Lecture 1

Introduction to Constraint-Based Data Quality

What's the problem and why constraints can help!

Data Cleaning Course

VLDB 2017 Summer School, Beijing

Outline

1. **The data quality problem**
2. Constraint-based data cleaning
3. Alternative approaches
4. What's next?

A real-world encounter

"Mr. Smith, our database records indicate that you owe us an amount of 1,921.76 GBP for council tax in Edinburgh in 2016"

A data quality problem:

- M. Smith moved from Edinburgh to London in 2015, and no longer lived in Edinburgh in 2016;
- The council database was not correctly updated: it retains both Smith's old address and his new address.

Customer table

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

Statistics

50% of bills have errors

Customer data

Customer table

country	AC	phn	street	city	zip
44	131	1234567	Mayfield	New York	EH8 9LE
44	131	3456789	Crichton	New York	EH8 9LE
01	908	3456789	Mountain Ave	New York	07974

Anything wrong?

Customer data

Customer table

country	AC	phn	street	city	zip
44	131	1234567	Mayfield	New York	EH8 9LE
44	131	3456789	Crichton	New York	EH8 9LE
01	908	3456789	Mountain Ave	New York	07974

Anything wrong?

- New York City is moved to the UK (country code: 44)
- Murray Hill (country/area code: 01/908) in New Jersey is moved to New York state.

Statistics

Customer records have error rates 10% – 75% (telecommunication)

Real-world data is often dirty

Dirty data:

Data that is inconsistent, inaccurate, incomplete, stale, or deliberately falsified.

Examples:

- US: Pentagon asked 200+ dead officers to re-enlist;
- UK: there are 81 million national insurance numbers but only 60 million people eligible;
- Australia: 500,000 dead people retain active medicare cards;
- In a database of 500,000 customers, 120,000 records become invalid within a year – death, divorce, marriage, move.

How does data get dirty?

Errors and inconsistencies may be introduced during data gathering, storage, transmission, transformation, integration, ...

Dirty data is costly

Telecommunication services: dirty data routinely leads to failure to bill for services, delay in repairing network problems, unnecessary leasing of equipment ⇒ loss of revenue, credibility, customers

More examples:

- Poor data costs US companies $600 billions annually;
- Erroneously priced data in retail databases costs US customers $2.5 billion each year;
- World-wide losses from payment card fraud reached $4.84 billion in 2006;
- 30% – 80% of the development time for data cleaning in a data integration project.

This is true also in finance, life science, e-government,...

No 1. problem

Data quality: The No 1. problem for data management!

The need for data quality tools

Manual effort: beyond reach in practice!

- For instance, editing a sample of census data easily took dozens of clerks several months (Winkler 04, US Census Bureau).

Data quality tools

- To automatically
  - discover data quality rules;
  - reason about these rules;
  - detect errors based on violations of these rules; and
  - repair (or suggest repairs) of data.

  and this in a **principled way**.

Existing tools

ETL

Most data quality tools adhere to ETL:

1. Extraction: Data is collected from sources;
2. Transformation: Rules and functions are applied on the data;
3. Loading: Results are loaded in customer's database (warehouse).

- For specific domain,e.g., address data;
- Transformation rules are manually designed;
- Low-level programs.

There are many good systems and prototypes around, e.g, AJAX, Potter's Wheel, Usher, Guided Data Repair, Nadeev, LLunatic,...

Our goal:

Is to complement existing tools by providing a uniform approach to several data quality tasks.

What is data quality? Some criteria

Consistency

Whether the data contains errors or conflicts that emerge as violations of certain semantic rules.

Example: age = 82 and age = 20 for the same patient.

Accuracy

How close a value representing a real-life entity is to the true value of the entity.

Example: age of high school students: ≤ 40 vs. age = 15.

Completeness

Whether a given query can be answered given the available information.

Example: age = **null** (missing value) in a patient record, or missing patient record (missing tuple).

Timeliness

Whether the data is too out-of-date to answer a query.

Example: Council tax collection in 2016 based on an old address of 2015.

Outline

1. The data quality problem
2. **Constraint-based data cleaning**
   1. Standard dependencies
   2. Conditional dependencies
   3. Other kinds of dependencies
3. Alternative approaches
4. What's next?

Outline

1. The data quality problem
2. **Constraint-based data cleaning**
   1. **Standard dependencies**
   2. Conditional dependencies
   3. Other kinds of dependencies
3. Alternative approaches
4. What's next?

Constraint-based data cleaning

Aim

We want various **integrity constraint formalisms** to help us to achieve a fundamental approach for improving the quality of data.

- As a warm-up, I illustrate this first by means of standard dependencies (functional & inclusion dependencies);
- Argue the need for extending these to accommodate for some of the data quality criteria; and
- Present various classes of data quality constraints.

Example: functional dependency

Functional dependencies

Consider FD: customer(NI# $\rightarrow$ name, AC, phn, street, city, zip)

- NI# is a key: there is a unique record for each distinct NI#.

Consider instance $\mathcal{D}$ of R:

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.
- For SC1234566, at least one of the records must be dirty.

Error detection

Functional dependencies help to detect errors in a **single** relation.

Functional dependencies

Let R be a relational schema and let $\mathcal{D}$ be a database instance of R.

Simple functional dependencies

If A_1, A_2,..., A_m, B are attributes of R, then we say that $\mathcal{D}$ satisfies the functional dependency (FD)

$$R([A_1, \ldots, A_m] \rightarrow B)$$

if whenever **two** tuples in $\mathcal{D}$ **agree** on the values in $A_1, \ldots, A_m$, then they **also agree** on the value of B.

This is also denoted by $\mathcal{D} \models R([A_1, \ldots, A_m] \rightarrow B)$.

Example - inclusion dependencies

Inclusion dependencies

Consider IND: book[asin, title, price] $\subseteq$ item[asin, title, price]

- Every book sold by a store must be an item carried by the store.

book:

	asin	isbn	title	price
t_1:	a23	b32	Harry Potter	17.99
t_2:	a56	b65	Snow white	7.94

item:

asin	title	type	price
a23	Harry Potter	book	17.99
a12	John Denver	CD	7.94

- These instances do not satisfy the IND.
- Tuple t_2 does not have a counter part in the item table.

Error detection

Inclusion dependencies help to detect errors **across** relations.

Inclusion dependencies

Let R and S be relational schemas and let $\mathcal{D}$ and $\mathcal{D}'$ be database instances of R and S, respectively.

Inclusion dependencies

If A_1, A_2,..., A_n are distinct attributes of R, and $B_1, \ldots, B_n$ distinct attributes of S, then we say that $(\mathcal{D}, \mathcal{D}')$ satisfies the inclusion dependency (IND)

$$R[A_1, \ldots, A_n] \subseteq S[B_1, \ldots, B_n]$$

if for **every** tuple t in $\mathcal{D}$, there **exists** a tuple s in J such that $t[A_1, \ldots, A_n] = s[B_1, \ldots, B_n]$.

This is also denoted by $(\mathcal{D}, \mathcal{D}') \models R[A_1, \ldots, A_n] \subseteq S[B_1, \ldots, B_n]$.

Outline

1. The data quality problem
2. **Constraint-based data cleaning**
    1. Standard dependencies
    2. **Conditional dependencies**
    3. Other kinds of dependencies
3. Alternative approaches
4. What's next?

The need for revising traditional constraints

Example

Consider the instance:

	CC	AC	phn	name	street	city	zip
t_1:	44	131	1234567	Mike	Mayfield	NYC	EH4 8LE
t_2:	44	131	3456789	Rick	Crichton	NYC	EH4 8LE
t_3:	01	908	3456789	Joe	Mtn Ave	NYC	07974

Consider the functional dependencies:

fd_1: [CC, AC, phn] $\rightarrow$ [street]
fd_2: [CC, AC] $\rightarrow$ [city, zip].

The database **satisfies** the FDs. But the data is NOT clean! (As we will see shortly.)

- Traditional constraints were designed for improving the quality of relational schemas.
- We need constraints for improving the quality of data.

Capturing inconsistencies in the data

Example cnt'd

	CC	AC	phn	name	street	city	zip
t_1:	44	131	1234567	Mike	Mayfield	NYC	EH4 8LE
t_2:	44	131	3456789	Rick	Crichton	NYC	EH4 8LE
t_3:	01	908	3456789	Joe	Mtn Ave	NYC	07974

This instance is not clean since we know the following **semantic properties**:

- "In the UK, the zip code uniquely determines the street".
- "In the UK, if the area code is 131, then the city must be Edinburgh (EDI)".
- "In the USA, if the area code is 908, then the city must be Murray Hill (MH)".

- These properties cannot be enforced by standard FDs.
- How to minimally extend FDs?

Intermezzo: First-order logic (FO)

Atoms:	$R(x_1, \ldots, x_k)$ with R a relation
	$x = y$ or $x = c$ for variables x and y and constant c
Inductive Def:	Atoms are FO formulas
	Let $\varphi(\bar{x})$ and $\psi(\bar{y})$ be FO formulas, then
	$\theta(\bar{x}, \bar{y}) = \varphi(\bar{x}) \vee \psi(\bar{y})$ is an FO formula (disjunction)
	$\theta(\bar{x}, \bar{y}) = \varphi(\bar{x}) \wedge \psi(\bar{y})$ is an FO formula (conjunction)
	$\theta(\bar{x}) = \neg\varphi(\bar{x})$ is an FO formula (negation)
	$\theta(\bar{x}) = \exists x\, \varphi(x, \bar{x})$ is an FO formula (exists)
	$\theta(\bar{x}) = \forall x\, \varphi(x, \bar{x})$ is an FO formula (forall)

Here, $\bar{x}$ stands for a sequence $x_1, \ldots, x_k$ of variables.
The quantifiers $\exists$ and $\forall$ remove one variable.

An FO **sentence** is a formula in which at the end no variables are left, e.g., $\exists x \forall y R(x, y)$.

The **semantics** of an FO formula φ on a database instance $\mathcal{D}$ is defined inductively.

Reminder - FDs as logical sentences

Logical formalism for FDs

An FD

$$R([A_1, \ldots, A_m] \rightarrow B)$$

can be written as:

$$\forall t_1, t_2 \Big(R(t_1) \wedge R(t_2) \wedge \bigwedge_{i \in [1,m]} t_1[A_i] = t_2[A_i] \rightarrow (t_1[B] = t_2[B]) \Big).$$

Here, t_1 and t_2 as shorthand notation for a bunch of variables.
Also, $\varphi(\bar{x}) \rightarrow \psi(\bar{y}) \equiv (\neg\varphi(\bar{x})) \vee \psi(\bar{y})$.

To express the previous semantic properties in a similar formalism we need to **add equality with constants**.

Conditional functional dependencies (CFDs)

Example

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

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

More compactly:

$$\text{cfd}_1 : R([\text{CC} = 44, \text{zip}] \rightarrow [\text{street}])$$

- It is a **conditional** FD: it may not hold for other countries, e.g., USA.
- It cannot be expressed as standard FDs: it needs constants.
- The example database does not satisfy this constraint.

Conditional functional dependencies (CFDs)

Example

"In the UK, if the area code is 131, then the city must be Edinburgh (EDI)".

$$\forall t_1, t_2 \big(R(t_1) \wedge R(t_2) \wedge t_1[\text{CC}] = t_2[\text{CC}] \wedge t_1[\text{AC}] = t_2[\text{AC}] \wedge t_1[\text{CC}] = 44 \wedge t_1[\text{AC}] = 131 \rightarrow (t_1[\text{city}] = t_2[\text{city}] \wedge t_1[\text{city}] = \text{EDI}) \big).$$

"In the USA, if the area code is 908, then the city must be Murray Hill (MH)".

$$\forall t_1, t_2 \big(R(t_1) \wedge R(t_2) \wedge t_1[\text{CC}] = t_2[\text{CC}] \wedge t_1[\text{AC}] = t_2[\text{AC}] \wedge t_1[\text{CC}] = 01 \wedge t_1[\text{AC}] = 908 \rightarrow (t_1[\text{city}] = t_2[\text{city}] \wedge t_1[\text{city}] = \text{MH}) \big).$$

---

cfd_2: ([CC = 44, AC = 13] → [city = 'EDI'])
cfd_3: ([CC = 01, AC = 908] →[city = 'MH'])

Conditional functional dependencies (CFDs)

Example

Given the CFDs:

cfd$_1$: ([CC = 44, zip] $\rightarrow$ [street])
cfd$_2$: ([CC = 44, AC = 13] $\rightarrow$ [city = 'EDI'])
cfd$_3$: ([CC = 01, AC = 908] $\rightarrow$[city = 'MH'])

All tuples in the instance are dirty:

	CC	AC	phn	name	street	city	zip
t_1:	44	131	1234567	Mike	Mayfield	NYC	EH4 8LE
t_2:	44	131	3456789	Rick	Crichton	NYC	EH4 8LE
t_3:	01	908	3456789	Joe	Mtn Ave	NYC	07974

This, despite the fact that the instance satisfied the FDs.

Conditional functional dependencies thus capture more dirtiness that standard FDs.

The need for extending inclusion dependencies

Inclusion dependencies

Consider IND: item[asin, title, price] $\subseteq$ book[asin, title, price]

item:

	asin	title	type	price
s_1:	a23	Harry Potter	book	17.99
s_2:	a12	John Denver	CD	7.94

book:

asin	isbn	title	price
a23	b32	Harry Potter	17.99
a56	b65	Snow white	7.94

- These instances do not satisfy the IND.
- Tuple s_2 does not have counter part in the item table.
- s_2 corresponds to a CD, not a book!

Semantic property

"The IND only makes sense for tuples corresponding to books"

Reminder - INDs as logical sentences

Logical formalism for INDs

An IND

$$R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$$

can be written as

$$\forall t \big(R(t) \to (\exists s\, S(s) \wedge \bigwedge_{i \in [1,m]} t[A_i] = s[B_i])\big).$$

To express the previous semantic property we need to **add equality with constants**.

Conditional inclusion dependencies

Example

"The IND item[asin, title, price] $\subseteq$ book[asin, title, price] only holds for books."

$$\forall t\big(\text{item}(t) \wedge t[\text{type}] = \text{"book"} \rightarrow (\exists s\, \text{book}(s) \wedge t[\text{asin}] = s[\text{asin}] \wedge t[\text{title}] = s[\text{title}] \wedge t[\text{price}] = s[\text{price}])\big).$$

Shorthand: item[asin, title, price, type=book] $\subseteq$ book[asin, title, price]

item:

asin	title	type	price
a23	Harry Potter	book	17.99
a12	John Denver	CD	7.94

book:

asin	isbn	title	price
a23	b32	Harry Potter	17.99
a56	b65	Snow white	7.94

Similarly as for CFDs, we thus add conditions to inclusion dependencies.

Capturing inconsistencies across relations

Example

Consider CIND:

item[asin, title, price, type=book] $\subseteq$ book[asin, title, price]

item:

asin	title	type	price
a23	Harry Potter	book	17.99
a56	Snow white	book	17.94

book:

asin	isbn	title	price
a23	b32	Harry Potter	17.99
a56	b65	Snow white	7.94

Conditional inclusion dependencies are more flexible than their standard counter parts, and capture more dirtiness.

What did we learn?

Observation:

- CFDs and CINDs have shown useful in the **detection** of dirty tuples.
- Only **minor modifications** to well-known constraint formalisms were needed (adding constants to FDs and INDs).

Can we detect other kinds of dirtiness as well?

References:

CFDs Conditional functional dependencies for capturing data inconsistencies, W Fan, F Geerts, X Jia, A Kementsietsidis, TODS, 2008.

CINDs Extending inclusion dependencies with conditions, S. Ma, W. Fan, L. Bravo, TCS, 2014.

Outline

1. The data quality problem
2. **Constraint-based data cleaning**
   1. Standard dependencies
   2. Conditional dependencies
   3. **Other kinds of dependencies: Matching dependencies, ordered dependencies, metric dependencies, ...**
3. Alternative approaches
4. What's next?

Constraints for detecting other kinds of dirtiness

General recipe:

1. Consider a specific data quality task;
2. Identify the minimal requirements needed to catch inconsistencies related to the data quality task;
3. Incorporate these requirements in simple kinds of dependencies (constraints); and
4. Investigate their properties, practical usefulness and ability to detect dirtiness and improve the quality of data.

Inconsistencies on ordered attributes

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."

Inconsistencies on ordered attributes

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."*

Ordered functional dependencies (OFDs)

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

$$\forall s, t : \text{Emp}\big(s[\text{Job}] > t[\text{Job}] \rightarrow s[\text{Salary}] > t[\text{Salary}]\big)$$

expresses that "the salary of an employee is greater than other employees who have junior job titles". Similarly,

$$\forall s, t : \text{Emp}\big(s[\text{Job}] = t[\text{Job}] \wedge s[\text{Years}] > t[\text{Years}] \rightarrow s[\text{Salary}] > t[\text{Salary}]\big)$$

expresses that "the salary for employees with the same job title is greater for those with more years in the company."

OFDs vs FDs

OFDs extend FDs on ordered domains by allowing $<$, $\leqslant$, $>$ and $\geqslant$ comparisons.

Reference:

OFDs Ordered functional dependencies in relational databases, Wilfred Ng, Information Systems, 1999.

Denial Constraints

Example

"Two people living the same state should have correct tax rates depending on their income"

$$\forall s, t \in \mathcal{D} \neg (s[\mathrm{AC}] = t[\mathrm{AC}] \wedge s[\mathrm{SAL}] < t[\mathrm{SAL}] \wedge s[\mathrm{TR}] > t[\mathrm{TR}])$$

In general a denial constraint says that something **should not** hold.

Can express

- Key constraints: $\neg(R(x, y) \wedge R(x, y') \wedge y \neq y'')$
- Functional dependencies (similar as key constraint)
- Many more...

Inconsistencies on metric data

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."*

Inconsistencies on metric data

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."*

Inconsistencies on metric data

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"*

Inconsistencies on metric data

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"*

Metric dependencies (MFDs)

Example MFDs

For the movie database, let $\text{dist}(x, y) = |x - y|$ be a distance function that measures the absolute value of the difference of two numeric values. Consider

$$\forall s, t : \text{Movie}\big(s[\text{Title}] = t[\text{Title}] \rightarrow \text{dist}(s[\text{Duration}], t[\text{Duration}]) \leqslant 6\big)$$

For the location database, let $\text{dist}((x_1, x_2), (y_1, y_2)) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$ be the Euclidean distance between points in the plane. Consider

$$\forall s, t : \text{Loc}\big(s[\text{Addr}] = t[\text{Addr}] \rightarrow \text{dist}((s[\text{Lat}], s[\text{Long}]), (t[\text{Lat}], t[\text{Long}])) \leqslant 0.005\big)$$

MFDs vs FDs

MFDs extend FDs by allowing distance predicates in the antecedent (left-hand side) of an FD.

Reference:

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

How to reason over most recent values?

Currency dependencies (CDs) for timeliness of data

- "Divorce comes after marriage": Tuples with "Divorce" are more recent than those of "Marriage" (provided that no remarriage happens of course...).
- Suppose that some currency information is provided then

$$\forall s, t : \text{Emp}\big(s[\text{eid}] = t[\text{eid}] \wedge s[\text{status}] = \text{divorced} \wedge t[\text{status}] = \text{married} \rightarrow t \prec_{\text{curr}} s\big).$$

  must be satisfied, i.e. t is more current than s.
- Semantic properties of the data are used to infer temporal relationships.

CDs vs FDs

CDs extend FDs by allowing a temporal partial order $\prec$ on each attribute: $a \prec b$ if b is more recent than a.

Incorporating user interaction?

Editing rules (eRs)

- "If we know that the zip code of a tuple is correct, and if a user can provide a correct area code, street and city, for that zip code, then take the values from the user.":
$$\forall s, t\, R(s) \wedge R_u(t) \wedge s[\text{zip}] = t[\text{zip}] \rightarrow \bigwedge_{B \in \text{AC, str, city}} s[B] = t[B].$$
- eRs provide a active semantics to CFDs and incorporate user interaction (by means of R_u).

eRs vs CFDs

Editing rules extend CFDs by incorporating special relations to model User interaction or to incorporate Master data.

Reference:

Record matching

Record matching/object identification

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

Credit card fraud

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

Statistics

World-wide losses in fraud in 2006: $4.84 billion (source: SAS)

Non-trivial problem

Reasons

- Real-life data is often dirty: errors in the data sources; and
- Data is often represented differently in different sources.

Credit card fraud

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	3256777	1pm/7/7/09	EDI	$3,500
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Max	Smith	PO Box 25, EDI	456789	2pm/7/7/09	NYC	$6,300

Pairwise comparing attributes **via equality** only does not work!

Matching dependencies

"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"

Matching dependency (MD):

$$\forall s, t\big(\text{card}(s) \wedge \text{trans}(t) \wedge s[LN] = t[LN] \wedge s[address] = t[post] \wedge s[FN] \asymp t[FN] \rightarrow s[X] = t[Y]\big),$$

where $\asymp$ is a similarity operator and X and Y are compatible attributes in card and trans, respectively.

Reference:

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

How are MDs used for matching?

Dynamic semantics

Matching tuples are obtained from an instance that satisfies the MDs.

Credit card fraud

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	3256777	1pm/7/7/09	EDI	$3,500
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Mark	Smith	PO Box 25, EDI	456789	2pm/7/7/09	NYC	$6,300

Matching keys

A minimal set of attributes can be identified that allow to match two tuples.

Observations

MDs vs FDs

A matching dependency is like an FD, except that

- equalities can be relaxed to similarities; and
- it relates to two, possibly distinct, relations.
- duplicate records are found by tuples that satisfy the MDs (rather than violate it).

Further extension

Conditional Matching Dependencies (CMDs): Extension of MDs with constant equalities (like in CFDs).

Reference:

CMDs Extending Matching Rules with Conditions, S. Song, L. Chen, J. Xu Yu, QDB, 2010.

Record Linkage - related declarative approaches

- Matching dependencies (semantics, algorithms):

  Data Cleaning and Query Answering with Matching Dependencies and Matching Functions. L. Bertossi, S. Kolahi, L. Lakshmanan, TCS, 2013. Theory Comput. Syst. 52(3): 441-482 (2013)

  Matching dependencies: semantics and query answering. J. Gardezi, L. Bertossi, I. Kiringa, Frontiers of Computer Science, 2012.

- Identifying matches as solutions for "link constraints" (close to matching dependencies)

  A Declarative Framework for Linking Entities, D. Burdick, R. Fagin, Ph. Kolaitis, L. Popa, W-C. Tan, ICDT, 2015.

We come back to record linkage/entity resolution later in the course.

Overview of different "data quality" dependencies

Comparison table

FDs	equality	key constraints
CFDs	equality + constants	data value inconsistencies
MDs	equality+ similarity	record matching
OFDs	equality + inequality	ordered value inconsistencies
MFDs	equality + distance (RHS)	distance-based inconsistencies
CDs	equality + partial order	currency conflicts
ERs	equality + user	user-value inconsistencies
⋮	⋮	⋮

and more ...

Differential dependencies, sequential dependencies, synonym rules.

Conclusion

They all tackle specific kinds of "dirtiness" yet are "simple" extensions of FDs.

RFD abbrev.	RFD name
ACOD	Approximate comparable dependency
ADD	Approximate differential dependency
AFD	Approximate functional dependency
COD	Comparable dependency
CFD	Conditional functional dependency
CFD^p	CFD with built-in predicates
CFD^c	CFD with cardinality constraints and synonym rules
CMD	Conditional matching dependency
CSD	Conditional sequential dependency
CD	Constrained functional dependency
DD	Differential dependency
eCFD	Extended conditional functional dependency
FFD	Fuzzy functional dependency
MD	Matching dependency
MFD	Metric functional dependency
ND	Neighborhood dependency
NuD	Numerical dependency
OD	Order dependency
OD_K	OD satisfied within bound k
OD_{EA}	OD satisfied almost everywhere
OFD	Ordered functional dependency
PD	Partial determination
POD	Polarized order dependencies
preFD	Preference functional dependency
PAC	Probabilistic approximate constraint
pFD	Probabilistic functional dependency
PuD	Purity dependency
RUD	Roll-up dependency
SD	Sequential dependency
SFD	Similarity functional dependency
soft FD	Soft functional dependency
TD	Trend dependency
TMFD	Type-M functional dependency
XCFD	XML conditional functional dependency
$\sigma\theta$XFD	XML FD with σ and θ approximation

Generalization

- Remember, I promised a **unified** approach...
- We ended up with a **zoo** of quality constraints
- They can, however, all be described in an **extension of First-order-logic**
  - Use **Built-in predicates**.

A **built-in predicates** is like an "atom" ($R(\bar{x})$ and $x = y$) in FO, instead that they can **represent whatever you want**, as long as it returns true or false on its input.

They can be used as atoms in FO (but of course this considerable extends FO).

For example x op y could mean that the Euclidean distance between x and y is smaller than a threshold.

Quality improving dependencies (QIDs): general notion

Consider

$$\forall s \forall t \left(R(s) \wedge R'(t) \wedge \bigwedge_{A} s[A] \, \mathrm{op}_A \, t[A] \rightarrow s[B] \, \mathrm{op}_B \, t[B] \right),$$

where op stands for $=$, $<$, $\leqslant$, $>$, $\geqslant$, $\asymp$, $\preceq$, some distance function, ...

- Provides a unified logical formalism.
- Can be even generalised further as an extension of so-called equality-generating dependencies.
- A similar formalism can be defined for INDs (and tuple-generating dependencies).

Why using QIDs?

Reasons:

- They capture a fundamental part of the semantics of data:
  - Errors and inconsistencies as violations of dependencies.
- They are declarative:
  - Their logical formalism allows to reason with them.
- As we will see later, they can not only be used to detect dirty data but also to repair the data.

Claim

Having a declarative specification helps a lot, especially when it comes to algorithms and optimizations!!

Outline

1. The data quality problem
2. Constraint-based data cleaning
3. **Alternative approaches**
4. What's next?

Alternative approaches

Of course, not all errors can be caught by QIDs:

- Many **specialized data cleaning algorithms** exist for
  - entity resolution
  - outlier detection
  - data analysis
- I (very briefly) overview methods for **single and multiple column analysis**
  - Cardinalities and datatypes
  - Co-occurrences and summaries
- These methods are part of what is known as **data profiling**

  See recent ICDE 2016 tutorial by Abedjan, Golab and Naumann for more on data profiling.

Cardinalities and distributions

- **Counting**
  - Number of values
  - Number of distinct values
  - Number of NULLs
- **Range information**
  - MIN and MAX value
- **Distributions**
  - Histograms
  - Probability distribution for numeric values
  - Detect whether data follows some well-known distribution

Count distinct in sublinear time and space?

- Linear Counting [Whang, Vander-Zanden, Taylor: A linear-time probabilistic counting algorithm for database applications. TODS, 1990]
- Stochastic Averaging [Flajolet, Martin: Probabilistic counting algorithms for data base applications. JCSS, 1985]
- Loglog Algorithm [Durand, Flajolet: Loglog counting of large cardinalities. Algorithms-ESA, 2003]
- SuperLogLog Algorithm [Durand, Flajolet: Loglog counting of large cardinalities. Algorithms-ESA, 2003]
- HyperLogLog Algorithm [Flajolet, Fusy, Gandouet, Meunier: Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. DMTCS, 2008]

$\Rightarrow$ A tutorial in its own...

Data types and value patterns

Ordered in increasing complexity to detect:

- String vs. number
- String vs. number vs. date
- Categorical vs. continuous (e.g., Days of the week vs. measurements)
- SQL data types (e.g., CHAR, INT, DECIMAL, TIMESTAMP, BIT, CLOB, ...)
- Domains (VARCHAR(12) vs.VARCHAR(13))
- Regular expressions
- Semantic domains (e.g., Address, phone, email, firstname, lastname)

These checks are often part of the **data preparation process** before any cleaning is done.

Frequency, Rules, Correlations

Most important when analyzing multiple columns (attributes):

Frequencies:

- Which values co-occur with each other?

Rules:

- Which values depend on a specific value?

Correlations:

- Which values correlate?
- Which values substitute each other?

Core step: Frequent Itemset Mining

Origin: Transactional Analysis

- Which products have been bought together?

Main step:

- Find frequencies for all item combinations

Optimization:

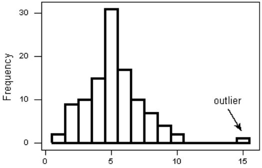
- Find frequencies for all relevant item combinations, i.e., item combinations with minimum support

Algorithms:

- Apriori [Aggrawal, Srikant: fast Algorithms for Mining Association rules, VLDB'94]
- FP-Growth [Han, Pei, Yin: Mining frequent patterns without candidate generation, SIGMOD'00]
- More information: Survey [Hipp, Guentzer, Nakhaeizadeh: Algorithms for Association Mining – A General Survey and Comparison, KDD'00].

Outlier detection

- Low-frequent values
- Structural outliers
  - Wrong value representations, e.g.: CA instead of California
  - Numerical outliers, e.g., according to Gaussian distribution
- Outlier combinations
  - Co-occurrence analysis



See Survey [Hodge, Austin: A survey of outlier detection methodologies, AI'04]

Sketches & Summaries

- **Use cases:**
  - Assess column similarity
  - Dimension reduction
  - Data stream samples
- **Techniques:**
  - Sampling
  - Hashing, e.g., Minhash, Locality sensitivity hashing
  - Sketches [Cormode, Garofalakis, Haas, Jermaine: Synopses for Massive Data:Samples, Histograms, Wavelets, Sketches, FTD'12]

$\Rightarrow$ Probably already have seen examples of this during the lectures in the previous two days.

Outline

1. The data quality problem
2. Constraint-based data cleaning
3. Alternative approaches
4. **What's next?**

We focus on constraint-based data cleaning

1. Error detection and constraint discovery
2. Data repairing using constraints
3. Entity resolution
4. Statistics & constraint-based data quality