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.

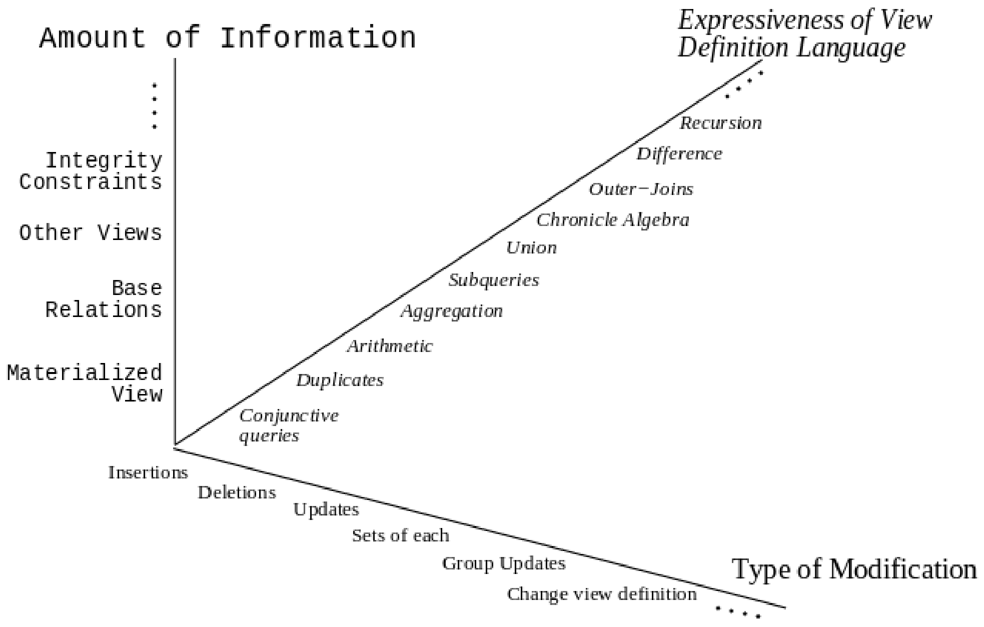
A second line of work is the incremental maintenance of materialized views. If we think of the target schema relations as materialized views of the source schema relations, we can identify four dimensions along which the problem can be studied, as defined in [1].

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.

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

2.1 Schema and Dependencies

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

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

Definition 2.2. An instance of a relation schema $R(A_1, \dots, A_k)$ is a finite set of tuples of the form $R(A_1 : v_1, \dots, A_k : v_k)$, 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 relational expression (not necessarily a conjunction) 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.

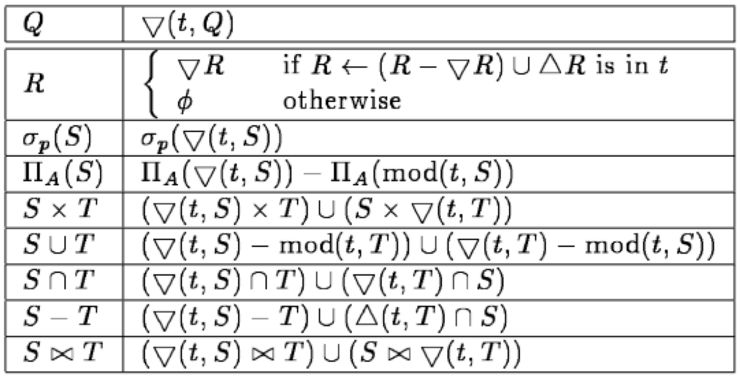
Recall that $R, R_1, R_2, \dots$ denote names of relations in a database schema and A a range over sets of attribute names. The relational algebra expressions we discuss are generated by the following grammar.

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

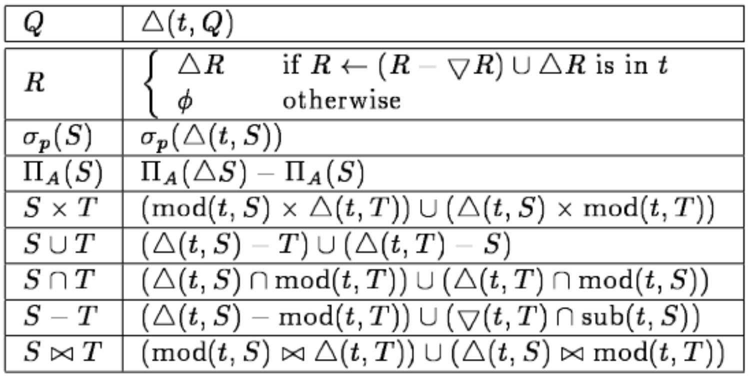
We use S, T and Q to denote arbitrary relations. We denote by s_t the state of the database represented by schema **S** at time t and $\triangle R_i$, $\triangledown R_i$ the sets inserted to and deleted from R_i at time t. These tuples were not present in R_i in time $t-1$ [3].

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



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

In the table above, we use the abbreviations $sub(t, S)$ for $S - \nabla(t, S)$ and $mod(t, S)$ for $(S - \nabla(t, S)) \cup \triangle(t, S)$ 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 $(Q_{t-1} - \nabla Q_{t-1}) \cup \triangle Q_{t-1}$ 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, 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 expressions. 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 Sceario

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* $conc(c)$. Let

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

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

- $\mathbf{S}^{\cup}$ is the collection containing materialized views of relations in $\mathbf{S} \cup \triangle \mathbf{S}$ that are constructed as follows: for $i = 1, \ldots, n$ define $S_i^{\cup} := S_i \cup \Delta S_i$.

- $\Sigma_{st}^{\cup}$ are the source-to-target tgds specifying the schema correspondences of the form
$$\forall \bar{x}, (\phi(\bar{x}) \rightarrow \exists \bar{y}(\psi(\bar{x}, \bar{y}))).$$
where $\phi(\cdot)$ is a conjunctive query over the materialized views $\mathbf{S}^{\cup}$ and $\psi(\cdot)$ a a conjunctive query over the relations of the target schema $\mathbf{T}$.

- Σ_t are target tgds.

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

Definition 4.1. Let $\mathcal{M}'$ be the schema mapping scenario defined as follows:

$$\mathcal{M}' =< \mathbf{S} \cup \triangle\mathbf{S} \cup \mathbf{S}^{-}, \mathbf{T}, \Sigma_{st} \cup \Sigma_{st}^{\Delta}, \Sigma_t >$$

where

- $\triangle\mathbf{S}$ is the collection containing relations $\triangle R_i$, for $i = 1, 2, \ldots, n$,

- $\mathbf{S}^{-}$ is the collection containing materialized views of relations in $\mathbf{S} \cup \triangle\mathbf{S}$ that are constructed as follows: for every tgd in Σ_{st} that represents a projection of attributes $\bar{x}' \in S_\ell$ we define $S_\ell^{-}(\bar{x}') := \Delta S_\ell(\bar{x}') - S_\ell(\bar{x}')$.

- Σ_{st} are the source-to-target tgds specifying the schema correspondences of the form
$$\forall \bar{x}, (\phi(\bar{x}) \rightarrow \exists \bar{y}(\psi(\bar{x}, \bar{y}))).$$
where $\phi(\cdot)$ is a conjunctive query over relations of $\mathbf{S}$ and $\psi(\cdot)$ a a conjunctive query over the relations of the target schema $\mathbf{T}$.

- Σ_{st}^{Δ} is the set source-to-target tgds constructed as follows:

  1. For every s-t tgd over $S_\ell \in \mathbf{S}$ with no conjunctions, i.e. $\forall \bar{x} \in S_\ell, (\phi(\bar{x}) \rightarrow \exists \bar{y}(\psi(\bar{x}, \bar{y}))) \in \Sigma_{st}$, add the tgd $\forall \bar{x} \in S_\ell^{-}, (\phi(\bar{x}) \rightarrow \exists \bar{y}(\psi(\bar{x}, \bar{y})))$ to Σ_{st}^{Δ}, where $\Delta S_\ell \in \Delta\mathbf{S}$.
  2. For every s-t tgd with conjunctions over relations $S_k, S_\ell \in \mathbf{S}$, i.e. $\forall \bar{x}_k \in S_k, \bar{x}_\ell \in S_\ell, (\phi_k(\bar{x}_k) \wedge \phi_\ell(\bar{x}_\ell) \rightarrow \exists \bar{y}(\psi(\bar{x}_k, \bar{x}_\ell, \bar{y}))) \in \Sigma_{st}$, add the following three tgds to Σ_{st}^{Δ}:
     (a) $\forall \bar{x}_k \in \Delta S_k, \bar{x}_\ell \in \Delta S_\ell, (\phi_k(\bar{x}_k) \wedge \phi_\ell(\bar{x}_\ell) \rightarrow \exists \bar{y}(\psi(\bar{x}_k, \bar{x}_\ell, \bar{y})))$
     (b) $\forall \bar{x}_k \in S_k, \bar{x}_\ell \in \Delta S_\ell, (\phi_k(\bar{x}_k) \wedge \phi_\ell(\bar{x}_\ell) \rightarrow \exists \bar{y}(\psi(\bar{x}_k, \bar{x}_\ell, \bar{y})))$
     (c) $\forall \bar{x}_k \in \Delta S_k, \bar{x}_\ell \in S_\ell, (\phi_k(\bar{x}_k) \wedge \phi_\ell(\bar{x}_\ell) \rightarrow \exists \bar{y}(\psi(\bar{x}_k, \bar{x}_\ell, \bar{y})))$.

- Σ_t are target tgds.

We will show that the schema mappings $\mathcal{M}$ and $\mathcal{M}'$ are logically equivalent, according to definition 2.2. 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 are logically equivalent or not. Firstly, it is ovious 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.*

Since $\mathcal{M}$ 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.[4]. 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. Let f be a tgd in Σ_{st} of $\mathcal{M}'$. We can be certain that the conclusion of f will be executed should we fire the set $\Sigma_{st}^{\cup}$ of $\mathcal{M}$ as the tgds in $\Sigma_{st}^{\cup}$ run over materialized views over $S_i \cup \Delta S_i, i = 1, \ldots, n$, whereas those in Σ_{st} run over $\mathbf{S} \subseteq (\mathbf{S} \cup \Delta\mathbf{S})$

Let f be a tgd in Σ_{st}^{Δ} of $\mathcal{M}'$. Consider the following cases:

1. f is a tgd of the form $\forall \bar{x} \in S_\ell^-, (\phi(\bar{x}) \to \exists \bar{y}(\psi(\bar{x}, \bar{y})))$.

   By construction, it is true that $S_\ell^- = \Delta S_\ell - S_\ell$ which is a subset of ΔS. Since tgds in $\Sigma_{st}^{\cup}$ run over data from both $\mathbf{S}$ and $\Delta\mathbf{S}$, it is certain that the conclusion of f will be applied.

2. f is a tgd of the form $\forall \bar{x}_k \in \Delta S_k, \bar{x}_\ell \in \Delta S_\ell, (\phi_k(\bar{x}_k) \wedge \phi_\ell(\bar{x}_\ell) \to \exists \bar{y}(\psi(\bar{x}_k, \bar{x}_\ell, \bar{y})))$.

   Similarly, this is an inner join run over a subset of $S_{\cup} \bowtie T_{\cup}$, therefore there exist tgds in $\Sigma_{st}^{\cup}$ that will reach the conclusion of f.

Let f be a tgd in $\Sigma_{st}^{\cup}$ of $\mathcal{M}$. By proposition 3.1., since the expressions of the premises of Σ_{st}^{Δ} and Σ_{st} are semantically equivalent to those in the premises of f, the conclusion of will be reached by applying the s-t tgds of $\mathcal{M}'$ to $\mathcal{M}$. □

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 this leads us to the fact that the *core* solution is identical.

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

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 a mapping task $\mathcal{M} = (\mathbf{S}.\mathbf{T}, \Sigma_{st}, \Sigma_t)$ with the following preconditions (see [6], section 5):

1. Target tgds are not considered[5].

2. The set Σ_{st} is *source-based.* A tgd $\phi(\bar{x}) \rightarrow \exists \bar{y} \psi(\bar{x}, \bar{y}))$ is source-based if (i) $\phi(\bar{x})$ is not empty, (ii) $\bar{x}$ is not empty and (iii) at least one of the variables in $\bar{x}$ appear in the right-hand side $\psi(\bar{x}, \bar{y}))$.

3. The input tgds are in normal form, i.e., each tgd uses distinct variables, and no tgd can be decomposed in two different tgds having the same left-hand side.

For a schema mapping $\mathcal{M}$ the incremental schema mapping $\mathcal{M}'$ (see Definition 4.1) follows these preconditions. Note that if the tgds of $\mathcal{M}$ are source-based and in normal form, then the tgds of $\mathcal{M}'$ are also source-based and in normal form.

The algorithm of ++Spicy We now describe the main intuition behind the algorithm of ++Spicy. The algorithm of Mecca et al. is based on *rewriting* the source-to-target tgds using equivalent expressions. In particular, they identify pairs of tgds where one *subsumes* the other. A tgd m *subsumes* an other tgd m' if there exists a homomorphism between the right-hand sides (conclusions) of the two tgds, which in a high-level means that after executing m, the execution of m' will add redundant tuples to the target schema. Then for each such pair, all tuples are generated for m (the more informative tgd), whereas for m' the tgd is rewritten so that only those tuples that add new content to the target are included. These tgds are rewritten using negation and each one of them can be expressed as a relational algebra expression using differences and unions. There are also some more complex subsumption scenarios which the authors call *coverages* and the main complexity at hand is the fact that there are several possible rewritings for the subsumed tgds[6].

An important advantage of having the tgds expressed in relational algebra the fact that these rewritten tgds can be directly translated to SQL and utilize the power and scalability of a relational database engine (e.g. Postgres).

[5]The key constraints can be applied to the final SQL script.

[6]These scenarios are simplified by foreign key constraints. A foreign key constraint can turn a coverage case into a simple subsumption case.

4.3 IVM Expressions as Conjunctive Queries

The algorithm with which we compute the core is rewriting *conjunctive queries* to more complex expressions. For a given mapping $\mathcal{M}$ as defined in subsection 4.1 and its equivalent IVM mapping $\mathcal{M}$, while an SPJ expression is always a conjunctive query, this is not true for its respective IVM counterpart (defined in section 3). For example, notice that the expression for inner joins contains disjunctions.

This limitation in expressiveness forces us to do some reformulation of the source schema when implementing SPJ expressions. We present our approach separately for selections, projections and inner joins. Let S_1, S_2 be two relations of the source schema S and $\Delta S_1, \Delta S_2$ their incremental additions.

1. **Selections** $\sigma(\cdot)$

   The relational expression representing a selection $\sigma(S_1)_{expr}$, where *expr* is a boolean expression on the tuples of S_1 is transformed into $\sigma(\Delta S_1)_{expr}$. In this case, the IVM expression is transformed to a conjunctive query. Therefore, no reformulation of the source schema is necessary.

2. **Projections** $\pi(\cdot)$

   The relational expression representing a projection $\pi(S_1)_X$ where X is a subset of the attributes of S_1 is transformed to $\pi(\Delta S_1)_X - \pi(S_1)_X$. This expression requires a negation, hence it is not a conjunctive query.

   To overcome this issue we add a new relation to the source schema called S^-, the data of which is the result of the relational expression $\pi(\Delta S_1)_X - \pi(S_1)_X$. Then, after computing this new relation, the conjunctive query we run is $\pi(S^-)_*$.

3. **Inner Joins** $(\cdot) \bowtie (\cdot)$

   An inner join $S_1 \bowtie S_2$ expression is transformed to $((S_1 \cup \Delta S_1) \bowtie \Delta S_2) \cup ((S_2 \cup \Delta S_2) \bowtie \Delta S_1)$, a *disjunction* of two inner joins.

   Our approach is to add the new relations $S_1^{\cup}$, $S_1^{\cup}$ which contain the data from $(S_i \cup \Delta S_i), i = 1, 2$. Then, the resulting expression $(S_1^{\cup} \bowtie \Delta S_2) \cup (S_2^{\cup} \bowtie \Delta S_1)$ becomes a disjunction of two conjunctive expressions. We then treat this expression as two separate conjunctive expressions.

5 Implementation in MIPMAP

The main abstraction that represents a mapping task in MIPMAP is the class `MappingTask`. We provide an extension of the class named `IncrementalMappingTask` with the following constructor:

```
IncrementalMappingTask(MappingTask baseMappingTask,
                       DataSource deltaSource)
```

The constructor's signature implies that one can instantiate an IVM mapping task by providing:

1. An other `MappingTask` which represents the data exchange scenario which we are going to transform to a logically equivalent one (as described in section 4).

2. A `Datasource`, which represents the data of the new batch, i.e. for every relation S its table ΔS. Note that the schema of `deltaSource` must be identical to the one outputted by `baseMappingTask.getDataSource().getSchema()`. Otherwise, an `IVMException` is thrown.

Note that the interface of this class is identical to the one of the class `MappingTask`. The `IncrementalMappingTask` is essentially the implementation of

$$\mathcal{M}' =< \mathbf{S} \cup \triangle\mathbf{S} \cup \mathbf{S}^{-}, \mathbf{T}, \Sigma_{st} \cup \Sigma_{st}^{\Delta}, \Sigma_t >$$

and it is generated using the `baseMappingTask`

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

and the differential source data ΔS represented by the instance `deltaSource`. Therefore to generate $\mathcal{M}'$ we follow the following two step-approach.

1. Firstly, we need to represent the view of source data of $\mathcal{M}'$ that is conceptually $< \mathbf{S} \cup \triangle\mathbf{S} \cup \mathbf{S}^{-} >$. We do so by introducing a new type of a synthetic `DataSourceProxy` to MIPMAP which we call `IncrementalDataSourceProxy`. This new class is essentially a chaining of three classes of the type `DataSourceProxy`: one for the data in $\mathbf{S}$, one for the data in $\Delta\mathbf{S}$ and one for $\mathbf{S}^{-}$.

2. After the computation of an instance of `IncrementalDataSourceProxy`, we rewrite the initial value correspondences of $\mathcal{M}$ so that when the tgds of $\mathcal{M}'$ are generated by MIPMAP they represent precisely $\Sigma_{st} \cup \Sigma_{st}^{\Delta}$ as defined in the previous section. The new `IncrementalMappingTask` contains these rewritten mapping correspondences.

An `IncrementalMappingTask` is nothing more but a regular `MappingTask` for MIPMAP satisfying the preconditions of the algorithm (see section 4.2). For MIPMAP developers, there is no change in terms of APIs at all. By construction, it is an equivalent mapping task to the one it is based on. However, this new mapping task generates an executable SQL script which is superior in terms of performance in scenarios of incremental data availability (see next section).

6 Experiments

You have given me guidelines on this. More on this very soon, that is if you are satisified with the remainder of the document.

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.