Maintenance of Queries Under Database Changes: A Unified Logic Based Approach

Elena V. Ravve(✉)

Ort Braude College, Karmiel, Israel
elena_v_m@hotmail.com

Abstract. This contribution deals with one single theme, the exploitation of logical reduction techniques in database theory. Two kinds of changes may be applied to databases: structural changes, known also as restructuring or schema evolution, and data changes. We present both of them in the terms of syntactically defined translation schemes.

At the same time, we have application programs, computing different queries on the database, which are oriented on some specific generation of the database. Systematically using the technique of translation scheme, we introduce the notion of Φ-sums and show how queries, expressible in extensions of First Order Logic (FOL) may be handled over different generations of the Φ-sums. Moreover, using the technique of translation scheme, we introduce the notions of an incremental view recomputations. We prove when queries expressible in extensions of FOL allow incremental view recomputations.

Our approach covers uniformly the cases we have encountered in the literature and can be applied to all existing query languages.

1 Introduction

Over time, databases undergo two kinds of changes: structural changes (i.e., changes in the schema), known also as restructuring or schema evolution, and data changes (i.e., insertion, deletions and modifications of tuples). Data changes are usually referred to as updates. In the same way, one may talk about selection queries or simply queries (non-modification queries) and updates.

Non-modification query usually must be answered lots of time. That is why, as a rule, such queries are maintained as auxiliary relations, called in the context of databases as *materialized views*. Non-materialized views are called *virtual views* and, as a rule, are not updatable. We consider only materialized views in the paper. Moreover, in this paper, we consider only relational databases.

In this paper, we are concentrated on two problems: handling queries under restructuring of databases and under database updates.

Handling Queries Under Restructuring of Databases: Database, during its life cycle, may be restructured several times. At the same time, we have several application programs, oriented on some specific generation of the database. The problem under investigation is:

M. Gyssens and G. Simari (Eds.): FoIKS 2016, LNCS 9616, pp. 191–208, 2016.
DOI: 10.1007/978-3-319-30024-5_11

Given: There are two different generations of the same database, g and $g+1$. There is an application, running on the g^{th} generation: Q_g.

Find: An application Q_{g+1}, running on the $(g+1)^{th}$ generation with the same results.

Let us consider a toy example. The g^{th} generation of the database contains only one relation P, while the $(g+1)^{th}$ generation contains two relations R and S, such that $P = (R \bowtie S)$. The application, running on the g^{th} generation Q_g is a simple modification query on P, which deletes tuples from P, according to some condition θ, expressed in terms of P. The set of deleted tuples is defined by $\bigtriangledown_\theta P$ rather than given by enumeration.

We have problems with this kind of rules like deletion over join. In fact: in $\bigtriangledown_\theta(R \bowtie S)$, we deal with formula θ that can be complicated. When we use the substitution of $(R \bowtie S)$ instead of P in θ, we receive a new formula in terms of R and S that contains a mix of attributes from both relations: R and S. In order to evaluate θ, we must first produce $(R \bowtie S)$ and then delete $\bigtriangledown_\theta$ from the join, while we are mostly interested to derive (if possible) from θ some formulae: $\theta_1^R, \ldots, \theta_i^R$ over R and $\theta_1^S, \ldots, \theta_j^S$ over S, which we will apply to R and S respectively in order to obtain the same desired result.

In logical notation, the formulae: $\theta_1^R, \ldots, \theta_i^R$ over R and $\theta_1^S, \ldots, \theta_j^S$ over S are Feferman-Vaught reduction sequences (or simply, reductions), cf. [19]. The sequences are sets of queries such that each such a query can be evaluated on the components: R and S. Next, from the local answers, and possibly some additional information, we compute the answer. In this paper, we generalize the notion of Feferman-Vaught reduction sequences to handling queries over Φ–sums.

Handling queries under database updates: Materialized views contain some derived portion of the database information and storing as new relations. In order to reflect the changes, made on the source relations, the views should be modified by adding or deletions tuples without total re-computation from the database.

Given: A materialized view and a database update.

Find: A set of view updates that uses the old content of the view and delete from and inserts in the view some set of tuples defined on the source database.

In the case of the incremental view maintenance, we try to find some effective way to refresh the content of the view by some updates on it. The updates should be derived from the update on the source database, without the total view re-computation. In many case, it permits to simplify the maintenance procedure.

Unfortunately, as a rule, the derived view contains only some small part of the database information, and it is just impossible to obtain the desired results as a map over only the view. Using extension of the logical machinery of syntactically defined *translations schemes*, first introduced in [29] and recently used in [21] in the context of the database theory, we give precise definition of *incremental view re-computation* and prove that every query expressible in several extensions of First Order Logic (FOL) allows the incremental view re-computation.

In general, this contribution deals with exploitation of logical reduction techniques in database theory. This approach unifies different aspects, related to both schema and data evolution in databases, into a single framework. It is assumed that the reader is familiar with database theory as presented in [1] and has logical background as described in [15].

The used logical reduction techniques come in the form of Feferman-Vaught reduction sequences and translation schemes, known also in model theory as interpretations. The interpretations give rise to two induced maps, translations and transductions. Transductions describe the induced transformation of database instances and the translations describe the induced transformations of queries. Translation schemes appear naturally in the context of databases: The first example is the vertical decomposition of a relation scheme into two relation schemes with overlapping attribute sets. Also the reconstruction of the original scheme of the vertical decomposition can be looked at as translation scheme. The same is true for horizontal decompositions and the definition of views. More surprisingly, also updates can be cast into this framework. Finally, translation schemes describe also the evolution of one database scheme over different generations of database designs.

The paper is structured in the following way. Section 2 presents short review of the related works. Section 3 provides the definitions and main results, related to syntactically defined translation schemes. Section 4 is dedicated to handling of queries under restructuring of databases. Section 5 is dedicated to handling of queries under database updates. Section 6 summarizes the paper.

2 Related Works

Maintaining dynamic databases, have a long history, cf. [9–11,13]. One of the most recent paper is [21], inspired by [31]. Like [21], we are also *"interested in some arbitrary but fixed query on a finite structure, which is subject to an ongoing sequence of local changes, and after each change the answer to the query should remain available."*

In [21], the local changes of the database were limited to elements, which are constantly inserted in and deleted from the database. We take the following verbatim from the Conclusion and future work section of [21]: *We think that it is interesting to consider updates that are induced by first-order formulae. On the one hand one can consider formulae which induce updates directly to the structure, i.e. consider updates that change all tuples with the property defined by the formula. On the other hand one can perform canonical updates to one structure and consider the changes that are induced on a first-order interpreted structure.* In this paper, we propose a unified logic based approach to maintenance of queries under database changes. We show how this approach works not only for FOL but also for different extensions of it, used in the database theory.

In [26], the incremental view maintenance problem was investigated from an algebraic perspective. The author constructed a ring of databases and used it as the foundation of the design of a query calculus that allowed to express powerful

aggregate queries. In this framework, a query language needed to be closed under computing an additive inverse (as a generalization of the union operation on relations to support insertions and deletions) and the join operation had to be distributive over this addition to support normalization, factorization, and the taking of deltas of queries.

Some propagation techniques for view updates may be found in [3]. In [24], the complexity of testing the correctness of an arbitrary update to a database view is analyzed, coming back to constant-complement approach of Bancilhon and Spyratos, cf. [5]. We must mention the recent exciting works [16–18,20,23], which use propagation techniques for view updates as well. However, no one of them considers the question in comparable generality. In fact, we do not need most of usually used additional assumptions. For example, we do not need the structures to be ordered. Moreover, we allow both restructuring of the database and insertion, deletion or set operations under the same logical framework. In addition, we do not restrict ourselves to the use of FOL but rather different its extensions.

3 Translation Schemes

In this section, we introduce the general framework for syntactically defined translation schemes in terms of databases. We assume that the reader is familiar with precise definitions of extensions of FOL, cf. [25]. The notion of abstract translation schemes comes back to Rabin, cf. [29]. The translation schemes are also known in model theory as interpretations, as described in particular in [25]. The definition is valid for a wide class of logics or query languages, including Datalog or Second Order Logic (SOL) as well as FOL, $MSOL$, TC, n-TC, LFP or n-LFP. However, we start from Relational Calculus in the form of FOL. Occasionally, we use Relational Algebra expressions when they are more convenient to readers.

We follow Codd's notations, cf. [7]. Database systems should present the user with tables called *relations* $(R_1, R_2, \ldots)$ and their columns are headed by *attributes* $(A_1, A_2, \ldots)$ for a relation is called the *schema* for that relation $(R_1[\bar{\mathcal{A}}], R_2[\bar{\mathcal{B}}], \ldots)$. The set of schema for the relations $(\mathbf{R}, \mathbf{S}, \ldots)$ is called a *relational database schema*, or just *database schema*. The row of a relation (t) are called *tuples*. A tuple has one *component* $(t[A_1], t[A_2], \ldots)$ for each attribute of the relation. We shall call a set of tuples for a given relation an *instance* $(I(R_1), I(R_2), \ldots)$ of that relation.

Definition 1 (Translation Schemes Φ). *Let* $\mathbf{R}$ *and* $\mathbf{S}$ *be two database schemes. Let* $\mathbf{S} = (S_1, \ldots, S_m)$ *and let* $\rho(S_i)$ *be the arity of* S_i*. Let* $\Phi = \langle \phi, \phi_1, \ldots, \phi_m \rangle$ *be FOL formulae over* $\mathbf{R}$*.* Φ *is* k–feasible for $\mathbf{S}$ over $\mathbf{R}$ *if* ϕ *has exactly* k *distinct free FOL variables and each* ϕ_i *has* $k\rho(S_i)$ *distinct free first order variables. Such a* $\Phi = \langle \phi, \phi_1, \ldots, \phi_m \rangle$ *is also called a* k–**R**–**S**–**translation scheme** *or, in short, a* translation scheme*, if the parameters are clear in the context.*

If $k = 1$ we speak of **scalar** *or* **non–vectorized** *translation schemes.*
If ϕ is a tautology, then the translation scheme is **non–relativized**.
Otherwise, ϕ defines **relativization** *of the new database domain.*

The formulae $\phi, \phi_1, \ldots, \phi_m$ can be thought of as queries. ϕ describes the new domain, and the ϕ_i's describe the new relations. Vectorization creates one attribute out of a finite sequence of attributes. The use of vectorized translation schemes in the context of databases is shown in particular in [2] and [27]. We shall discuss concrete examples after we have introduced the induced transformation of database instances.

A (partial) function Φ^* from $\mathbf{R}$ instances to $\mathbf{S}$ instances can be directly associated with a translation scheme Φ.

Definition 2 (Induced Map Φ^*). *Let $I(\mathbf{R})$ be a $\mathbf{R}$ instance and Φ be k–feasible for $\mathbf{S}$ over $\mathbf{R}$. The instance $I(\mathbf{S})_\Phi$ is defined as follows:*

1. *The universe of $I(\mathbf{S})_\Phi$ is the set $I(\mathbf{S})_\Phi = \{\bar{a} \in I(\mathbf{R})^k : I(\mathbf{R}) \models \phi(\bar{a})\}$.*
2. *The interpretation of S_i in $I(\mathbf{S})_\Phi$ is the set*

$$I(\mathbf{S})_\Phi(S_i) = \{\bar{a} \in I(\mathbf{S})_\Phi{}^{\rho(S_i)} : I(\mathbf{R}) \models (\phi_i(\bar{a}))\}.$$

   Note that $I(\mathbf{S})_\Phi$ is a $\mathbf{S}$ instance of cardinality at most $|\mathbf{R}|^k$.
3. *The partial function $\Phi^* : I(\mathbf{R}) \to I(\mathbf{S})$ is defined by $\Phi^*(I(\mathbf{R})) = I(\mathbf{S})_\Phi$. Note that $\Phi^*(I(\mathbf{R}))$ is defined iff $I(\mathbf{R}) \models \exists \bar{x} \phi$.*

Φ^* maps $\mathbf{R}$ instances into $\mathbf{S}$ instances, by computing the answers to the queries $\phi_1, \ldots, \phi_m$ over the domain of $\mathbf{R}$ specified by ϕ, see Fig. 1. The definition of Φ^* can be extended on the case of sub-sets of $\mathbf{R}$ instances in the regular way.

Next we want to describe the way formulae (query expressions) are transformed when we transform databases by Φ^*. For this a function $\Phi^\#$ from $\mathcal{L}_1$–formulae over $\mathbf{S}$ $\mathcal{L}_2$–formulae over $\mathbf{R}$ can be directly associated with a translation scheme Φ, see Fig. 1.

Definition 3 (Induced map $\Phi^\#$). *Let θ be a $\mathbf{S}$–formula and Φ be k–feasible for $\mathbf{S}$ over $\mathbf{R}$. The formula θ_Φ is defined inductively as follows:*

1. *For each $S_i \in \mathbf{S}$ and $\theta = S_i(x_1, \ldots, x_l)$ let $x_{j,h}$ be new variables with $j \leq l$ and $h \leq k$ and denote by $\bar{x}_j = \langle x_{j,1}, \ldots, x_{j,k} \rangle$. We make $\theta_\Phi = \phi_i(\bar{x}_1, \ldots, \bar{x}_l)$.*
2. *For the boolean connectives, the translation distributes, i.e. if $\theta = (\theta_1 \vee \theta_2)$ then $\theta_\Phi = (\theta_{1\Phi} \vee \theta_{2\Phi})$ and if $\theta = \neg\theta_1$ then $\theta_\Phi = \neg\theta_{1\Phi}$, and similarly for $\wedge$.*
3. *For the existential quantifier, we use relativization, i.e., if $\theta = \exists y \theta_1$, let $\bar{y} = \langle y_1, \ldots, y_k \rangle$ be new variables. We make $\theta_\Phi = \exists \bar{y}(\phi(\bar{y}) \wedge (\theta_1)_\Phi)$.*
4. *For infinitary logics: if $\theta = \bigwedge \Psi$ then $\theta_\Phi = \bigwedge \Psi_\Phi$.*
5. *For second order variables U of arity ℓ and $\bar{a}$ a vector of length ℓ of first order variables or constants we translate $V(\bar{a})$ by treating V like a relation symbol and put $\theta_\Phi = \exists V(\forall \bar{v}(V(\bar{v}) \to (\phi(\bar{v_1}) \wedge \ldots \phi(\bar{v_\ell}) \wedge (\theta_1)_\Phi)))$.*
6. *For LFP, if $\theta = n\text{-}LFP\bar{x}, \bar{y}, \bar{u}, \bar{v}\theta_1$ then $\theta_\Phi = nk\text{-}LFP\bar{x}, \bar{y}, \bar{u}, \bar{v}\theta_{1\Phi}$.*
7. *For TC: if $\theta = n\text{-}TC\bar{x}, \bar{y}, \bar{u}, \bar{v}\theta_1$ then $\theta_\Phi = nk\text{-}TC\bar{x}, \bar{y}, \bar{u}, \bar{v}\theta_{1\Phi}$.*

8. *The function* $\Phi^{\#} : \mathcal{L}_1$ *over* $\mathbf{S} \to \mathcal{L}_2$ *over* $\mathbf{R}$ *is defined by* $\Phi^{\#}(\theta) = \theta_{\Phi}$.
9. *For a set of* $\mathbf{S}$*–formulae* Σ *we define*

$$\Phi^{\#}(\Sigma) = \{\theta_{\Phi} : \theta \in \Sigma \text{ or } \theta = \forall \bar{y}(S_i \leftrightarrow S_i)\}$$

This is to avoid problems with Σ *containing only quantifier free formulae, as* $\Phi^{\#}(\Sigma)$ *need not be a set of tautologies even if* Σ *is. If* Σ *contains only quantifier free formulae, we can reflect effect of relativization.*

Observation 1. *1.* $\Phi^{\#}(\theta) \in FOL\ (SOL, TC, LFP)$ *if* $\theta \in FOL(SOL, TC, LFP)$, *even for vectorized* Φ.
2. $\Phi^{\#}(\theta) \in MSOL$ *provided* $\theta \in MSOL$, *but only for scalar* Φ.
3. $\Phi^{\#}(\theta) \in nk\text{-}TC(nk\text{-}LFP)$ *provided* $\theta \in n\text{-}TC(n\text{-}LFP)$ *and* Φ *is a* k*–feasible.*
4. $\Phi^{\#}(\theta) \in TC^{kn}(LFP^{kn}, L^{kn}_{\infty\omega})$ *provided* $\theta \in TC^{n}(LFP^{n}, L^{n}_{\infty\omega})$ *and* Φ *is a* k*–feasible.*

The following fundamental theorem is folklore and establishes the correctness of the translation, cf. [15]. Figure 1 illustrates the fundamental theorem.

Theorem 1. *Let* $\Phi = \langle \phi, \phi_1, \ldots, \phi_m \rangle$ *be a* k*–*$\mathbf{R}$*–*$\mathbf{S}$*–translation scheme,* $I(\mathbf{R})$ *be a* $\mathbf{R}$*-instance and* θ *be a* FOL*–formula over* $\mathbf{S}$. *Then* $I(\mathbf{R}) \models \Phi^{\#}(\theta)$ *iff* $\Phi^{*}(I(\mathbf{R})) \models \theta$.

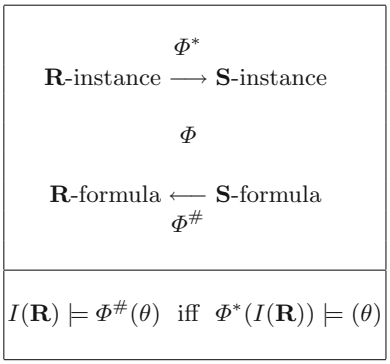

Fig. 1. Components of translation schemes the fundamental property

Now, we can define the composition of translation schemes:

Definition 4 (Composition of Translation Schemes). *Let* $\Psi = \langle \psi, \psi_1, \ldots, \psi_{m_1} \rangle$ *be a* k_1*–*$\mathbf{R}$*–*$\mathbf{S}$*–translation scheme, and let* $\Phi = \langle \phi, \phi_1, \ldots, \phi_{m_2} \rangle$ *be a* k_2*–*$\mathbf{S}$*–*$\mathbf{T}$*–translation scheme. Then we denote by* $\Psi \circ \Phi$ *the* $(k_1 \cdot k_2)$*–*$\mathbf{R}$*–*$\mathbf{T}$*–translation scheme given by* $\langle \Psi^{\#}(\phi), \Psi^{\#}(\phi_1), \ldots, \Psi^{\#}(\phi_{m_1}) \rangle$. $\Psi(\Phi)$ *is called the composition of* Φ *with* Ψ.

One can easily check that the syntactically defined composition of translation schemes has the following semantic property: $\Psi \circ \Phi(I(\mathbf{R})) = \Psi(\Phi(I(\mathbf{R})))$.

Now, we give a line of examples of translation schemes, relevant to the field of database theory. Assume that in all the examples, we a given database scheme $\mathbf{R} = (R_1, R_2, \ldots, R_n)$.

Example 1 (Restriction of the Domain). Assume we want to restrict the domain of $\mathbf{R}$ by allowing only elements, which satisfy some condition, defined by formula $\phi(x)$ in the chosen language (FOL, relation calculus, etc.). The corresponding translation scheme $\Phi_{Restriction}$ is:

$$\Phi_{Restriction} = \langle \phi, R_1, R_2, \ldots, R_n \rangle.$$

Example 2 (Deletion of a Definable set of Tuples from a Relation). Assume we want to delete from a relation, say, R_i of the $\mathbf{R}$ a set of tuples, which do not satisfy some condition, defined by formula θ. The corresponding translation scheme Φ_{DT} is:

$$\Phi_{DT} = \langle x \approx x, R_1, \ldots, R_{i-1}, R_i \wedge \neg\theta, R_{i+1}, \ldots, R_n \rangle.$$

Example 3 (Insertion of a Tuple into a Relation). Assume we want to insert a tuple into a relation, say, $R_i[A_1, \ldots, A_{k_i}]$ of the $\mathbf{R}$, where R_i contains k_i attributes. The corresponding translation scheme Φ_{InT} is a parametrized translation scheme with k_i parameters $a_1, \ldots, a_{k_i}$, which can be expressed, for example, in FOL in the following way:

$$\Phi_{InT} = \langle x \approx x, R_1, R_2, \ldots, R_{i-1}, (R_i \bigvee (\bigwedge_{1 \le j \le k_i} x_j \approx a_j)), R_{i+1}, \ldots, R_n \rangle.$$

Example 4 (Vertical Decomposition (Projections)). The *vertical decomposition*, given by a translation scheme in FOL notation, is:

$$\Phi_{VD} = \langle x \approx x, \phi_1, \ldots, \phi_n \rangle,$$

where each ϕ_i is of the form $\phi_i(\bar{x}_i) = \exists \bar{y}_i R_i(\bar{x}_i, \bar{y}_i)$. R_i is a relation symbol from $\mathbf{R}$ and $\bar{x}_i$ is a vector of free variables. In relational algebra notation, this amounts to $\phi_i(\bar{x}_i) = \pi_{\bar{x}_i} R_i$.

Example 5 (Vertical Composition (Join)). The *vertical composition*, given by a translation scheme in FOL notation, is:

$$\Phi_{VC} = \langle x \approx x, \phi_1, \ldots, \phi_n \rangle,$$

where each ψ_i is of the form $\phi_i(\bar{x}_i) = \bigwedge_{l=1}^{k} R_{i_l}(\bar{x}_{i_l})$, R_{i_l} is a relation symbol from $\mathbf{R}$ and $\bar{x}_i$ is a vector of free variables. Furthermore $\cup_j \bar{x}_{i_j} = \bar{x}_i$ and for all $\bar{x}_{i_{j_1}}$ there is $\bar{x}_{i_{j_2}}$ such that $\bar{x}_{i_{j_1}} \cap \bar{x}_{i_{j_2}} \neq \emptyset$. In relational algebra notation, this amounts to $\phi_i(\bar{x}_i) = \bowtie_{l=1}^{k} R_{i_l}$. If there are no common free variables, this just defines the Cartesian product.

Example 6 (Horizontal Decomposition (Exceptions)). Assume we want to decompose a relation, say, $R_i[A_1, \ldots, A_{k_i}]$ into two parts $R_i^1[A_1, \ldots, A_{k_i}]$ and $R_i^2[A_1, \ldots, A_{k_i}]$ such that all tuples of the first part satisfy some definable condition (formula) θ and all tuples of the second part do not. Such a transformation is called *horizontal decomposition of* R_i *along* θ and in FOL notation is:

$$\Phi_{HD} = \langle x \approx x, R_1, R_2, \ldots, R_{i-1}, R_i \wedge \theta, R_i \wedge \neg\theta, R_{i+1}, \ldots, R_n \rangle.$$

Example 7 (Horizontal Composition (Union)). Assume we want to compose a new relation, say, $R_{n+1}[A_1, \ldots, A_{k_{n+1}}]$ from two given relations $R_{i_1}[A_1, \ldots, A_{k_{n+1}}]$ and $R_{i_2}[A_1, \ldots, A_{k_{n+1}}]$. Such a transformation is called *horizontal composition of* R_{n+1} and in FOL notation is:

$$\Phi_{HC} = \langle x \approx x, R_1, R_2, \ldots, R_n, R_{i_1} \vee R_{i_2} \rangle.$$

The translation Φ_{HC} is called the *horizontal composition* (*union*) of R_{i_1} and R_{i_2}.

Example 8 (Definition of a View). Assume we are given a database scheme that contains four relations: $\mathbf{R} = (R_1, R_2, R_3, R_4)$. Assume that we want to define a view of a snapshot that is derived from the database by applying the following query, given in the format of relational algebra: $\phi_{View} = (\pi_A R_1 \cup R_2) \bowtie (R_3 - \sigma_\xi R_4)$. In this case, the corresponding translation scheme is:

$$\Phi_{View} = \langle x = x, \phi_{View} \rangle.$$

4 Handling Queries Under Restructuring of Databases

In terms of translation schemes, the problem of handling queries under restructuring of databases may be paraphrased in the following way, see Fig. 2:

Given: Two different generations of the same database, say, $\mathbf{R}^g$ and $\mathbf{R}^{g+1}$. Additionally, we have two maps: Φ_g and Ψ_g, where Φ_g produces $\mathbf{R}^{g+1}$ from $\mathbf{R}^g$ and Ψ_g is the corresponding reconstruction map. Finally, there is an application (translation scheme) Φ_g^{app} on the g^{th} generation.

Find: An application (translation scheme) Φ_{g+1}^{app} on the $(g+1)^{th}$ generation, such that: $\Phi_{g+1}^{app*}(\Phi_g^*(\mathbf{R}^g)) = \Phi_g^{app*}(\mathbf{R}^g)$.

Example 9. Assume that we are given database scheme $\mathbf{R^g} = (R^g)$ and two restructurings, defined by the following pair of translation schemes:

1. $\mathbf{R^{g+1}} = (R_1^{g+1}, R_2^{g+1})$, $\Psi_g = (\psi^g)$ and $\psi^g = (R_1^{g+2} \bowtie R_2^{g+2})$.
2. $\mathbf{R^{g+2}} = (R_1^{g+2}, R_2^{g+2}, R_3^{g+2}, R_4^{g+2})$, $\Psi_{g+1} = (\psi_1^{g+1}, \psi_2^{g+1})$ and $\psi_1^{g+1} = (\pi_A R_1^{g+2} \cup R_2^{g+2})$, $\psi_2^{g+1} = (R_3^{g+2} - \sigma_\zeta R_4^{g+2})$.

Assume that Q_g is a simple modification query on $\mathbf{R}^g$, which deletes tuples from R^g, according to some condition θ, expressed in terms of R^g. The set of deleted tuples is defined by $\bigtriangledown_\theta R^g$ rather than given by enumeration. In such a

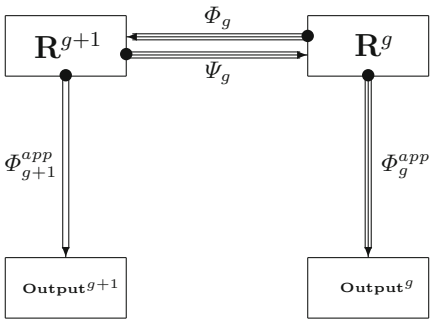

Fig. 2. Query on two different generations of database

case, we want to understand which tuples of which relations from $\mathbf{R}^{g+2}$ must be deleted, or moreover not only deleted, in order to produce the same output. Using substitutions, we obtain over $\mathbf{R}^{g+2}$:

$$\nabla_{\Psi^{\#}_{g+1}(\Psi^{\#}_g(\theta))}((\pi_A R_1^{g+2} \cup R_2^{g+2}) \bowtie (R_3^{g+2} - \sigma_\zeta R_4^{g+2})).$$

From Example 9, we observe that the derived set of tuples, defined by $\Psi^{\#}_{g+1}(\Psi^{\#}_g(\theta))$, seems to be already in terms of $\mathbf{R}^{g+2}$. However, the corresponding modification procedure can not be directly presented in terms of updates of **relations** from $\mathbf{R}^{g+2}$.

4.1 Handling of Queries over Disjoint Unions and Shufflings

The *Disjoint Union* (DJ) is the simplest example of juxtaposition, where none of the components are linked to each other. Assume we have a set of database schemes $\mathbf{R}_\imath$'s and we want to define a database scheme that represents their DJ. In this case, we add an, so called, *index scheme* $\mathbf{R}_I$, which specifies the parameters of the composition of the database schemes. The index scheme is a database scheme, whose instances are used in combining disjoint databases into a single database.

Definition 5 (Disjoint Union). *Let $\mathbf{R}_I$ be a database scheme chosen as an index scheme $\mathbf{R}_I = (R_1^I, \ldots, R_{\jmath^I}^I)$ with domain I and $\mathbf{R}_\imath = (R_1^\imath, \ldots, R_{\jmath^\imath}^\imath)$ be a database scheme with domain $D_\imath$. In the general case, the resulting database scheme $\mathbf{R}=\bigsqcup_{\imath\in I}\mathbf{R}_\imath$ with the domain $I \cup \dot{\bigcup}_{\imath\in I} D_\imath$ will be*

$$\mathbf{R} = (P(\imath, x), Index(x), R_j^I(1 \le j \le \jmath^I), R_{j^\imath}^\imath(\imath \in I, 1 \le j^\imath \le \jmath^\imath)) \text{ for all } \imath \in I,$$

- *the instance of $P(\imath, x)$ in $\mathbf{R}$ contains a tuple $(\imath, x)$ iff x came from $R_\imath$;*
- *the instance of $Index(x)$ in $\mathbf{R}$ contains x iff x came from I;*
- *$R_j^I(1 \le j \le \jmath^I)$ are from $\mathbf{R}_I$ and*
- *$R_{j^\imath}^\imath(\imath \in I, 1 \le j^\imath \le \jmath^\imath)$ are from $\mathbf{R}_\imath$*

Now, we give the classical theorem for the DJ, cf. [19,22].

Theorem 2 (Feferman-Vaught-Gurevich). *Let $\mathbf{R}_I$ be an index scheme with domain of size k and let $\mathbf{R}=\bigsqcup_{i\in I}\mathbf{R}_i$. For any FOL formula φ over $\mathbf{R}$ there are:*

1. *formulae of FOL $\psi_{1,1},\dots,\psi_{1,j_1},\dots,\psi_{k,1},\dots,\psi_{k,j_k}$*
2. *a formula of MSOL ψ_I*
3. *a boolean function $F_\varphi(b_{1,1},\dots,b_{1,j_1},\dots,b_{k,1},\dots,b_{k,j_k},b_I)$*

with the formulae in 1-2 having the following property:

$$I(\mathbf{R}_i)\models\psi_{i,j} \text{ iff } b_{i,j}=1 \text{ , and } I(\mathbf{R}_I)\models\psi_I \text{ iff } b_I=1$$

and, for the boolean function of 3, we have

$$I(\mathbf{R})\models\varphi \text{ iff } F_\varphi(b_{1,1},\dots,b_{1,j_1},\dots,b_{k,1},\dots,b_{k,j_k},b_I)=1.$$

Note that we require that F_φ and the $\psi_{i,j}$'s depend only on φ, k and $\mathbf{R}_1,\dots,\mathbf{R}_k$ but not on the instances involved.

For the case of the DJ, we assume that domains of databases in each site are disjoint. However, as a rule, the values of certain attributes may appear at several sites. We can assume that the domain of the index scheme is fixed and known, however we can not (without additional assumption) fix finite number of one place predicates. This puts the main limitation on the use of Theorem 2. Moreover, even if $\phi_=$ exists for some fixed database instance, it must be independent upon the current content of database and must be formulated ahead syntactically. In addition, it must be relatively small, as otherwise it causes explosion in size of other formulae. Now, we apply logical machinery.

Definition 6 (Partitioned Index Structure). *Let $\mathcal{I}$ be an index structure over τ_{ind}. $\mathcal{I}$ is called* finitely partitioned *into ℓ parts if there are unary predicates I_α, $\alpha<\ell$, in the vocabulary τ_{ind} of $\mathcal{I}$ such that their interpretation forms a partition of the universe of $\mathcal{I}$.*

In addition to the DJ, one may produce a new structure by shuffling.

Definition 7 (Shuffle over Partitioned Index Structure). *Let $\mathcal{A}_i, i\in I$ be a family of structures such that for each $i\in I_\alpha$: $\mathcal{A}_i\cong\mathcal{B}_\alpha$. In this case, we say that $\biguplus^{\mathcal{I}}_{\alpha<\beta}\mathcal{A}_\alpha$ is the* shuffle of $\mathcal{B}_\alpha$ along the partitioned index structure $\mathcal{I}$.

We generalize Theorem 2 by introducing abstract preservation properties in the following way:

Definition 8 (Preservation Properties with Fixed Index Set). *For two logics $\mathcal{L}_1$ and $\mathcal{L}_2$ we define Preservation Property for Disjoint Union*

Input of operation: *Indexed set of structures;*
Preservation Property: *if for each $i\in I$ (index set) $\mathcal{A}_i$ and $\mathcal{B}_i$ satisfy the same sentences of $\mathcal{L}_1$ then the disjoint unions $\bigsqcup_{i\in I}\mathcal{A}_i$ and $\bigsqcup_{i\in I}\mathcal{B}_i$ satisfy the same sentences of $\mathcal{L}_2$.*
Notation: $DJ\text{-}PP(\mathcal{L}_1,\mathcal{L}_2)$

Definition 9 (Preservation Properties with Variable Index Structures). *For two logics $\mathcal{L}_1$ and $\mathcal{L}_2$ we define Preservation Properties for Shuffle*

Input of operation: *A family of structures $\mathcal{B}_\alpha : \alpha < \beta$ and a (finitely) partitioned index structure $\mathcal{I}$ with I_α a partition.*

Preservation Property: *Assume that for each $\alpha < \beta$ the pair of structures $\mathcal{A}_\alpha, \mathcal{B}_\alpha$ satisfy the same sentences of $\mathcal{L}_1$, and $\mathcal{I}, \mathcal{I}$ satisfy the same MSOL-sentences. Then the schuffles $\biguplus^{\mathcal{I}}_{\alpha<\beta} \mathcal{A}_\alpha$ and $\biguplus^{\mathcal{I}}_{\alpha<\beta} \mathcal{B}_\alpha$ satisfy the same sentences of $\mathcal{L}_2$.*

Notation: *Shu-PP($\mathcal{L}_1, \mathcal{L}_2$) (FShu-PP($\mathcal{L}_1, \mathcal{L}_2$))*

Now, we list which Preservation Properties hold for which logics.

Theorem 3. *Let $\mathcal{I}$ be an index structure and $\mathcal{L}$ be any of FOL, $FOL^{m,k}$, $L^{\omega}_{\omega_1,\omega}$, $L^{k}_{\omega_1,\omega}$, $MSOL^m$, MTC^m, $MLFP^m$, or $FOL[\mathbf{Q}]^{m,k}$ ($L_{\omega_1,\omega}[\mathbf{Q}]^k$) with unary generalized quantifiers. Then DJ-PP($\mathcal{L}, \mathcal{L}$) and FShu-PP($\mathcal{L}, \mathcal{L}$) hold. Note that this includes DJ-PP($FOL^{m,k}, FOL^{m,k}$) and FShu-PP($FOL^{m,k}, FOL^{m,k}$) with the same bounds for both arguments, and similarly for the other logics.*

Proof.

FOL **and** *$FOL^{m,k}$***:** The proofs for *FOL* and *MSOL* are classical, see in particular [6]. Extension for $FOL^{m,k}$ can be done directly from the proof for *FOL*.
MLFP **and** *$MLFP^m$***:** The proof for *MLFP* was given in [4].
*$L_{\omega_1,\omega}(\mathbf{Q})^k$***:** The proof was given in [8].
*MTC^m***:** The proof was given in [30].

Now, we recall that analyzing Example 9, we decided that we are interested to derive from θ of $\bigtriangledown_{\Psi^{\#}_{g+1}(\Psi^{\#}_g(\theta))}((\pi_A R_1^{g+2} \cup R_2^{g+2}) \bowtie (R_3^{g+2} - \sigma_\zeta R_4^{g+2}))$ some formulae: $\theta^R_1, \ldots, \theta^R_i$ over R and $\theta^S_1, \ldots, \theta^S_j$ over S, which we will apply to R and S respectively. Now, we formulate the requirement more formally:

Definition 10 (Reduction Sequence). *Let $\mathcal{I}$ be a finitely partitioned τ_{ind}-index structure and $\mathcal{L}$ be logic.*

Let $\mathcal{A} = \biguplus^{\mathcal{I}}_{\alpha<\beta} \mathcal{B}_\alpha$ be the τ–structure which is the finite shuffle of the τ_α-structures $\mathcal{B}_\alpha$ over $\mathcal{I}$ or another combination of the components. A $\mathcal{L}_1$-reduction sequence for shuffling *for $\phi \in \mathcal{L}_2(\tau_{shuffle})$ is given by*

1. *a boolean function $F_\phi(b_{1,1}, \ldots, b_{1,j_1}, \ldots, b_{\beta,1}, \ldots, b_{\beta,j_\beta}, b_{I,1}, \ldots, b_{I,j_I})$*
2. *set Υ of $\mathcal{L}_1$–formulae $\Upsilon = \{\psi_{1,1}, \ldots, \psi_{1,j_1}, \ldots, \psi_{\beta,1}, \ldots, \psi_{\beta,j_\beta}\}$*
3. *MSOL–formulae $\psi_{I,1}, \ldots, \psi_{I,j_I}$*

and has the property that for every $\mathcal{A}$, $\mathcal{I}$ and $\mathcal{B}_\alpha$ as above with $\mathcal{B}_\alpha \models \psi_{\alpha,j}$ iff $b_{\alpha,j} = 1$ and $\mathcal{B}_I \models \psi_{I,j}$ iff $b_{I,j} = 1$ we have

$$\mathcal{A} \models \phi \text{ iff } F_\phi(b_{1,1}, \ldots, b_{1,j_1}, \ldots, b_{\beta,1}, \ldots, b_{\beta,j_\beta}, b_{I,1}, \ldots, b_{I,j_I}) = 1.$$

Note that we require that F_ϕ and the $\psi_{\alpha,j}$'s depend only on ϕ,β and $\tau_1, \ldots, \tau_\beta$ but not on the structures involved.

The following theorem partially answers the question of Example 9.

Theorem 4. *Let $\mathcal{L}$ be any of FOL, $FOL^{m,k}$, $L^{\omega}_{\omega_1,\omega}$, $L^{k}_{\omega_1,\omega}$ $MSOL^m$, MTC^m, $MLFP^m$, or $FOL[\mathbf{Q}]^{m,k}$ with unary generalized quantifiers. There is an algorithm, which for given $\mathcal{L}$, τ_{ind}, $\tau_\alpha, \alpha < \beta$, $\tau_{shuffle}$ and $\phi \in \mathcal{L}(\tau_{shuffle})$ produces a reduction sequence for ϕ for $(\tau_{ind}, \tau_{shuffle})$-shuffling. However, F_ϕ and the $\psi_{\alpha,j}$ are tower exponential in the quantifier rank of ϕ. Furthermore, F depends on the MSOL–theory of the index structure restricted to the same quantifier rank as ϕ.*

Proof. By analyzing the proof of Theorem 3.

Note that Theorem 4 is not true for all logics as shown in [30].

4.2 Handling Queries Over Φ–Sum

Combining Disjoint Unions and Shuffles with translation schemes, we can reach a very large set of useful structures. In this section, we present our new results in the field. We expend the classical Theorem 2 and more recent Theorems 3 and 4 to the cases, when translation schemes are involved in process of construction of the desired structure from the Disjoint Unions and Shuffles.

Definition 11 (Φ–Sum for extensions of *FOL*). *Let $\mathcal{I}$ be a finitely partitioned index structure and $\mathcal{L}$ be any of FOL, MSOL, MTC, MLFP, or FOL with unary generalized quantifiers. Let $\mathcal{A} = \bigsqcup_{i\in I}\mathcal{A}_i$ or $\mathcal{A} = \biguplus^{\mathcal{I}}_{\alpha<\beta}\mathcal{B}_\alpha$ be a τ–structure, where each $\mathcal{A}_i$ is isomorphic to some $\mathcal{B}_1,\dots,\mathcal{B}_\beta$ over the vocabularies $\tau_1,\dots,\tau_\beta$, in accordance with the partition.*

For a Φ be a scalar (non–vectorized) τ–σ $\mathcal{L}$–translation scheme, the Φ–sum of $\mathcal{B}_1,\dots,\mathcal{B}_\beta$ over I is the structure $\Phi^*(\mathcal{A})$, or rather any structure isomorphic to it.

Theorem 5. *Let $\mathbf{R}_I$ be a finitely partitioned index database scheme, $\mathcal{L}$ be any of FOL, MSOL, MTC, MLFP, MSOL or FOL with unary generalized quantifiers. Let $\mathbf{R}$ be the Φ–sum of $\mathbf{R}_{\mathcal{B}_1},\dots,\mathbf{R}_{\mathcal{B}_\beta}$ over I, as above. For every $\varphi \in \mathcal{L}(\tau)$ there are*

1. *a boolean function $F_{\Phi,\varphi}(b_{1,1},\dots,b_{1,j_1},\dots,b_{\beta,1},\dots,b_{\beta,j_\beta},b_{I,1},\dots,b_{I,j_I})$*
2. *$\mathcal{L}$–formulae $\psi_{1,1},\dots,\psi_{1,j_1},\dots,\psi_{\beta,1},\dots,\psi_{\beta,j_\beta}$*
3. *and MSOL–formulae $\psi_{I,1},\dots,\psi_{I,j_I}$*

such that for every $\mathbf{R}$, $\mathbf{R}_I$ and $\mathbf{R}_{\mathcal{B}_i}$ as above with $\mathbf{R}_{\mathcal{B}_i} \models \psi_{i,j}$ iff $b_{i,j} = 1$ and $\mathbf{R}_I \models \psi_{I,j}$ iff $b_{I,j} = 1$ we have

$$\mathbf{R} \models \varphi \text{ iff } F_{\Phi,\varphi}(b_{1,1},\dots,b_{1,j_1},\dots,b_{\beta,1},\dots,b_{\beta,j_\beta},b_{I,1},\dots,b_{I,j_I}) = 1.$$

Moreover, $F_{\Phi,\varphi}$ and the $\psi_{i,j}$ are computable from $\Phi^{\#}$ and φ , but are tower exponential in the quantifier depth of φ[1].

[1] Note that in most real applications, F_ϕ and the $\psi_{\alpha,j}$ are single exponential in the quantifier rank of ϕ.

Proof. By analyzing the proof of Theorem 4 and using Theorem 1.

Finally, we receive our main result, concerning handling of queries under restructuring of databases:

Theorem 6. *Let I be an index, $\mathcal{L}$ be FOL (or rather any language for which Theorem 5 holds), and let $\mathbf{R}^{g+1}$ be the generalized sum of $\mathbf{R}_1^{g+1\prime},\ldots,\mathbf{R}_\ell^{g+1\prime}$ over I, as usual. Let Φ_g, Ψ_g and Φ_g^{up} of the logic $\mathcal{L}$ be as above. Any query Φ_g^{app} over $\mathbf{R}^g$ gives the corresponding query Φ_{g+1}^{app} over $\mathbf{R}^{g+1}$, where $\Phi_{g+1}^{app} = \Phi_g^{app}(\Psi_g)$ and each $\varphi_{g+1,i}^{app}$ in Φ_{g+1}^{app} may be computed with the help of the corresponding boolean function $F_{\{\Phi_g,\Psi_g,\Phi_g^{app}\},\varphi_{g+1,i}^{app}}(b_{1,1},\ldots,b_{1,j_1},\ldots,b_{\ell,1},\ \ldots,b_{\ell,j_\ell},b_{I,1},\ldots,b_{I,j_I})$ as in Theorem 5.*

5 Handling Queries Under Database Updates

Assume that we have a database scheme $\mathbf{R}$ and a query (translation scheme) Φ_{View}, which defines the view. Assume that $\mathbf{R}$ was updated by translation scheme Φ^{up}. In terms of translation schemes, we obtain the following formulation:

Given: Translation scheme Φ_V, and the database update Φ_{DB}^{up}.

Find: A set of view updates Φ_V^{incr} that uses the old content of the view and delete from and inserts in the view some set of tuples defined on the source database.

This leads to the situation on Fig. 3, where Φ_V^{incr} uses both: database and the old view. For the case of queries defined in relational algebra and for updates given as deletion and insertion of a (undefined) set of tuples, the question was investigated in [28]. For the case of Datalog, the answer for the same kind of updates is given in [12]. However, the techniques were defined for the specific languages. Moreover, the update operations, used in both cases are *data changes*. It means that sets of tuples, which we insert in relations or delete from relations are not defined, but given by enumeration.

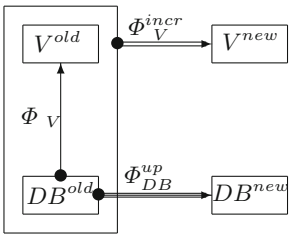

Fig. 3. Incremental view maintenance under update.

Recently, in [21], *dynamic problem* was introduced in the following way. For a sequence $w = \sigma_1,\ldots,\sigma_m \in \triangle_{can}^*(\tau)$ of operations (update translation schemes

like Φ^{up}_{DB} in our formulation) and a structure $\mathcal{U}$, $w(\mathcal{U})$ is the result of subsequently applying the operations to $\mathcal{U}$ (DB^{new}), and $\mathcal{U}$ (DB^{old}) if $w = \epsilon$.

Definition 12 ([21])**.** *Let S be a Boolean query on τ -structures. The dynamic problem $\mathcal{D}(S)$ associated with S is the set of pairs $\mathcal{D} = (\mathcal{U}, w)$ where $\mathcal{U} \in Fin(\tau)$ and $w \in \triangle_{can}(\tau)$ is an update sequence with $w(\mathcal{U}) \in S$. The query S is called the underlying static problem of $\mathcal{D}(S)$.*

The dynamic problems are handled by *incremental evaluation systems*. These systems allow auxiliary relations over the universe of the input structure $\mathcal{U}$. Incremental Evaluation System (IES) for a dynamic problem $\mathcal{D}(S)$ consists of a set of logical interpretations (translation schemes) and an additional logical sentence φ. Given an initial structure $\mathcal{D}$, the IES defines auxiliary relations over the universe of $\mathcal{U}$ by an interpretation called the initial interpretation.

In practice, we are interested in update operations, which we call *relational updates*, that means definable updates. Indeed, as a rule, a regular query that deletes (inserts) data from (to) a database looks like: delete from relation R all tuples, such that ...

Let us use one example from [28] for our purposes and paraphrase it the following way:

Example 10. Given database scheme $\mathbf{R} = (R_1, R_2, R_3, R_4)$ and $\Phi_V = (\phi)$, where

$$\phi = (\pi_A R_1 \cup R_2) \bowtie (R_3 - \sigma_\zeta R_4).$$

Suppose a database update causes a set of tuples $\bigtriangledown_\theta R_4$ to be deleted, where θ is a formula that defines the set of tuples to be deleted.

The update changes only one relation and its translation scheme is: $\Phi^{up}_{DB} = (R_1(x_1, \ldots, x_{n_1}), R_2(x_1, \ldots, x_{n_2}), R_3(x_1, \ldots, x_{n_3}), R_4(x_1, \ldots, x_{n_4}) \wedge \neg\theta(x_1, \ldots, x_{n_4}))$, where θ, in general, contains parameters.

In terms of FOL, the query that defines the view is:

$$\phi(x_1, \ldots, x_{n_3}) = ((\exists x_2 \ldots \exists x_{n_1} R_1(x_1, \ldots, x_{n_1}) \vee R_2(x_1)) \wedge (R_3(x_1, \ldots, x_{n_3}) \wedge \neg(R_4(x_1, \ldots, x_{n_3}) \wedge \zeta(x_1, \ldots, x_{n_3})))).$$

After the update, made by Φ^{up}_{DB}, the query is:

$$\Phi^{up\#}_{DB}(\phi(x_1, \ldots, x_{n_3})) = ((\exists x_2 \ldots \exists x_{n_1} R_1(x_1, \ldots, x_{n_1}) \vee R_2(x_1)) \wedge (R_3(x_1, \ldots, x_{n_3}) \wedge \neg((R_4(x_1, \ldots, x_{n_3}) \wedge \neg\theta(x_1, \ldots, x_{n_3})) \wedge \zeta(x_1, \ldots, x_{n_3})))).$$

First, we show:

$$\begin{aligned}
&((R_4(x_1, \ldots, x_{n_3}) \wedge \neg\theta) \wedge \zeta) = \\
&(R_4(x_1, \ldots, x_{n_3}) \wedge \zeta \wedge \neg\zeta) \vee (R_4(x_1, \ldots, x_{n_3}) \wedge \zeta \wedge \neg\theta) = \\
&(R_4(x_1, \ldots, x_{n_3}) \wedge \zeta) \wedge (\neg\theta \vee \neg\zeta) = \\
&(R_4(x_1, \ldots, x_{n_3}) \wedge \zeta \wedge \neg R_4(x_1, \ldots, x_{n_3})) \vee ((R_4(x_1, \ldots, x_{n_3}) \wedge \zeta) \wedge (\neg\theta \vee \neg\zeta)) = \\
&(R_4(x_1, \ldots, x_{n_3}) \wedge \zeta) \wedge (\neg R_4(x_1, \ldots, x_{n_3}) \vee \neg\zeta \vee \neg\theta) = \\
&(R_4(x_1, \ldots, x_{n_3}) \wedge \zeta) \wedge \neg(R_4(x_1, \ldots, x_{n_3}) \wedge \zeta \wedge \theta).
\end{aligned}$$

Now, we use the equivalence, obtained above, for $\Phi^{up\#}_{DB}(\phi(x_1, \ldots, x_{n_3}))$:

$$\begin{aligned}\Phi^{up\#}_{DB}(\phi) = &((\exists x_2 \ldots \exists x_{n_1} R_1(x_1, \ldots, x_{n_1}) \vee R_2(x_1)) \wedge \\ &(R_3(x_1, \ldots, x_{n_3}) \wedge \neg((R_4(x_1, \ldots, x_{n_3}) \wedge \neg\theta) \wedge \zeta))) = \\ &((\exists x_2 \ldots \exists x_{n_1} R_1(x_1, \ldots, x_{n_1}) \vee R_2(x_1)) \wedge \\ &(R_3(x_1, \ldots, x_{n_3}) \wedge \neg((R_4(x_1, \ldots, x_{n_3}) \wedge \zeta) \wedge \\ &\neg(R_4(x_1, \ldots, x_{n_3}) \wedge \zeta \wedge \theta)) = \\ &((\exists x_2 \ldots \exists x_{n_1} R_1(x_1, \ldots, x_{n_1}) \vee R_2(x_1)) \wedge \\ &((R_3(x_1, \ldots, x_{n_3}) \wedge \neg(R_4(x_1, \ldots, x_{n_3}) \wedge \zeta)) \vee \\ &(R_3(x_1, \ldots, x_{n_3}) \wedge (R_4(x_1, \ldots, x_{n_3}) \wedge \zeta \wedge \theta))) = \\ &((\exists x_2 \ldots \exists x_{n_1} R_1(x_1, \ldots, x_{n_1}) \vee R_2(x_1)) \wedge (R_3 \wedge \neg(R_4 \wedge \zeta))) \vee \\ &((\exists x_2 \ldots \exists x_{n_1} R_1(x_1, \ldots, x_{n_1}) \vee R_2(x_1)) \wedge (R_3 \wedge (R_4 \wedge \zeta \wedge \theta))) = \\ &\phi(x_1, \ldots, x_{n_3}) \vee \\ &((\exists x_2 \ldots \exists x_{n_1} R_1(x_1, \ldots, x_{n_1}) \vee R_2(x_1)) \wedge (R_3(x_1, \ldots, x_{n_3}) \wedge \\ &(R_4(x_1, \ldots, x_{n_3}) \wedge \zeta(x_1, \ldots, x_{n_3}) \wedge \theta(x_1, \ldots, x_{n_3})))).\end{aligned}$$

The second part of $\Phi^{up\#}_{DB}(\phi(x_1, \ldots, x_{n_3}))$ is exactly

$$((\pi_A R_1 \cup R_2) \bowtie (R_3 \cap \sigma_\zeta \bigtriangledown_\theta R_4)),$$

if written in relational algebra notation.

Example 10 shows that the only tools, which we really used in order to obtain the new propagation rules, were logical equivalences. Note additionally that, in general, any update translation scheme $\Phi^{up} = (\phi_1, \ldots, \phi_i, \ldots, \phi_n)$, which deletes (inserts) tuples, according to condition θ, from (to) relation R_i of database scheme $\mathbf{R} = (R_1, \ldots, R_i, \ldots, R_n)$ is in the form: $\phi_j = R_j$ if $i \neq j$ and $\phi_i = (R_i \wedge \neg\theta)$ (or $\phi_i = (R_i \vee \theta)$ for insertion of tuples, described by θ), without relativization but parametrized.

Now, the following proposition generalizes the example and gives the following answer:

Proposition 1. *For any formula ξ of FOL, MSOL or SOL and for any update translation scheme Φ^{up} of the same logic, it holds: $\Phi^{up\#}(\xi) = \xi$ or there is a set of formulae ξ'_i, $1 \leq i \leq n$ of the same logic, such that $\Phi^{up\#}(\xi) = (\ldots((\xi \circ_1 \xi'_1) \circ_2 \xi'_2) \ldots \circ_n \xi'_n)$, where $\circ_i \in \{\wedge, \vee\}$.*

Proof. By induction on ξ.

To show the same fact for LFP, IFP and TC, we use:

Theorem 7. *Given $\psi_1(\bar{x}, X, \bar{y})$ and $\psi_2(\bar{x}, X, \bar{z})$, it holds:*

$$LFP\bar{x}, X, \bar{u}(\psi_1(\bar{x}, X, \bar{y}) \vee \psi_2(\bar{x}, X, \bar{z})) = LFP\bar{x}, X, \bar{u}(LFP\bar{x}, X, \bar{u}(\psi_1(\bar{x}, X, \bar{y})) \vee \psi_2(\bar{x}, X, \bar{z}));$$
$$IFP\bar{x}, X, \bar{u}(\psi_1(\bar{x}, X, \bar{y}) \vee \psi_2(\bar{x}, X, \bar{z})) = IFP\bar{x}, X, \bar{u}(IFP\bar{x}, X, \bar{u}(\psi_1(\bar{x}, X, \bar{y})) \vee \psi_2(\bar{x}, X, \bar{z}));$$
$$TC\bar{x}, X, \bar{u}(\psi_1(\bar{x}, X, \bar{y}) \vee \psi_2(\bar{x}, X, \bar{z})) = TC\bar{x}, X, \bar{u}(TC\bar{x}, X, \bar{u}(\psi_1(\bar{x}, X, \bar{y})) \vee \psi_2(\bar{x}, X, \bar{z})).$$

The same holds for $\wedge$ as well.

Proof. The proof follows directly from the semantics of LFP, IFP and TC.

Now, it remains to combine Proposition 1 and Theorem 7 with the following results, proven in [14]:

Theorem 8. *If φ is an LFP-formula and φ' is an IFP-formula then there is a first-order formula ψ, such that φ is equivalent to $\exists(\forall)\bar{u}'LFP\bar{x}, X, \bar{u}\psi$ and there is an existential first-order formula ψ', such that φ' is equivalent to $\exists(\forall)\bar{u}'IFP\bar{x}, X, \bar{u}\psi'$.*

Theorem 9. *Suppose that we have two constant c and d and in our model $c \neq d$. Let φ be an existential pos-TC-formula. Then φ is equivalent to a formula of the form: $TC\bar{x}, \bar{x}', c, d\psi(\bar{x}, \bar{x}')$, where ψ is a first-order quantifier-free formula.*

Finally, we receive our main result, concerning handling of queries under database updates:

Theorem 10. *Every query expressible in FOL, $MSOL$, SOL, LFP, IFP and existential pos-TC allows incremental view re-computation.*

Proof. Use Proposition 1 with Theorems 7, 8 and 9.

As $I\text{-}DATALOG \equiv IFP$ and on ordered databases $LFP(TC)$ covers polynomial time (logarithmic space) computations, we, in particular, have:

Corollary 1. *1. Every $I\text{-}DATALOG$ program allows incremental re-computation.*

2. On ordered databases every program, computable in polynomial time or logarithmic space, allows incremental re-computation.

6 Discussion and Conclusions

The paper introduces a unified logic based approach to maintenance of queries under database changes and shows how known results in translations schemes transfer can be applied to particular problems in database maintenance. This approach unifies different aspects, related to both schema and data evolution in databases, into a single framework. The basic underlying notion of a *logical translation scheme* and its *induced maps*, is based on the classical syntactic notion of interpretability from logic, made explicit by M. Rabin in [29].

Analyzing computations on different generations of databases, using our general technique, we encountered several problems with some kinds of rules, for example, deletion over join. Systematically using the technique of translation scheme, we introduced the notion of Φ-sums and showed how queries, expressible in different extensions of FOL may be handled over different generations of the Φ-sums.

Moreover, using the technique of translation scheme, we introduced the notions of an *incremental view re-computations*. We proved that every query expressible in FOL, $MSOL$, SOL, LFP, IFP and existential *pos-TC* allows incremental view re-computations. The last results lead to the corollary that every $I\text{-}DATALOG$ program allows incremental re-computation. Moreover, it follows from our main results that on ordered databases every program, computable in polynomial time or logarithmic space, allows incremental re-computation.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Boston (1995)
2. Benedikt, M., Koch, C.: From XQuery to relational logics. ACM Trans. Database Syst. **34**(4), 25:1–25:48 (2009)
3. Buneman, P., Khanna, S., Tan, W.-C.: On propagation of deletions and annotations through views. In: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Madison, WI, 3–6 June 2002, pp. 150–158 (2002)
4. Bosse, U.: Ehrenfeucht-Fraïssé Games for Fixed Point Logic. Ph.D. thesis. Department of Mathematics, University of Freiburg, Germany (1995)
5. Bancilhon, F., Spyratos, N.: Update semantics of relational views. ACM Trans. Database Syst. **6**(4), 557–575 (1981)
6. Chang, C.C., Keisler, H.J.: Model Theory. Studies in Logic, 3rd edn. vol. 73. North-Holland, Amsterdam (1990)
7. Codd, E.F.: A relational model of large shared data banks. Commun. ACM **13**(2), 377–387 (1970)
8. Dawar, A., Hellat, L.: The expressive power of finitely many genaralized quantifiers. Technical report CSR 24–93. Computer Science Department, University of Wales, University College of Swansea, UK (1993)
9. Dong, G., Libkin, L., Wong, L.: Incremental recomputation in local languages. Inf. Comput. **181**(2), 88–98 (2003)
10. Dong, G., Su, J.: Deterministic FOIES are strictly weaker. Ann. Math. Artif. Intell. **19**(1–2), 127–146 (1997)
11. Dong, G., Su, J.: Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. J. Comput. Syst. Sci. **57**(3), 289–308 (1998)
12. Dong, G., Topor, R.: Incremental evaluation of Datalog queries. In: Hull, R., Biskup, J. (eds.) ICDT 1992. LNCS, vol. 646, pp. 282–296. Springer, Heidelberg (1992)
13. Dong, G., Zhang, L.: Separating auxiliary arity hierarchy of first-order incremental evaluation systems using $(3k+1)$-ary input relations. Int. J. Found. Comput. Sci. **11**(4), 573–578 (2000)
14. Ebbinghaus, H.D., Flum, J.: Finite Model Theory. Perspectives in Mathematical Logic. Springer, Berlin (1995)
15. Ebbinghaus, H.D., Flum, J., Thomas, W.: Mathematical Logic. Undergraduate Texts in Mathematics, 2nd edn. Springer, New York (1994)
16. Franconi, E., Guagliardo, P.: On the translatability of view updates. In: Freire, J., Suciu, D. (eds.) AMW. CEUR Workshop Proceedings, vol. 866, pp. 154–167 (2012)
17. Franconi, E., Guagliardo, P.: The view update problem revisited. CoRR abs/1211.3016 (2012)
18. Franconi, E., Guagliardo, P.: Effectively updatable conjunctive views. In: Proceedings of the 7th Alberto Mendelzon International Workshop on Foundations of Data Management, Puebla/Cholula, Mexico, 21–23 May 2013
19. Feferman, S., Vaught, R.: The first order properties of products of algebraic systems. Fundam. Math. **47**, 57–103 (1959)
20. Guagliardo, P., Pichler, R., Sallinger, E.: Enhancing the updatability of projective views. In: Proceedings of the 7th Alberto Mendelzon International Workshop on Foundations of Data Management, Puebla/Cholula, Mexico, 21–23 May 2013

21. Grädel, E., Siebertz, S.: Dynamic definability. In: Proceedings 15th International Conference on Database Theory ICDT, Berlin, Germany, 26–29 March 2012, pp. 236–248 (2012)
22. Gurevich, Y.: Modest theory of short chains, I. J. Symbolic Logic **44**, 481–490 (1979)
23. Guagliardo, P., Wieczorek, P.: Query processing in data integration. In: Kolaitis, P.G., Lenzerini, M., Schweikardt, N. (eds.) Data Exchange, Information, and Streams. Dagstuhl Follow-Ups, vol. 5, pp. 129–160. Schloss Dagstuhl, Leibniz-Zentrum für Informatik (2013)
24. Hegner, S.J.: The relative complexity of updates for a class of database views. In: Seipel, D., Turull-Torres, J.M. (eds.) FoIKS 2004. LNCS, vol. 2942, pp. 155–175. Springer, Heidelberg (2004)
25. Immerman, N.: Descriptive Complexity. Graduate Texts in Computer Science. Springer, New York (1999)
26. Koch, C.: Incremental query evaluation in a ring of databases. In: Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Indianapolis, IN, 6–11 June 2010, pp. 87–98 (2010)
27. Makowsky, J.A., Ravve, E.V.: BCNF via attribute splitting. In: Düsterhöft, A., Klettke, M., Schewe, K.-D. (eds.) Conceptual Modelling and Its Theoretical Foundations. LNCS, vol. 7260, pp. 73–84. Springer, Heidelberg (2012)
28. Quan, X., Wiederhold, G.: Incremental recomputation of active relational expressions. IEEE Trans. Knowl. Data Eng. **3**(3), 337–341 (1991)
29. Rabin, M.O.: A simple method for undecidability proofs and some applications. In: Bar Hillel, Y. (ed.) Logic, Methodology and Philosophy of Science II. Studies in Logic, pp. 58–68. North Holland, (1965)
30. Ravve, E.V., Volkovich, Z., Weber, G.-W.: A uniform approach to incremental reasoning on strongly distributed systems. In: Proceedings of GCAI2015 (2015, to appear)
31. Weber, V., Schwentick, T.: Dynamic complexity theory revisited. In: Proceedings 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, 24–26 February 2005, pp. 256–268 (2005)