Schema Decomposition via Transformation Patterns

Théo Abgrall[1,*]

[1]*Free University of Bozen-Bolzano*

Abstract

A goal of database reverse engineering techniques is the extraction of a conceptual model describing an otherwise complex or obtuse database system. In such contexts, and notably, when applied to legacy databases, thorough preservation of the knowledge lying in the source schema is a reasonable priority. In the common database reverse engineering pipeline, database normalization is the process in which we are most able to dictate and maintain the information preservation property. We make use of the transformation pattern formalism as our base to guarantee information preservation. We present how these theoretical templates can be applied over complex database schema and how they can serve as the foundation of an automatic, deterministic database normalization methodology.

Keywords

Schema Transformation, Information Preservation, Conceptual Modeling, Database Normalization

1. Introduction

As time passes, knowledge of and used in database management systems is at risk of becoming either lost or outdated. There are ways to mitigate this inevitability by requiring strict control and upkeep of a persistent conceptual model for each change done over the original database [1]. But in some scenarios, such as legacy database, the only approach left is recovery. Recovery of several forms of knowledge usually takes the form of a brand new conceptual model emerging from the bribes of information at its disposal. **Database Reverse Engineering** (DBRE) encompasses the set of methods and tools aiming for this result. Naturally, in such a context, the desire for the conceptual model generated to be precise and to reflect exactly the veiled information is quite high. Thus emerges our will to mix theory on information preservation with this process of conceptualization.

Our foundation is the research done in [2] which explores the application of information capacity preservation in the restricted context of first-order schemata. Focusing on a simple notation based on first-order logic in which we translate concepts of database theory allows for the logical verification of constraint and schema preservation. This research was done through the lens of DBRE to provide methods for transforming databases. First, into other ones in higher normal forms and then, into conceptual models after passing by an intermediary step incorporating the abstract relational model notation. This process eventually crystallises into specific templates designed to represent database transformations called **Transformation Patterns** (TPs). This strong formalism was however lacking a proper associated application

SEBD 2024: 32nd Symposium on Advanced Database Systems, June 23-26, 2024, Villasimius, Sardinia, Italy

[*] Corresponding author.

tabgrall@unibz.it (T. Abgrall)

0000-0001-9999-2756 (T. Abgrall)

methodology and served more as guidance for database engineer. To illustrate the various sections of the methodology we provide the following ongoing example Fig. 1:

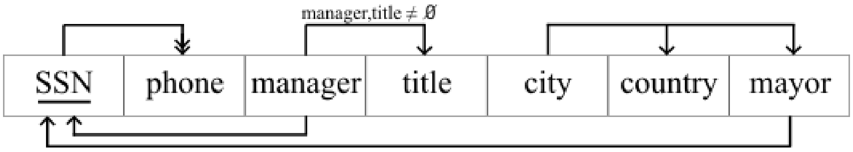


Figure 1: Ongoing Example

This example presents a legacy database schema with a single universal relation

$$Person : (SSN, phone, manager, title, city, country, mayor)$$

onto which are applied several constraints: - a primary key on SSN denoted by a bold underline - a functional dependency $city \rightarrow country, mayor$ written with a top single arrow - another functional dependency $manager \rightarrow title$, this time restrained to tuples satisfying a condition $manager, title \neq \mathit{NULL}$ requiring both attributes to be non-null - a multivalued dependency $SSN \twoheadrightarrow phone$ signed with a top double-headed arrow and finally - two inclusion dependency $manager \subseteq SSN$ and $mayor \subseteq SSN$, both shown as bottoms arrows.

Using transformation patterns as our base, we propose a methodology based on them for automatic database normalization. It starts with a primer on information capacity and what leads us to focus on TPs as a means of preserving knowledge with no loss. Once we present how these restrictive patterns could be applied to more complex schemas, we describe a methodology to remove all null values and all join dependencies from the source schema via schema decompositions. For closure, we contextualize our work among the many adjacent fields it grazes before discussing our future application of transformation patterns.

2. Background

2.1. Information Preservation

At its core, most of the literature regarding the preservation of information during a schema transformation emerged from Hull's [3] definition of four decreasingly strict notions of schema dominance. The main idea behind schema dominance is that for two relational schemas, any valid database instance in one of the schemas can be expressed in the other.

Definition 1 (Schema Transformation). *Let there be two schemas S and T. A* ***schema transformation*** *from S to T is a mapping function $f_{S \rightarrow T} : S \rightarrow T$ such that $f_{S \rightarrow T} I(S) \rightarrow I(T)$ where I correspond to all valid database instances of S and T, respectively. Furthermore, a mapping function $f_{S \rightarrow T} : S \rightarrow T$ is* ***total*** *if $\forall x \; \exists y \; f_{S \rightarrow T}(x) = y$,* ***injective*** *if $\forall x, x' \; f_{S \rightarrow T}(x) = f_{S \rightarrow T}(x')$,* ***surjective*** *if $\forall y \; \exists x \; f_{S \rightarrow T}(x) = y$ and* ***bijective*** *if $f_{S \rightarrow T}$ is total, injective and surjective.*

Definition 2 (Schema Dominance). *Let there be two schemas S and T and the two mapping functions $f_{S \to T}$ and $f_{T \to S}$. We say that S* ***dominates*** *T if for any instance $I(T)$ of T: $f_{S \to T} \circ f_{T \to S} I(T) = I(T)$. If S dominates T, then the mapping function $f_{T \to S}$ is injective and total.*

Definition 3 (Schema Equivalence). *Let there be two schemas S and T. We say that S and T are* ***equivalent*** *if and only if S dominates T and T dominates S. If S and T are equivalent, then the mapping function $f_{S \to T}$ is bijective.*

A genesis of schema dominance can be found in [4] where it is defined as a query-equivalence problem. Later, these definitions were expanded [5] by answering the conjuncture that schema transformations could preserve primary keys, thus implying the preservation of not only instances but also relational constraints. Once more, the notions of schema dominance and equivalence were extended in [6] which defines a correct schema transformation as one which preserves both constraints and instances, thus differentiating between instance preservation, constraint preservation and their combination, information capacity preservation.

2.2. Transformation Patterns

Walking in [2] footsteps, we restrict our definition of schema to that of first-order schemas. A *first-order schema* (FOS) S is a couple $[\mathbb{A}_S, \mathbb{C}_S]$ where $\mathbb{A}_S$ define the alphabet of predicates used, therefore databases tables, and $\mathbb{C}_S$ define the set of constraints applied over these relations. A predicate, written $r(a_1, a_2, ..., a_n)$ is composed of a predicate name r and a finite set of attribute names of arity n. FOS provides a representation of various kinds of constraints, ranging from functional dependencies to constraints on values. More interestingly, restricting ourselves to first-order logic semantics allows for simple proofs of FOS equivalence through mutual entailment via a process automated by theorem solvers, in our case we use Prover9 [7]. Now that we need a way of writing schema transformation per the semantics of first-order logic, we move toward a definition of view as a mapping function over both alphabets of two FOS:

Definition 4 (FO Mappings). *For two schema S and T, a First-Order (FO) Mappings $M_{S \to T}$ is a set of views $a_T = \phi_S$ where a_T is each predicate present in T and ϕ_S is an first-order expression over the alphabet $\mathbb{A}_S$ of S.*

For readability, we write those FO Mappings in relational algebra. We can now translate the previous definitions of schema dominance by interchanging schema transformation with FO mappings. Thus resulting in the notion of FO Dominance, informally a schema S FO Dominate another schema T if S dominates T via mapping functions $f_{S \to T}$ and $f_{T \to S}$ written as FO Mappings. Using the FOS notation, we can also write FO Dominance as such $(\mathbb{C}_S \cup M_{S \to T}) \models (\mathbb{C}_T \cup M_{T \to S})$ This lead to this final definition of Losslessness;

Definition 5 (Losslessness). *Let S and T be two first-order schemas, if S FO Dominates T and T FO Dominates S, then S and T are the lossless representations of each other. A result which we can also write as* $(\mathbb{C}_S \cup M_{S \to T}) \equiv (\mathbb{C}_T \cup M_{T \to S})$

Transformation patterns [2] are crafted templates describing a particular schema transformation and the constraints necessary to ensure its losslessness. It can be written as the triple $TP = (DP_S, DP_T, M)$ with DP_S and DP_T, two database patterns whose purpose is to represent the state of a database on both sides of a transformation M written as mappings pattern in relational algebra. **Database Pattern** and **Mapping Patterns** are both similar to FOS and relational algebra mappings respectively, with the difference that instead of manipulating attributes, they manipulate variables. **Variables** are sets of attributes defined by the constraints applied to them. For example, a relation $Worker(SSN, name, dob)$ with a sole primary key constraint on SSN fits the following relation pattern $R(PK, REST)$. A database pattern is a couple $DP = (R, C)$ composed of a set of relation patterns and one of constraint patterns. **Relations Patterns** are defined similarly to FOS predicate $R : (ATT_1, ATT_2, ..., ATT_n)$ with R standing as the relation name and $ATT_1, ..., ATT_n$ being variables either present in some constraint patterns or unconstrained and qualified as $REST$. **Constraint Patterns** forms vary from dependency to dependency, but they all can be written as $\psi^{\theta}[R_1, ...R_n](ATT_1, ..., ATT_m)(\sigma_{condition})$. ψ denotes the type of constraints used and θ precises that type. Notably differentiating between a total inclusion dependency written $ind^{=}[R_1, R_2](ATT_1, ATT_2)$ and a isA one $ind^{\subseteq}[R_1, R_2](ATT_1, ATT_2)$. Finally, $cond$ denotes a conjunction of simple conditions over values. Despite being a bit heavy, this formal notation is necessary as a thorough assignment of each constraint over their associated relation is mandatory.

2.2.1. Example

We present a short example illustrating the potential of TPs. In Tab. 1 (a) we have a simplified version of our ongoing example $Worker : (SSN, phone, manager, city)$ with only one constraint: $manager$ is nullable. We want to horizontally split it into a second database schema (b) containing two tables: one where all values of $manager$ are non-null ($Worker\_w\_Manager$) and in the other there is no $manager$ ($Worker\_wt\_Manager$). Because there is little worth in keeping a column full of null values, we also want to drop $manager$ from the second table.

Worker

SSN	phone	manager	city
012-3155	133039133	024-5143	Southfield
012-3155	125682994	024-5143	Southfield
024-5143	181332948	$NULL$	Pearland
031-7471	131297308	040-5166	Pearland
040-5166	156422537	$NULL$	Southfield
040-5166	187797355	$NULL$	Southfield

(a) Source Schema

Worker_w_Manager

SSN	phone	manager	city
012-3155	133039133	024-5143	Southfield
012-3155	125682994	024-5143	Southfield
031-7471	131297308	040-5166	Pearland

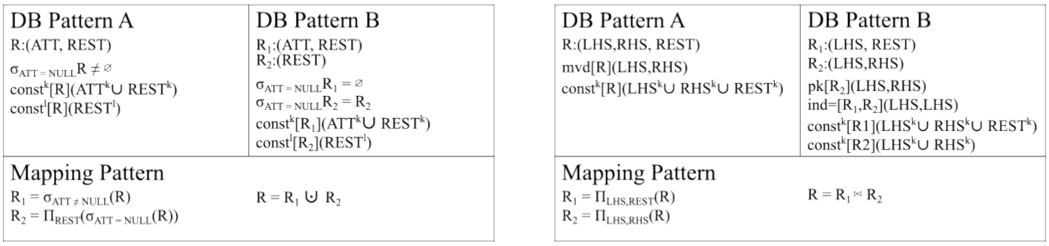
Worker_wt_Manager

SSN	phone	city
024-5143	181332948	Pearland
040-5166	156422537	Southfield
040-5166	187797355	Southfield

(b) After Horizontal Decomposition

Table 1
Example of a Database Horizontal Decomposition

First, we need a way to represent (a) in our formalism, and it starts with expressing that $manager$ is nullable. A condition over a set of attributes is commonly written as $\sigma_{cond}R \neq \varnothing$ in a TP to denote that in the relation R, tuples satisfying $cond$ could exists. This is called a guard constraint. They can also express conditions that either always hold or never do by writing $\sigma_{cond}R = R$ and $\sigma_{cond}R = \varnothing$ respectively. In our example, the nullable attribute manager would be written as such $\sigma_{manager=NULL}Worker \neq \varnothing$. Put as a database pattern, we have a constraint pattern $\sigma_{ATT=NULL}R \neq \varnothing$ applying over a set of attributes ATT, coalescing with any other number of unrelated attributes captured as the $REST$ into the relation pattern $R : (ATT, REST)$. This is the first half of a transformation pattern, the part onto which we can match our relation $Worker$. The second part denotes instead the two tables where $manager$ is either always null or non-null. Once again, in our formalism, these two conditions are expressed as $\sigma_{ATT=NULL}R_1 = \varnothing$ and $\sigma_{ATT=NULL}R_2 = R_2$ for $cond$ is never satisfied and $cond$ is always satisfied respectively. From these conditions, we can deduce two relation patterns $R_1 : (ATT, REST)$ and $R_2 : (REST)$. Finally, its mapping patterns is two sets of FO-Mappings $R_1 = \sigma_{ATT \neq NULL}(R)$, $R_2 = \Pi_{REST}(\sigma_{ATT=NULL}(R))$ and $R = R_1 \uplus R_2$. These mappings uses standards relational algebra notation plus the outer union operation denoted by $\uplus$. The full TP is shown in Fig. 2 (a) where an additional constraint $const$ is used to capture residual constraints, a notion developed in the next section. Applying this TP over our $Worker$ relation requires to match the variables in the former with the attributes in the latter. To do so, $Worker$ has to be rewritten as an FOS so that its semantics can match those of any TPs. Here, this match is trivial and it is quickly deduced that $ATT = \{manager\}$ and $REST = \{SSN, phone, city\}$. We call a transformation pattern, a database pattern or a mapping pattern instantiated once it has been matched with real values from a database schema.



DB Pattern A	DB Pattern B
R:(ATT, REST) $\sigma_{ATT = NULL}R \neq \varnothing$ $const^k[R](ATT^k \cup REST^k)$ $const^l[R](REST^l)$	R_1:(ATT, REST) R_2:(REST) $\sigma_{ATT = NULL}R_1 = \varnothing$ $\sigma_{ATT = NULL}R_2 = R_2$ $const^k[R_1](ATT^k \cup REST^k)$ $const^l[R_2](REST^l)$
Mapping Pattern	
$R_1 = \sigma_{ATT \neq NULL}(R)$ $R_2 = \Pi_{REST}(\sigma_{ATT = NULL}(R))$	$R = R_1 \uplus R_2$

(a) TP for Horizontal Decomposition

DB Pattern A	DB Pattern B
R:(LHS,RHS, REST) mvd[R](LHS,RHS) $const^k[R](LHS^k \cup RHS^k \cup REST^k)$	R_1:(LHS, REST) R_2:(LHS,RHS) pk[R_2](LHS,RHS) ind=[R_1,R_2](LHS,LHS) $const^k[R1](LHS^k \cup RHS^k \cup REST^k)$ $const^k[R2](LHS^k \cup RHS^k)$
Mapping Pattern	
$R_1 = \Pi_{LHS,REST}(R)$ $R_2 = \Pi_{LHS,RHS}(R)$	$R = R_1 \bowtie R_2$

(b) TP for Vertical Decomposition of a MVD

Figure 2: Example of a Transformation Patterns

The decomposition resulting from a TP application can be expressed as a set of views. This is crucial when working with legacy databases or in cases where it's neither practical nor desirable to actively modify the source database. And because the transformations are lossless and thus preserve information capacity, it is possible to update the source database from any of its equivalent views [8]. As they stand, TPs are useful tools for DBRE to database engineers as guidance for a normalization process, as well as an eventual translation into a conceptual model. TPs notably offer a way to resolve the issue of identity resolution in database schema. The crux is that in a traditional relational database, there is no way to identify an object with the same precision a conceptual model would allow. The closest is primary keys, but there are many cases in which it gets complicated. Arises the $Abstract\ Relational\ Model$ (ARM) [9], a model which explicitly gives an Object Identifier to relational schemas. An extensive catalogue of TP turning databases into their ARM counterparts was presented in [2]. However, they required the database pattern to already be in a fully decomposed form, a process never explored before.

3. Transformation Pattern Application

One thing missing for the application of TPs is a proper understanding of the $const$ constraint and more generally of the process of constraint propagation. If we added $mvd[R](SSN, phone)$ and $pk[R](SSN)$ to the relation $Worker$ used in last example, then they would be considered as residual constraints in the application of the TP a) in Fig. 2 and assigned to both $REST^k$ and $REST^l$ as constraints over $REST$. Here, ^{k} and ^{l} separate the residual constraints applied over all variables satisfying $cond$ and those that do not, respectively. Such basic propagation seems straightforward in our example but as we apply TPs over databases containing several tables, all linked via foreign keys, then we need to have a proper propagation methodology. One that has two main objectives: -matching the TP and the database schema and - merging the newly decomposed schema with the rest of the old one. In Fig. 3 we present the main components of a successful pattern application, this time starting with a vertical decomposition following $SSN \twoheadrightarrow phone$ (see Fig. 2 (b)) and showing how this affect the attribute $manager$.

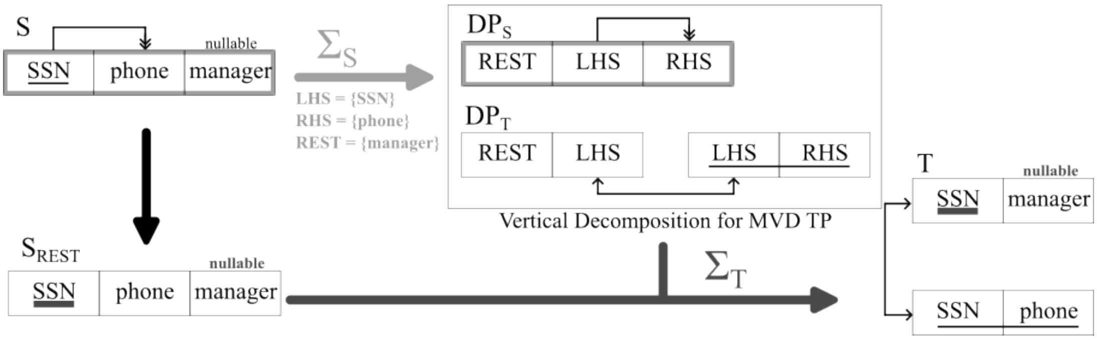


Figure 3: Showcase of a TP for multivalued dependencies decomposition (see Fig. 2 (b)) applied over a schema example. S_{REST} is another FOS corresponding to S minus the constraint matched by Σ_S

Here, Σ_S denote the mapping function between the source database S and the desired TP, matching with DP_S. The mapping is trivial when only one constraint needs to be matched. Once the TP is applied and we have a decomposition in DP_S, we need to merge this new database schema with the previous one, propagating all constraints from the latter in the process. We call the constraints left behind after the application of a TP leftover constraints. In Fig. 3, the leftover constraint is the guard constraint $\sigma_{manager=NULL} Worker \neq \emptyset$. The propagation of those leftover constraints, and of their associated tables is done via a set of rules defined in the merging function Σ_T. Later on, we will introduce further rules when dealing with overlapping guard constraints, a then-defined notion, but for now, the three most important rules are:

- **Basic Propagation:** If a leftover constraint applies over a set of attributes found again in DP_T, then it is propagated fully.
- **Inclusion Update:** Following a vertical decomposition, any inclusion dependencies over a set of attributes captured by the TP will get updated by changing the relations it applies over (i.e., one of the newly generated relations).
- **Inclusion Split:** Following a horizontal decomposition, a previously total inclusion dependency ($ind^=$) will split into two subset dependencies ($ind^\subseteq$) with the corresponding new relations names.

Now that we have an intuitive base to process multiple TPs it is time to move on to our proposed decomposition methodology.

4. Decomposition Process

Stemming out DBRE, TPs theory takes roots deeper inside the field of **Database Normalization** [10] centering around the question of data quality through the prism of structural redundancy. One of its solutions is the process of data decomposition in which a sequence of operations is applied over a database schema to remove redundancies and null values improving data consistency. Normal forms (NF) is the metric used to define the set of desirable properties a database satisfies. Ranging from the 1NF which checks for the atomization of each cell to the 5NF satisfied if all non-trivial join dependencies are only applied over superkeys. So far, most of the normalization approaches have taken the form of an algorithm [11] [12] focalizing on properties of functional dependencies and consequently rarely going beyond the Boyce-Codd NF. We believe that an approach based on lossless TPs can not only work just as well but also go beyond and reach the canonical 6NF, taking care of null values in the process. In this section, we will present several types of TPs along with a short methodology on how to apply them. Focusing only on a single database schema under the Universal Relation Assumption (URA), we assume that it is at least in 1NF and that its set of constraints is acyclic. We naturally expect the whole decomposition process to be lossless by definition, but another property we are aiming for is to attain some kind of determinism. In the last section, we unfolded a twice decomposed database schema, first vertically then horizontally. Of course, another sequence would have given another result. To say which final output is preferable might sound arbitrary, but we claim that the former, the one ending with fewer tables, is actually preferable and more in line with a restructuring of knowledge made for conceptual models.

4.1. Inclusion Dependency Extraction

We preemptively create a singleton table based on the right-hand side of an inclusion dependency, then change that constraint to target the newly generated table. In the database schema S made of $Worker : (SSN, phone, manager)$ and $ind^\subseteq[Worker, Worker](manager, SSN)$ decomposing into S' makes $Worker : (SSN, phone, manager)$ $R_{SSN} : (SSN)$ and $ind^\subseteq[Worker, R_{SSN}](manager, SSN)$. In the cases of transitive inclusion dependencies or sets of inclusion dependencies sharing the same right-hand side, additional patterns are necessary, some illustrated in Figure. 4.

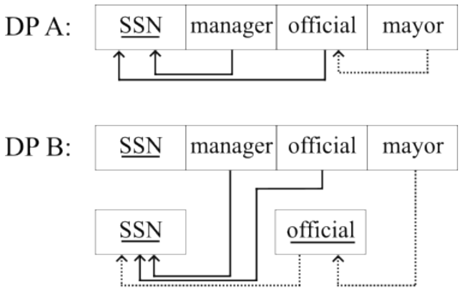


Figure 4: Instantiated Inclusion Dependency Extraction Pattern. Full line: regular case ; Dot line: transitive case

4.2. Vertical Decomposition

Vertical Decomposition (VD) is one of the core operations along with horizontal decompositions. Its principle is to project into a new table all attributes present in the join constraint and to drop its right-hand-side attributes from the source relation. This applies to either a functional dependency or a multivalued dependency and both have their kind of TPs, see Fig. 2 (b) for the latter. The TP defining the vertical decomposition of a relation $R : (REST, LHS, RHS)$, with regards to a join dependency we write as $\psi[R](LHS, RHS)$, always generate $R1 : (REST, LHS)$, $R2 : (LHS, RHS)$, a full inclusion between the tables $ind^{=}[R1, R2](LHS, LHS)$ and ends with this constraint $pk[R](LHS)$ or $pk[R](LHS, RHS)$ for functional dependency or multivalued dependency, respectively, see Fig. 5. This figure also depicts the transformation done to our ongoing example.

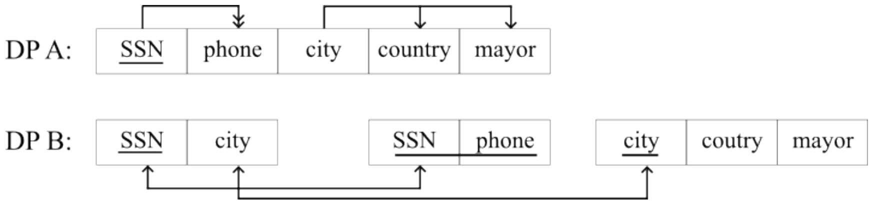


Figure 5: Instantiated Vertical Decomposition Pattern

In this scenario, no sequencing is required and both decomposition can be done independently from the other. Independent in this context means that a pair of constraints c_1, c_2 are in none of the three following combinations: shared RHS, transitive and critical.

Decomposing a set of overlapping constraints $c_1, c_2, ..., c_n$ of the form $c_i = \psi[R](LHS_i, RHS_i)$ such that $RHS_1 \wedge RHS_2 \wedge ... \wedge RHS_n \neq \varnothing$ require a TP which preserve the greatest common subset of attributes in the right-hand side at each step. To illustrate, the greatest common denominator for the set $c_1 = X \rightarrow T$, $c_2 = Y \rightarrow TUV$, $c_3 = Z \twoheadrightarrow TV$ in the relation $R : (X, Y, Z, T, U, V)$ where we choose to start by projecting c_1 is T, whereas for c_2 it would be TV. The decomposed schema for both scenarios are $R_1 : (X, Y, Z, T, U, V)$, $R_2 : (X, T)$ and $R'_1 : (X, Y, Z, T, V)$, $R'_2 : (Y, T, U, V)$ respectively.

Unlike the previous overlap, transitive overlaps do not require any new TPs. What they require however is an ordering of the decompositions. It is quite clear how, without any ordering, we lose information contained in $R : (X, Y, Z)$, $c_1 = X \to Y$, $c_2 = Y \to Z$ by decomposing c_1 first ending on $R_1 : (X, Z)$ $R_2 : (X, Y)$ thus losing c_2. The ordering we advocate is a relatively simple one in which, borrowing some tree terminology, all decompositions should start from the leaves then upward.

The last cases of overlap we have to detect are the critical ones. We borrow our definition of critical constraints from [13], denoting a functional dependency $X \to A$ is critical if and only if there exists in the closure of the set of constraints $Y \to B$ such that $YB \subseteq XA$ and $XA \nsubseteq \bar{Y}$ (the closure of an attribute is written $\bar{X}$). A common example is the pair $XY \to Z$, $Z \to X$. This is where our approach clashes with the literature on database normalization as we refuse to stop here and to lose information, something usually required to reach the BCNF [14]. What we propose to solve an overlap akin to $XY \to Z$, $Z \to X$ is to simply separate the problematic constraints into their own table, thus generating for any $R : (X, Y, Z, REST)$ the relations $R_1 : (X, Y, Z)$ and $R_2 : (X, Z)$. Because we do not pretend to present an exhaustive list of all possible types of constraint overlaps over join dependencies, we need a backup plan. That is, a solution which always guarantees both determinism and losslessness when used at the expense of redundancy. And the decomposition used to solve critical overlaps is that solution. We still consider this part of the methodology automatic under the assumption that we have a broader definition of overlaps. We now present a priority scale to follow whenever several types of overlaps, well, overlap: In order of priority, we start with dealing with critical constraints, then transitive ones and then ones that share an RHS.

4.3. Horizontal Decomposition

Going further than most database normalization methodologies we propose the use of TPs, such as the one presented in Fig. 2, to express Horizontal Decomposition (HD) with for intent to remove null values from a database schema. We already introduced guard constraint in our example section and now expend on its condition *cond*. That latter is a conjunction of atomic conditions $\bigwedge_i^n (ATT_i \neq \mathit{NULL})$. As long as all guard constraints are independent of each other, the decomposition methodology is trivial and deterministic. To verify if this property also extends to overlapping guard constraints, we first need to define them.

Definition 6 (Guard Overlaps). *In a relation $R : (ATT_n)$ onto which are applied two constraints $\sigma_{ATT_k=\mathit{NULL}} R \neq \emptyset$ and $\sigma_{ATT_l=\mathit{NULL}} R \neq \emptyset$ where, $ATT_k, ATT_l \subseteq ATT_n$. We distinguish two types of overlap:*

- ***Containment:*** $ATT_k \subset ATT_l$
- ***Non-Distinct:*** $ATT_k \wedge ATT_l \neq \emptyset$*,* $ATT_k \nsubseteq ATT_l$ *and* $ATT_l \nsubseteq ATT_k$

We illustrate these two cases with an example: there be $R : (X, Y, Z, T)$ and three overlapping guard constraints $c_1 = \sigma_{XY=\mathit{NULL}} R \neq \emptyset$, $c_2 = \sigma_{X=\mathit{NULL}} R \neq \emptyset$ and $c_3 = \sigma_{YT=\mathit{NULL}} R \neq \emptyset$. Both scenarios are shown with the pairs $\{c_1, c_2\}$ and $\{c_1, c_3\}$ clearly demonstrating the containment and non-distinct cases respectively. Focusing on the containment case, starting with an HD on c_1 will result in two relations $R_1 : (X, Y, Z, T)$ and $R_2 : (Z, T)$, one where no tuples satisfy $XY = NULL$ and one where all does. The constraint propagation formula we presented in Section 3 dictates that c_2 and c_3 would hold in both relations which is impossible for c_2. This is why we need two additional rules:

- **Guard Loss:** $\sigma_{cond_1} R \neq \emptyset$ do not get propagated in R' if R' contains one of the following constraints $\sigma_{cond_2} R = \emptyset$, $\sigma_{cond_2} R = R$ for $cond_2 \subset cond_1$ and $cond_1 \subset cond_2$ respectively.
- **Guard Intuition:** $\sigma_{cond_1} R \neq \emptyset$ get propagated in R' even if $\sigma_{cond_2} R = R$ holds, with $cond_1 \not\subset cond_2$ and $cond_1 \wedge cond_2 \neq \emptyset$.

The intuition behind the first rule is that a constraint can only get propagated if it is not directly contradicted by another. What the other rules allow is for nullable conditions to hold in a table where a subset of the attribute in said condition is non-existent. In our example, this permits c_3 to be propagated to R_2 even though this table is missing the Y column. Adding these rules to the one previously described leads to the following result:

Lemma 1. *All sequences of horizontal decompositions are lossless and their schema outputs are equivalents.*

Proof 1. *We want to show that for any binary relation $\mathbb{HD}(TP_1, TP_2)$ representing the sequential application of HD TPs $TP_1 \rightarrow TP_2$, then $\mathbb{HD}$ is symmetric. Losslessness is already guaranteed by the TP formalism, so only the deterministic aspect is touched. This property is trivial when it comes to independent guard constraints so we test it on both kinds of overlaps. Using the same notation used in the definition, in the containment case starting with a decomposition on the subsumed constraints gives $R_1 : (ATT_n)$ and $R_2 : (ATT_n \backslash ATT_k)$. The second decomposition here only affects R_2 because of the guard loss property, giving a final decomposition of R_1, R_2 and $R_3 : (ATT_n \backslash ATT_l)$. Starting from TP_2 would instead give us first $R'_1 : (ATT_n)$ and $R'_2 : (ATT_n \backslash ATT_l)$ and TP_1 here only affect R'_1 because of the guard loss property giving for final decomposition R'_1, R'_2 and $R'_3 : (ATT_n \backslash ATT_k)$. Both decompositions are the same, thus demonstrating the symmetric property of $\mathbb{HD}$ in this scenario. A similar logic is applied for non-distinct guard overlaps and the same result, that is $R_1 : (ATT_n)$, $R_2 : (ATT_n \backslash ATT_k)$, $R_3 : (ATT_n \backslash ATT_l)$, $R_4 : (ATT_n \backslash (ATT_k \cup ATT_l))$ is found in all sequences.* □

Because all possible scenarios of overlap over guard constraints can be represented, our methodology for their decomposition is complete. We can mention a specific case where one or more overlapping guard constraints exist and the union of their set of attributes $ATT_1 \cup ... \cup ATT_m = ATT_n$ is the size of the attribute set ATT_n in R. In this case, the relation satisfying all guard constraints would just be an empty table $R_\emptyset$ with m constraints of the form $\sigma_{ATT_i = NULL} R_\emptyset = R_\emptyset$. This is unnecessary and can be resolved by adding a last propagation rule:

- **Guard Total:** $\sigma_{cond_1} R \neq \emptyset$ with $cond_1$ over ATT_k gets propagated as $\sigma_{cond_1} R = \emptyset$ in R' if $ATT_n \subseteq ATT_k$.

4.4. Conditional Vertical Decomposition

A Conditional Function Dependency, defined in [15], is, within our formalism, a dependency $\psi[R](ATT_i)(\sigma_{cond})$ of type ψ which only gets applied over tuples that satisfy σ_{cond}. We return to our ongoing example, shorten this time as $Worker : (SSN, manager, title)$ onto which is applied a condition functional dependency $c1 = fd[Worker](manager, title)(manager, title \neq NULL)$. The functional dependency $manager \rightarrow title$ holds only whenever $manager$ and $title$ are both non-null. If only one CVD is applied we refer to the Fig. 6 instantiated with our recurring example. Otherwise, for any set of conditional dependencies, overlapping or not, we always start with all HDs followed by all VDs, respecting both methodologies for overlap along the way.

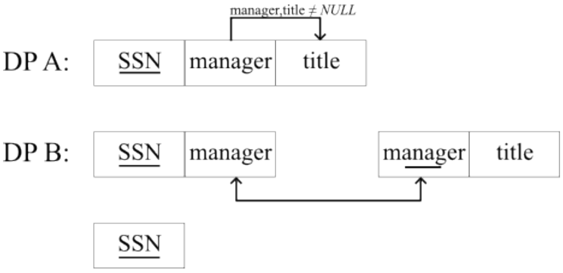


Figure 6: Conditional Vertical Decomposition Pattern

The final order is: IDE → VD → CVD → HD. What results is a pair of mapping patterns between an FOS schema and its equivalent 4NF decomposition, the latter shown in Fig. 7.

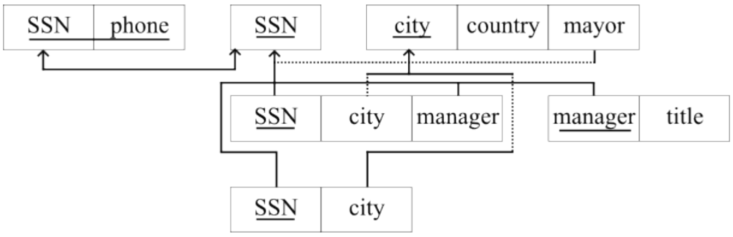


Figure 7: 4NF Decomposition of the Ongoing Example. Inclusion dependencies are drawn merged for clarity but they are all distinct

It is uncertain if a potential decomposition into 5NF and higher can be deterministic, meaning that any further process is likely to be semi-automatic. The end goal being a fully decomposed schema in 6NF which can easily be translated into a conceptual modeling language. Such process is prone to the arbitrary as is the nature of conceptual models. To illustrate, we argue for a new table $Person\_City : (SSN, city)$ to be added to our schema as it reflects better the relation between $Person$ and $City$ and for the drop of $City$ in the workers tables. Taking inspiration from conceptual modeling operations, such as attribute generalization in this case, to create new TPs is something I already encouraged in [16]. Another challenges is to properly draw out concepts from our database schema, an action fitting for TPs turning database schema into their ARM equivalents.

5. Related Work

With regards to the rest of the literature on DBRE [17], ignoring the aggregation of knowledge from different sources and the type of output, the closest method to ours would be [18]. This approach also emphasizes the preservation of information, but we believe that shifting to an ontological formalism limits the kind of operations we can apply over the source database, notably direct updates from any point in the framework. Several papers presented their own interpretation to Hull's information preservation criterion [3]: [8] brings up SIG, a formalism which represents via edges the relations between attributes as functions - [19] propose conflictfreeness as a restriction for the integration of two schemas - [20] put the focus on the transformation itself and how to embody them with instance preserving properties and - [21] which encapsulates how information preservation and bijective functions are related.

Throughout the literature provided so far, one common scenario outside of DBRE in which information preservation is desirable is in data integration [22]. Because most database transformations are written as mappings, ours included, it is quite intuitive to link them to the mappings used in data integration following either Global as View (GAV) or Local as View (LAV) paradigms. [23] notably proposed a framework based on sequences of simple operations to represent more complex transformations in a data integration context. Delving deeper into information preservation with regard to data integration can lead us to consider methodologies used in data exchange [24]. While the main focus of data exchange is the generation of a proper database instance from constraints, the problem of schema exchange introduced in [25] is closer to our ambitions. Mostly with regards to the grouping of similar schemas into templates making an automatic schema transformation possible.

6. Conclusion and Discussions

We presented a case study for the application of the transformation pattern formalism. Showing first how it can be applied to a database schema before asserting that it can serve as the base of a database decomposition process. Any database schema, once translated into FOS, can then be decomposed following a sequence of lossless transformations embodied by the TPs. What result is a pair of mapping patterns between the source database and its equivalent 4NF decomposition. The presented methodology places itself in the context of the SQL Semantic Transducer [26] in which the conceptual model of a database schema can act as its materialized view. Meaning that, once the jump from 6NF to any conceptual modeling language is done, queries and updates applied to the model would get propagated back to the source database. And so would the reverse, a property only guarantee because losslessness was preserved through the entire process. Upcoming works will therefore focus on transitions into both 6NF then to a conceptual model, likely going through an ARM schema in the process. Going further, we consider frameworks ensuring losslessness appealing to other database related fields such as data integration and data preparation. Both handle transformations where information loss is unavoidable, thus requiring both a definition of non-lossless, i.e. lossy, TPs and ways to enforce losslessness still, perhaps taking inspiration from the complementary view problem.

References

[1] M. Stonebraker, D. Deng, M. L. Brodie, Database decay and how to avoid it, in: 2016 IEEE International Conference on Big Data (Big Data), IEEE, 2016, pp. 7–16.

[2] N. Ndefo, E. Franconi, A Study on Information-Preserving Schema Transformations, International Journal of Semantic Computing 14 (2020) 27–53. URL: https://www.worldscientific.com/doi/abs/10.1142/S1793351X20400024. doi:10.1142/S1793351X20400024.

[3] R. Hull, Relative information capacity of simple relational database schemata, in: Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems, 1984, pp. 97–109.

[4] P. Atzeni, G. Ausiello, C. Batini, M. Moscarini, Inclusion and equivalence between relational database schemata, Theor. Comput. Sci. 19 (1982) 267–285.

[5] J. Albert, Y. Ioannidis, R. Ramakrishnan, Equivalence of Keyed Relational Schemas by Conjunctive Queries, Journal of Computer and System Sciences 58 (1999) 512–534. URL: https://linkinghub.elsevier.com/retrieve/pii/S0022000099916288. doi:10.1006/jcss.1999.1628.

[6] X. Qian, Correct schema transformations, in: G. Goos, J. Hartmanis, J. Van Leeuwen, P. Apers, M. Bouzeghoub, G. Gardarin (Eds.), Advances in Database Technology — EDBT '96, volume 1057, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 114–128. URL: http://link.springer.com/10.1007/BFb0014146. doi:10.1007/BFb0014146, series Title: Lecture Notes in Computer Science.

[7] W. McCune, Prover9 and mace4, 2005–2010. URL: http://www.cs.unm.edu/~mccune/prover9/.

[8] R. J. Miller, Y. E. Ioannidis, R. Ramakrishnan, The use of information capacity in schema integration and translation, in: VLDB, volume 93, 1993, pp. 120–133.

[9] A. Borgida, D. Toman, G. Weddell, On referring expressions in information systems derived from conceptual modelling, in: Conceptual Modeling: 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings 35, Springer, 2016, pp. 183–197.

[10] C. Beeri, P. A. Bernstein, N. Goodman, A sophisticate's introduction to database normalization theory, in: Readings in artificial intelligence and databases, Elsevier, 1989, pp. 468–479.

[11] A. H. Bahmani, M. Naghibzadeh, B. Bahmani, Automatic database normalization and primary key generation, in: 2008 Canadian Conference on Electrical and Computer Engineering, IEEE, Niagara Falls, ON, Canada, 2008, pp. 000011–000016. URL: http://ieeexplore.ieee.org/document/4564486/. doi:10.1109/CCECE.2008.4564486, iSSN: 0840-7789.

[12] D.-M. Tsou, P. C. Fischer, Decomposition of a relation scheme into boyce-codd normal form, ACM SIGACT News 14 (1982) 23–29.

[13] H. Koehler, Finding faithful boyce-codd normal form decompositions, in: Algorithmic Aspects in Information and Management: Second International Conference, AAIM 2006, Hong Kong, China, June 20-22, 2006. Proceedings 2, Springer, 2006, pp. 102–113.

[14] C. Beeri, P. A. Bernstein, Computational problems related to the design of normal form relational schemas, ACM Transactions on Database Systems 4 (1979) 30–59. URL: https://dl.acm.org/doi/10.1145/320064.320066. doi:10.1145/320064.320066.

[15] W. Fan, F. Geerts, X. Jia, A. Kementsietsidis, Conditional functional dependencies for capturing data inconsistencies, ACM Transactions on Database Systems 33 (2008) 1–48. URL: https://dl.acm.org/doi/10.1145/1366102.1366103. doi:10.1145/1366102.1366103.

[16] T. Abgrall, Formalization of Data Integration Transformations, in: S. Chiusano, T. Cerquitelli, R. Wrembel, K. Nørvåg, B. Catania, G. Vargas-Solar, E. Zumpano (Eds.), New Trends in Database and Information Systems, volume 1652, Springer International Publishing, Cham, 2022, pp. 615–622. URL: https://link.springer.com/10.1007/978-3-031-15743-1_56. doi:10.1007/978-3-031-15743-1_56, series Title: Communications in Computer and Information Science.

[17] N. A. Mian, S. A. Khan, N. A. Zafar, Database Reverse Engineering Methods: What is Missing? 2 (2013).

[18] L. Lubyte, S. Tessaris, Automatic Extraction of Ontologies Wrapping Relational Data Sources, in: S. S. Bhowmick, J. Küng, R. Wagner (Eds.), Database and Expert Systems Applications, volume 5690, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 128–142. URL: http://link.springer.com/10.1007/978-3-642-03573-9_10. doi:10.1007/978-3-642-03573-9_10, series Title: Lecture Notes in Computer Science.

[19] L. Ekenberg, P. Johannesson, Conflictfreeness as a basis for schema integration, in: International Conference on Information Systems and Management of Data, Springer, 1995, pp. 1–13.

[20] P. McBrien, A. Poulovassilis, A formalisation of semantic schema integration, Information Systems 23 (1998) 307–334.

[21] I. Kobayashi, Losslessness and semantic correctness of database schema transformation: Another look of schema equivalence, Information Systems 11 (1986) 41–59. URL: https://linkinghub.elsevier.com/retrieve/pii/0306437986900220. doi:10.1016/0306-4379(86)90022-0.

[22] A. Doan, A. Halevy, Z. Ives, Principles of data integration, Elsevier, 2012.

[23] P. McBrien, A. Poulovassilis, Data integration by bi-directional schema transformation rules, in: Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405), IEEE, Bangalore, India, 2003, pp. 227–238. URL: http://ieeexplore.ieee.org/document/1260795/. doi:10.1109/ICDE.2003.1260795.

[24] R. Fagin, P. G. Kolaitis, R. J. Miller, L. Popa, Data exchange: semantics and query answering, Theoretical Computer Science 336 (2005) 89–124. URL: https://linkinghub.elsevier.com/retrieve/pii/S030439750400725X. doi:10.1016/j.tcs.2004.10.033.

[25] P. Papotti, R. Torlone, Schema exchange: Generic mappings for transforming data and metadata, Data & Knowledge Engineering 68 (2009) 665–682. URL: https://linkinghub.elsevier.com/retrieve/pii/S0169023X09000184. doi:10.1016/j.datak.2009.02.005.

[26] T. Abgrall, E. Franconi, Understanding the sql semantic transducer, in: International Conference on Advanced Information Systems Engineering, Springer, 2024.