Data Quality

Eugene Wu

Data Quality

Prevention

- Normalization prevents inconsistencies
- Constraints prevent invalid values

Detection

- Incorrect/anomalous values
- Deduplication

Steps for a New Application

Requirements
- what are you going to build?

Conceptual Database Design
- pen-and-pencil description

Logical Design
- formal database schema

Schema Refinement:
- fix potential problems, normalization

Normalization

Physical Database Design
- use sample of queries to optimize for speed/storage

App/Security Design
- prevent security problems

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report

Redundancy = Bad

What was our solution?

- Break database into small relations
- Perform “good” ER modeling and SQL translation
- Kind of ... adhoc

Is there a systematic approach?

Redundancy = Bad

<u>sid</u>	name	address	<u>hobby</u>	cost
1	Eugene	amsterdam	trucks	$$
1	Eugene	amsterdam	cheese	$
2	Bob	40th	paint	$$$
3	Bob	40th	cheese	$
4	Shaq	florida	swimming	$

people have names and addrs
hobbies have costs
people many-to-many with hobbies
    What's primary key? sid? sid + hobby?

Update/insert/delete anomalies. Wastes space

Anomalies (Inconsistencies)

Update Anomaly

change one address, need to change all

Insert Anomaly

add person without hobby?

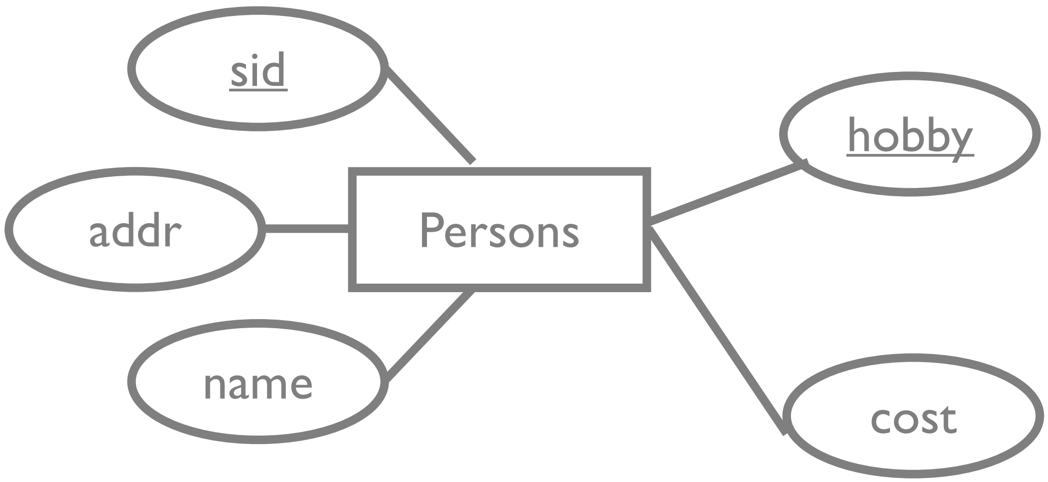
not allowed? dummy hobby?

Delete Anomaly

if delete a hobby. Delete the person?

Theory Can Fix This!

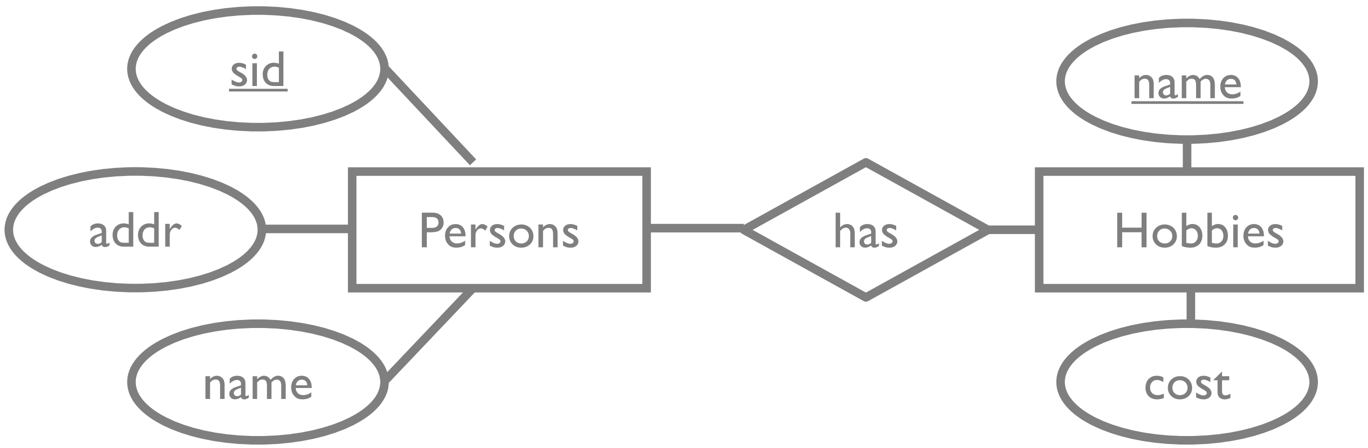
A Possible Approach



`data(sid, addr, name, hobby, cost)`

A Possible Approach

ER diagram was a heuristic



We have decomposed example table into:

```
person(sid, addr, name)
hobby(name, cost)
personhobby(hobbyname, sid)
```

WHY is this a good decomposition??

A Possible Approach

What if decompose into:

```
person(sid, name, addr, cost)
personhobby(sid, hobbyname)
```

sid	name	addr	cost
1	Eugene	amsterdam	$$
1	Eugene	amsterdam	$
2	Bob	40th	$$$
3	Bob	40th	$
4	Shaq	florida	$

sid	hobby
1	trucks
1	cheese
2	paint
3	cheese
4	swimming

but… which cost goes with which hobby?

lost information: *lossy decomposition*

Decomposition

Replace schema R with 2+ smaller schemas that

1. each contain subset of attrs in R
2. together include all attrs in R

ABCD replaced with AB, BCD or AB, BC, CD

Not free – may introduce problems!

1. lossy-join: not able to recover R from smaller relations
2. non-dependency-preserving: constraints on R cannot be enforced y only looking at an individual decomposed relation
3. performance: additional joins, may affect performance

Can we systematically decompose our relation to

prevent decomposition problems

&

remove redundancy?

Functional Dependencies (FD)

sid	name	address	hobby	cost
1	Eugene	amsterdam	trucks	$$
1	Eugene	amsterdam	cheese	$
2	Bob	40th	paint	$$$
3	Bob	40th	cheese	$
4	Shaq	florida	swimming	$

sid sufficient to identify name and addr, but not hobby

e.g., exists a function f(sid) → name, addr

sid → name, addr is a functional dependency

"sid determines name, addr"

"name, addr are functionally dependent on sid"

"if 2 records have the same sid, their name and addr are the same"

Functional Dependencies (FD)

$$X \rightarrow Y$$

holds on R

if $t_1.X = t_2.X$ then $t_1.Y = t_2.Y$

where X,Y are subsets of attrs in R

Examples of FDs in person-hobbies table

sid, hobby → name, address cost

hobby → cost

sid → name, address

Redundancy depends on FDs

consider R(ABC)

no FDs: no redundancy

if A→B: tuples with same A value means B is duplicated!

Fun Facts

Functional Dependency is an integrity constraint

- statement about all instances of relation
- Generalizes key constraints
  - if K is candidate key of R, then K → R

Given FDs, simple definition of redundancy

- when left side of FD is not table key

Where do FDs come from?

- thinking really hard aka application semantics
- can't stare at database to derive (like ICs)

Fun Facts

Functional Dependency is an integrity constraint

- statement about all instances of relation
- Generalizes key constraints
  - if K is candidate key of R, then K → R

Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms

Thorsten Papenbrock[2] Jens Ehrlich[1] Jannik Marten[1]
Tommy Neubert[1] Jan-Peer Rudolph[1] Martin Schönberg[1]
Jakob Zwiener[1] Felix Naumann[2]
[1] firstname.lastname@student.hpi.uni-potsdam.de
[2] firstname.lastname@hpi.de
Hasso-Plattner-Institut, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany

We’re going to need some theory

Closure of FDs

- armstrong’s axioms

Principled Decomposition

BCNF

Closure of FDs

Given

Name → Bday and Bday → age

We know

Name → age

f' is implied by set F if f' is true when F is true

F^+ closure of F is all FDs implied by F

Can we construct this closure automatically? YES

Closure of FDs

Inference rules called Armstrong's Axioms

Reflexivity	if $Y \subseteq X$ then $X \rightarrow Y$
Augmentation	if $X \rightarrow Y$ then $XZ \rightarrow YZ$ for any Z
Transitivity	if $X \rightarrow Y$ & $Y \rightarrow Z$ then $X \rightarrow Z$

These are sound and complete rules

sound	doesn't produce FDs not in the closure
complete	doesn't miss any FDs in the closure

Closure of FDs

Can we compute the closure? YES. slowly

expensive. exponential in # attributes

Can we *check* if X → Y is in the closure of F?

X^+ = *attribute closure* of X (expand X using axioms)

check if Y is implied in the attribute closure

Closure of FDs

F = {A→B, B→C, CB→E}

Is A→E in the closure?

A → B	given
A → AB	augmentation A
A → BB	apply A→B (transitivity)
A → BC	apply B→C (transitivity)
A → E	apply BC→E (transitivity)

We’re going to need some theory

- Closure of FDs
    - armstrong’s axioms
- Principled Decomposition
- BCNF

Decomposition

Eventually want to decompose R into $R_1 \ldots R_n$ wrt F

We've seen issues with decomposition.

Lost Joins: Can't recover R from $R_1 \ldots R_n$

Lost dependencies

Principled way of avoiding these?

Lossless Join Decomposition

Let's say relation R is decomposed into relations X, Y

Join the decomposed tables to get *exactly the* original

$\pi_X(R) \bowtie \pi_Y(R) = R$

Lossless wrt F if and only if F^+ contains

$X \cap Y \rightarrow X$ or $Y \cap X \rightarrow Y$

intersection of X, Y is a key for one of them

Lossless Join Decomposition

Lossless wrt F if and only if F^+ contains

$X \cap Y \rightarrow X$ or $Y \cap X \rightarrow Y$

intersection of X,Y is a key for one of them

FDs: $A \rightarrow C, A \rightarrow B$

A	B	C
1	2	1
5	3	4
9	2	6

A	B
1	2
5	3
9	2

B	C
2	1
3	4
2	6

A	B	C
1	2	1
5	3	4
9	2	6
1	2	6
9	2	1

Lossy! $AB \cap BC = B$ doesn't determine anything

Lossless Join Decomposition

Lossless wrt F if and only if F^+ contains

$X \cap Y \rightarrow X$ or $Y \cap X \rightarrow Y$

intersection of X,Y is a key for one of them

FDs: $A \rightarrow C, A \rightarrow B$

A	B	C
1	2	1
5	3	4
9	2	6

→

A	B
1	2
5	3
9	2

A	C
1	1
5	4
9	6

→

A	B	C
1	2	1
5	3	4
9	2	6

OK

Dependency-preserving Decomposition

F_R = Projection of F onto R

Subset of F^+ that are "valid" for R

FDs $X \rightarrow Y$ in F^+ where X and Y attrs are in R

If R decomposed into relations A and B.

FDs that hold on A, B are equivalent to all FDs on R

$(F_X \cup F_Y)^+ = F^+$

Consider ABCD, C is key, $AB \rightarrow C$, $D \rightarrow A$

BCNF decomposition: BCD, DA

$AB \rightarrow C$ doesn't apply to either table. Not dependency preserving.

We’re going to need some theory

Closure of FDs

- armstrong’s axioms

Principled Decomposition

BCNF

BCNF

Relation R in BCNF has *no redundancy* wrt FDs

(FDs are all key constraints)

```
F: set of functional dependencies over relation R

for (X→Y) in F+
   Y is in X or
   X is a superkey of R
```

Is this in BCNF?

sid → name

sid	hobby	name
x	y_1	z
x	y_2	?

BCNF

Relation R in BCNF has *no redundancy* wrt FDs

(FDs are all key constraints)

```
F: set of functional dependencies over relation R

for (X→Y) in F+
    Y is in X or
    X is a superkey of R
```

Functional Dependencies

SH → NAC (sid, hobby → name, addr, cost)
H → C
S → NA

What's in BCNF?

SHNAC	NO
SNA, SHC	NO
SNA, HC, SH	YES

BCNF

Relation R in BCNF has *no redundancy* wrt FDs

(FDs are all key constraints)

```
F: set of functional dependencies over relation R

for (X→Y) in F⁺
   Y is in X or
   X is a superkey of R
```

Remember that F^+ is *all* possible FDs valid over R!

Given A→B, B→C, then A→C is in F and should be checked

BCNF

Suppose we have

Client, Office → Account

Account → Office

What's in BCNF?

R(Account, Client, Office)

R(Account, Office)   R(Client, Account)

Where did CO→A go? *Lost a Functional Dependency*

BCNF does not preserve FDs

BCNF Decomposition

```
while BCNF is violated
    R with FDs F_R
    if an FD X→y violates BCNF
        turn R into R-y & Xy
```

y is single attr

ABCDE

key A, BC→A, D→B, C→D

Consider D→B

Decompose into DB, ACDE

DB is in BCNF

D→B

ACDE is in BCNF

A is key, C → A

Seems like we lost BC→A!

BCNF Decomposition in English

BCNF is based on checking *all* FDs (in F^+) for a given relation.

Simple procedure:

1. For each attribute or combination of attributes in R. Call it *attrs*
2. Does the closure of *attrs* contain all attributes in R?
3. If yes: continue. Else: decompose
4. Recall that *attrs*→*attrs* is trivially true.

Given A→B, B→C, and relation ABCD

Consider A:	it determines B, and C, but not D
Decompose using A→B:	AB, ACD
Consider A for ACD:	it determines C, but not D
Decompose using A→C:	AB, AC, AD

Other Normal Forms

Two different criterias for decomposing a relation

Boyce Codd Normal Form (BCNF)

- No redundancy, may lose dependencies

3rd Normal Form (3NF)

- May have redundancy, no decomposition problems
- Common

1st, 2nd, 4th, … normal forms

Common Data Quality Issues (incomplete list)

Value Errors

- Finding outliers
- Robustness to outliers
- Deal with outliers/value errors (Imputation)

Entity Resolution

Value Errors

Values in an attribute that appear wrong

- missing
- placeholder values e.g., 0, -9999, "NA", "Empty"
- "anomalous" values (outliers)

Why are outliers bad?

- causes estimates (AVG) to be arbitrarily wrong
- AVG(2, 2, 2, 2, 5) = 2.6
- AVG(2, 2, 2, 2, 100) = 21.6
- AVG(2, 2, 2, 2, 1000) = 201.6

Finding Outliers

How to detect (numeric) outliers?

- Fundamentally a modeling problem.
- What's a "normal" value? Assume a distribution

Example using age

- Assume a normal distribution
- Fit age values to distribution (compute avg, stddev)
- Low-probability ages (outside of mean±std) labeled outliers

Robustness to Outliers

Can the query be resilient to outliers?

Breakdown point:

- % of incorrect values before statistic is arbitrarily wrong
- AVG: one value
- Median: 50%

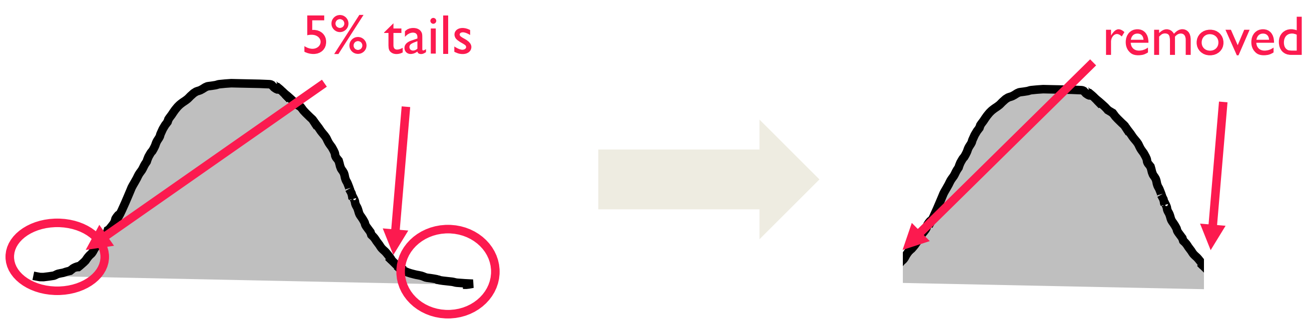
```
1, 2, 3, 4, 5, 6, 7, 8, 100
```

AVG	15.1	
Median	5	Sort and return middle value

Deal With Outliers

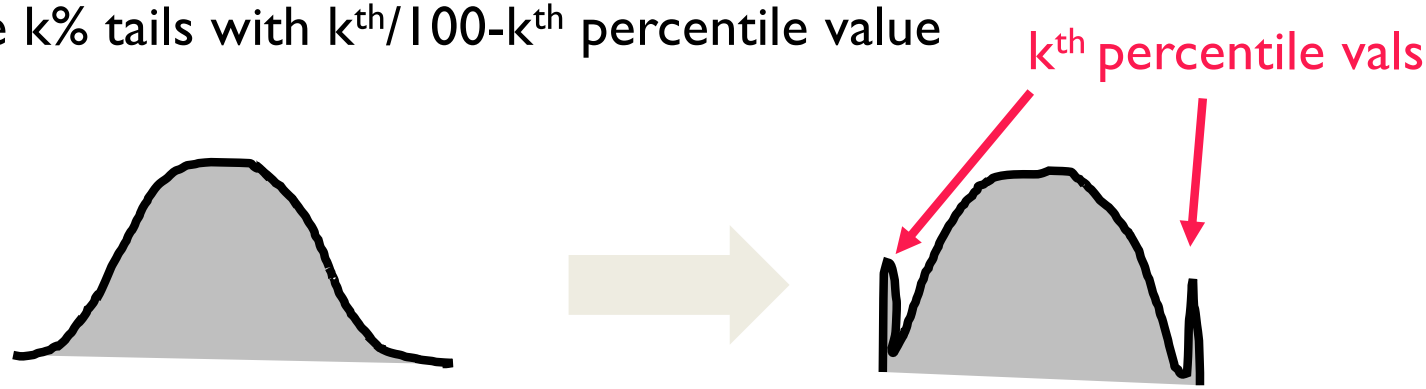
Trimming 5%

- remove top and bottom 5% percentile (the “tails”)
- p5, p95



Winsorize

- Replace extreme values with closest value
- Replace k% tails with k^{th}/100-k^{th} percentile value



Imputing Missing Value

Fill in with “likely” values

Why?

- Some apps don’t accept nulls
- Lead to bias/errors

How to define “likely”?

- default values (mean/median, user-defined val)
- interpolate (if ordered data)
- build a model and predict

Entity Resolution

Same entity, different "physical representations"

Example: text attributes due to e.g., misspellings

- New York, NYC, New York City, NY City, …
- Different forms of string similarity

```
SELECT *
FROM cities c1, cities c2
WHERE levenshtein(c1.name, c2.name) < 3
```

Entity Resolution

Example: different rows refer to the same entity.

- Seller 1: `(123, “iphone max 12Pro,” $1000)`
- Seller 2: `(00123, “IPhone12 Pro MAX”, “85722.40 Rubles”)`
- Fundamentally classification problem. is_same(row1, row2) → T/F

```
SELECT row1, row2
FROM data as d1, data as d2
WHERE is_same(d1.*, d2.*)
```

Entity Resolution

Very hard problem! No "Solution".

10000 records. 100,000,000 possible matches!

3 main steps

1. **Blocking**: group data into subsets (may overlap)
2. **Matching**: within each group:
    1. choose distance function between rows
    2. cluster rows using distance function.

Hope that same entities in same cluster, and

different entities in different clusters

Entity Resolution

Blocking:

- Extract grouping features/attributes A
- Form groups based on same/similar A values
- Each group is a "block"

Simple approach: group by an attribute

Example for strings: group by shared n-gram

2-gram for "eugene": eu, ug, ge, en, ne
All words containing "eu" in one block
Choose n via trial-and-error

Entity Resolution: Matching

Distance metrics

Single attribute

- string similarity functions
- synonym dictionary: fast = quick
- knowledge-base: *mandarin* closer to *orange* than banana
- embeddings

Multi-attribute:

- linear combination of attribute distances
  ```
  (4111, “intro to db”), (COMS4111, “Introducton to DB”),
  (4112, “db impl”)
  ```
- train a model if you have labels

Summary

Normalization

- Functional dependencies & redundancy
- FD closures: Armstrong's axioms
- Proper Decomposition into BCNF

Data Quality

- Value errors.
- Entity resolution

Summary

Trade-off

- Accidental redundancy is really really bad
- But adding lots of joins can hurt performance

In practice:

- List FDs
- normalize,
- decide which decompositions to skip as needed

What you should know

Purpose of normalization

- Anomalies
- Decomposition problems
- Functional dependencies & axioms

BCNF

- properties
- algorithm

Data Quality (level of lecture)

- Value errors: How to detect, avoid, fix
- Entity resolution: What it is, why it's hard, overall steps

Why The Term “Normalize Relations”?

Designing a database involves a process called normalization where the data model is broken-down according its smallest granularity. Invented by Ted Codd and Chris Date, the term normalization was borrowed from President Richard Nixon normalizing relations with China in the 1970's. Codd stated that if Nixon could normalize relations, then so could the relational model.