Incremental View Maintenance in Data Exchange Settings

1 Introduction

The integration of data from various disparate sources is an important task for many applications. To perform a data integration task, one has to define and manipulate a *mapping task*. A mapping task specifies how an instance of the source schema translates to an instance of the target schema. It is an executable transformation since it is dependent on the expressiveness of the underlying language, i.e. SQL or XQuery. A prominent way of expressing mapping tasks is through *tuple generating dependencies*. Systems such as Clio or +Spicy implement this approach: they generate a *schema mapping* using TGDs to specify correspondences between relational atoms of the source and target schema.

An important line of research is the theory of *data exchange*. The notion of a *data exchange problem* has been formalized after an extensive line of work. In this setting, researchers are interested in the characterization of the correspondences in a schema mapping and the understanding of these properties. Thus, the formalization of a solution concept for data exchange problems has been conceived which has been called a *core* solution. In a high level, the core solution is the answer to the natural question "which solution should be materialized in a data exchange problem?". In a sense, it is a superior solution with nice properties such as

1. It contains no redundancy, since it is the smallest solution that preserves the semantics of the exchange.

2. It represents a "good" instance for answering queries over the target database.

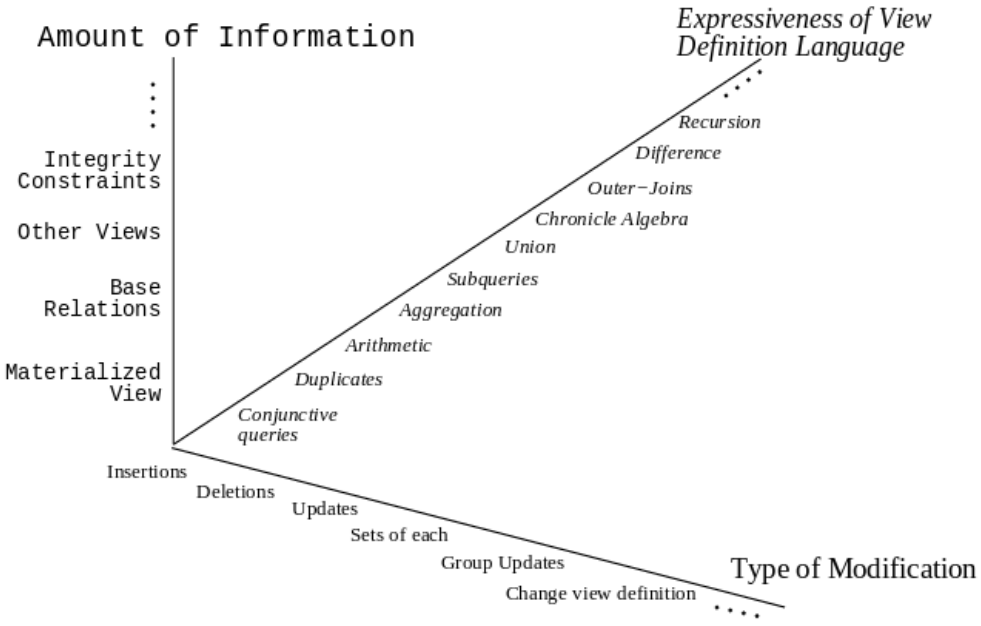
A second line of work is the incremental maintenance of materialized views. If we think of the target schema relations as materialized views of the source schema relations, we can identify four dimensions along which the problem can be studied, as defined in [1].

1. Information Dimension: Is there access to all source schema relations while doing the maintainance? What about integrity constraints and keys?

2. Modification Dimension: Which operations are we planning to support with our proposed view maintainance problem? Moreover, conceptually what does an update operation mean (insert-delete or stand-alone operation)?

3. Language Dimension: How is the materialized view expressed? Examples are SQL, subset of SQL, DataLog etc. Do we need to support duplicates or data aggregation?

4. Instance Dimension: Does the view maintainance algorithm work for every instance of the database of just for some subsets of data?



Contribution In this work we propose a way to perform incremental view maintenance in a Data Exchange setting. For a given schema mapping, we construct a logically equivalent one with semantically equivalent relational expressions. We the use +Spicy to find the core of the equivalent schema mapping problem. Our approach supports SPJ queries with the exception of self-joins (for now at least...).

2 Preliminaries

In this section we define the necessary components to describe a schema mapping task in a data exchange environments using an incremental view mainte-

nance approach. We fix a set of labels $\{A_1, A_2, \ldots A_k\}$ and a set of relations $\{R_1, R_2, \ldots, R_n\}$.

2.1 Schema and Dependencies

We begin by defining a schema and an instance of a relation schema.

Definition 2.1. A schema $\mathbf{S} = \{R_1, R_2, \ldots, R_n\}$ is a collection of relation schemas.

Definition 2.2. An instance of a relation schema $R(A_1, \ldots, A_k)$ is a finite set of tuples of the form $R(A_1 : v_1, \ldots, A_k : v_k)$, where, for each i, v_i is either a constant or a labeled null.

An instance of a schema $\mathbf{S}$ is a collection of instances one for each relation schema in S. It is possible to express *key contraints* and *foreign key constraints* over a schema. For the rest of this note, we blur the distinction on whether we point to the instance or the relation symbol when referring to $\mathbf{S}$.

An important formalism developed for the specification and analysis of integrity constraints is that of a *tuple-generating dependency*. Let $\mathbf{S}$ and $\mathbf{T}$ be the source and target schema instances respectively.

Definition 2.3. A tuple-generating dependency (tgd) is an assertion about the relationship between $\mathbf{S}$ and $\mathbf{T}$ of the form

$$\forall \bar{x}, (\phi(\bar{x}) \rightarrow \exists \bar{y}(\psi(\bar{x}, \bar{y})),$$

where $\bar{x}$ and $\bar{y}$ are vectors of variables, $\phi(\bar{x})$ is a conjunction of relational atoms such that all variables in $\bar{x}$ appear in it and $\psi(\bar{x}, \bar{y})$ is a conjunction of relational atoms.

An intuitive explanation of the above definition is that if $\phi(\cdot)$ holds, then $\psi(\cdot)$ must hold. $\phi(\cdot)$ and $\psi(\cdot)$ may contain equations of the form $v_i = v_j$, where v_i and v_j are variables.

Definition 2.4. A tgd is called a *source to target tgd* if $\phi(\bar{x})$ is a relational algebra expression over $\mathbf{S}$ and $\psi(\bar{x}, \bar{y})$ over $\mathbf{T}$.

Definition 2.5. A tgd is called *full* when the vector of variables $\bar{y}$ is empty, i.e.

$$\forall \bar{x}, (\phi(\bar{x}) \rightarrow (\psi(\bar{x})),$$

Definition 2.6. A tgd is called a *target tgd* if both $\phi(\bar{x})$ and $\psi(\bar{x}, \bar{y})$ are relational algebra expressions over $\mathbf{T}$.

If $\psi(\bar{x}, \bar{y})$ contains only equations then it is called an *equality generating dependency* (egd).

2.2 Data Exchange

The following definition formalizes the environment of data exchange in which we are working.

Definition 2.7 (Data Exchange Setting)**.** A data exchange setting is a quadruple $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where $\mathbf{S}$ is the source schema, $\mathbf{T}$ is the target schema, Σ_{st} is a set of tuple-generating dependencies describing a semantic mapping between $\mathbf{S}$ and $\mathbf{T}$, and Σ_t is a set of tuple-generating dependencies (usually key constraints) on the relations of $\mathbf{T}$.

Data exchange *solutions* are instances of the target schema that satisfy the constraints in the data exchange setting. A subset of such solutions are called *universal* when they preserve the semantics of the mapping and do not lose any information. Finally, the minimal universal solutions (add no redundancy) are called core solutions.

Researchers in schema mapping optimization have proposed several notions of schema mapping equivalence. The strongest one, is the logical equivalence although others have very interesting properties as well (Data-Exchange Equivalence, Conjunctive Query Equivalencem see [2]).

Definition 2.8 ([2])**.** Two schema mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma')$ are logically equivalent if for very source instance I and target instance J, (I, J) is a solution to $\mathcal{M}$ if and only if (I, J) is a solution to $\mathcal{M}'$.

This means that the two schema mappings have identical sets of solutions.

3 Efficient Maintenance of Materialized Views (not Outer Joins)

In a data exchange setting, we can consider the relations of target schema $\mathbf{T}$ as *materialized views* of source relations belonging to schema $\mathbf{S}$. The connections between the two schemas can be represented by a set of source-to-target tgds. [1]

A rich literature on incremental view maintenance techniques has been developed. We can consider each source-to-target tgd as view definition, since each tgd is an association of relational expressions. In this section we apply a prominent technique of incremental view maintenance from [3] that implements the incremental recomputation of relational algebra expressions. In addition, this technique guarantees to preserve a certain minimality condition: the resulting relational expressions do not contain any unnecessary tuples and no redundant computations are performed.[2] Note that in this section we work with all SPJ expressions except *outer joins*, which require a different treatment (see section 5).

[1] Foreign key constraints are expressed as target tgds.

[2] In fact, the minimality condition claimed by [3] was later fixed by [4], who presented slightly modified relational expressions. We will incorporate the latter ones in our work.

Recall that $R, R_1, R_2, \ldots$ denote names of relations in a database schema and A a range over sets of attribute names. The relational algebra expressions we discuss are generated by the following grammar.

$$
\begin{array}{rcll}
S & ::= & R & \text{base relation} \\
 & | & \sigma_{\boldsymbol{p}}(S) & \text{selection} \\
 & | & \Pi_{\boldsymbol{A}}(S) & \text{projection} \\
 & | & S \times S & \text{cartesian product} \\
 & | & S \cup S & \text{union} \\
 & | & S \cap S & \text{intersection} \\
 & | & S - S & \text{difference} \\
 & | & S \bowtie S & \text{natural join}
\end{array}
$$

We use S, T and Q to denote arbitrary relations. We denote by s_t the state of the database represented by schema $\mathbf{S}$ at time t and $\triangle R_i$, $\triangledown R_i$ the sets inserted to and deleted from R_i at time t. These tuples were not present in R_i in time $t-1$ [3].

Mutually Recursive functions $\triangledown$ and $\triangle$, by [4] Let Q be a relational expression that can be expressed with the grammar we presented above. Griffin et al. propose the following set of mutually recursive difference functions for Q, assuming there is a clear distinction between the base relations comprising Q at state t and $t-1$ and therefore their respective $\triangledown$ and $\triangle$ relations.

[3] An important precondition of our system is the ability to do this distinction: what data was modified at a particular time and what data was already present.

Q	$\nabla(t, Q)$
R	$\begin{cases} \nabla R & \text{if } R \leftarrow (R - \nabla R) \cup \Delta R \text{ is in } t \\ \phi & \text{otherwise} \end{cases}$
$\sigma_p(S)$	$\sigma_p(\nabla(t, S))$
$\Pi_A(S)$	$\Pi_A(\nabla(t, S)) - \Pi_A(\text{mod}(t, S))$
$S \times T$	$(\nabla(t, S) \times T) \cup (S \times \nabla(t, T))$
$S \cup T$	$(\nabla(t, S) - \text{mod}(t, T)) \cup (\nabla(t, T) - \text{mod}(t, S))$
$S \cap T$	$(\nabla(t, S) \cap T) \cup (\nabla(t, T) \cap S)$
$S - T$	$(\nabla(t, S) - T) \cup (\Delta(t, T) \cap S)$
$S \ltimes T$	$(\nabla(t, S) \ltimes T) \cup (S \ltimes \nabla(t, T))$

Q	$\Delta(t, Q)$
R	$\begin{cases} \Delta R & \text{if } R \leftarrow (R - \nabla R) \cup \Delta R \text{ is in } t \\ \phi & \text{otherwise} \end{cases}$
$\sigma_p(S)$	$\sigma_p(\Delta(t, S))$
$\Pi_A(S)$	$\Pi_A(\Delta S) - \Pi_A(S)$
$S \times T$	$(\text{mod}(t, S) \times \Delta(t, T)) \cup (\Delta(t, S) \times \text{mod}(t, T))$
$S \cup T$	$(\Delta(t, S) - T) \cup (\Delta(t, T) - S)$
$S \cap T$	$(\Delta(t, S) \cap \text{mod}(t, T)) \cup (\Delta(t, T) \cap \text{mod}(t, S))$
$S - T$	$(\Delta(t, S) - \text{mod}(t, T)) \cup (\nabla(t, T) \cap \text{sub}(t, S))$
$S \bowtie T$	$(\text{mod}(t, S) \bowtie \Delta(t, T)) \cup (\Delta(t, S) \bowtie \text{mod}(t, T))$

In the table above, we use the abbreviations $sub(t, S)$ for $S - \nabla(t, S)$ and $mod(t, S)$ for $(S - \nabla(t, S)) \cup \Delta(t, S)$ for every transaction t.

Proposition 3.1 (rf. Griffin, Libkin, Trickey [4]). *At time t , the relational expression Q_t and the (recursive) relational expression $(Q_{t-1} - \nabla Q_{t-1}) \cup \Delta Q_{t-1}$ defined by the above table are semantically equivalent.*

The authors show that their expressions preserve minimality and add no additional redundancy.

Relational Algebra and SQL Example(MIPMAP Terms): Consider the following inner join query:

`PatientMapping(p_ide, p_num), EncounterMapping(e_ide, e_num), Exam(p_ide, e_ide, exam, value)` $\mapsto$ `Observation(p_num,e_num,exam,value)`. For notational convenience let S_1 = `PatientMapping`, S_2 = `EncounterMapping`, S_3 = `Exam` and T = `Observation`.

We first write the relational algebra and SQL Expressions without incremental view maintenance. In relational algebra we are looking to compute

$$T = \Pi_{S_1.p\_num, S_2.e\_num, S_3.exam, S_3.value}(S_1 \bowtie S_3 \bowtie S_2)$$

or equivalently in SQL:

```
insert into T
select S1.p_num, S2.e_num, S3.exam, S3.value
from S1, S3, S2
where S1.p_ide = S3.p_ide, S3.e_ide = S2.e_ide
```

In this setting we insert into T the contents of S_1, S_3, S_2[4], for the IVM Scenario consider that S_1, S_3 and S_2 contain that inserted in a previous time only and no data is deleted. At the current time, new data arrive and are stored in relations $\Delta S_1, \Delta S_3, \Delta S_2$. Let $p = \{S_1.p\_num, S_3.e\_num, S_2.exam, S_2.value\}$ the projecting set of attributes. At time t, we compute $\Delta(t, T)$ as follows:

$$\begin{aligned}\Delta(t,T) &= \Delta(t, \Pi_p((S_1 \bowtie S_3) \bowtie S_2) \\ &= \Delta(t, \Pi_p((\texttt{mod(t, } S_1\texttt{)} \bowtie \Delta(t, S_3)) \cup (\Delta(t, S_1) \bowtie \texttt{mod(t, } S_3\texttt{)}) \bowtie S_2) \\ &= \Delta(t, \Pi_p(((S_1 \cup \Delta S_1) \bowtie \Delta S_3) \cup (\Delta S_1 \bowtie (S_3 \cup \Delta S_3)) \bowtie S_2).\end{aligned}$$

Let $Q = ((S_1 \cup \Delta S_1) \bowtie \Delta S_3) \cup (\Delta S_1 \bowtie (S_3 \cup \Delta S_3)$. Note that Q is a terminal expression (not a function of t!). We continue the recursion for the second join and we have:

$$\begin{aligned}\Delta(t,T) &= \Delta(t, \Pi_p(Q \bowtie S_2)) \\ &= \Delta(t, \Pi_p(Q \bowtie \Delta(t, S_2))) \\ &= \Delta(t, \Pi_p(Q \bowtie \Delta S_2)) = \Pi_p(Q \bowtie \Delta S_2).\end{aligned}$$

4 Incremental View Maintenance on SPJ TGDs

In this section we show that for a data exchange problem there is an equivalent one which incorporates incremental view maintenance. We conclude that the core solutions of these schema mapping scenarios coincide and propose a way of computing the core solution using the rewriting algorithm of +Spicy.

4.1 Construction of a logically equivalent IVM Schema Mapping Scenario

We will refer to the left-hand side of a tgd c as the *premise* $prem(c)$ and to the *right hand side* of c as the *conclusion* $conc(c)$. At a database state, let

$$\mathcal{M} =< \mathbf{S} \cup \triangle\mathbf{S} \cup \triangledown\mathbf{S}, \mathbf{T} \cup \triangle\mathbf{T} \cup \triangledown\mathbf{T}, \Sigma_{st} \cup \triangle\Sigma_{st} \cup \triangledown\Sigma_{st}, \Sigma_t >$$

be a data exchange problem (or a schema mapping scenario) where

- $\triangle\mathbf{S}$ and $\triangledown\mathbf{S}$ are the collections containing the relations $\triangle R_i$ and $\triangledown R_i$, for $i = 1, 2, \ldots, n$,

[4]But these are our deltas at each T for MIPMAP, right Taso? We don't have a separation of data getting in at times before t...Anyway, assume we had full data at all times!

- $\triangle\mathbf{T}$ and $\triangledown\mathbf{T}$ are the collections containing the relations $\triangle T_i$ and $\triangledown T_i$, for $i = 1, 2, \ldots, m$.[5].
- Σ_{st} are the source-to-target tgds specifying the actual schema correspondences.
- $\triangle\Sigma_{st}$ and $\triangledown\Sigma_{st}$ are the collections containing tgds of the form

$$\forall\bar{x}, (\triangle\phi(\bar{x}) \rightarrow \exists\bar{y}(\triangle\psi(\bar{x}, \bar{y})),$$

and

$$\forall\bar{x}, (\triangledown\phi(\bar{x}) \rightarrow \exists\bar{y}(\triangledown\psi(\bar{x}, \bar{y})),$$

for every tgd $c : \forall\bar{x}, (\phi(\bar{x}) \rightarrow \exists\bar{y}(\psi(\bar{x}, \bar{y})) \in \Sigma_{st}$.

- Σ_t are target constraints (key constraints, foreign key constraints).

We now define a second data exchange problem $\mathcal{M}'$ as follows.

Definition 4.1. Let $\mathcal{M}$ be the schema mapping scenario defined above. We define

$$\mathcal{M}' =< \mathbf{S} \cup \triangle\mathbf{S} \cup \triangledown\mathbf{S}, \mathbf{T} \cup \triangle\mathbf{T} \cup \triangledown\mathbf{T}, \triangle\Sigma_{st} \cup \triangledown\Sigma_{st}, \Sigma_t \cup \Sigma_{ft} > .$$

In this schema mapping scenario, Σ_{ft} is a set of $|\Sigma_{st}|$ target tgds with the form

$$\forall\bar{x}, (\psi(\bar{x}) - \triangledown\psi(\bar{x})) \cup \triangle\psi(\bar{x}) \rightarrow (\psi(\bar{x})), \tag{1}$$

where $\psi(\cdot)$ is a relational expression that is the conclusion of a particular tgd $c \in \Sigma_{st}$.

We will show that the schema mappings $\mathcal{M}$ and $\mathcal{M}'$ are logically equivalent, according to definition [definition]. In [2], Fagin et al. showed that it is decidable whether two schema mappings specified by finite sets of s-t tgds and finite weakly acyclic sets of target tgds. We first observe that both $\mathcal{M}$ and $\mathcal{M}'$ have these properties.

Proposition 4.1. *Schema mappings $\mathcal{M}$ and $\mathcal{M}'$ have*

1. *Finite sets of s-t tgds.*
2. *Finite sets of weakly acyclic sets of target tgds.*

Proof. We can show the first statement of the proposition by observing that the set Σ_{st} has a finite number of elements. Therefore, $\Sigma_{st} \cup \triangle\Sigma_{st} \cup \triangledown\Sigma_{st}$ and $\triangle\Sigma_{st} \cup \triangledown\Sigma_{st}$ are finite. Hence, both $\mathcal{M}$ and $\mathcal{M}'$ have finite sets of source to target tgds.

[5]The inclusion of $\triangle\mathbf{S} \cup \triangledown\mathbf{S}$ and $\triangle\mathbf{T} \cup \triangledown\mathbf{T}$ is a theoretical construction, the purpose of which will become apparent later. In a real world implementation of $\mathcal{M}$, these relations can be safely omitted.

For the second statement, we similarly conclude that the set of key constraints Σ_t is finite by the definition of the problem, while the set Σ_{ft} is finite since $|\Sigma_{ft}| = |\Sigma_{st}|$ by the definition of $\mathcal{M}'$.

To complete the proof we need to verify that Σ_t and Σ_{ft} contain weakly acyclic target tgds. We will use two well known facts from data-exchange theory to facilitate our claim.

Fact 4.1. Target tuple generating dependencies that express key constraints are weakly acyclic tgds.

Therefore, since Σ_t contains key constraints, $\mathcal{M}$ satisfies the second statement of the proposition. We now observe that the tgds contained in Σ_{ft} defined in equation (4.1) are full target tgds, since they adhere to definitions 2.5 and 2.6.

Fact 4.2. Full-target tgds are weakly acyclic tgds.

This concludes our proof. □

Now that we have shown that $\mathcal{M}$ and $\mathcal{M}'$ adhere to the characterization of the proposition, we need to show that the mappings are logically equivalent. As a matter of fact, the decidability result of Fagin et al. for the equivalence of the above schema mappings is constructive. The authors state that one can exploit a known property of the *chase* [5] algorithm to prove logical equivalence between two schema mappings. They argue that for two sets of tgds Σ and Σ' consisting of s-t tgds and weakly acyclic target tgds, to check whether a tgd c $in\Sigma'$ belongs to a solution of Σ (and $\mathcal{M}$),it is enough to check whether the conclusion of c appears in the result of chasing the premise of c with Σ.[6]. Repeating the process for every tgd in Σ' allows us to check whether Σ' satisfies $\mathcal{M}$. Then, it suffices to show the converse (Σ satisfies $\mathcal{M}'$) to show that the two mappings are logically equivalent or that their solution sets coincide. With this process in mind, we show that the mapping $\mathcal{M}'$ we have defined is logically equivalent to $\mathcal{M}$.

Theorem 4.1. *The mappings $\mathcal{M}$ and $\mathcal{M}'$ are logically equivalent.*

Proof. We begin by checking whether each one of the s-t tgds and target tgds of $\mathcal{M}'$ belong to the solution of $\mathcal{M}$. The source to target tgds of $\mathcal{M}'$ are $\triangle\Sigma_{st} \cup \triangledown\Sigma_{st}$ are already included in the source to target tgds of $\mathcal{M}$. The same holds for the key constraints Σ_t.

We investigate the set of full target tgds Σ_{ft}. Fix a full-target tgd $f \in \Sigma_{ft}$. The premise of f is the left hand side of equation (4.1). We can be certain that should we chase $prem(f)$ with the set of Σ_{st} of $\mathcal{M}$, the conclusion of f (right hand side of (4.1)) will appear in the result since there is a tgd $c \in \Sigma_{st}$ for which $prem(c)$ is semantically equivalent to $prem(f)$ due to proposition 3.1 that shows the equivalence of the incremental view maintenance relational expressions to

[6]We use the fact that the chase is terminating (always outputs a universal solution) for such schema mappings.

the original ones. This claim holds for every $f \in \Sigma_{ft}$, by the definition of Σ_{ft}. This concludes the one direction of the proof.

To prove the other direction, we repeat the process for each source-to-target tgd $c \in \Sigma_{st}$. Fix a source-to-target tgd $c \in \Sigma_{st}$. Once again, we can be certain that if we chase the premise of c with the set of Σ_{ft} of $\mathcal{M}'$, the conclusion of c will appear in the result since there is a tgd $f \in \Sigma_{ft}$ for which $prem(f)$ is semantically equivalent to $prem(c)$ due to proposition 3.1. □

By the definition of logical equivalence, we can conclude that the sets of solutions for $\mathcal{M}$ and $\mathcal{M}'$ coincide. This means that the sets of universal solutions coincide and leads to deduce that the *core* solution is identical.

4.2 Computation of core solution of a logically equivalent IVM Schema Mapping Scenario

In the previous subsection, we showed that we can construct a logically equivalent schema mapping scenario to any other schema mapping consisting of source to target tgds and weakly acyclic target tgds. The main property of this schema mapping is the utilization of semantically equivalent incremental view maintenance expressions with a guarantee of obtaining a core solution.

The +Spicy system [6] provides an algorithm that computes the core solution of schema mappings scenarios with the above characterization.The characterization is very strict! Adding an egd or a non weakly acyclic constraint to a data exchange problem can make the problem of finding the core immediately intractable.

The algorithm of Mecca et al. is based on a *rewriting* of the source-to-target tgds using equivalent expressions with *negation*. In this way, a different set of tgds is obtained. The main advantage of this approach is the fact that the output tgds can be translated to SQL and utilize the power and scalability of a relational database engine (e.g. Postgres).

We will now verify that it is possible to rewrite with +Spicy the schema mapping $\mathcal{M}'$ that we obtained above by showing that $\mathcal{M}'$ meets the preconditions of the algorithm. First of all, we need to check whether the tgds of the mapping are "useful" or *source based*. If the tgds of $\mathcal{M}$ are source based, then the tgds of $\mathcal{M}'$ are source-based. Elaborate on this and on the normal forms of the resulting tgds. The latter can be obtained by modifying the dual Gaifman graph, before "feeding" the tgds to the rewriting algorithms of +Spicy.

5 Incremental View Maintenance on Outer Join TGDs

A different treatment, still questions to work on... Disj. Normal Form Galindo-Legaria doesn't look efficient.

6 Discussion and Next Steps

References

[1] Gupta, A., & Mumick, I. S. (1995). Maintenance of materialized views: Problems, techniques, and applications. IEEE Data Eng. Bull., 18(2), 3-18.

[2] Fagin, R., Kolaitis, P. G., Nash, A., & Popa, L. (2008, June). Towards a theory of schema-mapping optimization. In Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (pp. 33-42). ACM.

[3] Qian, X., & Wiederhold, G. (1991). Incremental recomputation of active relational expressions. IEEE transactions on knowledge and data engineering, 3(3), 337-341.

[4] Griffin, T., Libkin, L., & Trickey, H. (1997). An improved algorithm for the incremental recomputation of active relational expressions. IEEE Transactions on Knowledge and Data Engineering, 9(3), 508-511.

[5] Abiteboul, S., Hull, R., & Vianu, V. (1995). Foundations of databases: the logical level. Addison-Wesley Longman Publishing Co., Inc..

[6] Mecca, G., Papotti, P., & Raunich, S. (2009, June). Core schema mappings. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of data (pp. 655-668). ACM.