# Improving Data Quality: Consistency and Accuracy


| | |
|---|---|
| Gao Cong, | Microsoft Research Asia |
| Wenfei Fan, | University of Edinburgh, Bell Laboratories |
| Floris Geerts, | Univ. of Edinburgh, |
| | Hasselt Univ., transnational Univ. Limburg |
| **Xibei Jia**, | University of Edinburgh |
| Shuai Ma, | University of Edinburgh |


25th September 2007

# Dirty data are costly

- Typical data error rate in industry: 1% - 5%, up to 30%
- Poor data cost US companies $600 billion annually
- 30%-80% of the development time for data cleaning in a **data warehousing** project
- CIA intelligence on WMD in Iraq!

These dirty data need to be cleaned (semi-)automatically !

# Constraint-based data cleaning

- Constraint-based data cleaning
  - Define a set of constraints to model the data
  - Errors in data are captured as violations of these constraints
  - These violations are then repaired to improve data quality
- Constraints used in previous data cleaning tools
  - Functional Dependencies
  - Inclusion Dependencies
  - Denial Constraints
  - …

**Are these traditional constraints sufficient for cleaning data?**

# Functional Dependencies (FDs)

**[ CC, AC ] → [ City ]**

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 19355 |
| t4 | John | 44 | 131 | CHI | EH8 9LE |

These data are consistent, but are they clean?

# FDs → CFDs: flashback

**[ CC, AC ] → [ City ]**

**【 CC , AC 】 → 【 City 】**

| CC | AC | City |
|---|---|---|
| - | - | - |
| 44 | 131 | EDI |

**FDs**

for schema design

**CFDs**

for data cleaning

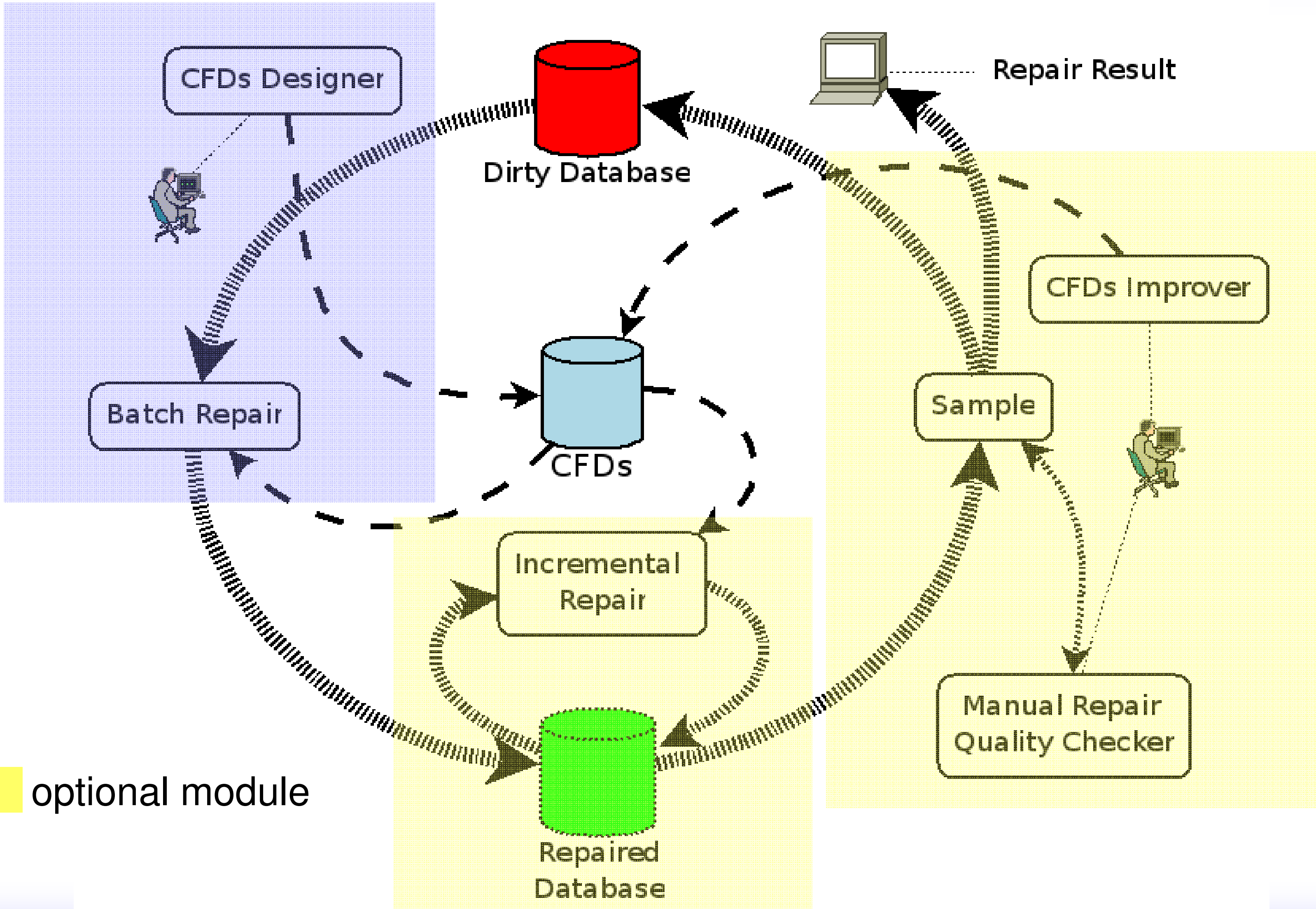
- Data integration in real-life: source constraints
  - hold on a subset of sources
  - hold conditionally on the integrated data
- They are NOT expressible as traditional FDs
  - do not hold on the entire relation
  - contain constant data values

# Conditional Functional Dependencies (CFDs)

| 【 CC , | AC 】 | → 【 City 】 |
|---|---|---|
| - | - | - |
| 44 | 131 | EDI |

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 19355 |
| t4 | John | 44 | 131 | CHI | EH8 9LE |

# Our data cleaning framework

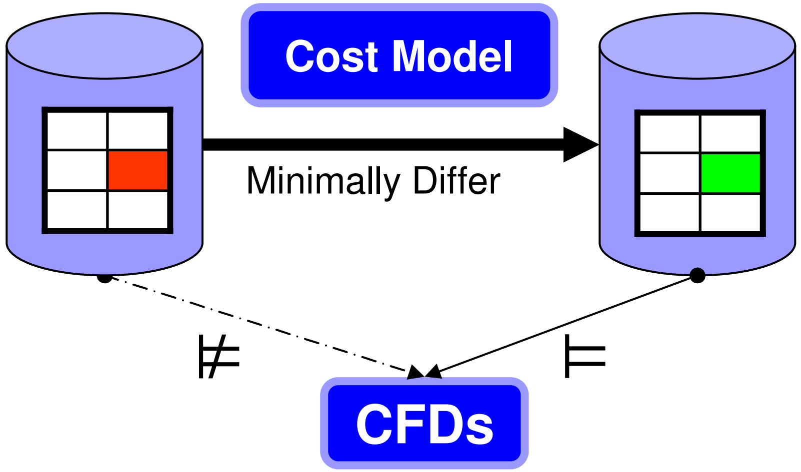



optional module

# Automatically find a repair

Input: a relational database DB, and a set Σ of CFDs
Output: a repair DB' of DB such that cost(DB', DB) is minimal

- repair: DB' ⊨ Σ
- "good": cost(DB', DB)
  - DB' is "close" to the original data in DB
  - Minimizing changes to "accurate" attributes




Complexity:
It is known that finding an optimal repair is NP-complete even for a fixed set of FDs. *It remains intractable for CFDs.*

**Find effective heuristics for repairing databases based on CFDs.**

# Equivalence Class

**[ CC, AC ]  →  [ City ]**

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 60132 |
| t4 | John | 1 | 312 | CHI | 60132 |

# Equivalence Class

**[ CC, AC ] → [ City ]**

E1

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 60132 |
| t4 | John | 1 | 312 | CHI | 60132 |

# Equivalence Class

**[ CC, AC ] → [ City ]**

E1

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 60132 |
| t4 | John | 1 | 312 | CHI | 60132 |

E2

- Separate
  - The decision of **which attribute values** need to be equivalent
  - The decision of exactly **what value** an EC should be assigned
- Avoid **poor local decisions**

# Merge equivalence classes

**[ CC, AC ] → [ City ]** **[ ZIP ] → [ City ]**

E1

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 60132 |
| t4 | John | 1 | 312 | CHI | 60132 |

E2

# Merge equivalence classes

**[ CC, AC ] → [ City ]** **[ ZIP ] → [ City ]**

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 60132 |
| t4 | John | 1 | 312 | CHI | 60132 |

**E1**

**E2**

**E3 = E1 ∪ E2**

# Merge equivalence classes

**[ CC, AC ] → [ City ]** **[ ZIP ]→ [ City ]**

**E3**

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 60132 |
| t4 | John | 1 | 312 | CHI | 60132 |

**E3 = E1 ∪ E2**

# FDs → CFDs: does it work?

| 【 CC , | AC 】 → | 【 City 】 |
|---|---|---|
| 1 | 215 | PHI |

| 【 ZIP 】 → | 【 City 】 |
|---|---|
| 60132 | CHI |

**E3: PHI**

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 60132 |
| t4 | John | 1 | 312 | CHI | 60132 |

# FDs → CFDs: does it work?

| 【CC , | AC】 | → 【City】 |
|---|---|---|
| 1 | 215 | PHI |

| 【ZIP】 | → 【City】 |
|---|---|
| 60132 | CHI |

E3: CHI

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 60132 |
| t4 | John | 1 | 312 | CHI | 60132 |

# FDs → CFDs: it doesn't work

| 【 CC , | AC 】 | → 【 City 】 |
|---|---|---|
| 1 | 215 | PHI |

| 【 ZIP 】 | → 【 City 】 |
|---|---|
| 60132 | CHI |

**E3: PHI**

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 60132 |
| t4 | John | 1 | 312 | CHI | 60132 |

**FD repair alg. doesn't even terminate for CFD!**

# CFD repair

- To resolve CFD violations, we allow
  - merge ECs
  - **upgrade EC** (different from repairing FD)
- Change both
  - RHS attributes
  - and **LHS attributes** (different from repairing FD)
    - We do not "**invent**" values: choose value from active domain
    - If there is no suitable value from active domain, put "null"
- Guarantees **termination** and **correctness** (DB' satisfies all constraints)

# Cost Model: weight and distance

Cost(u,v) = **weight**(t, A) * **distance**(u,v) / max(|u|,|v|)

- Based on both
  - **weight**: estimate the accuracy of the attributes values to be modified
    - Could be obtained by data provenance …
  - and **distance**: measure the “closeness” of the new value to the original one
- Intuitively
  - the more accurate the original value is
    - the less reasonable to change the value
  - the more distant the new value is from the original one
    - the less reasonable of this change
- As will be seen soon
  - although the cost model **incorporate** the weight information, the cleaning algorithm **also works** in the absence of it

# CFD: upgrade equivalence classes

Target value of equivalence class E

targ(E) = **not fixed ⇨ fixed** : upgrade

E1: PHI Fixed

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 60132 |
| t4 | John | 1 | 312 | CHI | 60132 |

| 【 CC , | AC 】 → | 【 City 】 |
|---|---|---|
| 1 | 215 | PHI |
| - | - | - |

| 【 ZIP 】 → | 【 City 】 |
|---|---|
| 60132 | CHI |

E2 Not Fixed

# Change LHS attribute

| 【 CC , | AC 】 → | 【 City 】 |
| --- | --- | --- |
| 1 | 215 | PHI |
| - | - | - |

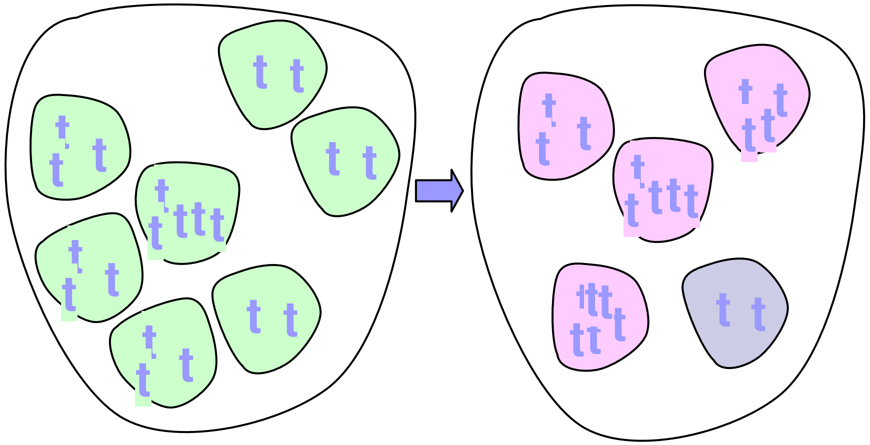
| 【 ZIP 】 → | 【 City 】 |
| --- | --- |
| 60132 | CHI |

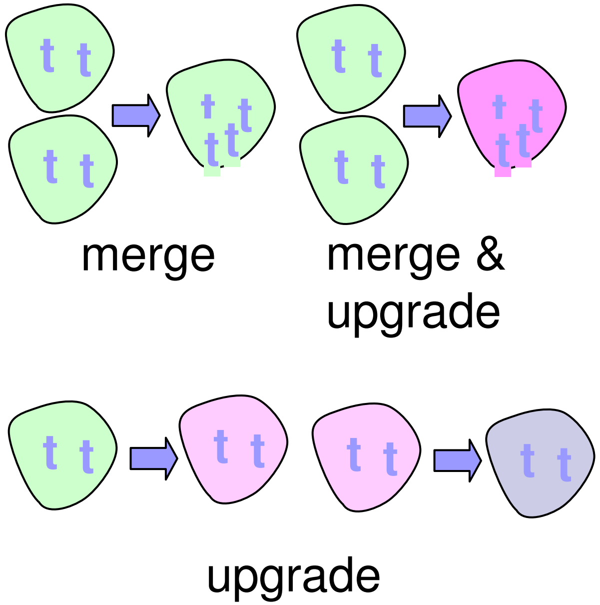
**E1: PHI Fixed**

| | Name | CC | AC | City | ZIP |
| --- | --- | --- | --- | --- | --- |
| t1 | Ben | 1 | 215 | PHI | 19132 |
| t2 | Joe | 1 | 215 | PHI | 19132 |
| t3 | Paul | 1 | 215 | PHI | 60132 |
| t4 | John | 1 | 312 | CHI | 60132 |

**E2 Not Fixed**

# Resolving CFD violations

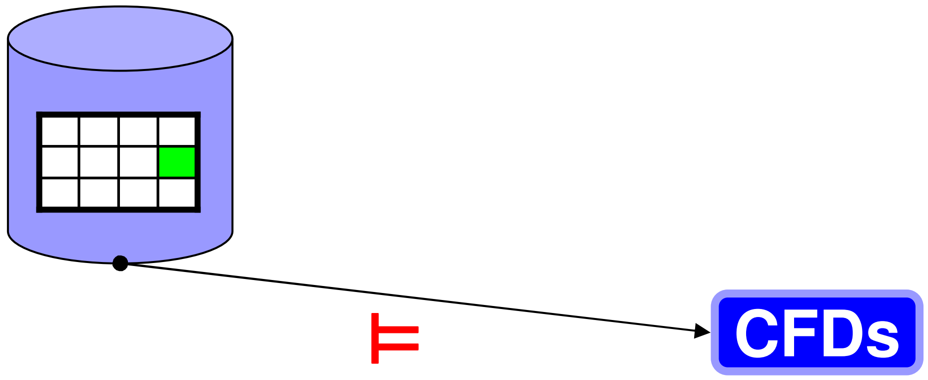





- Terminate
  - Each step
    - Either the number of **original ECs** is reduced
    - Or the number of **upgraded ECs** is increased
  - There are bounds for the number of **ECs** and **upgraded ECs**
- Correct
  - the output database is guaranteed to satisfy the CFDs

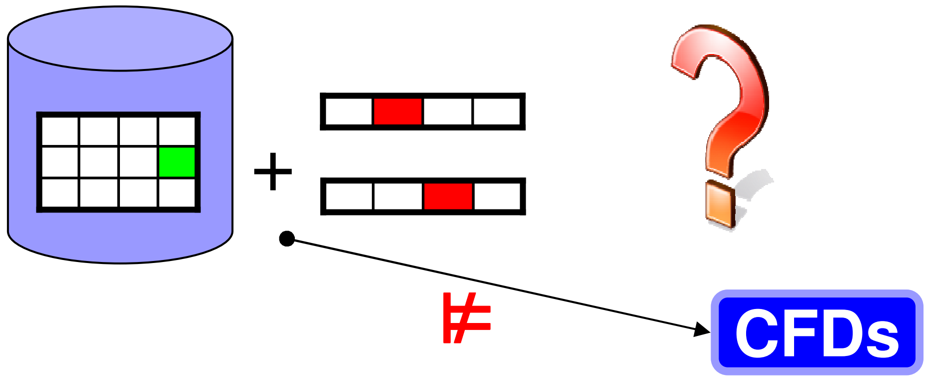
# Incremental repair

Now we have obtained a clean database:



# Incremental repair

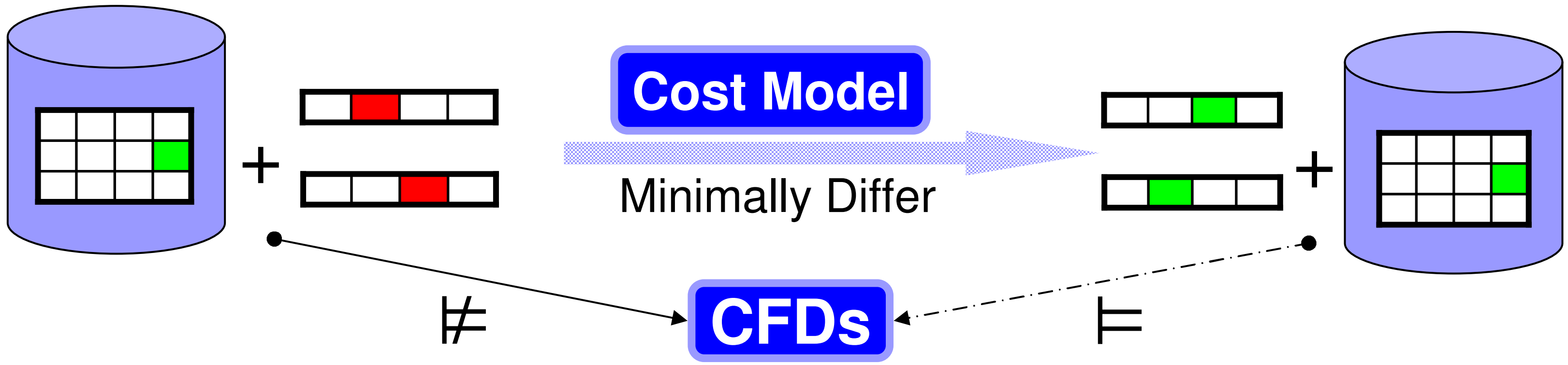
When the cleaned database is updated …



# Incremental repair

Input: a clean database DB, changes $\Delta DB$ to DB, and a set $\Sigma$ of CFDs

Output: a repair DB' of DB + $\Delta DB$

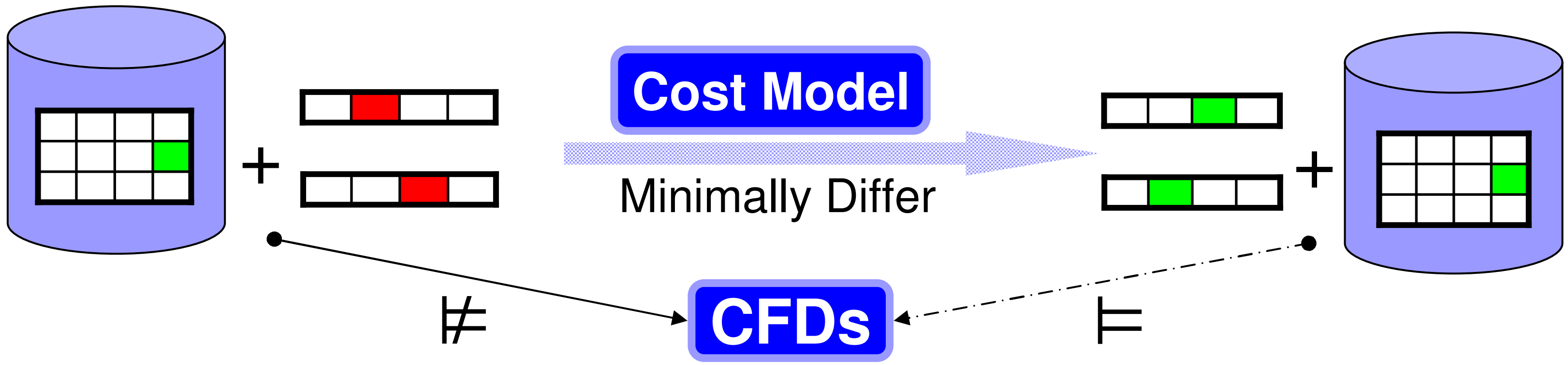



One might think that the incremental repairing problem is simpler than its batch (non-incremental) counterpart …

# Incremental repair

Input: a clean database DB, changes *ΔDB* to DB,
and a set Σ of CFDs

Output: a repair DB' of DB + *ΔDB*




*Complexity. The local data cleaning problem is also NP-complete, even if ΔDB consists of a single tuple.*

Find effective heuristic algorithms for incrementally repairing databases based on CFDs.

# Repair a tuple: local repair

| 【 CC , | AC 】 | → 【 City 】 |
|---|---|---|
| - | - | - |

| 【 ZIP 】 | → 【 City 】 |
|---|---|
| 10112 | NYC |

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Mark | 1 | 215 | PHI | 19112 |
| t2 | Peter | 44 | 131 | EDI | EH8 9LE |

| | | | | | |
|---|---|---|---|---|---|
| t3 | Eric | 1 | 215 | CHI | 10112 |

Greedily finds the "best" set of attributes to modify in order to create a repair.

# Repair a tuple: local repair

| 【 CC , | AC 】 | → City |
|---|---|---|
| - | - | - |

| 【 ZIP 】 | → City |
|---|---|
| 10112 | NYC |

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Mark | 1 | 215 | PHI | 19112 |
| t2 | Peter | 44 | 131 | EDI | EH8 9LE |
| t3 | Eric | 1 | 215 | CHI | 10112 |

Since one attribute is not enough to fix this violation, we consider two attributes …

# Repair a tuple: local repair

| 【 CC , | AC 】 | → City |
|---|---|---|
| - | - | - |

| 【 ZIP 】 | → City |
|---|---|
| 10112 | NYC |

| | Name | CC | AC | City | ZIP |
|---|---|---|---|---|---|
| t1 | Mark | 1 | 215 | PHI | 19112 |
| t2 | Peter | 44 | 131 | EDI | EH8 9LE |
| t3 | Eric | 1 | 215 | PHI | 19112 |

Techniques to reduce the search space and using index to optimize this process

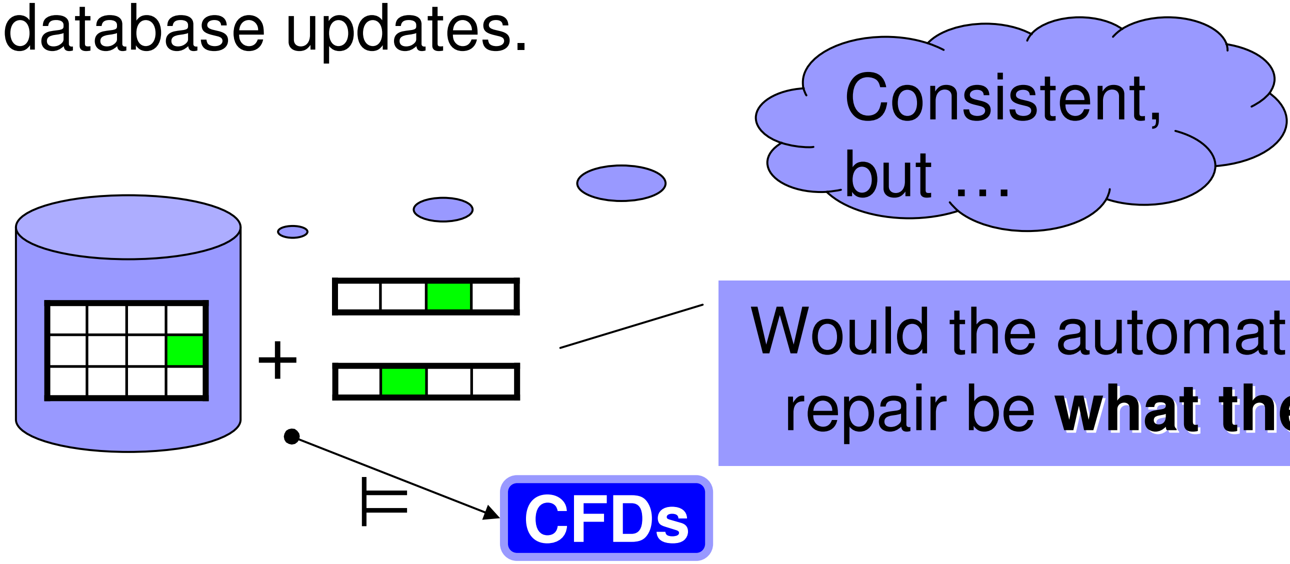
# Repair a group of tuples: ordering

- The order of the tuples to repair
  - has no impact on the **termination**
  - impact repairing **accuracy** and **performance**
- Orders used
  - linear-scan: bad
    - L-IncRepair
  - based on weights: good
    - W-IncRepair: repair tuples with **more weights** first
  - based on violations: good
    - V-IncRepair: repair tuples with **less violations** first
    - **Independent of weights**

# Consistent, but accurate?

We can **automatically** find a repair.

We can also **incrementally** find a repair in response to database updates.




Would the automatically generated repair be **what the user wants**?

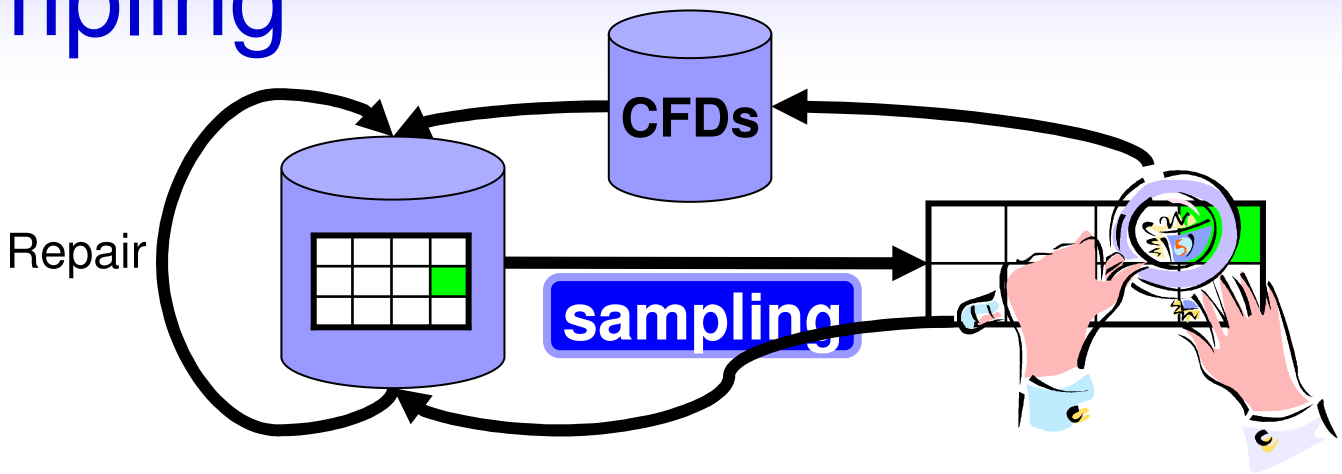
To meet the **expectation** of the user

it is better to involve domain experts to inspect the repairs.

# Assess accuracy of repairs

- However, it is **not realistic** to manually inspect each editing when dealing with large dataset
- How to ensure that the repairs are accurate enough **without excessive user interaction**?
  - A statistical method to guarantee the accuracy of the repairs are above a predefined bound with a high confidence.

# Sampling

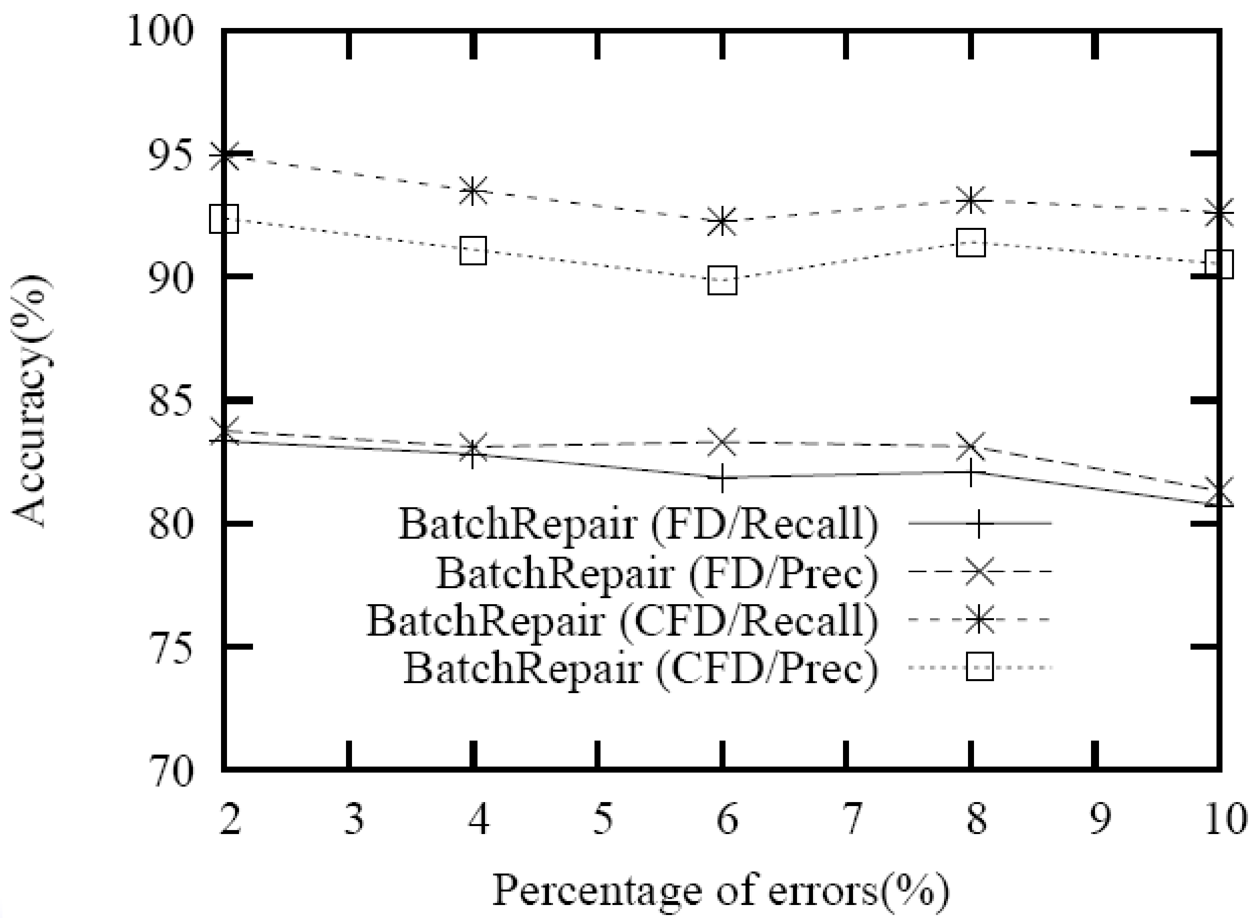



- Involve the user to
  - inspect small samples
  - edit both the **sample data** and **input CFDs** if necessary
  - invoke **automated repairing methods** to revise repairs
- Stratified sampling method
  - give priority to strata that are more likely to be inaccurate
  - ensure the **accuracy** of the repairs are above a predefined bound with a high confidence.

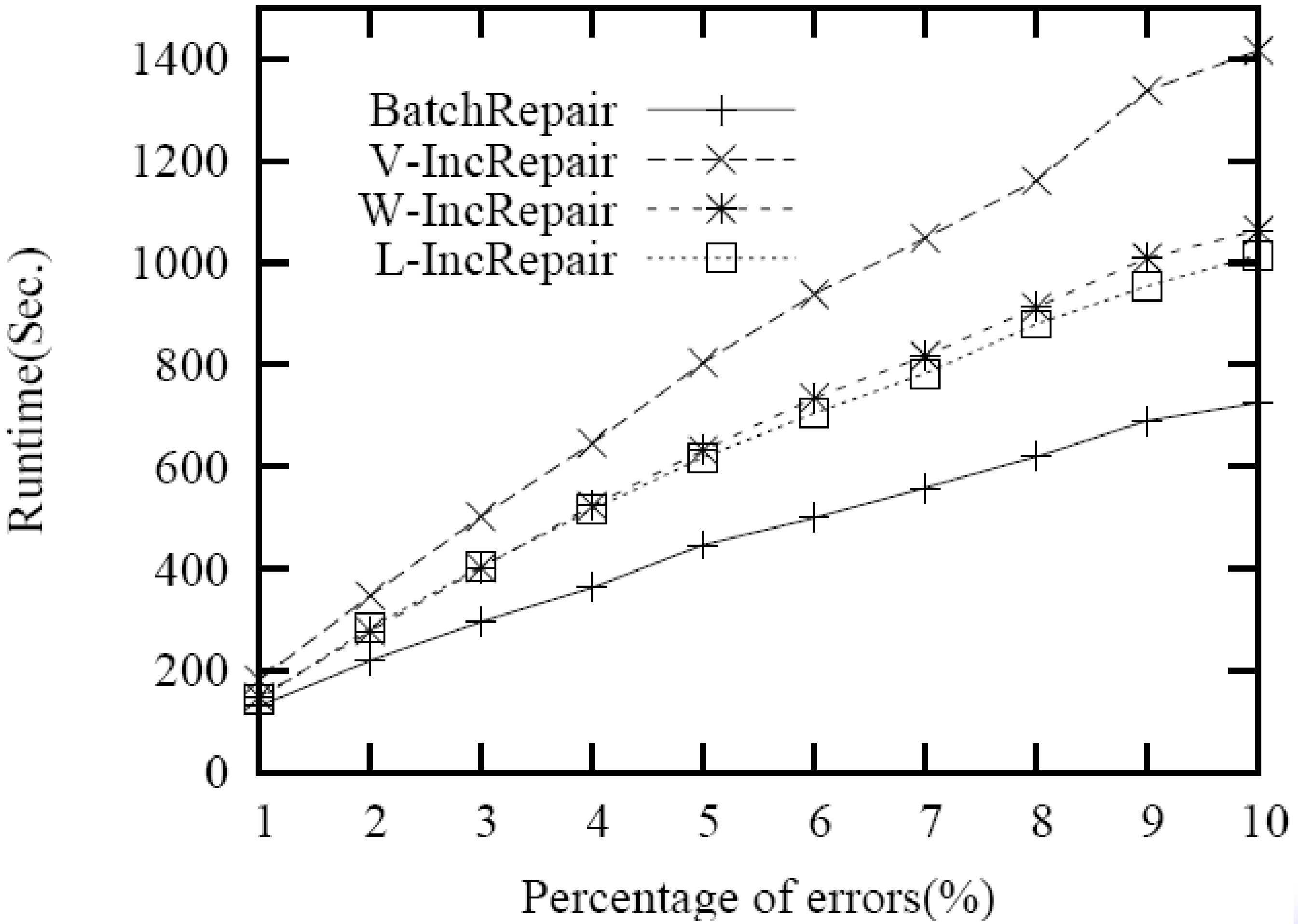
# Experimental setting

- Prototype system
  - **Con$^2$Clean** (in Java)
- Data
  - we scraped real-life data from web
  - Generate datasets of various sizes, 10k to 300k tuples
- Constraints
  - Fairly large since each pattern tuple is in fact a constraint
    - 7 CFDs
    - 300---5,000 pattern tuples for each of these CFDs
- Clean data
  - Initial datasets are “correct” data, consistent with all CFDs
- Dirty data: error rate 1% to 10%
  - Randomly add noise to an attribute
    - New value close to the original one
    - Or an arbitrary existing value taken from another tuple

# Accuracy of CFDs vs FDs



# Scalability over Noise Rate



# Conclusion and future work

- A framework for improving data quality: both consistency and accuracy
  - **Automatic** part: guarantee termination and correctness
    - Batch repair
    - Incremental repair: **optional**
  - **Semi-automatic** part
    - Statistical methods: **optional**
      - Guarantee accuracy above a predefined bound **without excessive user interaction**
- Future
  - Automated methods for discovering CFDs
  - Repair algorithms for other conditional constraints

**A data cleaning framework using constraints specially designed for improving data quality.**