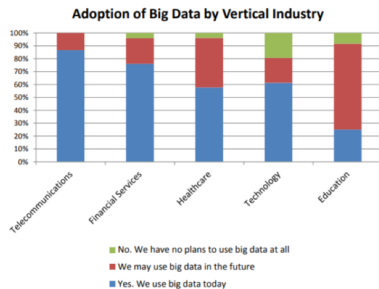


# Data Quality

Floris Geerts  
University of Antwerp

- Data is used **everywhere** and **abundant** (“big data”).



Source: Dresner Advisory Services LLC, Big Data Analytics Market Survey, 2017

- Data is **valuable**.

*“ The world’s most valuable resource is no longer oil, but data ” (The economist, 2017)*

- ▶ Data underlies **decision making**.

*“ Without algorithms and data at the center of their decision-making, traditional companies cannot compete with today’s digital giants.” (Chief Executive Magazine, 2019)*

- ▶ Data is key for **data analytics** and **machine learning**, i.e., to derive value from data.

*“If you train your machine learning models with garbage, it’s no surprise you’ll get garbage results.” (The Achilles Heel of AI, Forbes 2019)*

## The need for clean data

- ▶ Value derived from data is as good as the data itself.
- ▶ **Garbage in, garbage out.**

- ▶ Poor data costs US companies **\$600 billions** annually.
  - ▶ Erroneously priced data in retail databases costs US customers **\$2.5 billion** each year.
  - ▶ **30% – 80%** of the development time for data cleaning in a data integration project and most AI and Machine Learning projects.
  - ▶ The market for AI and machine learning relevant data preparation solutions is over **\$500 million** in 2018 growing to **\$1.2 billion** by end of 2023.
- ▶ This is true in all sectors dealing with data.

### Important problem

- ▶ Data quality: A very important problem for data management, data engineers and data scientists (and everyone else...).

- **Dirty data** comes in **many different forms...**

### Dirty data

NI#	name	DoB	Age	Gender	phone	zip
1234	Smith, John	18/2/1980	30	M	null	70567
1234	Jane Doe	32/4/1970	47	F	768-4511	5555
1235	John Smith	18/2/1980	37	M	567-3211	70567

zip	city
70567	Stuttgart
70567	Stuttgart
70569	Germany

(Date of example: 2017)

- **Dirty data** comes in **many different forms...**

## Dirty data

NI#	name	DoB	Age	Gender	phone	zip
1234	Smith, John	18/2/1980	30	M	null	70567
1234	Jane Doe	32/4/1970	47	F	768-4511	5555
1235	John Smith	18/2/1980	37	M	567-3211	70567

zip	city
70567	Stuttgart
70567	Stuttgart
70569	Germany

Typo

(Date of example: 2017)

- **Dirty data** comes in **many different forms...**

## Dirty data

NI#	name	DoB	Age	Gender	phone	zip
1234	Smith, John	18/2/1980	30	M	null	70567
1234	Jane Doe	32/4/1970	47	F	768-4511	5555
1235	John Smith	18/2/1980	37	M	567-3211	70567

zip	city
70567	Stuttgart
70567	Stuttgart
70569	Germany

Incorrect value

Typo

(Date of example: 2017)

- **Dirty data** comes in **many different forms...**

## Dirty data

### Representation

NI#	name	DoB	Age	Gender	phone	zip
1234	Smith, John	18/2/1980	30	M	null	70567
1234	Jane Doe	32/4/1970	47	F	768-4511	5555
1235	John Smith	18/2/1980	37	M	567-3211	70567

zip	city
70567	Stuttgart
70567	Stuttgart
70569	Germany

### Incorrect value

### Typo

(Date of example: 2017)



- **Dirty data** comes in **many different forms...**

## Dirty data

Representation			Missing value			
NI#	name	DoB	Age	Gender	phone	zip
1234	Smith, John	18/2/1980	30	M	null	70567
1234	Jane Doe	32/4/1970	47	F	768-4511	5555
1235	John Smith	18/2/1980	37	M	567-3211	70567

zip	city
70567	Stuttgart
70567	Stuttgart
70569	Germany

Incorrect value

Typo

(Date of example: 2017)

- **Dirty data** comes in **many different forms...**

## Dirty data

Representation			Missing value			
NI#	name	DoB	Age	Gender	phone	zip
1234	Smith, John	18/2/1980	30	M	null	70567
1234	Jane Doe	32/4/1970	47	F	768-4511	5555
1235	John Smith	18/2/1980	37	M	567-3211	70567

zip	city
70567	Stuttgart
70567	Stuttgart
70569	Germany

(Date of example: 2017)

- **Dirty data** comes in **many different forms...**

## Dirty data

Representation

Inconsistency

Missing value

Key violation

NI#	name	DoB	Age	Gender	phone	zip
1234	Smith, John	18/2/1980	30	M	null	70567
1234	Jane Doe	32/4/1970	47	F	768-4511	5555
1235	John Smith	18/2/1980	37	M	567-3211	70567

zip	city
70567	Stuttgart
70567	Stuttgart
70569	Germany

Incorrect value

Typo

(Date of example: 2017)

- Dirty data

(Date of example: 2017)

- Representation   Inconsistency   Missing value

NI#	name	DoB	Age	Gender	phone	zip
1234	Smith, John	18/2/1980	30	M	null	70567
1234	Jane Doe	32/4/1970	47	F	768-4511	5555
1235	John Smith	18/2/1980	37	M	567-3211	70567

Incorrect value

## Referential integrity

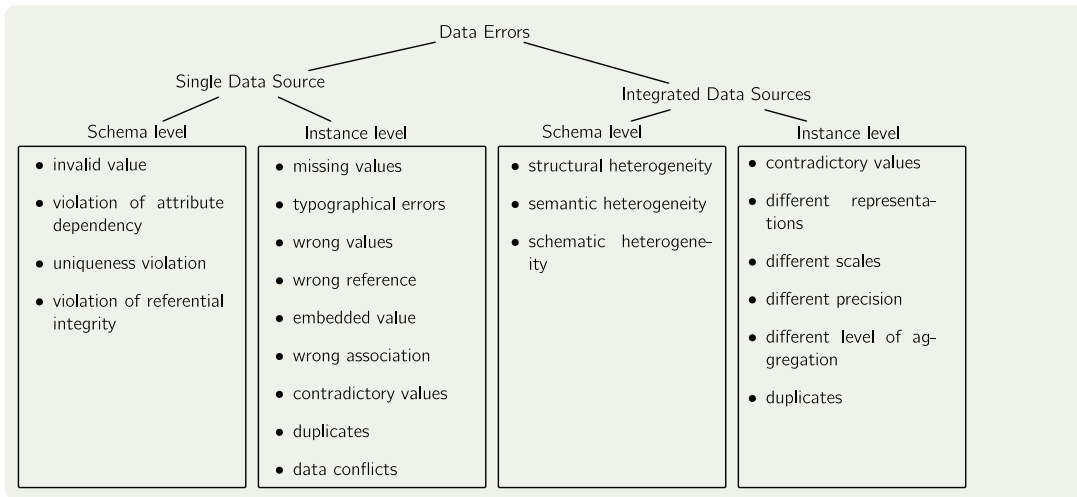
(Date of example: 2017)


- | Representation |             | Inconsistency |     | Missing value |          |       |
|----------------|-------------|---------------|-----|---------------|----------|-------|
| NI#            | name        | DoB           | Age | Gender        | phone    | zip   |
| 1234           | Smith, John | 18/2/1980     | 30  | M             | null     | 70567 |
| 1234           | Jane Doe    | 32/4/1970     | 47  | F             | 768-4511 | 5555  |
| 1235           | John Smith  | 18/2/1980     | 37  | M             | 567-3211 | 70567 |

zip	city
70567	Stuttgart
70567	Stuttgart
70569	Germany

- EDBT-INTENDED – Bordeaux



 E. Rahm & H.H. Do, *Data Cleaning: Problems and Current Approaches*, IEEE Data Eng. Bull, 2000.

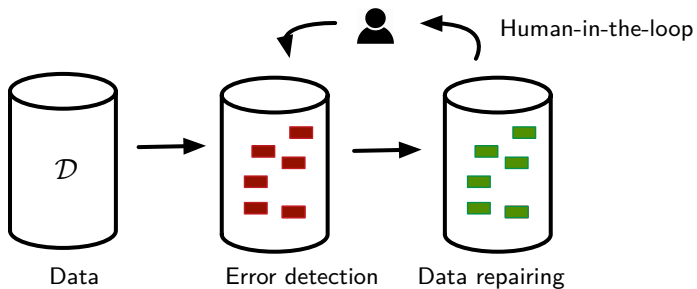
### Sources of dirty data?

- ▶ At time of **data entry**: human error, crowd, imprecise signal/sensors, ....
  - ▶ Conflicts when data is **merged**, **copied** and/or **integrated**.
  - ▶ **Updated** data or **not updated** data at all (staleness).
  - ▶ ...
- ▶ Best solution for ensuring good data quality is **prevention!!**

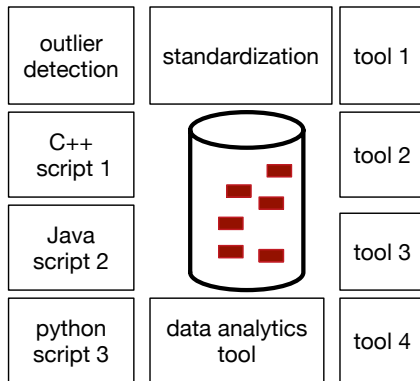
Unfortunately, irregardless of all efforts, dirty data is **everywhere** and **data cleaning** is needed!



- ▶ A **high-level** – and very simplistic– **overview** of the overall **data cleaning pipeline**:



- ▶ There is **no one-size-fits-all** solution to data quality.



- ▶ **Understanding tools** and **code** is difficult.
- ▶ **Different components** hard to **integrate**.
- ▶ **Interpreting results** and **debugging** is almost impossible.

- ▶ A **more principled** approach to data cleaning is desirable!
- ▶ The cleaning process should be **augmented** with a **specification indicating which data is dirty/clean**.
- ▶ The specification should be used **uniformly** across the **entire data quality pipeline**.
- ▶ Specification should (preferably) be **declarative**,<sup>1</sup> allowing to focus on semantics rather than implementation details and increases **explainability**.

---

<sup>1</sup>Declarative: Logic  $\leftrightarrow$  Procedural: Algebra

- ▶ I'll next give a **general overview** of the declarative approach to data quality.
- ▶ Focus on **relational data** only.
- ▶ Mostly **deterministic**, i.e., pieces of data are either dirty or not.
  - 👉 Benny's lecture for a more fine-grained quantitative approach to violations.
  - 👉 Antoine's lecture for a probabilistic model for dirty data.
- ▶ Provide a **recipe** of how to use this approach for **your own data quality problem**.

## Declarative Approach

Key Challenges

Static Analysis

Error Detection

Constraint Discovery

Data Repairing

Conclusion

## Key idea



Express **semantic properties** of clean data as **logical expressions**.

- ▶ Typically, small fragments of **first-order logic** are considered.
- ▶ Sometimes, **built-in predicates** are added.
- ▶ Tradeoff between **simplicity** of expressions, **expressive power** and **computational complexity** of decision problems (satisfiability, implication).
- ▶ In the context of data quality:

**logical sentences = data quality constraints**

In this talk, we use the **tuple relational calculus** (TRC) and **extensions thereof** instead of first-order logic to express constraints.

## Functional dependency

$$\forall r, s (R(r) \wedge R(s) \wedge r[A] = s[A] \rightarrow r[B] = s[B]).$$

## Definition (TRC formulas)

<b>Alphabet:</b>	<ul style="list-style-type: none"><li>– Relation names <math>R, S, T, \dots</math> with certain attributes</li><li>– Tuple variables <math>r, s, t, \dots</math></li></ul>
<b>Terms:</b>	<ul style="list-style-type: none"><li>– constants</li><li>– <math>t[A]</math> for tuple variable <math>t</math> and attribute <math>A</math></li></ul>
<b>Atomic formulas:</b>	<ul style="list-style-type: none"><li>– <math>R(r)</math> for relation name <math>R</math> and tuple variable <math>r</math></li><li>– <math>u_1 = u_2</math> when <math>u_1</math> and <math>u_2</math> are terms</li></ul>
<b>Formulas:</b>	<ul style="list-style-type: none"><li>– Atomic formulas</li><li>– If <math>\varphi_1</math> and <math>\varphi_2</math> are formulas, then <math>\neg\varphi_1</math>, <math>\varphi_1 \wedge \varphi_2</math>, <math>\varphi_1 \vee \varphi_2</math>, <math>\varphi_1 \rightarrow \varphi_2</math> are formulas.</li><li>– If <math>\varphi</math> is a formula, then <math>\exists r\varphi</math> and <math>\forall r\varphi</math> are formulas.</li></ul>

- ▶ The **semantics** of TRC formulas on a database instance is defined inductively, as one would expect.
- ▶ **Sentences** are TRC formulas that evaluate to **true** or **false**.

### Constraints

TRC sentences will be called **constraints** from here on.

### Definition (Satisfaction)

- ▶ Let  $\mathcal{D}$  be a database and  $\varphi$  be a constraint, then  $\mathcal{D} \models \varphi$  denotes that  $\mathcal{D}$  **satisfies**  $\varphi$ , or  $\varphi$  evaluates to true on  $\mathcal{D}$ .
- ▶ If  $\Sigma$  is a **set** of constraints,  $\mathcal{D} \models \Sigma$  if  $\mathcal{D} \models \varphi$  for **all**  $\varphi \in \Sigma$ .

### Functional dependency

$$\varphi = \forall r, s (R(r) \wedge R(s) \wedge r[A] = s[A] \rightarrow r[B] = s[B]).$$

Its **semantics**: If  $\mathcal{D} \models \varphi$  then all tuples in  $\mathcal{D}$  with the same  $A$ -attribute value have the same  $B$ -attribute value.



- ▶ The **semantics** of TRC formulas on a database instance is defined inductively, as one would expect.
- ▶ **Sentences** are TRC formulas that evaluate to **true** or **false**.

## Constraints

TRC sentences will be called **constraints** from here on.

## Definition (Satisfaction)

- ▶ Let  $\mathcal{D}$  be a database and  $\varphi$  be a constraint, then  $\mathcal{D} \models \varphi$  denotes that  $\mathcal{D}$  **satisfies**  $\varphi$ , or  $\varphi$  evaluates to true on  $\mathcal{D}$ .
- ▶ If  $\Sigma$  is a **set** of constraints,  $\mathcal{D} \models \Sigma$  if  $\mathcal{D} \models \varphi$  for **all**  $\varphi \in \Sigma$ .

## Functional dependency

$$\varphi = \forall r, s (R(r) \wedge R(s) \wedge r[A] = s[A] \rightarrow r[B] = s[B]).$$

Its **semantics**: If  $\mathcal{D} \models \varphi$  then all tuples in  $\mathcal{D}$  with the same  $A$ -attribute value have the same  $B$ -attribute value.

Our point of view to data quality:

### Clean/Dirty Data

Given:

- ▶ a database  $\mathcal{D}$ ; and
- ▶ a set  $\Sigma$  of constraints,

we say that  $\mathcal{D}$  is  $\Sigma$ -**clean** (or simply **clean**) if  $\mathcal{D} \models \Sigma$ . Otherwise,  $\mathcal{D}$  is said to be **dirty**.

This definition promotes:

- ▶ First-order logic as a specification language for data quality.
- ▶ In practice, full first-order logic is not used and constraints are often of a **simple form**.

### Violations

When  $\mathcal{D}$  is dirty, i.e.,  $\mathcal{D} \not\models \Sigma$ , and hence there must be tuples **causing** some constraints not to be satisfied. These will be the **errors**!

Let  $\varphi$  be a constraint and  $\mathcal{D}$  a database.

### Definition (Errors of a constraint)

A tuple  $s \in \mathcal{D}$  is an **error** for  $\varphi$

- ▶ if there exists a set  $S = \{s_1, \dots, s_k\}$  of tuples in  $\mathcal{D}$  such that  $S \cup \{s\} \not\models \varphi$  while  $S \models \varphi$ .

The set **Error** $(\varphi, \mathcal{D})$  consists of all errors for  $\varphi$  in  $\mathcal{D}$ .

Can be extended to tuple/attribute level granularity.

## Functional dependencies

Consider the constraint

$$\varphi : \forall t_1, t_2 \left( cust(t_1) \wedge cust(t_2) \wedge t_1[NI\#] = t_2[NI\#] \rightarrow \bigwedge_{B \in \{ac, phn, name, street, city, zip\}} t_1[B] = t_2[B] \right)$$

stating that  $Ni\#$  is a **key**: there is a unique record for each distinct  $Ni\#$ .

Consider instance  $\mathcal{D}$ :

$Ni\#$	ac	phn	name	street	city	zip
SC1234566	131	1234567	M. Smith	Mayfield	EDI	EH4 8LE
SC1234566	020	1234567	M. Smith	Portland	LDN	W1B 1JL

- ▶  $\mathcal{D}$  does not satisfy (or violates) the FD, i.e.,  $\mathcal{D} \not\models \varphi$ .
- ▶  $\text{Error}(\varphi, \mathcal{D})$  consist of both tuples.
- ▶ According to the definition, for the first tuple  $s$ ,  $S$  is the other tuple:  $\{s\} \models \varphi$  but  $S \cup \{s\} \not\models \varphi$ .

## Functional dependencies

Consider the constraint

$$\varphi : \forall t_1, t_2 \left( cust(t_1) \wedge cust(t_2) \wedge t_1[NI\#] = t_2[NI\#] \rightarrow \bigwedge_{\substack{B \in \{ac, phn, \\ name, street, city, zip\}}} t_1[B] = t_2[B] \right)$$

stating that  $Ni\#$  is a **key**: there is a unique record for each distinct  $Ni\#$ .

Consider instance  $\mathcal{D}$ :

$Ni\#$	ac	phn	name	street	city	zip
SC1234566	131	1234567	M. Smith	Mayfield	EDI	EH4 8LE
SC1234566	020	1234567	M. Smith	Portland	LDN	W1B 1JL

- ▶  $\mathcal{D}$  does not satisfy (or violates) the FD, i.e.,  $\mathcal{D} \not\models \varphi$ .
- ▶  $\text{Error}(\varphi, \mathcal{D})$  consist of both tuples.
- ▶ According to the definition, for the first tuple  $s$ ,  $S$  is the other tuple:  $\{s\} \models \varphi$  but  $S \cup \{s\} \not\models \varphi$ .

### Recipe for linking cleanliness and constraints

1. Identify what **kind of dirtiness** you want to capture.
2. Identify a **class of constraints** such that its errors correspond to what you want.
  - ▶ E.g., key violations  $\Rightarrow$  functional dependency constraints.

We see some more examples next.

## Local dependencies

- ▶ In UK ( $CC = 44$ ), Zip code determines street.
- ▶ You know that EDI in the UK has zip code 131.
- ▶ You know that MH in the US ( $CC = 01$ ) has zip code 908.

Consider instance  $\mathcal{D}$ :

CC	AC	phn	name	street	city	zip
44	131	1234567	Mike	Mayfield	NYC	EH4 8LE
44	131	3456789	Rick	Crichton	NYC	EH4 8LE
01	908	3456789	Joe	Mtn Ave	NYC	07974

- ▶ How to express these as constraints?
- ▶ Allow equality with constants in FD-like constraints.
- ▶ Called **conditional functional dependencies**.

[Wenfei Fan et al. Conditional functional dependencies for capturing data inconsistencies. ACM TODS 2008.]

## Local dependencies

- ▶ In UK ( $CC = 44$ ), Zip code determines street.
- ▶ You know that EDI in the UK has zip code 131.
- ▶ You know that MH in the US ( $CC = 01$ ) has zip code 908.

Consider instance  $\mathcal{D}$ :

CC	AC	phn	name	street	city	zip
44	131	1234567	Mike	Mayfield	NYC	EH4 8LE
44	131	3456789	Rick	Crichton	NYC	EH4 8LE
01	908	3456789	Joe	Mtn Ave	NYC	07974

- ▶ How to express these as constraints?
- ▶ Allow **equality with constants** in FD-like constraints.
- ▶ Called **conditional functional dependencies**.

 Wenfei Fan et al. Conditional functional dependencies for capturing data inconsistencies. ACM TODS 2008.




## Local dependencies

- ▶ In UK ( $CC = 44$ ), Zip code determines street.
- ▶ You know that EDI in the UK has zip code 131.
- ▶ You know that MH in the US ( $CC = 01$ ) has zip code 908.

Consider instance  $\mathcal{D}$ :

CC	AC	phn	name	street	city	zip
44	131	1234567	Mike	Mayfield	NYC	EH4 8LE
44	131	3456789	Rick	Crichton	NYC	EH4 8LE
01	908	3456789	Joe	Mtn Ave	NYC	07974

- ▶ How to express these as constraints?
- ▶ Allow **equality with constants** in FD-like constraints.
- ▶ Called **conditional functional dependencies**.

 Wenfei Fan et al. Conditional functional dependencies for capturing data inconsistencies. ACM TODS 2008.

## Conditional functional dependencies

$\text{cfd}_1: \forall t_1, t_2 ( \text{cust}(t_1) \wedge \text{cust}(t_2) \wedge t_1[\text{CC}] = t_2[\text{CC}] = "44" \rightarrow t_1[\text{street}] = t_2[\text{street}] )$

$\text{cfd}_2: \forall t_1 ( \text{cust}(t) \wedge t_1[\text{CC}] = "44" \wedge t_1[\text{AC}] = "131" \rightarrow t_1[\text{city}] = "EDI" )$

$\text{cfd}_3: \forall t_1 ( \text{cust}(t) \wedge t_1[\text{CC}] = "01" \wedge t_1[\text{AC}] = "908" \rightarrow t_1[\text{city}] = "MH" )$

Consider instance  $\mathcal{D}$ :

CC	AC	phn	name	street	city	zip
44	131	1234567	Mike	Mayfield	NYC	EH4 8LE
44	131	3456789	Rick	Crichton	NYC	EH4 8LE
01	908	3456789	Joe	Mtn Ave	NYC	07974

- $\mathcal{D}$  does not satisfy the CFDs, i.e.,

$$\mathcal{D} \notin \{\text{cfd}_1, \text{cfd}_2, \text{cfd}_3\}.$$

- Errors in all tuples.

### A typical salary situation

Records for Employees:

Name	Job	Years	Salary
Mark	Senior Programmer	15	35K
Edith	Junior Programmer	7	22K
Josh	Senior Programmer	11	50K
Ann	Junior Programmer	6	38K

We want to ensure:

*"The salary of an employee is greater than other employees who have junior job titles, or the same job title but less experience in the company."*

- ▶ How to express these as constraints?
- ▶ Allow built-in predicates such as  $<$  in constraints.
- ▶ Simple form: **Ordered functional dependencies**

[W. Ng, *Ordered functional dependencies in relational databases* Information Systems, 1999.

## A typical salary situation

Records for Employees:

Name	Job	Years	Salary
Mark	Senior Programmer	15	35K
Edith	Junior Programmer	7	22K
Josh	Senior Programmer	11	50K
Ann	Junior Programmer	6	38K

We want to ensure:

*"The salary of an employee is greater than other employees who have junior job titles, or the same job title but less experience in the company."*

- ▶ How to express these as constraints?
- ▶ Allow **built-in predicates** such as  $<$  in constraints.
- ▶ Simple form: **Ordered functional dependencies**

 W. Ng, *Ordered functional dependencies in relational databases* Information Systems, 1999.

## A typical salary situation


Records for Employees:

Name	Job	Years	Salary
Mark	Senior Programmer	15	35K
Edith	Junior Programmer	7	22K
Josh	Senior Programmer	11	50K
Ann	Junior Programmer	6	38K

We want to ensure:

*"The salary of an employee is greater than other employees who have junior job titles, or the same job title but less experience in the company."*

- ▶ How to express these as constraints?
- ▶ Allow **built-in predicates** such as  $<$  in constraints.
- ▶ Simple form: **Ordered functional dependencies**

 W. Ng, *Ordered functional dependencies in relational databases* Information Systems, 1999.

### Ordered functional dependencies

Assume that the domain of Job titles is **ordered**: “Junior Programmer” < “Senior Programmer”, then

$$\forall s, t (emp(s) \wedge emp(t) \wedge s[Job] > t[Job] \rightarrow s[Salary] > t[Salary])$$

expresses that

“the salary of an employee is greater than other employees who have junior job titles”.

Similarly,

$$\forall s, t (emp(s) \wedge emp(t) \wedge s[Job] = t[Job] \wedge s[Years] > t[Years] \rightarrow s[Salary] > t[Salary])$$

expresses that

“the salary for employees with the same job title is greater for those with more years in the company.”

### Discrepancies in movie durations

Integrated Movie database:

Source	Title	Duration
movies.aol.com	Aliens	110
finnguide.fi	Aliens	112
amazon.com	Clockwork Orange	140
movie-vault.com	A Beautiful Mind	144
walmart.com	Beautiful Mind	145
tesco.com	Clockwork Orange	131

We want to ensure:

*"Different durations of the same movie in the database may not exceed 6 minutes."*

### Discrepancies in movie durations

Integrated Movie database:

Source	Title	Duration
movies.aol.com	Aliens	110
finnguide.fi	Aliens	112
amazon.com	Clockwork Orange	140
movie-vault.com	A Beautiful Mind	144
walmart.com	Beautiful Mind	145
tesco.com	Clockwork Orange	131

We want to ensure:

*"Different durations of the same movie in the database may not exceed 6 minutes."*



### Discrepancies in geo locations

Integrated geo location database

Source	Address	Latitude	Longitude
google	65 N St Apt#C6, SLC	40.770896	-111.864066
geocoder	5 N St Apt#C6, SLC	40.770767	-111.863768
google	50 Cen Camp Dr, SLC	40.758951	-111.845246
geocoder	50 Cen Camp Dr, SLC	40.757599	-111.843995
google	35 S 700 E Apt#3, SLC	40.76837	-111.87064
geocoder	35 S 700 E Apt#3, SLC	40.77833	-111.870869

We want to ensure:

*"The same location should be appear within a specified level of accuracy, say within a circle of radius 0.005"*

► How to express these as constraints?

### Discrepancies in geo locations

Integrated geo location database

Source	Address	Latitude	Longitude
google	65 N St Apt#C6, SLC	40.770896	-111.864066
geocoder	5 N St Apt#C6, SLC	40.770767	-111.863768
google	50 Cen Camp Dr, SLC	40.758951	-111.845246
geocoder	50 Cen Camp Dr, SLC	40.757599	-111.843995
google	35 S 700 E Apt#3, SLC	40.76837	-111.87064
geocoder	35 S 700 E Apt#3, SLC	40.77833	-111.870869

We want to ensure:

*"The same location should be appear within a specified level of accuracy, say within a circle of radius 0.005"*

► How to express these as constraints?

### Discrepancies in geo locations

Integrated geo location database

Source	Address	Latitude	Longitude
google	65 N St Apt#C6, SLC	40.770896	-111.864066
geocoder	5 N St Apt#C6, SLC	40.770767	-111.863768
google	50 Cen Camp Dr, SLC	40.758951	-111.845246
geocoder	50 Cen Camp Dr, SLC	40.757599	-111.843995
google	35 S 700 E Apt#3, SLC	40.76837	-111.87064
geocoder	35 S 700 E Apt#3, SLC	40.77833	-111.870869

We want to ensure:

*"The same location should be appear within a specified level of accuracy, say within a circle of radius 0.005"*

- How to express these as constraints?

- ▶ Allow **built-in distance predicates** such as

- ▶  $\text{dist}_1(x, y) = |x - y|$

- ▶  $\text{dist}_2((x_1, x_2), (y_1, y_2)) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$

in constraints.

- ▶ Simple form: **Metric functional dependencies**

📖 N. Koudas, A. Saha, D. Srivastava, S. Venkatasubramanian, *Metric Functional Dependencies*, ICDE, 2009.

## Metric functional dependencies

Consider constraints:

$$\forall s, t \left( \text{Movie}(s) \wedge \text{Movie}(t) \wedge s[\text{Title}] = t[\text{Title}] \rightarrow \text{dist}_1(s[\text{Duration}], t[\text{Duration}]) \leq 6 \right)$$

for the Movie database, and

$$\forall s, t \left( \text{Loc}(s) \wedge \text{Loc}(t) \wedge s[\text{Addr}] = t[\text{Addr}] \rightarrow \text{dist}_2((s[\text{Lat}], s[\text{Long}]), (t[\text{Lat}], t[\text{Long}])) \leq 0.005 \right)$$

for the Location database.

## Record matching/object identification

- ▶ To **identify** tuples from one or more relations that refer to the **same real-world object**.
- ▶ Common problem in data integration, payment card fraud detection, ...

## Credit card fraud

- ▶ Records for Mark Smith and M. Smith **should match**.

Records for card holders:

FN	LN	address	tel	DoB	gender
Mark	Smith	10 Oak St, EDI, EH8 9LE	3256777	10/12/97	M

Transaction records:

FN	LN	post	phn	when	where	amount
M.	Smith	10 Oak St, EDI, EH8 9LE	null	1pm/7/7/09	EDI	\$3,500
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Max	Smith	PO Box 25, EDI	3256777	2pm/7/7/09	NYC	\$6,300

- ▶ How to express these as constraints?

## Record matching/object identification

- ▶ To **identify** tuples from one or more relations that refer to the **same real-world object**.
- ▶ Common problem in data integration, payment card fraud detection, ...

## Credit card fraud

- ▶ Records for Mark Smith and M. Smith **should match**.

Records for card holders:


FN	LN	address	tel	DoB	gender
Mark	Smith	10 Oak St, EDI, EH8 9LE	3256777	10/12/97	M

Transaction records:

FN	LN	post	phn	when	where	amount
M.	Smith	10 Oak St, EDI, EH8 9LE	null	1pm/7/7/09	EDI	\$3,500
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Max	Smith	PO Box 25, EDI	3256777	2pm/7/7/09	NYC	\$6,300

- ▶ How to express these as constraints?

- ▶ Allow **similarity-based** comparisons in constraints.
- ▶ Simple form: **Matching dependencies**

 Dynamic Constraints for Record Matching, W. Fan, H. Gao, J. Li, X. Jia, and S. Ma, The VLDB Journal, 2011.

## Matching dependency (MD):

“If two entities (tuples) agree on their last name and address and if their first names are similar, then the two tuples should be equal on all other related attributes”

$\forall s, t (\text{card}(s) \wedge \text{trans}(t) \wedge s[LN] = t[LN] \wedge s[\text{address}] = t[\text{post}] \wedge s[FN] \asymp t[FN] \rightarrow s[X] = t[Y])$ ,  
where  $\asymp$  is a **similarity operator** and  $X$  and  $Y$  are compatible attributes in card and trans, respectively.

- ▶ It is not a coincidence that constraints so far are **two-tuple constraints** of the form

$$\forall s, t (R(s) \wedge S(s) \wedge \text{LHS} \rightarrow \text{RHS})$$

with LHS and RHS **conjunctions** of atomic predicates.

- ▶ They are easy to interpret, efficient to reason about, and allow for fast error detection and other computational task (as we see later)
- ▶ Quite popular type of constraints in the data quality context.
- ▶ Can capture common types of errors.



- ▶ Another popular formalism are **denial constraints**:

$$\forall s, t \neg (R(s) \wedge S(t) \wedge \text{COND})$$

where COND is again a conjunction of atomic predicates.

- ▶ **More expressive** than dependencies
- ▶ **Still manageable** in terms of reasoning, efficiency,...

### Example

- ▶ Domain constraint:  $\forall s \neg (R(s) \wedge s[A] \neq \text{"yes"} \wedge s[A] \neq \text{"no"})$
- ▶ "Two people living the same state should have correct tax rates depending on their income":  
$$\forall s, t \neg (R(s) \wedge R(t) \wedge s[AC] = t[AC] \wedge s[SAL] < t[SAL] \wedge s[TR] > t[TR])$$

- ▶ Differential dependencies
  - ▶ Sequential dependencies
  - ▶ Glitch dependencies
  - ▶ Temporal dependencies
  - ▶ Similarity/comparable dependencies
  - ▶ Editing rules
  - ▶ Currency dependencies
  - ▶ Inclusion dependencies
  - ▶ Conditional inclusion dependencies
  - ▶ Conditional denial constraints
  - ▶ Association rules
  - ▶ ....
  - +
  - ▶ Approximate/relaxed versions thereof
- +
  - ▶ Graph versions for data quality in (knowledge) graphs...
  - ▶ +
  - ▶ Dependencies on time series
  - ▶ +
  - ▶ ....
  - ▶ More expressive languages are used for entity resolution, e.g., Datalog in Dedupalog (and more recent declarative approaches to ER),
  - ▶ FO and Markov Logic in Holoclean...

- ▶ Differential dependencies
  - ▶ Sequential dependencies
  - ▶ Glitch dependencies
  - ▶ Temporal dependencies
  - ▶ Similarity/comparable dependencies
  - ▶ Editing rules
  - ▶ Currency dependencies
  - ▶ Inclusion dependencies
  - ▶ Conditional inclusion dependencies
  - ▶ Conditional denial constraints
  - ▶ Association rules
  - ▶ ....
  - +
  - ▶ Approximate/relaxed versions thereof
- +
  - ▶ Graph versions for data quality in (knowledge) graphs...
  - +
  - ▶ Dependencies on time series
  - +
  - ▶ ....
  - ▶ More expressive languages are used for entity resolution, e.g., Datalog in Dedupalog (and more recent declarative approaches to ER),
  - ▶ FO and Markov Logic in Holoclean...

- ▶ We have seen various examples of constraints.
  - ▶ **Key message:** keep them simple
  - ▶ Constraints allow to focus on **semantic properties** of data
  - ▶ **No low-level code** is needed to express dirtiness.
- ▶ Hope I convinced you that **constraints are an elegant way** of describing when data is clean or dirty!

### Next...

- ▶ Let us fix some class of constraints.
- ▶ Let  $\Sigma$  be a set of such constraints.
- ▶ So, what do to with these?

- ▶ We have seen various examples of constraints.
  - ▶ **Key message:** keep them simple
  - ▶ Constraints allow to focus on **semantic properties** of data
  - ▶ **No low-level code** is needed to express dirtiness.
- ▶ Hope I convinced you that **constraints are an elegant way** of describing when data is clean or dirty!

### Next...

- ▶ Let us fix some class of constraints.
- ▶ Let  $\Sigma$  be a set of such constraints.
- ▶ So, what do to with these?

Declarative Approach

**Key Challenges**

Static Analysis

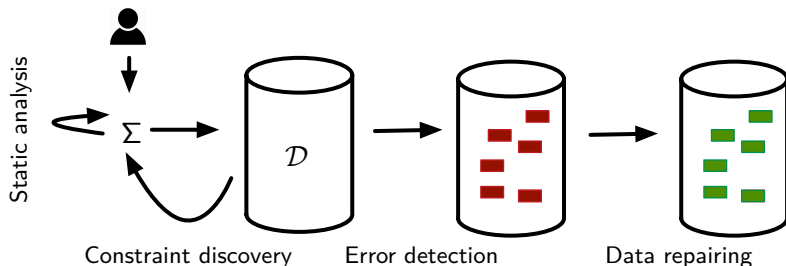
Error Detection

Constraint Discovery

Data Repairing

Conclusion

- ▶ A **high-level overview** of the overall **constraint-based data quality pipeline**:



- ▶ We discuss each of these aspects next.

Declarative Approach

Key Challenges

**Static Analysis**

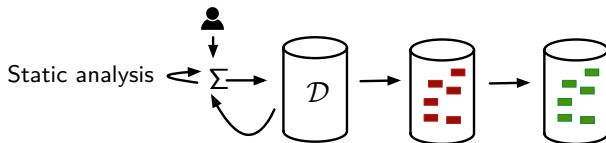
Error Detection

Constraint Discovery

Data Repairing

Conclusion





## Challenge:



When constraints are given, **check** constraints and **reduce** them in size without losing information.

When constraints  $\Sigma$  are present one may want to:

- ▶ Check whether they are **consistent**, i.e., whether there exists a database  $\mathcal{D}$  such that  $\mathcal{D} \models \varphi$ .
- ▶ This is a **sanity check** to ensure that the constraints are not dirty themselves.

### Inconsistent constraints

Consider

$$\varphi_1 : \forall s (R(s) \wedge s[A] = a \rightarrow s[B] = b_1)$$

$$\varphi_2 : \forall s (R(s) \wedge s[A] = a \rightarrow s[B] = b_2),$$

with  $b_1 \neq b_2$ . No database can satisfy both constraints.

- ▶ When  $\Sigma$  is inconsistent, find a **maximal consistent** set of constraints  $\Sigma' \subseteq \Sigma$ .
- ▶  $\Sigma \setminus \Sigma'$  are potentially “**wrong**” constraints.

When constraints  $\Sigma$  are present one may want to:

- ▶ Check whether they are **consistent**, i.e., whether there exists a database  $\mathcal{D}$  such that  $\mathcal{D} \models \varphi$ .
- ▶ This is a **sanity check** to ensure that the constraints are not dirty themselves.

### Inconsistent constraints

Consider

$$\varphi_1 : \forall s (R(s) \wedge s[A] = a \rightarrow s[B] = b_1)$$

$$\varphi_2 : \forall s (R(s) \wedge s[A] = a \rightarrow s[B] = b_2),$$

with  $b_1 \neq b_2$ . No database can satisfy both constraints.

- ▶ When  $\Sigma$  is inconsistent, find a **maximal consistent** set of constraints  $\Sigma' \subseteq \Sigma$ .
- ▶  $\Sigma \setminus \Sigma'$  are potentially “**wrong**” constraints.

When constraints  $\Sigma$  are present one may want to:

- ▶ Check whether they are **consistent**, i.e., whether there exists a database  $\mathcal{D}$  such that  $\mathcal{D} \models \varphi$ .
- ▶ This is a **sanity check** to ensure that the constraints are not dirty themselves.

### Inconsistent constraints

Consider

$$\varphi_1 : \forall s (R(s) \wedge s[A] = a \rightarrow s[B] = b_1)$$

$$\varphi_2 : \forall s (R(s) \wedge s[A] = a \rightarrow s[B] = b_2),$$

with  $b_1 \neq b_2$ . No database can satisfy both constraints.

- ▶ When  $\Sigma$  is inconsistent, find a **maximal consistent** set of constraints  $\Sigma' \subseteq \Sigma$ .
- ▶  $\Sigma \setminus \Sigma'$  are potentially “**wrong**” constraints.

When constraints  $\Sigma$  are present one may want to:

- ▶ Check whether no constraint in  $\Sigma$  is **redundant**, i.e., whether there exists a  $\varphi \in \Sigma$  such that for all  $\mathcal{D}$ ,

$$\mathcal{D} \models (\Sigma \setminus \varphi) \Rightarrow \mathcal{D} \models \varphi.$$

- ▶ **The smaller** the initial set of constraints, **the more efficient** things will be later on.

### Redundant constraints

Consider

$$\varphi_1 : \forall s, t (R(s) \wedge R(t) \wedge s[A] = t[A] \rightarrow s[B] = t[B])$$

$$\varphi_2 : \forall s, t (R(s) \wedge R(t) \wedge s[B] = t[B] \rightarrow s[C] = t[C])$$

$$\varphi_3 : \forall s, t (R(s) \wedge R(t) \wedge s[A] = t[A] \rightarrow s[C] = t[C]).$$

Any database  $\mathcal{D} \models \{\varphi_1, \varphi_2\}$  will also satisfy  $\varphi_3$ . So,  $\varphi_3$  is implied and can be regarded as redundant.

- ▶ When  $\Sigma$  is redundant, find a **minimal cover** of the set of constraints.
- ▶ That is, a minimal set of constraints equivalent to the original one.

When constraints  $\Sigma$  are present one may want to:

- ▶ Check whether no constraint in  $\Sigma$  is **redundant**, i.e., whether there exists a  $\varphi \in \Sigma$  such that for all  $\mathcal{D}$ ,

$$\mathcal{D} \models (\Sigma \setminus \varphi) \Rightarrow \mathcal{D} \models \varphi.$$

- ▶ **The smaller** the initial set of constraints, **the more efficient** things will be later on.

### Redundant constraints

Consider

$$\varphi_1 : \forall s, t (R(s) \wedge R(t) \wedge s[A] = t[A] \rightarrow s[B] = t[B])$$

$$\varphi_2 : \forall s, t (R(s) \wedge R(t) \wedge s[B] = t[B] \rightarrow s[C] = t[C])$$

$$\varphi_3 : \forall s, t (R(s) \wedge R(t) \wedge s[A] = t[A] \rightarrow s[C] = t[C]).$$

Any database  $\mathcal{D} \models \{\varphi_1, \varphi_2\}$  will also satisfy  $\varphi_3$ . So,  $\varphi_3$  is implied and can be regarded as redundant.

- ▶ When  $\Sigma$  is redundant, find a **minimal cover** of the set of constraints.
- ▶ That is, a minimal set of constraints equivalent to the original one.

When constraints  $\Sigma$  are present one may want to:

- ▶ Check whether no constraint in  $\Sigma$  is **redundant**, i.e., whether there exists a  $\varphi \in \Sigma$  such that for all  $\mathcal{D}$ ,

$$\mathcal{D} \models (\Sigma \setminus \varphi) \Rightarrow \mathcal{D} \models \varphi.$$

- ▶ **The smaller** the initial set of constraints, **the more efficient** things will be later on.

### Redundant constraints

Consider

$$\varphi_1 : \forall s, t (R(s) \wedge R(t) \wedge s[A] = t[A] \rightarrow s[B] = t[B])$$

$$\varphi_2 : \forall s, t (R(s) \wedge R(t) \wedge s[B] = t[B] \rightarrow s[C] = t[C])$$


$$\varphi_3 : \forall s, t (R(s) \wedge R(t) \wedge s[A] = t[A] \rightarrow s[C] = t[C]).$$

Any database  $\mathcal{D} \models \{\varphi_1, \varphi_2\}$  will also satisfy  $\varphi_3$ . So,  $\varphi_3$  is implied and can be regarded as redundant.

- ▶ When  $\Sigma$  is redundant, find a **minimal cover** of the set of constraints.
- ▶ That is, a minimal set of constraints equivalent to the original one.

When the constraints are **too expressive**, consistency and redundancy checking are **undecidable**.

- ▶ **Computational complexity** of these problems have been studied for **various small classes of constraints** (ptime, np, ...).
- ▶ Of particular interest are **axiomatizations of inference**, as this gives a **procedural way** for:
  - ▶ checking redundancy; and
  - ▶ computing **minimal covers**, i.e., a minimal set of equivalent constraints.

These are **important preprocessing steps** but we do not detail these further in this talk. See  Foundations of Data Quality Management, by Wenfei Fan, G., Morgan & Claypool, 2012 for more details.

- ▶  Andreas's lecture: chase-like procedure can be used for some of these tasks as well.



Declarative Approach

Key Challenges

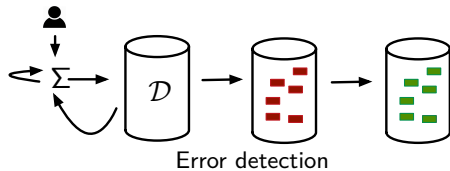
Static Analysis

**Error Detection**

Constraint Discovery

Data Repairing

Conclusion



## Challenge:



Given the constraints, **find the errors** in the data.

Recall:

### Definition (Errors of a constraint)

Let  $\varphi$  be a constraint and  $\mathcal{D}$  a database.

- ▶ A tuple  $s \in \mathcal{D}$  is an **error** for  $\varphi$ , if there exists a set  $S = \{s_1, \dots, s_k\}$  of tuples in  $\mathcal{D}$  such that  $S \cup \{s\} \not\models \varphi$  while  $S \models \varphi$ .
- ▶ The set **Error**( $\varphi, \mathcal{D}$ ) consists of all errors for  $\varphi$  in  $\mathcal{D}$ .

### Error detection problem

Given a database  $\mathcal{D}$  and a set  $\Sigma$  of constraints, **compute the errors**, i.e., the set of tuples

$$\mathbf{Error}(\Sigma, \mathcal{D}) = \bigcup_{\varphi \in \Sigma} \mathbf{Error}(\varphi, \mathcal{D}).$$

- ▶ For “standard” constraints (keys, foreign keys), supported by DBMS, simply execute:  
`DBCC CHECKCONSTRAINTS WITH ALL_CONSTRAINTS`
- ▶ For “non-standard” constraints, it is a matter of implementing them as **queries in SQL**.

Recall:

### Definition (Errors of a constraint)

Let  $\varphi$  be a constraint and  $\mathcal{D}$  a database.

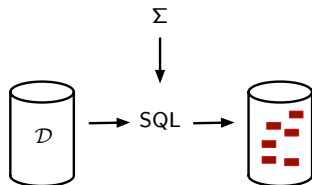
- ▶ A tuple  $s \in \mathcal{D}$  is an **error** for  $\varphi$ , if there exists a set  $S = \{s_1, \dots, s_k\}$  of tuples in  $\mathcal{D}$  such that  $S \cup \{s\} \not\models \varphi$  while  $S \models \varphi$ .
- ▶ The set **Error**( $\varphi, \mathcal{D}$ ) consists of all errors for  $\varphi$  in  $\mathcal{D}$ .

### Error detection problem

Given a database  $\mathcal{D}$  and a set  $\Sigma$  of constraints, **compute the errors**, i.e., the set of tuples

$$\mathbf{Error}(\Sigma, \mathcal{D}) = \bigcup_{\varphi \in \Sigma} \mathbf{Error}(\varphi, \mathcal{D}).$$

- ▶ For “standard” constraints (keys, foreign keys), supported by DBMS, simply execute:  
DBCC CHECKCONSTRAINTS WITH ALL\_CONSTRAINTS
- ▶ For “non-standard” constraints, it is a matter of implementing them as **queries in SQL**.



Positive side-effects of translation of constraints in SQL:

- ▶ Existing **query optimization techniques** can be used to efficiently detect errors of constraints.
- ▶ Can also be deployed in **incremental** (dynamic) and **distributed** setting.

Largely **taken for granted** that this can all be done efficiently.

- ▶ Unexplored how efficient error detection actually is for specific **classes of constraints**...
- ▶ E.g., Find FD errors in a distributed setting....

Declarative Approach

Key Challenges

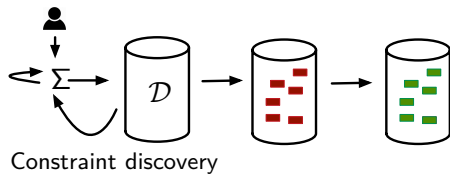
Static Analysis

Error Detection

**Constraint Discovery**

Data Repairing

Conclusion



A bigger problem:



A main issue is **how to get the constraints**.

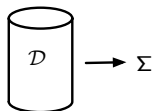
- ▶ They are either **manually** designed, by domain experts;
  - ▶ **Automatically discovered**, based on the data; or
  - ▶ **Learned** from errors, often in an **interactive way**.
- 
- ▶ Finding constraints is a part of **data profiling**, a more general process of obtaining information from the data.
    - 📖 Data Profiling, SIGMOD 2017 Tutorial by Ziawasch Abedjan, Lukasz Golab and Felix Naumann.
  - ▶ It is the **first step** in the constraint-based approach to data quality (once it is decided what kind of constraints will be considered).



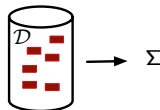
- ▶ They are either **manually** designed, by domain experts;
  - ▶ **Automatically discovered**, based on the data; or
  - ▶ **Learned** from errors, often in an **interactive way**.
- 
- ▶ Finding constraints is a part of **data profiling**, a more general process of obtaining information from the data.
    - 📖 Data Profiling, SIGMOD 2017 Tutorial by Ziawasch Abedjan, Lukasz Golab and Felix Naumann.
  - ▶ It is the **first step** in the constraint-based approach to data quality (once it is decided what kind of constraints will be considered).

Two main approaches:

1. **Data-driven discovery methods:** Find constraints by using only the **data**.



2. **Explanation-based methods:** Find constraints using the **data** and available **errors**.  
 $\Rightarrow$  Constraints should **explain** the errors.



Automatic methods are typically designed for a **specific class** of constraints.

Let  $C$  be a **class** of constraints, e.g., functional dependencies, denial constraints, inclusion dependencies, matching dependencies, ...

### Constraint discovery problem

Given a database  $\mathcal{D}$  and a class  $C$  of constraints, **find all constraints**  $\varphi$  in  $C$  such that  $\mathcal{D} \models \varphi$  holds.

### Does this make sense?

- ▶ You will discover constraints based on dirty data??
- ▶ You will use these constraints to find errors in the data, but  $\mathcal{D} \models \varphi$  implies that there are no errors??

Automatic methods are typically designed for a **specific class** of constraints.

Let  $C$  be a **class** of constraints, e.g., functional dependencies, denial constraints, inclusion dependencies, matching dependencies, ...

### Constraint discovery problem

Given a database  $\mathcal{D}$  and a class  $C$  of constraints, **find all constraints**  $\varphi$  in  $C$  such that  $\mathcal{D} \models \varphi$  holds.

### Does this make sense?

- ▶ You will discover constraints based on dirty data??
- ▶ You will use these constraints to find errors in the data, but  $\mathcal{D} \models \varphi$  implies that there are no errors??

Automatic methods are typically designed for a **specific class** of constraints.

Let  $C$  be a **class** of constraints, e.g., functional dependencies, denial constraints, inclusion dependencies, matching dependencies, ...

### Constraint discovery problem

Given a database  $\mathcal{D}$  and a class  $C$  of constraints, **find all constraints**  $\varphi$  in  $C$  such that  $\mathcal{D} \models \varphi$  holds.

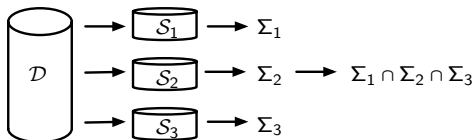
### Does this make sense?

- ▶ You will discover constraints based on dirty data??
- ▶ You will use these constraints to find errors in the data, but  $\mathcal{D} \models \varphi$  implies that there are no errors??

How to **reduce the impact** of dirty data in the discovery process?

Common approaches:

1. **Sampling**: Run the discovery process on **different samples** from the data, **only retain** constraints discovered in **each sample** (intersection).



2. **Approximate**: Find constraints that are “almost” true. For example, find all  $\varphi$  in the class  $\mathcal{C}$  of constraints such that

$$(\mathcal{D} \setminus \mathcal{E}) \models \varphi$$

for a small set  $\mathcal{E}$  of tuples, e.g.,  $|\mathcal{E}| < \epsilon |\mathcal{D}|$  for some parameter  $\epsilon > 0$ .

3. **Hybrid**: Combinations of **sampling** and **approximate** methods.

- ▶ There are over **60 methods** each discovering some specific kind of constraints.
- ▶ A **renewed interest** in recent years, with papers on this topic in all major database and data mining conferences.

We briefly discuss:

- ▶ a **schema-driven** functional dependency discovery algorithm (TANE)
  - ▶ a **data-driven** denial constraints discovery algorithm (FASTDC)
- 
- ▶ Most other methods use similar techniques.
  - ▶ Overlap with **data mining techniques** for finding patterns in data.

Declarative Approach

Key Challenges

Static Analysis

Error Detection

Constraint Discovery

- FD discovery

- Approximate FD discovery

- DC Discovery

- Explanation-Based

- Conclusion

Data Repairing

Conclusion



A **classical problem**: A lot of different methods exist for FD discovery (earliest algorithms 1999, latest 2019).

For convenience, we denote functional dependencies

$$\varphi = \forall s, t (R(s) \wedge R(t) \wedge s[X] = t[X] \rightarrow s[Y] = t[Y])$$

by  $X \rightarrow Y$  and write  $\mathcal{D} \models X \rightarrow Y$  for  $\mathcal{D} \models \varphi$ .

## FD discovery problem

Given a database  $\mathcal{D}$ , **find all FDs**  $X \rightarrow Y$  such that  $\mathcal{D} \models X \rightarrow Y$ .

It suffices to consider FDs of the form  $X \rightarrow A$ , for a single right-hand-side (RHS) attribute  $A$ .

## Naive Algorithm

```
1 Function: find_FDs ( $\mathcal{D}$ )  
2 return All valid FDs  $\varphi$  such that  $\mathcal{D} \models \varphi$ .  
3 for each attribute  $A$  in  $R$  do  
4   for each  $X \subseteq R \setminus \{A\}$  do  
5     for each pair  $(t_1, t_2) \in \mathcal{D}$  do  
6       if  $t_1[X] = t_2[X]$  &  $t_1[A] \neq t_2[A]$  then  
7         break  
8   return  $X \rightarrow A$ 
```

Complexity: For each of the  $|R|$  possibilities for RHS:

- ▶ check  $2^{|R|-1}$  combinations for left-hand-sides
- ▶ scan the db  $|\mathcal{D}|^2/2$  times for each combination.

Don't use this algorithm! Very inefficient!

- ▶ Avoid discovering redundant FDs.
- ▶ A better way of traversing the search space is needed.
- ▶ Improve efficiency of checking validity of FDs.

## Naive Algorithm

```
1 Function: find_FDs ( $\mathcal{D}$ )  
2 return All valid FDs  $\varphi$  such that  $\mathcal{D} \models \varphi$ .  
3 for each attribute  $A$  in  $R$  do  
4   for each  $X \subseteq R \setminus \{A\}$  do  
5     for each pair  $(t_1, t_2) \in \mathcal{D}$  do  
6       if  $t_1[X] = t_2[X]$  &  $t_1[A] \neq t_2[A]$  then  
7         break  
8   return  $X \rightarrow A$ 
```

Complexity: For each of the  $|R|$  possibilities for RHS:

- ▶ check  $2^{|R|-1}$  combinations for left-hand-sides
- ▶ scan the db  $|\mathcal{D}|^2/2$  times for each combination.

Don't use this algorithm! Very inefficient!

- ▶ Avoid discovering redundant FDs.
- ▶ A better way of traversing the search space is needed.
- ▶ Improve efficiency of checking validity of FDs.

We look in a bit more detail to the **TANE** algorithm.

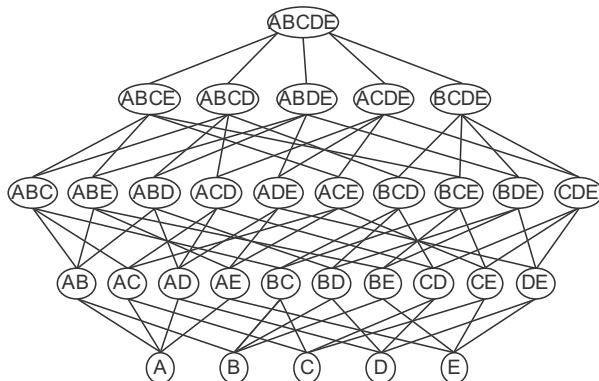
📖 TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. The Computer Journal 42 (2), 1999, by Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen

Idea behind the approach:

1. **Reduce column combinations** through pruning.
  - ▶ Modelling of search space as lattice; and
  - ▶ Reasoning over FDs.
2. **Reduce tuple sets** through partitioning.
  - ▶ Partition data according to attribute values; and
  - ▶ Level-wise increase of size of attribute set.

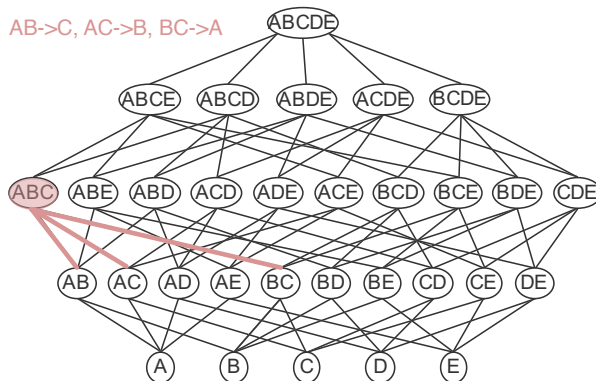
- ▶ Model search space as **power set lattice**: brings some structure to the search space.
- ▶ Traverse the lattice in a **bottom-up way**: use information from levels below.

Suppose  $R$  is a relation with attributes  $A, B, C, D, E$ .



- ▶ Model search space as **power set lattice**: brings some structure to the search space.
- ▶ Traverse the lattice in a **bottom-up way**: use information from levels below.
- ▶ A node  $X$  in the lattice corresponds to FDs:  $X \setminus A \rightarrow A$  for  $A \in X$ .

Suppose  $R$  is a relation with attributes  $A, B, C, D, E$ .



### Definition (Minimal FDs)

A **minimal FD**  $X \rightarrow A$  is one such that  $\mathcal{D} \models X \rightarrow A$  but  $\mathcal{D} \not\models Y \rightarrow A$  for all **strict** subsets  $Y$  of  $X$

### Non-minimal FD

Let  $X = \{ABC\}$ .

- ▶ **Naive:** Need to test three dependencies:

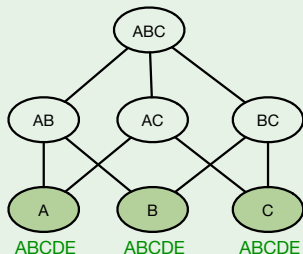
$$AB \rightarrow C, AC \rightarrow B, \text{ and } BC \rightarrow A.$$

- ▶ **Optimization:** Suppose we know from before that  $\mathcal{D} \models A \rightarrow B$ .
  - ▶ We should not be testing  $\mathcal{D} \models AC \rightarrow B$  because it is **not minimal**.

- ▶ TANE maintains a set of **candidate RHS attributes** during lattice traversal, to avoid checking non-minimal FDs.

### Candidate RHS

- Initially, all attributes are candidate RHS (marked in green).

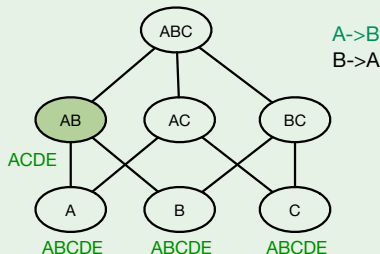






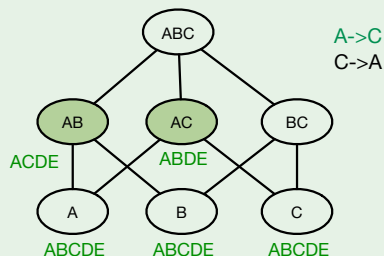
## Candidate RHS

- ▶ Initially, all attributes are candidate RHS (marked in green).
- ▶ FDs are verified on  $\mathcal{D}$ : green ones are valid FDs.
- ▶ Valid RHS attributes are removed from candidate set.



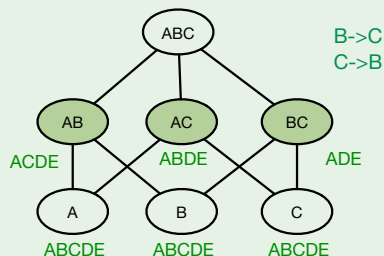
## Candidate RHS

- ▶ Initially, all attributes are candidate RHS (marked in green).
- ▶ FDs are verified on  $\mathcal{D}$ : green ones are valid FDs.
- ▶ Valid RHS attributes are removed from candidate set.



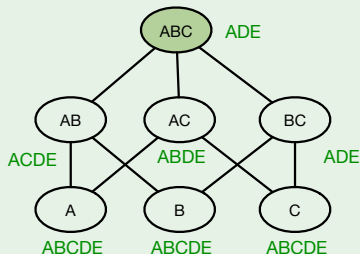
## Candidate RHS

- ▶ Initially, all attributes are candidate RHS (marked in green).
- ▶ FDs are verified on  $\mathcal{D}$ : green ones are valid FDs.
- ▶ Valid RHS attributes are removed from candidate set.



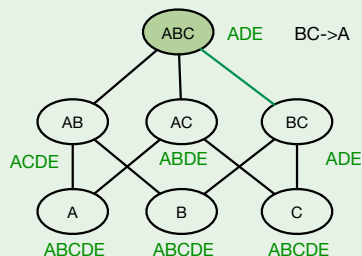
### Candidate RHS

- ▶ Initially, all attributes are candidate RHS (marked in green).
- ▶ FDs are verified on  $\mathcal{D}$ : green ones are valid FDs.
- ▶ Valid RHS attributes are removed from candidate set.
- ▶ Candidate set next level: **intersection** of candidate sets below.

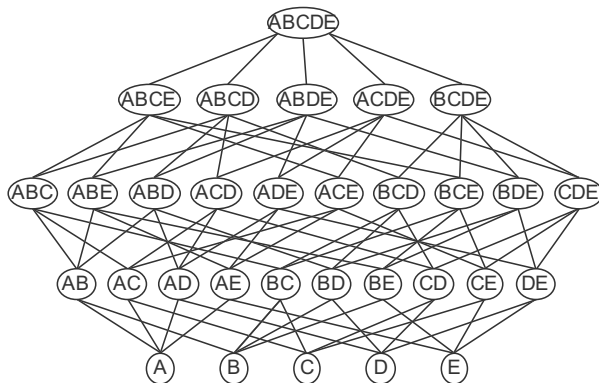


## Candidate RHS

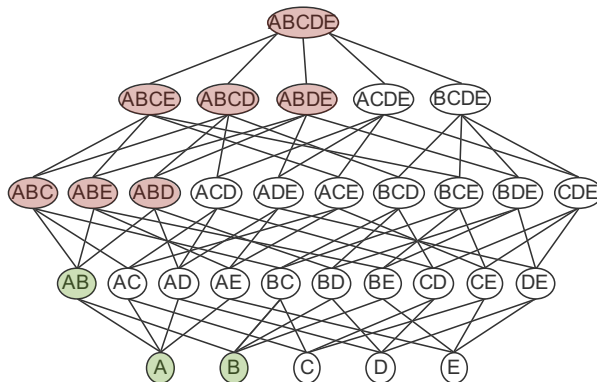
- ▶ Initially, all attributes are candidate RHS (marked in green).
- ▶ FDs are verified on  $\mathcal{D}$ : green ones are valid FDs.
- ▶ Valid RHS attributes are removed from candidate set.
- ▶ Candidate set next level: **intersection** of candidate sets below.
- ▶ Only candidate attributes tested for RHS.



- ▶ TANE uses **more advanced reasoning on FDs**  $\Rightarrow$  further reduction of candidate RHS sets.
- ▶ When a node  $X$  has no candidate RHS, then **all supersets**  $Y$  of  $X$  **can be pruned**.
  - ▶ No  $Y \setminus \{A\} \rightarrow A$  can be minimal and  $Y$  can be ignored.



- ▶ TANE uses **more advanced reasoning on FDs**  $\Rightarrow$  further reduction of candidate RHS sets.
- ▶ When a node  $X$  has no candidate RHS, then **all supersets**  $Y$  of  $X$  **can be pruned**.
  - ▶ No  $Y \setminus \{A\} \rightarrow A$  can be minimal and  $Y$  can be ignored.





## TANE

```
1 Function: tane( $\mathcal{D}$ )
2 return All valid minimal FDs  $\varphi$  such that  $\mathcal{D} \models \varphi$ .

3 level0 :=  $\emptyset$ 
4 candRHS := all attributes in  $R$ 
5 level1 :=  $\{A \mid A \in R\}$ 
6  $\ell = 1$ 
7 while level $\ell$   $\neq \emptyset$  do
8   VerifyDependencies&UpdateCandRHS(level $\ell$ , candRHS)
9   prune(level $\ell$ , candRHS)
10  level $\ell+1$  := generate_next_level(level $\ell$ )
11   $\ell := \ell + 1$ 
```

- ▶ Dependency checking is **optimised** in TANE as well, based on a **partitioning approach**.
- ▶ This partitioning approach is used in **many other discovery methods** as well.

Let  $X$  be a set of attributes.

### Definition ( $X$ -equivalence)

Tuples  $s$  and  $t$  in  $\mathcal{D}$  are  **$X$ -equivalent** w.r.t. attribute set  $X$  if  $t[A] = s[A]$  for all  $A \in X$ .

### Definition ( $X$ -Partition)

Attribute set  $X$  **partitions**  $\mathcal{D}$  into **equivalence classes**:

$$[t]_X = \{s \in \mathcal{D} \mid \forall A \in X, s[A] = t[A]\}.$$

Clearly,

$$\mathcal{D} = [t_1]_X \dot{\cup} [t_2]_X \dot{\cup} \dots \dot{\cup} [t_k]_X.$$

for some tuples  $t_1, \dots, t_k$  in  $\mathcal{D}$ . We denote the set of **parts** by  $\pi_X$ .

## Partition

tuple id	A	B	C	D
1	$a_1$	$b_1$	$c_1$	$d_1$
2	$a_1$	$b_2$	$c_2$	$d_3$
3	$a_2$	$b_2$	$c_1$	$d_4$
4	$a_2$	$b_2$	$c_1$	$d_1$
5	$a_2$	$b_3$	$c_3$	$d_5$
6	$a_3$	$b_3$	$c_1$	$d_6$
7	$a_3$	$b_4$	$c_4$	$d_1$
8	$a_3$	$b_4$	$c_5$	$d_7$

$$[1]_A = [2]_A = \{1, 2\}$$

$$\pi_A = \{\{1, 2\}, \{3, 4, 5\}, \{6, 7, 8\}\}$$

$$\pi_{BC} = \{\{1\}, \{2\}, \{3, 4\}, \{5\}, \{6\}, \{7\}, \{8\}\}$$

$$\pi_D = \{\{1, 4, 7\}, \{2\}, \{3\}, \{5\}, \{6\}, \{8\}\}$$

$$\vdots$$

### Definition (Refining)

A partition  $\pi$  **refines partition**  $\pi'$  if every equivalence class in  $\pi$  is a subset of some equivalence class in  $\pi'$ .

Let  $X$  be a set of attributes.

- ▶  $\pi_{XA}$  always refines  $\pi_A$ .
- ▶ If  $\pi_X$  also refines  $\pi_{XA}$ ,  $\pi_X = \pi_{XA}$  and  $\mathcal{D} \models X \rightarrow A$ .

- ▶ Validity checking of FDs can use the partitions.
- ▶ Furthermore, we simply need the **sizes** of the partitions.

We have that  $\mathcal{D} \models X \rightarrow A$  if and only if  $|\pi_X| = |\pi_{XA}|$ .

### Definition (Refining)

A partition  $\pi$  **refines partition**  $\pi'$  if every equivalence class in  $\pi$  is a subset of some equivalence class in  $\pi'$ .

Let  $X$  be a set of attributes.

- ▶  $\pi_{XA}$  always refines  $\pi_A$ .
- ▶ If  $\pi_X$  also refines  $\pi_{XA}$ ,  $\pi_X = \pi_{XA}$  and  $\mathcal{D} \models X \rightarrow A$ .

- ▶ Validity checking of FDs can use the partitions.
- ▶ Furthermore, we simply need the **sizes** of the partitions.

We have that  $\mathcal{D} \models X \rightarrow A$  if and only if  $|\pi_X| = |\pi_{XA}|$ .

## Testing validity of FDs:

We have that  $\mathcal{D} \models X \rightarrow A$  if and only if  $|\pi_X| = |\pi_{XA}|$ .

## Example

tuple id	A	B
1	$a_1$	$b_1$
2	$a_1$	$b_1$
3	$a_2$	$b_1$
4	$a_2$	$b_1$
5	$a_2$	$b_1$
6	$a_3$	$b_2$
7	$a_3$	$b_2$
8	$a_3$	$b_2$

$$\pi_A = \{\{1, 2\}, \{3, 4, 5\}, \{6, 7, 8\},$$

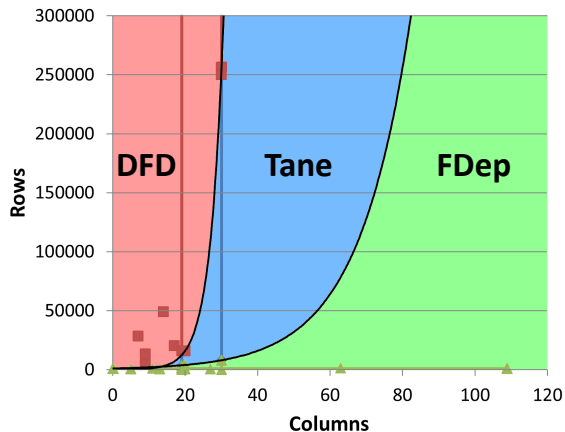
$$\pi_B = \{\{1, 2, 3, 4, 5\}, \{6, 7, 8\}\}$$


$$\pi_{AB} = \{\{1, 2\}, \{3, 4, 5\}, \{6, 7, 8\}\}$$

Hence,  $|\pi_{AB}| = |\pi_A|$  and  $A \rightarrow B$ . Note,  $|\pi_{AB}| > |\pi_B|$  and  $B \rightarrow A$  does not hold.

► These partitions can be generated **efficiently during lattice traversal**.

- ▶ **Schema-driven:** Usually sensitive to the size of the schema  $\Rightarrow$  Good for long thin tables!
  - ▶ TANE [📖 Huhtala et al, 1999]
  - ▶ FUN [📖 Novelli et al., 2001]
  - ▶ FDMine [📖 Yao et al., 2002]
  - ▶ DepMiner [📖 Lopez et al., 2000]
- ▶ **Instance-driven:** Usually sensitive to the size of the data  $\Rightarrow$  Good for fat short tables!
  - ▶ FASTFD [📖 Wyss et al, DaWaK, 2001]
- ▶ **Hybrid:** Try to get the best of both worlds.
  - ▶ FDEP [📖 Flach et al., 1999]
  - ▶ DFD [📖 Abedjan et al. 2015]
  - ▶ PYRO [📖 Kruse et al. 2018]
  - ▶ HyFD [📖 Papenbrock et al. 2016]
- ▶ **Other aspects:**
  - ▶ Distributed FD Discovery [📖 Saxena et al., VLDB 2019]
  - ▶ ...



- ▶  Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms, Paperbrock et al, VLDB 2016
- ▶ <https://hpi.de/naumann/projects/repeatability/data-profiling/fds.html>



Declarative Approach

Key Challenges

Static Analysis

Error Detection

**Constraint Discovery**

- FD discovery

- Approximate FD discovery**

- DC Discovery

- Explanation-Based

- Conclusion

Data Repairing

Conclusion

TANE can also be used to discover **approximate functional dependencies**

An **approximate FD**  $X \rightarrow A$  holds on  $\mathcal{D}$  if

$$\text{err}(X \rightarrow A, \mathcal{D}) \leq \varepsilon,$$

where

$$\text{err}(X \rightarrow A, \mathcal{D}) = \frac{\min\{|S| \mid S \subseteq \mathcal{D}, \mathcal{D} \setminus S \models X \rightarrow A\}}{|\mathcal{D}|},$$

i.e., **minimum number of tuples to be removed from  $\mathcal{D}$  such that  $X \rightarrow A$  holds.**

- Using the partitions,  $\text{err}(X \rightarrow A, \mathcal{D})$  can be easily computed.

Declarative Approach

Key Challenges

Static Analysis

Error Detection

**Constraint Discovery**

- FD discovery

- Approximate FD discovery

- DC Discovery**


- Explanation-Based

- Conclusion

Data Repairing

Conclusion

We next consider the FASTDC Algorithm finds **all minimal valid DCs**.

 Discovering Denial Constraints, VLDB 2013, by Xu Chu, Ihab F. Ilyas and Paolo Papotti.

By contrast to TANE, this is an **instance-based** algorithm.

1. **Build evidence sets**, based on the data.
2. **Return minimal covers** of these sets.

### Denial constraint

$$\neg \exists s, t (R(s) \wedge R(t) \wedge s[AC] = t[AC] \wedge s[SAL] < t[SAL] \wedge s[TR] > t[TR])$$

*“Two people living in the same state should have correct tax rates depending on their income”*

We here only consider denial constraints involving **two tuples**.

In a denial constraint, we may have different kinds of **predicates**:

$$s[A] = t[A] \quad s[A] \neq t[A] \quad s[A] < t[A] \quad \dots$$

The **predicate space** consists of all (valid/invalid) instantiations of these predicates.

## Predicate space

tuple id	A	B	C
1	$a_1$	$a_1$	50
2	$a_2$	$a_1$	40
3	$a_3$	$a_1$	40

Space of predicates  $\mathcal{P}$ : for  $i, j \in \{1, 2, 3\}$

$$P_1 : t_i.A = t_j.A \quad P_2 : t_i.A \neq t_j.A$$

$$P_3 : t_i.B = t_j.B \quad P_4 : t_i.B \neq t_j.B$$

$$P_{11} : t_i.A = t_i.B \quad P_{12} : t_i.A \neq t_i.B$$

$$P_{21} : t_i.A = t_j.B \quad P_{22} : t_i.A \neq t_j.B$$

$$P_5 : t_i.C = t_j.C \quad P_6 : t_i.C \neq t_j.C$$

$$P_6 : t_i.C > t_j.C \quad P_8 : t_i.C \geq t_j.C$$

$$P_9 : t_i.C < t_j.C \quad P_{10} : t_i.A \leq t_i.B$$

- ▶ Any combination of these predicates may be a valid DC (exponentially many!)
- ▶ Only valid DCs with a minimal number of predicates are discovered.

Based on the database instance  $\mathcal{D}$  and predicate space  $\mathcal{P}$ , the **evidence sets** are computed:

- For each pair of tuples, list all **valid predicates**.

## Evidence sets

tuple id	A	B	C
1	$a_1$	$a_1$	50
2	$a_2$	$a_1$	40
3	$a_3$	$a_1$	40

Evidence sets  $\mathcal{E}$ :

$$(2, 3), (3, 2) = \{P_2, P_3, P_5, P_8, P_{10}, P_{12}, P_{14}\}$$

$$(2, 1), (3, 1) = \{P_2, P_3, P_6, P_8, P_9, P_{12}, P_{14}\}$$

$$(1, 2), (1, 3) = \{P_2, P_3, P_6, P_7, P_{10}, P_{11}, P_{13}\}$$

- A **minimal cover** of an evidence set is a minimal set of predicates **intersecting all evidence sets**.

$$\mathcal{D} \models \neg(P_i \wedge P_j \wedge P_k)$$



For every pair of tuples in  $\mathcal{D}$ ,  $P_i$ ,  $P_j$  and  $P_k$  cannot be all true



For every pair of tuples in  $\mathcal{D}$ , at least one of  $P_i$ ,  $P_j$  and  $P_k$  is false



For every pair of tuples in  $\mathcal{D}$ , at least one of  $\neg P_i$ ,  $\neg P_j$  and  $\neg P_k$  is true



$\neg P_i$ ,  $\neg P_j$  and  $\neg P_k$  covers the **evidence sets**

### Theorem

$\neg(P_1 \wedge \dots \wedge P_k)$  is a minimal valid DC if and only if  $\{\neg P_1, \dots, \neg P_k\}$  is a **minimal set cover** for all evidence sets.

## Minimal Denial constraints

Evidence sets  $\mathcal{E}$ :

tuple id	A	B	C
1	$a_1$	$a_1$	50
2	$a_2$	$a_1$	40
3	$a_3$	$a_1$	40

$$(2, 3), (3, 2) = \{P_2, P_3, P_5, P_8, P_{10}, P_{12}, P_{14}\}$$

$$(2, 1), (3, 1) = \{P_2, P_3, P_6, P_8, P_9, P_{12}, P_{14}\}$$

$$(1, 2), (1, 3) = \{P_2, P_3, P_6, P_7, P_{10}, P_{11}, P_{13}\}$$

$\Rightarrow P_2$  covers the set of true predicates minimally.

Hence,  $\neg(\neg P_2) = P_2$  is a valid minimal DC.

$\Rightarrow P_{10}, P_{14}$  cover the set of true predicates minimally.

Hence,  $\neg(\neg P_{10} \wedge \neg P_{14})$  is a valid minimal DC

```

1 Function: FastDC ( $\mathcal{D}$ )
2 return Set  $\Sigma$  of all valid denial constraints on  $\mathcal{D}$ .

3  $P \leftarrow$  build the predicate space for  $\mathcal{D}$ 
4  $\mathcal{E} \leftarrow$  build the evidence sets based on  $P$  and  $\mathcal{D}$ 
5 for minimal cover  $C$  of  $E$  do
6    $\Sigma := \Sigma \cup \{\neg \bar{C}\}$ 

```



## Minimal Denial constraints

tuple id	A	B	C
1	$a_1$	$a_1$	50
2	$a_2$	$a_1$	40
3	$a_3$	$a_1$	40

Evidence sets  $\mathcal{E}$ :

$$(2, 3), (3, 2) = \{P_2, P_3, P_5, P_8, P_{10}, P_{12}, P_{14}\}$$

$$(2, 1), (3, 1) = \{P_2, P_3, P_6, P_8, P_9, P_{12}, P_{14}\}$$

$$(1, 2), (1, 3) = \{P_2, P_3, P_6, P_7, P_{10}, P_{11}, P_{13}\}$$

$\Rightarrow P_2$  covers the set of true predicates minimally.

Hence,  $\neg(\neg P_2) = P_2$  is a valid minimal DC.

$\Rightarrow P_{10}, P_{14}$  cover the set of true predicates minimally.

Hence,  $\neg(\neg P_{10} \wedge \neg P_{14})$  is a valid minimal DC

```

1 Function: FastDC ( $\mathcal{D}$ )
2 return Set  $\Sigma$  of all valid denial constraints on  $\mathcal{D}$ .


---


3  $P \leftarrow$  build the predicate space for  $\mathcal{D}$ 
4  $\mathcal{E} \leftarrow$  build the evidence sets based on  $P$  and  $\mathcal{D}$ 
5 for minimal cover  $C$  of  $E$  do
6    $\Sigma := \Sigma \cup \{\neg \bar{C}\}$ 

```

## The actual FASTDC algorithm

📖 Discovering Denial Constraints, VLDB 2013, by Xu Chu, Ihab F. Ilyas and Paolo Papotti.

has:











- ▶ More advanced pruning techniques, based on **axiomatization** of implication of DCs;
- ▶ Different methods for **finding all minimal covers**; and
- ▶ Extended to **approximate DCs**.

Extended with **sampling** techniques and more efficient **evaluation strategies** in the HYDRA algorithm:

📖 Efficient Denial Constraint Discovery with Hydra, VLDB 2017, Tobias Bleifuß, Sebastian Kruse and Felix Naumann

## Other Discovery Methods (very partial list)

---

- CFDs**  Discovering Conditional Functional Dependencies, W. Fan, G., L. Jianzhong, M. Xiong, TKDE, 2010.
- CFDs**  Discovering Data Quality Rules, F. Chiang, R. Miller, VLDB, 2008.
- CFDs**  Estimating the confidence of conditional functional dependencies, G. Cormode, L. Golab, F. Korn, A. McGregor, D. Srivastava, X. Zhang, SIGMOD 2009.
- DDs**  Differential dependencies: Reasoning and discovery, S. Song, L. Chen, TODS, 2011
- INDs**  Unary and n-ary inclusion dependency discovery in relational databases. F. De Marchi, S. Lopes, and J.-M. Petit., JIIS 2009.
- INDS**  Divide & conquer-based inclusion dependency discovery. T. Papenbrock, S. Kruse, J.-A. Quianè-Ruiz, and F. Naumann. VLDB, 2015.
- CINDs**  Discovering conditional inclusion dependencies, J. Bauckmann Z. Abedjan, U. Leser, H. Müller, F. Naumann, CIKM 2012.
- eRs**  Discovering editing rules for data cleaning. T. Diallo, J.-M. Petit, and S. Servigne. AQB, 2012.
- MDs**  Discovering matching dependencies, S. Song and L. Chen. CIKM, 2009.
- ODs**  Efficient distributed discovery of bidirectional order dependencies S Schmidl, T Papenbrock. The VLDB Journal, 2022.

Declarative Approach

Key Challenges

Static Analysis

Error Detection

**Constraint Discovery**

- FD discovery

- Approximate FD discovery

- DC Discovery

- Explanation-Based**

- Conclusion

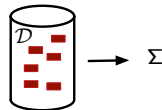
Data Repairing

Conclusion

So far, discovery methods **only used the database**  $\mathcal{D}$ .

In practice, we may have additional information:

- ▶ errors,
- ▶ corrections, or
- ▶ user feedback.



- ▶ **Explanation-based methods** attempt to find constraints that “**explain**” these errors and corrections.

## ► FALCON

👉 Interactive and deterministic data cleaning: A tossed stone raises a thousand ripples. SIGMOD 2016, J. He et al.

- It finds constant CFDs based on a single modification.
- relies on a “user oracle” to (in)validate the proposed rules.

## ► UGUIDE

👉 UGuide: User-guided discovery of FD-detectable errors. SIGMOD 2017, L. Thirumuruganathan et al.

- Starting from an initial set of FDs, find FD that explain errors by means of questions to users.
- Cost-function: Explain as many errors as possible, minimize false positives.

## ► Xplode

👉 Explaining Repaired Data with CFDs VLDB 2018, J. Rammelaere and G.

- We discuss this in a bit more detail.

## ► Crowd-based methods... (not covered)

## ► FALCON

👉 Interactive and deterministic data cleaning: A tossed stone raises a thousand ripples. SIGMOD 2016, J. He et al.

- It finds constant CFDs based on a single modification.
- relies on a “user oracle” to (in)validate the proposed rules.

## ► UGUIDE

👉 UGuide: User-guided discovery of FD-detectable errors. SIGMOD 2017, L. Thirumuruganathan et al.

- Starting from an initial set of FDs, find FD that explain errors by means of questions to users.
- Cost-function: Explain as many errors as possible, minimize false positives.

## ► Xplode

👉 Explaining Repaired Data with CFDs VLDB 2018, J. Rammelaere and G.

- We discuss this in a bit more detail.

## ► Crowd-based methods... (not covered)

## ► FALCON

👉 Interactive and deterministic data cleaning: A tossed stone raises a thousand ripples. SIGMOD 2016, J. He et al.

- It finds constant CFDs based on a single modification.
- relies on a “user oracle” to (in)validate the proposed rules.

## ► UGUIDE

👉 UGuide: User-guided discovery of FD-detectable errors. SIGMOD 2017, L. Thirumuruganathan et al.

- Starting from an initial set of FDs, find FD that explain errors by means of questions to users.
- Cost-function: Explain as many errors as possible, minimize false positives.

## ► Xplode

👉 Explaining Repaired Data with CFDs VLDB 2018, J. Rammelaere and G.

- We discuss this in a bit more detail.

## ► **Crowd-based** methods... (not covered)



Given: Dirty database with **errors** and **corrections**.

- We regard (corrections) as **modifications** changing a dirty instance to a (partially) clean one:

$$D_{dirty} \mapsto \text{modifications} \mapsto D_{rep}$$

TID	CC	AC	PN	NM	STR	CT	ZIP
$t_1$	01	908	1111111	Mike	Tree Ave.	MH←LA	07974
$t_2$	01	908	1111111	Rick	Tree Ave.	MH←NYC	07974
$t_3$	01	212	2222222	Joe	5th Ave	NYC	01202
$t_4$	01	908	2222222	Jim	Elm Str.	MH	07974
$t_5$	44	131	3333333	Ben	High St.	EDI	EH4 1DT
$t_6$	44	131	4444444	Ian	High St.	EDI	EH4 1DT
$t_7$	44	908	4444444	Ian	Port Pl	MH	W1B 1JH
$t_8$	01←44	131	2222222	Sean	3rd Str.	UN	01202

### Definition (Explanation)

Let  $M$  be a set of modifications. Then a CFD  $\varphi$  of the form

$$\forall s, t \left( R(s) \wedge R(t) \wedge \bigwedge_{i \in L} t[A] = s[A] = a_i \rightarrow s[B] = t[B] (= b) \right)$$

is an  **$M$ -explanation** if:

- ▶ The **error** of the CFD **decreases** after applying the modifications.  
 $\Rightarrow$  the partial repair satisfies the CFD more than the original database.
- ▶ **At least one tuple** that is modified should be an **error** of the CFD.  
 $\Rightarrow$  CFD should not explain irrelevant modifications.
- ▶ When restricted to the **modified tuples**, the CFD should be **satisfied**.  
 $\Rightarrow$  This ensures that future modifications keep the CFD satisfied.

It is a **local  $M$ -explanation** if it is an  $M'$ -explanation for any  $M' \subseteq M$ .

- ▶ Should be independent of the order of the modifications.

### Definition (Explanation)

Let  $M$  be a set of modifications. Then a CFD  $\varphi$  of the form

$$\forall s, t (R(s) \wedge R(t) \wedge \bigwedge_{i \in L} t[A] = s[A] = a_i \rightarrow s[B] = t[B] (= b))$$

is an  **$M$ -explanation** if:

- ▶ The **error** of the CFD **decreases** after applying the modifications.  
 $\Rightarrow$  the partial repair satisfies the CFD more than the original database.
- ▶ **At least one tuple** that is modified should be an **error** of the CFD.  
 $\Rightarrow$  CFD should not explain irrelevant modifications.
- ▶ When restricted to the **modified tuples**, the CFD should be **satisfied**.  
 $\Rightarrow$  This ensures that future modifications keep the CFD satisfied.

It is a **local  $M$ -explanation** if it is an  $M'$ -explanation for any  $M' \subseteq M$ .

- ▶ Should be independent of the order of the modifications.

### Definition (Explanation)

Let  $M$  be a set of modifications. Then a CFD  $\varphi$  of the form

$$\forall s, t (R(s) \wedge R(t) \wedge \bigwedge_{i \in L} t[A] = s[A] = a_i \rightarrow s[B] = t[B] (= b))$$

is an  **$M$ -explanation** if:

- ▶ The **error** of the CFD **decreases** after applying the modifications.  
 $\Rightarrow$  the partial repair satisfies the CFD more than the original database.
- ▶ **At least one tuple** that is modified should be an **error** of the CFD.  
 $\Rightarrow$  CFD should not explain irrelevant modifications.
- ▶ When restricted to the **modified tuples**, the CFD should be **satisfied**.  
 $\Rightarrow$  This ensures that future modifications keep the CFD satisfied.

It is a **local  $M$ -explanation** if it is an  $M'$ -explanation for any  $M' \subseteq M$ .

- ▶ Should be independent of the order of the modifications.

### Definition (Explanation)

Let  $M$  be a set of modifications. Then a CFD  $\varphi$  of the form

$$\forall s, t (R(s) \wedge R(t) \wedge \bigwedge_{i \in L} t[A] = s[A] = a_i \rightarrow s[B] = t[B] (= b))$$

is an  **$M$ -explanation** if:

- ▶ The **error** of the CFD **decreases** after applying the modifications.  
 $\Rightarrow$  the partial repair satisfies the CFD more than the original database.
- ▶ **At least one tuple** that is modified should be an **error** of the CFD.  
 $\Rightarrow$  CFD should not explain irrelevant modifications.
- ▶ When restricted to the **modified tuples**, the CFD should be **satisfied**.  
 $\Rightarrow$  This ensures that future modifications keep the CFD satisfied.

It is a **local  $M$ -explanation** if it is an  $M'$ -explanation for any  $M' \subseteq M$ .

- ▶ Should be independent of the order of the modifications.

### Definition (Explanation)

Let  $M$  be a set of modifications. Then a CFD  $\varphi$  of the form

$$\forall s, t \left( R(s) \wedge R(t) \wedge \bigwedge_{i \in L} t[A] = s[A] = a_i \rightarrow s[B] = t[B] (= b) \right)$$

is an  **$M$ -explanation** if:

- ▶ The **error** of the CFD **decreases** after applying the modifications.  
 $\Rightarrow$  the partial repair satisfies the CFD more than the original database.
- ▶ **At least one tuple** that is modified should be an **error** of the CFD.  
 $\Rightarrow$  CFD should not explain irrelevant modifications.
- ▶ When restricted to the **modified tuples**, the CFD should be **satisfied**.  
 $\Rightarrow$  This ensures that future modifications keep the CFD satisfied.

It is a **local  $M$ -explanation** if it is an  $M'$ -explanation for any  $M' \subseteq M$ .

- ▶ Should be independent of the order of the modifications.

TID	CC	AC	PN	NM	STR	CT	ZIP
$t_1$	01	908	1111111	Mike	Tree Ave.	LA	07974
$t_2$	01	908	1111111	Rick	Tree Ave.	NYC	07974
$t_3$	01	212	2222222	Joe	5th Ave	NYC	01202
$t_4$	01	908	2222222	Jim	Elm Str.	MH	07974
$t_5$	44	131	3333333	Ben	High St.	EDI	EH4 1DT
$t_6$	44	131	4444444	Ian	High St.	EDI	EH4 1DT
$t_7$	44	908	4444444	Ian	Port Pl	MH	W1B 1JH
$t_8$	44	131	2222222	Sean	3rd Str.	UN	01202

- Candidate explanation: FD

$$\varphi = \forall s, t (R(s) \wedge R(t) \wedge s[\text{CC}] = t[\text{CC}] \wedge s[\text{AC}] = t[\text{AC}] \rightarrow s[\text{ZIP}] = t[\text{ZIP}])$$

has an error of 1/8 (tuple  $t_8$  needs to be removed for it be satisfied).

TID	CC	AC	PN	NM	STR	CT	ZIP
$t_1$	01	908	1111111	Mike	Tree Ave.	LA	07974
$t_2$	01	908	1111111	Rick	Tree Ave.	NYC	07974
$t_3$	01	212	2222222	Joe	5th Ave	NYC	01202
$t_4$	01	908	2222222	Jim	Elm Str.	MH	07974
$t_5$	44	131	3333333	Ben	High St.	EDI	EH4 1DT
$t_6$	44	131	4444444	Ian	High St.	EDI	EH4 1DT
$t_7$	44	908	4444444	Ian	Port Pl	MH	W1B 1JH
$t_8$	01	131	2222222	Sean	3rd Str.	UN	01202

- Candidate explanation: FD

$$\varphi = \forall s, t (R(s) \wedge R(t) \wedge s[CC] = t[CC] \wedge s[AC] = t[AC] \rightarrow s[ZIP] = t[ZIP])$$

has an error of 1/8 (tuple  $t_8$  needs to be removed for it be satisfied).

- After the green modification in  $t_8$ , its error becomes 0 (it is satisfied)



TID	CC	AC	PN	NM	STR	CT	ZIP
$t_1$	01	908	1111111	Mike	Tree Ave.	MH	07974
$t_2$	01	908	1111111	Rick	Tree Ave.	MH	07974
$t_3$	01	212	2222222	Joe	5th Ave	NYC	01202
$t_4$	01	908	2222222	Jim	Elm Str.	MH	07974
$t_5$	44	131	3333333	Ben	High St.	EDI	EH4 1DT
$t_6$	44	131	4444444	Ian	High St.	EDI	EH4 1DT
$t_7$	44	908	4444444	Ian	Port Pl	MH	W1B 1JH
$t_8$	01	131	2222222	Sean	3rd Str.	UN	01202

- Candidate explanation: FD

$$\varphi = \forall s, t (R(s) \wedge R(t) \wedge s[CC] = t[CC] \wedge s[AC] = t[AC] \rightarrow s[ZIP] = t[ZIP])$$

has an error of 1/8 (tuple  $t_8$  needs to be removed for it be satisfied).

- After the green modification in  $t_8$ , its error becomes 0 (it is satisfied)
- These next two green modifications in tuples  $t_1$  and  $t_2$ , do not reduce error further and  $\varphi$  remains satisfied.
- It is **not local explanation**, however.

## Example

TID	CC	AC	PN	NM	STR	CT	ZIP
$t_1$	01	908	1111111	Mike	Tree Ave.	LA	07974
$t_2$	01	908	1111111	Rick	Tree Ave.	NYC	07974
$t_3$	01	212	2222222	Joe	5th Ave	NYC	01202
$t_4$	01	908	2222222	Jim	Elm Str.	MH	07974
$t_5$	44	131	3333333	Ben	High St.	EDI	EH4 1DT
$t_6$	44	131	4444444	Ian	High St.	EDI	EH4 1DT
$t_7$	44	908	4444444	Ian	Port Pl	MH	W1B 1JH
$t_8$	44	131	2222222	Sean	3rd Str.	UN	01202

- Candidate explanation: FD

$$\varphi = \forall s, t (R(s) \wedge R(t) \wedge s[CC] = t[CC] \wedge s[AC] = t[AC] \rightarrow s[ZIP] = t[ZIP])$$

has an error of 1/8 (tuple  $t_8$  needs to be removed for it be satisfied).

## Example

TID	CC	AC	PN	NM	STR	CT	ZIP
$t_1$	01	908	1111111	Mike	Tree Ave.	MH	07974
$t_2$	01	908	1111111	Rick	Tree Ave.	MH	07974
$t_3$	01	212	2222222	Joe	5th Ave	NYC	01202
$t_4$	01	908	2222222	Jim	Elm Str.	MH	07974
$t_5$	44	131	3333333	Ben	High St.	EDI	EH4 1DT
$t_6$	44	131	4444444	Ian	High St.	EDI	EH4 1DT
$t_7$	44	908	4444444	Ian	Port Pl	MH	W1B 1JH
$t_8$	44	131	2222222	Sean	3rd Str.	UN	01202

- Candidate explanation: FD

$$\varphi = \forall s, t (R(s) \wedge R(t) \wedge s[CC] = t[CC] \wedge s[AC] = t[AC] \rightarrow s[ZIP] = t[ZIP])$$

has an error of 1/8 (tuple  $t_8$  needs to be removed for it be satisfied).

- When given first modifications to  $t_1$  and  $t_2$ , the error remains 1/8.
- Also note  $t_1$  and  $t_2$  do not violate  $\varphi$ .
- So, indeed not a good explanation.

## Example

TID	CC	AC	PN	NM	STR	CT	ZIP
$t_1$	01	908	1111111	Mike	Tree Ave.	LA	07974
$t_2$	01	908	1111111	Rick	Tree Ave.	NYC	07974
$t_3$	01	212	2222222	Joe	5th Ave	NYC	01202
$t_4$	01	908	2222222	Jim	Elm Str.	MH	07974
$t_5$	44	131	3333333	Ben	High St.	EDI	EH4 1DT
$t_6$	44	131	4444444	Ian	High St.	EDI	EH4 1DT
$t_7$	44	908	4444444	Ian	Port Pl	MH	W1B 1JH
$t_8$	44	131	2222222	Sean	3rd Str.	UN	01202

- Candidate explanation: CFD

$$\forall s, t (R(s) \wedge R(t) \wedge s[CC] = t[CC] \wedge s[AC] = t[AC] \rightarrow s[CT] = t[CT])$$

has an error of 2/8 (tuple  $t_8$  and  $t_1$  or  $t_2$  need to be removed for it be satisfied).

## Example

TID	CC	AC	PN	NM	STR	CT	ZIP
$t_1$	01	908	1111111	Mike	Tree Ave.	MH	07974
$t_2$	01	908	1111111	Rick	Tree Ave.	MH	07974
$t_3$	01	212	2222222	Joe	5th Ave	NYC	01202
$t_4$	01	908	2222222	Jim	Elm Str.	MH	07974
$t_5$	44	131	3333333	Ben	High St.	EDI	EH4 1DT
$t_6$	44	131	4444444	Ian	High St.	EDI	EH4 1DT
$t_7$	44	908	4444444	Ian	Port Pl	MH	W1B 1JH
$t_8$	01	131	2222222	Sean	3rd Str.	UN	01202

- Candidate explanation: CFD

$$\forall s, t (R(s) \wedge R(t) \wedge s[CC] = t[CC] \wedge s[AC] = t[AC] \rightarrow s[CT] = t[CT])$$

has an error of 2/8 (tuple  $t_8$  and  $t_1$  or  $t_2$  need to be removed for it be satisfied).

- Applying any single modification reduces the error to 1/8.
- Applying two modifications reduces it to 0.
- Each modified tuple is involved in a violation.
- This is a local explanation of the modifications.

To further distinguish between local  $M$ -explanations:

### Definition (Score)

$$\text{score}(\varphi, M) := \max\{|M'| \mid M' \subseteq M \text{ and } \varphi \text{ locally explains } M'\}.$$

- ▶ If  $\varphi$  has a score close to  $|M|$ , then almost all modifications in  $M$  are both globally and locally explained.
- ▶ We are thus interested in global explanations with a high score.

### Problem statement

Given instances  $D_{\text{dirty}}$  and  $D_{\text{rep}}$  and modifications  $M$ , find an explanation (CFD) such that  $\text{score}(\varphi, M)$  is maximal.

- ▶ Xplode visits a lattice, such as in TANE, ensuring that the highest score CFD is found without exploring the entire lattice.
- ▶ Pruning strategies are in place, based on an upper bounding technique.
- ▶ “on-demand”, score increases in each step.

```

1 Function: Xplode ( $D_{dirty}, D_{rep}, M, score$ )
2 return best local  $M$ -explanation.

```

---

```

3  $\Phi \leftarrow (\emptyset, \emptyset)$ , score =  $+\infty$ 
4  $\varphi_{max} = \text{nil}$ , max = 0
5 while  $\Phi$  is not empty do
6    $(X, t_p) = \text{Pop}(\Phi)$ 
7   Find best local  $M$ -explanation  $\forall s, t (R(s) \wedge R(t) \wedge \bigwedge_{A_i \in X \setminus A} s[A_i] = t[A_i] = t_p[A_i] \rightarrow s[A] = t[A] = t_p[A])$ 
   based on  $(X, t_p)$ 
8   if score( $\varphi$ ) > max then
9      $\varphi_{max} = \varphi$ , max = score( $\varphi$ )
10    Delete from  $\Phi$  all elements with lower score
11    Test whether any children of  $(X, t_p)$  is a better explanation. If so, add these to  $\Phi$ .
12 return  $\varphi_{max}$ .

```

Declarative Approach

Key Challenges

Static Analysis

Error Detection

**Constraint Discovery**

- FD discovery

- Approximate FD discovery

- DC Discovery

- Explanation-Based

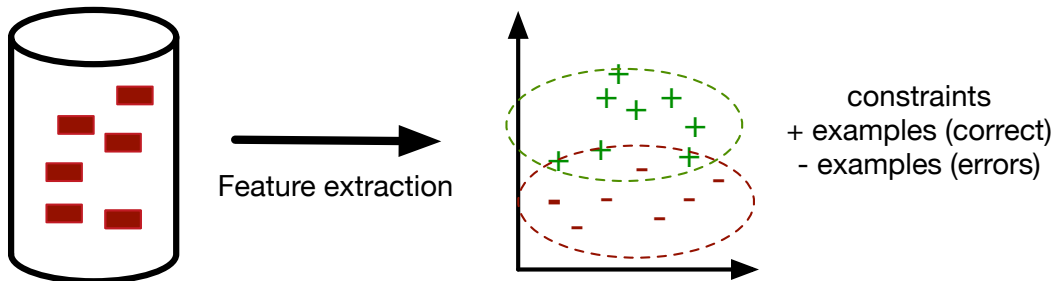
- Conclusion**

Data Repairing

Conclusion



Use **machine learning** approaches for discovering explanations:



Seems a **natural fit**. Promising research direction!

📖 Zhihan Guo and Theodoros Rekatsinas, Unsupervised Functional Dependency Discovery for Data Preparation, ICLR, Learning from Limited Data Workshop 2019

When you identified the right constraints for your data quality problem

- ▶ **Invest** in a discovery method.
- ▶ It allows to **assist domain experts** in formulating constraints that are relevant for your data.
- ▶ **Rich body of techniques** available...

Next ...

- ▶ How to use constraints to **repair the data**.

Declarative Approach

Key Challenges

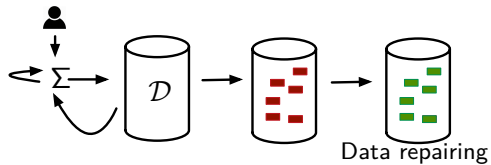
Static Analysis

Error Detection

Constraint Discovery

**Data Repairing**

Conclusion



## Repairing:



How to **fix** the data??

## Definition

A **repair** of a database  $D_{dirty}$  relative to a set  $\Sigma$  of constraints is a database  $D_{rep}$  such that

- ▶  $D_{rep}$  **satisfies** all the constraints in  $\Sigma$ ; and
- ▶  $D_{rep}$  is **close to**  $D_{dirty}$ .

▶ In practice, repairing methods always keep the “**human in the loop**”.

- ▶ A **repair model** indicates what kind of operations are allowed to modify the dirty database into a clean one, e.g., by means of tuple deletions, insertions, value modifications.
- ▶ A **cost function** is used to ensure that a repair minimally differ from original database, e.g., edit distance,...
- ▶ 📖 Recall Jef's and Benny's lectures.

## Example

$\Sigma$ : Key constraint  $StdId \rightarrow StdName$ .

**Dirty database:**

StdId	StdName
123	John
123	Steve
456	Anna
789	Geoff

**Repair model:** tuple deletion

**Cost function:** number of deleted tuples.

**Result:** Two possible repairs, if only tuple deletions are allowed:

StdId	StdName
123	John
456	Anna
789	Geoff

and

StdId	StdName
123	Steve
456	Anna
789	Geoff

In general, there may be exponentially many repairs.

We have seen that a **repair is not unique**.

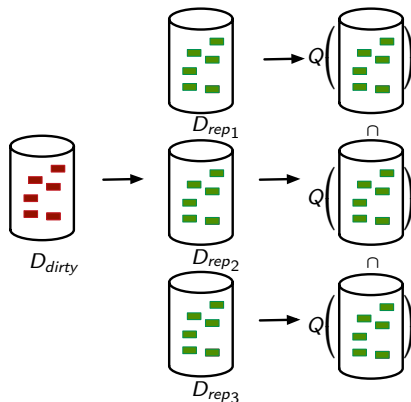
When one wants to **query repairs**, one distinguishes between the following two approaches:

### Consistent query answering

- ▶ **Avoid selecting** a repair; and
- ▶ at query time only return query answers that are common to **all repairs**.

### Data repairing

- ▶ Select the **best possible repair**; and
- ▶ which is subsequently queried.

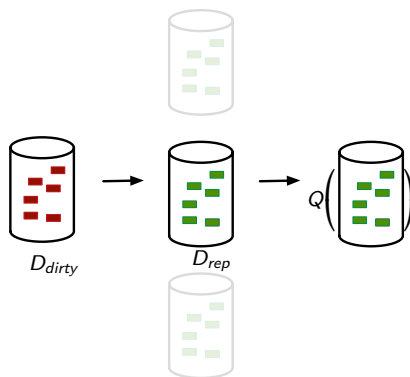


## Challenge

How to compute certain answers **without** computing all repairs.

👉 See Jef's lecture.





Of course, queries can be replaced by any other data analytical tasks.

One distinguishes between:

1. **chase-based** repair methods.

- ▶ Repair is obtained by “firing” constraints until all constraints are satisfied.

2. **Holistic** repair methods:

- ▶ Repair is obtained by taking a “global” view of all errors involved.

3. **Probabilistic** repair methods:

- ▶ Use probabilistic inference to find good repair, possibly not satisfying all constraints.

One distinguishes between:

1. **chase-based** repair methods.

- ▶ Repair is obtained by “firing” constraints until all constraints are satisfied.

2. **Holistic** repair methods:

- ▶ Repair is obtained by taking a “global” view of all errors involved.

3. **Probabilistic** repair methods:

- ▶ Use probabilistic inference to find good repair, possibly not satisfying all constraints.

One distinguishes between:

1. **chase-based** repair methods.
  - ▶ Repair is obtained by “firing” constraints until all constraints are satisfied.
2. **Holistic** repair methods:
  - ▶ Repair is obtained by taking a “global” view of all errors involved.
3. **Probabilistic** repair methods:
  - ▶ Use probabilistic inference to find good repair, possibly not satisfying all constraints.

Declarative Approach

Key Challenges

Static Analysis

Error Detection

Constraint Discovery

Data Repairing

- Chase-based

- Holistic

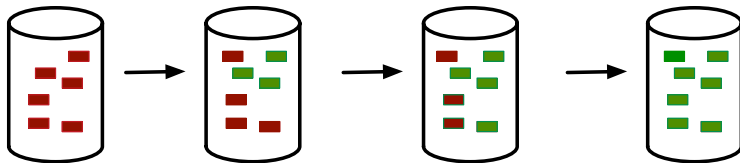
- Probabilistic

- Conclusion

Conclusion

The majority of existing repair methods decide how to repair based in a **local manner**:

- ▶ Repair  $\text{Error}(\varphi, D_{\text{dirty}})$  for each constraint  $\varphi$ , one at a time.
- ▶ Keep doing so, until (hopefully) a repair is obtained.



- ▶ This process is also known as the **chase**: one chases the constraints.

We discuss briefly one chase-based repair method: **LLunatic**

📖 G., G. Mecca, P. Papotti, D. Santoro. The LLUNATIC Data-Cleaning Framework, VLDB 2013

📖 G., G. Mecca, P. Papotti, D. Santoro. Cleaning data with LLUNATIC, The VLDB Journal, 2019

SSN	Name	Phone	STR	CITY	#CC
222	L. Lennon	122-1874	null	SF	7842554
222	L. Lennon	102-111	Fry	SF	7842545
111	J. White	110-1000	Maple	NY	1010101

A key constraint:  $SSN \rightarrow Name, Phone, STR, CITY, \#CC$  or

$$\varphi = \forall s, t (R(s) \wedge R(t) \wedge s[SSN] = t[SSN] \rightarrow \bigwedge_A s[A] = t[A])$$

- In the chase, **changes** must be made to errors of  $\varphi$  such that  $\varphi$  is **satisfied**.

But **which changes?**

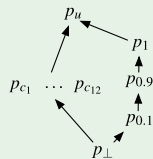
- Is the phone number of L. Lennon 122-1874 or 102111?
- Is the street of L. Lennon Fry?
- What is his credit card number?
- Perhaps they are different Lennon's? So SSN is incorrect?

- ▶ **collect extra information** about the values in the database.
- ▶ each cell is extended with **preference level**.
- ▶ Preference levels related by means of a **partial order**.

SSN	Name	Phone	STR	CITY	#CC
$\langle p_u, 222 \rangle$	$\langle p_{c1}, \text{L. Lennon} \rangle$	$\langle p_{0.9}, 122-1874 \rangle$	$\langle p_{\perp}, \text{null} \rangle$	$\langle p_{c6}, \text{SF} \rangle$	$\langle p_{c10}, 7842554 \rangle$
$\langle p_u, 222 \rangle$	$\langle p_{c2}, \text{L. Lennon} \rangle$	$\langle p_{0.1}, 102-111 \rangle$	$\langle p_{c4}, \text{Fry} \rangle$	$\langle p_{c7}, \text{SF} \rangle$	$\langle p_{c11}, 7842545 \rangle$
$\langle p_u, 111 \rangle$	$\langle p_{c3}, \text{J. White} \rangle$	$\langle p_1, 110-1000 \rangle$	$\langle p_{c5}, \text{Maple} \rangle$	$\langle p_{c9}, \text{NY} \rangle$	$\langle p_{c12}, 1010101 \rangle$

Preference levels:

- ▶  $p_{c1}-p_{c12}$ : no other information
- ▶  $p_u$ : user certified
- ▶  $p_x$ : confidence degree of  $x$
- ▶  $p_{\perp}$ : null value





## Chasing with constraint

$$\varphi = \forall s, t (R(s) \wedge R(t) \wedge s[\text{SSN}] = t[\text{SSN}] \rightarrow \bigwedge_A s[A] = t[A])$$

means that **relevant cells** are **merged** in order to satisfy  $\bigwedge_A s[A] = t[A]$

- Relevant cells: involved in errors of the constraint.

SSN	Name	Phone	STR	CITY	#CC
$\langle p_u, 222 \rangle$	$\{\langle p_{c1}, \text{L. Lennon} \rangle, \langle p_{c2}, \text{L. Lennon} \rangle\}$	$\{\langle p_{0.9}, 122-1874 \rangle, \langle p_{0.1}, 102-111 \rangle\}$	$\{\langle p_{\perp}, \text{null} \rangle, \langle p_{c4}, \text{Fry} \rangle\}$	$\{\langle p_{c6}, \text{SF} \rangle, \langle p_{c7}, \text{SF} \rangle\}$	$\{\langle p_{c10}, 7842554 \rangle, \langle p_{c11}, 7842545 \rangle\}$
$\langle p_u, 111 \rangle$	$\langle p_{c3}, \text{J. White} \rangle$	$\langle p_1, 110-1000 \rangle$	$\langle p_{c5}, \text{Maple} \rangle$	$\langle p_{c9}, \text{NY} \rangle$	$\langle p_{c12}, 1010101 \rangle$

- This **process continues** for other constraints and errors.
- In each step, cells **gather information** (preference levels/values)
- Result: database with sets of preference level attribute pairs.
- **Termination**: cells grow in each step and there is an upper bound on information that can be collected.

- ▶ Zoom in on a specific cell.
- ▶ It carries a set  $S$  of preference level/value pairs  $\langle p_1, v_1 \rangle, \dots, \langle p_k, v_k \rangle$

▶ **conflict resolution** is applied based on preference levels.

### Definition

Given a set  $S = \{\langle p_1, v_1 \rangle, \dots, \langle p_k, v_k \rangle\}$  its **preferred value** is defined as

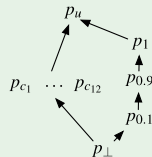
$$\begin{cases} v & \text{if } v \text{ is the } \mathbf{value} \text{ associated with the } \mathbf{highest} \text{ preference level} \\ \mathbf{llun} & \text{if no such value exist.} \end{cases}$$

- ▶ **Llun is special value** indicating that insufficient information is available to repair the conflict to an actual value.

SSN	Name	Phone	STR	CITY	#CC
$\langle p_u, 222 \rangle$	$\{\langle p_{c1}, \text{L. Lennon} \rangle, \langle p_{c2}, \text{L. Lennon} \rangle\}$	$\{\langle p_{0.9}, 122-1874 \rangle, \langle p_{0.1}, 102-111 \rangle\}$	$\{\langle p_{\perp}, \text{null} \rangle, \langle p_{c4}, \text{Fry} \rangle\}$	$\{\langle p_{c6}, \text{SF} \rangle, \langle p_{c7}, \text{SF} \rangle\}$	$\{\langle p_{c10}, 7842554 \rangle, \langle p_{c11}, 7842545 \rangle\}$
$\langle p_u, 111 \rangle$	$\langle p_{c3}, \text{J. White} \rangle$	$\langle p_1, 110-1000 \rangle$	$\langle p_{c5}, \text{Maple} \rangle$	$\langle p_{c9}, \text{NY} \rangle$	$\langle p_{c12}, 1010101 \rangle$

Preferred values:

- ▶  $\{\langle p_{c1}, \text{L. Lennon} \rangle, \langle p_{c2}, \text{L. Lennon} \rangle\}$ : L. Lennon
- ▶  $\{\langle p_{0.1}, 102-111 \rangle, \langle p_{0.9}, 122-1874 \rangle\}$ : 122-1874
- ▶  $\{\langle p_{\perp}, \text{null} \rangle, \langle p_{c4}, \text{Fry} \rangle\}$ : Fry
- ▶  $\{\langle p_{c6}, \text{SF} \rangle, \langle p_{c7}, \text{SF} \rangle\}$ : SF
- ▶  $\{\langle p_{c10}, 7842554 \rangle, \langle p_{c11}, 7842545 \rangle\}$ : llun.




- ▶ Preferred values are obtained based on all constraints and errors in the data.

- ▶ Preferred values are put in cells: Repair.

SSN	Name	Phone	STR	CITY	#CC
222	L. Lennon	122-1874	Fry	SF	llun
111	J. White	110-1000	Maple	NY	1010101

- ▶ Llunatic comes with a **variety of preference levels** encoding
  - ▶ User, data from master data, constants in constraints, ...
- ▶ Provides **fine-grained control** of the chase by
  - ▶ using cost functions to only perform changes that minimize the cost function
- ▶ Also allows **left-hand side repairs**
  - ▶ basically disabling constraints.

 <https://github.com/donatelloosantoro/Llunatic>

Declarative Approach

Key Challenges

Static Analysis

Error Detection

Constraint Discovery

**Data Repairing**

Chase-based


**Holistic**

Probabilistic

Conclusion

Conclusion

We next discuss repair algorithm **Holistic**:

 X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. ICDE 2013

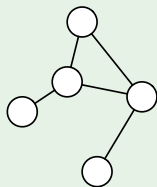
### Underlying idea:

1. Consider simple denial constraints
2. Encode all violations of constraints as a **conflict hypergraph**
3. Find a **minimal vertex cover** of this hypergraph
4. Collect information from the cover to **resolve conflicts**
5. **Repair** the data accordingly, if new conflicts, repeat.

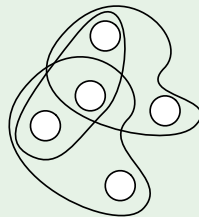
## Definition (hypergraph)

A hypergraph  $G = (V, E)$  is an extension of a graph in which the edges are **sets of vertices**. Edges are called **hyper edges**.

## Example of graph and hypergraph



graph



hypergraph

### Errors

	TID	FN	LN	ROLE	ZIP	ST	SAL
$t_1$		Anne	Nash	E	85376	NY	110
$t_2$		Mark	White	M	90012	NY	80
$t_3$		Mark	Lee	E	85376	AZ	75

Two denial constraints:

$$e_1 : \neg(\exists s, t R(s) \wedge R(t) \wedge s[\text{ZIP}] = t[\text{ZIP}] \wedge s[\text{ST}] \neq t[\text{ST}])$$

which is just an FD:  $\text{ZIP} \rightarrow \text{ST}$ . and

$e_2 : \neg(\exists s, t R(s) \wedge R(t) \wedge s[\text{ST}] = t[\text{ST}] \wedge s[\text{ROLE}] = M \wedge t[\text{ROLE}] = E \wedge s[\text{SAL}] < t[\text{SAL}])$ ,  
requiring that “managers should earn more than employees when they live in the same state”.



## Errors

TID	FN	LN	ROLE	ZIP	ST	SAL
$t_1$	Anne	Nash	E	85376	NY	110
$t_2$	Mark	White	M	90012	NY	80
$t_3$	Mark	Lee	E	85376	AZ	75

Two denial constraints:

$$e_1 : \neg(\exists s, t R(s) \wedge R(t) \wedge s[\text{ZIP}] = t[\text{ZIP}] \wedge s[\text{ST}] \neq t[\text{ST}])$$

which is just an FD:  $\text{ZIP} \rightarrow \text{ST}$ . and

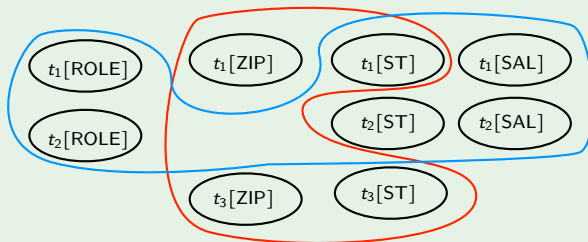
$e_2 : \neg(\exists s, t R(s) \wedge R(t) \wedge s[\text{ST}] = t[\text{ST}] \wedge s[\text{ROLE}] = M \wedge t[\text{ROLE}] = E \wedge s[\text{SAL}] < t[\text{SAL}])$ ,  
 requiring that “managers should earn more than employees when they live in the same state”.

## Step 2: Create Hypergraph

### Conflict hypergraph

TID	FN	LN	ROLE	ZIP	ST	SAL
$t_1$	Anne	Nash	E	85376	NY	110
$t_2$	Mark	White	M	90012	NY	80
$t_3$	Mark	Lee	E	85376	AZ	75

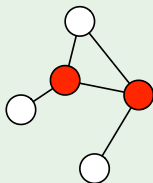
Conflict hypergraph:



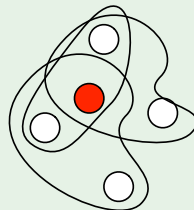
## Definition

Given a hypergraph  $G = (V, E)$ . A **minimal** set of vertices that are **intersecting with every hyperedge**,

## Example of covers for graph and hypergraph



graph

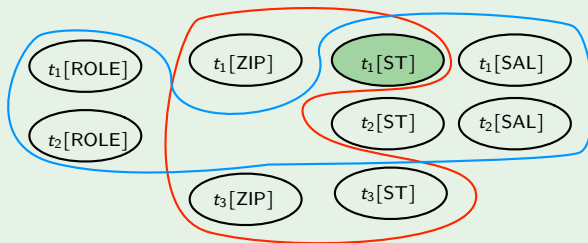


hypergraph

NP-hard problem. Good heuristics are available.

### Minimal cover

In our conflict hypergraph:  $t_1[\text{ST}]$  is a minimal cover



- ▶ A repair can be found by changing the value of nodes in the minimal cover!
- ▶ Minimal number of changes!

**Extract a set of conditions** (repair requirements) that need to be satisfied to resolve all violations

### Repair conditions

Conditions on  $t_1[ST]$ :

► For

$$e_1 : \neg(\exists s, t R(s) \wedge R(t) \wedge s[ZIP] = t[ZIP] \wedge s[ST] \neq t[ST])$$

**change  $t_1[st]$  into  $t_3[st]$**

► For

$$e_2 : \neg(\exists s, t R(s) \wedge R(t) \wedge s[ST] = t[ST] \wedge s[ROLE] = M \wedge t[ROLE] = E \wedge s[SAL] < t[SAL]),$$

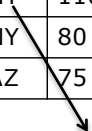
**change  $t_1[st]$  into something different than  $t_2[st]$ .**

In general, inspect constraints and identify constraints that would eliminate all **current** hyperedges (=no previous conflicts)

## Step 4: Get Updates

A set of **assignments** satisfying the repair conditions, with **minimal number** of changed cells  
⇒ Perhaps  $t_1[st] = AZ??$  Ok!

	ID	FN	LN	ROLE	ZIP	ST	SAL
$t_1$	105	Anne	Nash	E	85376	NY	110
$t_2$	211	Mark	White	M	90012	NY	80
$t_3$	386	Mark	Lee	E	85376	AZ	75



AZ

If not possible to find such a value: introduce a **fresh constant** different from anything else.

- ▶ If the change incurs new conflicts: Repeat the whole procedure.

**Termination** is guaranteed since in the worst case all values have **new “fresh” constants**.

- ▶ Most constraint formalisms won't detect errors on such datasets with only unique values.

## The Holistic repair algorithm

📖 X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. ICDE 2013

- ▶ comes with many extra optimizations
- ▶ different (approximation) algorithms for finding minimal covers
- ▶ and heuristics to reduce the number of “fresh” constants...



Declarative Approach

Key Challenges

Static Analysis

Error Detection

Constraint Discovery

**Data Repairing**

Chase-based


Holistic

**Probabilistic**

Conclusion

Conclusion

Most notable example is **Holoclean**

 Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré, Holoclean: Holistic data repairs with probabilistic inference, PVLDB 10 (2017)

Key ideas:

1. Associate **random variables to cells** in instances
2. Describe constraints (and other background knowledge) and relax these with **weights**
3. **Probability distribution on possible worlds**

► Most probable possible world=best repair

- ▶ Each cell is assigned a **variable**.
- ▶ **Assignments** of these variables to domain values: **possible world**
- ▶ 📖 Recall Antoine's, Benny and Jef's lectures.

## Possible world

TID	Name	Address	City	State	Zip
$t_1$	N1	S1	A1	ST1	Z1
$t_2$	N2	S2	A2	ST2	Z2
$t_3$	N3	S3	A3	ST3	Z3
$t_4$	N4	S4	A4	ST4	Z4

Example assignment, corresponding to the actual dirty database:

TID	Name	Address	City	State	Zip
$t_1$	John	Maple Av	Chicago	IL	60608
$t_2$	John	Maple Av	Chicago	IL	60609
$t_3$	John	Maple Av	Chicago	IL	60609
$t_4$	Johnn	Maple Av	Cicago	IL	60608

Different assignment: different database (exponentially many).

- Compile constraints and background knowledge in constraints+ weights.

## Weighted constraints

TID	Name	Address	City	State	Zip
$t_1$	N1	S1	A1	ST1	Z1
$t_2$	N2	S2	A2	ST2	Z2
$t_3$	N3	S3	A3	ST3	Z3
$t_4$	N4	S4	A4	ST4	Z4

Variables get assigned **domains**:

- $N_i \leftarrow \text{Domain}(\text{Name})$
- $S_i \leftarrow \text{Domain}(\text{Address})$
- $A_i \leftarrow \text{Domain}(\text{City})$
- $ST_i \leftarrow \text{Domain}(\text{State})$
- $Z_i \leftarrow \text{Domain}(\text{Zip})$

Initial values get **weights**:

- $N_i \leftarrow \text{Initial}(N_i, \text{John}), w_i$ , for  $i = 1, 2, 3$
- $N_4 \leftarrow \text{Initial}(N_4, \text{Johnn}), w_4$ .
- ...

These reflect **prior** information about correctness of cell values.

## Weighted constraints

TID	Name	Address	City	State	Zip
$t_1$	N1	S1	A1	ST1	Z1
$t_2$	N2	S2	A2	ST2	Z2
$t_3$	N3	S3	A3	ST3	Z3
$t_4$	N4	S4	A4	ST4	Z4

Constraints get added:

- ▶  $\forall s, t \neg (R(s) \wedge R(t) \wedge s[ZIP] = t[ZIP] \wedge s[State] \neq t[State])$ .
- ▶  $\neg((Z_i \wedge Z_j) \wedge (ST_i \wedge ST_j)) \leftarrow Z_i = Z_j \wedge ST_i \neq ST_j, w$

If  $w = +\infty$ , then this is a hard constraint.

If  $w < +\infty$ , this is a soft constraint.

- ▶ Result: a set of weighted constraints on the random variables in cells.

- Each assignment of variables results in a **weighted possible world**.

### Definition

Let  $\mathcal{V}$  a set of random variables and  $\alpha$  an assignment of variables to domain values. Let  $D_\alpha$  denote the corresponding possible world. Given a set of weighted constraints  $(\Sigma, \mathbf{w})$  the weight of  $D_\alpha$

$$W(D_\alpha, \Sigma, \mathbf{w}) = \sum_{\varphi \in \Sigma} W(D_\alpha, \varphi, w)$$

where, intuitively,

$$W(D_\alpha, \varphi, w) = w \times f(\text{size of the set of satisfying tuples}).$$

This leads to a **probability distribution on possible worlds**:

### Definition

$$\text{Prob}_{\Sigma, \mathbf{w}}(D_\alpha) = \frac{1}{Z} e^{W(D_\alpha, \Sigma, \mathbf{w})}.$$

## Problem statement

Find the assignment  $\alpha$  such that  $\text{Prob}_{\Sigma, \mathbf{w}}(D_\alpha)$  is **maximized**.

- ▶ **Intractable problem**: search is over all possible worlds
- ▶ HoloClean: lot of effort to make this **scalable in practice**.

Solving this problem:

- ▶ Efficient probabilistic inference techniques, factor graphs, ....
- ▶ Sampling, approximation, ...

See <http://www.holoclean.io/>

- ▶  $t_2[ZIP] = 60608$ : probability 0.84
- ▶  $t_2[ZIP] = 60609$ : probability 0.16
- ▶  $t_4[CITY] = \text{Chicago}$ : probability 0.95
- ▶  $t_4[CITY] = \text{Cicago}$ : probability 0.05
- ▶  $t_4[Name] = \text{John}$ : probability 0.99
- ▶  $t_4[Name] = \text{Johnn}$ : probability 0.01

TID	Name	Address	City	State	Zip
$t_1$	John	Maple Av	Chicago	IL	60608
$t_2$	John	Maple Av	Chicago	IL	<b>60608</b>
$t_3$	John	Maple Av	Chicago	IL	<b>60608</b>
$t_4$	<b>John</b>	Maple Av	<b>Cicago</b>	IL	60608



- ▶ Large-scale deduplication with constraints using dedupalog
- ▶ On approximating optimum repairs for functional dependency violations
- ▶ Guided data repair
- ▶ ERACER: a database approach for statistical inference and data cleaning
- ▶ Interaction between record matching and data repairing
- ▶ Towards certain fixes with editing rules and master data
- ▶ NADEEF: a commodity data cleaning system
- ▶ Data quality aware queries in collaborative information systems
- ▶ A unified model for data and constraint repair
- ▶ Holistic data cleaning: Putting violations into context
- ▶ GDR: a system for guided data repair
- ▶ Sampling from repairs of conditional functional dependency violations
- ▶ Continuous data cleaning
- ▶ Active repair of data quality rules
- ▶ Ranking for data repairs
- ▶ Descriptive and prescriptive data cleaning
- ▶ Repair checking in inconsistent databases: algorithms and complexity.

Declarative Approach

Key Challenges

Static Analysis

Error Detection

Constraint Discovery

**Data Repairing**

Chase-based

Holistic

Probabilistic

**Conclusion**

Conclusion

Similar to error detection and explanation-based constraint discovery:

- ▶ Techniques start to shift to machine learning
- ▶ Learn good repair modification in a unsupervised, or semi-supervised way.

*“ Data Cleaning is a Machine Learning Problem that Needs Data Systems. ”(Ihab Ilyas, ACM SIGMOD blog post 2019)*

- ▶ Interesting new repairing methods are around the corner...

Declarative Approach

Key Challenges

Static Analysis

Error Detection

Constraint Discovery

Data Repairing

Conclusion

- ▶ I have painted a **positive** picture of the constraint-based approach to data cleaning.
  - ▶ There are, however, lots of things that need to be done:
    - ▶ Open-source implementations
    - ▶ Use cases
    - ▶ Benchmarking
    - ▶ Seamless integration of different components
- ▶ So, although there is ample room for further research (especially in the ML context)
  - ▶ Engineering effort is needed to push forward this approach!!
- ▶ You're invited to join this effort!

- ▶ Use of **machine learning** techniques will become prominent
- ▶ Different constraints needed to clean labels of data for ML
- ▶ Constraint-based approach is currently being revisited for **graph data** (Semantic Web)
- ▶ Incremental, fast, approximate methods needed to deal with **fast changing data**.
- ▶ ....

More info:

📖 Ihab F. Ilyas and Xu Chu, Data Cleaning, ACM Books, 2019

📖 Ihab F. Ilyas and Xu Chu, Trends in Cleaning Relational Data: Consistency and Deduplications, Foundations and Trends in Databases 2015.

📖 V. Ganti & A. Das Sarma, Data Cleaning: A Practical Perspective, Synthesis Lectures on Data Management, 2013.

📖 Foundations of Data Quality Management, by Wenfei Fan, Floris Geerts, Morgan & Claypool, 2012.

📖 Herzog, Scheuren & Winkler, Data Quality and Record Linkage Techniques, Springer, 2007

- ▶ Use of **machine learning** techniques will become prominent
- ▶ Different constraints needed to clean labels of data for ML
- ▶ Constraint-based approach is currently being revisited for **graph data** (Semantic Web)
- ▶ Incremental, fast, approximate methods needed to deal with **fast changing data**.
- ▶ ....

More info:

📖 Ihab F. Ilyas and Xu Chu, Data Cleaning, ACM Books, 2019

📖 Ihab F. Ilyas and Xu Chu, Trends in Cleaning Relational Data: Consistency and Deduplications, Foundations and Trends in Databases 2015.

📖 V. Ganti & A. Das Sarma, Data Cleaning: A Practical Perspective, Synthesis Lectures on Data Management, 2013.

📖 Foundations of Data Quality Management, by Wenfei Fan, Floris Geerts, Morgan & Claypool, 2012.

📖 Herzog, Scheuren & Winkler, Data Quality and Record Linkage Techniques, Springer, 2007