Dependency-Based Query/View Synchronization upon Schema Evolutions

Loredana Caruccio(✉), Giuseppe Polese, and Genoveffa Tortora

Department of Computer Science, University of Salerno,
via Giovanni Paolo II n.132, 84084 Fisciano, SA, Italy
{lcaruccio,gpolese,tortora}@unisa.it

Abstract. Query/view synchronization upon the evolution of a database schema is a critical problem that has drawn the attention of many researchers in the database community. It entails rewriting queries and views to make them continue work on the new schema version. Although several techniques have been proposed for this problem, many issues need yet to be tackled for evolutions concerning the deletion of schema constructs, hence yielding loss of information. In this paper, we propose a new methodology to rewrite queries and views whose definitions are based on information that have been lost during the schema evolution process. The methodology exploits (relaxed) functional dependencies to automatically rewrite queries and views trying to preserve their semantics.

Keywords: Schema evolution · Query rewriting
Functional dependency

1 Introduction

Query/view synchronization (QVS in the following) upon schema evolutions is an extremely critical problem, since it has been estimated that each schema evolution step might affect up to 70% of the queries and views operating on the old schema [12]. The QVS problem has been tackled in several studies, and many approaches and tools have been proposed to support DBAs during the schema evolution process (see [9] for a survey). However, the shortage of proper automated tools has not made the proposed solutions sufficiently practical [1, 11, 15], since they often require programmer's work to manually rewrite portions of affected queries and views. Moreover, there are many cases in which it is not possible to correctly synchronize the affected queries and views, especially when the evolution entails loss of information. In fact, all the query/view definitions based on the lost information cannot be rewritten in a simple way, so limiting the possibility to have an automatic synchronization process.

In general, approaches aiming to solve the QVS problem in presence of information loss either prescribe to remove queries and views that cannot be synchronized, or to block the evolution operations preventing the synchronization

C. Woo et al. (Eds.): ER 2018 Workshops, LNCS 11158, pp. 91–105, 2018.
https://doi.org/10.1007/978-3-030-01391-2_17

of some queries or views. Some other approaches prescribe to ask the DBA about the best choice to be made.

In this paper, we propose a methodology aiming to automatically rewrite queries and views relying on the schema constructs that have been deleted during the schema evolution. The methodology exploits semantic correlations of data expressed in terms of relaxed functional dependencies (RFDs) [7], for which automatic tools are now available to extract them from data [6,8]. In particular, the methodology tries to rewrite affected queries and views by seeking possible RFDs between the deleted information and the remaining ones, in a way to preserve or approximate the original semantics of queries and views.

The paper is organized as follows. In Sect. 2 we recall several basic concepts on the theory of Schema Evolution and on the theory of Functional Dependencies. Then, we characterize the whole query/view synchronization problem, and specifically the necessity to manage the loss of information in Sect. 3. In Sect. 4 we present our solution to automatically synchronize queries and views when a loss of information occurs. In Sect. 5 we validate the proposed methodology by presenting the results of several experiments conducted on two public datasets. Related works have been provided in Sect. 6. Finally, conclusions and future research directions are discussed in Sect. 7.

2 Background

In this section we will recall basic notions on the database theory underlying the proposed solution.

2.1 Schema Evolution

Schema evolutions might occur when a system is first released, due to bugs or incomplete functionalities, or to reflect changes in the real world, which might also entail the evolution of the underlying database. Problems related to the schema evolutions have been studied not only for databases, but also for Data Warehouses [14,24] and for Ontologies [23]. In this paper, we focus on the evolution of database schemas.

Definition 1 (Schema evolution). *Let S be a database schema, and $Inst(S)$ be the set of possible instances of S; an evolution of S is the result of one or more changes to the data structures, constraints, or any other artifact of S, entailing modifications to the system catalog.*

Changes might consist of *simple schema modifications*, such as addition, deletion, or renaming of an attribute, of a constraint, or of a relation, and/or *composed schema modifications*, such as join, partition, and decomposition [20]. In what follows, we denote with $S \rightarrow S'$ the evolution of the schema S into S', where S and S' are called schema versions.

Example 1. Let us consider the following database schema:

$$S : \mathsf{Doctor}(\underline{\mathsf{idDoctor}}, \mathsf{Name}, \mathsf{Specialization}, \mathsf{Role}, \mathsf{Experience}, \mathsf{Salary}, \mathsf{Level}, \mathsf{Tax}) \quad (1)$$

where the underlined attribute represents the primary key. The schema represents a table of doctors, containing name, specialization, role, experience, annual salary, contractual level, and annual tax. An evolution $S \rightarrow S'$ of this schema might concern the removal of the attribute Level, yielding the following new schema version:

$$S' : \mathsf{Doctor}(\underline{\mathsf{idDoctor}}, \mathsf{Name}, \mathsf{Specialization}, \mathsf{Role}, \mathsf{Experience}, \mathsf{Salary}, \mathsf{Tax}) \quad (2)$$

There are two basic strategies to specify a schema evolution $S \rightarrow S'$; one strategy describes the operation commands of the procedure to transform S into S', and another first specifies S', and then finds schema correspondences between S and S', which can be represented by means of *mappings*[1] [1–3,18,20,22]. Moreover, there are hybrid strategies mixing the characteristics of the two previous ones. As a consequence, we can have the following three types of schema evolution approaches: *operation-based*, *mapping-based*, and *hybrid*, respectively.

In general, *operation-based* approaches exploit the advantage of knowing a priori how the schema can evolve and the effects of each evolution. On the other hand, *mapping-based* approaches allow us to handle every type of modification [22], but without a complete view of the effects that a single modification will have on the schema.

The impact of schema modifications on the database instances can be characterized through the concept of *information capacity* [17]. The latter specifies whether the set $Inst(S')$ is equivalent to, extends, or reduces $Inst(S)$ [1]. For instance, when an attribute is dropped from a database schema, the corresponding information is lost, hence $Inst(S')$ reduces $Inst(S)$. In this case, we will denote by $NotInst(S, S')$ the set of all instances of S that do not have a corresponding instance in $Inst(S')$.

Schema construct deletions are among the most critical capacity reducing variations, since they entail loss of information that might irremediably invalidate some database components, like queries and application programs.

2.2 Relaxed Functional Dependencies

Let us recall some basic concepts of relational databases.

A relational database schema $\mathcal{R}$ is defined as a collection of relation schemas $(R_1, \ldots, R_n)$, where each R_i is defined over a fixed set of attributes $attr(R_i)$. Each attribute A_k has associated a domain $dom(A_k)$, which can be finite or infinite. A relation instance (or simply a relation) r_i of R_i is a set of tuples such that for each attribute $A_k \in attr(R_i)$, $t[A_k] \in dom(A_k)$, $\forall\ t \in r_i$, where $t[A_k]$

[1] A mapping m between two schemas S and S' is a set of assertions of the form $q_S \rightsquigarrow q_{S'}$, where q_S and $q_{S'}$ are queries over S and S', respectively, with the same set of distinct variables, and $\rightsquigarrow \in \{\subseteq, \supseteq, \equiv\}$.

denotes the projection of t onto A_k. A database instance r of $\mathcal{R}$ is a collection of relations $(r_1,\ldots,r_n)$, where r_i is a relation instance of R_i, for $i \in [1,n]$.

In the context of relational databases, data dependencies have been used to define data integrity constraints, aiming to improve the quality of database schemas and to reduce manipulation anomalies. There are several types of data dependencies, including functional, multivalued, and join dependencies. Among these, functional dependencies (FDs) are the most commonly known, mainly due to their use in database normalization processes. Since RFDs extend FDs, let us recall the definition of FD.

Definition 2 (Functional dependency). *A functional dependency (FD) φ, denoted by $X \rightarrow Y$, between two sets of attributes $X, Y \subseteq attr(\mathcal{R})$, specifies a constraint on the possible tuples that can form a relation instance r of $\mathcal{R}$: $X \rightarrow Y$ iff for every pair of tuples (t_1, t_2) in r, if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$. The two sets X and Y are also called Left-Hand-Side (LHS) and Right-Hand-Side (RHS), resp., of φ.*

RFDs extend FDs by relaxing some constraints of their definition. In particular, they might relax on the *attribute comparison* method, and on the fact that the dependency must be satisfied by the entire database.

Relaxing on the attribute comparison method means to adopt an approximate tuple comparison operator, say $\approx$, instead of the "equality" operator. In order to define the type of attribute comparison used within an RFD, we use the concept of *similarity constraint* [10].

Definition 3 (Similarity constraint). *Given an attribute A on a given domain $\mathbb{D}$, let $\phi[A] : \mathbb{D} \times \mathbb{D} \rightarrow \mathbb{R}^+$ be a function which evaluates the similarity between value pairs of A.*

As an example, ϕ can be defined in terms of a similarity metric $\approx$, like for instance the edit or Jaro distances [13], such that, given two values $a_1, a_2 \in A$, $a_1 \approx a_2$ is true if a_1 and a_2 are "close" enough w.r.t. a predefined threshold α. We denote the similarity constraint associated to an attribute A as $A_{\leq\alpha}$, which indicates that a pair of values (a_1, a_2) can be considered similar on A if and only if the $\phi[A](a_1, a_2) \leq \alpha$.

Definition 4 (Set of similarity constraints). *Given a set of attributes $X \subseteq attr(R)$ with $X = \{A_1, \ldots, A_k\}$, a set of similarity constraints, denoted as X_Φ, collects the similarity constraints $X_\Phi = \{A_{1\leq\alpha_1}, \ldots, A_{k\leq\alpha_k}\}$ associated to attributes of X.*

A dependency holding for "almost" all tuples or for a "subset" of them is said to relax on the extent [7]. In case of "almost" all tuples, a *coverage measure* should be specified to quantify the degree of satisfiability of the RFD. Whereas in case of "subset" (*constrained domains* in the following), conditions on the attribute domains should be specified to define the subset of tuples satisfying the RFD.

Definition 5 (Coverage measure). *Given a database instance r of $\mathcal{R}$, and an* FD $\varphi : X \rightarrow Y$, *a coverage measure Ψ on φ quantifies the satisfiability degree of φ on r, $\Psi : dom(X) \times dom(Y) \rightarrow \mathbb{R}^+$ measuring the amount of tuple pairs in r satisfying φ.*

As an example, the *confidence measure* introduced in [16] evaluates the maximum number of tuples $r_1 \subseteq r$ for which φ holds in r_1.

Several coverage measures can be used to define the satisfiability degree of an RFD, but usually they return a value normalized on the total number of tuple pairs $\binom{n}{2}$ for n cardinality of $\mathcal{R}$, so producing a value $v \in [0, 1]$. In the context of the canonical FDs, this measure evaluates to 1.

Definition 6 (Constrained domain). *Given a relation schema $\mathcal{R}$ with attributes $\{A_1, \ldots, A_k\}$ with attributes $\{A_1, \ldots, A_k\}$ of a given domain $\mathbb{D} = \mathbb{D}_1 \times \mathbb{D}_2 \times \cdots \times \mathbb{D}_k = dom(\mathcal{R})$, let c_i $\forall i = 1, dots, k$ be a condition on $\mathbb{D}_i$ the constrained domain $\mathbb{D}_c$ is defined as follows*

$$\mathbb{D}_c = \{t \in dom(\mathcal{R}) | \bigwedge_{i=1}^{k} c_i(t[A_i])\}.$$

Constrained domains enable the definition of "subsets" of tuples on which dependencies apply [5].

Then, a general definition of RFD can be given:

Definition 7 (Relaxed functional dependency). *Let us consider a relational schema $\mathcal{R}$. An* RFD ϱ *on $\mathcal{R}$ is denoted by*

$$\left[X_{\Phi_1} \xrightarrow{\Psi \geq \varepsilon} Y_{\Phi_2}\right]_{\mathbb{D}_c} \tag{3}$$

where

- $\mathbb{D}_c$ is the constrained domain that filters the tuples on which ϱ applies;
- $X, Y \subseteq attr(R)$, with $X \cap Y = \emptyset$;
- Φ_1 and Φ_2 are sets of similarity constraints on attributes X and Y, respectively;
- Ψ is a coverage measure defined on $\mathbb{D}_c$;
- ε is a threshold.

Given $r \subseteq \mathbb{D}_c$, a relation instance r on R *satisfies* the RFD ϱ, denoted by $r \models \varrho$, if and only if: $\forall$ $(t_1, t_2) \in r$, if Φ_1 is true for each constraint $A_{\leq \alpha} \in \Phi_1$, then *almost always* Φ_2 is true for each constraint $B_{\leq \beta} \in \Phi_2$. Here, *almost always* means that $\Psi(\pi_X(r), \pi_Y(r)) \geq \varepsilon$.

In other words, if $t_1[X]$ and $t_2[X]$ agree with the constraints specified by Φ_1, then $t_1[Y]$ and $t_2[Y]$ agree with the constraints specified by Φ_2 with a degree of certainty (measured by Ψ) greater than ε.

In the following we use RFDs having only one attribute on the RHS; this condition can always be reached by means of the usual transformations of FDs.

Example 2. As an example, for the database schema of Doctor shown in (1), it is likely to have the same Specialization for doctors having the same Name and PlaceOfBirth. An FD {Name,PlaceOfBirth} → Specialization might hold. However, the names, places and specializations might be stored by using different abbreviations. Thus, the following RFD might hold:

$$\{\mathsf{Name}_{\approx}, \mathsf{PlaceOfBirth}_{\approx}\} \longrightarrow \mathsf{Specialization}_{\approx}$$

where ≈ is the string similarity function. On the other hand, the few cases of homonyms for the doctors born in the same place have to be considered. For this reason, the previous RFD should also admit exceptions. This can be modeled by introducing a different coverage measure into the RFD, making it approximate:

$$\{\mathsf{Name}_{\approx}, \mathsf{PlaceOfBirth}_{\approx}\} \xrightarrow{\psi(Name,PlaceOfBirth,Specialization)\geq 0.98} \mathsf{Specialization}_{\approx}$$

3 Problem Description

Let us define the query/view synchronization problem.

Definition 8 (Query/View Synchronization). *Let Q be the set of queries and views defined on a database schema S; upon a schema evolution $S \rightarrow S'$, the QVS problem consists of finding a transformation τ of Q producing a set Q' of queries and views on S', such that Q' on S' preserves the semantics of Q on S. If such a transformation exists, we say that there exists a synchronization $Q \rightarrow Q'$, or even that Q' represents a legal rewrite of Q.*

Example 3. Let us consider the view on the database schema S of Example 1:

```
CREATE VIEW getDoctorByLevel AS
SELECT Name, Specialization
FROM Doctor
WHERE Level = "D"
```
(4)

It extracts the tuples getDoctorByLevel(Name, Specialization) for all the stored doctors with a contractual level equal to "D". Clearly, even such a simple view needs be rewritten upon the schema evolution defined in (2).

The preferred automated tools will be those that give the user the illusion of defining queries and views on an older version of the schema even though it has evolved [18].

Example 4. Let us consider the schema S of Example 1. If it evolves into the following schema:

```
DoctorData(idDoctor, Name, Specialization, Role, Experience)
DoctorEconomy(idDoctor, Salary, Level, Tax)
```
(5)

then it is possible to automatically synchronize all queries and views involving attributes of both DoctorData and DoctorEconomy, by introducing a *join* between the new two relations.

Unfortunately, it is not always possible to find a synchronization for all queries and views, especially when the modification concerns the deletion of schema constructs instantiated within some queries or views. In the following, we analyze the schema constructs that can affect the results of a query or of a view.

Definition 9. *Given a relation R and a selection condition c $Inst(R, c)$ is the set of instances in R satisfying c [28].*

Definition 10. *Given a schema S, and a query/view Q with selection condition C_Q*

$$s_Q = \{s \in Inst(S, C_Q)\} \tag{6}$$

is the result instance of the query/view Q.

Definition 11. *Let Q be a query/view, and $S \rightarrow S'$ an evolution of S, we say that Q can be automatically rewritten upon the deletion of a schema construct if and only if the following properties hold:*

- *There exists an instance $s_i \in Inst(S')$, which corresponds to (is the same of) the result instance of Q when applied on S (result data preserving), i.e. $s_i = s_Q$.*
- *For each selection condition c of Q, there exists a constraint c' such that $s_Q \in Inst(S, c) \cap Inst(S', c')$ (result construction preserving).*

It is worth to noticing that the lack of preservation in result construction does not necessarily produce the lack of preservation in result data. In fact, when $s_Q \notin NotInst(S, S')$, there can exist a condition c_j such that $s_Q \notin Inst(S, c_j) \cap Inst(S', c'_j)$ for each c'_j definable on S'. In other words, this case occurs when the information on the attributes whose values must be outputted by executing Q have not been lost, but there is a condition useful to produce the result instance of Q that cannot be redefined in S'.

4 Methodology

In the literature, there are few methodologies addressing the synchronization of queries and views upon schema evolutions yielding information loss [9]. They propose solutions ranging from the possibility of not including all the information required by Q in the result instance [1], to the possibility of defining a priori some parameters and/or policies on Q that block evolutions yielding loss of information declared as essential for Q [19]. In some other cases, the DBA intervention is required to manage the loss of information [25], or to replace lost data with approximated ones within the result instance [26].

The proposed methodology exploits RFDs [7], aiming to produce result instances equivalent to or approximating those produced on the old schema version. In particular, we focus on schema evolutions yielding attribute deletions, since they can invalidate conditions of queries and views. However, the proposed methodology can be easily extended to the removal of relations, since they can be rewritten in terms of removal of several attributes.

More formally, given a schema evolution $S \rightarrow S'$, where $NotInst(S, S') \neq \emptyset$. Let $c_1, \ldots, c_k$ in Q, and let $\bar{c}_1, \ldots, \bar{c}_h$ be the conditions which involve attribute deleted by the evolution $S \rightarrow S'$. For each $c \in \{\bar{c}_1, \ldots, \bar{c}_h\}$ we construct the modified condition c' such that $s'_Q \in Inst(S', c')$ and $s'_Q = s_Q$, where $s_Q \in Inst(S, c)$.

In order to construct c', we verify the existence of some RFDs ψ of the form:

$$X_{\Phi_1} \rightarrow A_{EQ} \tag{7}$$

where A represents the attribute instantiated in c, and EQ is the equality constraint.

In this way, according to the type of each query/view condition c corrupting Q, it is possible to transform c into c' through the following general formula:

$$c' = \bigvee_{i=1}^{k} (X_1 = x_{1i} \wedge X_2 = x_{2i} \wedge \cdots \wedge X_n = x_{ni}) \tag{8}$$

$\forall X_j \in X$, with $j = 1, \ldots, n$, and $\forall t_i$ such that $t_i[A]\theta y$, where y is a *constant* or an *attribute*, and θ represents one of the possible operators that can be used in c.

In this way, each query/view Q including the condition c can be rewritten into Q' by replacing c with c'.

Example 5. Let us consider the database instance in Table 1, and the view Q in (3).

After the evolution $S \rightarrow S'$ defined in (2), if the following RFD φ $\{\mathsf{Role}_{\approx}, \mathsf{Experience}_{\approx}\} \rightarrow \mathsf{Level}_{EQ}$ holds, then we can automatically transform Q in Q' in the following way:

```
CREATE VIEW getDoctorByLevel AS
SELECT Name, Specialization
FROM Doctor
WHERE (Role = "Specialized" AND Experience = 2 years)
      OR (Role = "SpecializedDr." AND Experience = 3 years)
```
(9)

producing the same result instance of the original view.

The transformation rule defined in (8) can be specialized based on the condition used in the query/view (see Table 2).

In general, the proposed solution can be applied whenever an RFD is included within the previously defined general form. Moreover, the whole methodology

Table 1. A portion of a doctor database instance.

idDoctor	Name	Specialization	Role	Experience	Salary	Level	Tax
1	George Johnson	Neurology	Junior Dr.	2 years	$118,000	E	$27,140
2	Joe House	Cardiology	Head Physician	10 years	$314,000	B	$94,200
3	Derek Williams	Pediatrician	Specialized Dr.	2 years	$156,000	D	$39,000
4	Henry Jones	Neurology	Specialized	3 years	$158,000	D	$39,500
5	Victor Sanchez	Radiology	Senior Surgeon	5 years	$225,000	C	$63,000
...							

Table 2. The transformation rule (8) specialized for each type of query condition.

Condition Type	**Formula**
attribute = constant $(A = 'a')$	$\bigvee_i (X_1 = x_{1i} \wedge X_2 = x_{2i} \wedge \cdots \wedge X_n = x_{ni})$ $\forall t_i$ s.t. $t_i[A] = 'a'$, and $\forall X_j \in X$ with $j = 1, \ldots, n$
attribute = attribute $(A = B)$	$\bigvee_i (X_1 = x_{1i} \wedge X_2 = x_{2i} \wedge \cdots \wedge X_n = x_{ni})$ $\forall t_i$ s.t. $t_i[A] = t_i[B]$, and $\forall X_j \in X$ with $j = 1, \ldots, n$
attribute θ constant $(A\ \theta\ 'a')$	$\bigvee_i (X_1 = x_{1i} \wedge X_2 = x_{2i} \wedge \cdots \wedge X_n = x_{ni})$ $\forall t_i$ s.t. $t_i[A]\ \theta\ 'a'$, and $\forall X_j \in X$ with $j = 1, \ldots, n$
attribute θ attribute $(A\ \theta\ B)$	$\bigvee_i (X_1 = x_{1i} \wedge X_2 = x_{2i} \wedge \cdots \wedge X_n = x_{ni})$ $\forall t_i$ s.t. $t_i[A]\ \theta\ t_i[B]$, and $\forall X_j \in X$ with $j = 1, \ldots, n$
attribute *in* Query $(A\ in\ Q)$	$\bigvee_i (X_1 = x_{1i} \wedge X_2 = x_{2i} \wedge \cdots \wedge X_n = x_{ni})$ $\forall t_i$ s.t. $t_i[A] = (v_1 \vee, \ldots, \vee v_k)\ \forall v_z \in \{v_1, \ldots, v_k\}$ result value of Q, and $\forall X_j \in X$ with $j = 1, \ldots, n$
attribute *not in* Query $(A\ not\ in\ Q)$	$\bigvee_i (X_1 = x_{1i} \wedge X_2 = x_{2i} \wedge \cdots \wedge X_n = x_{ni})$ $\forall t_i$ s.t. $t_i[A] <> (v_1 \wedge, \ldots, \wedge v_k)\ \forall v_z \in \{v_1, \ldots, v_k\}$ result value of Q, and $\forall X_j \in X$ with $j = 1, \ldots, n$
attribute θ *any* Query $(A\ \theta\ any\ Q)$	$\bigvee_i (X_1 = x_{1i} \wedge X_2 = x_{2i} \wedge \cdots \wedge X_n = x_{ni})$ $\forall t_i$ s.t. $t_i[A]\ \theta\ (v_1 \vee, \ldots, \vee v_k)\ \forall v_z \in \{v_1, \ldots, v_k\}$ result value of Q, and $\forall X_j \in X$ with $j = 1, \ldots, n$
attribute θ *all* Query $(A\ \theta\ all\ Q)$	$\bigvee_i (X_1 = x_{1i} \wedge X_2 = x_{2i} \wedge \cdots \wedge X_n = x_{ni})$ $\forall t_i$ s.t. $t_i[A]\ \theta\ (v_1 \wedge, \ldots, \wedge v_k)\ \forall v_z \in \{v_1, \ldots, v_k\}$ result value of Q, and $\forall X_j \in X$ with $j = 1, \ldots, n$
attribute *between* $'a_1'$ *and* $'a_2'$ $(A\ between\ 'a_1'\ and\ 'a_2')$	$\bigvee_i (X_1 = x_{1i} \wedge X_2 = x_{2i} \wedge \cdots \wedge X_n = x_{ni})$ $\forall t_i$ s.t. $t_i[A] \geq 'a_1' \wedge t_i[A] \leq 'a_2'$, and $\forall X_j \in X$ with $j = 1, \ldots, n$

can be effectively used in practice, due to the fact that many RFDs can be automatically extracted from data [6,21].

Finally, it is fair to note that it is possible to find more than one correct rewritings of queries/views corrupted by an evolution. This is also another issue related to the general QVS problem. To this end, in order to choose the best candidate synchronization, we will select the query/view rewrite maximizing the length of c' in term of $\vee$ sentences (i.e. c' such that the value of k is minimum). In this way, more general rewritings of queries/views will be used.

4.1 Query/View Synchronization with Approximate Results

In case there is no RFD suitable to accomplish the synchronization process, it would be desirable to rewrite queries/views in a way to produce results approximating those of the original query/view version. This can be done due to approximate nature of RFDs.

More formally, in order to transform c into c', so that $s'_Q \in Inst(S', c')$ and $s'_Q \approx s_Q$, we verify the existence of some RFD φ of the form:

$$X_{\Phi_1} \rightarrow A_{\Phi_2} \tag{10}$$

where A represents an attribute instantiated in c that has been deleted during the schema evolution, with Φ_2 a similarity constraint.

Example 6. Let us consider again the database instance in Table 1, and the view Qin (3). After the evolution $S \rightarrow S'$ defined in (2), if the RFD φ $\mathsf{Experience}_{\approx} \rightarrow \mathsf{Level}_{\approx}$ holds, then we can automatically transform Q in Q' in the following way:

```
CREATE VIEW getDoctorByLevel AS
SELECT Name, Specialization
FROM Doctor
WHERE Experience = 2 years OR Experience = 3 years
```
(11)

producing a result that is not equal, but similar to the one of the original view, as shown in Table 3. It is worth to notice that a level of acceptance for the result approximations can be managed by domain experts, by restricting similarity constraint thresholds of valid RFDs.

Table 3. An approximate version of the view getDoctorByLevel.

idDoctor	Name	Specialization
1	George Johnson	Neurology
3	Derek Williams	Pediatrician
4	Henry Jones	Neurology

5 Evaluation

We made a prototype implementation of the proposed methodology in Java. When synchronizing queries affected by an evolution, the prototype considers all the RFDs that have one of the removed attributes on the RHS and none of the removed attributes on the LHS. The choice of the RFD to be used for the synchronization follows this rule:

Choice the RFD *with an LHS producing the minimum result set of a query constructed as* `SELECT LHS_Attributes FROM r WHERE affected_Condition`

where affected_Condition represents the condition instantiating one of the attributes removed during the evolution process, which corresponds to the RHS of the RFD used for the synchronization.

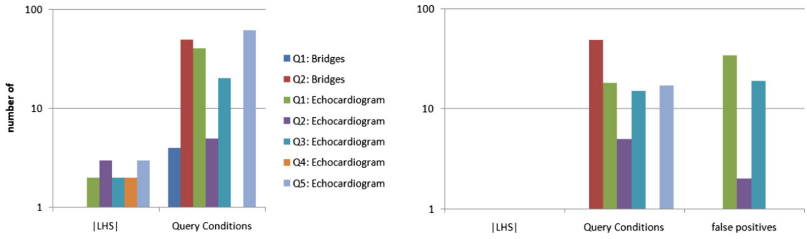
We evaluated the proposed methodology on two different datasets, the *Bridges* and the *Echocardiogram* datasets, drawn from the UC Irvine Machine Learning repository [4]. Statistics on the characteristics of the considered datasets are reported in Table 4.

In order to evaluate the proposed methodology, we defined several queries on each dataset, and randomly removed three attributes from each of them. In particular, there were two queries affected by the attribute removal for the *Bridges* and five for the *Echocardiogram* dataset. We observed the number of attributes on the LHS belonging to the RFD selected for the synchronization, and the growth of the query conditions. In general, the execution time to accomplish the complete synchronization was very small, varying in the range $[0.2, 0.5]$ seconds. Finally, we performed another experimental session, by forcing the selection of RFDs considering approximate matches on the RHS. In this case, we also analyzed the number of false positives introduced during the synchronization process.

Table 4. Statistics on the datasets considered in the evaluation.

Datasets	# Columns	# Rows	# FD	Size [KB]
Bridges	13	108	142	6
Echocardiogram	13	132	538	6

Evaluation results are shown in Fig. 1. We can notice that the growth of query conditions (number of conditions into the synchronized query) does not depend on the LHS cardinality (|LHS|, number of attributes) of the selected RFD, as shown in Fig. 1(a). However, since the conditions are automatically processed, their size does not affect the human effort in the synchronization process. Moreover, in the case of rewritings with approximate results (Fig. 1(b)), we notice that by using RFDs with similarity constraints on the RHS yields a reduced growth of the query condition, and the LHS cardinalities does not increase. However, as expected, in some cases several false positives are generated. Finally, it is worth to notice that a synchronization process accomplished through an RFD with a higher LHS cardinality increases the risk of possible future synchronizations, since it is highest the probability that one of them could be involved in future schema evolutions. This represents the main limitation of the proposed approach.



(a) Exact Rewritings (b) Rewritings with Approximate results

Fig. 1. Evaluation results of the proposed methodology.

6 Related Work

QVS approaches and tools defined in the literature are based on one of the three schema evolution strategies defined in Sect. 2.1: *operation-based*, *mapping-based*, and *hybrid*. Moreover, the QVS problem raises several issues, ranging from the automation level of the synchronization process to the management of information loss [9]. QVS approaches should provide solutions for all the issues related to the general problem. However, in some cases, solutions useful for some issues might be not appropriate for other cases.

In general, the QVS process permits the synchronization of all the queries/views affected by the evolution of a given schema. In the previous sections we discussed the fact that corrupted queries/views cannot be always synchronized. To this end, the concept of policies guiding the synchronization process can be used [25]. In particular, three types of policies have been defined in the literature: (i) *propagate*, (ii) *block*, and (iii) *prompt*, which prescribe how to handle the portions of the view definitions affected by the schema modification: *propagate* prescribes to apply changes and synchronize Q, *block* forbids changes, and *prompt* prescribes to ask the DBA for the action to be undertaken [11]. Although the latter might appear the most suitable policy, it should not be abused in order to keep the automation level of the QVS process sufficiently high.

Another solution to the QVS problem relies on the concept of *View Evolution Parameters (VEPs)* [29], which enables the possibility to define how to handle the single components of a view during the synchronization process, by specifying a priori whether the component (e.g. an attribute) is replaceable, or whether it is mandatory. In addition, the *View Extent*[2] *Parameter (VE)* [29] associated to a view Q specifies a condition on the extent of a view Q' in order for Q' to be considered an acceptable synchronization of Q. In other words, the VE parameter $\phi \in \{\equiv, \subseteq, \supseteq, \approx\}$, specifies a priori whether the extent of Q' must be equivalent ($\equiv$), be included ($\subseteq$), include ($\supseteq$), or approximate ($\approx$) the extent of Q, in order for Q' to be considered a legal rewrite of Q. In other words, the

[2] The view extent is the usually adopted term indicating the result-set of a view statement, i.e. the materialized view.

VE parameter establishes a relationship that must hold between the projections of Q' and Q on their common attributes.

Fault tolerance is the idea underlying the approach [27], in which attempts are made to recover data from the source schema, even with some errors. This has been done by means of default mappings, a formalism based on default logic, which is suitable to express rules allowing exceptions. This solution permits to define approximate versions of queries/views invalidated from the evolution.

Finally, a cooperative approach to querying has been used in the context of logic-based data integration of heterogeneous databases [18]. In particular, this solution exploits heuristics to produce approximate answers, which are submitted to the user or to the DBA for approval. Such a process exploits a dialogue for information seeking, in which participants (user and system) aim at finding an adequate mapping between the query and the modified schema.

All of approaches previously mentioned aim to provide a general solution to the QVS problem. However, they do not propose a specific solution w.r.t. the fact that queries/views might be corrupted even when their result instances can be recovered upon the evolution, and the latter only affects the query/view selection conditions. The proposed methodology aims to isolate this kind of situations, by providing possible rewritings of queries/views based on semantic correlations among data, expressed in terms of RFDs. For this reason, the proposed solution not only can be embedded within more generic solutions, but should also permit to increase the automation level of the QVS process.

7 Conclusion and Future Work

We have proposed a new methodology to automatically rewrite queries/views corrupted as a consequence of schema evolutions causing loss of information on which they were defined. The transformation procedure exploits the semantic correlations among data provided by RFDs, in oder to derive a general transformation formula for the QVS problem. It can be included in every approach or tool aiming to solve the general QVS problem. Moreover, since RFDs can be automatically extracted [6,21], the proposed methodology allows to improve the automation level of the synchronization process.

In the future, we would like to investigate the effects of instance evolutions of previous query/view synchronization processes. In particular, the addition or deletion of some data might change the set of RFDs holding on the dataset, possibly altering parts of previous synchronization processes. Thus, we would like to investigate how to make the query/view synchronization methodology incremental w.r.t. instance updates. To this end, it might be useful to investigate the possibility of exploiting RFDs that relax on the extent [7], that is, RFDs holding only on a subset of the database.

References

1. Bernstein, P.A., Melnik, S.: Model management 2.0: manipulating richer mappings. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (COMAD), pp. 1–12. ACM (2007)
2. Bernstein, P.A., Rahm, E.: Data warehouse scenarios for model management. In: Laender, A.H.F., Liddle, S.W., Storey, V.C. (eds.) ER 2000. LNCS, vol. 1920, pp. 1–15. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45393-8_1
3. Bertino, E.: A view mechanism for object-oriented databases. In: Pirotte, A., Delobel, C., Gottlob, G. (eds.) EDBT 1992. LNCS, vol. 580, pp. 136–151. Springer, Heidelberg (1992). https://doi.org/10.1007/BFb0032428
4. Blake, C.L., Merz, C.J.: UCI repository of machine learning databases. http://archive.ics.uci.edu/ml/index.php. Accessed 3 Mar 2018
5. Bohannon, P., Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for data cleaning. In: 2007 IEEE 23rd International Conference on Data Engineering, pp. 746–755. IEEE (2007)
6. Caruccio, L., Deufemia, V., Polese, G.: On the discovery of relaxed functional dependencies. In: Proceedings of the 20th International Database Engineering & Applications Symposium (IDEAS), pp. 53–61 (2016)
7. Caruccio, L., Deufemia, V., Polese, G.: Relaxed functional dependencies - a survey of approaches. IEEE Trans. Knowl. Data Eng. **28**(1), 147–165 (2016)
8. Caruccio, L., Deufemia, V., Polese, G.: Evolutionary mining of relaxed dependencies from big data collections. In: Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics, WIMS 2017, p. 5 (2017)
9. Caruccio, L., Polese, G., Tortora, G.: Synchronization of queries and views upon schema evolutions: a survey. ACM Trans. Database Syst. (TODS) **41**(2), 9 (2016)
10. Cohen, W., Ravikumar, P., Fienberg, S.: A comparison of string metrics for matching names and records. In: KDD Workshop on Data Cleaning and Object Consolidation, vol. 3, pp. 73–78 (2003)
11. Curino, C.A., Moon, H.J., Zaniolo, C.: Graceful database schema evolution: the prism workbench. Proc. VLDB Endow. **1**(1), 761–772 (2008)
12. Curino, C.A., Tanca, L., Moon, H.J., Zaniolo, C.: Schema evolution in wikipedia: toward a web information system benchmark. In: Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS), pp. 323–332. Citeseer (2008)
13. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007)
14. Golfarelli, M., Lechtenbörger, J., Rizzi, S., Vossen, G.: Schema versioning in data warehouses: enabling cross-version querying via schema augmentation. Data Knowl. Eng. **59**(2), 435–459 (2006)
15. Hick, J.M., Hainaut, J.L.: Database application evolution: a transformational approach. Data Knowl. Eng. **59**(3), 534–558 (2006)
16. Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H.: TANE: an efficient algorithm for discovering functional and approximate dependencies. Comput. J. **42**(2), 100–111 (1999)
17. Hull, R.: Relative information capacity of simple relational database schemata. SIAM J. Comput. **15**(3), 856–886 (1986)
18. Lakshmanan, L.V.S., Sadri, F., Subramanian, I.N.: On the logical foundations of schema integration and evolution in heterogeneous database systems. In: Ceri, S., Tanaka, K., Tsur, S. (eds.) DOOD 1993. LNCS, vol. 760, pp. 81–100. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57530-8_6

19. Lee, A.J., Nica, A., Rundensteiner, E.A.: The EVE approach: view synchronization in dynamic distributed environments. IEEE Trans. Knowl. Data Eng. **14**(5), 931–954 (2002)
20. Lerner, B.S.: A model for compound type changes encountered in schema evolution. ACM Trans. Database Syst. (TODS) **25**(1), 83–127 (2000)
21. Liu, J., Li, J., Liu, C., Chen, Y.: Discover dependencies from data - a review. IEEE Trans. Knowl. Data Eng. **24**(2), 251–264 (2012)
22. Melnik, S.: Generic Model Management: Concepts and Algorithms. LNCS, vol. 2967. Springer, Heidelberg (2004). https://doi.org/10.1007/b97859
23. Noy, N.F., Klein, M.: Ontology evolution: not the same as schema evolution. Knowl. Inf. Syst. **6**(4), 428–440 (2004)
24. Oueslati, W., Akaichi, J.: A survey on data warehouse evolution. Int. J. Database Manag. Syst. (IJDMS) **2**(4), 11–24 (2010)
25. Papastefanatos, G., Vassiliadis, P., Simitsis, A., Vassiliou, Y.: Policy-regulated management of ETL evolution. In: Spaccapietra, S., Zimányi, E., Song, I.-Y. (eds.) Journal on Data Semantics XIII. LNCS, vol. 5530, pp. 147–177. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03098-7_6
26. Polese, G., Vacca, M.: A dialogue-based model for the query synchronization problem. In: IEEE 5th International Conference on Intelligent Computer Communication and Processing (ICCP) (2009)
27. Polese, G., Vacca, M.: Notes on view synchronization using default logic. In: Proceedings of 17th Italian Symposium on Advanced Database Systems (SEBD), pp. 253–260 (2009)
28. Poulovassilis, A., McBrien, P.: A general formal framework for schema transformation. Data Knowl. Eng. **28**(1), 47–71 (1998)
29. Rundensteiner, E.A., Lee, A.J., Nica, A.: On preserving views in evolving environments. In: Knowledge Representation meets DataBases (KRDB), CEUR Workshop Proceedings, vol. 8, pp. 13.11–13.11 (1997)