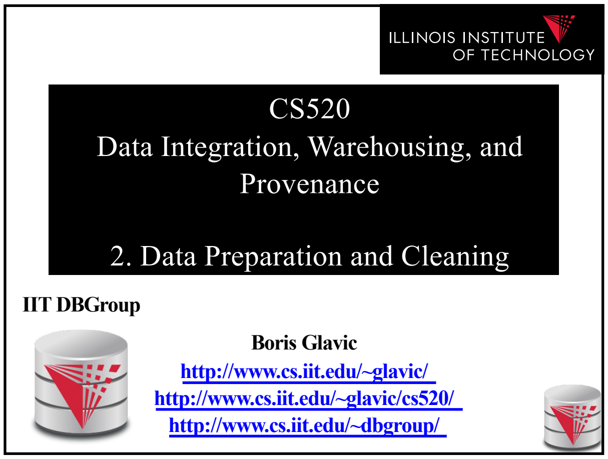


ILLINOIS INSTITUTE OF TECHNOLOGY

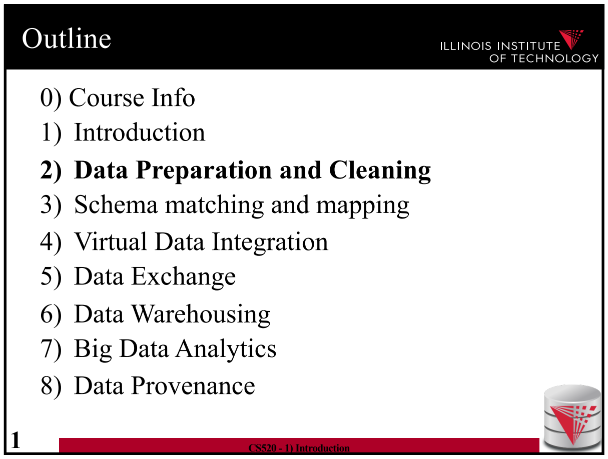
CS520
Data Integration, Warehousing, and Provenance

2. Data Preparation and Cleaning

IIT DBGroup

Boris Glavic
http://www.cs.iit.edu/~glavic/
http://www.cs.iit.edu/~glavic/cs520/
http://www.cs.iit.edu/~dbgroup/

0

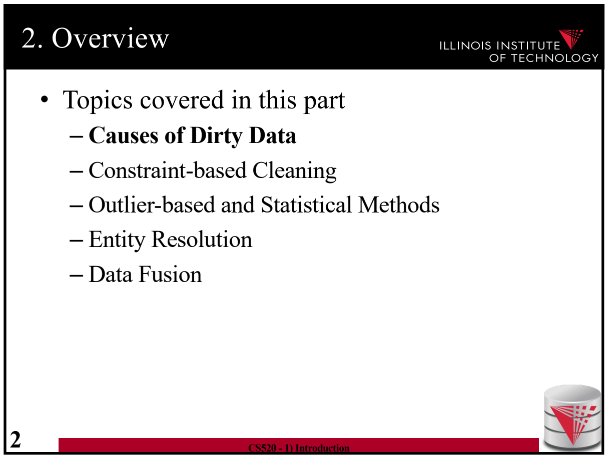


Outline

0) Course Info
1) Introduction
2) Data Preparation and Cleaning
3) Schema matching and mapping
4) Virtual Data Integration
5) Data Exchange
6) Data Warehousing
7) Big Data Analytics
8) Data Provenance

1 CS520 - 1) Introduction

1

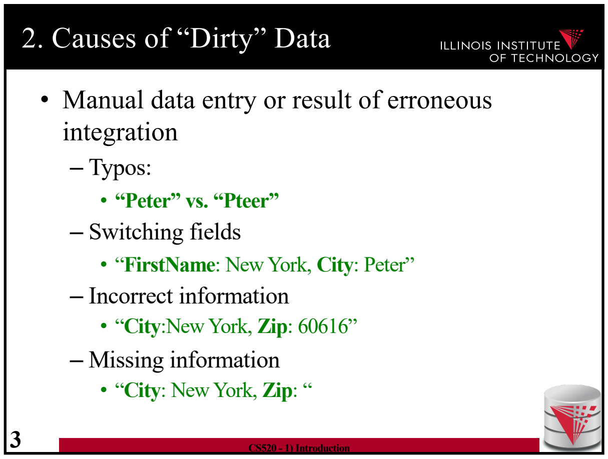


2. Overview

- Topics covered in this part
  - **Causes of Dirty Data**
  - Constraint-based Cleaning
  - Outlier-based and Statistical Methods
  - Entity Resolution
  - Data Fusion

2 CS520 - 1) Introduction

2

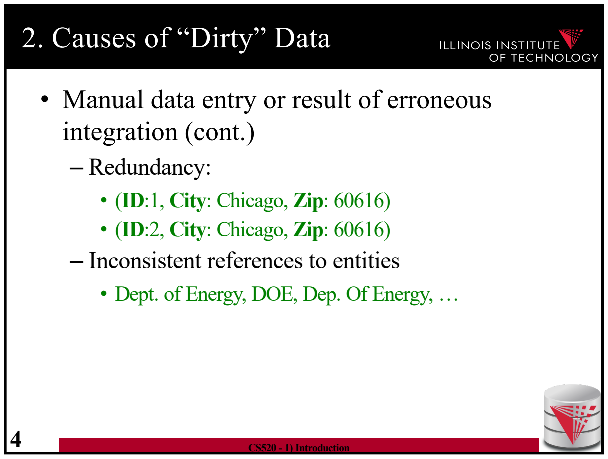


2. Causes of “Dirty” Data

- Manual data entry or result of erroneous integration
  - Typos:
    - **“Peter” vs. “Pteer”**
  - Switching fields
    - “**FirstName**: New York, **City**: Peter”
  - Incorrect information
    - “**City**:New York, **Zip**: 60616”
  - Missing information
    - “**City**: New York, **Zip**: “

3 CS520 - 1) Introduction

3

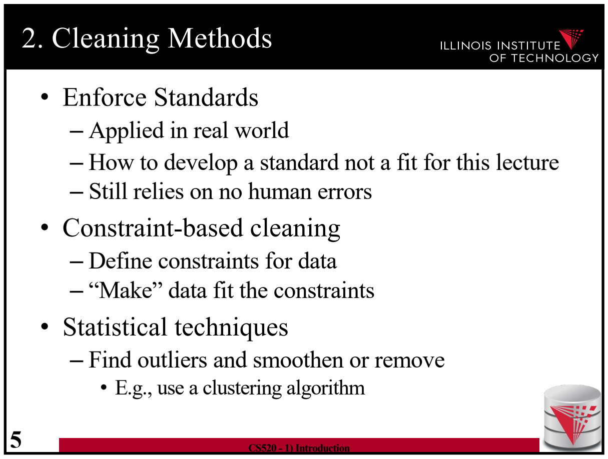


2. Causes of “Dirty” Data

- Manual data entry or result of erroneous integration (cont.)
  - Redundancy:
    - (**ID**:1, **City**: Chicago, **Zip**: 60616)
    - (**ID**:2, **City**: Chicago, **Zip**: 60616)
  - Inconsistent references to entities
    - Dept. of Energy, DOE, Dep. Of Energy, …

4 CS520 - 1) Introduction

4

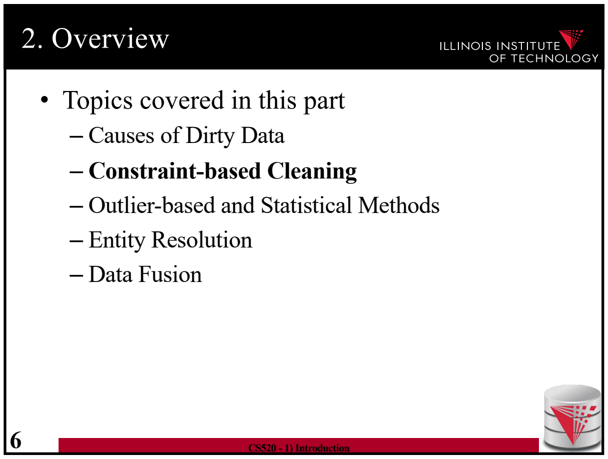


2. Cleaning Methods

- Enforce Standards
  - Applied in real world
  - How to develop a standard not a fit for this lecture
  - Still relies on no human errors
- Constraint-based cleaning
  - Define constraints for data
  - “Make” data fit the constraints
- Statistical techniques
  - Find outliers and smoothen or remove
    - E.g., use a clustering algorithm

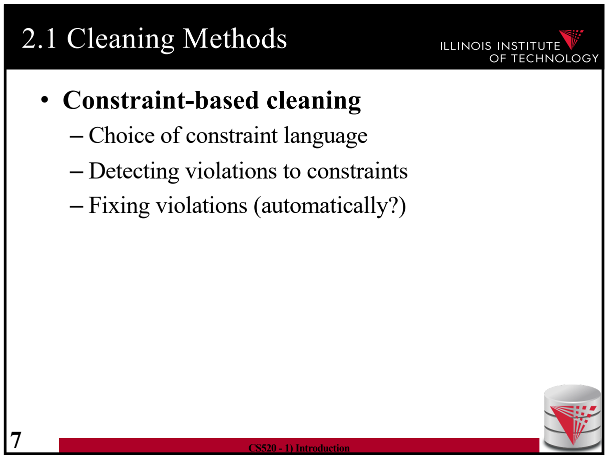
5 CS520 - 1) Introduction

5



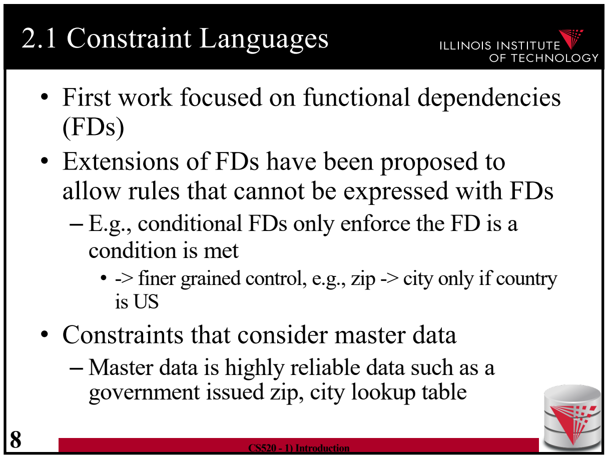
2. Overview

- Topics covered in this part
  - Causes of Dirty Data
  - **Constraint-based Cleaning**
  - Outlier-based and Statistical Methods
  - Entity Resolution
  - Data Fusion



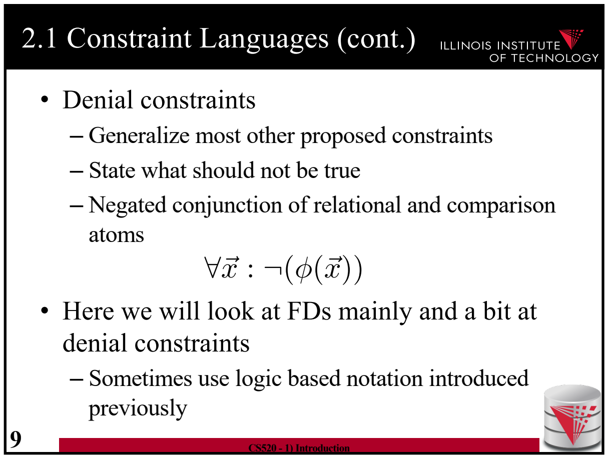
2.1 Cleaning Methods

- **Constraint-based cleaning**
  - Choice of constraint language
  - Detecting violations to constraints
  - Fixing violations (automatically?)



2.1 Constraint Languages

- First work focused on functional dependencies (FDs)
- Extensions of FDs have been proposed to allow rules that cannot be expressed with FDs
  - E.g., conditional FDs only enforce the FD is a condition is met
    - -> finer grained control, e.g., zip -> city only if country is US
- Constraints that consider master data
  - Master data is highly reliable data such as a government issued zip, city lookup table

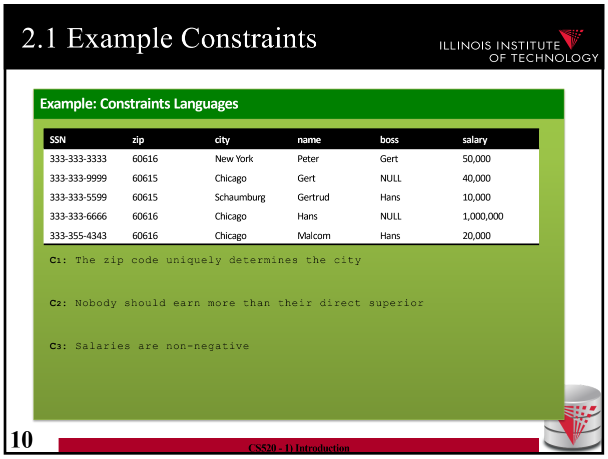


2.1 Constraint Languages (cont.)

- Denial constraints
  - Generalize most other proposed constraints
  - State what should not be true
  - Negated conjunction of relational and comparison atoms

$$\forall \vec{x} : \neg(\phi(\vec{x}))$$

- Here we will look at FDs mainly and a bit at denial constraints
  - Sometimes use logic based notation introduced previously

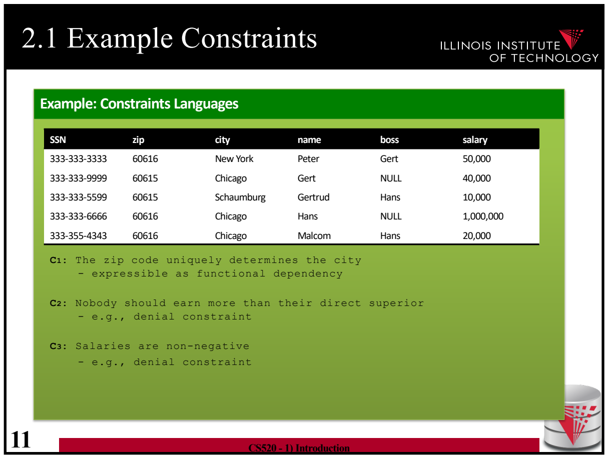


2.1 Example Constraints

Example: Constraints Languages

SSN	zip	city	name	boss	salary
333-333-3333	60616	New York	Peter	Gert	50,000
333-333-9999	60615	Chicago	Gert	NULL	40,000
333-333-5599	60615	Schaumburg	Gertrud	Hans	10,000
333-333-6666	60616	Chicago	Hans	NULL	1,000,000
333-355-4343	60616	Chicago	Malcom	Hans	20,000

```
C1: The zip code uniquely determines the city

C2: Nobody should earn more than their direct superior

C3: Salaries are non-negative
```

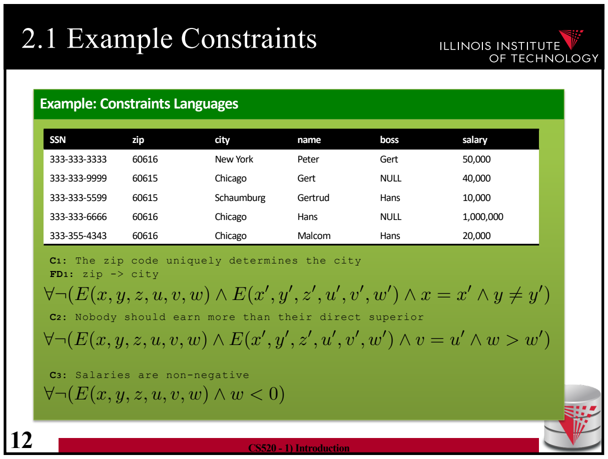


2.1 Example Constraints

Example: Constraints Languages

SSN	zip	city	name	boss	salary
333-333-3333	60616	New York	Peter	Gert	50,000
333-333-9999	60615	Chicago	Gert	NULL	40,000
333-333-5599	60615	Schaumburg	Gertrud	Hans	10,000
333-333-6666	60616	Chicago	Hans	NULL	1,000,000
333-355-4343	60616	Chicago	Malcom	Hans	20,000

```
C1: The zip code uniquely determines the city
    - expressible as functional dependency

C2: Nobody should earn more than their direct superior
    - e.g., denial constraint

C3: Salaries are non-negative
    - e.g., denial constraint
```



2.1 Example Constraints

ILLINOIS INSTITUTE OF TECHNOLOGY

Example: Constraints Languages

SSN	zip	city	name	boss	salary
333-333-3333	60616	New York	Peter	Gert	50,000
333-333-9999	60615	Chicago	Gert	NULL	40,000
333-333-5599	60615	Schaumburg	Gertrud	Hans	10,000
333-333-6666	60616	Chicago	Hans	NULL	1,000,000
333-355-4343	60616	Chicago	Malcom	Hans	20,000

C1: The zip code uniquely determines the city

FD1: zip -> city

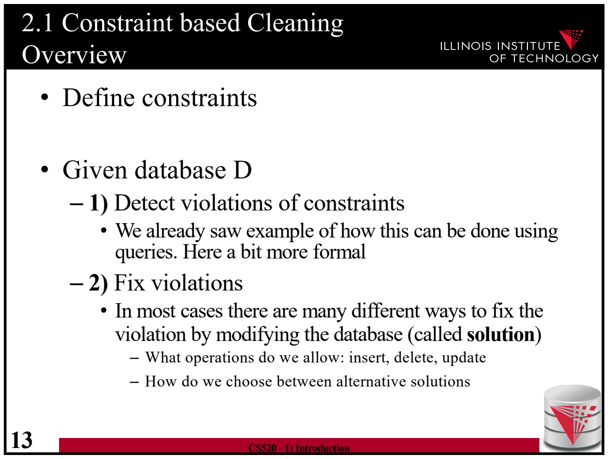
$$\forall \neg (E(x,y,z,u,v,w) \wedge E(x',y',z',u',v',w') \wedge x = x' \wedge y \neq y')$$

C2: Nobody should earn more than their direct superior

$$\forall \neg (E(x,y,z,u,v,w) \wedge E(x',y',z',u',v',w') \wedge v = u' \wedge w > w')$$

C3: Salaries are non-negative

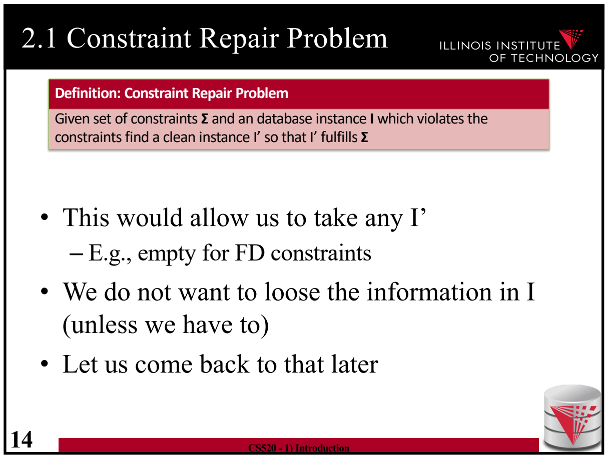
$$\forall \neg (E(x,y,z,u,v,w) \wedge w < 0)$$



2.1 Constraint based Cleaning Overview

ILLINOIS INSTITUTE OF TECHNOLOGY

- Define constraints
- Given database D
  - **1)** Detect violations of constraints
    - We already saw example of how this can be done using queries. Here a bit more formal
  - **2)** Fix violations
    - In most cases there are many different ways to fix the violation by modifying the database (called **solution**)
      - What operations do we allow: insert, delete, update
      - How do we choose between alternative solutions



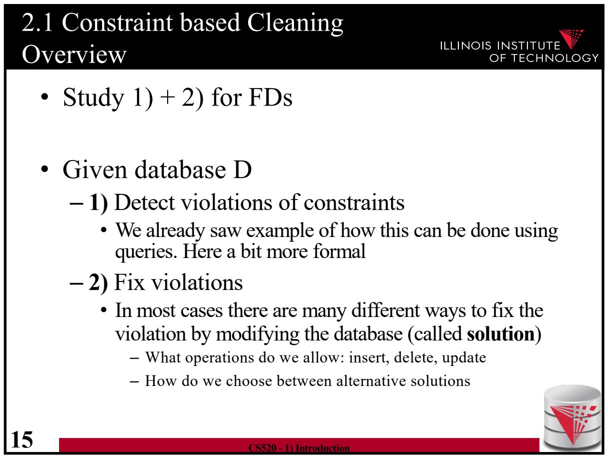
2.1 Constraint Repair Problem

ILLINOIS INSTITUTE OF TECHNOLOGY

Definition: Constraint Repair Problem

Given set of constraints **Σ** and an database instance **I** which violates the constraints find a clean instance I' so that I' fulfills **Σ**

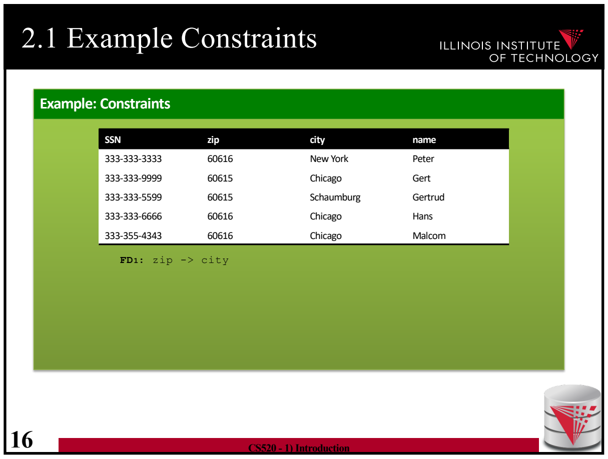
- This would allow us to take any I'
  - E.g., empty for FD constraints
- We do not want to loose the information in I (unless we have to)
- Let us come back to that later



2.1 Constraint based Cleaning Overview

ILLINOIS INSTITUTE OF TECHNOLOGY

- Study 1) + 2) for FDs
- Given database D
  - **1)** Detect violations of constraints
    - We already saw example of how this can be done using queries. Here a bit more formal
  - **2)** Fix violations
    - In most cases there are many different ways to fix the violation by modifying the database (called **solution**)
      - What operations do we allow: insert, delete, update
      - How do we choose between alternative solutions



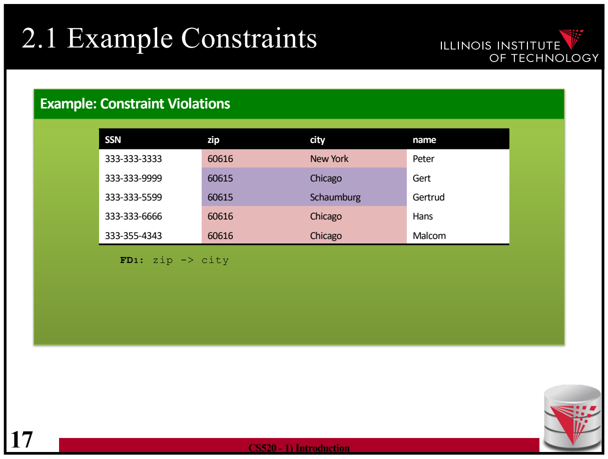
2.1 Example Constraints

ILLINOIS INSTITUTE OF TECHNOLOGY

Example: Constraints

SSN	zip	city	name
333-333-3333	60616	New York	Peter
333-333-9999	60615	Chicago	Gert
333-333-5599	60615	Schaumburg	Gertrud
333-333-6666	60616	Chicago	Hans
333-355-4343	60616	Chicago	Malcom

FD1: zip -> city



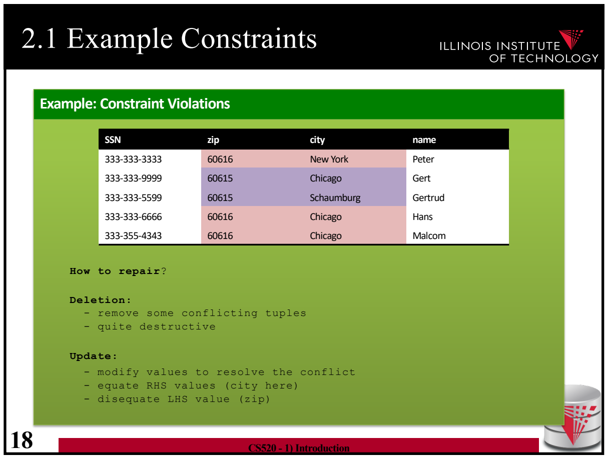
2.1 Example Constraints

ILLINOIS INSTITUTE OF TECHNOLOGY

Example: Constraint Violations

SSN	zip	city	name
333-333-3333	60616	New York	Peter
333-333-9999	60615	Chicago	Gert
333-333-5599	60615	Schaumburg	Gertrud
333-333-6666	60616	Chicago	Hans
333-355-4343	60616	Chicago	Malcom

FD1: zip -> city



2.1 Example Constraints

ILLINOIS INSTITUTE OF TECHNOLOGY

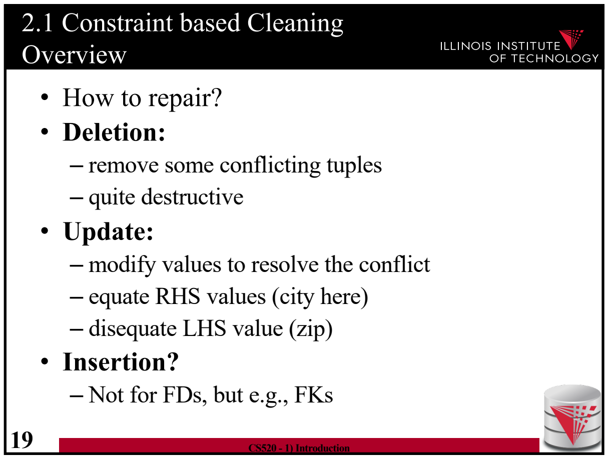
Example: Constraint Violations

SSN	zip	city	name
333-333-3333	60616	New York	Peter
333-333-9999	60615	Chicago	Gert
333-333-5599	60615	Schaumburg	Gertrud
333-333-6666	60616	Chicago	Hans
333-355-4343	60616	Chicago	Malcom

```
How to repair?

Deletion:
  - remove some conflicting tuples
  - quite destructive

Update:
  - modify values to resolve the conflict
  - equate RHS values (city here)
  - disequate LHS value (zip)
```

18 CS520 - 1) Introduction

18



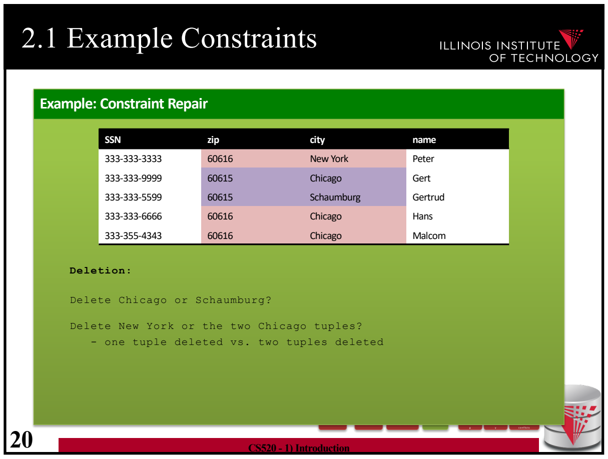
2.1 Constraint based Cleaning Overview

ILLINOIS INSTITUTE OF TECHNOLOGY

- How to repair?
- **Deletion:**
  - remove some conflicting tuples
  - quite destructive
- **Update:**
  - modify values to resolve the conflict
  - equate RHS values (city here)
  - disequate LHS value (zip)
- **Insertion?**
  - Not for FDs, but e.g., FKs

19 CS520 - 1) Introduction

19



2.1 Example Constraints

ILLINOIS INSTITUTE OF TECHNOLOGY

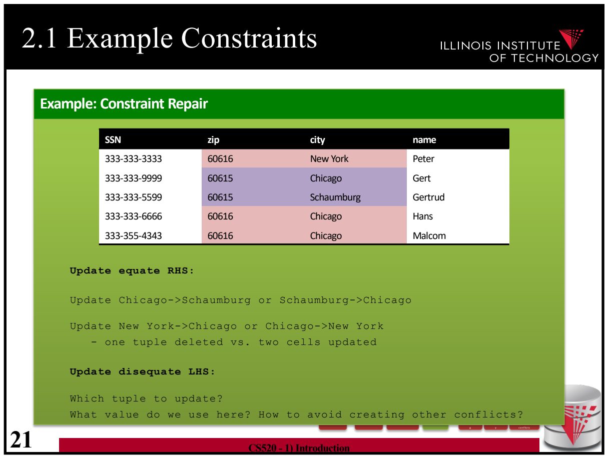
Example: Constraint Repair

SSN	zip	city	name
333-333-3333	60616	New York	Peter
333-333-9999	60615	Chicago	Gert
333-333-5599	60615	Schaumburg	Gertrud
333-333-6666	60616	Chicago	Hans
333-355-4343	60616	Chicago	Malcom

```
Deletion:

Delete Chicago or Schaumburg?

Delete New York or the two Chicago tuples?
   - one tuple deleted vs. two tuples deleted
```

20 CS520 - 1) Introduction

20



2.1 Example Constraints

ILLINOIS INSTITUTE OF TECHNOLOGY

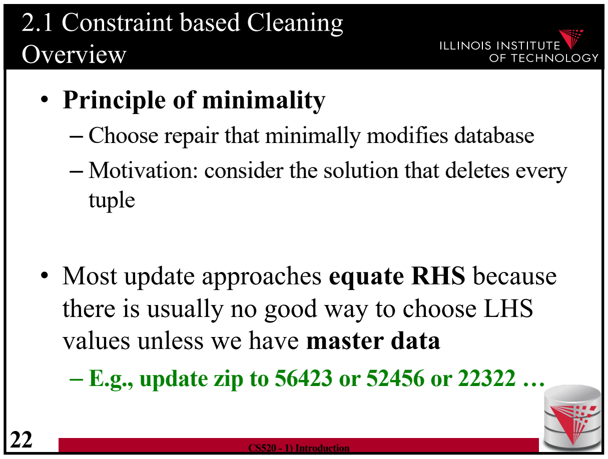
Example: Constraint Repair

SSN	zip	city	name
333-333-3333	60616	New York	Peter
333-333-9999	60615	Chicago	Gert
333-333-5599	60615	Schaumburg	Gertrud
333-333-6666	60616	Chicago	Hans
333-355-4343	60616	Chicago	Malcom

```
Update equate RHS:

Update Chicago->Schaumburg or Schaumburg->Chicago

Update New York->Chicago or Chicago->New York
   - one tuple deleted vs. two cells updated

Update disequate LHS:

Which tuple to update?
What value do we use here? How to avoid creating other conflicts?
```

21 CS520 - 1) Introduction

21



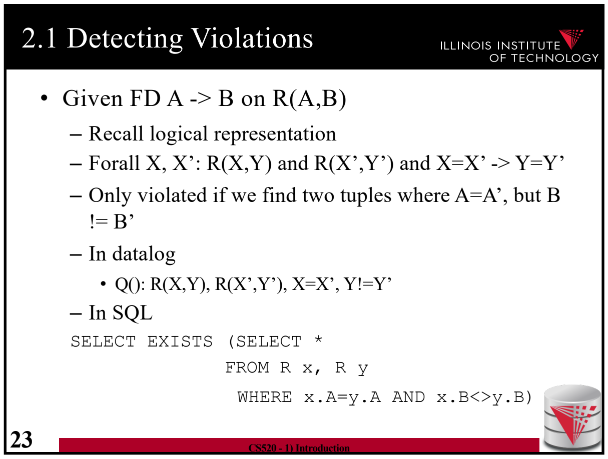
2.1 Constraint based Cleaning Overview

ILLINOIS INSTITUTE OF TECHNOLOGY

- **Principle of minimality**
  - Choose repair that minimally modifies database
  - Motivation: consider the solution that deletes every tuple
- Most update approaches **equate RHS** because there is usually no good way to choose LHS values unless we have **master data**
  - **E.g., update zip to 56423 or 52456 or 22322 …**

22 CS520 - 1) Introduction

22



2.1 Detecting Violations

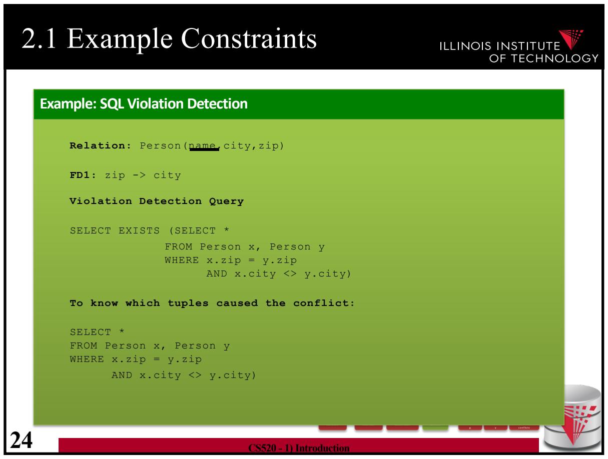
ILLINOIS INSTITUTE OF TECHNOLOGY

- Given FD A -> B on R(A,B)
  - Recall logical representation
  - Forall X, X': R(X,Y) and R(X',Y') and X=X' -> Y=Y'
  - Only violated if we find two tuples where A=A', but B != B'
  - In datalog
    - Q(): R(X,Y), R(X',Y'), X=X', Y!=Y'
  - In SQL

```
SELECT EXISTS (SELECT *
                FROM R x, R y
                 WHERE x.A=y.A AND x.B<>y.B)
```

23 CS520 - 1) Introduction

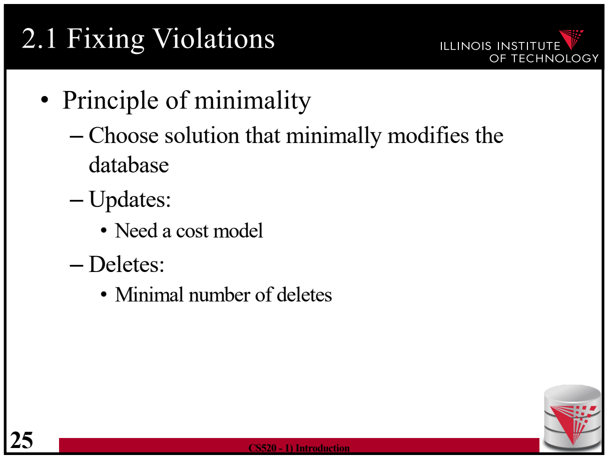
23



2.1 Example Constraints

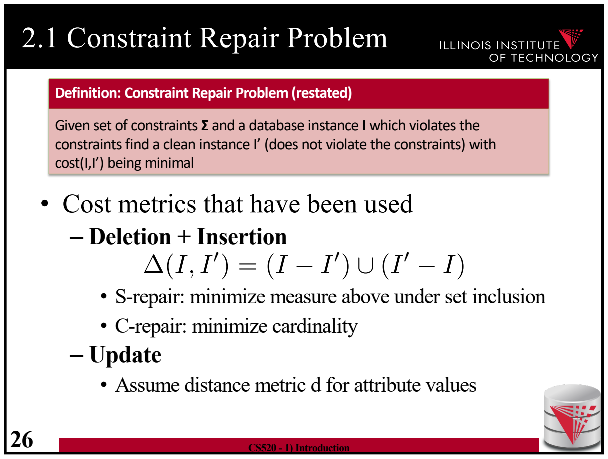
Example: SQL Violation Detection

```
Relation: Person(name,city,zip)

FD1: zip -> city

Violation Detection Query

SELECT EXISTS (SELECT *
              FROM Person x, Person y
              WHERE x.zip = y.zip
                    AND x.city <> y.city)

To know which tuples caused the conflict:

SELECT *
FROM Person x, Person y
WHERE x.zip = y.zip
      AND x.city <> y.city)
```



2.1 Fixing Violations

- Principle of minimality
  - Choose solution that minimally modifies the database
  - Updates:
    - Need a cost model
  - Deletes:
    - Minimal number of deletes



2.1 Constraint Repair Problem

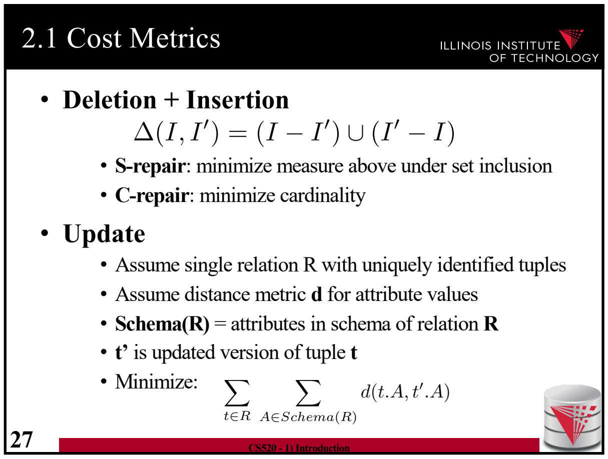
Definition: Constraint Repair Problem (restated)

Given set of constraints **Σ** and a database instance **I** which violates the constraints find a clean instance I' (does not violate the constraints) with cost(I,I') being minimal

- Cost metrics that have been used
  - **Deletion + Insertion**

$$\Delta(I, I') = (I - I') \cup (I' - I)$$

    - S-repair: minimize measure above under set inclusion
    - C-repair: minimize cardinality
  - **Update**
    - Assume distance metric d for attribute values

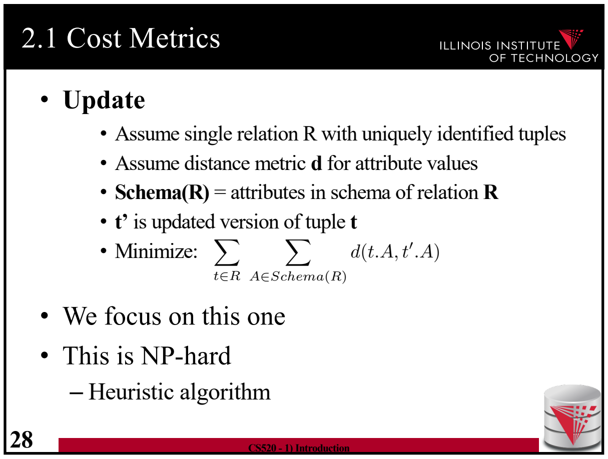


2.1 Cost Metrics

- **Deletion + Insertion**

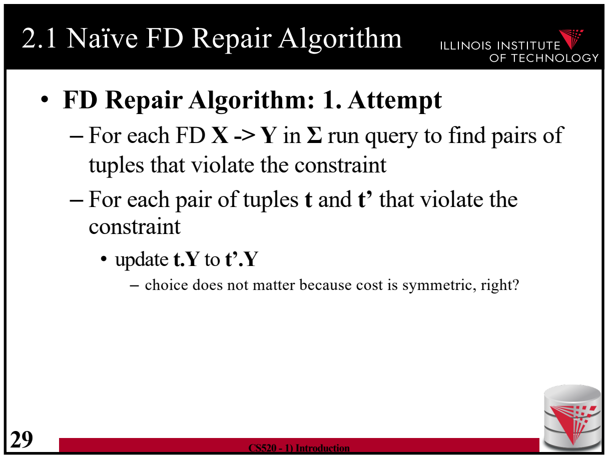
$$\Delta(I, I') = (I - I') \cup (I' - I)$$

  - **S-repair**: minimize measure above under set inclusion
  - **C-repair**: minimize cardinality
- **Update**
  - Assume single relation R with uniquely identified tuples
  - Assume distance metric **d** for attribute values
  - **Schema(R)** = attributes in schema of relation **R**
  - **t'** is updated version of tuple **t**
  - Minimize: $\sum_{t \in R} \sum_{A \in Schema(R)} d(t.A, t'.A)$



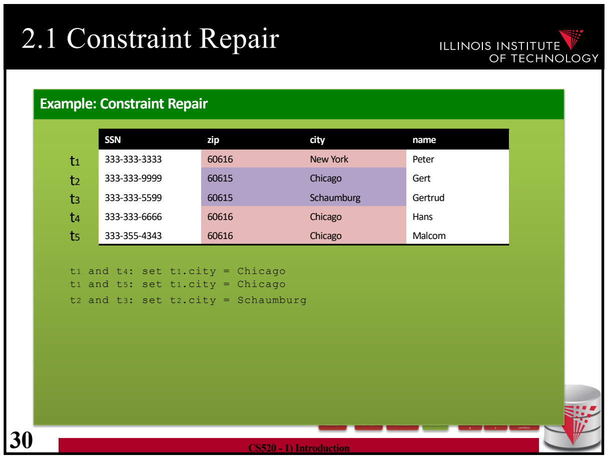
2.1 Cost Metrics

- **Update**
  - Assume single relation R with uniquely identified tuples
  - Assume distance metric **d** for attribute values
  - **Schema(R)** = attributes in schema of relation **R**
  - **t'** is updated version of tuple **t**
  - Minimize: $\sum_{t \in R} \sum_{A \in Schema(R)} d(t.A, t'.A)$
- We focus on this one
- This is NP-hard
  - Heuristic algorithm



2.1 Naïve FD Repair Algorithm

- **FD Repair Algorithm: 1. Attempt**
  - For each FD **X -> Y** in **Σ** run query to find pairs of tuples that violate the constraint
  - For each pair of tuples **t** and **t'** that violate the constraint
    - update **t.Y** to **t'.Y**
      - choice does not matter because cost is symmetric, right?

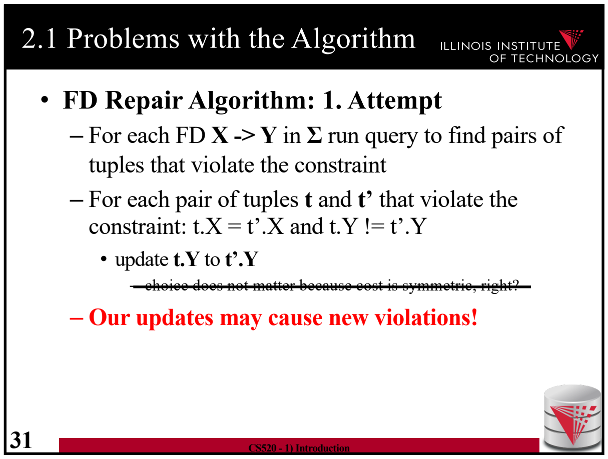


2.1 Constraint Repair

Example: Constraint Repair

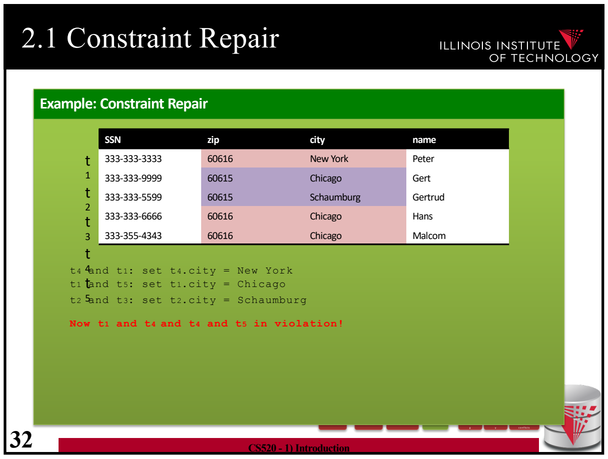
	SSN	zip	city	name
t_1	333-333-3333	60616	New York	Peter
t_2	333-333-9999	60615	Chicago	Gert
t_3	333-333-5599	60615	Schaumburg	Gertrud
t_4	333-333-6666	60616	Chicago	Hans
t_5	333-355-4343	60616	Chicago	Malcom

```
t1 and t4: set t1.city = Chicago
t1 and t5: set t1.city = Chicago
t2 and t3: set t2.city = Schaumburg
```



2.1 Problems with the Algorithm

- **FD Repair Algorithm: 1. Attempt**
  - For each FD **X -> Y** in **Σ** run query to find pairs of tuples that violate the constraint
  - For each pair of tuples **t** and **t'** that violate the constraint: t.X = t'.X and t.Y != t'.Y
    - update **t.Y** to **t'.Y**
    - ~~choice does not matter because cost is symmetric, right?~~
  - **Our updates may cause new violations!**

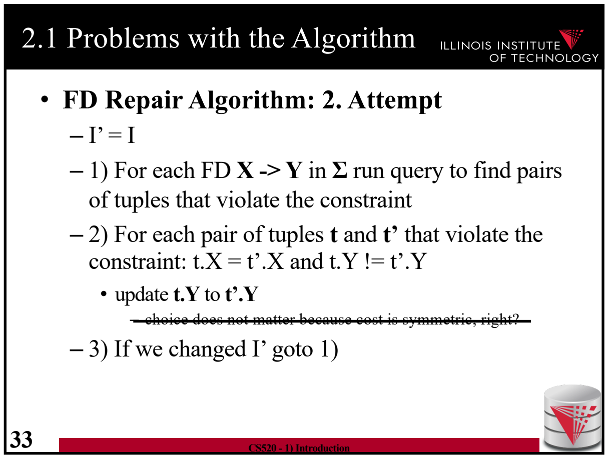


2.1 Constraint Repair

Example: Constraint Repair

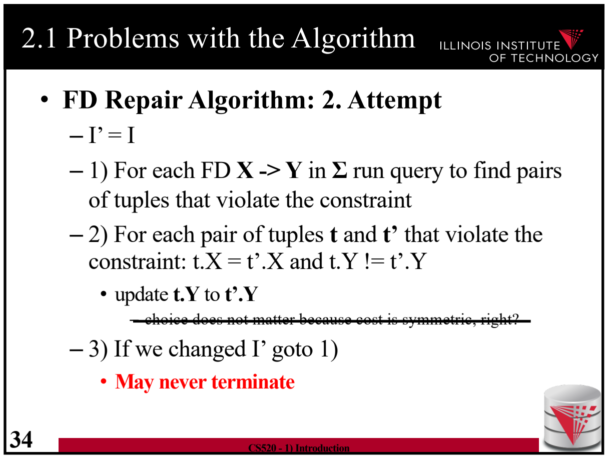
	SSN	zip	city	name
t_1	333-333-3333	60616	New York	Peter
t_2	333-333-9999	60615	Chicago	Gert
t_3	333-333-5599	60615	Schaumburg	Gertrud
t_4	333-333-6666	60616	Chicago	Hans
t_5	333-355-4343	60616	Chicago	Malcom

```
t4 and t1: set t4.city = New York
t1 and t5: set t1.city = Chicago
t2 and t3: set t2.city = Schaumburg

Now t1 and t4 and t4 and t5 in violation!
```



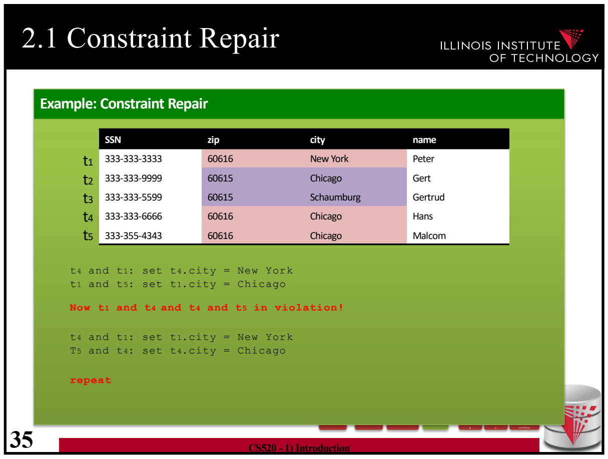
2.1 Problems with the Algorithm

- **FD Repair Algorithm: 2. Attempt**
  - I' = I
  - 1) For each FD **X -> Y** in **Σ** run query to find pairs of tuples that violate the constraint
  - 2) For each pair of tuples **t** and **t'** that violate the constraint: t.X = t'.X and t.Y != t'.Y
    - update **t.Y** to **t'.Y**
    - ~~choice does not matter because cost is symmetric, right?~~
  - 3) If we changed I' goto 1)



2.1 Problems with the Algorithm

- **FD Repair Algorithm: 2. Attempt**
  - I' = I
  - 1) For each FD **X -> Y** in **Σ** run query to find pairs of tuples that violate the constraint
  - 2) For each pair of tuples **t** and **t'** that violate the constraint: t.X = t'.X and t.Y != t'.Y
    - update **t.Y** to **t'.Y**
    - ~~choice does not matter because cost is symmetric, right?~~
  - 3) If we changed I' goto 1)
    - **May never terminate**

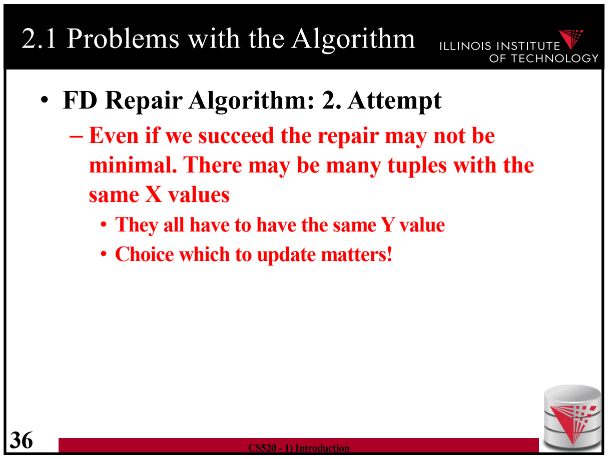


2.1 Constraint Repair

Example: Constraint Repair

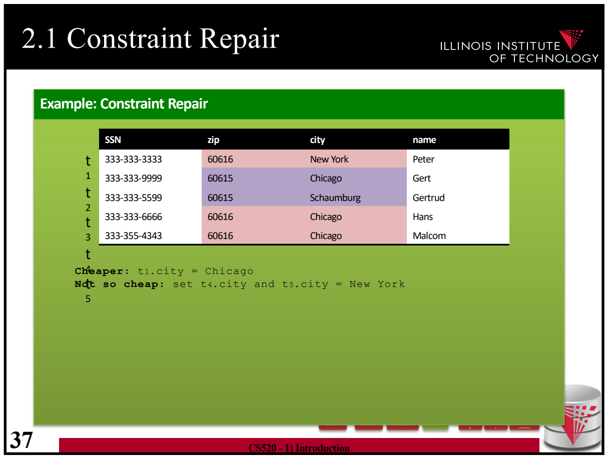
	SSN	zip	city	name
t_1	333-333-3333	60616	New York	Peter
t_2	333-333-9999	60615	Chicago	Gert
t_3	333-333-5599	60615	Schaumburg	Gertrud
t_4	333-333-6666	60616	Chicago	Hans
t_5	333-355-4343	60616	Chicago	Malcom

```
t4 and t1: set t4.city = New York
t1 and t5: set t1.city = Chicago

Now t1 and t4 and t4 and t5 in violation!

t4 and t1: set t1.city = New York
T5 and t4: set t4.city = Chicago

repeat
```



2.1 Problems with the Algorithm

- **FD Repair Algorithm: 2. Attempt**
  - **Even if we succeed the repair may not be minimal. There may be many tuples with the same X values**
    - **They all have to have the same Y value**
    - **Choice which to update matters!**

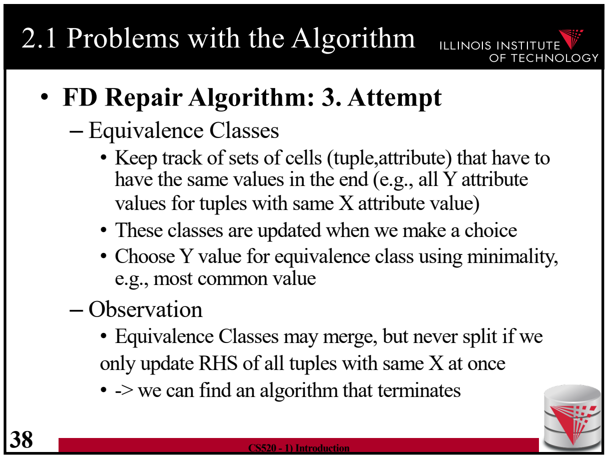


2.1 Constraint Repair

Example: Constraint Repair

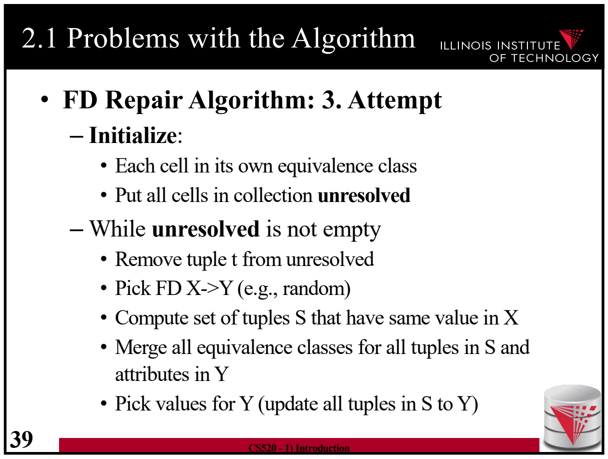
	SSN	zip	city	name
t1	333-333-3333	60616	New York	Peter
t2	333-333-9999	60615	Chicago	Gert
t3	333-333-5599	60615	Schaumburg	Gertrud
t4	333-333-6666	60616	Chicago	Hans
t5	333-355-4343	60616	Chicago	Malcom

```
Cheaper: t1.city = Chicago
Not so cheap: set t4.city and t5.city = New York
```



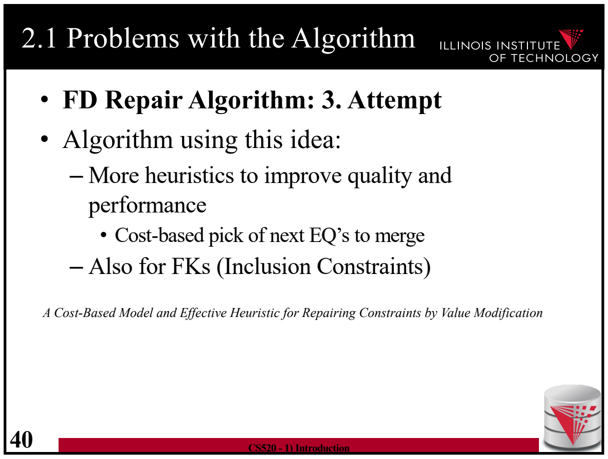
2.1 Problems with the Algorithm

- **FD Repair Algorithm: 3. Attempt**
  - Equivalence Classes
    - Keep track of sets of cells (tuple,attribute) that have to have the same values in the end (e.g., all Y attribute values for tuples with same X attribute value)
    - These classes are updated when we make a choice
    - Choose Y value for equivalence class using minimality, e.g., most common value
  - Observation
    - Equivalence Classes may merge, but never split if we only update RHS of all tuples with same X at once
    - -> we can find an algorithm that terminates



2.1 Problems with the Algorithm

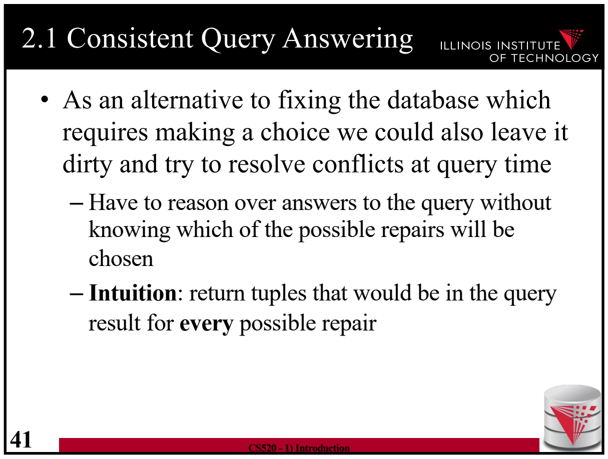
- **FD Repair Algorithm: 3. Attempt**
  - **Initialize**:
    - Each cell in its own equivalence class
    - Put all cells in collection **unresolved**
  - While **unresolved** is not empty
    - Remove tuple t from unresolved
    - Pick FD X->Y (e.g., random)
    - Compute set of tuples S that have same value in X
    - Merge all equivalence classes for all tuples in S and attributes in Y
    - Pick values for Y (update all tuples in S to Y)



2.1 Problems with the Algorithm

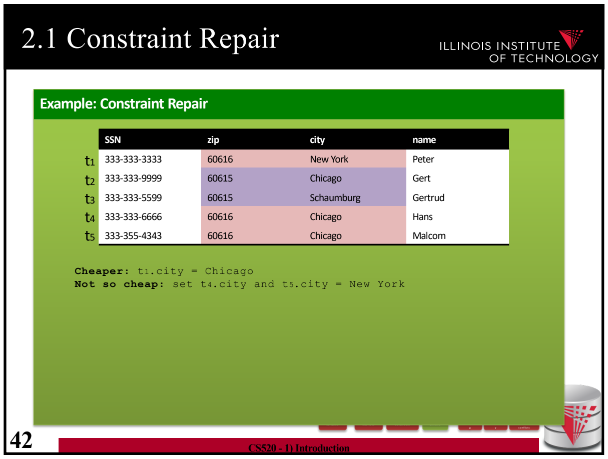
- **FD Repair Algorithm: 3. Attempt**
- Algorithm using this idea:
  - More heuristics to improve quality and performance
    - Cost-based pick of next EQ's to merge
  - Also for FKs (Inclusion Constraints)

A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification



2.1 Consistent Query Answering

- As an alternative to fixing the database which requires making a choice we could also leave it dirty and try to resolve conflicts at query time
  - Have to reason over answers to the query without knowing which of the possible repairs will be chosen
  - **Intuition**: return tuples that would be in the query result for **every** possible repair

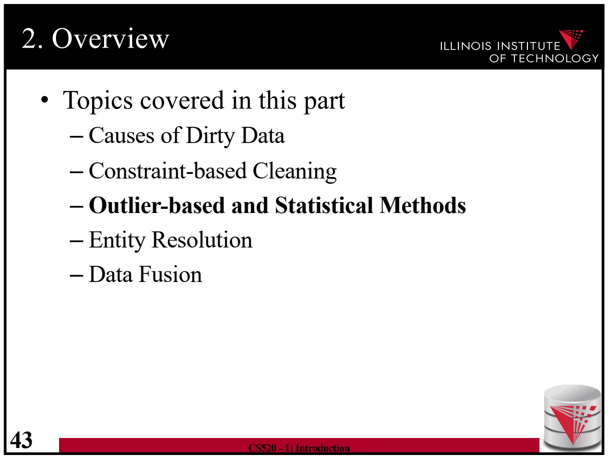


2.1 Constraint Repair

Example: Constraint Repair

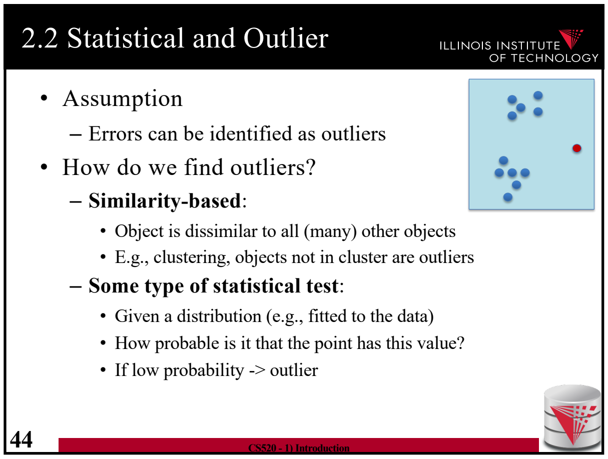
	SSN	zip	city	name
t_1	333-333-3333	60616	New York	Peter
t_2	333-333-9999	60615	Chicago	Gert
t_3	333-333-5599	60615	Schaumburg	Gertrud
t_4	333-333-6666	60616	Chicago	Hans
t_5	333-355-4343	60616	Chicago	Malcom

```
Cheaper: t1.city = Chicago
Not so cheap: set t4.city and t5.city = New York
```



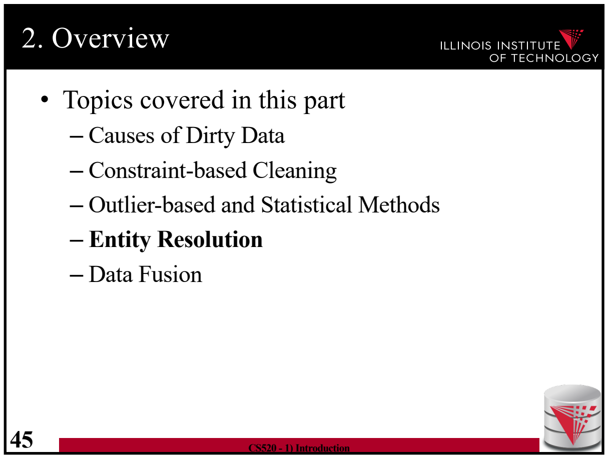
2. Overview

- Topics covered in this part
  - Causes of Dirty Data
  - Constraint-based Cleaning
  - **Outlier-based and Statistical Methods**
  - Entity Resolution
  - Data Fusion



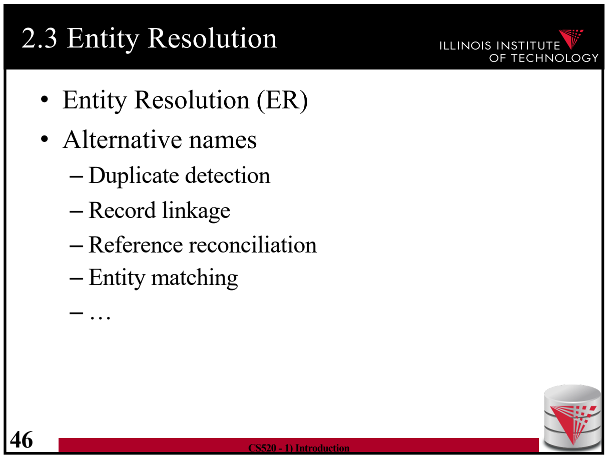
2.2 Statistical and Outlier

- Assumption
  - Errors can be identified as outliers
- How do we find outliers?
  - **Similarity-based**:
    - Object is dissimilar to all (many) other objects
    - E.g., clustering, objects not in cluster are outliers
  - **Some type of statistical test**:
    - Given a distribution (e.g., fitted to the data)
    - How probable is it that the point has this value?
    - If low probability -> outlier



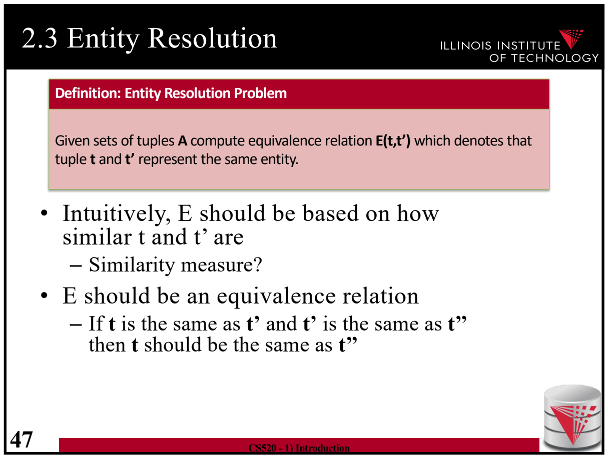
2. Overview

- Topics covered in this part
  - Causes of Dirty Data
  - Constraint-based Cleaning
  - Outlier-based and Statistical Methods
  - **Entity Resolution**
  - Data Fusion



2.3 Entity Resolution

- Entity Resolution (ER)
- Alternative names
  - Duplicate detection
  - Record linkage
  - Reference reconciliation
  - Entity matching
  - …

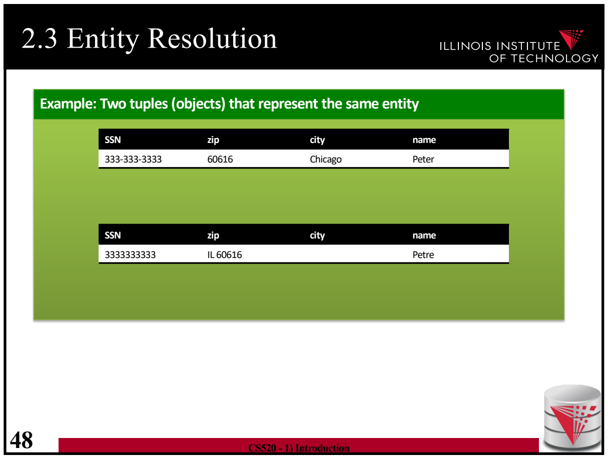


2.3 Entity Resolution

Definition: Entity Resolution Problem

Given sets of tuples **A** compute equivalence relation **E(t,t')** which denotes that tuple **t** and **t'** represent the same entity.

- Intuitively, E should be based on how similar t and t' are
  - Similarity measure?
- E should be an equivalence relation
  - If **t** is the same as **t'** and **t'** is the same as **t''** then **t** should be the same as **t''**



2.3 Entity Resolution

ILLINOIS INSTITUTE OF TECHNOLOGY

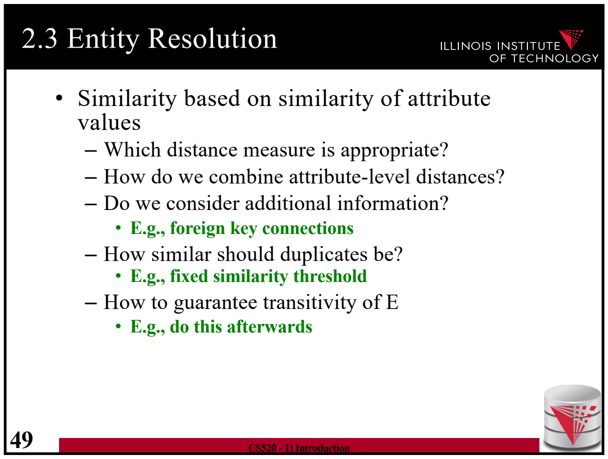
Example: Two tuples (objects) that represent the same entity

SSN	zip	city	name
333-333-3333	60616	Chicago	Peter

SSN	zip	city	name
3333333333	IL 60616		Petre

48 CS520 - 1) Introduction

48



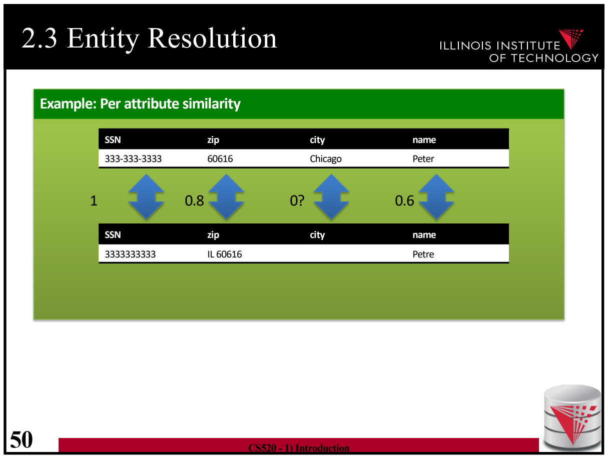
2.3 Entity Resolution

ILLINOIS INSTITUTE OF TECHNOLOGY

- Similarity based on similarity of attribute values
  - Which distance measure is appropriate?
  - How do we combine attribute-level distances?
  - Do we consider additional information?
    - **E.g., foreign key connections**
  - How similar should duplicates be?
    - **E.g., fixed similarity threshold**
  - How to guarantee transitivity of E
    - **E.g., do this afterwards**

49 CS520 - 1) Introduction

49



2.3 Entity Resolution

ILLINOIS INSTITUTE OF TECHNOLOGY

Example: Per attribute similarity

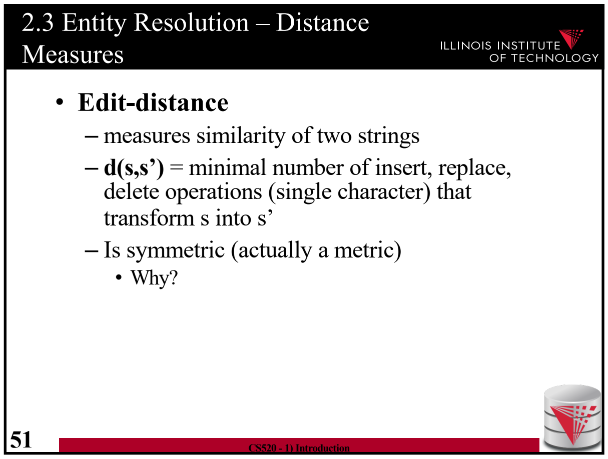
SSN	zip	city	name
333-333-3333	60616	Chicago	Peter

1 0.8 0? 0.6

SSN	zip	city	name
3333333333	IL 60616		Petre

50 CS520 - 1) Introduction

50



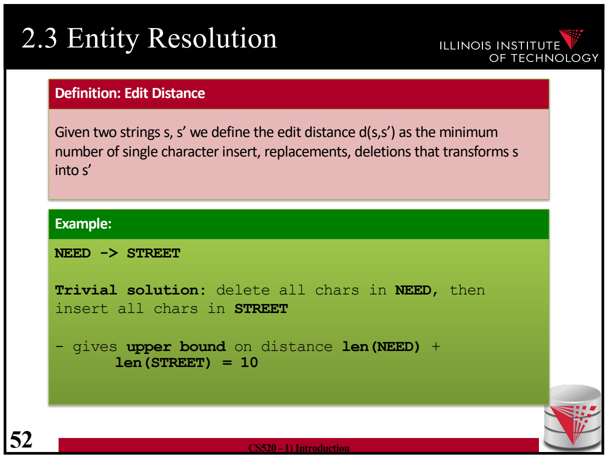
2.3 Entity Resolution – Distance Measures

ILLINOIS INSTITUTE OF TECHNOLOGY

- **Edit-distance**
  - measures similarity of two strings
  - **d(s,s')** = minimal number of insert, replace, delete operations (single character) that transform s into s'
  - Is symmetric (actually a metric)
    - Why?

51 CS520 - 1) Introduction

51



2.3 Entity Resolution

ILLINOIS INSTITUTE OF TECHNOLOGY

Definition: Edit Distance

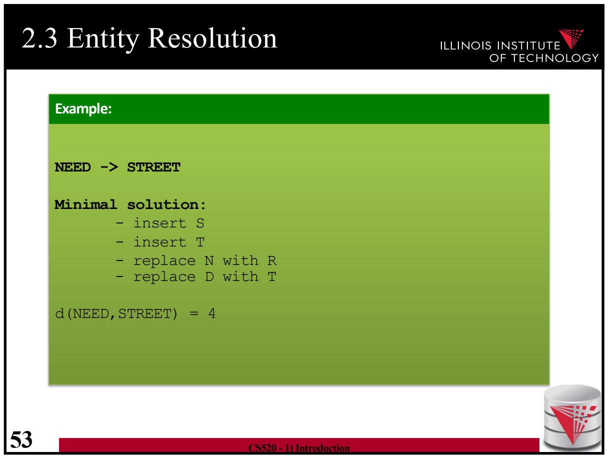
Given two strings s, s' we define the edit distance d(s,s') as the minimum number of single character insert, replacements, deletions that transforms s into s'

Example:

```
NEED -> STREET

Trivial solution: delete all chars in NEED, then
insert all chars in STREET

- gives upper bound on distance len(NEED) +
       len(STREET) = 10
```

52 CS520 - 1) Introduction

52



2.3 Entity Resolution

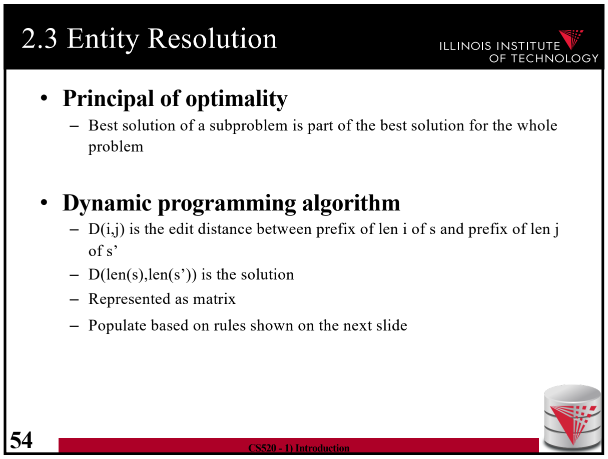
ILLINOIS INSTITUTE OF TECHNOLOGY

Example:

```
NEED -> STREET

Minimal solution:
       - insert S
       - insert T
       - replace N with R
       - replace D with T

d(NEED,STREET) = 4
```

53 CS520 - 1) Introduction

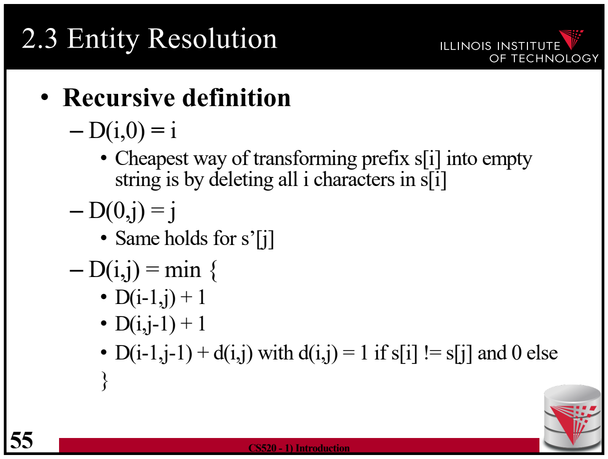
53



2.3 Entity Resolution

ILLINOIS INSTITUTE OF TECHNOLOGY

- **Principal of optimality**
  - Best solution of a subproblem is part of the best solution for the whole problem
- **Dynamic programming algorithm**
  - D(i,j) is the edit distance between prefix of len i of s and prefix of len j of s'
  - D(len(s),len(s')) is the solution
  - Represented as matrix
  - Populate based on rules shown on the next slide

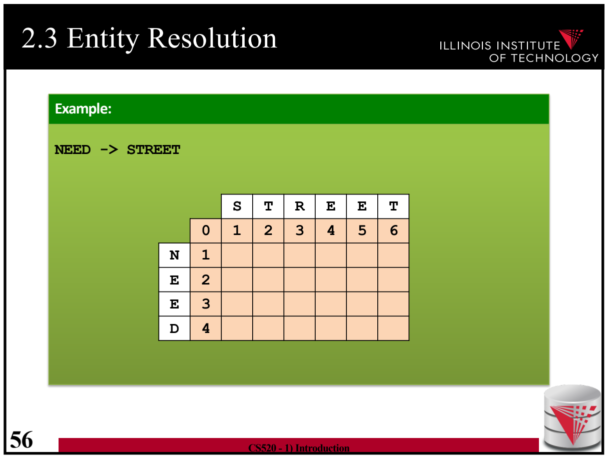


2.3 Entity Resolution

ILLINOIS INSTITUTE OF TECHNOLOGY

- **Recursive definition**
  - D(i,0) = i
    - Cheapest way of transforming prefix s[i] into empty string is by deleting all i characters in s[i]
  - D(0,j) = j
    - Same holds for s'[j]
  - D(i,j) = min {
    - D(i-1,j) + 1
    - D(i,j-1) + 1
    - D(i-1,j-1) + d(i,j) with d(i,j) = 1 if s[i] != s[j] and 0 else

    }



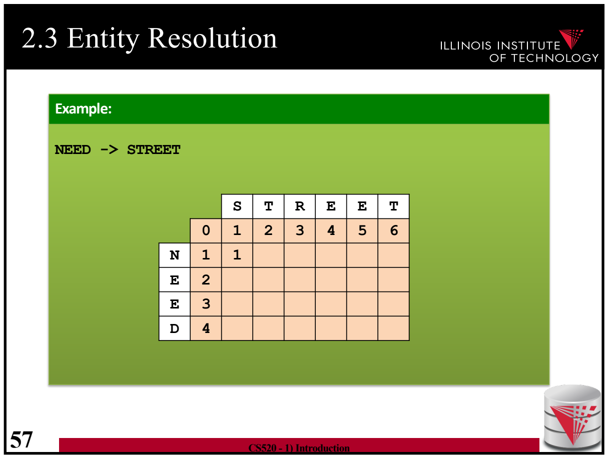
2.3 Entity Resolution

ILLINOIS INSTITUTE OF TECHNOLOGY

Example:

`NEED -> STREET`

		S	T	R	E	E	T
	0	1	2	3	4	5	6
N	1						
E	2						
E	3						
D	4						



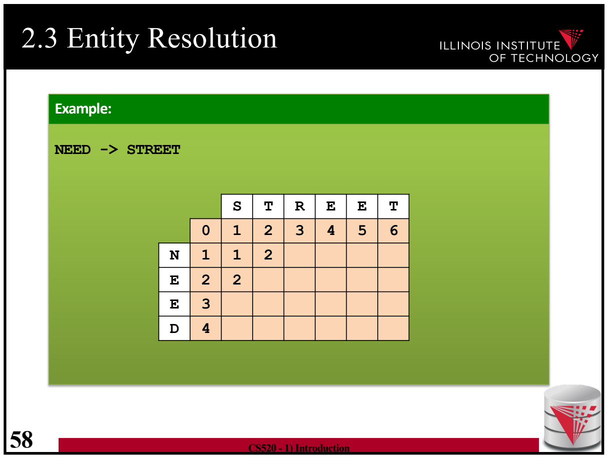
2.3 Entity Resolution

ILLINOIS INSTITUTE OF TECHNOLOGY

Example:

`NEED -> STREET`

		S	T	R	E	E	T
	0	1	2	3	4	5	6
N	1	1					
E	2						
E	3						
D	4						



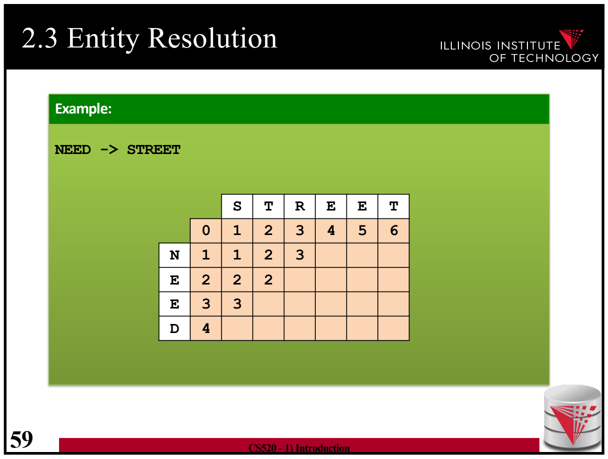
2.3 Entity Resolution

ILLINOIS INSTITUTE OF TECHNOLOGY

Example:

`NEED -> STREET`

		S	T	R	E	E	T
	0	1	2	3	4	5	6
N	1	1	2				
E	2	2					
E	3						
D	4						



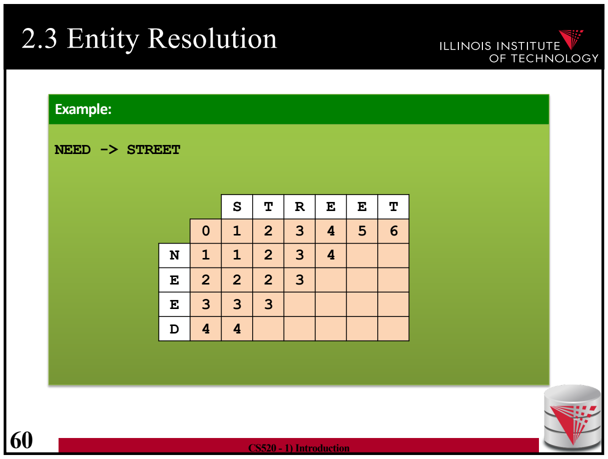
2.3 Entity Resolution

ILLINOIS INSTITUTE OF TECHNOLOGY

Example:

`NEED -> STREET`

		S	T	R	E	E	T
	0	1	2	3	4	5	6
N	1	1	2	3			
E	2	2	2				
E	3	3					
D	4						

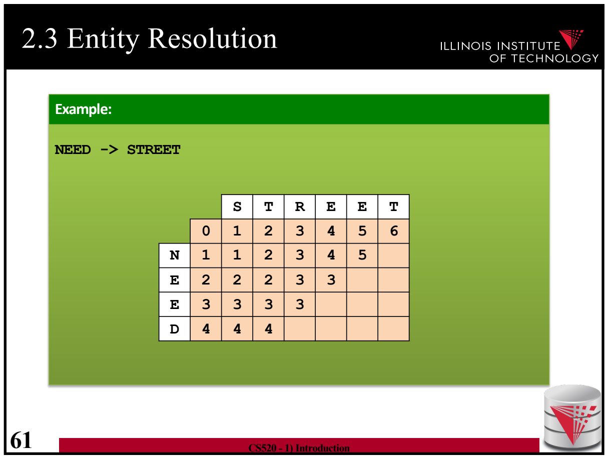


2.3 Entity Resolution

Example:

NEED -> STREET

		S	T	R	E	E	T
	0	1	2	3	4	5	6
N	1	1	2	3	4		
E	2	2	2	3			
E	3	3	3				
D	4	4					

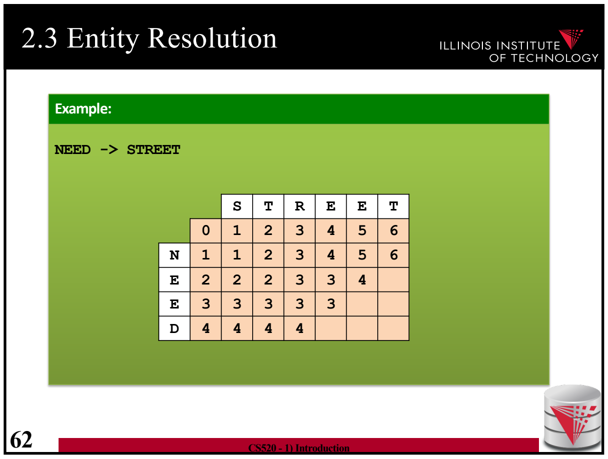


2.3 Entity Resolution

Example:

NEED -> STREET

		S	T	R	E	E	T
	0	1	2	3	4	5	6
N	1	1	2	3	4	5	
E	2	2	2	3	3		
E	3	3	3	3			
D	4	4	4				

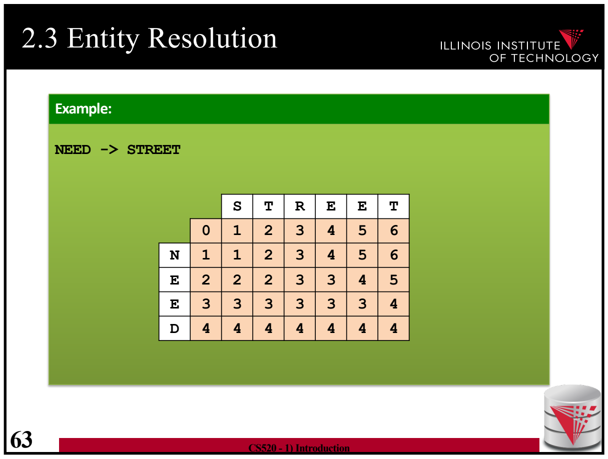


2.3 Entity Resolution

Example:

NEED -> STREET

		S	T	R	E	E	T
	0	1	2	3	4	5	6
N	1	1	2	3	4	5	6
E	2	2	2	3	3	4	
E	3	3	3	3	3		
D	4	4	4	4			

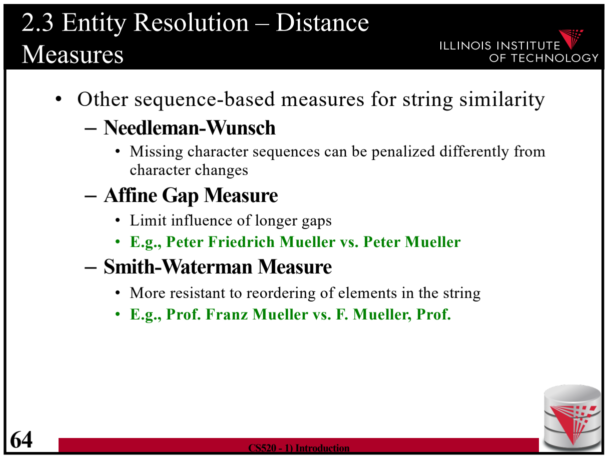


2.3 Entity Resolution

Example:

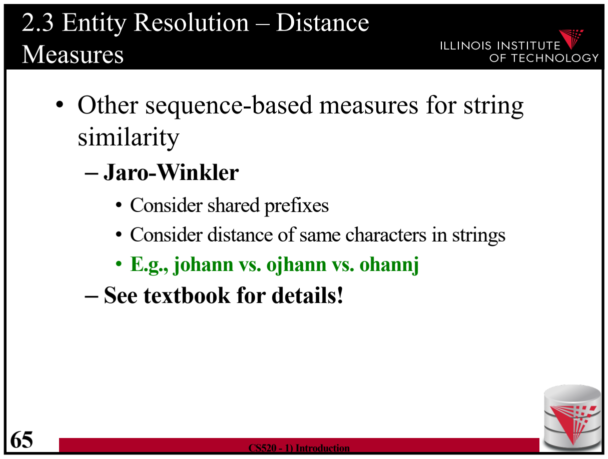
NEED -> STREET

		S	T	R	E	E	T
	0	1	2	3	4	5	6
N	1	1	2	3	4	5	6
E	2	2	2	3	3	4	5
E	3	3	3	3	3	3	4
D	4	4	4	4	4	4	4



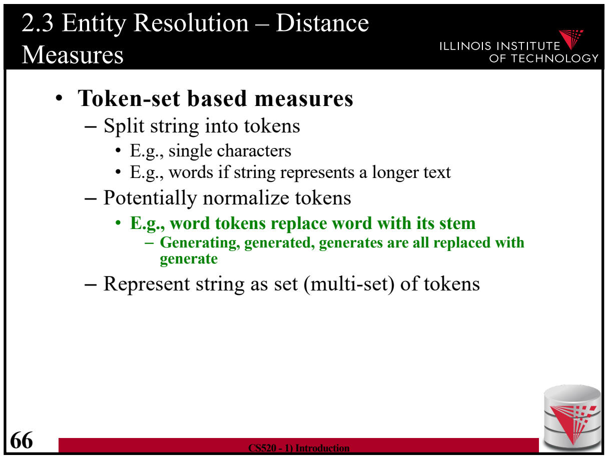
2.3 Entity Resolution – Distance Measures

- Other sequence-based measures for string similarity
  - **Needleman-Wunsch**
    - Missing character sequences can be penalized differently from character changes
  - **Affine Gap Measure**
    - Limit influence of longer gaps
    - **E.g., Peter Friedrich Mueller vs. Peter Mueller**
  - **Smith-Waterman Measure**
    - More resistant to reordering of elements in the string
    - **E.g., Prof. Franz Mueller vs. F. Mueller, Prof.**



2.3 Entity Resolution – Distance Measures

- Other sequence-based measures for string similarity
  - **Jaro-Winkler**
    - Consider shared prefixes
    - Consider distance of same characters in strings
    - **E.g., johann vs. ojhann vs. ohannj**
  - **See textbook for details!**

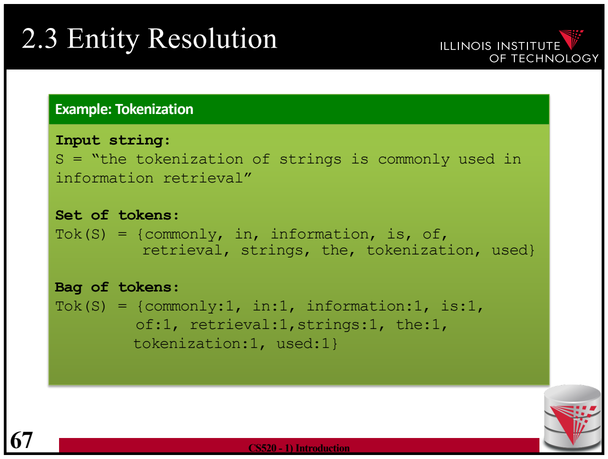


2.3 Entity Resolution – Distance Measures

- **Token-set based measures**
  - Split string into tokens
    - E.g., single characters
    - E.g., words if string represents a longer text
  - Potentially normalize tokens
    - **E.g., word tokens replace word with its stem**
      - **Generating, generated, generates are all replaced with generate**
  - Represent string as set (multi-set) of tokens

66 CS520 - 1) Introduction

66



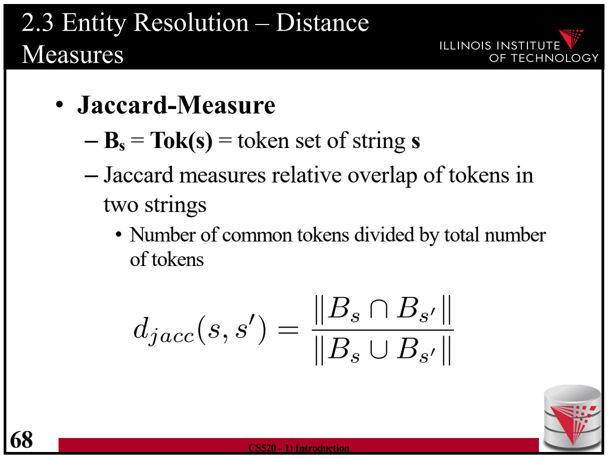
2.3 Entity Resolution

Example: Tokenization

```
Input string:
S = "the tokenization of strings is commonly used in
information retrieval"

Set of tokens:
Tok(S) = {commonly, in, information, is, of,
            retrieval, strings, the, tokenization, used}

Bag of tokens:
Tok(S) = {commonly:1, in:1, information:1, is:1,
           of:1, retrieval:1,strings:1, the:1,
           tokenization:1, used:1}
```

67 CS520 - 1) Introduction

67



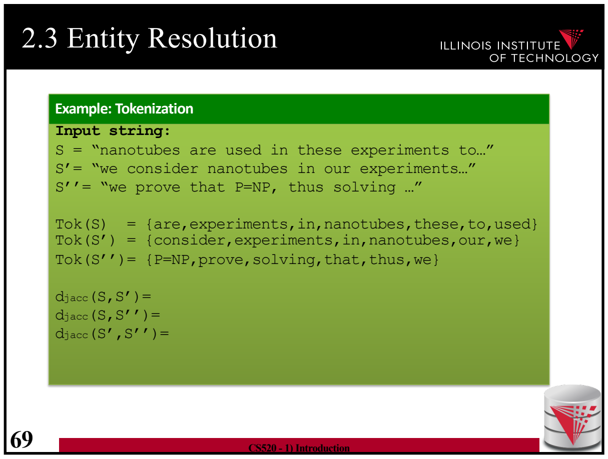
2.3 Entity Resolution – Distance Measures

- **Jaccard-Measure**
  - **B_s** = **Tok(s)** = token set of string **s**
  - Jaccard measures relative overlap of tokens in two strings
    - Number of common tokens divided by total number of tokens

$$d_{jacc}(s, s') = \frac{\|B_s \cap B_{s'}\|}{\|B_s \cup B_{s'}\|}$$

68 CS520 - 1) Introduction

68



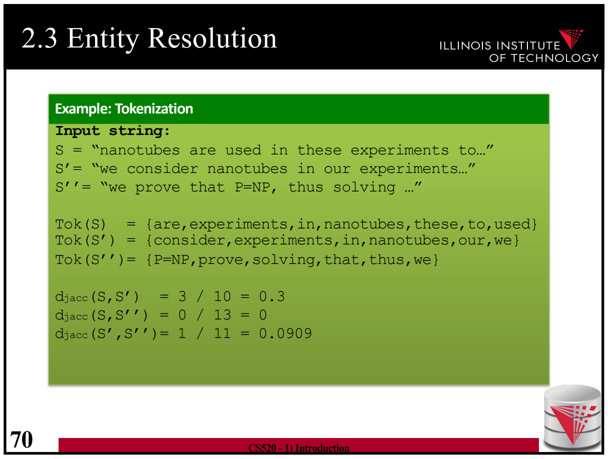
2.3 Entity Resolution

Example: Tokenization

```
Input string:
S = "nanotubes are used in these experiments to…"
S'= "we consider nanotubes in our experiments…"
S''= "we prove that P=NP, thus solving …"

Tok(S)   = {are,experiments,in,nanotubes,these,to,used}
Tok(S')  = {consider,experiments,in,nanotubes,our,we}
Tok(S'')= {P=NP,prove,solving,that,thus,we}

djacc(S,S')=
djacc(S,S'')=
djacc(S',S'')=
```

69 CS520 - 1) Introduction

69



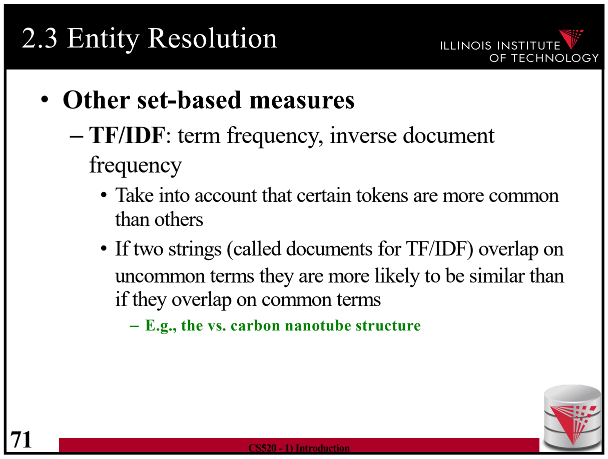
2.3 Entity Resolution

Example: Tokenization

```
Input string:
S = "nanotubes are used in these experiments to…"
S'= "we consider nanotubes in our experiments…"
S''= "we prove that P=NP, thus solving …"

Tok(S)   = {are,experiments,in,nanotubes,these,to,used}
Tok(S')  = {consider,experiments,in,nanotubes,our,we}
Tok(S'')= {P=NP,prove,solving,that,thus,we}

djacc(S,S')   = 3 / 10 = 0.3
djacc(S,S'')  = 0 / 13 = 0
djacc(S',S'')= 1 / 11 = 0.0909
```

70 CS520 - 1) Introduction

70



2.3 Entity Resolution

- **Other set-based measures**
  - **TF/IDF**: term frequency, inverse document frequency
    - Take into account that certain tokens are more common than others
    - If two strings (called documents for TF/IDF) overlap on uncommon terms they are more likely to be similar than if they overlap on common terms
      - **E.g., the vs. carbon nanotube structure**

71 CS520 - 1) Introduction

71

2.3 Entity Resolution

- **TF/IDF**: term frequency, inverse document frequency
  - Represent documents as feature vectors
    - One dimension for each term
    - Value computed as frequency times IDF
      - Inverse of frequency of term in the set of all documents
  - Compute cosine similarity between two feature vectors
    - Measure how similar they are in term distribution (weighted by how uncommon terms are)
    - Size of the documents does not matter
  - **See textbook for details**

2.3 Entity Resolution

- **Entity resolution**
  - Concatenate attribute values of tuples and use string similarity measure
    - Loose information encoded by tuple structure
    - **E.g., [Gender:male,Salary:9000]**
      -> "Gender:male,Salary:9000"
      or -> "male,9000"
  - Combine distance measures for single attributes
    - Weighted sum or more complex combinations
      - E.g., $d(t,t') = w_1 \times d_A(t.A, t'.A) + w_2 \times d_B(t.B, t'.B)$
  - Use quadratic distance measure
    - E.g., earth-movers distance

2.3 Entity Resolution

- **Entity resolution**
  - Rule-based approach
    - Set of **if this than that** rules
  - Learning-based approaches
  - Clustering-based approaches
  - Probabilistic approaches to matching
  - Collective matching

2.3 Entity Resolution

- **Weighted linear combination**
  - Say tuples have **n** attributes
  - **w$_i$**: predetermined weight of an attribute
  - **d$_i$(t,t')**: similarity measure for the **ith** attribute

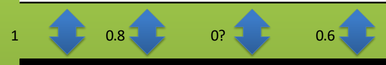
$$d(t,t') = \sum_{i=0}^{n} w_i \times d_i(t,t')$$

- Tuples match if **d(t,t') > β** for a threshold **β**

2.3 Entity Resolution

Example: Weighted sum of attribute similarities

SSN	zip	city	name
333-333-3333	60616	Chicago	Peter



SSN	zip	city	name
3333333333	IL 60616		Petre

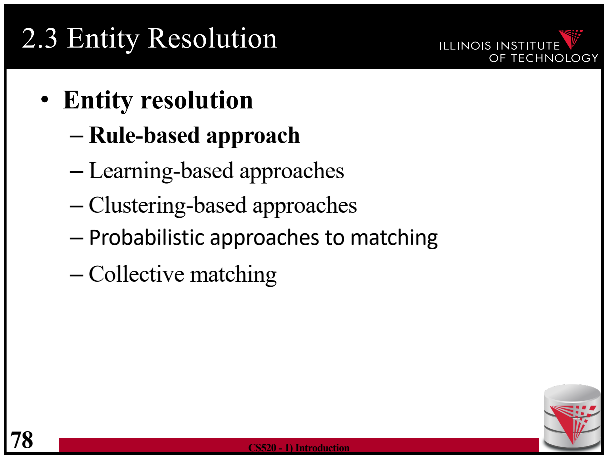
Assumption: SSNs and names are most important, city and zip are not very predictive

$$w_{SSN} = 0.4, w_{zip} = 0.05, w_{city} = 0.15, w_{name} = 0.4$$

$$\begin{aligned} d(t,t') &= 0.4 \times 1 + 0.05 \times 0.8 + 0.15 \times 0 + 0.4 \times 0.6 \\ &= 0.4 + 0.04 + 0 + 0.24 \\ &= 0.68 \end{aligned}$$

2.3 Entity Resolution

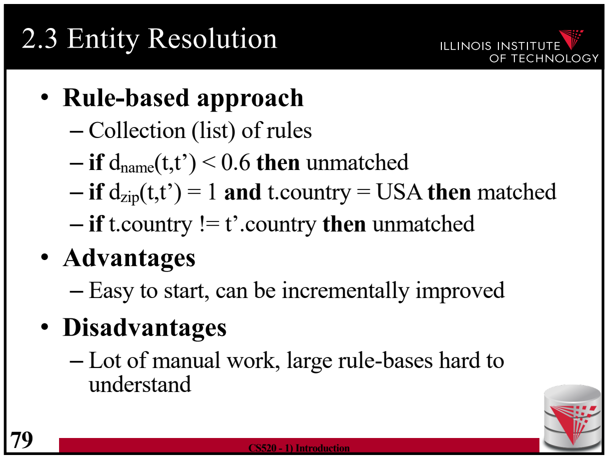
- **Weighted linear combination**
  - How to determine weights?
    - **E.g., have labeled training data and use ML to learn weights**
  - Use non-linear function?



2.3 Entity Resolution

- **Entity resolution**
  - **Rule-based approach**
  - Learning-based approaches
  - Clustering-based approaches
  - Probabilistic approaches to matching
  - Collective matching

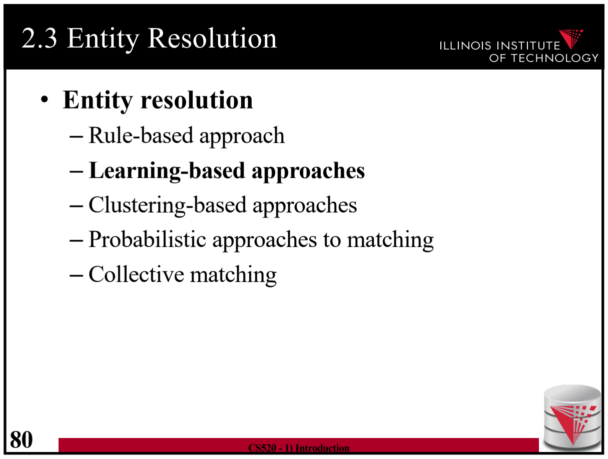
78



2.3 Entity Resolution

- **Rule-based approach**
  - Collection (list) of rules
  - **if** d_{name}(t,t') < 0.6 **then** unmatched
  - **if** d_{zip}(t,t') = 1 **and** t.country = USA **then** matched
  - **if** t.country != t'.country **then** unmatched
- **Advantages**
  - Easy to start, can be incrementally improved
- **Disadvantages**
  - Lot of manual work, large rule-bases hard to understand

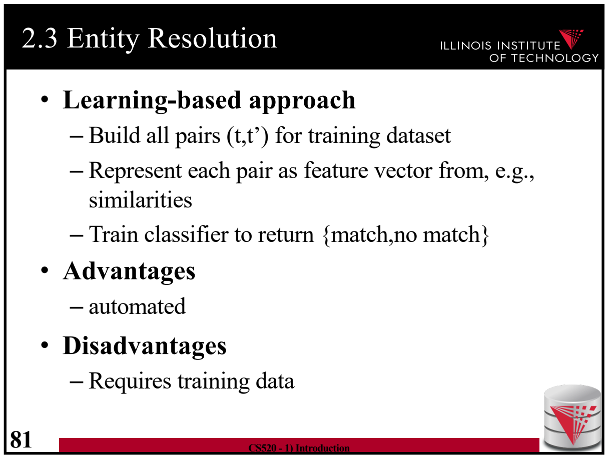
79



2.3 Entity Resolution

- **Entity resolution**
  - Rule-based approach
  - **Learning-based approaches**
  - Clustering-based approaches
  - Probabilistic approaches to matching
  - Collective matching

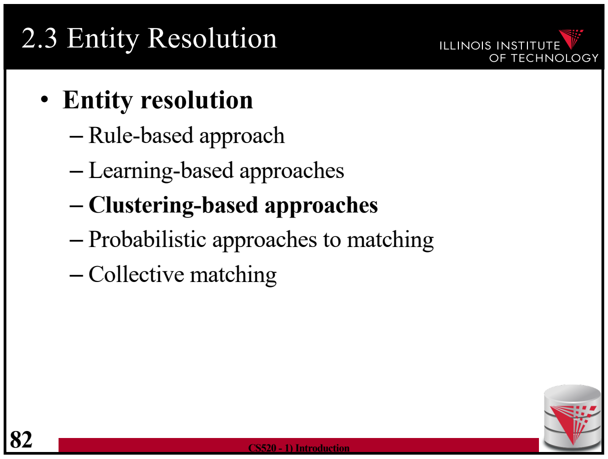
80



2.3 Entity Resolution

- **Learning-based approach**
  - Build all pairs (t,t') for training dataset
  - Represent each pair as feature vector from, e.g., similarities
  - Train classifier to return {match,no match}
- **Advantages**
  - automated
- **Disadvantages**
  - Requires training data

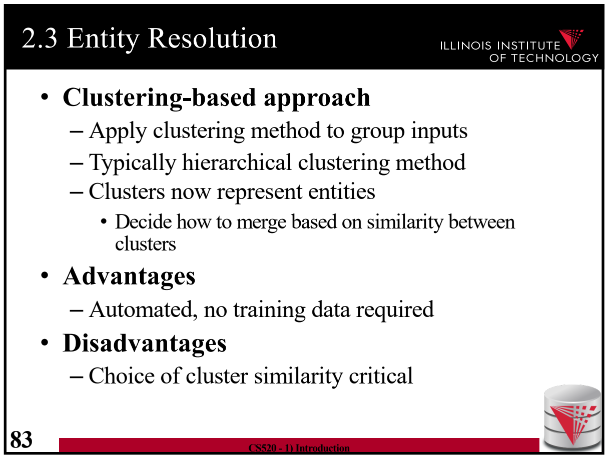
81



2.3 Entity Resolution

- **Entity resolution**
  - Rule-based approach
  - Learning-based approaches
  - **Clustering-based approaches**
  - Probabilistic approaches to matching
  - Collective matching

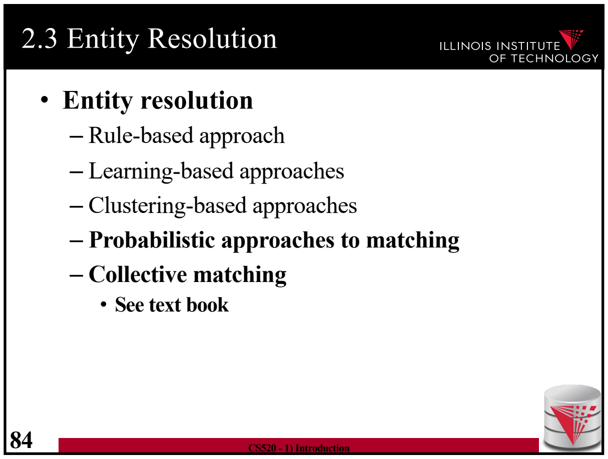
82



2.3 Entity Resolution

- **Clustering-based approach**
  - Apply clustering method to group inputs
  - Typically hierarchical clustering method
  - Clusters now represent entities
    - Decide how to merge based on similarity between clusters
- **Advantages**
  - Automated, no training data required
- **Disadvantages**
  - Choice of cluster similarity critical

83

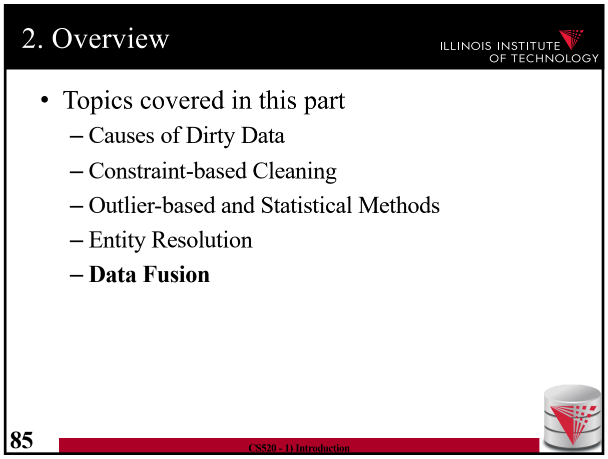


2.3 Entity Resolution

ILLINOIS INSTITUTE OF TECHNOLOGY

- **Entity resolution**
  - Rule-based approach
  - Learning-based approaches
  - Clustering-based approaches
  - **Probabilistic approaches to matching**
  - **Collective matching**
    - **See text book**

84

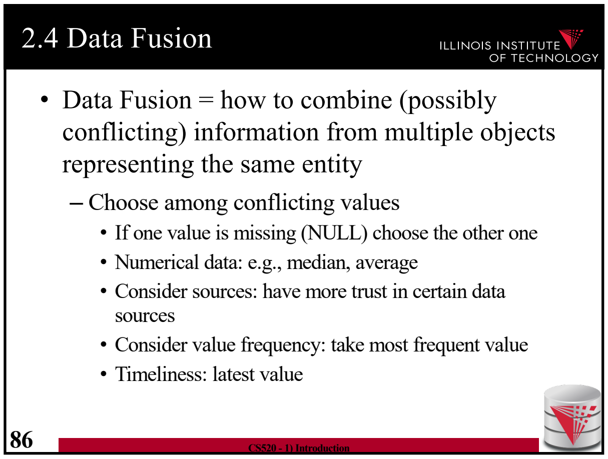


2. Overview

ILLINOIS INSTITUTE OF TECHNOLOGY

- Topics covered in this part
  - Causes of Dirty Data
  - Constraint-based Cleaning
  - Outlier-based and Statistical Methods
  - Entity Resolution
  - **Data Fusion**

85

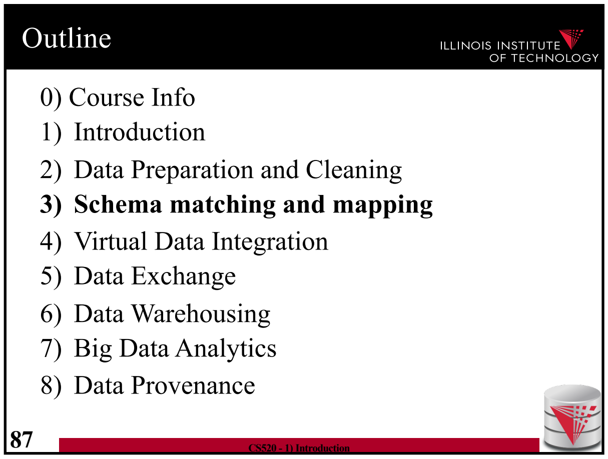


2.4 Data Fusion

ILLINOIS INSTITUTE OF TECHNOLOGY

- Data Fusion = how to combine (possibly conflicting) information from multiple objects representing the same entity
  - Choose among conflicting values
    - If one value is missing (NULL) choose the other one
    - Numerical data: e.g., median, average
    - Consider sources: have more trust in certain data sources
    - Consider value frequency: take most frequent value
    - Timeliness: latest value

86



Outline

ILLINOIS INSTITUTE OF TECHNOLOGY

0) Course Info
1) Introduction
2) Data Preparation and Cleaning
3) Schema matching and mapping
4) Virtual Data Integration
5) Data Exchange
6) Data Warehousing
7) Big Data Analytics
8) Data Provenance

87