Data Processing and Analytics (DISS-DPA)

Principles of Data Quality – Repairing with Quality Improving Constraints

Christopher Spinrath

Database Group (BD) – CNRS – LIRIS – Université Lyon 1

Fall 2025

This presentation is based on slides by Angela Bonifati

Outline

QIDs and The Repair Problem

Motivation

Previously

- Data quality is an important problem in data management
- Dirty data is everywhere and costly
- A principled approach to detect inconsistencies and similar objects based on quality dependencies
  - Conditional FDs, Matching Dependencies, etc.

Motivation

Previously

- Data quality is an important problem in data management
- Dirty data is everywhere and costly
- A principled approach to detect inconsistencies and similar objects based on quality dependencies
  - Conditional FDs, Matching Dependencies, etc.

In this Episode

Can these dependencies also be used to repair data?

Ingredients: Dependencies and Repair Models

Ingredients for the Repair Problem

1. Quality dependencies
   - For instance, (conditional) FDs, Matching dependencies, etc.
2. A dirty database
3. A repair model
   - What kind of operations are allowed to modify the database?
   - Examples: tuple deletions, tuple insertions, value modifications
4. A cost model
   - the repair should differ minimally
   - Examples: number of deletions, edit distance

Goal

A clean database that satisfies all the dependencies

Ingredients – Example

Example (Ingredients for the Repair Problem)

1. Key FD: Student[Id → Name]
2. The dirty database with

Relation Student

Id	Name
123	Volta
123	Marconi
456	Avogadro
789	Fermi

3. Repair model: only tuple deletions
4. Cost model: number of deletions

Ingredients – Example

Example (Ingredients for the Repair Problem)

1. Key FD: Student[Id → Name]
2. The dirty database with

Relation Student

Id	Name
123	Volta
123	Marconi
456	Avogadro
789	Fermi

3. Repair model: only tuple deletions
4. Cost model: number of deletions

Two Possible Repairs

Relation Student

Id	Name
123	Marconi
456	Avogadro
789	Fermi

or

Relation Student

Id	Name
123	Volta
456	Avogadro
789	Fermi

Repairs

Definition (Repair)

A repair D' of database D with respect to

- a set Σ of data quality dependencies and
- a quality metric $\mathtt{qty}$ governed by underlying repair and cost models

is a database such that

1. $D' \models \Sigma$, and
2. $\mathtt{qty}(D, D')$ is maximal

We will shortly make more precise

- what Σ is, i.e., which data quality dependencies we consider; and
- what repair models and quality metrics are used

Repairs

Definition (Repair)

A repair D' of database D with respect to

- a set Σ of data quality dependencies and
- a quality metric $\texttt{qty}$ governed by underlying repair and cost models

is a database such that

1. $D' \models \Sigma$, and
2. $\texttt{qty}(D, D')$ is maximal

Example

In the previous example

- Σ consisted of a key FD
- the repair model/metric was the so-called subset repair, i.e., the maximal repair included in the original database which only allows for deletions

We will shortly make more precise

- what Σ is, i.e., which data quality dependencies we consider; and
- what repair models and quality metrics are used

Different Approaches to Data Repairing

Observation

We have seen that a repair is not unique

Different Approaches to Data Repairing

Observation

We have seen that a repair is not unique

- The research community has studied two different ways of dealing with (multiple) repairs and queries over them

Different Approaches to Data Repairing

Observation

We have seen that a repair is not unique

- The research community has studied two different ways of dealing with (multiple) repairs and queries over them

Consistent Query Answering

- Avoid selecting a repair; and
- at query time only return query answers that are common to all repairs
- Has been studied for quite some time now

Different Approaches to Data Repairing

Observation

We have seen that a repair is not unique

- The research community has studied two different ways of dealing with (multiple) repairs and queries over them

Consistent Query Answering

- Avoid selecting a repair; and
- at query time only return query answers that are common to all repairs
- Has been studied for quite some time now

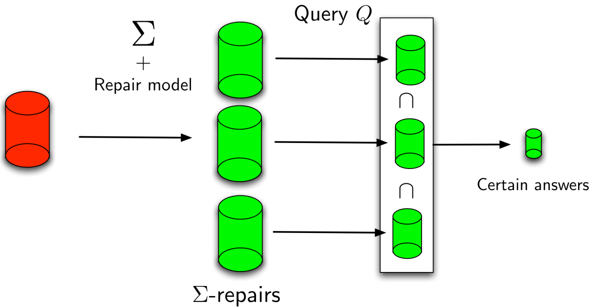
Data Repairing

- Select the best possible repair
- which is subsequently queried. Has only recently received attention in the database community

Consistent Query Answering

Idea of Consistent Query Answering

Consider all repairs but only retrieve common answers



Challenge

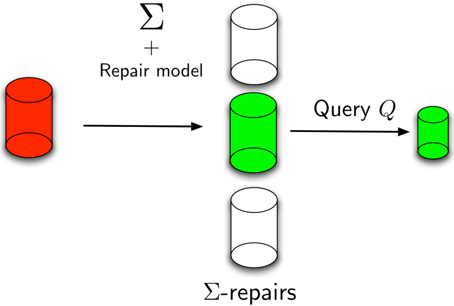
How to compute certain answers without computing all repairs?

- This is an independent subject on its own

Data Repairing and Querying

Idea of Data Repairing

Select a best repair and query it



Challenge

How to compute a best repair?

- We will focus on this

Data Quality Dependencies

Specification of Data Quality Rules

- The formalism should be expressive enough to specify data quality rules; and
- simple enough such that reasoning over them is (rather) efficient

Data Quality Dependencies

Specification of Data Quality Rules

- The formalism should be expressive enough to specify data quality rules; and
- simple enough such that reasoning over them is (rather) efficient

How are Data Qualities Specified?

Using a logical formalism

- Note that unrestricted use of logic leads to undecidable problems
  - For example, it is well-known that the satisfiability problem of first-order logic is undecidable

Recall: Conditional Function Dependencies (CFDs)

Extension of FDs with constants on both premise and consequence

Example (Conditional Functional Dependency (CFD))

"In the UK, the zip code uniquely determines the street"

$$\forall t_1 \forall t_2 \Big(\big(\text{Address}(t_1) \wedge \text{Address}(t_2) \wedge t_1[\text{zip}] = t_2[\text{zip}] \wedge t_1[\text{CC}] = t_2[\text{CC}] \wedge t_1[\text{CC}] = 44\big) \rightarrow t_1[\text{street}] = t_2[\text{street}] \Big)$$

Recall: Matching Dependencies

Extension of FDs with similarity relations in the premise

Example (Matching Dependencies (MDs))

> *"If two entities (tuples) agree on their last name and address and if their first names are similar, then the two tuples should be identified on related attributes"*

Recall: Matching Dependencies

Extension of FDs with similarity relations in the premise

Example (Matching Dependencies (MDs))

"If two entities (tuples) agree on their last name and address and if their first names are similar, then the two tuples should be identified on related attributes"

$$\forall t_1 \forall t_2 \Big(\big(\text{CardHolder}(t_1) \wedge \text{Transaction}(t_2)\big) \wedge t_1[\text{LN}] = t_2[\text{LN}] \wedge t_1[\text{address}] = t_2[\text{post}] \wedge t_1[\text{FN}] \asymp t_2[\text{FN}]\big) \rightarrow t_1[X] = t_2[Y] \Big)$$

- $\asymp$ is a similarity operator
- X and Y are compatible attributes of CardHolder and Transaction, respectively.

A Language for Data Quality Dependencies

Quality Improving Dependency (QID)

A quality improving dependency (QID) is a first-order sentence of the following form

$$\forall t_1 \forall t_2 \Big(\big(R(t_1) \wedge S(t_2) \wedge \bigwedge_{i \in [1,n]} t_1[A_i] \ \mathrm{op}_i \ t_2[B_i] \big) \rightarrow \bigwedge_{j \in [1,m]} t_1[C_j] \ \mathrm{op}'_j \ t_2[D_j] \Big)$$

where the operators op_i and op'_j form the signature of the dependency

A Language for Data Quality Dependencies

Quality Improving Dependency (QID)

A quality improving dependency (QID) is a first-order sentence of the following form

$$\forall t_1 \forall t_2 \Big(\big(R(t_1) \wedge S(t_2) \wedge \bigwedge_{i \in [1,n]} t_1[A_i] \text{ op}_i \; t_2[B_i] \big) \rightarrow \bigwedge_{j \in [1,m]} t_1[C_j] \text{ op}'_j \; t_2[D_j] \Big)$$

where the operators op_i and op'_j form the signature of the dependency

Operators

- Equality: $t_1[A] = t_2[B]$ iff attribute A of t_1 and B of t_2 have the same value
- Equality with constant: $t_1[A] =_c t_2[B]$ iff attribute A of t_1 and B of t_2 have value c
- Similarity: $t_1[A] \sim t_2[B]$ iff the values of attribute A of t_1 and B of t_2 are similar relative to some similarity relation $\sim$

Quality Improving Dependencies

Subclasses of QIDs

FDs Signatures consist of equality only

CFDs Signatures consist of equalities and equalities with constants

MDs Signatures consist of equality and similarity relations

Note

We will not consider inclusion dependencies (INDs) or conditional INDs in the remainder of this lecture

Repair Models

- determine which modifications are allowed to repair a database; and
- which cost function (if any) is optimized

Repair Models

Repair Models

- determine which modifications are allowed to repair a database; and
- which cost function (if any) is optimized

Subset Repair (S-Repair)

A S-repair D' of a database D w.r.t. a set Σ of QIDs is a database D' such that

- $D' \models \Sigma$ and $D' \subseteq D$; and
- there is no database D'' such that $D'' \models \Sigma$ and $D' \subsetneq D'' \subseteq D$.

Observation

- S-repairs are obtained by tuple deletions

Repair Models

Subset Repair (S-Repair)

A S-repair D' of a database D w.r.t. a set Σ of QIDs is a database D' such that

- $D' \models \Sigma$ and $D' \subseteq D$; and
- there is no database D'' such that $D'' \models \Sigma$ and $D' \subsetneq D'' \subseteq D$.

Observation

- S-repairs are obtained by tuple deletions

Repair Models

Subset Repair (S-Repair)

A S-repair D' of a database D w.r.t. a set Σ of QIDs is a database D' such that

- $D' \models \Sigma$ and $D' \subseteq D$; and
- there is no database D'' such that $D'' \models \Sigma$ and $D' \subsetneq D'' \subseteq D$.

Observation

- S-repairs are obtained by tuple deletions

Symmetric-Difference Repair (Δ-Repair)

A Δ-repair D' of a database D w.r.t. a set Σ of QIDs is a database D' such that

- $D' \models \Sigma$; and
- there is no database D'' such that $D'' \models \Sigma$ and $D \Delta D'' \subseteq D \Delta D'$.

Repair Models

Subset Repair (S-Repair)

A S-repair D' of a database D w.r.t. a set Σ of QIDs is a database D' such that

- $D' \models \Sigma$ and $D' \subseteq D$; and
- there is no database D'' such that $D'' \models \Sigma$ and $D' \subsetneq D'' \subseteq D$.

Observation

- S-repairs are obtained by tuple deletions

Symmetric-Difference Repair (Δ-Repair)

A Δ-repair D' of a database D w.r.t. a set Σ of QIDs is a database D' such that

- $D' \models \Sigma$; and
- there is no database D'' such that $D'' \models \Sigma$ and $D \Delta D'' \subsetneq D \Delta D'$.

Observations

- Recall: the symmetric difference $X \Delta Y$ of two sets X, Y is $X \Delta Y = (X \setminus Y) \cup (Y \setminus X)$
- Δ-repairs are obtained by tuple deletions and insertions

Repair Models

Subset Repair (S-Repair)

A S-repair D' of a database D w.r.t. a set Σ of QIDs is a database D' such that

- $D' \models \Sigma$ and $D' \subseteq D$; and
- there is no database D'' such that $D'' \models \Sigma$ and $D' \subsetneq D'' \subseteq D$.

Observation

- S-repairs are obtained by tuple deletions

Symmetric-Difference Repair (Δ-Repair)

A Δ-repair D' of a database D w.r.t. a set Σ of QIDs is a database D' such that

- $D' \models \Sigma$; and
- there is no database D'' such that $D'' \models \Sigma$ and $D \Delta D'' \subsetneq D \Delta D'$.

Observations

- Recall: the symmetric difference $X \Delta Y$ of two sets X, Y is $X \Delta Y = (X \setminus Y) \cup (Y \setminus X)$
- Δ-repairs are obtained by tuple deletions and insertions

Observation

The quality dependencies considered here can never be resolved by inserting tuples

Repair models

Value-Modification Repair (V-Repair)[1]

A V-Repair D' of a database D w.r.t. a set Σ of QIDs is a database D' such that

- $D' \models \Sigma$; and
- the cost

$$\mathrm{cost}(D', D) = \sum_{\substack{t' \in D', t \in D \\ t \to t'}} \sum_{\text{Attribute } A} w(t, A) \cdot \mathrm{dist}(t[A], t'[A])$$

  is minimized, where
  - $t \to t'$ means that t' is a tuple in D' derived from t in D;
  - $w(t, A)$ denotes the accuracy of attribute A;
  - dist is a distance measure.

Observation

V-repairs can be obtained by tuple deletions, insertions and attribute-value modifications

[1] Hao et al., "A Novel Cost-Based Model for Data Repairing", *IEEE Trans. Knowl. Data Eng.*, 2017

Example: V-Repair

Example (V-Repair)

Key constraint: Student[Id $\rightarrow$ Name]

Dirty Database

Relation Student

Id	Name
123	Volta
123	Marconi
456	Avogadro
789	Fermi

Example: V-Repair

Example (V-Repair)

Key constraint: Student[Id $\rightarrow$ Name]

Dirty Database

Relation Student

Id	Name
123	Volta
123	Marconi
456	Avogadro
789	Fermi

Example: V-Repair

Example (V-Repair)

Key constraint: Student[Id → Name]

Dirty Database

Relation Student

Id	Name
123	Volta
123	Marconi
456	Avogadro
789	Fermi

Repaired, Clean Database

Relation Student

Id	Name
123	Volta
345	Marconi
456	Avogadro
789	Fermi

Repairing by Chasing

Finding Repairs

Idea

- To find repairs we take some inspiration from the classic chase procedure

Finding Repairs

Idea

- To find repairs we take some inspiration from the classic chase procedure

Why the Chase?

The chase takes as input

- a set Σ of (equality and tuple generating) dependencies; and
- an input database D, possibly containing null (i.e. unknown) values,

and, if the chase terminates successfully, then it outputs a database D' such that $D' \models \Sigma$

Finding Repairs

Idea

- To find repairs we take some inspiration from the classic chase procedure

Why the Chase?

The chase takes as input

- a set Σ of (equality and tuple generating) dependencies; and
- an input database D, possibly containing null (i.e. unknown) values,

and, if the chase terminates successfully, then it outputs a database D' such that $D' \models \Sigma$

Notes

- It seems that the chase solves the problem of data repairing
  - at least for equality and tuple generating dependencies, and
  - without taking any cost function into account
- However, we will see that we have to extend the standard chase

The Standard Chase for QIDs

Let

$$\varphi = \forall t_1 \forall t_2 \Big(\underbrace{\big(R(t_1) \wedge S(t_2) \wedge \bigwedge_{i \in [1,n]} t_1[A_i] \ \mathtt{op}_i \ t_2[B_i] \big)}_{\psi} \rightarrow t_1[C] = t_2[D] \Big)$$

be a non-constant QID.

- here non-constant means that the operator in the consequence is equality

The Standard Chase for QIDs

Let

$$\varphi = \forall t_1 \forall t_2 \Big(\underbrace{\big(R(t_1) \wedge S(t_2) \wedge \bigwedge_{i \in [1,n]} t_1[A_i] \ \mathtt{op}_i \ t_2[B_i] \big)}_{\psi} \rightarrow t_1[C] = t_2[D] \Big)$$

be a non-constant QID.

- here non-constant means that the operator in the consequence is equality

Firing of a QID

The QID φ can be fired on a database D if there are two tuples $t_1, t_2 \in D$ such that

- $(D, t_1, t_2) \models \psi$ holds
- but $(D, t_1, t_2) \models t_1[C] = t_2[D]$ does not hold

The Standard Chase for QIDs

The Chase Procedure

Input: a database D, possibly with labelled nulls representing missing values

1. Initialize $D' = D$
2. As long as there is a QID φ and tuples $t_1, t_2 \in D'$ for which φ can be fired do
   2.1 If $t_1[C] = \texttt{null}_i$ and $t_2[D] = c$ is a constant, replace $\texttt{null}_i$ in every tuple in D' with c
   2.2 If $t_1[C] = \texttt{null}_i$ and $t_2[D] = \texttt{null}_j$, replace $\texttt{null}_j$ in every tuple in D' with $\texttt{null}_i$
   2.3 If $t_1[C] = c$ and $t_2[D] = d$ are both constants, then report failure

Preferences

Intuitively, constants overwrite labelled nulls as these are less informative

The Standard Chase for QIDs – Example

Example (Case 2.1: Null vs. Constant)

Key constraint: $\varphi = \text{Student}[\text{Id} \rightarrow \text{Name}]$

Dirty Database

Relation Student

Id	Name
123	$\texttt{null}_1$
123	Marconi
456	Avogadro
444	$\texttt{null}_1$
789	Fermi
888	$\texttt{null}_2$

Firing φ →

After Firing φ

Relation Student

Id	Name
123	Marconi
456	Avogadro
444	Marconi
789	Fermi
888	$\texttt{null}_2$

The Standard Chase for QIDs – Example

Example (Case 2.2: Null vs. Null)

Key constraint: $\varphi = \text{Student}[\text{Id} \rightarrow \text{Name}]$

Dirty Database

Relation Student

Id	Name
123	$\texttt{null}_1$
123	$\texttt{null}_2$
456	Avogadro
789	Fermi
888	$\texttt{null}_1$

Firing φ →

After Firing φ

Relation Student

Id	Name
123	$\texttt{null}_2$
456	Avogadro
789	Fermi
888	$\texttt{null}_2$

The Standard Chase for QIDs – Example

Example (Case 2.3: Constant vs. Constant)

Key constraint: $\varphi = \text{Student}[\text{Id} \rightarrow \text{Name}]$

Dirty Database

Relation Student

Id	Name
123	Volta
123	Marconi
456	Avogadro
789	Fermi
888	$\texttt{null}_2$

Firing φ →

Failure!
because Volta $\neq$ Marconi

The Standard Chase for QIDs – Example

Example (Conditional Dependencies)

CFD: $\varphi = \text{Student}[\text{Id} = 123 \rightarrow \text{Name} = \textit{Marconi}]$

Dirty Database

Relation Student

Id	Name
123	$\texttt{null}_1$
123	Marconi
456	Avogadro
444	$\texttt{null}_1$
789	Fermi
888	$\texttt{null}_2$

The chase is not defined!

Extending the Chase

To find a repair, we have to extend the chase procedure

Extending the Chase

Problem

The chase fails when meeting two different constants or constants in the consequence of QIDs

Extending the Chase

Problem

The chase fails when meeting two different constants or constants in the consequence of QIDs

Ideas

Modify the chase procedure to

1. choose between different constants when QIDs are fired
   - based on some additional information
2. overwrite values based on constants in the consequence of QIDs
3. replace different constants with a special value, if no additional information is available

Extending the Chase

Problem

The chase fails when meeting two different constants or constants in the consequence of QIDs

Ideas

Modify the chase procedure to

1. choose between different constants when QIDs are fired
   - based on some additional information
2. overwrite values based on constants in the consequence of QIDs
3. replace different constants with a special value, if no additional information is available

Note

This modifications should happen locally and not over the entire table

The Extended Chase

The Chase Procedure

Input: a database D, possibly with labelled nulls representing missing values

1. Initialize $D' = D$
2. As long as there is a QID φ and tuples $t_1, t_2 \in D'$ for which φ can be fired do
   2.1 If $t_1[C] = \texttt{null}_i$ and $t_2[D] = c$ is a constant, replace $\texttt{null}_i$ in every tuple in D' with c
   2.2 If $t_1[C] = \texttt{null}_i$ and $t_2[D] = \texttt{null}_j$, replace $\texttt{null}_j$ in every tuple in D' with $\texttt{null}_i$
   2.3 If $t_1[C] = c$ and $t_2[D] = d$ are both constants, then report failure

The Extended Chase Procedure

Input: a database D, possibly with labelled nulls representing missing values

1. Initialize $D' = D$
2. As long as there is a QID φ and tuples $t_1, t_2 \in D'$ for which φ can be fired do
   - 2.1 If $t_1[C] = \texttt{null}_i$ and $t_2[D] = c$ is a constant, replace $\texttt{null}_i$ in every tuple in D' with c
   - 2.2 If $t_1[C] = \texttt{null}_i$ and $t_2[D] = \texttt{null}_j$, replace $\texttt{null}_j$ in every tuple in D' with $\texttt{null}_i$
   - 2.3 If $t_1[C] = c$ and $t_2[D] = d$ are both constants, then ~~report failure~~

The Extended Chase

The Extended Chase Procedure

Input: a database D, possibly with labelled nulls representing missing values

1. Initialize $D' = D$
2. As long as there is a QID φ and tuples $t_1, t_2 \in D'$ for which φ can be fired do
   - 2.1 If $t_1[C] = \texttt{null}_i$ and $t_2[D] = c$ is a constant, replace $\texttt{null}_i$ in every tuple in D' with c
   - 2.2 If $t_1[C] = \texttt{null}_i$ and $t_2[D] = \texttt{null}_j$, replace $\texttt{null}_j$ in every tuple in D' with $\texttt{null}_i$
   - 2.3 If $t_1[C] = c$ and $t_2[D] = d$ are both constants, then
     - If the consequence of φ is an equality with a constant e, i.e. $t_1[C] =_e t_2[D]$, then assign $t_1[C]$ and $t_2[D]$ value e
     - If additional information indicates a value for c and d, then assign this value to $t_1[C]$ and $t_2[D]$
     - If no information is available, then symbolically unify $t_1[C]$ and $t_2[D]$ by means of a special symbol

Chasing with the Extended Chase

Example (Extended Chase)

Key constraint: $\varphi = \text{Student}[\text{Id} \rightarrow \text{Name}]$

Chasing with the Extended Chase

Example (Extended Chase)

Key constraint: $\varphi = \text{Student}[\text{Id} \rightarrow \text{Name}]$

Dirty Database

Relation Student

Id	Name
123	Volta
123	Marconi
456	Avogadro
789	Fermi

Firing φ →

Marconi is preferred

Relation Student

Id	Name
123	Marconi
456	Avogadro
789	Fermi

Firing φ →

Vorta is preferred

Relation Student

Id	Name
123	Volta
456	Avogadro
789	Fermi

Firing φ →

None is preferred

Relation Student

Id	Name
123	special symbol $\perp_1$
456	Avogadro
789	Fermi

Chasing with the Extended Chase

Example (Extended Chase)

Key constraint: $\varphi = \text{Student}[\text{Id} \rightarrow \text{Name}]$

Challenge

How do we obtain additional information to resolve conflicts between different constants?

Dirty Database

Relation Student

Id	Name
123	Volta
123	Marconi
456	Avogadro
789	Fermi

Firing φ

Marconi is preferred

Relation Student

Id	Name
123	Marconi
456	Avogadro
789	Fermi

Firing φ

Vorta is preferred

Relation Student

Id	Name
123	Volta
456	Avogadro
789	Fermi

Firing φ

None is preferred

Relation Student

Id	Name
123	special symbol $\perp_1$
456	Avogadro
789	Fermi

Repairing with QIDs

Chasing with Functional Dependencies

Key Ideas

- Use a V-repair cost function to choose between values when chasing

$$\text{cost}(D', D) = \sum_{\substack{t' \in D', t \in D \\ t \to t'}} \sum_{\text{Attribute } A} w(t, A) \cdot \text{dist}(t[A], t'[A])$$

- As before, only local changes are done
- The result may contain special symbols in case no clear choice can be made

Chasing with Functional Dependencies

Example

fd_1 : Address[zip → city] fd_2 : Address[name, street, city → phn]

	CC	AC	phn	name	street	city	zip
t_1 :	44	131	1234567	Mike	Mayfield	EDI	EH4 8LE
t_2 :	44	131	3456789	Alex	Crichton	NYC	EH4 8LE
t_3 :	44	131	5678910	Alex	Crichton	EDI	EH4 8LE
t_4 :	01	908	3456789	Jane	Mth Ave	NYC	07974

Chasing with Functional Dependencies

Example

fd_1 : Address[zip $\rightarrow$ city] fd_2 : Address[name, street, city $\rightarrow$ phn]

	CC	AC	phn	name	street	city	zip
t_1 :	44	131	1234567	Mike	Mayfield	EDI	EH4 8LE
t_2 :	44	131	3456789	Alex	Crichton	NYC	EH4 8LE
t_3 :	44	131	5678910	Alex	Crichton	EDI	EH4 8LE
t_4 :	01	908	3456789	Jane	Mth Ave	NYC	07974

Chasing with Functional Dependencies

Example

fd_1 : Address[zip → city] fd_2 : Address[name, street, city → phn]

	CC	AC	phn	name	street	city	zip
t_1 :	44	131	1234567	Mike	Mayfield	EDI	EH4 8LE
t_2 :	44	131	3456789	Alex	Crichton	NYC	EH4 8LE
t_3 :	44	131	5678910	Alex	Crichton	EDI	EH4 8LE
t_4 :	01	908	3456789	Jane	Mth Ave	NYC	07974

- The extended chase chooses between EDI and NYC based on the minimal number of incurred changes (V-repair cost function): In this case EDI

Example

fd_1 : Address[zip → city] fd_2 : Address[name, street, city → phn]

	CC	AC	phn	name	street	city	zip
t_1 :	44	131	1234567	Mike	Mayfield	EDI	EH4 8LE
t_2 :	44	131	3456789	Alex	Crichton	EDI	EH4 8LE
t_3 :	44	131	5678910	Alex	Crichton	EDI	EH4 8LE
t_4 :	01	908	3456789	Jane	Mth Ave	NYC	07974

- The extended chase chooses between EDI and NYC based on the minimal number of incurred changes (V-repair cost function): In this case EDI

Chasing with Functional Dependencies

Example

fd_1 : Address[zip → city] fd_2 : Address[name, street, city → phn]

	CC	AC	phn	name	street	city	zip
t_1 :	44	131	1234567	Mike	Mayfield	EDI	EH4 8LE
t_2 :	44	131	3456789	Alex	Crichton	EDI	EH4 8LE
t_3 :	44	131	5678910	Alex	Crichton	EDI	EH4 8LE
t_4 :	01	908	3456789	Jane	Mth Ave	NYC	07974

- The extended chase chooses between EDI and NYC based on the minimal number of incurred changes (V-repair cost function): In this case EDI
- If the extended chase has no real information to choose between the two phone numbers, then a special symbol is written

Chasing with Functional Dependencies

Example

fd_1 : Address[zip $\rightarrow$ city] fd_2 : Address[name, street, city $\rightarrow$ phn]

	CC	AC	phn	name	street	city	zip
t_1 :	44	131	1234567	Mike	Mayfield	EDI	EH4 8LE
t_2 :	44	131	special symbol	Alex	Crichton	EDI	EH4 8LE
t_3 :	44	131	special symbol	Alex	Crichton	EDI	EH4 8LE
t_4 :	01	908	3456789	Jane	Mth Ave	NYC	07974

- The extended chase chooses between EDI and NYC based on the minimal number of incurred changes (V-repair cost function): In this case EDI
- If the extended chase has no real information to choose between the two phone numbers, then a special symbol is written

Example

fd_1 : Address[zip $\rightarrow$ city] fd_2 : Address[name, street, city $\rightarrow$ phn]

	CC	AC	phn	name	street	city	zip
t_1 :	44	131	1234567	Mike	Mayfield	EDI	EH4 8LE
t_2 :	44	131	3456789	Alex	Crichton	EDI	EH4 8LE
t_3 :	44	131	3456789	Alex	Crichton	EDI	EH4 8LE
t_4 :	01	908	3456789	Jane	Mth Ave	NYC	07974

- The extended chase chooses between EDI and NYC based on the minimal number of incurred changes (V-repair cost function): In this case EDI
- If the extended chase has no real information to choose between the two phone numbers, then a special symbol is written
- If the extended chase has information to choose between the two phone numbers, then that phone number is selected

Chasing with Conditional Functional Dependencies

The Extended Chase and CFDs

Can the same approach still be applied for CFDs?

Chasing with Conditional Functional Dependencies

The Extended Chase and CFDs

Can the same approach still be applied for CFDs?

Example

$\text{cfd}_1 : R[C = c_1 \rightarrow B = b_1]$ $\text{cfd}_2 : R[C = c_2 \rightarrow B = b_2]$ $\text{cfd}_3 : R[A \rightarrow B]$

	A	B	C
t_1 :	a	b_1	c_1
t_2 :	a	b_2	c_2

- cfd_1 and cfd_2 are satisfied

Chasing with Conditional Functional Dependencies

The Extended Chase and CFDs

Can the same approach still be applied for CFDs?

Example

$\text{cfd}_1 : R[C = c_1 \rightarrow B = b_1]$ $\text{cfd}_2 : R[C = c_2 \rightarrow B = b_2]$ $\text{cfd}_3 : R[A \rightarrow B]$

	A	B	C
t_1 :	a	b_1	c_1
t_2 :	a	b_2	c_2

- cfd_1 and cfd_2 are satisfied
- cfd_3 can be fired, suppose that b_1 is preferred over b_2

Chasing with Conditional Functional Dependencies

The Extended Chase and CFDs

Can the same approach still be applied for CFDs?

Example

$\text{cfd}_1: R[C = c_1 \rightarrow B = b_1]$ $\text{cfd}_2: R[C = c_2 \rightarrow B = b_2]$ $\text{cfd}_3: R[A \rightarrow B]$

	A	B	C
t_1 :	a	b_1	c_1
t_2 :	a	b_1	c_2

- ~~cfd_1 and cfd_2 are satisfied~~
- cfd_3 can be fired, suppose that b_1 is preferred over b_2
- But now cfd_2 is no longer satisfied

Chasing with Conditional Functional Dependencies

The Extended Chase and CFDs

Can the same approach still be applied for CFDs?

Example

$\text{cfd}_1: R[C = c_1 \rightarrow B = b_1]$ $\text{cfd}_2: R[C = c_2 \rightarrow B = b_2]$ $\text{cfd}_3: R[A \rightarrow B]$

	A	B	C
t_1 :	a	b_1	c_1
t_2 :	a	b_1	c_2

- ~~cfd_1 and cfd_2 are satisfied~~
- cfd_3 can be fired, suppose that b_1 is preferred over b_2
- But now cfd_2 is no longer satisfied

Conclusion

The extended chase can be used with CFDs but does not always lead to a repair

Chasing with Matching Dependencies

The Extended Chase and MDs

Can the same approach be applied for MDs?

The Extended Chase and MDs

Can the same approach be applied for MDs?

Yes!

- The chase proceeds as for functional dependencies
- except that it takes into account the similarity relations when firing a QID

Repairing in the Presence of Master Data

Master Data and Certified Attributes

Problem

- We have seen that the extended chase does not always know how to resolve errors
- And sometimes multiple choices may be available

More Information is Required

The user needs to provide more information to the chase:

- Master Data: reference data that is trusted and clean
- Certified Attributes: attributes whose values are assured to be correct

Chasing with Master Data and Certified Attributes

Quality Improving Dependency with Master Data

$$\forall t \forall t_m \Big(\big(R(t) \wedge R_m(t_m) \wedge \bigwedge_{i \in [1,n]} t[A_i] \ \mathtt{op}_i \ t_m[B_i] \big) \rightarrow \bigwedge_{j \in [1,\ell]} t[C_j] \ \mathtt{op}'_j \ t_m[D_j] \Big)$$

where R_m is the master data (for relation R)

Adapting the Chase

- The values of the master data are always preferred; and
- QIDs are fired only when attributes in the premise are certified

Chasing with Master Data and Certified Attributes – Example

Examples

$$\forall t \forall t_m \Big(\big(\mathsf{Address}(t) \wedge \mathsf{Address}_m(t_m) \wedge t[\mathsf{zip}] = t_m[\mathsf{zip}]\big) \\ \rightarrow \big(t[\mathsf{AC}] = t_m[\mathsf{AC}] \wedge t[\mathsf{street}] = t_m[\mathsf{street}] \wedge t[\mathsf{city}] = t_m[\mathsf{city}]\big)\Big)$$

$$\forall t \forall t_m \Big(\big(\mathsf{Address}(t) \wedge \mathsf{Address}_m(t_m) \wedge t[\mathsf{phn}] = t_m[\mathsf{phn}] \wedge t[\mathsf{type}] = 2\big) \\ \rightarrow \big(t[\mathsf{FN}] = t_m[\mathsf{FN}] \wedge t[\mathsf{LN}] = t_m[\mathsf{LN}]\big)\Big)$$

Chasing with Master Data and Certified Attributes

Chasing with Master Data and Certified Attributes

- Provides a uniform way of repairing data for QIDs
- By selecting certified attributes carefully, one can impose that only a unique repair is obtained
  - this is called a certain fix

Challenges

- Finding a "good" set of certified attributes (certain regions)
- How to repair incrementally
  - for instance, when data or QIDs are updated

Confidence-Based Repairing

Key Idea of Confidence-Based Repairing

- Annotate attribute/values with confidence values (how sure one is that a value is correct)
- During the chase, these confidence values get propagated
- A QID is fired only if the confidence of the involved values does not decrease
- In this way, each chase step improves the quality of the data
  - as measured by the confidence values

Summary

Repairing

- The extended chase alone is a first step towards a clean and elegant repair algorithm
- In the presence of master data, one often finds better quality repairs
- Although this approach shows promise in practice, the properties of the extended chase are not fully understood yet and further investigation is necessary

Summary

Repairing

- The extended chase alone is a first step towards a clean and elegant repair algorithm
- In the presence of master data, one often finds better quality repairs
- Although this approach shows promise in practice, the properties of the extended chase are not fully understood yet and further investigation is necessary

Take away message

Data repairing: a rich source of problems and challenges