Outline

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

About me

Hi, I am Boris Glavic, Assistant Professor in CS
I am a database guy!
I will teach you: database stuff

What is information integration?

• Combination of data and content from multiple sources into a common format
  – Completeness
  – Correctness
  – Efficient

Why Information Integration?

• Data is already available, right?
• ..., but
• Heterogeneity
  – Structural
    • Data model (relational, XML, unstructured)
    • Schema (if there)
  – Semantic
    • Naming and identity conflicts
    • Data conflicts
  – Syntactic
    • Interfaces (web form, query language, binary file)

Why Information Integration?

• Autonomy
  – Sources may not give you unlimited access
    • Web form only support a fixed format of queries
    • Does not allow access to unlimited amounts of data
  – Source may not be available all the time
    • Naming and identity conflicts
    • Data conflicts
  – Data, schema, and interfaces of sources may change
    • Potentially without notice
“Real World” Examples?

- Portal websites
  - Flight websites (e.g., Expedia) gather data from multiple airlines, hotels
- Google News
  - Integrates information from a large number of news sources
- Science:
  - Biomedical data source
- Business
  - Warehouses: integrate transactional data

Example Integration Problem [1]

- Integrate stock ticker data from two web services A and B
  - Service A: Web form (Company name, year)
  - Service B: Web form (year)

Example Integration Problem [2]

- Service A:
  - <Stock>
    - <Company>IBM</Company>
    - <DollarValue>155.8</DollarValue>
    - <Month>12</Month>
  - </Stock>

- Service B:
  - <Stock>
    - <Company>International Business Machines</Company>
    - <Date>2014-08-01</Date>
    - <Value>106.8</Value>
    - <Currency>Euro</Currency>
  - </Stock>

Example Integration Problem [2]

- Service A:
  - <Stock>
    - <Company>IBM</Company>
    - <DollarValue>155.8</DollarValue>
    - <Month>12</Month>
  - </Stock>

- Service B:
  - <Stock>
    - <Company>International Business Machines</Company>
    - <Date>2014-08-01</Date>
    - <Value>106.8</Value>
    - <Currency>Euro</Currency>
  - </Stock>

Example Integration Problem [2]

- SQL interface for integrated service

  SELECT month, value
  FROM ticker
  WHERE year = 2014
  AND cmp = 'IBM'

- Service A: (IBM, 2014)
- Service B: (2014)
Example Integration Problem [4]

• For web service A we can either
  – Get stocks for IBM in all years
  – Get stocks for all companies in 2014
  – Get stocks for IBM in 2014
• Trade-off between amount of processing that we have to do locally, amount of data that is shipped, …

Example Integration Problem [5]

• Service A: (IBM, 2014)
• Service B: (2014)

Example Integration Problem [6]

• Service A:
  <Stock>
  <Company>IBM</Company>
  <DollarValue>155.8</DollarValue>
  <Month>12</Month>
  …
• Service B:
  <Stock>
  <Company>International Business Machines</Company>
  <Date>2014-12-01</Date>
  …

Example Integration Problem [7]

• IBM vs. Integrated Business Machines

Example Integration Problem [8]

• Granularity of time attribute
  – Month vs. data
• What if both services return different values (after adapting granularity)
  – Average?
  – Median?
  – Trust-based?

Example Integration Problem [9]

• Return final results:
  <Stock>
  <Month>01</Month>
  <Value>105</Value>
  …
  <Stock>
  <Month>12</Month>
  <Value>107</Value>
  …
Why hard?

- System challenges
  - Different platforms (OS/Software)
  - Efficient query processing over multiple heterogeneous systems
- Social challenges
  - Find relevant data
  - Convince people to share their data
- Heterogeneity of data and schemas
  - A problem that even exists if we use same system


- Often called AI-complete
  - Meaning: “It requires human intelligence to solve the problem”
  - Unlikely that general completely automated solutions will exit
- So why do we still sit here
  - There exist automated solutions for relevant less general problems
  - Semi-automated solutions can reduce user effort (and may be less error prone)

AI completeness

- Yes, but still why is this problem really so hard?
  - Lack of information: e.g., the attributes of a database schema have only names and data types, but no computer interpretable information on what type of information is stored in the attribute
  - Undecidable computational problems: to decide whether a user query can be answered from a set of sources that provide different views on the data requires query containment checks which are undecidable for certain query types

Relevant less general problems

- Data cleaning:
  - Clean dirty data before integration
  - Conformance with a set of constraints
  - Deal with missing and outlier values
- Entity resolution
  - Determine which objects from multiple dataset represent the same real world entity
- Data fusion
  - Merge (potentially conflicting) data for the same entity

Relevant less general problems

- Schema matching
  - Given two schemas determine which elements store the same type of information
- Schema mapping
  - Describe the relationships between schemas
  - Allows us to rewrite queries written against one schema into queries of another schema
  - Allows us to translate data from one schema into

Relevant less general problems

- Virtual data integration
  - Answer queries written against a global mediated schema by running queries over local sources
- Data exchange
  - Map data from one schema into another
- Warehousing: Extract, Transform, Load
  - Clean, transform, fuse data and load it into a data warehouse to make it available for analysis
Relevant less general problems

• **Integration in Big Data Analytics**
  – Often “pay-as-you-go”:
    • No or limited schema
    • Engines support wide variety of data formats

• **Provenance**
  – Information about the origin and creation process of data
  – Very important for integrated data
    • E.g., “from which data source is this part of my query result”

Webpage and Faculty

• **Course Info**
  – Course Webpage: [http://cs.iit.edu/~cs520](http://cs.iit.edu/~cs520)
    • Used for announcements
    • Use it to discuss with me, TA, and fellow students

• **Faculty**
  – Boris Glavic ([http://cs.iit.edu/~glavic](http://cs.iit.edu/~glavic))
  – Email: bglavic@iit.edu
  – Phone: 312.567.5205
  – Office: Stuart Building, room 226C
  – Office Hours: Mondays, 12pm-1pm (and by appointment)

TAs

• **TAs**
  – TBA

Course Objectives

• Understand the problems that arise with querying heterogeneous and autonomous data sources

• Understand the differences and similarities between the data integration/exchange, data warehouse, and Big Data analytics approaches

• Be able to build parts of a small data integration pipeline by “glueing” existing systems with new code

Course Objectives cont.

• Have learned formal languages for expressing schema mappings

• Understand the difference between virtual and materialized integration (data integration vs. data exchange)

• Understand the concept of data provenance and know how to compute provenance

Workload and Grading

• **Exams (60%)**
  – Final

• **Homework Assignments** (preparation for exams!)
  – Practice theory for final exam
  – Practice the tools we discuss in class

• **Literature Review (40%)**
  – In groups of 2 students
  – Topics will be announced soon
  – You have to read a research paper
  – Papers will be assigned in the first few weeks of the course
  – You will give a short presentation (15min) on the topic in class
  – You will write a report summarizing and criticizing the paper (up to 4 pages)
Fraud Policies

- All work has to be original!
  - Cheating = 0 points for review/exam
  - Possibly E in course and further administrative sanctions
  - Every dishonesty will be reported to office of academic honesty

- Late policy:
  - -20% per day
  - You have to give your presentation to pass the course!
  - No exceptions!

Fraud Policies cont.

- Literature Review:
  - Every student has to contribute in both the presentation and report!
  - Don’t let others freeload on you hard work!
    - Inform me or TA immediately

Reading and Prerequisites

- Textbook: Doan, Halevy, and Ives.
  - Principles of Data Integration, 1st Edition
  - Morgan Kaufmann
  - Publication date: 2012
  - Prerequisites:
    - CS 425

Additional Reading

- Papers assigned for literature review
- Optional: Standard database textbook

Outline

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

- Topics covered in this part
  - Heterogeneity and Autonomy
  - Data Integration Tasks
  - Data Integration Architectures (Methods)
  - Some Formal Background (sorry!)

1.1 System Heterogeneity

- Hardware/Software
  - Different hardware capabilities of sources
  - Different protocols, binary file formats, …
  - Different access control mechanism
- Interface Heterogeneity
  - Different interfaces for accessing data from a source
    - HTML forms
    - XML-Webservices
    - Declarative language

1.1 Heterogeneity + Autonomy

- Taxonomy of Heterogeneity

- Mobile phone vs. server: Cannot evaluate cross-product of two 1GB relations on a mobile phone
- Different protocols, binary file formats, …
- FTP-access to files: public, ssh authentication, …
1.1 System Heterogeneity

- Interface Heterogeneity
  - Different interfaces for accessing data from a source
    - HTML forms
    - Services (SOA)
    - Declarative language
    - Files
    - Proprietary network protocol
    - ...

- Interface Heterogeneity – Expressiveness
  - Keyword-search vs. query language
  - Predicates: equality (=), inequality (<, !=)
  - Logical connectives: conjunctive (AND), disjunctive (OR), negation
  - Complex operations: aggregation, quantification
  - Limitations: restriction to particular tables, predicates, fixed queries with parameters, ...

- Interface Heterogeneity – Examples
  - Google search (+/-, site:, intitle:, filetype:
  - SQL
  - Web-form (with DB backend?)

- Interface Heterogeneity – Examples
  - Fixed choices: "Bound parameter"
1.1 System Heterogeneity

• Interface Heterogeneity – Examples
  – Email-client

Problems with interface heterogeneity
  – Global query language is more powerful
    • User queries may not be executable
    • Integration system has to evaluate part of the query
  – Bound parameters are incompatible with query
    • User query may not be executable

Example: more expressive global language
  – SQL with one table
    • books (title, author, year, isbn, genre)
  – Web form for books about history shown below
  – What problems do may arise translating user queries?

Query requires multiple requests
  SELECT title
  FROM books
  WHERE author LIKE 'King%';

How do we know what authors exist?

Query cannot be answered
  SELECT title
  FROM books
  WHERE genre = 'SciFi';

Web form is for history book only!
1.1 Heterogeneity + Autonomy

- Taxonomy of Heterogeneity

1.1 Structural Heterogeneity

- Data model
  - Different semantic/expressiveness
  - Different structure

- Schema
  - Integrity constraints, keys
  - Schema elements:
    - use attribute or separate relations
  - Structure:
    - e.g., normalized vs. denormalized relational schema

1.1 Structural Heterogeneity

- Example: data model
  - Relational model
  - XML model
  - JSON
  - OO

- Person and their addresses

1.1 Structural Heterogeneity

- Schema
  - Modeling choices
    - Relation vs. attribute
    - Attribute vs. value
    - Relation vs. value
  - Naming
  - Normalized vs. denormalized (relational concept)
  - Nesting vs. reference
1.1 Structural Heterogeneity

- Relation-relation conflicts
  - Naming conflicts
    - Relations with different name representing the same data (synonym)
    - Relations with same name representing different information (homonym)
  - Structural conflicts
    - Missing attributes
    - Many-to-one
    - Missing, but derivable attributes
  - Integrity constraint conflicts

- Attribute-attribute conflicts
  - Naming conflicts
    - Attributes with different name representing the same data (synonym)
    - Attributes with same name representing different information (homonym)
  - Default value conflict
  - Integrity constraint conflicts
    - Datatype
    - Constraints restricting values
1.1 Structural Heterogeneity

- **Normalized vs. denormalized**
  - E.g., relational model: Association between entities can be represented using multiple relations and foreign keys or one relation

1.1 Structural Heterogeneity

- **Nested vs. flat**
  - Association between entities can be represented using nesting or references (previous slides)

1.1 Structural Heterogeneity

- **Problems caused by schema heterogeneity**
  - Unified access to multiple schemas or integrate schemas into new schema
    - **Schema level**: schema mapping, model management operators, schema languages
    - **Data Level**: virtual data integration, data exchange, warehousing (ETL)

1.1 Semantic Heterogeneity

- **Semantic Heterogeneity**
  - Naming Conflicts
  - Identity Conflicts (Entity resolution)
  - Value Conflicts (Data Fusion)

1.1 Semantic Heterogeneity

- **Naming Conflicts**
  - Ontological (concepts)
    - Birds vs. Animals
  - Synonyms
    - Surname vs. last name
  - Homonyms
  - Units
    - Gallon vs. liter
  - Values
    - Manager vs. Boss
1.1 Semantic Heterogeneity

- **Ontological concepts**
  - Relationships between concepts
    - $A = B$ - Equivalence
    - $A \subseteq B$ - Inclusion
    - $A \cap B$ - Overlap
    - $A \neq B$ - Disjunction

Example:
- **Equivalence**: Human vs Homo sapiens
- **Inclusion**: Bird vs Animal
- **Overlap**: Animal vs aquatic lifeform
- **Disjunction**: Fish vs Mammal

---

1.1 Semantic Heterogeneity

- **Naming concepts (synonyms)**
  - Different words with same meaning

Example:
- Person (Title, Name, Age)
- Human (LastName, Age)

---

1.1 Semantic Heterogeneity

- **Naming concepts (homonyms)**
  - Same words with different meaning

Example:
- Person (Title, Name)
- Movie (Title, Year)

---

1.1 Semantic Heterogeneity

- **Naming concepts (units)**

Example:
- Person (Title, Name, Salary)
- Person (Title, Name, Salary)

---

1.1 Semantic Heterogeneity

- **Identity Conflicts**
  - What is an object?
    - E.g., multiple tuples in relational model
  - Central question:
    - Does object A represent the same entity as B
  - This problem has been called
    - Entity resolution
    - Record linkage
    - Deduplication
    - …
1.1 Semantic Heterogeneity

- Identity Conflicts
  
  Example
  (IBM, 300000000, USA)
  (International Business Machines Corporation, 50000)

- Value Conflicts
  - Objects representing the same entities have conflicting values for semantically equivalent attributes
    - We have to identify that these objects are the same entity first!
  - Resolving such conflicts require **Data Fusion**
    - Pick value from conflicting values
    - Numerical methods: e.g., average
    - Preferred value

1.1 Autonomy

- How autonomous are data sources
  - One company
    - Can enforce, e.g., schema and software
  - The web
    - Website decides
      - Interface
      - Determines access restrictions and limits
      - Availability
      - Format
      - Query restrictions

1.2 Data integration tasks

- Cleaning and preparation
- Entity resolution
- Data Fusion
- Schema matching
- Schema mapping
- Query rewrite
- Data translation

1.3 Data integration architectures

- Virtual data integration
- Data Exchange
- Peer-to-peer data integration
- Data warehousing
- Big Data analytics

1.4 Formal Background

- Query Equivalence
  - Complexity for different query classes
- Query Containment
  - Complexity for different query classes
- Datalog
  - Recursion + Negation
- Integrity Constraints
  - Logical encoding of integrity constraints
- Similarity Measures/Metrics
1.4 Integrity constraints

- You know some types of integrity constraints already
  - Functional dependencies
    - Keys are a special case
  - Foreign keys
    - We have not really formalized that

- Other types are
  - Conditional functional dependencies
  - Equality-generating dependencies
  - Multi-valued dependencies
  - Tuple-generating dependencies
  - Join dependencies
  - Denial constraints
  - …

- How to manage all these different types of constraints?
  - Has been shown that these constraints can be expressed in a logical formalism.
  - Formulas which consist of relational and comparison atoms. Variables represent values
    - $R(x,y,z)$
    - $x = y$

- Types of constraints we will use a lot
  - Tuple-generating dependencies ($tgds$)
    - Implication with conjunction of relational atoms
    - Foreign keys and schema mappings (later)
      $$\forall \mathbf{x} : \phi(\mathbf{x}) \rightarrow \exists \mathbf{y} : \psi(\mathbf{x}, \mathbf{y})$$
  - Equality-generating dependencies ($egds$)
    - Generalizes keys, FDs
      $$\forall \mathbf{x} : \phi(\mathbf{x}) \rightarrow \bigwedge_{k=1}^{n} x_{i_k} = x_{j_k}$$

- Other types are
  - Conditional functional dependencies
    - E.g., used in cleaning
  - Equality-generating dependencies
  - Multi-valued dependencies
  - Tuple-generating dependencies
  - Join dependencies
  - Denial constraints
    - …

- How to manage all these different types of constraints?
  - Has been shown that these constraints can be expressed in a logical formalism.
  - Formulas which consist of relational and comparison atoms. Variables represent values
    - $R(x,y,z)$
    - $x = y$

- Types of constraints we will use a lot
  - Tuple-generating dependencies ($tgds$)
    - Implication with conjunction of relational atoms
    - Foreign keys and schema mappings (later)
      $$\forall \mathbf{x} : \phi(\mathbf{x}) \rightarrow \exists \mathbf{y} : \psi(\mathbf{x}, \mathbf{y})$$
  - Equality-generating dependencies ($egds$)
    - Generalizes keys, FDs
      $$\forall \mathbf{x} : \phi(\mathbf{x}) \rightarrow \bigwedge_{k=1}^{n} x_{i_k} = x_{j_k}$$

- What is datalog?
  - Prolog for databases (syntax very similar)
  - A logic-based query language
  - Queries (Program) expressed as set of rules
    $$Q(\mathbf{x}) : R_1(x_1), \ldots, R_n(x_n).$$
  - One Q is specified as the answer relation (the relation returned by the query)
1.4 Datalog - Intuition

• A Datalog rule
  $Q(\vec{x}) : \neg R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n)$.

• For all bindings of variables in the right-hand side (RHS) that makes the RHS true (conjunction) return bindings of $\vec{x}$

Example

| Q(Name) :- Person(Name,Age). | Return names of persons |

1.4 Datalog - Syntax

• A Datalog program is a set of datalog rules
  – Optionally a distinguished answer predicate

• A Datalog rule is
  $Q(\vec{x}) : \neg R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n)$.

• $\vec{x}$’s are lists of variables and constants
• $R_i$’s are relation names
• $Q$ is a relation name

1.4 Datalog - Terminology

• Left-hand side of a rule is called it’s head
• Right-hand side of a rule is called it’s body
• Relation are called predicates
• $R(\vec{x})$ is called an atom
• An instance $I$ of a database is the data
• The active domain $\text{adom}(I)$ of an instance $I$ is the set of all constants that occur in $I$
  $Q(\vec{x}) : \neg R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n)$.

Example

<p>| N, A are variables |</p>
<table>
<thead>
<tr>
<th>N</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>peter</td>
<td>34</td>
</tr>
<tr>
<td>bob</td>
<td>45</td>
</tr>
</tbody>
</table>

Activate domain

$\text{adom}(I) = \{peter,bob,34,45\}$

1.4 Datalog - Terminology

• Intensional vs. extensional
  – Extensional database (edb)
    • What we usually call database
  – Intensional database (idb)
    • Relations that occur in the head of rules (are populated by the query)
    • Usually we assume that these do not overlap

$Q(\vec{x}) : \neg R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n)$.

1.4 Datalog - Safety

• A datalog program is safe if all its rules are safe
• A rule is safe if all variables in $\vec{x}$ occur in at least one $x_i$
  $Q(\vec{x}) : \neg R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n)$.

Example

$Q(\text{Name}) :- \text{Person}(\text{Name}, \text{Age}). \text{ (safe)}$
$Q(\text{Name}, \text{Sal}) :- \text{Person}(\text{Name}, \text{Age}). \text{ (unsafe)}$
1.4 Datalog - Semantics

- The instance of an idb predicate \( Q \) in a datalog program for an edb instance \( I \) contains all facts that can be derived by applying rules with \( Q \) in the head.
- A rule derives a fact \( Q(c) \) if we can find a binding of variables of the rule to constants from \( \text{adom}(I) \) such that \( x \) is bound to \( c \) and the body is true:
  \[
  Q(\bar{x}) : \neg R_1(\bar{x}_1), \ldots, \neg R_n(\bar{x}_n).
  \]

1.4 Datalog

- Different flavors of datalog
  - Conjunctive query
    - Only one rule
    - Expressible as Select-project-join (SPJ) query in relational algebra
  - Union of conjunctive queries
    - Also allow union
    - SPJ + set union in relational algebra
    - Rules with the same head in Datalog
  - Conjunctive queries with inequalities
    - Also allow inequalities, e.g., \(<\)

Example

\[
\begin{align*}
Q_1(x,y) : & \; R(x,y), R(x,z). \\
Q_2(x,y) : & \; R(x,y). \\
Q_3(x,x) : & \; R(x,x). \\
Q_4(x,y) : & \; R(x,y). \\
Q_5(x,y) : & \; R(x,y), R(x,x). \\
Q_6(x,x) : & \; R(x,y), R(x,x). \\
Q_7(x,z) : & \; R(x,y), R(y,z).
\end{align*}
\]

Example

Relation \( \text{hops}(A,B) \) storing edges of a graph.

\[
\begin{align*}
Q_{\text{2hop}}(x,z) : & \; \text{hop}(x,y), \text{hop}(x,z). \\
Q_{\text{reach}}(x,y) : & \; \text{hop}(x,y). \\
Q_{\text{reach}}(x,z) : & \; Q_{\text{reach}}(x,y), Q_{\text{reach}}(y,z). \\
Q_{\text{code}}(x) : & \; \text{hop}(x,y). \\
Q_{\text{code}}(x) : & \; \text{hop}(y,x).
\end{align*}
\]

1.4 Datalog - Semantics

Example

\[
Q(N) : = \text{Person}(N,A).
\]

\[
\begin{array}{c|c}
N & A \\
\hline
\text{peter} & \text{peter} \\
\text{bob} & \text{peter} \\
34 & \text{peter} \\
34 & \text{bob} \\
\end{array}
\]

Activate domain

\( \text{adom}(I) = \{ \text{peter}, \text{bob}, 34 \} \)
1.4 Datalog

Example

Relation \( \text{hops}(A, B) \) storing edges of a graph.

- \( Q_{\text{node}}(x) : \) hop \((x, y)\).
- \( Q_{\text{node}}(x) : \) hop \((y, x)\).
- \( Q_{\text{notReach}}(x, y) : \) \( Q_{\text{node}}(x) \), \( Q_{\text{node}}(y) \), not \( Q_{\text{reach}}(x, y) \).

1.4 Containment and Equivalence

Definition: Query Equivalence

Query \( Q \) is equivalent to \( Q' \) if for every database instance \( I \) both queries return the same result.

\[ Q \equiv Q' \iff \forall I : Q(I) = Q'(I) \]

Definition: Query Containment

Query \( Q \) is contained in query \( Q' \) if for every database instance \( I \) the result of \( Q \) is contained in the result of \( Q' \).

\[ Q \subseteq Q' \iff \forall I : Q(I) \subseteq Q'(I) \]

1.4 Equivalence

- The problem of checking query equivalence is of different complexity depending on the query language and whether we consider set or bag semantics.

Example

- \( Q_1(x, y) : \) \( R(x, y) \), \( R(x, z) \).
- \( Q_2(x, y) : \) hop \((x, y)\), hop \((x, z)\).
- \( Q_3(x, x) : \) \( R(x, x) \).
- \( Q_4(x, y) : \) \( R(x, y) \).
- \( Q_5(x, x) : \) \( R(x, y) \), \( R(x, x) \).
- \( Q_6(x, z) : \) \( R(x, y) \), \( R(y, z) \).

1.4 Complexity of Eq. and Cont.

<table>
<thead>
<tr>
<th>Set semantics</th>
<th>Relational Algebra</th>
<th>Conjunctive Queries (CQ)</th>
<th>Union of Conjunctive Queries (UCQ)</th>
<th>Monotone Queries/ CQ*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Evaluation (Combined Complexity)</td>
<td>PSpace-complete</td>
<td>Np-complete</td>
<td>Np-complete</td>
<td>Np-complete</td>
</tr>
<tr>
<td>Query Evaluation (Data Complexity)</td>
<td>LogSpace (that means in P)</td>
<td>LogSpace (that means in P)</td>
<td>LogSpace (that means in P)</td>
<td>LogSpace (that means in P)</td>
</tr>
<tr>
<td>Query Equivalence</td>
<td>Undecidable</td>
<td>Np-complete</td>
<td>Np-complete</td>
<td>( \Pi_2 )-complete</td>
</tr>
<tr>
<td>Query Containment</td>
<td>Undecidable</td>
<td>Np-complete</td>
<td>Np-complete</td>
<td>( \Pi_2 )-complete</td>
</tr>
</tbody>
</table>
### 1.4 Complexity of Eq. and Cont.

<table>
<thead>
<tr>
<th>Bag semantics</th>
<th>Relational Algebra</th>
<th>Conjunctive Queries (CQ)</th>
<th>Union of Conjunctive Queries (UCQ)</th>
<th>Monotone Queries/CDQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Equivalence</td>
<td>Undecidable</td>
<td>Equivalent to graph isomorphism</td>
<td>It is in PSPACE, lower-bound unknown</td>
<td></td>
</tr>
<tr>
<td>Query Containment</td>
<td>Undecidable</td>
<td>Open Problem</td>
<td>Undecidable</td>
<td>(\Pi_2) -complete</td>
</tr>
</tbody>
</table>

### 1.4 Containment Mappings

- NP-completeness for set semantics CQ and UCQ for the containment, evaluation, and equivalence problems is based on reducing these problems to the same problem
  - [Chandra & Merlin, 1977]
- Notational Conventions:
  - \(\text{head}(Q)\) = variables in head of query Q
  - \(\text{body}(Q)\) = atoms in body of Q
  - \(\text{vars}(Q)\) = all variable in Q

### 1.4 Boolean Conjunctive Queries

- A conjunctive query is boolean if the head does not have any variables
  - \(Q() :\) hop\((x,y)\), hop\((y,z)\)
  - We will use \(Q :\) \(\ldots\) as a convention for \(Q() :\) \(\ldots\)
  - What is the result of a boolean query
    - Empty result [], e.g., no hop\((x,y)\), hop\((y,z)\)
    - If there are tuples matching the body, then a tuple with zero attributes is returned [0]
- -> We interpret {} as \textit{false} and {()} as \textit{true}
- Boolean query is essentially an existential check

### 1.4 Boolean Conjunctive Queries in SQL

- **BCQ in SQL**

  - **Example**
    ```sql
    Hop relation: Hop\((A,B)\)
    Q :\) hop\((x,y)\)
    SELECT EXISTS (SELECT * FROM hop)
    ```
  - Note: in Oracle and DB2 we need a from clause

- **Example**
  ```sql
  SELECT CASE WHEN EXISTS (SELECT * FROM hop) THEN 1 ELSE 0 END AS x
  FROM dual;
  ```
  - Notes:
    - Oracle and DB2 FROM not optional
    - Oracle has no boolean datatype

- **Example**
  ```sql
  Q :\) hop\((x,y)\), hop\((y,z)\)
  SELECT EXISTS (SELECT * FROM hop l, hop r WHERE l.B = r.A)
  ```
1.4 Containment Mappings

- How to check for containment of CQs (set)

**Definition: Variable Mapping**
A variable mapping $\psi$ from query $Q$ to query $Q'$ maps the variables of $Q$ to constants or variables from $Q'$.

**Definition: Containment Mapping**
A containment mapping from query $Q$ to $Q'$ is a variable mapping $\psi$ such that:

$$
\psi(\text{head}(Q)) = \text{head}(Q')
$$

$$
\forall R(x_i) \in \text{body}(Q) : \psi(x_i) \in \text{body}(Q')
$$

Theorem: Containment Mapping and Query Containment

Query $Q$ is contained in query $Q'$ if there exists a containment mapping $\psi$ from $Q'$ to $Q$.

Example

$Q_1(u, z) : R(u, z)$.
$Q_2(x, y) : R(x, y)$.

Can we find a containment mapping?

Example

$Q_1(a, b) : R(a, b), R(c, b)$.
$Q_2(x, y) : R(x, y)$.

Do containment mappings exist?

$Q_1 \rightarrow Q_2 : \text{none exists}$
$Q_2 \rightarrow Q_1 : \psi(x) = a, \psi(y) = b$
1.4 Containment Background

- It was shown that query evaluation, containment, equivalence as all reducible to homomorphism checking for CQ
  - Canonical conjunctive query $Q_I$ for instance I
    - Interpret attribute values as variables
    - The query is a conjunction of all atoms for the tuples
    - $I = \{\text{hop}(a,b), \text{hop}(b,c)\} \rightarrow Q_I = \text{hop}(a,b), \text{hop}(b,c)$
  - Canonical instance $I^Q$ for query $Q$
    - Interpret each conjunct as a tuple
    - Interpret variables as constants
    - $Q = \text{hop}(a,a) \rightarrow I^Q = \{\text{hop}(a,a)\}$

1.4 Containment Mappings

**Example**

$Q_1() : R(a,b), R(c,b)$.  
$Q_2() : R(x,y)$.  
$Q_2 \rightarrow Q_1 : \psi(x)=a, \psi(y)=b$

$D = \{R(1,1), R(1,2)\}$

$Q_1(D) = \{(1,1), (1,2)\}$

$\phi(x)=1, \phi(b)=2, \phi(c)=1$

$\psi \phi(x)=1, \psi \phi(y)=2$

$1^Q = \{(a,b), (c,b)\}$

$Q_2(1^Q) = \{(\)\}$

$\phi(x)=a, \phi(y)=b$

$\phi$ is our containment mapping $\psi$

1.4 Containment Mappings

**Example**

$Q_1() : R(a,b), R(c,b)$.  
$Q_2() : R(x,y)$.  
$Q_2 \rightarrow Q_1 : \psi(x)=a, \psi(y)=b$

$I_{Q_2}^Q = \{(a,b), (c,b)\}$

$Q_2(I_{Q_2}^Q) = \{(\)\}$

$\phi(x)=a, \phi(y)=b$

$\phi$ is our containment mapping $\psi$

1.4 Containment Background

- Containment Mapping $\leftrightarrow$ Containment
- Proof idea (boolean queries)
  - (if direction)
    - Assume we have a containment mapping $Q_1$ to $Q_2$
    - Consider database $D$
    - $Q_2(D)$ is true then we can find a mapping from $\text{vars}(Q_2)$ to $D$
    - Compose this with the containment mapping and prove that this is a result for $Q_1$

1.4 Containment Background

- If you are not scared and want to know more:
  - Look up Chandra and Merlins paper(s)
  - The text book provides a more detailed overview of the proof approach
  - Look at the slides from Phokion Kolaitis excellent lecture on database theory
    - [https://classes.soee.ucsc.edu/cmps277/Winter10/](https://classes.soee.ucsc.edu/cmps277/Winter10/)
1.4 Containment Background

- A more intuitive explanation why containment mappings work
  - Variable naming is irrelevant for query results
  - If there is a containment mapping \( Q \) to \( Q' \)
    - Then every condition enforced in \( Q \) is also enforced by \( Q' \)
    - \( Q' \) may enforce additional conditions

1.4 Containment Mappings

Example

\[
Q_1(a) : R(a,b), R(c,b) .
Q_2(x) : R(x,y) .
Q_2 \rightarrow Q_1 : \Psi(x)=a, \Psi(y)=b
\]

If there exists tuples \( R(a,b) \) and \( R(c,b) \) in \( R \) that make \( Q_1 \) true, then we take \( R(a,b) \) to fulfill \( Q_2 \)

1.4 Containment Background

- From boolean to general conjunctive queries
  - Instead of returning true or false, return bindings of variables
  - Recall that containment mappings enforce that the head is mapped to the head
  - \( \rightarrow \) same tuples returned, but again \( Q' \)’s condition is more restrictive

1.4 Containment Mappings

Example

\[
Q_1(a) : R(a,b), R(c,b) .
Q_2(x) : R(x,y) .
Q_2 \rightarrow Q_1 : \Psi(x)=a, \Psi(y)=b
\]

For every \( R(a,b) \) and \( R(c,b) \)
\( Q_1 \) returns \( (a,b) \) and for every \( R(a,b) \)
\( Q_2 \) returns \( a \)

1.4 Similarity Measures

- Problem faced by multiple integration tasks
  - Given two objects, how similar are they
  - E.g., given two attribute names in schema matching, given two values in data fusion/entity resolution, ...

1.4 Similarity Measures

- Object models
  - Multidimensional (feature vector model)
    - Object is described as a vector of values - one for each dimension out of a given set of dimensions
    - E.g., Dimensions are gender (male/female), age (0-120), and salary (0-1,000,000). An example object is [male, 80, 70,000]
  - Strings
    - E.g., how similar is “Poeter” to “Peter”
  - Graphs and Trees
    - E.g., how similar are two XML models
1.4 Similarity Measures

Definition: Similarity Measure

Function $d(p,q)$ where $p$ and $q$ are objects, that returns a real score with

- Interpretation: the lower the score the "more similar" the objects are
- We require $d(p,p)=0$, because nothing can be more similar to an object than itself
- Note: often scores are normalized to the range $[0,1]$

Example

String equality: $d(p,q) = 0$ if $p=q$

Strings $d(p,q) = 1$ else

Euclidean distance: $d(p,q) = \sqrt{\sum_{i=1}^{m} (p[i]-q[i])^2}$

Edit distance: $d(p,q) = \text{minimum number of single character insertions, deletions, replacements to transform } p \text{ into } q$

Definition: Metric

Function $d(p,q)$ where $p$ and $q$ are objects, that returns a real score with

- Non-negative $d(p,q) \geq 0$
- Symmetry $d(p,q) = d(q,p)$
- Identity of indiscernibles $d(p,q) = 0$ if $p=q$
- Triangle inequality $d(p,q) + d(q,r) \geq d(p,r)$

Metric is a stricter definition

Which of the previous similarity measure is a metric?

- All of them!

Summary

- Heterogeneity
  - Types of heterogeneity
  - Why do they arise?
  - Hint at how to address them
- Autonomy
- Data Integration Tasks
- Data Integration Architectures
- Background
  - Datalog + Query equivalence/containment + Similarity + Integrity constraints

Why do we care whether $d$ is a metric?

- Some data mining algorithms only work for metrics
  - E.g., some clustering algorithms such as k-means
  - E.g., clustering has been used in entity resolution
- Metric spaces allow optimizations of some methods
  - E.g., Nearest Neighborhood-search: find the most similar object to an object $p$. This problem can be efficiently solved using index structures that only apply to metric spaces
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
2. Data Preparation and Cleaning

Boris Glavic

http://www.cs.iit.edu/~glavic/

http://www.cs.iit.edu/~cs520/

http://www.cs.iit.edu/~dbgroup/

IIT DBGroup

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

2. Overview

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

2. Causes of “Dirty” Data

• Manual data entry or result of erroneous integration
  – Typos:
    • “Peter” vs. “Pteer”
  – Switching fields
  – Incorrect information
  – Missing information

• 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, …

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
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
    - \( \_ \rightarrow \text{finer-grained control, e.g., zip} \rightarrow \text{city only if country is US} \)
- Constraints that consider master data
  - Master data is highly reliable data such as a government-issued zip, city lookup table

2.1 Example Constraints

<table>
<thead>
<tr>
<th>SSN</th>
<th>zip</th>
<th>city</th>
<th>name</th>
<th>boss</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>New York</td>
<td>Peter</td>
<td>Gert</td>
<td>50,000</td>
</tr>
<tr>
<td>333-333-9999</td>
<td>60615</td>
<td>Chicago</td>
<td>Gert</td>
<td>NULL</td>
<td>40,000</td>
</tr>
<tr>
<td>333-333-5599</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gertrud</td>
<td>Hans</td>
<td>10,000</td>
</tr>
<tr>
<td>333-333-6666</td>
<td>60616</td>
<td>Chicago</td>
<td>Hans</td>
<td>NULL</td>
<td>1,000,000</td>
</tr>
<tr>
<td>333-355-4343</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
<td>Hans</td>
<td>20,000</td>
</tr>
</tbody>
</table>

\( \forall \overrightarrow{x} : \neg (\phi(\overrightarrow{x})) \)

- 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**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Zip</th>
<th>City</th>
<th>Name</th>
<th>Boss</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>New York</td>
<td>Peter</td>
<td></td>
<td>50,000</td>
</tr>
<tr>
<td>333-333-9999</td>
<td>60615</td>
<td>Chicago</td>
<td>Gert</td>
<td></td>
<td>40,000</td>
</tr>
<tr>
<td>333-333-5599</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gertrud</td>
<td>Hans</td>
<td>10,000</td>
</tr>
<tr>
<td>333-333-6666</td>
<td>60616</td>
<td>Chicago</td>
<td>Hans</td>
<td>NULL</td>
<td>1,000,000</td>
</tr>
<tr>
<td>333-355-4343</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
<td>Hans</td>
<td>20,000</td>
</tr>
</tbody>
</table>

\[ x = x' \land y = y' \land z = z' \land u = u' \land v = v' \land w = w' \]

\[ x = x' \land y = y' \land z = z' \land u = u' \land v = v' \land w = w' \]

\[ x = x' \land y = y' \land z = z' \land u = u' \land v = v' \land w = w' \]

\[ x = x' \land y = y' \land z = z' \land u = u' \land v = v' \land w = w' \]

\[ x = x' \land y = y' \land z = z' \land u = u' \land v = v' \land w = w' \]

\[ x = x' \land y = y' \land z = z' \land u = u' \land v = v' \land w = w' \]

**Example Constraints Languages**

- **FD 1**: The zip code uniquely determines the city
- **FD 2**: Nobody should earn more than their direct superior
- **FD 3**: Salaries are non-negative

### 2.1 Constraint based Cleaning Overview

**Explanation:** 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

**1. 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

<table>
<thead>
<tr>
<th>SSN</th>
<th>Zip</th>
<th>City</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>New York</td>
<td>Peter</td>
</tr>
<tr>
<td>333-333-9999</td>
<td>60615</td>
<td>Chicago</td>
<td>Gert</td>
</tr>
<tr>
<td>333-333-5599</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gertrud</td>
</tr>
<tr>
<td>333-333-6666</td>
<td>60616</td>
<td>Chicago</td>
<td>Hans</td>
</tr>
<tr>
<td>333-355-4343</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
</tr>
</tbody>
</table>

**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)

**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

**2.1 Constraint based Cleaning Overview**

- 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

**2.1 Detecting Violations**

- Given FD A -> B on R
  - Recall logical representation
  - forall X, X': R(X) and R(X') and A=A' -> B=B'
  - Only violated if we find two tuples where A=A', but B != B'
  - In datalog
    - QO: R(X), R(X'), A=A', B=B'
  - In SQL
    - SELECT EXISTS (SELECT *
                         FROM R x, R y
                         WHERE A=A' AND B>B')
2.1 Example Constraints

Example: SQL Violation Detection

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

FD: `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 \( \Sigma \) and a database instance \( I \) which violates the constraints find a clean instance \( I' \) (does not violate the constraints) with \( \text{cost}(I,I') \) 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
    - 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:
      \[ \sum_{t \in R} \sum_{A(\text{Schema}(R))} d(t.A, t'.A) \]

2.1 Cost Metrics

- Deletion + Insertion
  - 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:
    \[ \sum_{t \in R} \sum_{A(\text{Schema}(R))} d(t.A, t'.A) \]

2.1 Naïve FD Repair Algorithm

- FD Repair Algorithm: 1. Attempt
  - For each FD \( X \Rightarrow Y \) in \( \Sigma \) 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

<table>
<thead>
<tr>
<th>SSN</th>
<th>Zip</th>
<th>City</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>New York</td>
<td>Peter</td>
</tr>
<tr>
<td>333-333-9999</td>
<td>60615</td>
<td>Chicago</td>
<td>Gert</td>
</tr>
<tr>
<td>333-333-5599</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gertrud</td>
</tr>
<tr>
<td>333-333-6666</td>
<td>60616</td>
<td>Chicago</td>
<td>Hans</td>
</tr>
<tr>
<td>333-355-4343</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
</tr>
</tbody>
</table>

2.1 Problems with the Algorithm

• FD Repair Algorithm: 1. Attempt
  – For each FD \( X \rightarrow Y \) in \( \Sigma \) 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 \( t.Y \neq t'.Y \)
    • update \( t.Y \) to \( t'.Y \)
      – choice does not matter because cost is symmetric, right?
    – Our updates may cause new violations!

2.1 Problems with the Algorithm

• FD Repair Algorithm: 2. Attempt
  – \( I' = I \)
    – 1) For each FD \( X \rightarrow Y \) in \( \Sigma \) 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 \( t.Y \neq t'.Y \)
      • 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

<table>
<thead>
<tr>
<th>SSN</th>
<th>Zip</th>
<th>City</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>New York</td>
<td>Peter</td>
</tr>
<tr>
<td>333-333-9999</td>
<td>60615</td>
<td>Chicago</td>
<td>Gert</td>
</tr>
<tr>
<td>333-333-5599</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gertrud</td>
</tr>
<tr>
<td>333-333-6666</td>
<td>60616</td>
<td>Chicago</td>
<td>Hans</td>
</tr>
<tr>
<td>333-355-4343</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
</tr>
</tbody>
</table>

2.1 Problems with the Algorithm

• FD Repair Algorithm: 2. Attempt
  – \( I' = I \)
    – 1) For each FD \( X \rightarrow Y \) in \( \Sigma \) 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 \( t.Y \neq t'.Y \)
      • 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 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!

- 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 Constraint Repair

Example: Constraint Repair

<table>
<thead>
<tr>
<th>SSN</th>
<th>zip</th>
<th>city</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>New York</td>
<td>Peter</td>
</tr>
<tr>
<td>333-333-9999</td>
<td>60615</td>
<td>Chicago</td>
<td>Gert</td>
</tr>
<tr>
<td>333-333-5599</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gertrud</td>
</tr>
<tr>
<td>333-333-6666</td>
<td>60616</td>
<td>Chicago</td>
<td>Hans</td>
</tr>
<tr>
<td>333-355-4343</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>id</th>
<th>age</th>
<th>city</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>23</td>
<td>Chicago</td>
<td>Peter</td>
</tr>
<tr>
<td>002</td>
<td>30</td>
<td>New York</td>
<td>Gert</td>
</tr>
<tr>
<td>003</td>
<td>27</td>
<td>Schaumburg</td>
<td>Gertrud</td>
</tr>
<tr>
<td>004</td>
<td>32</td>
<td>Chicago</td>
<td>Hans</td>
</tr>
<tr>
<td>005</td>
<td>29</td>
<td>Chicago</td>
<td>Malcom</td>
</tr>
</tbody>
</table>

Changes: t1.city = Chicago
Not no change: set t4.city and t5.city = New York

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

2.3 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

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

<table>
<thead>
<tr>
<th>SSN</th>
<th>Zip</th>
<th>City</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>Chicago</td>
<td>Peter</td>
</tr>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>Chicago</td>
<td>Peter</td>
</tr>
</tbody>
</table>

2.3 Entity Resolution

- 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

2.3 Entity Resolution

- E.g., foreign key connections

2.3 Entity Resolution

- E.g., fixed similarity threshold

2.3 Entity Resolution

- How to guarantee transitivity of E

2.3 Entity Resolution

- E.g., do this afterwards

2.3 Entity Resolution

Example: Per attribute similarity

<table>
<thead>
<tr>
<th>SSN</th>
<th>Zip</th>
<th>City</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>Chicago</td>
<td>Peter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.3 Entity Resolution

- Distance Measures

- 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)

2.3 Entity Resolution

Example:

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 \(s'\).

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 \(l(s) + l(s')\), in this case 10

**NEED -> STREET**

Minimal solution:

- Insert T
- Replace E with R
- Replace U with T

\(d(\text{NEED}, \text{STREET}) = 4\)
2.3 Entity Resolution

- **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'
  - \( D(\text{len}(s), \text{len}(s')) \) is the solution
  - Represented as matrix
  - Populate based on rules shown on the next slide

**Recursive definition**
- \( D(i,0) = i \)
  - Cheapest way of transforming prefix \( s[j] \) 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 \( s[i] \neq s[j] \) and 0 else

Example:

\[
\begin{array}{cccccc}
\text{S} & \text{T} & \text{R} & \text{E} & \text{E} & \text{T} \\
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\text{N} & 1 &   &   &   &   &   \\
\text{E} & 2 &   &   &   &   &   \\
\text{E} & 3 &   &   &   &   &   \\
\text{D} & 4 &   &   &   &   &   \\
\end{array}
\]

Example:

\[
\begin{array}{cccccc}
\text{S} & \text{T} & \text{R} & \text{E} & \text{E} & \text{T} \\
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\text{N} & 1 & 1 &   &   &   &   \\
\text{E} & 2 & 2 &   &   &   &   \\
\text{E} & 3 &   &   &   &   &   \\
\text{D} & 4 &   &   &   &   &   \\
\end{array}
\]

Example:

\[
\begin{array}{cccccc}
\text{S} & \text{T} & \text{R} & \text{E} & \text{E} & \text{T} \\
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\text{N} & 1 & 1 & 2 &   &   &   \\
\text{E} & 2 & 2 & 2 &   &   &   \\
\text{E} & 3 &   &   &   &   &   \\
\text{D} & 4 &   &   &   &   &   \\
\end{array}
\]

Example:

\[
\begin{array}{cccccc}
\text{S} & \text{T} & \text{R} & \text{E} & \text{E} & \text{T} \\
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\text{N} & 1 & 1 & 2 & 3 &   &   \\
\text{E} & 2 & 2 & 2 & 2 &   &   \\
\text{E} & 3 &   &   &   &   &   \\
\text{D} & 4 &   &   &   &   &   \\
\end{array}
\]
2.3 Entity Resolution

Example:

<table>
<thead>
<tr>
<th>NEED -&gt; STREET</th>
</tr>
</thead>
<tbody>
<tr>
<td>S 7 R E E E Z</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6</td>
</tr>
<tr>
<td>N 1 1 2 3 4</td>
</tr>
<tr>
<td>E 2 2 2 3</td>
</tr>
<tr>
<td>E 3 3 3 3</td>
</tr>
<tr>
<td>D 4 4 4 4</td>
</tr>
</tbody>
</table>

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.

• 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

\[
d_{jacc}(S, S') = \frac{|B_S \cap B_{S'}|}{|B_S \cup B_{S'}|}
\]

Example:

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

- **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
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
  - Compute 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**
  - 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 \( i \)th attribute
  - Tuples match if \( d(t, t') > \beta \) for a threshold \( \beta \)

Example: Weighted sum of attribute similarities

<table>
<thead>
<tr>
<th>SSN</th>
<th>age</th>
<th>city</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>08</td>
<td>Chicago</td>
<td>Peter</td>
</tr>
<tr>
<td>3333333333</td>
<td>07</td>
<td>IL</td>
<td>Peta</td>
</tr>
</tbody>
</table>

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

- \( w_{SSN} = 0.4, w_{age} = 0.05, w_{city} = 0.15, w_{name} = 0.4 \)
- \( d(t, t') = 0.4 \times 1 + 0.05 \times 0.8 + 0.15 \times 0 + 0.4 \times 0.6 \)
  - = 0.68

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

**Rule-based approach**
- Collection (list) of rules
- If \( d_{name}(t, t') < 0.6 \) then unmatched
- If \( d_{zip}(t, t') = 1 \) and \( t\.country = \text{USA} \) then matched
- If \( t\.country \neq t'\.country \) then unmatched

**Advantages**
- Easy to start, can be incrementally improved

**Disadvantages**
- Lot of manual work, large rule-bases hard to understand

**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

**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
2.3 Entity Resolution

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

2.4 Data Fusion

- 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

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
3. Why matching and mapping?

- **Problem:** Schema Heterogeneity
  - Sources with different schemas store overlapping information
  - Want to be able to translate data from one schema into a different schema
    - Datawarehousing
    - Data exchange
  - Want to be able to translate queries against one schema into queries against another schema
    - Virtual data integration

- **Why both mapping and matching**
  - Split complex problem into simpler subproblems
    - Determine matches and then correlate with constraint information into mappings
  - Some tasks only require matches
    - E.g., matches can be used to determine attributes storing the same information in data fusion
  - Mappings are naturally a generalization of matchings

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

3. Why matching and mapping?

- **Problem:** Schema Heterogeneity
  - We need to know how elements of different schemas are related!
    - Schema matching
      - Simple relationships such as attribute name of relation person in the one schema corresponds to attribute lastname of relation employee in the other schema
    - Schema mapping
      - Also model correlations and missing information such as links caused by foreign key constraints

3. Overview

- **Topics covered in this part**
  - Schema Matching
  - Schema Mappings and Mapping Languages
3.1 Schema Matching

- **Problem: Schema Matching**
  - Given two (or more schemas)
  - For now called source and target
  - Determine how elements are related
    - Attributes are representing the same information
      - name = lastname
    - Attribute can be translated into an attribute
      - MonthlySalary * 12 = Yearly Salary
    - 1-1 matches vs. M-N matches
      - name to lastname
      - name to concat(firstname, lastname)

- **Why is this hard?**
  - Insufficient information: schema does not capture full semantics of a domain
  - Schemas can be misleading:
    - E.g., attributes are not necessarily descriptive
    - E.g., finding the right way to translate attributes not obvious

- **What information to consider?**
  - Attribute names
    - or more generally element names
  - Structure
    - e.g., belonging to the same relation
  - Data
    - Not always available
  - Need to consider multiple types to get reasonable matching quality
    - Single types of information not predictable enough

3.1 Schema Mapping

**Example: Types of Matching**

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Office-phone</th>
<th>Office-address</th>
<th>Home-phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>1</td>
<td>(312) 123 456</td>
<td>Chicago, IL</td>
<td>(333) 123 456</td>
</tr>
<tr>
<td>Alice</td>
<td>3</td>
<td>(312) 555 789</td>
<td>Chicago, IL</td>
<td>(123) 321 678</td>
</tr>
<tr>
<td>Bob</td>
<td>3</td>
<td>(465) 123 456</td>
<td>New York, NY</td>
<td>(888) 321 456</td>
</tr>
</tbody>
</table>

Based on element names we could match Office-contact to both Office-phone and Office-address.

Based on data we could match Office-contact to both Office-phone and Home-phone:

- **Typical Matching System Architecture**
  - Determine actual matches
  - Use constraints to modify similarity matrix
  - Combine individual similarity matrices
  - Each matcher uses one type of information to compute similarity matrix
3.1 Schema Matching

- **Matcher**
  - **Input:** Schemas
  - Maybe also data, documentation
  - **Output:** Similarity matrix
  - Storing value [0,1] for each pair of elements from the source and the target schema

3.1 Schema Mapping

Example: Types of Matching

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Office-phone</th>
<th>Office-address</th>
<th>Home-phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Address</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>Id</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>City</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Office-contact</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

3.1 Schema Matching

- **Name-Based Matchers**
  - String similarities measures
    - E.g., Jaccard and other measure we have discussed
  - **Preprocessing**
    - Tokenization?
    - Normalization
      - Expand abbreviations and replace synonyms
    - Remove stop words
      - In, and, the

3.1 Schema Matching

- **Data-Based Matchers**
  - Determine how similar the values of two attributes are
  - Some techniques
    - **Recognizers**
      - Dictionaries, regular expressions, rules
    - **Overlap matcher**
      - Compute overlap of values in the two attributes
    - **Classifiers**

3.1 Schema Matching

- **Recognizers**
  - Dictionaries
    - Countries, states, person names
  - Regular expression matchers
    - **Phone numbers:** (\+\d{2})? \(\d{3}\) \d{3} \d{4}

3.1 Schema Mapping

- **Overlap of attribute domains**
  - Each attribute value is a token
  - Use set-based similarity measure such as Jaccard

- **Classifier**
  - Train classifier to identify values of one attribute A from the source
    - Training set are values from A as positive examples and values of other attributes as negative examples
  - Apply classifier to all values of attributes from target schema
    - Aggregate into similarity score
3.1 Schema Matching

- **Combiner**
  - **Input:** Similarity matrices
    - Output of the individual matchers
  - **Output:** Single Similarity matrix

- **Constraint Enforcer**
  - **Input:** Similarity matrix
    - Output of Combiner
  - **Output:** Similarity matrix

**Example: Constraints**

- Constraint 1: An attribute matched to `source.cust-phone` has to get a score of 1 from the phone regexpr matcher.
- Constraint 2: Any attribute matched to `source.fax` has to have fax in its name.
- Constraint 3: If an attribute is matched to `source.firstname` with score > 0.9 then there has to be another attribute from the same target table that is matched to `source.lastname` with score > 0.9.

**How to search match combinations**

- Full search
  - Exponentially many combinations potentially
- Informed search approaches
  - A* search
  - Local propagation
  - Only local optimizations
3.1 Schema Matching

- **A* search**
  - Given a search problem
  - Set of states: start state, goal states
  - Transitions about states
  - Costs associated with transitions
  - Find cheapest path from start to goal states
  - Need admissible heuristics \( h \)
    - For a path \( p \), \( h \) computes lower bound for any path from start to goal with prefix \( p \)
  - Backtracking best-first search
    - Choose next state with lowest estimated cost
    - Expand it in all possible ways

- **Algorithm**
  - Data structures
    - Keep a priority queue \( q \) of states sorted on \( f(n) \)
      - Initialize with start state
    - Keep set \( v \) of already visited nodes
      - Initially empty
  - While \( q \) is not empty
    - pop state \( s \) from head of \( q \)
    - If \( s \) is goal state return
    - Foreach \( s' \) that is direct neighbor of \( s \)
      - If \( s' \) not in \( v \)
        - Compute \( f(s') \) and insert \( s' \) into \( q \)

- **Application to constraint enforcing**
  - Source attributes: \( A_1 \) to \( A_n \)
  - Target attributes: \( B_1 \) to \( B_m \)
  - States
    - Vector of length \( n \) with values \( B_i \) or * indicating that no choice has not been taken
    - Initial state
      - \([*, *, *, *]\)
    - Goal states
      - All states without *

- **Match Selector**
  - **Input**: Similarity matrix
    - Output of the individual matchers
  - **Output**: Matches

- **Match Selection**
  - Merge similarity matrices produced by the matchers into single matrix
  - Typical strategies
    - Average, Minimum, Max
    - Weighted combinations
    - Some script
3.1 Schema Matching

- **Many-to-many matchers**
  - Combine multiple columns using a set of functions
    - E.g., concat, +, currency exchange, unit exchange
  - Large or even unlimited search space
  - Need method that explores interesting part of the search space
  - Specific searchers
    - Only concatenation of columns (limit number of combinations, e.g., 2)

3.2 Schema Mapping

**Example: Matching Result**

<table>
<thead>
<tr>
<th>Person</th>
<th>Name</th>
<th>Address</th>
<th>Id</th>
<th>City</th>
<th>Office-contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pete</td>
<td>Peter</td>
<td>Chicago</td>
<td>1</td>
<td>Chicago</td>
<td>(312) 123-4567</td>
</tr>
<tr>
<td>Alice</td>
<td>Alice</td>
<td>Chicago</td>
<td>2</td>
<td>Chicago</td>
<td>(312) 555-7890</td>
</tr>
<tr>
<td>Bob</td>
<td>Bob</td>
<td>New York</td>
<td>3</td>
<td>New York</td>
<td>(999) 123-4560</td>
</tr>
</tbody>
</table>

Assume: We have data in the source as shown above

What data should we create in the target? Copy values based on matches?

3.2 Schema Mapping

- Matches do not determine completely how to create the target instance data! (Data Exchange)
  - How do we choose values for attributes that do not have a match?
  - How do we combine data from different source tables?
- Matches do not determine completely what the answers to queries over a mediated schema should be! (Virtual Data Integration)

3.2 Schema Mapping

- **Schema mappings**
  - Generalize matches
  - Describe relationship between instances of schemas
  - Mapping languages
    - LAV, GAV, GLAV
    - Mapping as Dependencies: tuple-generating dependencies

- **Mapping generation**
  - Input: Matches, Schema constraints
  - Output: Schema mappings
3.2 Schema Mapping

- **Instance-based definition of mappings**
  - Global schema $G$
  - Local schemas $S_1$ to $S_n$
  - Mapping $M$ can be expressed as for each set of instances of the local schemas what are allowed instances of the global schema
    - Subset of $(I_G \times I_1 \times \ldots \times I_n)$
  - Useful as a different way to think about mappings, but not a practical way to define mappings

- **Languages for Specifying Mappings**
  - Describing mappings as inclusion relationships between views:
    - Global as View ($GAV$)
    - Local as View ($LAV$)
    - Global and Local as View ($GLAV$)
  - Describing mappings as dependencies
    - Source-to-target tuple-generating dependencies ($st$-$tgds$)

- **Excursion Virtual Data Integration**
  - More in next section of the course

- **Certain answers**
  - Given mapping $M$ and $Q$
  - Instances $I_1$ to $I_n$ for $S_1$ to $S_n$
  - Tuple $t$ is a certain answer for $Q$ over $I_1$ to $I_n$
    - If for every instance $I_G$ so that $(I_G \times I_1 \times \ldots \times I_n)$ in $M$
    - $t \in Q(I_G)$

- **Describing mappings as inclusion relationships between views:**
  - Global as View ($GAV$)
  - Local as View ($LAV$)
  - Global and Local as View ($GLAV$)
  - Terminology stems from virtual integration
    - Given a global (or mediated, or virtual) schema
    - A set of data sources (local schemas)
    - Compute answers to queries written against the global schema using the local data sources

- **Global-as-view ($GAV$)**
  - Express the global schema as views over the local schemata
  - What query language do we support?
    - CQ, UCQ, SQL, …?
  - Closed vs. open world assumption
    - Closed world: $R = Q(S_1, \ldots, S_n)$
      - Content of global relation $R$ is defined as the result of query $Q$ over the sources
    - Open world: $R \supseteq Q(S_1, \ldots, S_n)$
      - Relation $R$ has to contain the result of query $Q$, but may contain additional tuples
3.2 Schema Mapping

Example: Types of Matching

<table>
<thead>
<tr>
<th>Local Schema</th>
<th>Global Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Person</td>
</tr>
<tr>
<td>Address</td>
<td>Address</td>
</tr>
<tr>
<td>Office-contact</td>
<td>Office-address</td>
</tr>
</tbody>
</table>

Person(X',Y',Z',A',B') = Q(X,Z,A, NULL, NULL) :- Person(X,Y), Address(Y,Z,A)

Since heads of LHS and RHS have to be the same we can use simpler notation without the head of the view expression:

Person(X,Z,A, NULL, NULL) = Person(X,Y), Address(Y,Z,A)

Example: Types of Matching

Consider switching local and global schema

Person(X,NULL) = Person(X,Y,Z,A,B)
Address(NULL,Y,Z) = Person(X,Y,Z,A,B)

3.2 Schema Mapping

- **Global-as-view (GAV)**
- **Solutions (mapping M)**
  - Unique solutions (1 solution!)
  - Intuitively, execute queries over local instance that produced global instance

Example: Types of Matching

<table>
<thead>
<tr>
<th>Local Schema</th>
<th>Global Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Person</td>
</tr>
<tr>
<td>Address</td>
<td>Address</td>
</tr>
<tr>
<td>Office-contact</td>
<td>Office-address</td>
</tr>
</tbody>
</table>

Person(X',Y',Z',A',B') = Q(X,Z,A, NULL, NULL) :- Person(X,Y), Address(Y,Z,A)

Example: Types of Matching

Consider switching local and global schema

Person(X,NULL) = Person(X,Y,Z,A,B)
Address(NULL,Y,Z) = Person(X,Y,Z,A,B)

3.2 Schema Mapping

- **Global-as-view (GAV)**
- **Answering Queries**
  - Simply replace references to global tables with the view definition

Example: Types of Matching

<table>
<thead>
<tr>
<th>Local Schema</th>
<th>Global Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Person</td>
</tr>
<tr>
<td>Address</td>
<td>Address</td>
</tr>
<tr>
<td>Office-contact</td>
<td>Office-address</td>
</tr>
</tbody>
</table>

Person(X,NULL) = Person(X,Y,Z,A,B)
Address(NULL,Y,Z) = Person(X,Y,Z,A,B)

3.2 Schema Mapping

- **Global-as-view (GAV)**
- **Local-as-view (LAV)**
  - Express the local schema as views over the global schemata
  - What query language do we support?
    - CQ, UCQ, SQL, ...?
  - **Closed vs. open world** assumption
    - Closed world: $S_q = Q(G)$
      - Content of local relation $S_q$ is defined as the result of query $Q$ over the sources
    - Open world: $S_q \supseteq Q(G)$
      - Local relation $S_q$ has to contain the result of query $Q$, but may contain additional tuples

Example: Types of Matching

<table>
<thead>
<tr>
<th>Local Schema</th>
<th>Global Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Person</td>
</tr>
<tr>
<td>Address</td>
<td>Address</td>
</tr>
<tr>
<td>Office-contact</td>
<td>Office-address</td>
</tr>
</tbody>
</table>

Person(X,NULL) = Person(X,Y,Z,A,B)
Address(NULL,Y,Z) = Person(X,Y,Z,A,B)
3.2 Schema Mapping

- **Local-as-view (LAV)**
  - **Solutions (mapping M)**
    - May be many solutions

- **Answering Queries**
  - Need to find equivalent query using only the views (this is a hard problem, more in next course section)
  - Mapping \( S(X,Z) = R(X,Y), T(Y,Z) \)
  - \( Q(X) : R(X,Y) \)
  - Rewrite into ???
    - Need to come up with missing values
    - Give up query equivalence?

- **Global-Local-as-view (GLAV)**
  - Express both sides of the constraint as queries
  - What query language do we support?
    - CQ, UCQ, SQL, ...?
  - Closed vs. open world assumption
    - Closed world: \( Q'(G) = Q(S) \)
    - Open world: \( Q'(G) \not\equiv Q(S) \)

Example: Types of Matching

Local Schema

- Person
  - Name
  - Address
- Address
  - City
  - Office-contact

Global Schema

- Person
  - Name
  - Address
  - Office-phone
  - Office-address
  - Home-phone

Source: \( Q(X,Y,Z) : Person(X',Y'), Address(Y',Z',A') \)
Target: \( Q(X',Y',Z') : Person(X',Y',Z',A',B') \)
3.2 Schema Mapping

- **Local-as-view (GLAV) Discussion**
  - Kind of best of both worlds (almost)
  - Complexity of query answering is the same as for LA V
  - Can address the lost correlation and missing values problems we observed using GAV and LAV

- **Source-to-target tuple-generating dependencies (st-tgds)**
  - Local way of expressing GLAV mappings
    \[ \forall x : \phi(x) \rightarrow \exists y : \psi(x, y) \]
    - Equivalence to a containment constraint:
      \[ Q'(G) \supseteq Q(S) \]

- **Generating Schema Mappings**
  - **Input**: Schemas (Constraints), matches
  - **Output**: Schema mappings
  - **Ideas**:
    - Schema matches tell us which source attributes should be copied to which target attributes
    - Foreign key constraints tell us how to join in the source and target to not lose information

- **Clio**
  - Clio is a data exchange system prototype developed by IBM and University of Toronto researchers
  - The concepts developed for Clio have been implemented in IBM InfoSphere Data Architect
  - Clio does matching, mapping generation, and data exchange
    - For now let us focus on the mapping generation
3.2 Schema Mapping

• Clio Algorithm Steps
  – 1) Use foreign keys to determine all reasonable ways of joining data within the source and the target schema
    • Each alternative of joining tables in the source/target is called a logical association
  – 2) For each pair of source-target logical associations: Correlate this information with the matches to determine candidate mappings

3.2 Schema Mapping

• Clio Algorithm: 1) Find logical associations
  – This part relies on the chase procedure that first introduced to test implication of functional dependencies (’77)
  – The idea is that we start use a representation of foreign keys are inclusion dependencies (tgds)
    • There are also chase procedures that consider edges (e.g., PKs)
  – Starting point are all single relational atoms
    • E.g., R(X,Y)

3.2 Schema Mapping

• Chase step
  – Works on tableau: set of relational atoms
  – A chase step takes one tgd t where the LHS is fulfilled and the RHS is not fulfilled
    • We fulfill the tgd t by adding new atoms to the tableau and mapping variables from t to the actually occuring variables from the current tableau

• Chase
  – Applying the chase until no more changes
  – Note: if there are cyclic constraints this may not terminate

3.2 Schema Mapping

• Clio Algorithm: 2) Generate Candidate Mappings
  – For each pair of logical association A_s in the source and A_t in the target produced in step 1
  – Find the matches that are covered by A_s and A_t
    • Matches that lead from an element of A_s to an element from A_t
  – If there is at least one such match then create mapping by equating variables as indicated by the matches and create st-tgd with A_s in LHS and A_t in RHS

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
4. Virtual Data Integration

- Virtual Data Integration

Problems:
- How to create mappings?
  - Discussed in previous part of the course
- How to compute query Q
  - This is the main focus of this part

4. Query Answering with Views

- How to compute query Q over global schema based on source schemas only?
  - What language is used to express mappings?
  - What language do we allow for Q?
  - What language(s) can we use to query local sources?
  - What language can we use to compute Q from query results returned by local sources?
  - How to deal with incompleteness?
4. Query Answering with Views

- Problems
  - How to determine whether a query can be answered at all?
  - Given a rewriting of the query using views, how do we know it is correct?
  - What to do if views can only return some of the query results?

Motivating Example (Part 1)

\[
\forall x, y, z, a, b, c : Person(x, y) \land Address(x, a, b, c) \Rightarrow \exists h : Person(x, z, a, h, c)
\]

Query:

\[
Q(\text{Name}) :- \text{Person}(\text{Name}, A, \text{OP, OA, HP}).
\]

Rewritten query over the source:

\[
Q(\text{Name}) :- \text{Person}(\text{Name}, A, \text{OP, OA, HP}).
\]

Example: Solutions

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>3</td>
<td>Chicago</td>
<td>(312)1234343</td>
</tr>
<tr>
<td>Alice</td>
<td>2</td>
<td>Chicago</td>
<td>(312)5557777</td>
</tr>
<tr>
<td>Peter</td>
<td>1</td>
<td>New York</td>
<td>(465)1231234</td>
</tr>
</tbody>
</table>

Motivating Example (Part 2)

\[
Q(T,Y,D) : \neg \text{Movie}(T,Y,G), Y \geq 1950, G = "comedy"
\]

\[
\forall I, T, Y, D : \text{Director}(I,D), \text{Actor}(I,D)
\]

\[
V_2(1) = \neg \text{Movie}(T,Y,G), Y \geq 1950, G = "comedy"
\]

\[
V_1(I,D) : \neg \text{Director}(I,D), \text{Actor}(I,D)
\]

**Containment** does not hold, but intuitively, \(V_2\) and \(V_1\) are useful for answering \(Q\).

\[
Q''(T,Y,D) : \neg V_2(I,T,Y), V_1(I,D)
\]

How do we express that intuition?

**Answering queries using views!**

Problem Definition

**Input:** Query \(Q\)

**View definitions:** \(V_1, \ldots, V_n\)

**A rewriting:** a query \(Q'\) that refers only to the views and interpreted predicates (comparisons)

**An equivalent rewriting of \(Q\) using \(V_1, \ldots, V_n\):**

a rewriting \(Q'\), such that \(Q' \Leftrightarrow Q\)
Naïve approach

- **Given Q and views**
  - Randomly combine views into a query Q'
  - Check equivalence of Q' and Q
  - If Q' is equivalent we are done
  - Else repeat

- **Why is this not good?**
  - There are infinitely many ways of combining views
    - E.g., V, V x V, V x V x V, ...
  - We are not using any information in the query

Motivating Example (Part 3)

- **Movie**(ID, title, year, genre)
- **Director**(ID, director)
- **Actor**(ID, actor)

**Q**

\[Q(T,Y,D) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]
\[\text{Director}(I,D) \]
\[V_1(I,T,Y) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]
\[V_2(I,D) : \neg \text{Director}(I,D), \text{Actor}(I,D) \]
\[Q'''(T,Y,D) : V_1(I,T,Y), V_2(I,D) \]

maximally-contained rewriting

Maximally-Contained Rewritings

**Input:** Query Q
**Rewriting query language L**
**View definitions:** \(V_1, \ldots, V_n\)

Q' is a maximally-contained rewriting of Q given \(V_1, \ldots, V_n\) and L if:

1. \(Q' \in L\)
2. \(Q' \subseteq Q\), and
3. there is no \(Q''\) in L such that \(Q'' \subseteq Q\) and \(Q' \subset Q''\)

Why again?

Exercise: which of these views can be used to answer Q?

\[Q(T,Y,D) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]
\[\text{Director}(I,D), \text{Actor}(I,D) \]
\[V_1(I,T,Y) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]
\[V_2(I,D) : \neg \text{Director}(I,D), \text{Actor}(I,D) \]
\[V_3(I,T,Y) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]
\[V_4(I,T,Y) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]
\[V_5(I,T) : \neg \text{Movie}(I,T,Y,G), Y \geq 1940, G = \text{"comedy"} \]

Other use-cases

- Query optimization with materialized views
  - Need equivalent rewritings
  - Implemented in many commercial DBMS
  - Here interest is cost: how to speed-up query processing by using materialized views
Algorithms for answering queries using views

• **Step 1:** we’ll bound the space of possible query rewritings we need to consider (no comparisons)

• **Step 2:** we’ll find efficient methods for searching the space of rewritings
  – Bucket Algorithm, MiniCon Algorithm

• **Step 2b:** we consider “logical approaches” to the problem:
  – The Inverse-Rules Algorithm

Bounding the Rewriting Length

**Theorem:** if there is an equivalent rewriting, there is one with at most \( n \) subgoals.

Query: \( Q(\overline{X}) : -p_1(\overline{X}_1),...,p_n(\overline{X}_n) \)

Rewriting: \( Q'(\overline{X}) : -V_1(\overline{X}_1),...,V_m(\overline{X}_m) \)

Expansion: \( Q''(\overline{X}) : -g_1^{1},...,g_1^{j},...,g_m^{1},...,g_m^{j} \)

Proof: Only \( n \) subgoals in \( Q \) can contribute to the image of the containment mapping \( \varphi \)

Complexity Result

[LMSS, 1995]

• Applies to queries with no interpreted predicates.

• Finding an equivalent rewriting of a query using views is NP-complete
  – Need only consider rewritings of query length or less.

• Maximally-contained rewriting:
  – Union of all conjunctive rewritings of length \( n \) or less.

The Bucket Algorithm

**Key idea:**

– Create a bucket for each subgoal \( g \) in the query.
– The bucket contains views that contribute to \( g \).
– Create rewritings from the Cartesian product of the buckets (select one view for each goal)

• **Step 1:** assign views with renamed vars to buckets

• **Step 2:** create rewritings, refine them, until equivalent/all contained rewriting(s) are found

The Bucket Algorithm

**Step 1 Intuition**

– A view can only be used to provide information about a goal \( R(X) \) if it has a goal \( R(Y) \)
  – \( Q(X) : -R(X,Y) \)
  – \( V(X) : -S(X,Y) \)

– If the query goal contains variables that are in the head of the query, then the view is only useful if it gives access to these values (they are in the head)
  – \( Q(X) : -R(X,Y) \)
  – \( V(X) : -S(X,Y), R(Y,Z) \)
Bucket Algorithm in Action

$$Q(ID, Dir) : \rightarrow Movie(ID, title, year, genre), Revenues(ID, amount), Director(ID, dir), amount \geq 100M$$

$$V_1(I, Y) : \rightarrow Movie(I, Y, G), Revenues(I, A), I \geq 5000, A \geq 2000M$$

$$V_2(I, A) : \rightarrow Movie(I, Y, G), Revenues(I, A)$$

$$V_3(I, A) : \rightarrow Revenues(I, A), A \leq 50M$$

$$V_4(I, D, Y) : \rightarrow Movie(I, Y, G), Director(I, D), I \leq 3000$$

View atoms that can contribute to $Movie$:

$$V_1(ID, year'), V_2(ID, A'), V_3(ID, D', year')$$

Next Candidate Rewriting

<table>
<thead>
<tr>
<th>Movie(ID, title, year, genre)</th>
<th>Revenues(ID, amount)</th>
<th>Director(ID, dir)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1(ID, year)$</td>
<td>$V_2(ID, Y')$</td>
<td>$V_3(ID, Dir, Y')$</td>
</tr>
<tr>
<td>$V_2(ID, A')$</td>
<td>$V_4(ID, amount)$</td>
<td></td>
</tr>
<tr>
<td>$V_3(ID, D', year)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$q_1'(ID, dir) : \rightarrow V_1(ID, A'), V_2(ID, amount), V_3(ID, dir, y')$$

$$q_2'(ID, dir) : \rightarrow V_4(ID, amount), V_4(ID, dir, y'), amount \geq 100M$$

The Bucket Algorithm

**Step 2:**
- For each combination of one element of each bucket:
- Create query $Q'$ with query $Q$’s head and list all these view atoms in the body
- If $Q'$ equivalent to $Q$ (or contained in $Q$)
  - Done (equivalent)
  - Add to union of CQs for contained case
- If not try to add comparisons

The MiniCon Algorithm

$$Q(title, year, dir) : \rightarrow Movie(ID, title, year, genre), Director(ID, dir), Actor(ID, dir)$$

$$V_3(D, A) : \rightarrow Director(I, D), Actor(I, A)$$

**Intuition:** The variable $I$ is not in the head of $V_3$, hence $V_3$ cannot be used in a rewriting. **MiniCon** discards this option early on, while the Bucket algorithm does not notice the interaction.
MinCon Algorithm Steps

1) Create MiniCon descriptions (MCDs):
   - Homomorphism on view heads
   - Each MCD covers a set of subgoals in the query
     with a set of subgoals in a view

2) Combination step:
   - Any set of MCDs that covers the query subgoals
     (without overlap) is a rewriting
   - No need for an additional containment check!

MiniCon Descriptions (MCDs)
An atomic fragment of the ultimate containment mapping

Q(title,act,dir) :- Movie(ID,title,yr,genre),
Director(ID,dir),Actor(ID,act)
V(I,D,A) :- Director(I,D),Actor(I,A)

MCD:
   ID -> I
   dir -> D
   act -> A
   covered subgoals of Q: {2,3}

MCDs: Detail 1

Q(title,yr,dir) :- Movie(ID,title,yr,genre),
Director(ID,dir),Actor(ID,dir)
V(I,D,A) :- Director(I,D),Actor(I,A)

Need to specialize the view first:
V'(I,D,D) :- Director(I,D),Actor(I,D)

MCD:
   ID -> I
   dir -> D
   covered subgoals of Q: {2,3}

MCDs: Detail 2

Q(title,yr,dir) :- Movie(ID,title,yr,genre),
Director(ID,dir),Actor(ID,dir)
V(I,D,D) :- Director(I,D),Actor(I,D),
   Movie(I,T,Y,G)

Note: the third subgoal of the view is not included
in the MCD.

MCD:
   ID -> I
   dir -> D
   covered subgoals of Q still: {2,3}

Inverse-Rules Algorithm

• A "logical" approach to AQUV
• Produces maximally-contained rewriting in
  polynomial time
  - To check whether the rewriting is equivalent to
    the query, you still need a containment check.
• Conceptually simple and elegant
  - Depending on your comfort with Skolem
    functions...

Inverse Rules by Example

Given the following view:
V_7(I,T,Y,G) :- Movie(I,T,Y,G),Director(I,D),Actr(I,D)

And the following tuple in V_7:
V_7(79,Manhattan,1979,Comedy)

Then we can infer the tuple:
Movie(79,Manhattan,1979,Comedy)

Hence, the following `rule` is sound:
IN_7 : Movie(I,T,Y,G) :- V_7(I,T,Y,G)
Skolem Functions

\[ V(I,T,Y,G) : - Movie(I,T,Y,G), Director(I,D), Actor(I,D) \]

Now suppose we have the tuple \( V_7(79,\text{Manhattan},1979,\text{Comedy}) \)

Then we can infer that there exists some director. Hence, the following rules hold (note that they both use the same Skolem function):

\[ \text{IN}_2: \text{Director}(I, f(I,T,Y,G)) : - V_7(I,T,Y,G) \]
\[ \text{IN}_3: \text{Actor}(I, f(I,T,Y,G)) : - V_7(I,T,Y,G) \]

Inverse Rules in General

Rewriting = Inverse Rules + Query

\[ Q_2(\text{title}, \text{year}, \text{genre}) : - Movie(\text{ID}, \text{title}, \text{year}, \text{genre}) \]

Given \( Q_2 \), the rewriting would include:

\[ \text{IN}_1, \text{IN}_2, \text{IN}_3, Q_2. \]

Given input: \( V_7(79,\text{Manhattan},1979,\text{Comedy}) \)

Inverse rules produce:

\( \text{Movie}(79,\text{Manhattan},1979,\text{Comedy}) \)
\( \text{Director}(79, f(79,\text{Manhattan},1979,\text{Comedy})) \)
\( \text{Actor}(79, f(79,\text{Manhattan},1979,\text{Comedy})) \)

(in the last tuple is produced by applying \( Q_2 \)).

Comparing Algorithms

- Bucket algorithm:
  - Good if there are many interpreted predicates
  - Requires containment check. Cartesian product can be big
- MiniCon:
  - Good at detecting interactions between subgoals

Algorithm Comparison (Continued)

- Inverse-rules algorithm:
  - Conceptually clean
  - Can be used in other contexts (see later)
  - But may produce inefficient rewritings because it "undoes" the joins in the views (see next slide)
- Experiments show MiniCon is most efficient.
- Even faster:
  Konstantinidis, G. and Ambite, J.L., Scalable query rewriting: a graph-based approach. SIGMOD '11

Inverse Rules Inefficiency Example

Query and view:
\[ Q(X,Y) : - e_1(X,Z), e_2(Z,Y) \]
\[ V(A,B) : - e_1(A,C), e_2(C,B) \]

Inverse rules:
\[ e_1(A, f_1(A,B)) : - V(A,B) \]
\[ e_2(f_1(A,B), B) : - V(A,B) \]

Now we need to re-compute the join...

View-Based Query Answering

- Maximally-contained rewritings are parameterized by query language.
- More general question:
  - Given a set of view definitions, view instances and a query, what are all the answers we can find?
- We introduce certain answers as a mechanism for providing a formal answer.
Consider the two views:

\[ V_8(dir): \neg \text{Movie}(ID, dir, actor) \]
\[ V_9(actor): \neg \text{Movie}(ID, dir, actor) \]

And suppose the extensions of the views are:

\[ V_8: \{\text{Allen, Copolla}\} \]
\[ V_9: \{\text{Keaton, Pacino}\} \]

Certain Answers: Formal Definition

[Open-world Assumption]

- Given:
  - Views: \( V_1, \ldots, V_n \)
  - View extensions \( v_1, \ldots, v_n \)
  - A query \( Q \)
- A tuple \( t \) is a certain answer to \( Q \) under the open-world assumption if \( t \in Q(D) \) for all databases \( D \) such that:
  - \( V_i(D) \subseteq v_i \) for all \( i \).

Certain Answers: Example

\[ V_8(dir): \neg \text{Director}(ID, dir) \quad V_8: \{\text{Allen}\} \]
\[ V_9(actor): \neg \text{Actor}(ID, actor) \quad V_9: \{\text{Keaton}\} \]
\[ Q(dir, actor): \neg \text{Director}(ID, dir), \text{Actor}(ID, actor) \]

Under closed-world assumption:
- single DB possible \( \Rightarrow \) (Allen, Keaton)

Under open-world assumption:
- no certain answers.

Certain Answers

[Closed-world Assumption]

- Given:
  - Views: \( V_1, \ldots, V_n \)
  - View extensions \( v_1, \ldots, v_n \)
  - A query \( Q \)
- A tuple \( t \) is a certain answer to \( Q \) under the open-world assumption if \( t \in Q(D) \) for all databases \( D \) such that:
  - \( V_i(D) = v_i \) for all \( i \).

The Good News

- The MiniCon and Inverse-rules algorithms produce all certain answers
  - Assuming no interpreted predicates in the query (ok to have them in the views)
  - Under open-world assumption
  - Corollary: they produce a maximally-contained rewriting

Possible Databases

There are multiple databases that satisfy the above view definitions: (we ignore the first argument of Movie below)

\[ DB_1. \{(\text{Allen, Keaton}), (\text{Coppola, Pacino})\} \]
\[ DB_2. \{(\text{Allen, Pacino}), (\text{Coppola, Keaton})\} \]

If we ask whether Allen directed a movie in which Keaton acted, we can’t be sure.

Certain answers are those true in all databases that are consistent with the views and their extensions.
In Other News…

- Under closed-world assumption finding all certain answers is co-NP hard!

**Proof:** encode a graph - $G = (V,E)$

\[
\begin{align*}
v_1(X) & : \neg \text{color}(X,Y) \quad h(V_1) = V \\
v_2(Y) & : \neg \text{color}(X,Y) \quad h(V_2) = \{\text{red}, \text{green}, \text{blue}\} \\
v_3(X,Y) & : \neg \text{edge}(X,Y) \quad h(V_3) = E
\end{align*}
\]

$q() : \neg \text{edge}(X,Y), \text{color}(X,Z), \text{color}(Y,Z)$

$q$ has a certain tuple iff $G$ is not 3-colorable

Interpreted Predicates

- In the views: no problem (all results hold)
- In the query $Q$:
  - If the query contains interpreted predicates, finding all certain answers is co-NP-hard even under open-world assumption
  - Proof: reduction to CNF.

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
5. Data Exchange

- **Virtual Data Integration**
  - Never materialize instances for the global schema
  - Data of global schema only “visible” through queries

- **Data Exchange**
  - Materialize instance of global instance
  - Based on information from an instance of the local schema
  - We call this the “source schema”

**Data Exchange Problem Statement**

**Input:**
- Given a source and a target schema
- + instance of the source schema
- + set of schema mappings (here st-tgds)

**Output:**
- Instance of the target schema that fulfills constraints

Example: Types of Matching

\[ \forall x, y, z, a : \text{Person}(x, y) \land \text{Address}(y, z, a) \rightarrow \exists b, c : \text{Person}(x, z, a, b, c) \]
5.1 Data Exchange Setting

Data Exchange setting is a tuple \((S, T, I, \Sigma)\)
- Source Schema \(S'\)
- Target Schema \(T'\)
- Instance \(I\) of \(S'\)
- Mappings \(\Sigma\) from \(S\) to \(T'\)

Definition: Data Exchange Setting

Given data exchange setting is a tuple \((S, T, I, \Sigma)\)
- Find instance \(J\) of \(T\) so that \((I,J)\) fulfills mappings \(\Sigma\)

5.1 Data Exchange Solutions

Example: Solutions

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>Office-contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>Chicago</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Alice</td>
<td>Chicago</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Bob</td>
<td>New York</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

\(\forall x, y, z, a : \text{Person}(x, y) \land \text{Address}(y, z, a) \rightarrow \exists b, c : \text{Person}(x, z, a, b, c)\)

Can we come up with a solution?

Example: Solutions

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>Office-contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>Chicago</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Alice</td>
<td>Chicago</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Bob</td>
<td>New York</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

\(\forall x, y, z, a : \text{Person}(x, y) \land \text{Address}(y, z, a) \rightarrow \exists b, c : \text{Person}(x, z, a, b, c)\)

5.1 Number of Solutions

- How many solutions exists?
  - Depends on how whether we use existentially quantified variables in the mappings?
    - i.e., do we have attributes for which we have to invent values?
  - What attribute values do we allow?
    - Surely values from the source instance (active domain)
    - NULL?
      - Need multiple NULL values as placeholders for missing values that have to be the same
    - Note that this is the open-world assumption
    - there are infinitely many solutions (if domains infinite)

- Target instance domain
  - Consider a universe \(U\)
    - Source instance can only use values from \(U\)
  - Consider an infinite set \(N\) of labeled nulls
    - Target instance can use these as placeholders for missing values
5.1 Data Exchange Solutions

Example: Multiple Solutions

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Office-phone</th>
<th>Office-address</th>
<th>Home-phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>Chicago</td>
<td>(312) 123-456</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>Alice</td>
<td>Chicago</td>
<td>(312) 555-777</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Bob</td>
<td>New York</td>
<td>(465) 123-456</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Example: Solution generality

Consider query:
\[ \text{Q(n)} : \text{P(n, a, op, oa, hp), } \text{oa} = \text{Hometown} \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Office-phone</th>
<th>Office-address</th>
<th>Home-phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>Chicago</td>
<td>(312) 123-456</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>Alice</td>
<td>Chicago</td>
<td>(312) 555-777</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Bob</td>
<td>New York</td>
<td>(465) 123-456</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

5.1 Certain answers (... again)

- Have multiple solutions
  - Define certain answers for queries as before
  - Every tuple t so that t is in the result of query Q over any valid solution J

- What's new?
  - Want to materialize an instance so that computing certain answers over this instance is easy
    - Not immediately clear that this actually possible

5.1 Universal solutions

- Universal solution
  - Want a solution that is as general as possible
  - We call such most general solutions universal solutions
  - How do we know whether it is most general
    - We can map the tuples in this solution to any other less general solution by replacing unspecified values (labelled nulls) with actual data values

- Query answering with universal solutions
  - For UCQs: run query over universal instance
  - Remove tuples with labelled nulls
  - Result are the certain answers!

5.1 Universal Solutions

Definition: Homomorphism

A homomorphism \( h \) from instance J to instance J' maps the constants and nulls of J to the constants and nulls of J' and fulfills the following conditions:

- Constants are mapped onto themselves: \( h(c) = c \)
- Every tuple \( R(a_1, \ldots, a_n) \) in J is mapped to a tuple in J': \( h(R(a_1, \ldots, a_n)) = R(h(a_1), \ldots, h(a_n)) \)

Definition: Universal solution

Given data exchange setting \((S, T, I, \Sigma)\), an instance J of T is called an universal solution for a source instance I if it is a solution and for every other solution J' hold that:

- There exists a homomorphism from J to J'

Example: Solution generality

Consider query:
\[ \text{Q(n)} : \text{P(n, a, op, oa, hp), } \text{oa} = \text{Hometown} \]
5.1 Data Exchange Solutions

Example: Solution generality

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Office-phone</th>
<th>Home-phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>Chicago</td>
<td>(312) 123-4567</td>
<td>111-222-3333</td>
</tr>
<tr>
<td>Alice</td>
<td>Chicago</td>
<td>(312) 123-4567</td>
<td>111-222-3333</td>
</tr>
<tr>
<td>Bob</td>
<td>New York</td>
<td>(465) 123-4567</td>
<td>123-456-7890</td>
</tr>
</tbody>
</table>

Above is universal solution.

How to map to below non-universal solution?

Replace generic labelled Nulls with values:

R = Hometown, V = 111-222-3333, C = other town.

Example: Solution generality

Name          Address    Office-phone  Home-phone
Peter          Chicago    (312) 123-4567  111-222-3333
Alice          Chicago    (312) 123-4567  111-222-3333
Bob            New York   (465) 123-4567  123-456-7890

5.2 Computing Solutions

• Can we use a database system to compute solutions?
  – Yes, systems such as Clio generate queries that compute universal solutions!
    • SQL
    • Java
    • XSLT (for XML docs)

• Generating Executable Transformations
  – How to preserve semantics of labeled nulls
    • n = n’ is true if we have the same labeled null only
    • n = n’ if one is a constant and the other one is a labeled null

5.2 Skolem Functions

• Skolem functions for labeled nulls
  – For each existential variable in a tgd we create a new skolem function
  – What should be the arguments of the function?
    • Naive: all universally quantified variables
    • Better: only relevant ones

Example: Skolem functions

Person
  Name
  Address
  Age

Person
  Name
  Address
  Office-phone
  Office-address
  Home-phone

Example: Skolem functions

Person
  Name
  Address
  Age

Person
  Name
  Address
  Office-phone
  Office-address
  Home-phone
5.2 Skolem Functions

Example: Skolem Functions

\[ \forall a, b, c, d, e : \text{Person}(a, b, c, d, e) \rightarrow \exists f, g \text{Person}(a, f, g) \land \text{Address}(f, b, c) \]

Introduce skolem functions sk1 and sk2 for f and g.

What arguments to choose for sk1 and sk2?

E.g., f should be fixed for a certain address and should not depend on the person.

5.2 Skolem Functions

• Clio Schema Graph Algorithm

• Nodes
  – Create a graph with one node for every target attribute and one node for every target relation
  – Also add nodes for source attribute if they are copied to the target according to the mapping

• Edges
  – Edges between a relation and its attributes
  – Edges between target attributes that use the same variable
  – Edges between source attributes and target attributes if they use the same variable
5.2 Skolem Functions

Example: Skolem Functions

∀a, b, c, d, e : Person(a, b, c, d, e) → ∃f, g Person(a, f, g) ∧ Address(f, h, c)

Example: Skolem Functions

∀a, b, c, d, e : Person(a, b, c, d, e) → ∃f, g Person(a, f, g) ∧ Address(f, h, c)

5.1 Data Exchange Solutions

Example: Skolem Functions

∀a, b, c, d, e : Person(a, b, c, d, e) → ∃f, g Person(a, f, g) ∧ Address(f, h, c)

5.2 Executable Transformations

Example: Skolem Functions

∀a, b, c, d, e : Person(a, b, c, d, e) → ∃f, g Person(a, f, g) ∧ Address(f, h, c)

5.3 Recap Data Exchange Steps

• Schema Matching
  • Generate Schema Mappings
    – Use constraints
  • Generate Executable Transformations
    – SQL, XSLT, XQuery
    – Skolems for missing values
  • Run Transformations over source instance to generate target instance
    – Universal solution
5.3 Comparison with virtual integration

- Pay cost upfront instead of at query time
- Making decisions early vs. at query time
  - When generating a solution
  - Caution: bad decisions stick!
- **Universal solutions** allow efficient computation of certain types of queries using, e.g., SQL

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