MuSe: Multiple Deletion Semantics for Data Repair

Amir Gilad[1], Yihao Hu[2], Daniel Deutch[1], Sudeepa Roy[2]
[1]Tel Aviv University, [2]Duke University
amirgilad@mail.tau.ac.il, yihao.hu@duke.edu, danielde@post.tau.ac.il, sudeepa@cs.duke.edu

ABSTRACT

We propose to demonstrate MuSe, a system for Database repairs where constraints are expressed as *Declarative Rules* and can be interpreted in different ways by using four different semantics. Our framework may capture common, cross-relation, repair semantics such as that of SQL deletion triggers, causal rules, and denial constraints. Our demonstration will show the usefulness of the system in easing specification of database repair policies, for different use cases.

PVLDB Reference Format:
Amir Gilad, Yihao Hu, Daniel Deutch, Sudeepa Roy. MuSe: Multiple Deletion Semantics for Data Repair. *PVLDB*, 13(12): 2921-2924, 2020.
DOI: https://doi.org/10.14778/3415478.3415509

1. INTRODUCTION

Database repair through tuple deletion has been extensively studied [2]. In particular, repairing the database using a minimum number of deletions is a desired feature [1, 4]. There are various approaches for database repair for different use cases and scenarios. These include repairing via integrity constraints, such as Denial Constraints (DCs) [2], and expressing dependencies between relations via causal rules [10] or SQL triggers. All of these approaches have specific semantics associated with them; namely, DCs point to a set of tuples violating a constraint, but do not specify which tuple in this set should be removed. On the other hand, causal rules and SQL triggers also point to a specific tuple that should be deleted and allow for cascade deletions. However, even for the latter there is no single accepted semantics for cases where several triggers are satisfied at the same time: one approach may be to fire the triggers according to lexicographic order [9], while another is to fire them in the order in which they were created [8].

EXAMPLE 1.1. *Consider the database in Figure 1 based on Microsoft Academic Search Database [7]. It includes the tables* `Grants` *(grant foundations),* `Author` *(paper authors),* `AuthGrant` *(a relationship of authors and grants given by a foundation),* `Pub` *(a publication table),* `Writes` *(a connecting table between* `Author` *and* `Pub`*), and* `Cite` *(a citation table of citing and cited relationships). For each tuple, we also have an identifier on the leftmost column of each table (e.g.,* $\mathtt{ag}_1$ *is the identifier of* `AuthGrant`$(2,1)$*). Consider the following three constraints specifying how to repair the tables (there could be other rules capturing different repair scenarios):*

1. *If a* `Grants` *tuple is deleted and there is an author who was awarded a grant by this foundation, denoted as an* `AuthGrant` *tuple, then delete the winning author.*
2. *If an* `Author` *tuple is deleted and the corresponding* `Writes` *and* `Pub` *tuples exist in the database, delete the corresponding* `Writes` *tuple (as in cascade delete semantics for foreign keys). Under the same condition as above, delete the corresponding* `Pub` *tuple (not standard foreign keys, but suggesting that every author is important for a publication to exist).*
3. *If a publication p from the* `Pub` *table is deleted, and is cited by publication c, while some authors of these papers still exist in the database, then delete the* `Cite` *tuple.*

Suppose we are analyzing a subset of this database containing only authors affiliated with U.S. schools. ERC grants are given only to European institutions and its `Grants` *tuple was incorrectly added to the U.S. database, so this tuple needs to be deleted. However, this deletion causes violations in the above constraints. To repair the database based on these constraints, we could proceed in various ways: considering the semantics of triggers and causal rules, we can delete tuples* $\mathtt{a}_2$, $\mathtt{w}_1$, $\mathtt{p}_1$, $\mathtt{a}_3$, $\mathtt{w}_2$, $\mathtt{p}_2$ *and* `c`*, and regain the integrity*

Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415509

Grants

	gid	name
$\mathtt{g}_1$	1	NSF
$\mathtt{g}_2$	2	ERC

AuthGrant

	aid	gid
$\mathtt{ag}_1$	2	1
$\mathtt{ag}_2$	4	2
$\mathtt{ag}_3$	5	2

Author

	aid	name
$\mathtt{a}_1$	2	Maggie
$\mathtt{a}_2$	4	Marge
$\mathtt{a}_3$	5	Homer

Cite

	citing	cited
c	7	6

Writes

	aid	pid
$\mathtt{w}_1$	4	6
$\mathtt{w}_2$	5	7

Pub

	pid	title
$\mathtt{p}_1$	6	x
$\mathtt{p}_2$	7	y

Figure 1: Academic database instance D

(0) $\Delta_{\mathtt{Grants}}(g,n)$:− `Grants`(g,n), n = 'ERC'
(1) $\Delta_{\mathtt{Author}}(a,n)$:− `Author`(a,n), `AuthGrant`(a,g), $\Delta_{\mathtt{Grants}}(g,gn)$
(2) $\Delta_{\mathtt{Pub}}(p,t)$:− `Pub`(p,t), `Writes`(a,p), $\Delta_{\mathtt{Author}}(a,n)$
(3) $\Delta_{\mathtt{Writes}}(a,p)$:− `Pub`(p,t), `Writes`(a,p), $\Delta_{\mathtt{Author}}(a,n)$
(4) $\Delta_{cite}(c,p)$:− `Cite`(c,p), $\Delta_{\mathtt{Pub}}(p,t)$, `Writes`$(a_1,c)$, `Writes`$(a_2,p)$

Figure 2: Delta program

of the database but at the cost of deleting seven tuples. A different approach is to delete $\mathtt{a}_2$ *and either* $\mathtt{w}_1$ *or* $\mathtt{p}_1$*, and delete* $\mathtt{a}_3$ *and either* $\mathtt{w}_2$ *or* $\mathtt{p}_2$*, which would only delete four tuples. However, if we consider the semantics of DCs, we could delete any tuple out of the set of tuples that violates the constraint. So, we can just delete the tuples* $\mathtt{ag}_2, \mathtt{ag}_3$*. This would satisfy the first constraint and thus the second, third and fourth constraints will also be satisfied.*

To account for these different approaches and to provide clear and reliable semantics for database repair with deletions, *we propose to demonstrate* `MuSe`*, a novel unified framework for database repair through tuple deletions that includes four different semantics of minimum repairs* based on [5]. Constraints are expressed as *delta rules* that resemble datalog rules. The rules can be used with different semantics. For instance, *independent semantics* emulates DCs, but also allows to account for violations that include tuples deleted subsequently, and aims for a "global" minimum repair. *Step semantics* only deletes tuples in the head of rules (constraints) but aims for the minimum number of deletions through one tuple deletion at a time, via the head of the rules. *Stage semantics* is similar to step semantics in its interpretation of the rules and only allows for deletions of derived tuples, but instead, derives all tuples of satisfied rules at each stage and deletes them at the end of the stage, until no more tuples can be derived. *End semantics* is based on standard Datalog evaluation and derives all possible tuples first, and only deletes them at the end of the evaluation process (see Section 2).

Through `MuSe` we make this framework accessible by allowing users to formulate delta rules in different ways: users may formulate Functional Dependencies or Foreign Key Dependencies and `MuSe` will automatically translate them into delta rules and display them in real time, or they can gradually build a rule assisted by the system that guides them in every step. Additionally, before the actual repair occurs, `MuSe` presents a summary of the number of tuples that will be deleted under each of the four semantics. Finally, `MuSe` also provides explanations for the repairs under the different semantics, showing the process that resulted in a specific tuple deletion. Then, users can choose whether they wish to commit the repair to the database or try a different semantics. In this paper, we give an overview of these semantics, algorithms, and discuss the proposed system and demonstrations. More details including theoretical and experimental results, and related work with references can be found in [5].

2. RULE-BASE REPAIR FRAMEWORK

Delta rules: We consider a schema that includes (1) the standard relations of the database schema $\mathbf{R} = \{R_1, \ldots, R_k\}$, and (2) the corresponding *delta relations* maintaining records of deleted tuples $\mathbf{\Delta} = \{\Delta_1, \ldots, \Delta_k\}$. When a delta rule is satisfied, its derivation deletes a tuple $R_i(a)$ and it adds its record of deletion to the corresponding delta relation $\Delta_i(a)$. The syntax of delta rules resembles a datalog rule with a delta atom at the head of the rule. Intuitively, when satisfied, a delta rule is designed to delete the tuple at its head. For instance, in Figure 2, rule (2) is meant to delete any `Pub` tuple after its `Author` tuple has been deleted. We have $\Delta_{\mathtt{Pub}}(p, t)$ in the head of the rule and in the body we have the atom $\mathtt{Pub}(p, t)$ to ensure the deleted tuple exists in the database.

Database stabilization: Given a database that satisfies some of the delta rules, our aim is to regain the integrity of the database, or to delete tuples until no delta rule is satisfied. When the database is in this state, we call it *stable*. In Figures 1 and 2, if we remove the tuples $\mathtt{g}_2$, $\mathtt{a}_2$, $\mathtt{a}_3$, $\mathtt{w}_1$, $\mathtt{w}_2$, $\mathtt{p}_1$, $\mathtt{p}_2$, $\mathtt{c}$ and add the corresponding delta tuples $\Delta(\mathtt{g}_2)$, $\Delta(\mathtt{a}_2)$, $\Delta(\mathtt{a}_3)$, $\Delta(\mathtt{w}_1)$, $\Delta(\mathtt{w}_2)$, $\Delta(\mathtt{p}_1)$, $\Delta(\mathtt{p}_2)$, $\Delta(\mathtt{c})$, we would have a stable database, as none of the rules are satisfied.

We now discuss the definitions of the four semantics. Given a delta program P and a database D, we denote the result of a semantics σ by $\sigma(P, D)$.

Independent semantics: The result of this semantics is a globally optimal solution, i.e. a minimal-size set of tuples that need to be deleted to make all the rules unsatisfied. This semantics correspond to the standard DC semantics where a set of tuples violates a constraints and anyone of the violating tuples may be deleted. In Figures 1 and 2, the result of independent semantics is $\{\mathtt{g}_2, \mathtt{ag}_2, \mathtt{ag}_3\}$. Note that after removing these tuples and adding their delta counterparts, there are no satisfying assignments to any of the rules.

Step semantics: This semantics resembles SQL triggers as it allows for cascade deletion, yet it is a fine-grained semantics that evaluates one rule at a time and updates the database immediately by deleting the original tuple and adding the derived delta tuple. If there is more than one satisfied rule in some step, this semantics makes a nondeterministic choice of which rule to fire. Reconsider the database in Figure 1 and the rules in Figure 2.

In our running example, the following is a possible sequence of activations of rules under step semantics: *At step* 1, there is one satisfying assignments to rule (0) deriving $\Delta(\mathtt{g}_2)$. We update $\Delta^1_{\mathtt{Grants}} = \{\mathtt{g}_2\}$, $\mathtt{Grants}^1 = \{\mathtt{g}_1\}$ *At step* 2, there are two satisfying assignments to rule (1). We choose the assignment to rule (1) deriving $\Delta(\mathtt{a}_2)$, and update D^1 so it includes the change $\Delta^2_{\mathtt{Author}} = \{\mathtt{a}_2\}$, $\mathtt{Author}^2 = \{\mathtt{a}_1, \mathtt{a}_3\}$ *At step* 3, we have three satisfying assignments: to rules (1), (2), and (3). Suppose we choose the one satisfying rule (1) and derive $\Delta(\mathtt{a}_3)$. D^2 is now updated such that $\Delta^3_{\mathtt{Author}} = \{\mathtt{a}_2, \mathtt{a}_3\}$, $\mathtt{Author}^3 = \{\mathtt{a}_1\}$, and so on. Continuing this process we can get a stable database by removing $\{\mathtt{g}_2, \mathtt{a}_2, \mathtt{a}_3, \mathtt{w}_1, \mathtt{w}_2\}$.

Stage semantics: Stage semantics resembles semi-naive evaluation of datalog, and fires all satisfied rules in each stage based on the previous stage of the database (instead of choosing one rule like step semantics and therefore is deterministic). After deriving all possible tuples at a specific stage, this semantics updates the database.

Reconsider the database in Figure 1 and the rules in Figure 2. In our running example, stage semantics works as follows: *At the first stage*, there is one assignments to rule (0) deriving $\Delta(\mathtt{g}_2)$, we update $\Delta_{\mathtt{Grants}} = \{\mathtt{g}_2\}$, $\mathtt{Grants} = \{\mathtt{g}_1\}$. *At the second stage*, we use the two assignments to rule (1) to derive $\Delta(\mathtt{a}_2)$ and $\Delta(\mathtt{a}_3)$. We update the database so that $\mathtt{Author} = \{\mathtt{a}_1\}$, $\Delta_{\mathtt{Author}} = \{\mathtt{a}_2, \mathtt{a}_3\}$. *In the next stage*, we use the two assignments to rule (2) and the two assignments to rule (3) to derive $\Delta(\mathtt{p}_1)$, $\Delta(\mathtt{p}_2)$, $\Delta(\mathtt{w}_1)$ and $\Delta(\mathtt{w}_2)$, and update the database as $\mathtt{Writes} = \emptyset$, $\mathtt{Pub} = \emptyset$, $\Delta_{\mathtt{Writes}} = \{\mathtt{w}_1, \mathtt{w}_2\}$, $\Delta_{\mathtt{Pub}} = \{\mathtt{p}_1, \mathtt{p}_2\}$. For any stage > 3, the state of the database does not change, and the evaluation stops.

End semantics: This last semantics basically treats the delta relations as derived (intensional) relations in standard

datalog evaluation and executes the delta rules as a datalog program. It first derives all possible delta tuples and updates the database at the end of the process. In our running examples, tuples $\{g_2, a_2, a_3, w_1, w_2, p_1, p_2, c\}$ will be derived by datalog execution and will be deleted from the database.

Complexity: Given a delta program P and database D, it can be shown that computing $End(P, D)$ and $Stage(P, D)$ is poly-time solvable, while, given an integer k, it is NP-hard to decide if $|Ind(P, D)| \leq k$ or $|Step(P, D)| \leq k$ (we refer the reader to [5] for details).

3. ALGORITHMS & IMPLEMENTATION

We next give a brief overview of the main algorithms employed by MuSe.

For **stage and end semantics**, we use strategies similar to datalog evaluation, evaluating all rules and updating the database (either after each stage for stage semantics or at the end of the evaluation process for end semantics).

For **independent semantics**, we reduce the problem to min-ones SAT. We evaluate the program over the database as if we have all tuples in the original database and also the delta tuples. We store the *provenance* as a Boolean formula [6], negate it and find the satisfying assignment that assigns the minimum number of negated literals the value True. The algorithm generates the provenance expression of each derived delta tuple and includes it in a single formula, connected by $\vee$ between different assignments. Each delta tuple is represented as the negation of its counterpart in the initial database. In the running example, the formula F is:

$$g_2 \vee (a_2 \wedge ag_2 \wedge \neg g_2) \vee (a_3 \wedge ag_3 \wedge \neg g_2) \vee (p_1 \wedge w_1 \wedge \neg a_2) \vee (p_2 \wedge w_2 \wedge \neg a_3) \vee (c \wedge \neg p_1 \wedge w_1 \wedge w_2)$$

The fourth and fifth clauses stand for two identical assignments to both rules (2) and (3) as both rules have the same body. Next, the algorithm finds the minimum satisfying assignment of $\neg F$ (shown below), i.e., the assignment giving the value True to the smallest number of negated literals.

$$\neg g_2 \wedge (\neg a_2 \vee \neg ag_2 \vee g_2) \wedge (\neg a_3 \vee \neg ag_3 \vee g_2) \wedge (\neg p_1 \vee \neg w_1 \vee a_2) \wedge (\neg p_2 \vee \neg w_2 \vee a_3) \wedge (\neg c \vee p_1 \vee \neg w_1 \vee \neg w_2)$$

The satisfying assignment giving the minimum number of negated literals the value True is α such that $\alpha(g_2) = \alpha(ag_2) = \alpha(ag_3) = False$ and α gives every other variable the value True. Finally, the algorithm returns the set of tuples that α mapped to False, i.e., $\{g_2, ag_2, ag_3\}$.

For **step semantics,** we use a greedy approach that utilizes the *provenance graph* [3] as generated by the standard datalog evaluation (like end semantics). For each non-delta tuple in the graph, we store its *benefit*, which intuitively quantifies the effect of deleting a certain tuple on the size of the result. The higher the score, the more benefit gained. We then traverse the graph by layer and for each layer, remove the tuple with the highest benefit until no delta tuples exist in this layer. The removed tuples are returned by the algorithm.

The provenance graph for our running example is shown in Figure 3. After computing the benefit for all the leaf tuples, we begin iterating over the layers of the graph. In layer 1 we only have $\Delta(g_2)$, with $b_{g_2} = -1$, so we choose it. Since g_2 is only connected to $\Delta(g_2)$, we do not change G. We then continue to layer 2 where we have $\Delta(a_2)$ and $\Delta(a_3)$. We arbitrarily choose a_2 as $b_{a_2} = b_{a_3} = -1$, and do not change G. After that, we choose a_3 and again not change G. In layer 3, we have w_1, w_2, p_1, p_2 where $b_{p_2} = b_{p_1} < b_{w_1} = b_{w_2}$, so we

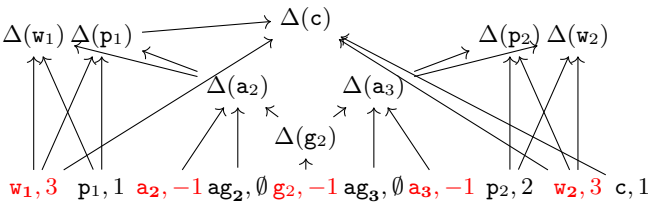


Figure 3: Provenance graph for step semantics algorithm. Red tuples are chosen for the output set

choose arbitrarily to include w_1 in S. We then delete from G the subgraph induced by $\Delta(w_1)$. Since there are more delta tuples in this layer we continue to choose w_2 and delete from G the subgraph induced by $\Delta(w_2)$. Since there are no more delta tuples in layers 3 and 4 except $\Delta(w_1), \Delta(w_2)$ where $w_1, w_2 \in S$, we return $S = \{g_2, a_2, a_3, w_1, w_2\}$.

4. USER INTERFACE AND DEMO

We will primarily use the academic MAS database [7] to demonstrate the usability of MuSe (integrated with other real databases). We next detail the demonstration scenario according to each component of the system, and through the screenshots in Figure 4 and our running example.

Formulating delta rules: We will begin the demonstration with the rules of our running example and allow the participants to formulate additional rules. MuSe provides three intuitive ways to formulate delta rules, demonstrated in Figure 4a. First, participants will gradually 'build' their desired delta rule using a pattern that guides them in choosing the table from which they would like to delete tuples, and then formalizing the conditions for which this deletion would occur. The pattern is revealed in stages, allowing participants to fill-in one step at a time and make sure they formulate a correct and accurate rule. Participants could then experiment with two more methods of formulating delta rules. MuSe provides a translation from Functional Dependencies (FDs) to delta rules, so participants will be able to formulate an FD and it will be automatically translated into a delta rule. Third, participants could formulate delta rules by using the syntax of foreign-key dependency [10] which will be automatically translated to a delta rule.

Choosing the repair semantics: In the input screen, participants will further choose their desired repair semantics from a drop-down menu. All four semantics will be available and each semantics may imply a different repair. To better gauge the effect of each semantics on the database, clicking the 'Preview' button will open a pop-up screen that shows the number of tuples from each table that are going to be deleted under each of the semantics. After choosing the semantics, participants will click on the 'Repair' button to make the system compute the repair that will be depicted in the output screen (shown for step semantics in Figure 4b).

Viewing repairs: Once the delta rules are formulated, a repair semantics has been chosen, and the participants have clicked the 'Repair' button, the output screen (shown for step semantics in Figure 4b) will display the deleted tuples. The tuples will be presented in a suitable manner according to the chosen semantics as follows. Independent semantics computes the repair "in one stroke", meaning that there is no process to present here. Thus, MuSe will portray the set of deleted tuples, along with their relations. For end semantics, in addition to displaying the deleted tuples, the system

further shows the delta rule that has caused the deletion of each tuple, when the mouse pointer hovers over this tuple. Step (resp. stage) semantics use multiple steps (stages) of repair, thus, `MuSe` will show for each step (stage), the tuples that were deleted in that step (stage) and, for each deleted tuple, the rule that is responsible for its deletion.

Under the hood: To investigate the repair according to independent or step semantics, participants will click the 'Under the Hood' button in the output screen will show an explanation describing the deletion process, based on the repair algorithm for the chosen semantics. For independent semantics, the Boolean formula describing the formula will be depicted with the chosen tuples highlighted (Figure 4c). For step semantics, `MuSe` will depict the provenance graph with the chosen tuples highlighted.

Commit a repair to the database: If the participants would want to use the generated repair according to their chosen delta rules and semantics, they will click the 'Commit' button that will commit the repair to the database and delete the tuples shown in the output screen (Figure 4b). Otherwise, they will click the 'Back' button and change the rules or choose a different semantics.

Free exploration of `MuSe`: Participants will be encouraged to change the delta rules, delete some of them and add new ones, and witness the effects of their changes on the repairs under different semantics. They will further be able to change the semantics, the database and view the results.

Acknowledgements. This research has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 804302), the Israeli Science Foundation (ISF) Grant No. 978/17, NSF awards IIS-1552538 and IIS-1703431, NIH award R01EB025021, and a Google Ph.D. Fellowship.

5. REFERENCES

[1] F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: Algorithms and complexity. In *ICDT*, pages 31–41, 2009.

[2] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

[3] D. Deutch, T. Milo, S. Roy, and V. Tannen. Circuits for datalog provenance. In *ICDT*, pages 201–212, 2014.

[4] R. Fagin, B. Kimelfeld, and P. G. Kolaitis. Dichotomies in the complexity of preferred repairs. In *PODS*, pages 3–15, 2015.

[5] A. Gilad, D. Deutch, and S. Roy. On multiple semantics for declarative database repairs. In *SIGMOD*, pages 817–831, 2020.

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

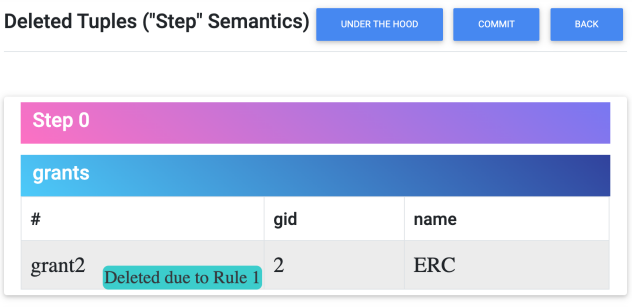
[7] Microsoft. Mas. `http://academic.research.microsoft.com/`.

[8] MySQL. Mysql trigger syntax. `https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html`, 2019.

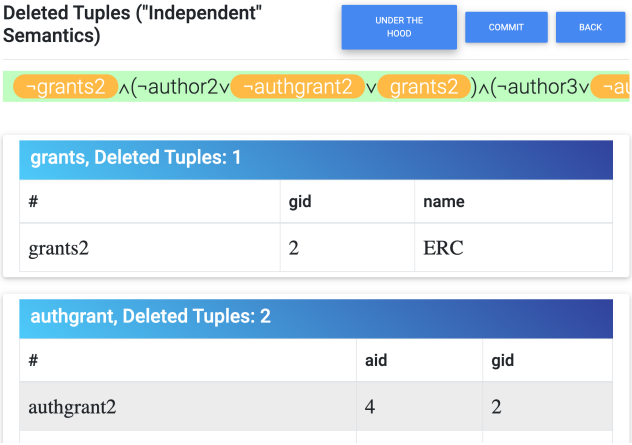
[9] PostgreSQL. Postgresql trigger behavior. `https://www.postgresql.org/docs/12/trigger-definition.html`, 2019.

[10] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *SIGMOD*, pages 1579–1590, 2014.

(a) Input screen



(b) Output for step semantics



(c) Explanation for independent semantics

Figure 4: `MuSe` User Interface