# Provenance for Entity Resolution

Sarah Oppold and Melanie Herschel

IPVS, University of Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany
{firstname.lastname}@ipvs.uni-stuttgart.de

**Abstract.** Data provenance can support the understanding and debugging of complex data processing pipelines, which are for instance common in data integration scenarios. One task in data integration is entity resolution (ER), i.e., the identification of multiple representations of a same real world entity. This paper focuses of provenance modeling and capture for typical ER tasks. While our definition of ER provenance is independent of the actual language or technology used to define an ER task, the method we implement as a proof of concept instruments ER rules specified in HIL, a high-level data integration language.

**Keywords:** Data provenance · entity resolution · data integration.

## 1 Motivation

Entity resolution (ER) refers to the problem of identifying duplicates, i.e., multiple representations of or references to a same real-world entity within or across data sources [1]. While numerous different solutions exist, they typically follow the same steps, building a generic ER pipeline.

Provenance may facilitate the understanding and debugging of data processing pipelines. While several provenance types exist for various applications [5], to the best of our knowledge, no solution has been tailored to ER. Here, determining which input data led to a duplicate is not very informative (the data provenance equals the duplicates). Instead, it is more relevant to see how these data affect ER processing. We therefore define a provenance model for ER.

One means to collect ER provenance is to instrument the original program defining ER by modifying it to return, in addition to the ER result, the corresponding provenance. We opt for this solution for two reasons: (i) the modified program can run on the same system as the original program, leveraging any optimizations implemented and (ii) the returned provenance is in the same format as the original output data, facilitating further processing.

In summary, we present a model for provenance describing how data was processed during ER. This model is independent of the actual language or processing engine used to specify and run ER. Indeed, we first abstract ER tasks to algebraic operators to then define provenance on this abstract representation (Sec. 2). In Sec. 3, we discuss how to capture provenance conforming to the abstract model by instrumenting ER rules specified in HIL, a data integration language developed at IBM [2]. We conclude the paper in Sec. 4 with a summary and an outlook on further research questions.

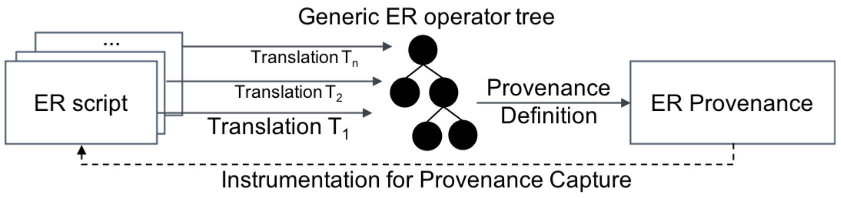


**Fig. 1.** Overview of the general approach

## 2 Provenance model for abstract ER pipelines

Fig. 1 summarizes our general approach. While many different formalisms to specify ER exist, they typically map to a generic model consisting of algebraic operators. This model forms the basis to define general ER provenance. Conceptually, ER provenance capture can then be achieved by first translating an ER script to the abstract model, to then execute provenance capture defined over this abstract model. This however would entail significant overhead, both in runtime and system complexity compared to running ER alone. A more lightweight solution that we pursue is the instrumentation of ER scripts such that both ER and ER provenance capture run on the original data processing system.

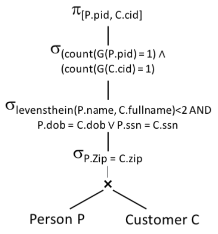
ER tasks typically divide into a pipeline consisting of several common steps (e.g., see [1]). The input comprises two datasets $A$ and $B$ (it is possible that $A$ = $B$). The output are partitions of pairs of entity descriptions (e.g., tuples in relational data) in $A \times B$ such that all entity descriptions in a partition refer to the same entity while no two partitions share entity descriptions referring to the same entity. Considering all pairs in $A \times B$ is computationally prohibitive, so *blocking* prunes pairs from further processing. During *pairwise classification*, the remaining pairs are compared, e.g., using similarity measures or domain-knowledge to ultimately decide whether or not the pair is a duplicate. Finally, *post-processing* views the classifications as a graph where vertices represent entity descriptions and edges connect duplicates. It partitions the graph, handling conflicting classifications (e.g., $a, b$ and $b, c$ duplicates, but $a, c$ non-duplicates) and enforcing additional constraints (e.g., on cardinalities [2]).

Fig. 2(a) shows a sample ER rule specified in HIL [2]. The syntax is not important, we use the rule to illustrate the different steps of ER. Here, persons in one source are matched with customers from another source. To avoid comparing all persons with all customers, blocking requires them to have the same ZIP code. Then, persons with same ZIP code are classified as duplicates if they match at least one of two rules (labeled match1 and match2). Finally, a constraint requires that a person can match at most one customer and vice versa. This is enforced during post-processing by pruning pairwise matches violating the constraint.

The ER steps described above are common to many different ER solutions, which we can thus map to an abstract representation. We have defined an abstract description of ER pipelines using operators of the Nested Relational Alge-

```
CREATE LINK PersonLink AS
SELECT [pid: P.pid, cid: C.cid]
FROM Person P, Customer C
BLOCK P ON P.zip, C ON C.zip
MATCH USING
   match1: levensthein(P.name,
             C.fullname)<2
             AND P.dob = C.dob,
   match2: P.ssn = C.ssn
CARDINALITY P.pid 1:1 C.cid;
```

(a) HIL ER rule


π[P.pid, C.cid]
σ(count(G(P.pid) = 1) ∧ (count(G(C.cid) = 1)
σlevensthein(P.name, C.fullname)<2 AND P.dob = C.dob ∨ P.ssn = C.ssn
σP.Zip = C.zip
×
Person P
Customer C

(b) Operator tree

<u>(i) Operator level granularity:</u>
[block: P.Zip = C.zip,
matches: levensthein(P.name, C.fullname)<2 AND P.dob = C.dob ∨ P.ssn = C.ssn,
cardinality: (count(G(P.pid) = 1) ∧ (count(G(C.cid) = 1)]

<u>(ii) Predicate level granularity:</u>
[block: P.Zip = C.zip,
matches: {[levensthein(P.name, C.fullname)<2 AND P.dob = C.dob ], [P.ssn = C.ssn]},
cardinality: {[count(G(P.pid) = 1], [count(G(C.cid) = 1]}]

(c) ER provenance schema

```
INSERT INTO
PersonLink_Matches_ProvLink
SELECT [entity: [pid: L.pid, cid: L.cid],
match_prov:[name: match1, condition:
levensthein(P.name, C.fullname)<2 AND
P.dob = C.dob]
FROM PersonLink L, Person P, Customer C
WHERE L.pid = P.pid AND L.cid = C.cid;
```

```
INSERT INTO
PersonLink_Matches_ProvLink
SELECT [entity: [pid: L.pid, cid: L.cid],
match_prov:[name: match2, condition:
P.ssn, C.ssn]
FROM PersonLink L, Person P, Customer C
WHERE L.pid = P.pid AND L.cid = C.cid;
```

```
INSERT INTO PersonLink_Matches_Prov!
[i.entity]
SELECT [match_prov: i.match_prov]
FROM PersonLink_Matches_ProvLink i;
```

(d) Provenance capture using HIL EP rules

**Fig. 2.** Running example

bra for Bags (NRAB) [3]. Indeed, we want to cover ER both on flat relational and nested data and cannot assume that data is free of exact duplicates. Fig. 2(b) illustrates the operator tree for the HIL script shown in Fig. 2(a). For HIL, we have defined a full set of inference rules to map any HIL ER script to an operator tree, similarly to our previous work where we compile PigLatin to NRAB [4]. We cannot cover the details here, but highlight a few principles based on the example. First, we form all pairs of entity descriptions using the Cartesian product $\times$. Blocking prunes pairs and can thus be modeled using a selection $\sigma$. The matching performs pairwise classification that returns only those pairs that satisfy either of the match conditions, resulting in a selection operator with a complex predicate in DNF. Finally, the constraint of the 1:1 cardinality requires both grouping $G$ and selection based on the size of groups. The ER rules in HIL return pairs of duplicates, which translates to the final projection $\pi$.

Given ER pipelines defined by trees of NRAB operators with clear semantics, data provenance (i.e., why- and how-provenance) [5] is a candidate choice for ER provenance. However, it focuses on data flow, which is not informative for ER, as the provenance of a duplicate pair simply consists of the pair members. Instead, we propose to capture the control flow of the pipeline. More specifically, we record, at each processing stage, the results of function calls, comparisons etc. in addition to data valuations. For efficiency and understandability reasons, we define two granularities of ER provenance: (i) the granularity of operators and (ii) the granularity of individual predicates in the operator parameters. Fig. 2(c) shows the schema of the provenance at these two granularities for our running example. An example instance of the second granularity is $[70569 = 70569, \{[1 < 2, 5/29/60 = 6/29/60], 123\text{-}45\text{-}678 = 123\text{-}45\text{-}678\}, 1 = 1]$. This evaluates to $[true, \{[true, false], true\}, true]$, clearly indicating that the duplicate was found based on match2 only, as unequal dates let match1 fail.

## 3 Implementing provenance capture for HIL ER rules

As motivated previously, we opt for program instrumentation to capture provenance for ER. As a proof of concept, we have formalized and implemented the instrumentation of HIL scripts [2] that allow the specification of ER rules, illustrated in Fig. 2(a). Our implementation supports the full set of HIL ER rule clauses (not all are illustrated here). Given a HIL script with an ER rule, we generate additional entity population (EP) rules. These rules are an integral part of the HIL language and will, when executed, produce the provenance conforming to our general ER provenance. Fig. 2(d) shows EP rules generated for the sample HIL ER rule of Fig. 2(a). Capturing the necessary provenance using HIL constructs requires several intermediate steps, the final provenance being stored in the result labeled $PersonLink\_Matches\_Prov$. A detailed discussion is out of the scope of this paper, but the example showcases the complexity of instrumenting HIL for provenance capture.

## 4 Conclusion and outlook

This paper presented a framework for defining and capturing provenance for typical ER pipelines. We showed how ER pipelines consisting of several common steps map to trees of algebraic operators. We then defined ER provenance over this abstract ER representation, thus providing a language-independent provenance model. To capture ER provenance in practice, we showed how to instrument a particular language to specify ER, namely HIL ER rules to capture ER provenance conforming to our model.

In the future, we plan to extend provenance capture to further data integration tasks from both a language-independent and a HIL specific perspective. Further important issues for making provenance capture practical and relevant for users are runtime optimizations and provenance visualization, exploration, and querying.

**Acknowledgements.** The authors thank the German Research Foundation (DFG) for financial support within project D03 of SFB/ Transregio 161. This research was also partly funded by an IBM Faculty Award.

## References

1. P. Christen. Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Data-Centric Systems and Applications. Springer (2012).
2. M. A. Hernández, G. Koutrika, R. Krishnamurthy, L. Popa, R. Wisnesky. HIL: a high-level scripting language for entity integration. EDBT (2013).
3. S. Grumbach, T. Milo. Towards Tractable Algebras for Bags. PODS (1993).
4. J. Camacho-Rodríguez, D. Colazzo, M. Herschel, I. Manolescu, S. Roy Chowdhury. Reuse-based Optimization for Pig Latin. CIKM (2016).
5. M. Herschel, R. Diestelkämper, H. B. Lahmar. A survey on provenance: What for? What form? What from? VLDB Journal (2017).