Reverse Data Management

Alexandra Meliou Wolfgang Gatterbauer Dan Suciu
Department of Computer Science and Engineering,
University of Washington, Seattle, WA, USA
{ameli,gatter,suciu}@cs.washington.edu

ABSTRACT

Database research mainly focuses on forward-moving data flows: source data is subjected to transformations and evolves through queries, aggregations, and view definitions to form a new target instance, possibly with a different schema. This *Forward Paradigm* underpins most data management tasks today, such as querying, data integration, data mining, etc. We contrast this forward processing with *Reverse Data Management (RDM)*, where the action needs to be performed on the input data, on behalf of desired outcomes in the output data. Some data management tasks already fall under this paradigm, for example updates through views, data generation, data cleaning and repair. RDM is, by necessity, conceptually more difficult to define, and computationally harder to achieve. Today, however, as increasingly more of the available data is derived from other data, there is an increased need to be able to modify the input in order to achieve a desired effect on the output, motivating a systematic study of RDM.

We define the Reverse Data Management problem, and classify RDM problems into four categories. We illustrate known examples of RDM problems and classify them under these categories. Finally, we introduce a new type of RDM problem, *How-To Queries*.

1. DATA TRANSFORMATIONS

Informally, a *data transformation* consists of a function from an input data source to an output data source. The natural evolution of data follows the directionality of the transformations, i.e. from source to target. Most data management tasks fall under this forward paradigm from a variety of perspectives: query processing, data integration, data mining, clustering and indexing.

We study here a class of problems that focus on the reverse direction, i.e. against the direction of the data transformation (Fig. 1). In these problems one wants achieve a certain effect in the output data, and needs to act on the input data in order to achieve that effect. Examples include updating through views [12], data generation [8], causality computation [26], data cleaning [3]. We thus refer to these areas under the common term of *Reverse Data Management*, or RDM. Thus, RDM consists of the problems where one needs to compute a database input, or modify an existing database input, in order to achieve a desired effect in the output.

Proceedings of the VLDB Endowment, Vol. 4, No. 12

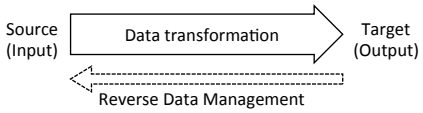

Figure 1: Reverse Data Management reasons from a desired output instance or specification, to the required input.

All these problems share a common premise: they essentially reverse a transformation in order to achieve a desired target instance, or target properties. Our goal in this paper is to identify the commonalities and differences among these problems, based on the specifications of the problem requirements, and to propose a systematic study of RDM. We define a taxonomy of RDM problems, categorizing them into four groups, allowing us to identify and describe a new type of RDM that we call *How-To* queries. Often users, administrators, and analysts are interested in changing their data in ways that would achieve certain conditions and constraints: "What advertisements will result in the best sales increase, at a cost bounded by X?", "How can I increase my clients' return on investment with the minimum number of trades?". How-to queries come as a natural extension of the reverse management space as we will observe in the following section, and are useful for strategy decisions and for modeling various data optimization problems.

RDM is more difficult to define and to implement than direct data management, because of the simple fact that the inverse of a function is not necessarily a function. Given a desired output, or a desired change of the output, there are multiple inputs (or none at all) that can satisfy it. This difficulty shows up in all RDM problems: in updates through views one restricts the view to the most simple *updateable views* [12] or searches for updates that minimize the number of side-effects [10]; data cleaning and repair often results in NP-hard problems [3, 24]; in causality too one has to compute an NP-hard problem to find the causes of an output [26].

To circumvent this difficulty in a general framework, we propose the adoption of SAT or MaxSAT solvers as general purpose tools in RDM. These solvers handle tens of thousands of variables and clauses, and for many practical instances even millions (cf. [1]), making them attractive for RDM, and have recently been deployed for specific RDM problems [27]. We posit that, by using SAT or MaxSAT as primitives (oracles), many of the RDM problems become tractable in practice.

2. THE PROBLEM SPACE FOR RDM

We classify RDM problems along two dimensions.

Target Data. We distinguish between *explicit target* and *implicit target* specifications. In the case of an explicit target, the output is

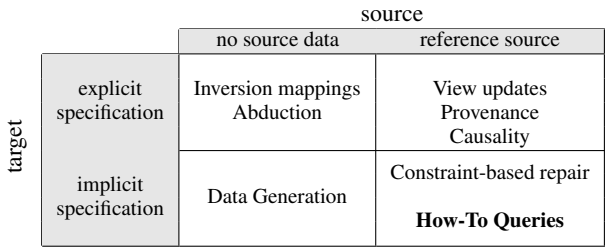

Figure 2: The space of Reverse Data Management, seen under source and target specifications.

given as a specific data instance. Sometimes there is a distinction between different versions of the target (e.g. before and after a view update), but it always involves tuple-level instances. In the case of an implicit target, the output is described indirectly, through constraints, or through statistics, like in constraint-base data cleaning, or in declarative data generation. Since statistics and other constraints can also be viewed as transformations on the data, implicit descriptions can be transformed into explicit ones. However, we prefer to preserve the distinction, as we view the two cases as conceptually different: one is based on specific tuples, while the other on collective measures over an instance. In the former case, we know exactly what target data we aim for; in the latter case we only desire some general effect, usually given through some constraints, or aggregate statistics. Thus, the two cases differ in what restrictions they impose on the target data. Since the reverse data management paradigm involves a "backward" derivation, from target to source, some specification of the target data always needs to be part of the problem description.

Source Data. We distinguish between RDM problems *with a reference source data* and *without source data*. In the first case we have a source data and we want to modify it to reflect a desired effect on the output: updates through views is the classic example. In the second case, we have to compute the entire input from scratch: the goal of *inverse schema mapping* is to give a simple tool to compute the input from the output [18]; another example is data generation, where we do not have any input and need to construct one from scratch.

This classification of the two dimensions results in four types of RDM problems, which are illustrated in Fig. 2. Figure 2 depicts the reverse data management domain using the distinctions we made on source and target constraints. We classify some RDM problems using their most common form, but note that some of them could potentially be classified differently based on variations of the problem description. This exercise helps us identify an interesting new direction of this line of research that we call *How-To* queries, which we motivate and describe in the following section. We will cover in more detail the examples mentioned in Fig. 2, but this is not meant to be an exhaustive list. In the next section we describe a specific RDM problem that has not yet been explored in literature, which falls under the fourth category.

View Updates. In the various flavors of the view update problem ([10, 12, 13]), a source dataset is given, along with a query that specifies a view over the data. In the classical formulation, the goal is to describe the class of views that are updateable. In other formulations of the problem, one wants to modify the source data to achieve the desired update, without introducing additional side-effects (or by minimizing them) [10]. The target data in this case is the view itself, and the problem is to determine how updates, insertions, and deletions to the view would be reflected to the source data. Thus, there is a reference source, and the output is explicit.

Data Provenance; Database Causality. While data provenance ([11, 9, 20]) is computed "forwards", from the source to the target, its purpose is often to enable users to trace information backwards: given an output tuple, its provenance describes which input tuples have contributed to it. Both target and source data are given in this problem statement, and the goal is to select only the relevant parts of the source that correspond to the target tuples of interest. *Database Causality* [26, 25] is a refinement of data provenance, where the target tuples may deviate from what the user expects (unexpected tuples appear in the result, or expected ones do not), and the goal is to select the appropriate source tuples as causes of given target tuples. In both cases, the problem statement specifies an *explicit* target where the output is given as a specific data instance, and a source data instance from which the proper parts will be selected.

Inversion Mappings. Data exchange deals with a problem that often arises in cases of data integration, where we need to transform data from one schema to another. A schema mapping is a specification that describes how data from a source schema $\mathcal{A}$ is to be mapped to a target schema $\mathcal{B}$. The inversion of a mapping ([4, 17, 2, 5]) describes the reverse transformation from $\mathcal{B}$ to $\mathcal{A}$, the goal being to recover the original source data. Various definitions of schema mappings and their inversions exist, but in general we still have an explicit specification of the target dataset, however, we do not necessarily have any reference source as was the case in the problems we've covered so far.

Abduction. Abduction [15, 28] is a related problem where the goal is to find hypothetical explanations for an observed consequence. In the general case no reference source needs to be assumed, but abduction techniques have also been linked to repairs [6] and view updates [22] (possibly classifying it in a different box).

Data Generation. In order to test correctness and performance of algorithms and systems, synthetic data is often necessary, as it allows arbitrary scale up, and exhaustive exploration of properties and parameters that can not always be found in real data. Generating synthetic data with meaningful characteristics that sufficiently resembles real data behavior is a very hard and challenging problem in database research ([21, 19, 8]). In data generation, there is usually no reference source data, but rather the data needs to be generated from scratch, commonly based on specific statistics. Some common statistics involve the size of relations, number of unique attribute values, as well as correlations between them. A valid solution to the data generation problem is source data that satisfies all constraints given by the problem description. As opposed to the previous problem settings, the target here is *implicitly* specified using statistics and constraints, rather than a fully defined data instance.

Constraint-based Repair. In this problem we are given a source database instance and a target constraint, such as a key constraint or some conditional functional dependencies. The goal is to repair the source data in order to satisfy the constraint [3]. While the source data exists (reference source), there is no target data, instead there is only a target constraint, so the target specification is implicit.

3. HOW-TO QUERIES

How-to queries are motivated from a related research problem within the forward processing paradigm: *what-if* or *hypothetical* queries [23, 7]. They use source and target data to ask questions of the form "How would the output change for a given change in the source?". They are motivated by a variety of business applications that require strategy evaluation and decisions. However, it is more meaningful for such applications to be treated under the reverse framework of how-to queries, i.e. "How should the input change in

order to achieve the desired output?". Here, we give motivating applications for how-to queries, overview the challenges of this new direction, and give some early suggestions for solutions.

3.1 Example Applications

A strong motivation for how-to queries comes from the domain of business intelligence [16, 14, 29]. Key performance indicators are commonly extracted from source data using complex aggregate functions and those may be evaluated against different strategies. Common questions in this domain are of the form "How could we maintain revenue growth while still reduce risks (at least partially)?", "Should we increase sales volume by opening new sales channels, or offer promotions, or both?".

EXAMPLE 3.1 (PORTFOLIO ANALYSIS). *An analyst at a brokerage company wants to investigate strategies that could achieve better returns and volatility of customer portfolios, based on the company's recommendations during the last three years. He would like to receive a list of possible modifications to the company's stock recommendations, that would achieve the desired output in the customer's portfolios (e.g. 10% return). Out of all the possible scenarios, the analyst wants to give preference to those that are closest to the company's current strategy as they would require fewer trades.*

Example 3.1 is a modification of an example that appears in [7]. Note the fundamental difference between this example and the one geared towards hypothetical queries: in contrast with [7] where the analyst manually selects relevant hypotheses for testing, in our how-to setting the relevant scenarios are automatically selected based on the specifications on the target data. Along with the target restrictions on the portfolio returns, *optimization criteria* can also be used to restrict the solution space (e.g. minimize the number of trades). These are used to define a distance metric from the reference source (the current portfolios), and pick the solution closest to it.

EXAMPLE 3.2 (COMPANY REORGANIZATION). *A software company going through some financial strain hires consultants to help reduce operational costs. The consultants may suggest layoffs, salary decreases, or department and project merging, within certain constraints specified by the company's requirements. For instance, any salary decreases should be uniform across employees of the same department, every project should have at least a certain number of employee hours devoted to it, and the solution should be achieved with the minimum number of employee reassignments.*

How-to queries can be used to describe in an abstract way datasets that satisfy a set of constraints, and can find applications in a variety of domains beyond business intelligence tools.

EXAMPLE 3.3 (SHIPMENT CONSOLIDATION). *A product supplier receives orders for various products from different clients. He would like to minimize costs by consolidating shipments to the same client. However, the supplier needs to make sure that all orders arrive within the agreed delivery window, and no shipment exceeds the maximum order size.*

EXAMPLE 3.4 (RESOURCE UTILIZATION). *A system administrator has access to system logs of a server cluster with information on resource allocation and utilization, and job arrival, execution, and waiting times. During peak times within the day the system can become overloaded, and job wait times reach undesirable highs. The administrator wants to determine which machines in the cluster should be kept in operation so that job wait times are bounded by a small constant, and throughput remains high, while minimizing the operational cost of the cluster (there is a cost associated with keeping a machine in operation).*

As seen through our examples, how-to queries have implicit target specifications, and operate on a reference source. The target specifications may be simple constraints on the target instance (e.g. no salary should exceed \$150k), or *differential* constraints that restrict the amount of modification from the current instance (e.g. no salary should face more than 10% reduction). Commonly, an *objective function* will be defined in relation to the source instance, which defines a score for the feasible solutions, allowing us to pick the one that minimizes or maximizes the criterion.

3.2 Challenges

Based on the motivation examples of Sect. 3.1, we identify several desiderata for the implementation of how-to queries:

- Support for defining simple target as well as differential constraints. For example, a cost may be defined based on a constant value (e.g. $< 10k$), or in reference to a current cost metric (e.g. 10% reduction in cost).
- Support for defining optimization criteria as objective functions (e.g. minimize the number of required trades).
- Support value updates, as well as insertions and deletions to the reference source data.
- Maintain declarativity, possibly through SQL extensions.
- Efficient evaluation. In order to be useful, how-to queries should be computed efficiently.

We next discuss some first ideas on tackling these problems.

Language. It is natural to consider SQL extensions to support how-to queries. The language should allow us to express constraints, objective functions, and the how-to query itself. We will use Example 3.1 as a reference point, and will give sample syntax constructs that could describe it. We start with defining target constraints over the `Portfolios` relation containing among other fields a unique `custID` for each customer, a `curValue` and `buyPrice` for each transaction. The constraint will resemble assertion declarations, and its function will be similar.

```
CREATE CONSTRAINT Constr1
AS NOT EXISTS
   (SELECT   sum(curValue) as v, sum(buyPrice) as p
    FROM     Portfolios
    GROUP BY custID
    HAVING   v < 1.1*p)
```

The above constraint ensures that every customer receives a return of at least 10% on the total of their investments. As a simplification we excluded trading costs.

Objective functions could be defined in a similar fashion, but define a minimization or maximization objective instead of a strict constraint. The statement below is a simplified version of the objective of Example 3.1. Note the use of _O and _N, refering to "old" and "new", in reference to the relation `Portfolios`: when defining a differential constraint or objective that compares to the current reference source, we need to differentiate between the original source data (`Portfolio_O`) and the how-to result (`Portfolio_N`).

```
CREATE OBJECTIVE Obj1
AS SELECT count(*)
   FROM   ((SELECT distinct stock, custID
             FROM   (Portfolios_O UNION Portfolios_N))
           EXCEPT
           (SELECT distinct O.stock, O.custID
            FROM   Portfolio_O O, Portfolio_N N
            WHERE  O.stock=N.stock
                   AND O.custID=N.custID))
```

The objective above counts the total number of trades[1] required to modify the reference source relation to a new candidate solution `Portfolios_N`. The analyst can then issue a how-to query as follows:

```
HOW TO      minimize(Obj1)
SUBJECT TO Constr1
```

There are further challenges relating to the language definition. One important component is allowing the user to specify the types and areas of permissible modifications. For example, a user may only want solutions that modify a certain relation and not another, or even restrict modifications to specific attributes. This is directly related to the concept of endogenous and exogenous variables introduced in [26].

Evaluation. A major challenge of how-to queries is making their evaluation fast. A naive approach would be to generate multiple hypothetical queries, evaluate those, and select as a solution the one that satisfies the how-to constraints and is optimal based on the objective function. This straw man approach however is not expected to get us far, as even the evaluation of hypothetical queries is itself hard. We believe that the solution lies in the appropriate reduction of a how-to query to a clean mathematical optimization problem that can be solved with other existing techniques. Our recent work on causality [27] (which is also a reverse data management problem), has shown that such reductions hold great potential as specialized SAT solvers have been optimized over the years to tackle even hard problems very efficiently[2]. Specifically for how-to queries, we will also need to study reductions to linear, integer, and quadratic programming, as they are a basic component of constrained optimizations. Such a reduction will not be straightforward, as the choice of variables, as well as the translation of the declarative objectives and constraints to mathematical equations is not obvious.

4. CONCLUSIONS

In this vision paper we discuss the paradigm of *Reverse Data Management*, and identify problems that fall within this domain. We provide insights on the similarities and differences of various data management subfields within this context and identify a challenging new problem. We describe the concept of *how-to queries*, establish a practical motivation, and introduce some first challenges, as well as promising directions.

Acknowledgements. This work was partially funded by NSF IIS-0911036, IIS-0915054, and IIS-0713576.

5. REFERENCES

[1] SAT Competition webpage: http://www.satcompetition.org/.

[2] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Relational and XML Data Exchange*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[3] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pp. 68–79, 1999.

[4] M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Composition and inversion of schema mappings. *SIGMOD Rec.*, 38(3):17–28, 2009.

[5] M. Arenas, J. Pérez, and C. Riveros. The recovery of a schema mapping: Bringing exchanged data back. *ACM Trans. Database Syst.*, 34(4), 2009.

[6] O. Arieli, M. Denecker, B. V. Nuffelen, and M. Bruynooghe. Coherent integration of databases by abductive logic programming. *J. Artif. Intell. Res. (JAIR)*, 21:245–286, 2004.

[1]The objective is simplified to only consider stocks by name, and not their number in the customer portfolio.

[2]Modern SAT solvers can solve problems with millions of variables and clauses in less than 10 minutes of run time (cf. [1]), and they can therefore provide a viable solution to some theoretically hard problems in our field.

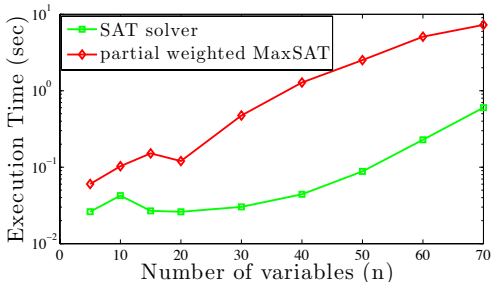

Figure 3: SAT solver performance as demonstrated in [27].

[7] A. Balmin, T. Papadimitriou, and Y. Papakonstantinou. Hypothetical queries in an OLAP environment. In *VLDB*, pp. 220–231, 2000.

[8] N. Bruno and S. Chaudhuri. Flexible database generators. In *VLDB*, pp. 1097–1107, 2005.

[9] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pp. 316–330, 2001.

[10] P. Buneman, S. Khanna, and W. C. Tan. On propagation of deletions and annotations through views. In *PODS*, pp. 150–158, 2002.

[11] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[12] S. S. Cosmadakis and C. H. Papadimitriou. Updates of relational views. In *PODS*, pp. 317–331, 1983.

[13] U. Dayal and P. A. Bernstein. On the correct translation of update operations on relational views. *ACM Trans. Database Syst.*, 7(3):381–416, 1982.

[14] L. Dolman, F. Tompa, I. Kiringa, R. Pottinger, and J. Mylopoulos. Next generation business intelligence (bi) tools. In *CASCON*, pp. 352–354, 2010.

[15] T. Eiter and G. Gottlob. The complexity of logic-based abduction. *J. ACM*, 42(1):3–42, 1995.

[16] N. A. Ernst, J. Mylopoulos, A. Borgida, and I. Jureta. Reasoning with optional and preferred requirements. In *ER*, pp. 118–131, 2010.

[17] R. Fagin. Inverting schema mappings. *ACM TODS*, 32(4), 2007.

[18] R. Fagin and A. Nash. The structure of inverses in schema mappings. *J. ACM*, 57(6):31, 2010.

[19] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger. Quickly generating billion-record synthetic databases. In *SIGMOD*, pp. 243–252, 1994.

[20] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pp. 31–40, 2007.

[21] K. Houkjær, K. Torp, and R. Wind. Simple and realistic data generation. In *VLDB*, pp. 1243–1246, 2006.

[22] A. C. Kakas and P. Mancarella. Database updates through abduction. In *VLDB*, pp. 650–661, 1990.

[23] L. V. S. Lakshmanan, A. Russakovsky, and V. Sashikanth. What-if olap queries with changing dimensions. In *ICDE*, pp. 1334–1336, 2008.

[24] A. Lopatenko and L. E. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *ICDT*, pp. 179–193, 2007.

[25] A. Meliou, W. Gatterbauer, J. Y. Halpern, C. Koch, K. F. Moore, and D. Suciu. Causality in databases. *IEEE Data Eng. Bull.*, 33(3):59–67, 2010.

[26] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010.

[27] A. Meliou, W. Gatterbauer, S. Nath, and D. Suciu. Tracing data errors with view-conditioned causality. In *SIGMOD*, pp. 505–516, 2011.

[28] T. Menzies and P. Compton. Applications of abduction: hypothesis testing of neuroendocrinological qualitative compartmental models. *Artificial Intelligence in Medicine*, 10(2):145–175, 1997.

[29] R. Sebastiani, P. Giorgini, and J. Mylopoulos. Simple and minimum-cost satisfiability for goal models. In *CAiSE*, pp. 20–35, 2004.