Carleton
UNIVERSITY

Query Rewriting in Datalog for On-The-Fly Entity Resolution

Leopoldo Bertossi[⋆]

Carleton University
School of Computer Science
Ottawa, Canada

Joint work with Jaffer Gardezi (SITE, Ottawa U.)

⋆: Faculty Fellow of the IBM CAS

Duplicate Resolution and MDs

A (relational) database may contain several representations of the same external entity

The database contains "duplicates", which is in general considered to be undesirable

The database has to be cleaned ...

The problem of duplicate- or entity-resolution is about:

(a) detecting duplicates, and

(b) merging duplicate representations into single ones

This is a classic and complex problem in data management, and data cleaning in particular

We concentrate on the merging part of the problem

A generic way to approach the problem consists in specifying attribute values that have to be matched (made identical) under certain conditions

A declarative language with a precise semantics could be used for this purpose

In this direction, matching dependencies (MDs) were recently introduced (Fan et al., PODS'08, VLDB'09)

They represent rules for resolving pairs of duplicate representations (two tuples at a time)

An MD indicates attribute values that have to be matched when certain similarities between attribute values hold

Example: The similarities of phone and address indicate that the tuples refer to the same person, and the names should be matched

People (P)	Name	Phone	Address
	John Smith	723-9583	10-43 Oak St.
	J. Smith	(750) 723-9583	43 Oak St. Ap. 10

Here: 723-9583 $\approx$ (750) 723-9583 and 10-43 Oak St. $\approx$ 43 Oak St. Ap. 10

An MD capturing this cleaning policy:

$$P[Phone] \approx P[Phone] \wedge P[Address] \approx P[Address] \rightarrow$$
$$P[Name] \doteq P[Name]$$

(an MD may involve two different relations)

Matching Dependencies

MDs are rules of the form

$$\bigwedge_{i,j} R[A_i] \approx_{ij} S[B_j] \;\rightarrow\; \bigwedge_{k,l} R[A_k] \doteq S[B_l]$$

LHS captures a similarity condition on pairs of tuples, in relations R and S

Abbreviation: $R[\bar{A}] \approx S[\bar{B}] \;\rightarrow\; R[\bar{C}] \doteq S[\bar{E}]$

Static interpretation: If antecedent is true for a pair of tuples, then the values $R[A_k]$ and $S[B_l]$ should be the same

Dynamic interpretation: Those values on the RHS should be updated to some (unspecified) common value

Hence, attributes on a RHS are called *changeable attributes*

The similarity operators $\approx$ satisfy:

(a) Symmetry: If $x \approx y$, then $y \approx x$

(b) Equality Subsumption: If $x = y$, then $x \approx y$

Transitivity is *not* assumed (and usually does not hold)

MDs are to be "applied" iteratively until duplicates are solved

To keep track of changes and comparing tuples and instances, we use global tuple identifiers (a non-changeable surrogate key)

This attribute (when shown) appears as the first attribute in a relation, e.g. t is the identifier in $R(t, \bar{x})$

A *position*: A pair (t, A) with t a tuple id, A an attribute

The *position's value* is $t[A]$, the value for A in tuple (with id) t

MD Semantics

A semantics for MDs acting on database instances was introduced in (Gardezi and Bertossi; LID'11; FofCS'12)

It is based on a chase procedure iteratively applied to the original instance D

A *resolved instance* D' is obtained from a finitely terminating sequence of instances $D \mapsto D_1 \mapsto D_2 \mapsto \cdots \mapsto D'$

D' satisfies the MDs in the sense of the static interpretation, seeing MDs as "FDs" (EGDs)

The semantics specifies the one-step transitions ($\mapsto$), i.e. the updates used/allowed to go from D_{i-1} to D_i

Only *modifiable positions* within the instance are allowed to change their values in such a step, and as forced by the MDs

Modifiable positions depend on a whole set $\mathcal{M}$ of MDs and the instance at hand

They are recursively defined, but intuitively

Position (t, A) is modifiable iff:

1. There is a t' such that the pair $\{t, t'\}$ satisfies the similarity condition of an MD with A on the RHS
2. $t[A]$ has not already been resolved, i.e. it is different from one of its other duplicates

Example 1: $R[A] = R[A] \rightarrow R[B] \doteq R[B]$

The positions of the values in red in D are modifiable, because their values are unresolved (wrt the MD)

$R(D)$	A	B
t_1	a	b
t_2	a	c

$\mapsto$

$R(D')$	A	B
t_1	a	d
t_2	a	d

D' is a resolved instance since it satisfies the MD interpreted as an FD (the update value d is arbitrary)

D' has no modifiable positions with unresolved values: the values for B are already the same, so no reason to change them

Example 2:

$$R[A] = R[A] \rightarrow R[B] \doteq R[B]$$
$$R[B] = R[B] \rightarrow R[C] \doteq R[C]$$

$R(D)$	A	B	C
t_1	a	b	d
t_2	a	c	e
t_3	a	b	e

Attribute $R(C)$ is changeable

Position (t_2, C) is not modifiable wrt $\mathcal{M}$ and D: No justification to change its value in one step on the basis of an MD and D

Position (t_1, C) is modifiable

With the restrictions (to be) imposed, two resolved instances:

$R(D_1)$	A	B	C
t_1	a	b	d
t_2	a	b	d
t_3	a	b	d

$R(D_2)$	A	B	C
t_1	a	b	e
t_2	a	b	e
t_3	a	b	e

D_1 cannot be obtained in a single (one step) update: the red value is for a non-modifiable position D_2 can ...

Single Step Semantics: $(\mapsto)$

Each ordered pair D_i, D_{i+1} in an update sequence (a chase step) must *satisfy* the set $\mathcal{M}$ of MDs

Denoted $(D_i, D_{i+1}) \models \mathcal{M}$

$(D_i, D_{i+1}) \models \mathcal{M}$ holds iff:

1. For every MD, say $R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \doteq S[\bar{D}]$ and pair of tuples t_R and t_S, if $t_R[\bar{A}] \approx t_S[\bar{B}]$ in D_i, then $t_R[\bar{C}] = t_S[\bar{D}]$ in D_{i+1}
2. The value of a position can only differ between D_i and D_{i+1} if it is modifiable wrt D_i

This semantics stays as close as possible to the spirit of the MDs as originally introduced, and also uncommitted

Other semantics have been proposed and investigated:

- As above, but modifying the chase conditions, e.g. one MD at a time, previous resolutions cannot be unresolved, etc.
- Using matching functions to choose a value for a match

  (Bertossi, Kolahi, Lakshmanan; ICDT'11, TofCSs),

  (Bahmani, Bertossi, Kolahi, Lakshmanan; KR'12)

Back to the previously defined resolved instances ...

Minimally Resolved Instances:

Among the resolved instances we prefer those that are the closest to the original instance

A minimally resolved instance (MRI) of D is a resolved instance D' such that the number of changes of attribute values comparing D with D' is a minimum

Instance D_2 in Example 2 is an MRI, but not D_1 (2 vs. 3)

Resolved Answers:

Given a conjunctive query $\mathcal{Q}$, a set of MDs $\mathcal{M}$, and an instance D, *resolved answers* are invariant under the ER process

That is, the *resolved answers* are those answers to $\mathcal{Q}$ that are returned by each of MRIs of D

Resolved answers are "semantically correct and certain"

Similar in spirit to consistent query answering (CQA) from an instance that fails to satisfy a set of integrity constraints

(Arenas, Bertossi, Chomicki; PODS'99)

Developing (polynomial-time) query rewriting methodologies for CQA has been the focus of intensive research

Such rewritings, in cases when CQA is polynomial-time, have all been first-order

With the rewriting into $\mathcal{Q}'$ of an initial query $\mathcal{Q}$, we answer $\mathcal{Q}'$ from the dirty instance D as usual, obtaining the resolved answers to $\mathcal{Q}$ from D

Avoiding generation, materialization and querying of possibly many MRIs ...

Doing something similar for MDs has not been attempted before and brings new challenges:

- MDs contain the non-transitive similarity predicates
- Enforcing consistency of updates requires computing their transitive closures (TC)
- The minimality of *number* of value changes
  (not considered for rewriting in CQA)

Example: $R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$

$R(D)$	A	B
t_1	a	e
t_2	b	f
t_3	c	g

It holds $a \approx b \approx c$, but $a \not\approx c$

Consistent updates requires considering $\approx$'s TC

Duplicate resolution requires $t_1[B]$ and $t_3[B]$ to be updated to the same value

Query Rewriting

We developed a query rewriting methodology (G&B; Datalog 2.0'12)

The rewritten queries turn out to be Datalog queries

There are two main classes of sets of MDs for which query rewriting can be applied: (G&B; SUM'12)

1. Sets where MDs do not depend on each other (*non-interacting* sets of MDs)
2. Sets where MDs depend cyclically on each other

E.g.

$$R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$$
$$R[B] \approx R[B] \rightarrow R[A] \doteq R[A]$$

But not

$$R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$$
$$R[B] \approx R[B] \rightarrow R[C] \doteq R[C]$$

(actually, intractable for simple queries)

Cycles help: chase termination condition imposes a simple form on the minimally resolved instances (easier to capture and characterize)

For these sets of MDs a conjunctive query can be rewritten to retrieve the resolved answers

Provided there are no joins on existentially quantified variables corresponding to changeable attributes: *unchangeable attribute join conjunctive queries* (UJCQ) (G&B; SUM'12)

Tuple-Attribute Closure:

Given instance D and MDs $\mathcal{M}$

$TA_{\mathcal{M},D}$, the tuple-attribute closure, is the transitive closure of a binary relation $\approx'$ on positions

$\approx'$ depends on $D, \mathcal{M}$, and $\approx$

$\approx'$ and its TC can be defined in Datalog

$TA_{\mathcal{M},D}$ turns out to be an equivalence relation

For an equivalence class E of $TA_{\mathcal{M},D}$:

$$freq^D(a, E) := |\ \{(t, A) \mid (t, A) \in E,\ t[A] = a \text{ in } D\}\ |$$

In an MRI, all positions in each equivalence class E must take a common value a that maximizes $freq^D(a, E)$

Example: $\mathcal{M}$: $R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$

It holds $a \approx b \approx c$

$R(D)$	A	B
t_1	a	e
t_2	b	e
t_3	c	g

$\rightarrow$

$R(D')$	A	B
t_1	a	e
t_2	b	e
t_3	c	e

$TA_{\mathcal{M},D}$ is TC of the relation $(t_1, B) \approx' (t_2, B) \approx' (t_3, B)$

D' is the only MRI

The Rewriting:

Input: A conjunctive query in UJCQ

Output: A stratified Datalog^{not} program with recursion and aggregation (no disjunction)

Recursion arises in the computation of the TC $TA_{\mathcal{M},D}$

Aggregation is needed to compute $freq^D(a, E)$

Negation is needed to minimize the frequency

To retrieve the resolved answers to $\mathcal{Q}$ from D:

1. Generate a query rule defining auxiliary query predicate $\mathcal{Q}'$ from $\mathcal{Q}$
2. Combine with the Datalog program mentioned above
3. Run on top of (possibly dirty) D as usual

Resolved answers can be obtained in polynomial time (in data)

Query Rewriting Example

Example: (the gist)

Schema: $R[A, B]$ Query: $\mathcal{Q}(x, y) \leftarrow R(x, y)$

$\mathcal{M}$:

$$R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$$
$$R[B] \approx R[B] \rightarrow R[A] \doteq R[A]$$

Resolved answers to original query are the (usual) answers to:

$$\mathcal{Q}'(x, y) \leftarrow R(t, x', y'), not\ Compare^A(t, x), not\ Compare^B(t, y), C^A(t, x, z_1), C^B(t, y, z_2)$$

Four new predicates in the body are defined using TC and aggregation, independently from the query

Final Remarks

- In general, query rewriting cannot be applied when the MDs have a non-cyclic dependence on each other: retrieving the resolved answers is intractable (G&B; SUM'12)

- Next step: Experiments ...

- Other definitions of resolved answer can be considered in our setting, such as

  (a) Answers that are true in *all* resolved instances (not necessarily minimal)

  In some cases the chase rules can be expressed directly as Datalog rules

(b) Answers in all (minimally) resolved instances obtained with a modified chase procedure

Adding the requirement that two values, once made equal, cannot subsequently be made unequal

The same rewriting technique applies, but it applies to all sets of MDs