0.5, total number of tuples to be indexed in RT_{STV} is 1.45 time more than in RT_{PMTV}. If we raise the percentage of tuples need to be converted after schema change, i.e., β, from 0.5 to 1, the ratio $n(RT_{STV})$ / $n(RT_{PMTV})$ is increased to 2.0.

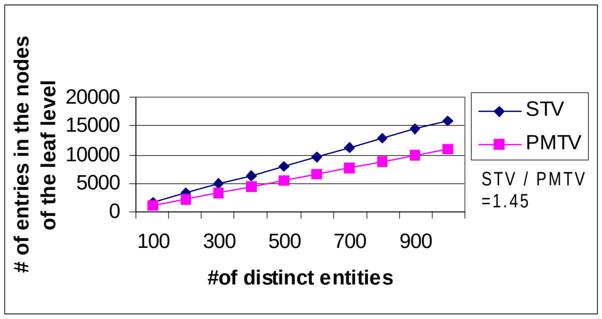

Figure 8. The number of entries in the nodes of the leaf level of the index R-tree with V_e=10 and lower weight of data conversion, β=0.5.

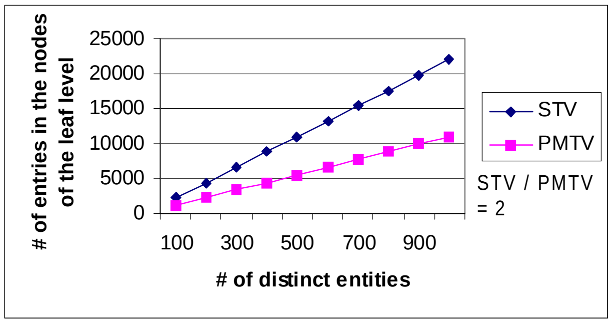

Figure 9. The number of entries in the nodes of the leaf level of the index R-tree with higher weight of data conversion, β =1.

For the entity-join in the PMTV approach, consider the query showed earlier, "List employee John's history during time 25 to 65 as of time 60". After retrieving all the valid tuples, these tuples can be entity-joined in memory without much time overhead.

From the discussion in the previous sections, we can see that the PMTV approach performs better than STV and MTV approaches in both database conversion and query processing (when a bitemporal indexing structure is used) thus make PMTV a better approach for bi-temporal schema versioning. The same idea can be also applied to other bi-temporal indexing techniques without difficulty.

6 Conclusion

In this paper, we first use an example to review two schema versioning approaches for temporal databases: Single Table Version (STV) and Multiple Table Version (MTV), and the proposed Partial Table Version (PMTV) approach. In most of the current literature, only transaction time is considered on schema versioning. The research that discusses schema versioning involving both transaction time and valid time does not consider some of the more complex problems concerning schema version creation and database conversion. We discuss the problems associated with the three approaches concerning space-time overhead, bitemporal querying, and database conversion.

For our proposed partial multiple table version approach, when a new attribute is added, it creates a new bi-temporal relation with only the new attribute, plus the key attribute of the relation being modified. This way, no null values will be introduced, no searching for the overlapped current versions is needed, no database restructuring and data duplication is required, and no extra effort is needed for the problem of mandatory version creation. In addition, when compared with the two previous approaches, the time and space cost for database conversion has been largely reduced.

Although the PMTV approach performs much better than the STV and MTV approaches in the process of databases conversion, it requires expensive Entity-Join to retrieve the complete object information. To solve this problem, we propose a method that incorporates indexing and entity-join into one index structure. We use R-tree bi-temporal index for our presentation, the same methodology can be adopted to other bi-temporal indexing scheme.

References

[1] J. Banerjee, H-T Chou, H. J. Kim , and H.F. Korth. Semantics and Implementation of Schema Evolution in Object-oriented databases. *SIGMOD RECORD*, 16(3): 311 – 322, 1987.

[2] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and Robust Access Method for Points and Rectangles. In *Proceedings of ACM SIGMOD*, pages 322 – 331, 1990.

[3] J. Clifford, C. Dyreson, T. Isakowitz, C. S. Jensen, and R.T. Snodgrass. On the Semantics of “now” in Databases. *ACM Transactions on Database Systems*, 22(2):171 – 214, June 1997.

[4] J. Clifford and D.S. Warren. Formal Semantics for Time in Databases. *ACM Transactions on Database Systems*, pages 214-254, 1983.

[5] Christina DeCastro, Fabio Grandi, and Maria Rita Scalas. Schema Versioning for Multitemporal Relational Databases. *Information Systems*, pages 249-290, July 1997.

[6] R. Elmasri, G. Wuu, and Y. Kim. The Time Index: An Access Structure for temporal data. In *Proceedings of the 16th VLDB Conference*, 1990.

[7] H. Gunadhi and A. Segev. Query Processing Algorithms for Temporal Intersection Joins. In *Proceedings of 7th International Conference on Data Engineering, IEEE*, 1991.

[8] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD*, pages 47 – 57, 1984.

[9] C. Jensen *et al.* A consensus glossary of temporal database concepts. *SIGMOD RECORD*, 23(1):52-64, 1994.

[10] C. Kolovson, M. Stonebraker. Segment Indexes: Dynamic Indexing Techniques for Multi-dimensional Interval Data. In *Proceedings ACM SIGMOD*, pages 138 – 147, 1991.

[11] A. Kumar, V. J. Tsotras, and C. Faloutsos. Access Methods for Bi-temporal Databases. In *Recent Advances in Temporal Databases*, J. Clifford, and A. Tuzhilin (eds), pages 235 – 254, 1995.

[12] Priti Mishra and M. H. Eich. Join Processing in Relational Databases. *ACM Computing Surveys*, Vol. 24, No. 1, March 1992.

[13] S. B. Navathe and R. Ahmed. A Temporal Relation Model and a Query Langue. *Information Sciences*, pages 147-175, 1989.

[14] J. F. Roddick. Dynamically Changing Schemas within Database Models. *Australian Computer Journal*, pages 105-109, 1991.

[15] J. F. Roddick. Schema Evolution in Database Systems – An Annotated Bibliography. Technical Report No. CIS-92-004, School of Computer and Information Science, University of South Australia, 1992.

[16] J. F. Roddick. A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7), 1995.

[17] B. Salzberg and V. J. Tsotras. A Comparison of Access Methods For Time-Evolving Data. In Computing Surveys, 31(2):158-121, 1999.

[18] M. R. Scalas, A. Cappelli, and C. De Castro. A Model for Schema evolution in Temporal Relational Databases. In *Proceedings of 1993 CompEuro, Computers in Design, Manufacturing, and Production*, pages 223 – 231, May 1993.

[19] A. Segev. Join Processing and Optimization in Temporal Relational Databases. Chapter 15 of *Temporal Databases: Theory, Design, and Implementation*, Benjamin/Cummings, 1993.

[20] A. Segev and H. Gunadhi. Event-join optimization in temporal relational databases. In *Proceedings of the Conference on Very Large Data Base*, pages 205-215, August, 1989.

[21] A. Shrufi and T. Topaloglou. Query Processing for Knowledge Bases Using Join Indices. In *Proceedings of the 4th International Cinference on Information and Knowledge management (CIKM)*, 1995.

[22] R. T. Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, pages 247 – 298, June 1987.

[23] R. T. Snodgrass, editor. The TSQL2 Temporal Query Language, chapter 10. Kluwer Academic Publishers, 1995.

[24] M. D. Soo, R. T. Snodgrass and C. S. Jensen. Efficient Evaluation of the Valid-Time Natural Join. In *Proceedings of the 10th International Conference on Data Engineering*, IEEE, 1994.

[25] D. Son and R. Elamsri. Efficient Temporal Join Processing Using Time Index. In *Proceedings of the 8th International Conference on Scientific and Statistical Database Management*, pages 252 – 261, June 18 – 20, 1996.

[26] T. Topaloglou. Storage Management for Knowledge Bases. In *Proceedings of the 2nd International Conference on Information and Knowledge Management (CIKM'93)*, 1993.

[27] V. J. Tsotras, C.S. Jensen, and R. T. Snodgrass. A Notation for Spatiotemporal Queries. *In TimeCenter Technical Report TR-10*, April 1997.

[28] P. Valduriez. Join Indices. ACM Transactions on Database Systems, 12(2): 218 – 246, June 1987.

[29] H.C. Wei and R. Elmasri. Study and Comparison of Schema Versioning and Database Conversion Techniques for Bi-Temporal databases. *In Proceedings of the 6th International Workshop on Temporal Representation and Reasoning (TIME-99)*, pages 88-98, May 1 – 2, 1999.