GRIIS

Groupe de recherche interdisciplinaire en informatique de la santé

Faculté des sciences et Faculté de médecine et des sciences de la santé
Université de Sherbrooke, Québec

RR0020A
Automated bitemporal database schema design with a unified framework

Christina Khnaisser[1,2], Luc Lavoie[1], Anita Burgun[2], and Jean-Francois Ethier[1,2,3] (✉)

[1] Département d'informatique, Université de Sherbrooke, Sherbrooke, Canada
{christina.khnaisser, luc.lavoie}@usherbrooke.ca

[2] INSERM UMR 1138 team 22 Centre de Recherche des Cordeliers, Université Paris Descartes
anita.burgun@aphp.fr

[3] Département de médecine, Université de Sherbrooke, Sherbrooke, Canada
jf.ethier@usherbrooke.ca

Abstract: Information evolution over time is critical for many analysis purposes and essential to many applications. Temporalization adds some time related (temporal) attributes to relations but it is not sufficient to track all changes. Historicization is the process of transforming a non-historical database schema into a historical schema allowing data evolution traceability. There are complex design, query and modification issues that need to be addressed. Existing solutions in temporal databases are based on different data models with different structures and semantics, making comparison and selection difficult. Many of them are unable to use directly existing DBMS to store, manage or query temporal data. Furthermore, methods published until now propose transformation rules by providing various examples. As a result, real-world applications require manual adaptations and implementations. This article demonstrates the integration and generalization of two major temporal models (the Bitemporal Conceptual Data Model and the Date-Darwen-Lorentzos Model) in terms of views defined using the Unified Bitemporal Historicization Framework. Using this framework, historicization is defined as a suite of simple automated steps. The primary aim of this work is to help database designers to model historicized schema based on a sound theory ensuring a sound temporal semantic, data integrity, query expressiveness and guided automation.

Keywords: Data warehouse design, Temporal data warehouse, Temporal indeterminacy.

Revision history:

version	date	author	description
1.0.0a	2017-mm-jj	XX	...
0.2.0a	2016-03-15	CK	Revue et ajouts en lien avec la rédaction de l'article pour VLDB 2017.
0.1.0a	2017-03-01	LL	Première esquisse à partir du modèle GRIIS_gabarit, version 022a.

Table of contents

1 Introduction

While temporal aspects in databases have been explored for more than thirty years, it remains a major issue which does not [16] yet have a consensual solution in part because proposed approaches are various in nature and focus on a specific domain which creates significant challenges to extend or reuse them. Consequently, interoperability remains a key issue when temporal data need to be collected from different sources (which may have used different temporal models), a typical problem in data warehouses [15]. The goal of historicization is to produce a schema with a unified temporal semantic and temporal constraint management capable of keeping track of data evolution in a general manner independently from the application domain. The database schema must be based on a sound, comprehensive and formalized temporal model to improve expressiveness and interoperability. Another major challenge appears when designing a historical schema: to guarantee data consistency respective to intrinsic temporal rules and specific business rules. As a result, designing a historical schema is a complex and error-prone process if done manually. In fact, it is essential to have a general model independent of the field of application to reflect the intrinsic nature of the historical data.

Two major temporal models have emerged in the literature and in our practice (clinical data warehousing) [15]. The first one is the Bitemporal conceptual data model (BCDM) a bitemporal model based on SQL. Snodgrass presents design “best practices” to build a bitemporal schema starting from a conceptual model (entity relationship) ending with SQL code ()[1]. BCDM was initially defined in [13], a more recent presentation can be found in [21] and an extension to temporal indeterminacy in [3].

[1] TSQL2 [22], a temporal SQL extension, is also a possible target.

The second one is the Date-Darwen-Lorentzos Model (DDLM) which is based on the relational theory. It proposes three sub-models, two unitemporal model and one bitemporal model based on the third manifesto relational model [8] and Allen's interval logic [1]. DDLM also includes a simple design technique and an extension for TutorialD (e.g. PACK, UNPACK, and USING relational operators) to facilitate temporal data management and temporal queries. DDLM was initially defined in [17] and a more recent definition can be found in [10].

Both models need adaptation and contextualization to manage a specific schema historicization. They are difficult to compare one to another and seems hardly interoperable when sources, built according to them, must be included in the same data warehouse. Also, neither of those two models offers a strong solution to cope with missing information, which is often required. Consequently, a unifying model able to represent correctly, at least, BCDM and DDLM with the provision to cope with missing data would be a very valuable tool.

This article presents a new approach to automate the historicization of a database schema based on fundamental temporal relational concepts, a uniform structure and domain-independent constraints. First a unified bitemporal historicization framework (UBHF) is described and an automatic historicization is defined. BCDM and DDLM are then represented as relational views defined on a UBHF-compliant model (UBHM) and compared.

The article is organized as follows. Section 2 presents definitions of UBHF base concepts. Section 3 defines the historicization built on UBHF concepts, including schema structure, constraints and temporal modification operations. Section 4 presents a DDLM and BCDM as views using UBHM. Section 5 discusses the similarities and differences between BCDM and DDLM. Finally, section 6 concludes with the contribution summary and future work proposals.

2 UBHF concepts

UBHF is a conceptual framework that defines temporal stereotypes based on fundamental relational and temporal concepts [10, 21]. More precisely, the proposed framework is based on the relational theory and normalization (especially fifth and sixth normal form — 5NF and 6NF) as described in [6, 8] and interval temporal logic as described in [1, 2]. Using a close variant of the TutorialD language, this section presents the basic concepts (most of them being defined in [10], as well as the TutorialD language itself) and stereotypes used in UBHF.

2.1 Time model

UBHF uses a discrete time model based on points and intervals derived from the one defined in [1] and used in [10] where more comprehensive definitions may be found.

A point type is any discrete, ordinal, bounded type. The minimum value is denoted alpha (α) and the maximum value omega (ω). Given a point type $P = \{p_0, \ldots, p_{n-1}\}$, then $\alpha = p_0 < \ldots < p_{n-1} = \omega$ and the following operators are defined for $p \in P$: $FIRST(p) = \alpha$; $LAST(p) = \omega$; if $p_i > \alpha$, $PRIOR(p_i) = p_{i-1}$; if $p_i < \omega$, $NEXT(p_i) = p_{i+1}$.

An interval type over a point type P, denoted INTERVAL[P], is the set of all non-empty sets of contiguous points bounded on both ends. An interval where the begin point is equal to the end point is a singleton (aka unit interval), it is a non-decomposable interval (the empty set is not an interval). Given $p_b, p_e \in P$, $p_b \leq p_e$, $i = [p_b:p_e] \in INTERVAL[P]$, then $i = \{p_i \in P \mid p_b \leq p_i \leq p_b\}$ and the following operators are defined: $BEGIN(i) = p_b$; $END(i) = p_e$; if $p_b > \alpha$, $PRE(i) = PRIOR(p_b)$; if $p_e < \omega$, $POST(i) = NEXT(p_b)$; CARD(i) is the cardinality of the set i.

Note also that different period notation (closed-closed, open-closed, closed-open, open-open) can be found in the literature. Without loss of generality, UBHF uses the closed-closed notation since the other notations :

$$[b:e] = (b-1:e] = [b:e+1) = (b-1:e+1)$$

are equivalent except when the bounds (α and ω) are involved. In these cases, only the closed-closed notation covers all the values in INTERVAL[P] in a well-defined manner.

2.1.1 Other operators

For comparison purposes, 13 Allen's interval operators [1] are used to compute relationships between two intervals. Furthermore, generalized definitions of the set operators and of the standard comparison operators can be applied to intervals to manage data inconsistency. Extension to the relational operators are also defined to address relations with interval type attributes. Here, we especially consider PACK, UNPACK and USING. Informally, the PACK operator collapses value-equivalent[2] tuples (of a relation) having periods that merge (overlap or meet), the UNPACK operator expands a tuple (of a relation) into multiple tuples where each has a singleton interval value and USING is a shorthand of a combination of UNPACK and PACK.

2.1.2 Time representation

A "time point type", say TIMEPOINT, is a point type associated with two more properties: the granularity denoted gamma (γ) and the origin denoted phi (φ). The granularity represents the constant duration between two consecutive points. The origin represents the time point value associated to α in a convenient agreed time reference (such as a calendar). Let DATETIME be a totally ordered set of Calendar time reference values, let date(p) be a function from TIMEPOINT to DATETIME such that:

- ◊ $date(\alpha) = \varphi$
- ◊ $p_i < p_j \Rightarrow date(p_i) \leq date(p_j)$
- ◊ $d_k < d_\ell \Rightarrow date^{-1}(d_k) \leq date^{-1}(d_\ell)$

An interval defined over a time point type is a "time interval", e.g.: INTERVAL[TIMEPOINT]. A time unit interval is called a moment [2]. Figure 1 below illustrates informally the concepts of the time model. Let i be a time interval:

- ◊ $duration(i) = CARD(i) * \gamma$

The figure below illustrates these definitions.

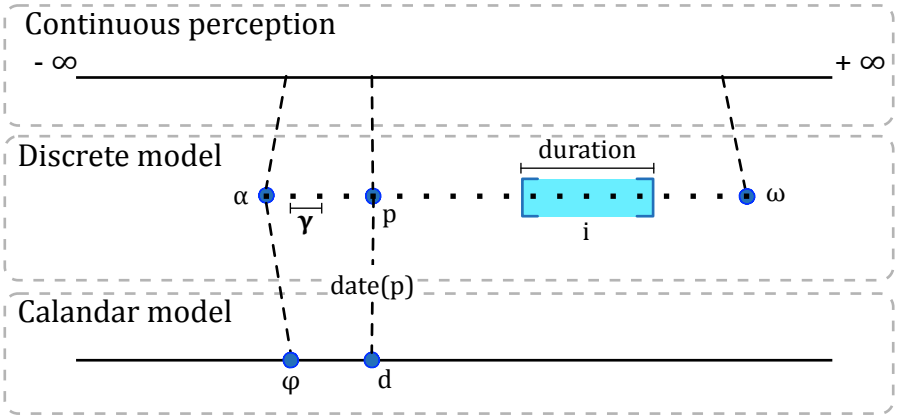

Figure 1. UBHF time model.

Example: TIMEPOINT ⊂ ℕ with α = 1 ≤ p ≤ ω = 99; γ = 84 600 s and φ = 1970-01-01T00:00:00Z. Given p = 3, FIRST(p) = 1, LAST(p) = 99, PRIOR(p) = 2 and NEXT(p) = 4; with PRIOR(1) and NEXT(99) being undefined.

Example: PERIOD = INTERVAL[TIMEPOINT]. Given an interval value i = [3:6] = {3, 4, 5, 6}, BEGIN(i) = 3, END(i) = 6, PRE(i) = 2, POST(i) = 7 and CARD(i) = 4.

TIMPEPOINT and PERIOD will be used as a representative time point type and time interval type in the following.

2.2 Timelines categorization

There are several timelines (aka time dimensions and time axis) defined in the database literature: valid time, transaction time, decision time, event time, etc. [7]. One of our goal is to integrate BCDM et DDLM in a single framework, UBHF. Consequently, valid and transaction times are needed (although various labels are used for

[2] Two tuples are value-equivalent when all their non-key attributes have the same value.

valid (e.g.: validation, user, stated) and transaction (e.g.: system, logged). The other timelines are considered as domain specific timelines and do not have a special treatment in the framework. As in the consensus glossary of temporal database [12], the retained timelines are defined as follows:

- ◊ Transaction time (@T): "The transaction time of a fact is the time when the fact is current in the database and may be retrieved." It cannot be modified by the user. It is used to provide tuple correction traceability.
- ◊ Valid time (@V): "The valid time of a fact is the time when the fact is true in the modelled reality." It can be modified by the user or the system. It is used to provide fact traceability in the real world.

2.3 Timeline attributes and values

In a relation, a timeline is represented by an attribute (called timeline attribute). A timeline attribute can have different types and values defined as follow:

- ◊ *be*: A period timeline attribute where the beginning and the end point values are known. The associated "proposition is true from b to e". In other words, for each point included in the period the proposition is true.
- ◊ *bx*: A point timeline attribute with unknown end value where the beginning point is known and the end is unknown. The associated "proposition is true from b until (*now*, *ufn*, or ω)".

Depending on the timeline represented, the table below defines the notation used.

Table 1. Timeline attributes notation.

Notation	Definition	Timeline
@Vbe	Valid time period	Valid
@Vbx	Valid time point	
@Tbe	Transaction time period	Transaction
@Tbx	Transaction time point	

2.4 Attributes categorization

In a non-historical schema, we conventionally distinguish between key and non-key attributes. A non-historicized relation R is denoted R(K, A) where:

- ◊ $K = \{k_1,...,k_{|K|}\}$ is the set of key attributes ($|K| \geq 1$). Without loss of generality, we consider that each relation contains only one key - although this key may have more than one attribute.
- ◊ $A = \{a_1,...,a_{|A|}\}$ is the set of non-key attributes ($|A| \geq 0$).

A historicized relation R is denoted R'(K, B, C, D_V, D_T) with $A = B \cup C$ where:

- ◊ $B = \{b_1,...,b_{|B|}\}$ is the set of non-key attributes ($|B| \geq 0$) associated with a valid timeline attribute (called historicized attributes); B is a subset of A.
- ◊ $C = \{c_1,...,c_{|C|}\}$ is the set of non-key attributes ($|C| \geq 0$) **not** associated with a valid timeline attribute(called non-historicized attributes); C is a subset of A.
- ◊ $D_V = \{@V, b_1@V,..., b_{|B|}@V\}$ is the set of valid timeline attributes, with the following notations @V is associated with K, and $b_i@V$ is associated to $b_i \in B$.
- ◊ $D_T = \{@T, b_1@T,..., b_{|A|}@T\}$ is the set of transaction timeline attribute where @T is associated with K, and $b_i@T$ is associated with $b_i \in B$.

Remark. Regarding R', K and "key" do not refer to the same set of attributes. K is the key set of R and "key" refers to the key set of R' (that contains K and their associated timeline attributes, if applicable).

2.5 Relation categorization

Like attributes, relations in a temporal schema are categorized depending on the timelines contained in D_V and D_T. Four categories are distinguished.

- ◊ **Non-historicized relation (!N):** a relation R is non-historicized, denoted R!N, if it contains only non-historicized attributes. Formally, R is defined as R@N(K, {}, C, {}, {}) with $|B| = 0$, $|C| \geq 0$, $|D_V| = 0$ and $|D_T| = 0$.

- ◊ **Valid-time relation (!V):** a relation R is a valid-time relation (and denoted R!V), if it contains at least one valid timeline attribute and no transaction timeline attribute. Formally, R is defined as R@V(K, B, C, D_V, {}) with |B| ≥ 0, |C| ≥ 0, $|D_v| = |B|+1$ and $|D_T| = 0$. One valid timeline attribute denoted @V is associated to the set K and each b_i in B is associated to valid timeline attribute denoted b_i@V.
- ◊ **Transaction-time relation (!T):** a relation R is a transaction-time relation, denoted R!T, if it contains at least one transaction timeline attribute and no valid timeline attribute. Formally, R is defined as R@T(K, {}, C, {}, D_T) with |B| = 0, |C| ≥ 0, $|D_V| = 0$ and $|D_T| \geq 1$. One transaction timeline attribute denoted @T is associated to the set K, each b_i in B is associated to the transaction timeline attribute denoted b_i@T, and each c_i in C is associated to the transaction timeline attribute denoted c_i@T.
- ◊ **Bitemporal relation (!VT):** a relation R is a bitemporal relation, denoted R!VT, if it contains valid timeline and transaction timeline attributes. Formally, R is defined as R@VT(K, B, C, D_V, D_T) with |B| ≥ 0 |C| ≥ 0, $|D_V| = |B| + 1$ and $|D_T| \geq 1$.

Example. Consider the "non-historicized" predicate: *The patient identified by "pNo" is born on "birth" and has a health status "hS".* With PN, DATE and HS being the corresponding attribute types, the standard notation of the relation (variable) PatientHS is:

```
PatientHS(pNo:PN, birth:DATE, hS:HS)
```

The non-historicized relation is denoted as follows (types are omitted for simplicity):

```
PatientHS({pNo}, {hS, birth})
```

and the same historicized relation is denoted as follows:

```
PatientHS!N({pNo}, {}, {hS, birth}, {}, {})
```

A relation value is assigned to the variable PatientHS as follows (for simplicity health status is considered as an code represented as an integer):

```
PatientHS := RELATION
{
TUPLE {pNo PN(N01), birth "1980-08-08", hS HS(20)}
TUPLE {pNo PN(N02), birth "1960-06-06", hS HS(10)}
};
```

Assume now that we want to keep track of the validity evolution of the patient identifier and this health status only[3]. Valid timeline attribute @V and hS@V are added and associated respectively with pNo and hS. The predicate becomes: *The patient identified by "pNo"* ***during "@V"*** *is born on "birth" and has a health status "hS"* ***during "hS@V".*** The relation denotation becomes:

```
PatientHS!V({pNo}, {hS}, {birth}, {@V, hS@V}, {})
```

We may then be interested to keep track of the modifications as they were registered by the database server. We do so by adding a transaction timeline attribute associated with the whole tuple. The predicate becomes: ***The fact that*** *the patient identified by "pNo" is born on "birth" and has a health status "hS"* ***is represented in the database during "@T".*** The relation denotation becomes:

```
PatientHS!T({pNo}, {}, {hS, birth}, {}, {@T})
```

Now, we may be interested in the combination of the last two definitions. The predicate becomes: ***The fact that*** *the patient identified by "pNo"* ***during "@V"*** *is born on "birth" and has a health status "hS"* ***during "hS@V" is representd in the database during "@T".*** The relation denotation becomes:

```
PatientHS!VT({pNo},{hS},{birth},{@V, hS@V},{@T})
```

[3] Note that the "birth" attribute could have been historicized, but we decided not to keep track of its evolution so a C set member can be shown.

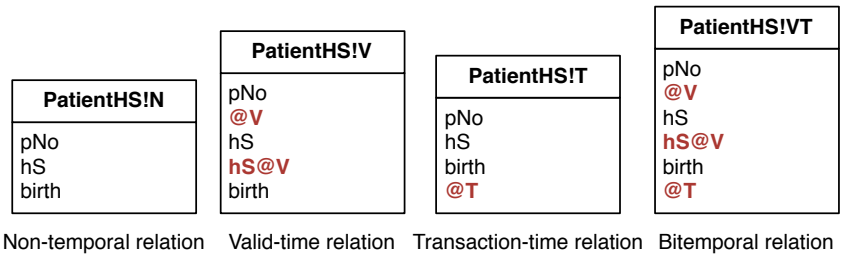

Figure 2. Illustration of the temporal relations

2.6 Data inconsistency

Data inconsistency is mainly related to the following problems: missing information, volatility and integrity (which includes redundancy, contradiction, circumlocution and non-denseness) [10]. Data inconsistency makes temporal updates, and querying tedious and error-prone. More precisely, data inconsistency forces the application developer to build *ad hoc* functions that are difficult to maintain and can easily be misinterpreted by others causing un-intended effects and unsatisfying (even inaccurate) results.

2.6.1 Temporal volatility

Dealing with temporal volatility (also called "moving point", "now", or "now-relative data" problem) is very important in a temporal model. Different solutions have been proposed in the literature: DDLM avoid maintaining a *now* special value using design techniques [10]; BCDM uses special values *forever* and *until changed* for valid timeline and transaction timeline respectively [21]; others use NULL, FIRST(), LAST(), etc. An interesting survey of different solutions can be found in [4]. In UBHF, the DDLM foundation is used to avoid data inconsistencies and ambiguities that can be so easily be introduced.

2.6.2 Temporal integrity

Keeping history changes may introduce redundancy, contradiction, circumlocution and non-denseness when attributes in the same relation are modified independently. Here's an informal definition of these problems:

- ◊ Redundancy occurs when two value-equivalent tuples overlap in time.
- ◊ Contradiction occurs when two tuples having identical key values but different non-key attributes value overlap in time.
- ◊ Circumlocution occurs when two value-equivalent tuples meets in time (one follows immediately the other).
- ◊ Denseness guarantees that when an attribute value is known for a key value at some time point, the value of all other attributes dependent of the same key is known as well at the same time point.

DDLM studied these problems in detail [10] (chap. 5 and 13) and proposed constraints to avoid them. See section 3.3 for the constraint definitions.

2.6.3 Missing information

Missing information is unavoidable in many context. Many approaches has been proposed in the literature using 3-valued logic (SQL NULL approach), 4-valued logic [5], fuzzy logic [19], relational design with vertical and horizontal decomposition [9, 11], default values, etc. The presentation and the defence of the various positions to address the missing information problem is out of the scope of this paper, but a neutral basis will be suggested in section 3.5.

3 UBHF historicization

Given clear requirements, handling data inconsistency in a historical schema can be achieved by defining a suitable historical structure and appropriate constraints.

3.1 Schema requirements

The initial (non-historicized) schema must respect some requirements to allow the automatic historicization and to ensure data consistency of the final (historicized) schema. The initial schema must be a relational schema with respect to the relational theory and must satisfy the requirements below:

- ◊ The schema is in5NF, ensuring that unnecessary duplicates are already eliminated.
- ◊ Each relation contains only one (candidate) key. However, this key may have more than one attribute.
- ◊ If valid timeline information is not available for some attributes, they must be identified (in the C set).

The final historical schema (the output of the historicization) must satisfy the requirements below:

- ◊ The schema is in 6NF, ensuring historical independence of each attribute and query performance [20].
- ◊ The set of constraints is complete regarding data consistency (as defined in 2.6).
- ◊ All relations are historicized either by valid and transaction timeline (for the K and the B sets) or by transaction timeline only (for the C set).

The process itself must do this transformation while keeping the traceability of the required steps and the initial schema conceptual view**.**

3.2 Schema structure

The historicization of a schema structure is obtained by iterative normalization of each of its relation according to its category. The objective is to address first the temporal volatility at the structure level by normalizing relations into 6NF ensuring that the relation represents one and only one predicate. Then, the remaining temporal volatility and temporal integrity are addressed by adding proper constraints.

3.2.1 Relational decomposition

The normalization process uses lossless relational decomposition. Relational decomposition is the process of decomposing a relation into smaller relations ("relparts" for short), using relational operators without losing data. Two types of decomposition are used:

- ◊ **Projection-join decomposition (PJ):** it consists of separating attributes into two or more relparts using projection operations ensuring that the recomposition using the join operation (on the key) leads to a lossless decomposition. PJ decomposition will be used to obtain a 6FN representation of each initial relation.
- ◊ **Restriction-union decomposition (RU):** it consists of separating tuples into two relparts regarding a restriction condition ensuring that the recomposition using their union leads to a lossless decomposition.

The historicization is performed on each relation in the initial schema. Given a relation R(K, A) to be historicized into a bitemporal relation R!VT(K, B, C, D_V, D_T) the following steps are required:

1. Categorize A into either B or C (see section 2.4), add @V and b_i@V to D_V.
2. Decompose the relation into the 6NF using PJ decomposition, so that K, ai and bi are kept with their respective timeline attributes as: PJ({K,@V}, {K, b_1, b_1@V}, …, {K, b_n, b_n@V}, {K, c_1}, …, {K, c_n}).
3. In each relpart containing a b_i attribute, rename the b_i@V as @V[4].
4. Decompose each resulting historicized relpart (obtained in 2 and 3) using RU decomposition over the timeline attribute to separate timeline between be-type and bx-type.
5. Define two relparts with @Tbx and @Tbe for each relpart obtained in (4).

[4] This step is for convenience purpose, to facilitate constraint definition or to help query expressiveness (especially the JOIN is involved).

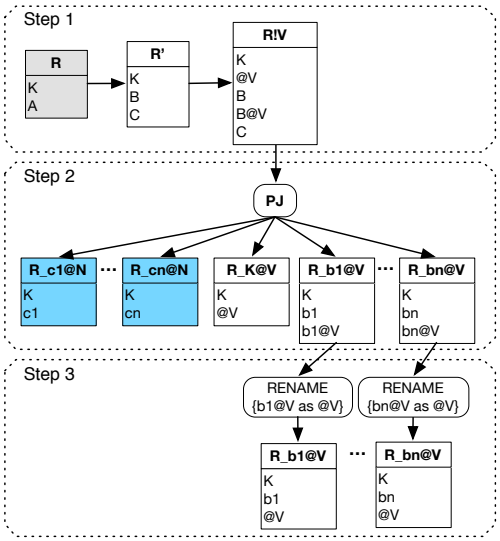

Figure 3. Historicization steps 1 to 3.

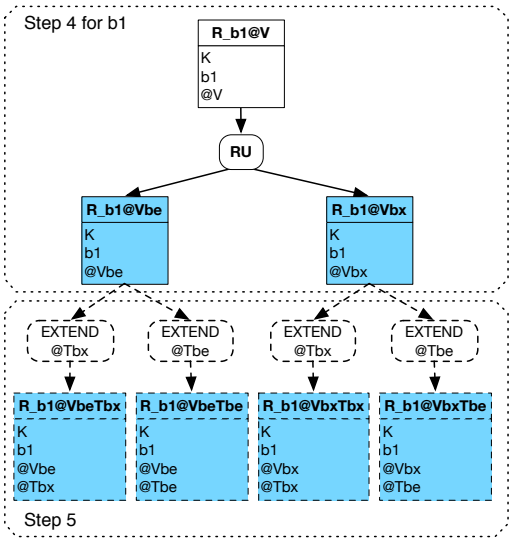

Figure 4. Historicization step 3 and 4 of a_1.

Figure 3 and 4 illustrate the historicization. Each relpart is represented as a rectangle. The gray rectangle (of Step 1) is a initial relation, a white rectangle represents an intermediate relpart, and the blue one is a final relpart. A fully lined rectangle border is a base relation, and a dotted rectangle border is a view or an automatically maintained relation (e.g. through a trigger). The arrows show the direction of the process. The rounded corner rectangle shows the operation involved.

Figure 4 shows the steps for a_1 attribute only. Step 4 must be applied to K (R_K@V) and A relparts (R_b_1@V, … R_b_n@V); Step 5, to K, B and C (R_c_1@N, …, R_c_n@N) relparts.

3.2.2 Relational grouping

To facilitate constraint definition, these (numerous) relparts are conceptually grouped into three types of groupings:

- The *K-grouping* of a relation R is the set of all the K relparts. A K-relpart, denoted R_K, is a relation with K and the associated timeline attributes only.
- A *b_i-grouping* of a relation R is the set of all b_i-relparts including b_i-*present*-relparts R_b_i_P and b_i-*missing*-relparts R_b_i_M. Collectively, the b_i-groupings are called *B-groupings*.
- A *c_i-grouping* of a relation R is the set of all c_i-relparts including c_i-*present*-relparts R_c_i_P and c_i-*missing*-relparts R_c_i_M. Collectively, the c_i-groupings are called *C-groupings*.

3.2.3 Relational partition

Relparts are conceptually split into "partitions" to facilitate constraint definition and query expressiveness. A partition is defined regarding the relation category and timeline category. The figure below shows the hierarchy of potential relational partitions that can be found in a historical schema depending on the decomposition. In UBFH, the structure and the constraints are defined over the leaf partitions (in blue).

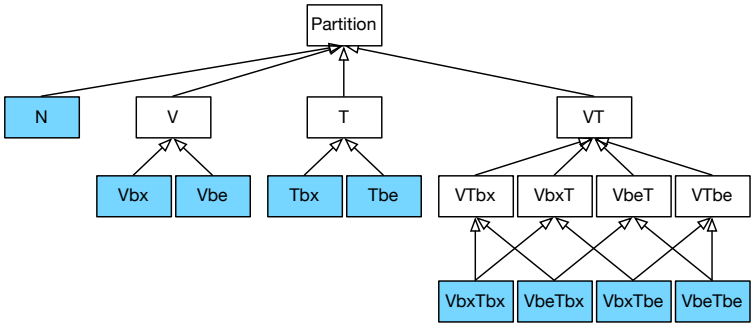

Figure 5. Hierarchy of all potential relational partitions.

The relpart in a partition category *S* of a *g* grouping is denoted R_*g*@*S*.

Example. The historicization of PatientHS into a valid-time relation generates de following relparts:

- **Partition N:** PatientHS_birth@N (*the patient identified by pNo is born on "birth" date*).
- **Partition Vbx:** PatientHS_K@Vbx (*the patient identified by pNo from @Vbx*), PatientHS_hS@Vbx (*the patient identified by pNo has a health status hS from @Vbx*).
- **Partition Vbe:** PatientHS_K@Vbe (*the patient identified by pNo during @Vbe*), PatientHS_hS@Vbe (*the patient identified by pNo has a health status hS during @Vbe*).

- **K-grouping:** PatientHS_K@Vbx and PatientHS_K@Vbe.
- **hS-grouping:** PatientHS_hS@Vbx and PatientHS_hS@Vbe.
- **birth-grouping:** PatientHS_birth@N.

3.3 Constraints

Keeping history may introduce data inconsistency as described in section 2.6. Even if a relation is in 5NF or 6NF, data inconsistency may occur within the relation and between related ones. In other words, a constraint must be defined for each relpart according to its partition to maintain data consistency. Constraint templates are presented and can be automatically adapted to generate constraints for a specified relation.

The constraints are only defined on V partitions because they can be modified by the user. There is no need to define constraints for T and VT partitions because their value cannot be modified by the user as they are managed by the DBMS using any of the following solutions: (a) as views when the values of the transaction timeline attribute are obtained by a function call to the DBMS journal as in DDLM or (b) as base relations (table) when the values of the transaction timeline attribute are set by the DBMS (as in SQL:2011 with SYSTEM TIME), or by triggers (as in BCDM [21]). In UBHF, all solutions are supported if they satisfy the transaction timeline semantic.

The following constraints must be defined regarding each relation of the initial schema. A constraint is defined with a unique identifier and a boolean expression. As a convention, the identifier starts with the relparts identifier followed by the constraint name and the boolean expression is written using a variant of TutorialD.

3.3.1 Candidate keys

For non-historicized relparts and historicized relparts with bx-type timeline, the key constraint is the same as in the initial relation. More specifically, for each grouping g, and each partition S in {N, Vbx, Tbx, VbxTbx} the key constraint of R_g@S is defined as:

```
RELATION R_g@S (K, B, C, DV, DT)
  KEY {K};
```

For historicized relparts with be-type timeline, the key constraint must be applied to each time point in the period. For each grouping g, and each partition S in {Vbe, VbeTbx} the key constraint of R_g@S is defined as:

```
RELATION R_g@S (K, B, C, DV, DT)
  USING(@Vbe) : KEY {K, @Vbe};
```

For each grouping g, and each partition S in {Tbe, VbxTbe} the key constraint of R_g@S is defined as:

```
RELATION R_g@S (K, B, C, DV, DT)
  USING(@Tbe) : KEY {K, @Tbe};
```

For each grouping g, and each partition S in {VbeTbe} the key constraint of R_g@S is defined as:

```
RELATION R_g@S (K, B, C, DV, DT)
  USING(@Vbe, @Tbe) : KEY {K, @Vbe, @Tbe};
```

3.3.2 Foreign keys

A foreign key is evaluated regarding related attributes in different relparts. In a historicized relation, the associated timeline attributes must be considered to guarantee that their related values are asserted at the same time (at each moment of the unpacked relation). On the one hand, the foreign key in the initial schema must be maintained. Let Rs{X}→Rd be a foreign key in the initial schema where Rs is the source relation with X being any subset of attributes of Rs equivalent to the key (K) of Rd, the destination relation. The constraint guarantees temporal referential consistency between relparts by verifying that the projection of Rs on X is included in the projection of Rd on K (with suitable renaming).

To increase expressiveness, a shorthand operator, **gSpace**, is defined, extracting the history (if any) of a specified grouping. The operation returns a relation equal to the union of all relparts of a specific grouping **g** in {R_K, R_b_1,...R_b_n, R_c_1,...,R_c_n} with respect to the applicable partition[5] (@N or @V). The operation is defined as:

```
OPERATOR gSpace(g GROUPING) RETURNS RELATION;
  IF g is a C-grouping THEN
    R_g@N
  ELSE // K-grouping or B-grouping
    WITH (
      r_bx :=
        (EXTEND R_g@Vbx : {@V:=[@Vbx:ω]})
          {ALL BUT @Vbx}),
      r_be :=
        R_g@Vbe RENAME {@Vbe AS @V}
    ): USING (@V): r_bx UNION r_be
  END IF
END OPERATOR
```

In the foreign key case, note two potential issues:

- ◊ Each $x_i \in X$ can be used in different relparts and may belong to K, B or C, so we need the union of the different relparts x_i belongs to by using gSpace(R_x_i).
- ◊ x_i may belong to C, so we need to verify that x_i exists **only** at every time point of gSpace(Rd_K). According to the denseness constraint (described in the next section) x_i exists at all time points of gSpace(Rs_K).

Using gSpace, another shorthand operator, **gUnpack**, is defined, extracting the unpacked history of an attribute **x** of a specified relation **R**:

[5] gSpace can be also defined to deal with @T and @VT partition. In the scope of this paper only @N and @V are illustrated.

```
OPERATOR gUnpack(R RelationName, x AttributeName)
RETURNS RELATION;
  IF (x in K of R) THEN
    UNPACK (@V) : (gSpace(R_K){x,@V})
  ELSIF (x in B of R) THEN
    UNPACK (@V) : (gSpace(R_x){x,@V})
  ELSE //(x in C of R)
    UNPACK (@V) : gSpace(R_x){x} JOIN gSpace(R_K)
  END IF
END OPERATOR
```

Each foreign key Rs{X}→Rd in the initial schema is defined as:

```
CONSTRAINT Rs_Rd_@V_fk
  gUnpack(Rs, x1) RENAME {x1 AS k1} ⊆
    gUnpack(Rd_K, k1)
  AND … AND
  gUnpack(Rs, xn) RENAME {xn AS kn} ⊆
    gUnpack(Rd_K, kn);
```

3.3.3 Temporal denseness

The temporal denseness is defined over all relparts of a relation. It ensures the history completeness (continuity) between historicized relparts of the K-grouping, B-groupings and C-groupings (if applicable). This constraint is inspired by the requirements 3 and 6 in [10] (chap.14).

For each $b_i \in$ B the constraint must verify the relation equality between relparts of the K-grouping and a b_i-grouping. The temporal denseness constraint is defined as follows:

```
CONSTRAINT R_k_R_bi_denseness USING(@V) :
  gSpace(R_K) = gSpace(R_ai){K, @V};
```

For each $c_i \in$ C the constraint must verify the relation equality between relparts of the K-grouping and a c_i-grouping. The temporal denseness constraint is defined as follows:

```
CONSTRAINT R_k_R_ci_denseness
  gSpace(R_K){K} = gSpace(R_ci){K};
```

3.3.4 Key history uniqueness

The key history uniqueness is defined over the value of the timeline associated with K. It ensures non-redundancy and non-contradiction of the key attributes values over time. In other words, it ensures consistency of the history of a tuple by verifying that the same proposition is represented once. This constraint is similar to requirements 1 in [10] (chap.14).

```
CONSTRAINT R_key_uniqueness IS_EMPTY
  (R_K@Vbx JOIN R_K@Vbe WHERE @Vbx < POST(@Vbe));
```

3.3.5 Attribute history uniqueness

The attribute history uniqueness is defined over the value of the timeline attribute associated with a b_i attribute. It ensures non-redundancy and non-contradiction of b_i values over time. In other words, it ensures that the same value of a_i appears only once for a specific period and that different value of b_i appears at different period. This constraint is similar to requirement 4 in [10] (chap.14). For each b_i, the constraint verifies that no tuples with @Vbx is less than the posterior point of @Vbe exists.

```
CONSTRAINT R_bi_uniqueness IS_EMPTY
  (R_ai@Vbx{K, @Vbx} JOIN R_bi@Vbe{K, @Vbe}
    WHERE @Vbx < POST(@Vbe));
```

3.3.6 Key history non-circumlocution

The key history non-circumlocution is defined over the value of the timeline attribute associated to K. It ensures non-circumlocution of the key attributes values over time. This constraint is similar to requirements 2 in [10] (chap.14). The constraint verifies that no tuples with @Vbx is equal to the posterior point of @Vbe exists.

```
CONSTRAINT R_key_circumlocution IS_EMPTY
  (R_K@Vbx JOIN R_K@Vbe WHERE @Vbx = POST(@Vbe));
```

The two key history constraints (uniqueness and non-circumlocution) may be combined in one:

```
CONSTRAINT R_key_history IS_EMPTY
  (R_K@Vbx JOIN R_K@Vbe WHERE @Vbx ≤ POST(@Vbe));
```

3.3.7 Attribute history non-circumlocution

The attribute history non-circumlocution is defined over the value of the timeline attribute associated to an a_i attribute. It ensures non-circumlocution of a_i values over time. This constraint is similar to requirement 5 in [10] (chap.14). For each a_i the constraint verifies that no tuple with @Vbx equal to the posterior point of @Vbe exists.

```
CONSTRAINT R_a_i_circumlocution IS_EMPTY
  ((R_a_i@Vbx{K, ai, @Vbx}
    JOIN R_a_i@Vbe{K, a_i, @Vbe})
      WHERE @Vbx = POST(@Vbe));
```

4 Modification

Many modification operators can be derived from the basic relational assignment expression. The variety of propositions is even greater when comes the time to define new ones addressing timelines. [13, 18, 21, 14, 10]. The operators proposed in this paper have the distinct advantage to be expressed with respect to relations as defined before the historicization along one timeline parameter.

Since the @T timeline is handled by the DBMS, the operations are defined over the @V timeline only (if applicable). The modification process is done in two steps:

- ◊ Initial modification (insertI, deleteI) address the denseness property by distributing the attribute values at their corresponding relpart groupings.
- ◊ Grouping modification (insertG, deleteG) address the non-circumlocution property by modifying the grouping partitions according to the timeline values.

The update operators will be defined, using the following equivalence (AL being an attribute assignment list):

```
UPDATE R WHERE cond : { AL } ≡
  BEGIN
    DELETE R (R WHERE cond),
    INSERT R (EXTEND (R WHERE cond) : { AL })
  END
```

For each relparts modification two operators depending on the timeline category are defined as:

- ◊ Assert a proposition (tuple) since a given time point vbx.
- ◊ Assert a proposition (tuple) during a given period vbe.

4.1 Insertion

Inserting tuples from a 5NF relation into the 6FN relparts is done by « distributing » insertion of each attribute in the appropriate grouping, so denseness is maintained. For K and B-groupings, this is done through insertG statement defined afterwards. For C-groupings, a "standard" INSERT suffice. Note that, proposed operators address the single tuple insertion, extension to relations is straightforward.

The insertI operator ***since*** vbx is defined as:

```
OPERATOR insertI
  (T TUPLE SAME_HEADING_AS R, TIMEPOINT vbx)
  UPDATES RELPARTS_OF (R);
POSSIBLE SYNTAX 'insertI' R T 'SINCE' vbx;
  insertG R_K (EXTEND T{K} : {@Vbx := vbx}),
  insertG R_b1 (EXTEND T{K,b1} : {@Vbx := vbx}),
  …
  insertG R_bn (EXTEND T{K,bn} : {@Vbx := vbx}),
  INSERT R_c1@N T{K,c1},
  …
  INSERT R_cn@N T{K,cn}
END OPERATOR
```

The insertI operator ***during*** vbe is defined as:

```
OPERATOR insertI
  (T TUPLE SAME_HEADING_AS R, PERIOD vbe)
  UPDATES RELPARTS_OF (R);
POSSIBLE SYNTAX 'insertI' R T 'DURING' vbe;
  insertG R_K (EXTEND T{K} : {@Vbe := vbe}),
  insertG R_b1 (EXTEND T{K,b1} : {@Vbe := vbe}),
  …
  insertG R_bn (EXTEND T{K,bn} : {@Vbe := vbe}),
  INSERT R_b1@N T{K,c1},
  …
  INSERT R_bn@N T{K,cn}
END OPERATOR
```

The insertG operators are defined to manage the circumlocution that may occur inside the @Vbe relpart or between the @Vbx relpart and the @Vbe relpart. A special case arises when two periods meet: the possible circumlocution depends on the non-key attribute (if any) having the same value in both tuples or not. This is managed by the join operator used to calculate the t1 temporary relation variable in the following code. Previous constraints guarantee that t1 contains at most one tuple and take care of non-redundancy and non-contradiction (in a similar way a standard insert will cope with a duplicate key insert) :

```
OPERATOR insertG (T TUPLE SAME_HEADING_AS R_g@Vbx)
  UPDATES R_g;
POSSIBLE SYNTAX 'insertG' R_g T;
  WITH (t1 := (R_g@Vbe JOIN RELATION{T})
    WHERE (END(@Vbe)=PRIOR(@Vbx))) :
  IF IS_EMPTY(t1) THEN
    INSERT R_g@Vbx T
  ELSE
    DELETE R_g@Vbe (t1 {ALL BUT @Vbx}),
    INSERT R_g@Vbx
      (EXTEND T : {@Vbx := MIN(t1, BEGIN(@Vbe))
  END IF
END OPERATOR
```

For the ***during*** @Vbe version, a t2 temporary relation variable is needed to test the case when the inserted tuple is fitting exactly between the @Vbx relpart and the @Vbe relpart.

```
OPERATOR insertG (T TUPLE SAME_HEADING_AS R_g@Vbe)
  UPDATES R_g;
POSSIBLE SYNTAX 'insertG' R_g T;
  WITH (
    t1 := (R_g@Vbx JOIN RELATION{T})
      WHERE (END(@Vbe)=PRIOR(@Vbx)) ) :
  IF IS_EMPTY(t1) THEN
    USING (@Vbe) : INSERT R_g@Vbe T
  ELSE
    WITH (
      t2 := R_g@Vbe WHERE (@Vbe MEETS T{@Vbe}),
      t3 := t1{ALL BUT @Vbx} UNION t2) :
    BEGIN
      DELETE R_g@Vbe t2,
      INSERT R_g@Vbx
        (EXTEND t1{ALL BUT @Vbe} :
          {@Vbx := MIN(t3, BEGIN(@Vbe))})
    END
  END IF
END OPERATOR
```

4.2 Deletion

The delete operation is defined as the inserting operation. The delete operators (deleteI ***since*** vbx and deleteI ***during*** vbe) are the direct transposition of the corresponding insertI operators. Still the groupings delete operators must be defined. The delete operation removes tuples having a specific value for specific time points. A special case arises when vbx (of the new tuple) is higher than the @Vbx of the existing tuple, all the prior period must be conserved to maintain the denseness. Note that, proposed operators address the single tuple deletion, extension to relations is straightforward.

The deleteG ***since*** vbx operator is defined as:

```
OPERATOR deleteG (T TUPLE SAME_HEADING_AS R_g@Vbx)
  UPDATES R_g;
POSSIBLE SYNTAX 'deleteG' R_g T;
  WITH (t1 := (R_g@Vbx JOIN
    RELATION{T{ALL BUT @Vbx}}) :
  BEGIN
    DELETE R_g@Vbx t1,
    IF IS_EMPTY (t1 WHERE T{@Vbx} > @Vbx)) THEN
      USING (@Vbe) :
        DELETE R_g@Vbe
          (((R_g@Vbe JOIN RELATION{T})
            WHERE @Vbx ∈ @Vbe)
              {ALL BUT @Vbx})
    ELSE
      USING (@Vbe) :
        INSERT R_g@Vbe
          EXTEND t1 :{@Vbe:=[@Vbx:PRIOR(T{@Vbx})]}
            {ALL BUT @Vbx}
    END IF
  END
END OPERATOR
```

The deleteG ***during*** vbe operator is defined as:

```
OPERATOR deleteG (T TUPLE SAME_HEADING_AS R_g@Vbe)
  UPDATES R_g;
POSSIBLE SYNTAX 'deleteG' R_g T;
 WITH (t1 := (R_g@Vbx JOIN
    RELATION{T{ALL BUT @Vbe}}) :
  BEGIN
    DELETE R_g@Vbx t1,
    IF IS_EMPTY(t1 WHERE BEGIN(T{@Vbe})>@Vbx))
    THEN
      USING (@Vbe) :
        DELETE R_g@Vbe (R_g@Vbe JOIN RELATION{T})
    ELSE
      USING (@Vbe) :
        INSERT R_g@Vbe EXTEND t1 :
          {@Vbe := [@Vbx:PRE(T{@Vbe})]}
          {ALL BUT @Vbx}
    END IF
  END
END OPERATOR
```

4.3 Update

The update operations differ a lot depending on the context of the application. Nevertheless, the basic principle of an update is to replace old tuples with new ones.

The updateI ***since*** vbx operator is defined as:

```
OPERATOR updateI
  (c BoolExp, AL AssignList, TIMEPOINT vbx)
  UPDATES RELPARTS_OF (R);
POSSIBLE SYNTAX 'updateI' R WHERE c 'SINCE' vbx;
  WITH (S := R WHERE cond) :
    deleteI R S SINCE vbx,
    insertI R (EXTEND S : { AL }) SINCE vbx
  END
END OPERATOR
```

The updateI ***during*** vbe operator is defined as:

```
OPERATOR updateI
  (c BoolExp, AL AssignList, PERIOD vbe)
  UPDATES RELPARTS_OF (R);
POSSIBLE SYNTAX 'updateI' R WHERE c 'DURING' vbe;
  WITH (S := R WHERE cond) :
    DELETE R S DURING vbe,
    INSERT R (EXTEND S : { AL }) DURING vbe
  END
END OPERATOR
```

5 Coping with missing information

The objective of this section is to show that the UBHF can be used with minimum effort in presence of missing information.

The ideas of McGoveran [11] and Darwen [9] (chap. 23) can easily be used in UBHF. They suggest normalizing all relations into 6NF to keep track of missing information at the attribute level. Then, each relation is split into two or more parts to represent the following cases: known information, unknown information, not applicable information, etc.

McGoveran and Darwen approach may be simplified by the relational grouping concept defined in UBHF. The process consists of creating for each relpart in B-groupings and C-groupings (a_i in B and C) two relparts (or more relparts, depending on the number of "missing information cases"):

- ◊ R_a_iP: a relpart containing K and the attribute a_i when the attribute value is present;
- ◊ R_a_iM: a relpart containing only K of the tuple when the attribute value is missing.

Constraints are then added to maintain denseness and to avoid contradictions as follows.

For a Vbx partition, the constraint is:

```
CONSTRAINT R_bi@Vbx_MPinavriant
  R_K@Vbx = (R_biP@Vbx{K,@Vbx} D_UNION R_biM@Vbx)
```

For a Vbe partition, the constraint is:

```
CONSTRAINT R_bi@Vbe_MPinavriant USING (@Vbe) :
  R_K@Vbe = (R_biP@Vbe{K,@Vbe} D_UNION R_biM@Vbe)
```

For a N partition, the constraint is:

```
CONSTRAINT R_ci_MPinavriant
  R_K@N = (R_ciP@N{K} D_UNION R_ciM@N)
```

This "MP" decomposition can be done before or after the historicization. In the "before" approach the initial relation in 5NF is normalized to 6NF than for each b_i-relparts and c_i-relparts where missing information can occur MP decomposition are applied. Finally, the resulted relparts are historicized regarding the process described in section 3. The main advantage of the "before" approach is that the constraints described in section 3.3 stay unchanged. The "after" approach applies to the MP decomposition for each b_i-relparts or c_i-relparts in the leaf partition. The gSpace can then be extended to take P-relparts and M-relpart into account so the new constraints maybe "merged" in the original ones. At the end picking either of these two approaches result in similar leaves relparts. The figure below illustrates the example of extending the relparts of a Vbx partition.

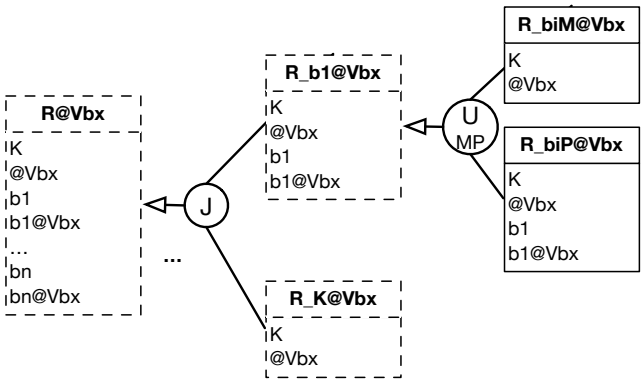

Figure 6. Coping with missing information in Vbx partition

More work is required to delineate and define good modification operations in the context of missing information.

6 Models as UBHF views

In this section, we demonstrate the integration and generalization of BCDM and DDLM as views over UBHM, a model defined using UBHF. Thus, there is no need to redefine the constraints. The view definition algorithms are described below and an example is presented in section 8.1.

6.1 UBHM as a UBHF view

A unified bitemporal historical model is defined by retaining all the leaves of UBHF. The view definition of the model is then a simple one to one correspondence with them. UBFM is useful in many circumstance as we discuss in section 5.

6.2 BCDM a UBHM view

BCDM defines a bitemporal model. The bitemporal relations are explicitly stored in the schema and are PJ decomposed and RU decomposed regarding the value of the transaction timeline attribute. The transaction timeline is managed by user-defined triggers. According to UBHF semantics for each relation R, BCDM schema contains the following views:

- ◊ union of VbxTbx and VbeTbx for each K and each b_i;
- ◊ union of VbxTbe and VbeTbe for each K and each b_i;
- ◊ a Tbx and a Tbe for each c_i.

GENRATE BCDM SCHEMA AS A VIEW

```
for all R = (K, B, C, Dv, Dt) in {the initial schema}
  for all relparts X of R in {R_K and B-groupings (R_bi) }
   DEFINE VIEW BCDM6.X@VTbx AS
    UBHF. X@VbxTbx EXTEND : (@V := INTERVAL[@Vbx:ω])
     {ALL BUT @Vbx};
    UNION
    UBHF.X@VbeTbx RENAME (@Vbe as @V)
   DEFINE VIEW BCDM6.X@VTbe AS
    UBHF.X@VbxTbe EXTEND : (@V := [@Vbx:ω])
     {ALL BUT @Vbx};
    UNION
    UBHF.X@VbeTbe RENAME (@Vbe as @V)
  end for;
  for all relparts Z of R in {C-groupings (R_ci)}
   DEFINE VIEW BCDM6.Z@Tbx AS UBHF.Z@Tbx
   DEFINE VIEW BCDM6.Z@Tbe AS UBHF.Z@Tbe
  end for
end for
```

6.3 Bitemporal DDLM as a UBHM view

DDLM defines a unitemporal model called *full-temporal model* and its bitemporal extension. The unitemporal part contains valid-time relations RU decomposed regarding the value of the valid timeline attribute: @Vbx (*SINCE relation*) and @Vbe (*DURING relation*). Only, the Vbe partition is PJ decomposed over K, and a_i-attributes, The bitemporal part extended the unitemporal part by generating for each relation a view with known transaction timeline period (@Tbe) extracted from the DBMS journal. According to UBHF semantics for each relation R, DDLM schema contains the following views:

- ◊ Vbx (SINCE): join of K and B-groupings of Vbx partition;
- ◊ Vbe (DURING): one for K and one for each ai-attribute over the Vbe partition;
- ◊ VbxTbe and VbeTbe;
- ◊ Tbe: one for each c_i.

Note. DDLM does not explicitly mention how to cope with b_i-attributes. The b_i-relparts of the schema view is then, strictly speaking, an extension to the bitemporal DDLM.

GENRATE BITEMPORAL DDLM SCHEMA AS A VIEW

```
for all R = (K, B, C, Dv, Dt) in {the initial Schema}
  //Key relparts views
  DEFINE VIEW DDLM.R_SINCE AS
    (UBHF.R_K@Vbx RENAME {@Vbx AS since})
    JOIN
    (UBHF.R_a1@Vbx RENAME {@Vbx AS b1_since})
    JOIN ... JOIN
    (UBHF.R_an@Vbx RENAME {@Vbx AS bn_since});
  DEFINE VIEW DDLM.R_SINCE_LOG AS
    UBHF.R_K@VbxTbx RENAME {@Vbx AS since} EXTEND:
      (since_log:= INTERVAL[@Tbx:now()]) {ALL BUT @Tbx}
    UNION
    UBHF.R_K@VbxTbe
      RENAME {@Vbx AS since, @Tbe AS since_log};
  DEFINE VIEW DDLM.R_DURING AS
    UBHF.R_K@Vbe RENAME {@Vbe AS during};
  DEFINE VIEW DDLM.R_DURING_LOG AS
    UBHF.R_K@VbeTbx RENAME {@Vbe AS during} EXTEND:
      (since_log:= [@Tbx:now()]) {ALL BUT @Tbx}
    UNION
    UBHF.R_K@VbeTbe
      RENAME {@Vbe AS during, @Tbe AS during_log};
  //A relpart views
  for bi in B
    DEFINE VIEW DDLM6.R_bi_DURING AS
      UBHF.R_bi@Vbe RENAME {@Vbe AS during};
    DEFINE VIEW DDLM6.R_bi_DURING_LOG AS
      UBHF.R_bi@VbeTbx RENAME {@Vbe AS during} EXTEND:
        (since_log:= [@Tbx:now()]) {ALL BUT @Tbx}
      UNION
      UBHF.R_bi@VbeTbe
        RENAME {@Vbe AS during, @Tbe AS during_log};
  end for;
//B relpart views (transaction only)
  for ci in C
    DEFINE VIEW DDLM6.R_ci_LOG AS
      UBHF.R_ci@Tbx EXTEND:
        (during_log:= [@Tbx:now()])
          {ALL BUT @Tbx}
      UNION
      UBHF.R_ci@Tbe RENAME {@Tbe AS during_log}
  end for
end for;
```

7 Discussion

Many not so evident properties of BCDM and DDLM becomes apparent when UBHF is used to express them.

7.1 BCDM and DDLM comparison

7.1.1 Structure

The partition categorization shows that the structure of DDLM and BCDM differ. For a valid-time relation, DDLM separate the relation into tow relparts Vbx and Vbe, BCDM does not. For a transaction-time relation, DDLM does not mention specific treatment, BCDM separates the relation into tow relparts Tbx and Tbe. For a bitemporal relation, DDLM separates the relation into two layers: a valid unitemporal one (with Vbx and Vbe) and a bitemporal one (with Tbe); BCDM defines only a bitemporal layer with VTbx and VTbe. Figure 7 shows the partitions used in each model.

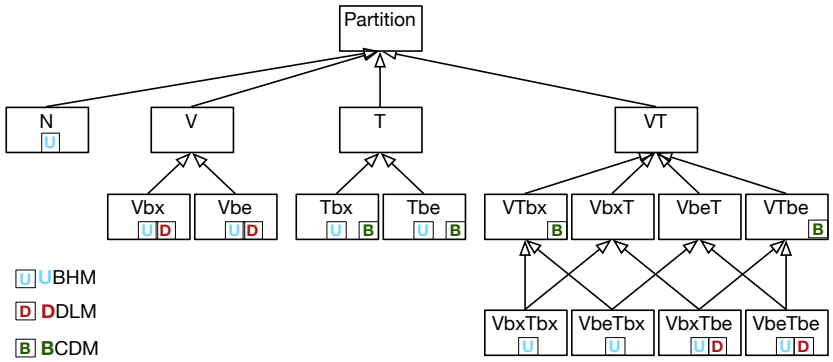

Figure 7. Partitions composing UBHM, DDLM and BCDM

The difference is the consequence of the decomposition sequence and conditions defined through the historicization. In DDLM the RU decomposition is over the valid timeline and in BCDM it is over transaction timeline. However, the two models contain the same data but are represented differently. Furthermore, notice that the two models are complementary by offering data access at different timeline level. DDLM put the emphasis on the fact validity, and BCDM on the fact correction. Despite this, the two models are interoperable, and can be easily derived from each other without losing data.

7.1.2 Constraints

DDLM set the semantic of both the valid and the transaction timelines. It is then possible to (pre)define the appropriate constraints. On the other hand, in BCDM, valid timeline depends on the application domain. The constraints definition in DDLM is more straightforward because of its uniform definition and the separation between known and unknown timeline attributes values. The BCDM constraints are harder to define since they must cope simultaneously with validation semantic and domain semantic.

7.2 BCDM and DDLM usage

DDLM presents a standard process to historicized uniquely each relation, BCDM presents a process depending on the application domain by choosing the temporal category of each relation at the entity-relationship level. However, with UBHF, the two processes can be standardized and used jointly, using the views presented in section 4. Designing a data warehouse using UBHM enables the integration of sources structured with either model, and making them interoperable. The separation between the Vbx relparts and the Vbe relparts in DDLM is well suited for transactional operations and for analytic operations respectively. The separation between the Tbx relparts and the Tbe relparts in BCDM also gives more expressivity to transactional operations. These parts can physically be organized and optimized along this separation.

Defining both BCDM and DDLM in terms of UBFM views allows automatic integration of the two models, so the definition domain constraints are clearly distinct from the domain ones and specifying is made easier.

7.3 Limitations

With UBHF, a sound temporal schema can be designed ensuring a unified data integrity semantic, query expressiveness and guided automation. Currently proposed methods define transformation rules “by-example” and must largely be tailored and applied manually. This paper presents a unified bitemporal framework to help reach guide automation for designing historicized database schema reducing errors and

costs. More specifically, UBHF defines (a) relation, attribute and timeline categorization to provide unique semantic; (b) unified temporal structure and general constraints to be independent of the domain (or context) yet providing formal definition and superior automation capabilities; (c) historicization process with traceability over the transformation steps without losing the initial schema conceptual view, including some hints to cope with missing information; (d) UBHF views of DDLM and BCDM with an extension to model multi-relational historicization categories; and (e) basic modification operations. UBFM illustrates the capability of using UBHF to define new models. With UBHF the historicization is done with minimum intervention of the database designer. Furthermore, BCDM and DDLM can be easily automated with respect to the concepts and historicization process of UBHF as shown in figure 8. Furthermore, our proof of concept illustrates that it can be applied using a subset of standard SQL available in many currently available DBMS. Finally, the proposed framework does not need any extension to the relational theory.

7.4 Further work

Further work is planned to produce a fully operational solution. Here is a short list:

- ◊ Define the full missing information approach (missing, no applicable), with convenient modification operators.
- ◊ Support past indeterminacy (xe) in a similar way as it is done for future indeterminacy (bx).
- ◊ Support temporal uncertainty.
- ◊ Support other timelines (event time, decision time, etc.).
- ◊ Define an optimized SQL implementation of UBHF through a modified TutorialD to SQL translator.
- ◊ Build a UBHF engine to natively execute interval logic analysis and translate it into SQL.

8 Conclusion

A sound temporal schema plays an essential role in increasing query expressiveness making easier for developers to define and maintains their algorithms. “On the other hand, a theoretically grounded solution should be provided once and for all, so that application developers can safely adopt it, and just focus on the application-dependent aspects of their problems” [3]. This work is a starting point for the development of a temporal database modelling tool based on solid models and already existing theoretical solutions.

9 References

1. Allen, J.F.: Maintaining Knowledge About Temporal Intervals. Commun ACM. 26, 11, 832–843 (1983).
2. Allen, J.F., Ferguson, G.: Actions and Events in Interval Temporal Logic. The University of Rochester, Computer Science Department (1994).
3. Anselma, L., Piovesan, L., Terenziani, P.: A 1NF temporal relational model and algebra coping with valid-time temporal indeterminacy. J. Intell. Inf. Syst. 1–30 (2015).
4. Anselma, L., Stantic, B., Terenziani, P., Sattar, A.: Querying now-relative data. J. Intell. Inf. Syst. 41, 2, 285–311 (2013).
5. Codd, E.F.: Extending the Database Relational Model to Capture More Meaning. ACM Trans Database Syst. 4, 4, 397–434 (1979).
6. Codd, E.F.: The Relational Model for Database Management: Version 2. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1990).
7. Combi, C., Keravnou, E.T., Shahar, Y.: Temporal Information Systems in Medicine. Springer (2010).
8. Darwen, H., Date, C.J.: The Third Manifesto. SIGMOD Rec. 24, 1, 39–49 (1995).
9. Date, C.J., Darwen, H.: Database Explorations: essays on the Third Manifesto and related topics. Trafford Publishing (2010).
10. Date, C.J., Darwen, H., Lorentzos, N.A.: Time and relational theory: temporal databases in the relational model and SQL. Morgan Kaufmann, Waltham, MA (2014).
11. Date, C.J., Darwen, H., McGoveran, D.: Nothing from Nothing (part 4). Relational database writings, 1994-1997. pp. 377–394 Addison-Wesley, Harlow, England ; Reading, Mass (1998).
12. Jensen, C.S., Dyreson, C.E., Bohlen, M., Cliford, J., Elmasri, R., Gadia, S.K., Grandi, F., Hayes, P., Jajodia, S., 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.T., Soo, M.D., Tansel, A., Tiberio, P.,

Wiederhold, G.: The consensus glossary of temporal database concepts-February 1998 version. Proceedings of Seminar Temporal Databases: Research and Practice, 23-27 June 1997. pp. 367–405 Springer-Verlag (1998).

13. Jensen, C.S., Soo, M.D., Snodgrass, R.T.: Unifying Temporal Data Models via a Conceptual Model. Inf. Syst. 19, 513–547 (1993).
14. Johnston, T., Weis, R.: Managing time in relational databases: how to design, update and query temporal data. Morgan Kaufmann/Elsevier, Amsterdam ; Boston (2010).
15. Khnaisser, C., Lavoie, L., Diab, H., Ethier, J.-F.: Data Warehouse Design Methods Review: Trends, Challenges and Future Directions for the Healthcare Domain. In: Morzy, T., Valduriez, P., and Bellatreche, L. (eds.) New Trends in Databases and Information Systems. pp. 76–87 Springer International Publishing (2015).
16. Kline, N.: An Update of the Temporal Database Bibliography. SIGMOD Rec. 22, 4, 66–80 (1993).
17. Lorentzos, N.A., Johnson, R.G.: TRA: A Model for a Temporal Relational Algebra. In: Rolland, C., Bodart, F., and Léonard, M. (eds.) Temporal Aspects in Information Systems, Proceedings of the IFIP TC 8/WG 8.1 Working Conference on Temporal Aspects in Information Systems, Sophia-Antipolis, France, 13-15 May, 1987. pp. 95–108 North-Holland / Elsevier (1987).
18. Lorentzos, N.A., Poulovassilis, A., Small, C.: Implementation of Update Operations for Interval Relations. Comput J. 37, 3, 164–176 (1994).
19. Meyden, R. van der: Logical Approaches to Incomplete Information: A Survey. In: Chomicki, J. and Saake, G. (eds.) Logics for Databases and Information Systems. pp. 307–356 Springer US (1998).
20. Rönnbäck, L., Regardt, O., Bergholtz, M., Johannesson, P., Wohed, P.: Anchor modeling—Agile information modeling in evolving data environments. Data Knowl. Eng. 69, 12, 1229–1253 (2010).
21. Snodgrass, R.T.: Developing time-oriented database applications in SQL. Morgan Kaufmann Publishers, San Francisco, California (2000).
22. Snodgrass, R.T.: The TSQL2 Temporal Query Language. Springer (1995).

10 Appendix

The figure below illustrates the results relparts of the historicization of a relation R(K, {b_1, ..., b_n}, {c_1, ..., c_m},) according to UBHF and the generation of BCDM and bitemporal DDLM views.

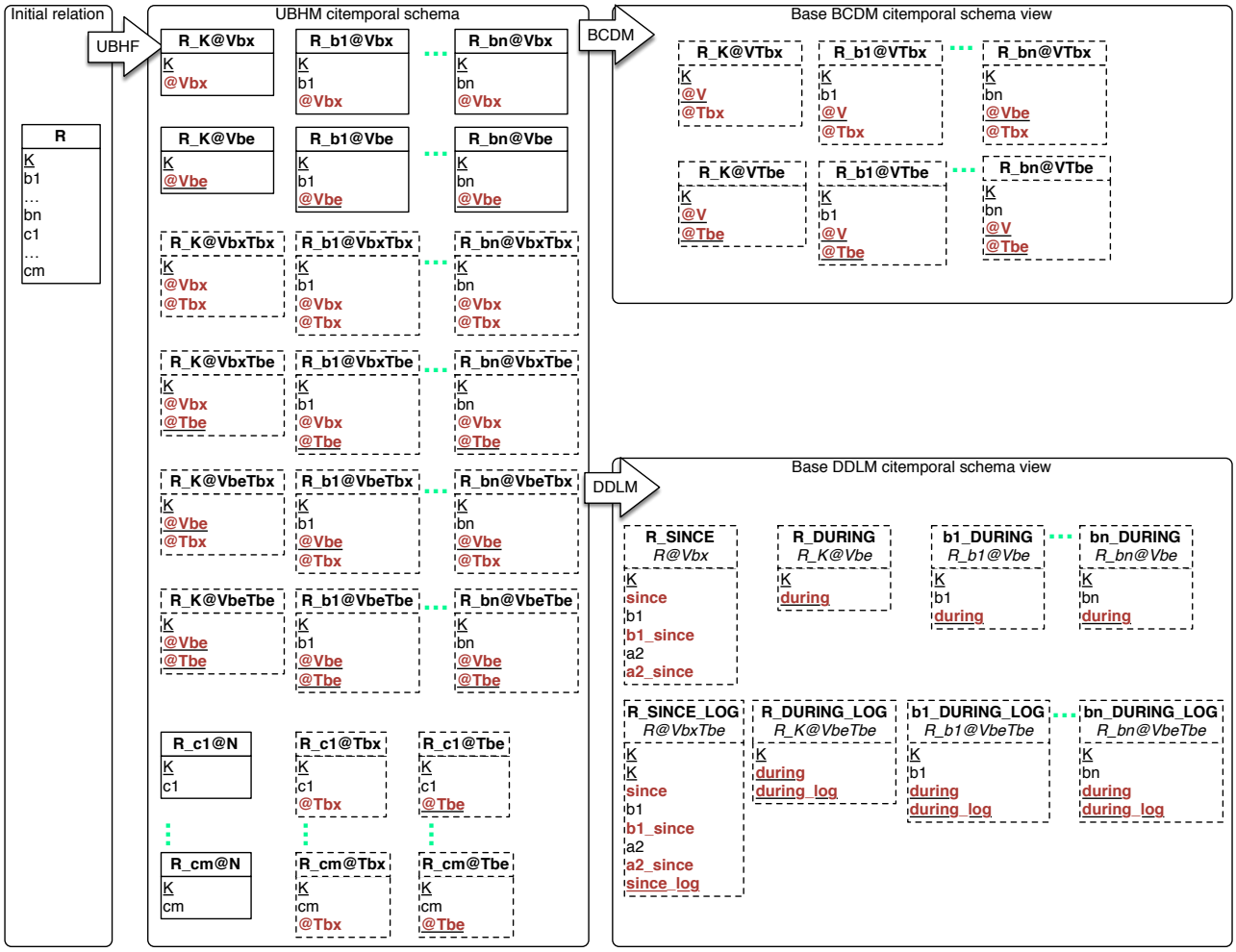

Figure 8. Historicization structure of DDLM, BCDM and UBHM in terms of UBHF.