CS520
Data Integration, Warehousing, and Provenance

Course Info

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

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

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)

What is information integration?
- Combination of data and content from multiple sources into a common format
  - Completeness
  - Correctness
  - Efficient

I am a database guy!

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 [3]

- 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>  
  <Value>106.8</Value>  
  <Currency>Euro</Currency>  
  …

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>  
  …  
  <Stock>  
  <Month>12</Month>  
  <Value>107</Value>  
  </Stock>
**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

**Why hard? Cont.**

- 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
- 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
- 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
  – Google Group:
    https://groups.google.com/d/forum/cs520-2015-spring-group
    • Used for announcements
    • Use it to discuss with me, TA, and fellow students

• Faculty
  – Boris 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

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)

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
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
1. Introduction

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

Outline

0) Course Info
1) Introduction
2) Data Preparation and Cleaning
3) Schema matching and mapping
4) Virtual Data Integration
5) Data Exchange
6) Data Warehousing
7) Big Data Analytics
8) Data Provenance
1.1 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?)
  - Fixed choices
    - Keyword search
    - "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%’;

- Query cannot be answered
  SELECT title
  FROM books
  WHERE genre = ‘SciFi’;

- How do we know what authors exist?

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

• Data model
  – Relational model
  – XML model
  – Object-oriented model
  – Ontological model
  – JSON
  – ...

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

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

• Person and their addresses
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

Example

Person
Name
Address
City Zip

Person
Name
City Zip

1.1 Structural Heterogeneity

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

Example

Person
Name
{
 Address
  Id
  City
  Zip
 }

Person
Name
City Zip

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 Heterogeneity + Autonomy

• Taxonomy of Heterogeneity

System Structural Semantic

Heterogeneity

Software Interface Datamodel Schema Naming Identity Value conflicts
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 Mamal

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

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

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

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

- Naming concepts (units)

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

- 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, 301000000, 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 represent the same entity first!
  - Resolving such conflicts requires **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 prepreparation**
- **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**
- **Datawarehousing**
- **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
    - 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
    - \[ R(x, y, z) \]
    - \[ x = y \]

1.4 Integrity constraints

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

1.4 Datalog

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

- A Datalog rule
  \[ Q(\vec{x}) : -R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n). \]

- 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
  \[ Q(\vec{x}) : -R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n). \]

- \( X \)'s are lists of variables and constants
- Ri'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
  \[ Q(\vec{x}) : -R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n). \]

```
Example
Q(N):- Person(N,A).
N, A are variables
Q(N) is an atom
Person and Q are predicates

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>peter</td>
<td>34</td>
</tr>
<tr>
<td>bob</td>
<td>45</td>
</tr>
</tbody>
</table>

Activate domain
\( \text{adom}(I) = \{ \text{peter, bob, 34, 45} \} \)
```

1.4 Datalog - Terminology

- **Intensional** vs. **extensional**
  - Extensional database (\( \text{edb} \))
    - What we usually call database
  - Intensional database (\( \text{idb} \))
    - Relations that occur in the head of rules (are populated by the query)
    - Usually we assume that these do not overlap
    \[ Q(\vec{x}) : -R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n). \]

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 \( \vec{x}_i \)
  \[ Q(\vec{x}) : -R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n). \]

```
Example
Q(Name):= Person(Name,Age) .  (safe)
Q(Name,Sal):- Peron(Name,Age) . (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 adom(I) such that x is bound to c and the body is true:

\[ Q(\bar{x}) : \neg R_1(\bar{x}_1), \ldots, \neg R_m(\bar{x}_m). \]

Example:

\[ Q(N) : Person(N, A). \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>peter</td>
<td>34</td>
</tr>
<tr>
<td>bob</td>
<td>34</td>
</tr>
</tbody>
</table>

Activate domain:

adom(I) = {peter, bob, 34}

Example:

Relation hops(A, B) storing edges of a graph.

\[ Q_{\text{hop}}(x, z) : \text{hop}(x, y), \text{hop}(x, z). \]

\[ Q_{\text{reach}}(x, y) : \text{hop}(x, y). \]

\[ Q_{\text{node}}(x) : \text{hop}(x, y). \]
1.4 Datalog

Example
Relation hops(A, B) storing edges of a graph.
Q_node(x): hop(x, y).
Q_node(x): hop(y, x).
Q_contReach(x, y): Q_node(x), Q_node(y),
not Q_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 ≡ Q’ ⇔ ∀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 ⊆ Q’ ⇔ ∀I : Q(I) ⊆ 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

1.4 Containment and Equiv.

Example
Relation hops(A, B) storing edges of a graph.
Q2hop(x, z): hop(x, y), hop(x, z).
Q_op2Hop(x, z): hop(x, y), hop(x, z).
Q2Hop(x, z): hop(x, z).
Q_sym(x, y): hop(x, y).
Q_sym(x, y): hop(y, x).
Q_sym2Hop(x, y): Q_sym(x, y), Q_sym(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>P^#-complete</td>
</tr>
<tr>
<td>Query Containment</td>
<td>Undecidable</td>
<td>NP-complete</td>
<td>NP-complete</td>
<td>P^#-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</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</td>
<td>Undecidable</td>
<td>Undecidable</td>
<td>Undecidable</td>
<td>Π⁰ᵖ⁻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:
  - head(Q) = variables in head of query Q
  - body(Q) = atoms in body of Q
  - 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 :- … as a convention for Q() :- …
  - 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 {()}
  - -> We interpret {} as false and {()} as true
  - Boolean query is essentially an existential check

1.4 Boolean Conjunctive Queries

- BCQ in SQL

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

  Note: in Oracle and DB2 we need a FROM clause

- BCQ in SQL

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

  Notes:
  - Oracle and DB2 FROM not optional
  - Oracle has no boolean datatype
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 $Q'$ iff there exists a containment mapping $\psi$ from $Q'$ to $Q$.

Example

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

Do containment mappings exist?

$Q_1 \rightarrow Q_2 : \psi(a) = x, \psi(b) = y$
$Q_2 \rightarrow Q_1 : \psi(x) = a, \psi(y) = b$
### 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**

<table>
<thead>
<tr>
<th>$Q_1()$</th>
<th>$R(a,b), R(c,b)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_2()$</td>
<td>$R(x,y)$</td>
</tr>
<tr>
<td>$Q_2 \rightarrow Q_1: \psi(x) = a, \psi(y) = b$</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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

### 1.4 Containment Background

- Containment Mapping <-> 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 Mappings

<table>
<thead>
<tr>
<th>$Q_1()$</th>
<th>$R(a,b), R(c,b)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_2()$</td>
<td>$R(x,y)$</td>
</tr>
<tr>
<td>$Q_2 \rightarrow Q_1: \psi(x) = a, \psi(y) = b$</td>
<td></td>
</tr>
</tbody>
</table>

$I^{Q_1} = \{(a,b), (c,b)\}$

$Q_2(I^{Q_1}) = \{(\)}$

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

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

### 1.4 Containment Background

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

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

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 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
- \( d(p,p) = 0 \)
- \( d(p,q) = 0 \)

- 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 \( p=q \)
- Euclidean distance: \( d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i-q_i)^2} \)
- Edit distance: \( d(p,q) = \) minimum number of single character insertions, deletions, replacements to transform \( p \) 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!

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

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
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/

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
    • “FirstName: New York, City: Peter”
  – Incorrect information
    • “City:New York, Zip: 60616”
  – Missing information
    • “City: New York, Zip: ”

2. Causes of “Dirty” Data (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 if a condition is met
    - Example: zip -> city only if country is US
- Constraints that consider master data
  - Master data is highly reliable data such as a government issued zip, city lookup table

2.1 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>

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

Example: Constraints Languages
<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>60615</td>
<td>Chicago</td>
<td>Hans</td>
<td>NULL</td>
<td>1,000,000</td>
</tr>
<tr>
<td>333-355-4343</td>
<td>60615</td>
<td>Chicago</td>
<td>Malcom</td>
<td>Hans</td>
<td>20,000</td>
</tr>
</tbody>
</table>

- The zip code uniquely determines the city
- Nobody should earn more than their direct superior
- Salaries are non-negative

\[ \forall \neg \left( E(x, y, z, u, v, w) \land w < w_0 \right) \]

2.1 Constraint Repair Problem

Definition: Constraint Repair Problem
Given set of constraints \( \Sigma \) and an database instance \( I \) which violates the constraints find a clean instance \( I' \) so that \( I' \) fulfills \( \Sigma \)

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

2.1 Constraint based Cleaning Overview

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

2.1 Constraint Repair Problem

- 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

Example: Constraints Languages
<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>60615</td>
<td>Chicago</td>
<td>Hans</td>
</tr>
<tr>
<td>333-355-4343</td>
<td>60615</td>
<td>Chicago</td>
<td>Malcom</td>
</tr>
</tbody>
</table>

- The zip code uniquely determines the city

\[ E(x, y, z, u, v, w) \land w < w_0 \]

2.1 Constraint based Cleaning Overview

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

2.1 Constraint Repair Problem

- 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

Example: Constraint Violations

<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)

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>
</tbody>
</table>

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

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

• Principle of minimality
  - Choose repair that minimally modifies database
  - Motivation: consider the solution that deletes every tuple

Most update approaches **equate RHS** because there is usually no good way to choose LHS values unless we have master data
  - E.g., update zip to 56423 or 52456 or 22322 ...

• Given FD A -> B on R
  - Recall logical representation
  - For all 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
    - Q0: 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) |
| POS: zip \(\rightarrow\) city |
| Violation Detection Query |
| SELECT EXISTS (SELECT * FROM Person x, Person y WHERE x.zip = y.zip AND x.city \(\neq\) y.city) |
| To know which tuples caused the conflict: |
| SELECT * FROM Person x, Person y WHERE x.zip = y.zip AND x.city \(\neq\) 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 (revisited)

Given a 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

2.1 Cost Metrics

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

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!

• 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

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

• FD Repair Algorithm: 3. Attempt
  – More heuristics to improve quality and performance
    • Cost-based pick of next EQ’s to merge
    • Also for FKs (Inclusion Constraints)

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

2.1 Consistent Query Answering

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

Example: Constraint Repair

<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 a 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. Overview

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

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>3333333333</td>
<td>IL</td>
<td>60616</td>
<td>Petre</td>
</tr>
</tbody>
</table>

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

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>
</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)
    - Why?

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:

\( \text{NEED} \rightarrow \text{STREET} \)

- Trivial solution: delete all chars in \( \text{NEED} \), then insert all chars in \( \text{STREET} \)
- g: gives upper bound on distance \( \text{len(NEED)} + \text{len(STREET)} = 10 \)

Example:

\( \text{NEED} \rightarrow \text{STREET} \)

Minimal solution:
- insert \( S \)
- insert \( T \)
- replace \( N \) with \( R \)
- replace \( D \) 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 length $i$ of $s$ and prefix of length $j$ of $s'$
  - $D(\text{len}(s), \text{len}(s'))$ is the solution
  - Represented as a matrix
  - Populated based on rules shown on the next slide

**Recursive definition**

- $D(i,0) = i$
  - Cheapest way of transforming prefix $s[i]$ into an 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:

NEED $\rightarrow$ STREET

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>T</th>
<th>R</th>
<th>E</th>
<th>E</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.3 Entity Resolution

Example:

NEED -> STREET

<table>
<thead>
<tr>
<th>S</th>
<th>T</th>
<th>R</th>
<th>E</th>
<th>E</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>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.
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

- **Jaccard-Measure**
  - $J_B(s, s') = \frac{|B_s \cap B_{s'}|}{|B_s \cup B_{s'}|}$
  - Jaccard measures relative overlap of tokens in two strings
  - Number of common tokens divided by total number of tokens

**Example:**

Input string:
- $S = \text{"the tokenization of strings is commonly used in information retrieval"}$
- $S' = \text{"we consider nanotubes in our experiments..."}$
- $S'' = \text{"we prove that P=NP, thus solving..."}$

- $\text{Tok}(S) = \{\text{commonly, in, information, is, of, retrieval, strings, the, tokenization, used}\}$
- $\text{Tok}(S') = \{\text{consider, experiments, in, nanotubes, our, we}\}$
- $\text{Tok}(S'') = \{\text{P=NP, prove, solving, that, thus, we}\}$

- $d_{jacc}(S, S') = \frac{3}{10} = 0.3$
- $d_{jacc}(S, S'') = 0 / 13 = 0$
- $d_{jacc}(S', S'') = 1 / 11 = 0.0909$

- **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
  - Inverse of frequency of term in the set of all documents
  - Compute cosine similarity between two feature vectors
  - Measure how similar they are in term distribution (weighted by how uncommon terms are)
  - Size of the documents does not matter
  - See textbook for details

2.3 Entity Resolution

• **Entity resolution**
  - Concatenate attribute values of tuples and use string similarity measure
  - Loose information encoded by tuple structure
  - E.g., `[Gender: male, Salary: 9000]` or `~ "Gender: male, Salary: 9000"
  - Combine distance measures for single attributes
  - Weighted sum or more complex combinations
  - E.g., earth-movers distance

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 $\beta$

2.3 Entity Resolution

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

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

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

- **Learning-based approaches**
  - 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

2. Overview

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

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. Schema Matching and Mapping

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
    - Data warehousing
    - Data exchange
  - Want to be able to translate queries against one schema into queries against another schema
    - Virtual data integration

- **Why both matching and mapping**
  - 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
3.1 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

- **Typical Matching System Architecture**
  - Matcher
  - Combiner
  - Constraint Enforcer
  - Match Selector
  - Determine actual matches
  - Use constraints to modify similarity matrix
  - Combine individual similarity matrices
  - Each matcher uses one type of information to compute similarity matrix

- **Example: Types of Matching**

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Office-contact</th>
<th>Home-phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>1</td>
<td>Chicago</td>
<td>(312) 123 456</td>
</tr>
<tr>
<td>Alice</td>
<td>2</td>
<td>Chicago</td>
<td>(312) 567 890</td>
</tr>
<tr>
<td>Bob</td>
<td>3</td>
<td>New York</td>
<td>(999) 123 456</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>Office-contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>1</td>
<td>Chicago</td>
<td>(312) 123 456</td>
</tr>
<tr>
<td>Alice</td>
<td>2</td>
<td>Chicago</td>
<td>(312) 567 890</td>
</tr>
<tr>
<td>Bob</td>
<td>3</td>
<td>New York</td>
<td>(999) 123 456</td>
</tr>
</tbody>
</table>
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

- **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

- **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

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

**Typical strategies**

- **Average, Minimum, Max**
- **Weighted combinations**
- **Some script**

**Encode domain knowledge into constraints**

- **Hard constraints:** Only consider match combinations that fulfill constraints
- **Soft constraints:** Violating constraints results in penalty of scores
  - Assign cost for each constraint
  - Return combination that has the maximal score
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., \text{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>Name</th>
<th>Address</th>
<th>Id</th>
<th>City</th>
<th>Office-contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>1</td>
<td>Chicago</td>
<td>(312) 123-4567</td>
<td></td>
</tr>
<tr>
<td>Alice</td>
<td>3</td>
<td>Chicago</td>
<td>(312) 555-7777</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>3</td>
<td>New York</td>
<td>(465) 123-4567</td>
<td></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?

3.2.6.3 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

**Example:** Types of Matching

<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>1</td>
<td>Chicago</td>
<td>(312) 123-4567</td>
</tr>
<tr>
<td>Alice</td>
<td>3</td>
<td>Chicago</td>
<td>(312) 555-7777</td>
</tr>
<tr>
<td>Bob</td>
<td>3</td>
<td>New York</td>
<td>(465) 123-4567</td>
</tr>
</tbody>
</table>

How do we know that we should join tables Person and Address to get the matching address for a name?

**What values should we use for Office-address and Home-phone**

**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

3.2 Schema Mapping

- **Certain answers**
  - Given mapping $M$ and $Q$
  - Instances $I_1$ to $I_n$
  - 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 $M$ then $t \in Q(I_G)$

3.2 Schema Mapping

- **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)

3.2 Schema Mapping

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

3.2 Schema Mapping

- **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

- Local Schema
  - Person
  - Name
  - Address
  - ID
- Global Schema
  - Person
  - Name
  - Address
  - Phone

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)

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)

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

3.2 Schema Mapping

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

  Mapping R(X, Y) = S(X, Y), T(Y, Z)

  Q(X) :- R(X, Y)

  Rewrite into

  Q(X) :- S(X, Y), T(Y, Z)

3.2 Schema Mapping

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

**Example: Types of Matching**

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

- **Input:** $S(X,Z) = R(X,Y), T(Y,Z)$
- **Output:** $Q(X) :- R(X,Y)$
  - 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: $Q'(G) \n Q(S)$

---

**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

---

3.2 Schema Mapping

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

---

3.2 Schema Mapping

- **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

---

3.2 Schema Mapping

- **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

• **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* (tgd’s)
      • There are also chase procedures that consider *edges* (e.g., PKs)
    – Starting point are all single relational atoms
      • E.g., R(X,Y)

• **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 occurring variables from the current tableau
  • **Chase**
    – Applying the chase until no more changes
    – Note: if there are cyclic constraints this may not terminate

• **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
CS520
Data Integration, Warehousing, and Provenance

4. Virtual Data Integration

• Virtual Data Integration

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

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 due 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)

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

\[ Q(Y,D) \colon \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]
Director(ID,D),Actor(ID,D)

\[ V_1(I,T,Y) \colon \neg \text{Movie}(I,T,Y,G), Y \geq 1940, G = \text{"comedy"} \]
Director(ID,D),Actor(ID,D)

\[ V_1 \supseteq Q \implies Q(T,Y,D) \colon \neg V_1(T,Y,D), Y \geq 1950 \]

**Containment** is enough to show that \( V_1 \) can be used to answer \( Q \).

Motivating Example (Part 2)

\[ Q(T,Y,D) \colon \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]
Director(ID,D),Actor(ID,D)

\[ V_2(I,T,Y) \colon \neg \text{Movie}(I,T,Y,G), Y \geq 1940, G = \text{"comedy"} \]
Director(ID,D),Actor(ID,D)

\[ V_1 \supseteq Q \]

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

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

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 \( Q' \iff Q \)
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(T,Y,D):
Movie(I,T,Y,G), Y ≥ 1950, G = "comedy"
Director(I,D), Actor(I,D)

V_3(I,D):
Director(I,D), Actor(I,D)

V_4(I,T,Y):
Movie(I,T,Y,G), Y ≥ 1960, G = "comedy"

Q''(T,Y,D):
V_4(I,T,Y), V_3(I,D)

maximally-contained rewriting

Maximally-Contained Rewritings

Input: Query Q
Rewriting query language L
View definitions: V_1, ..., V_n

Q' is a maximally-contained rewriting of Q given V_1, ..., V_n and L if:
1. Q' ∈ L,
2. Q' ⊆ Q, and
3. there is no Q'' in L such that Q'' ⊆ Q and Q'' ⊂ Q'

Why again?

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

Q(T,Y,D):
Movie(I,T,Y,G), Y ≥ 1950, G = "comedy"
Director(I,D), Actor(I,D)

V_3(I,T,Y):
Movie(I,T,Y,G), Y ≥ 1950, G = "comedy"

V_3(I,D):
Director(I,D), Actor(I,D)

V_4(T,Y):
Movie(I,T,Y,G), Y ≥ 1950, G = "comedy"

V_4(I,T,Y):
Movie(I,T,Y,G), Y ≥ 1950, G = "comedy"
Award(I,W)

V_8(I,T):
Movie(I,T,Y,G), Y ≥ 1940, G = "comedy"
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**: $Q(X) : -p_1(X_1),...,-p_n(X_n)$

**Rewriting**: $Q'(X) : -V_1(X_1),...,-V_m(X_m)$

**Expansion**: $Q''(X) : -g_1^{1},...,-g_k^{1},...,-g_1^{m},...,-g_j^{m}$

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)
  - $Q(X) : - R(X,Y)$
  - $V(X) : - S(X,Y), R(Y,Z)$

The Bucket Algorithm

**Step 1**:
- We want to construct buckets with views that have partially mapped variables
  - For each goal $g = R$ in query
  - For each view $V$
    - For each goal $v = R$ in $V$
      - If the goal has head variables in the same places as $g$ then
        - Rename the view head variables to match the query goal vars
        - Choose a new unique name for each other var
        - Add the resulting view atom to the bucket
Bucket Algorithm in Action

\[ Q'(ID, Dir) \Rightarrow \text{Movie}(ID, \text{title}, \text{year}, \text{genre}), \text{Revenues}(ID, \text{amount}), \text{Director}(ID, \text{dir}), \text{amount} \geq \$100M \]

\[ V_1(I, Y) \Rightarrow \text{Movie}(I, T, Y, G), \text{Revenues}(I, A), I \geq 5000, A \geq \$200M \]

\[ V_2(I, A) \Rightarrow \text{Movie}(I, T, Y, G), \text{Revenues}(I, A) \]

\[ V_3(I, A) \Rightarrow \text{Movie}(I, T, Y, G), \text{Director}(I, D), I \leq 3000 \]

View atoms that can contribute to Movie:

\[ V_4(ID, \text{year'}), V_5(ID, A'), V_6(ID, \text{dir'}, \text{y'}) \]

Next Candidate Rewriting

<table>
<thead>
<tr>
<th>Movie(ID, year, genre)</th>
<th>Revenues(ID, amount)</th>
<th>Director(ID, dir)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_4(ID, \text{year}) )</td>
<td>( V_3(ID, Y') )</td>
<td>( V_6(ID, \text{dir'}, y') )</td>
</tr>
<tr>
<td>( V_5(ID, A') )</td>
<td>( V_3(ID, \text{amount}) )</td>
<td></td>
</tr>
<tr>
<td>( V_6(ID, \text{dir'}, \text{year}) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ q_4'(ID, \text{dir}) \Rightarrow \neg V_5(ID, A'), V_6(ID, \text{amount}), V_6(ID, \text{dir'}, y'), \text{amount} \geq \$100M \]

The Bucket Algorithm: Summary

- Cuts down the number of rewriting that need to be considered, especially if views apply many interpreted predicates.
- The search space can still be large because the algorithm does not consider the interactions between different subgoals.
  - See next example.

The Buckets and Cartesian product

<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_4(ID, \text{year}) )</td>
<td>( V_3(ID, Y') )</td>
<td>( V_6(ID, \text{dir'}, y') )</td>
</tr>
<tr>
<td>( V_5(ID, A') )</td>
<td></td>
<td>( V_3(ID, \text{amount}) )</td>
</tr>
<tr>
<td>( V_6(ID, \text{dir'}, \text{year}) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Consider first candidate rewriting: first V1 subgoal is redundant, and V1 and V4 are mutually exclusive.

\[ q_4'(ID, \text{dir}) \Rightarrow \neg V_5(ID, X), V_5(ID, y'), q_4'(ID, \text{dir}, y') \]

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

\[ Q(\text{title, year, dir}) \Rightarrow \text{Movie}(ID, \text{title}, \text{year, genre}), \text{Director}(ID, \text{dir}), \text{Actor}(ID, \text{dir}) \]

\[ V_3(D, A) \Rightarrow \neg \text{Director}(I, D), \text{Actor}(I, A) \]

Intuition: The variable I is not in the head of V₃, hence V₃ 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!

MinCon Descriptions (MCDs)

An atomic fragment of the ultimate containment mapping

\[ Q(\text{title}, \text{act}, \text{dir}) : \neg \text{Movie}(\text{ID}, \text{title}, \text{year}, \text{genre}), \]
\[ \text{Director}(\text{ID}, \text{dir}), \text{Actor}(\text{ID}, \text{act}) \]
\[ V(I,D,A) : \neg \text{Director}(I,D), \text{Actor}(I,A) \]
\[ \text{MCD:} \quad \begin{cases} \text{ID} \rightarrow I \\ \text{dir} \rightarrow D \\ \text{act} \rightarrow A \end{cases} \]
\[ \text{covered subgoals of } Q : \{2,3\} \]

MCDs: Detail 1

\[ Q(\text{title}, \text{year}, \text{dir}) : \neg \text{Movie}(\text{ID}, \text{title}, \text{year}, \text{genre}), \]
\[ \text{Director}(\text{ID}, \text{dir}), \text{Actor}(\text{ID}, \text{dir}) \]
\[ V(I,D,A) : \neg \text{Director}(I,D), \text{Actor}(I,A) \]
\[ \text{Need to specialize the view first:} \]
\[ V'(I,D,A) : \neg \text{Director}(I,D), \text{Actor}(I,D) \]
\[ \text{MCD:} \quad \begin{cases} \text{ID} \rightarrow I \\ \text{dir} \rightarrow D \end{cases} \]
\[ \text{covered subgoals of } Q : \{2,3\} \]

MCDs: Detail 2

\[ Q(\text{title}, \text{year}, \text{dir}) : \neg \text{Movie}(\text{ID}, \text{title}, \text{year}, \text{genre}), \]
\[ \text{Director}(\text{ID}, \text{dir}), \text{Actor}(\text{ID}, \text{dir}) \]
\[ V(I,D,D) : \neg \text{Director}(I,D), \text{Actor}(I,D), \]
\[ \text{Movie}(I,T,Y,G) \]
\[ \text{Note: the third subgoal of the view is not included in the MCD.} \]
\[ \text{MCD:} \quad \begin{cases} \text{ID} \rightarrow I \\ \text{dir} \rightarrow D \end{cases} \]
\[ \text{covered subgoals of } Q \text{ still: } \{2,3\} \]

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:
\[ V_s(I,T,Y,G) : \neg \text{Movie}(I,T,Y,G), \text{Director}(I,D), \text{Actor}(I,D) \]
And the following tuple in \( V_s \):
\[ V_s(79,\text{Manhattan},1979,\text{Comedy}) \]
Then we can infer the tuple:
\[ \text{Movie}(79,\text{Manhattan},1979,\text{Comedy}) \]
Hence, the following ‘rule’ is sound:
\[ \text{IN}_s : \text{Movie}(I,T,Y,G) : \neg V_s(I,T,Y,G) \]
Skolem Functions

$V_7(I,T,Y,G) : - Movie(I,T,Y,G), Director(I,D), Actor(I,D)$

Now suppose we have the tuple $V_7(79, Manhattan, 1979, Comedy)$

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

IN$_2$: $Director(I,f_1(I,T,Y,G)):- V_7(I,T,Y,G)$

IN$_3$: $Actor(I,f_1(I,T,Y,G)):- V_7(I,T,Y,G)$

Inverse Rules in General

$Q_2(title, year, genre) : - Movie(ID, title, year, genre)$

Given $Q_2$, the rewriting would include:

IN$_1$, IN$_2$, IN$_3$, Q$_2$.

Given input: $V_7(79, Manhattan, 1979, Comedy)$

Inverse rules produce:

Movie(79, Manhattan, 1979, Comedy)
Director(79, f$_1(79, Manhattan, 1979, Comedy)$)
Actor(79, f$_1(79, Manhattan, 1979, Comedy)$)
Movie(Manhattan, 1979, 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.
View Instances = Possible DB’s

Consider the two views:

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

And suppose the extensions of the views are:

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

Possible Databases

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

DB1. \{Allen, Keaton\}, \{Copolla, Pacino\}
DB2. \{Allen, Pacino\}, \{Copolla, 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.

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 \( i \).

[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 \( i \).

Certain Answers: Example

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

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

Under open-world assumption:
  – no certain answers.

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

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

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

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

\( q \) has a certain tuple iff \( G \) is not 3-colorable

• 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.
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
    - We call it the “target schema”
    - 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

Definition: Data Exchange Setting
Data Exchange setting is a tuple \((S,T,I,\Sigma)\)
- Schema \(S\)
- Schema \(T\)
- Instance \(I\) of \(S\)
- Mappings \(\Sigma\) from \(S\) to \(T\)

5.1 Data Exchange Solutions

Example: Solutions

<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>John</td>
<td>Peter</td>
<td>Chicago</td>
<td>1</td>
<td>Chicago</td>
<td>(312) 123 456</td>
</tr>
<tr>
<td>Alice</td>
<td>Alice</td>
<td>Chicago</td>
<td>2</td>
<td>Chicago</td>
<td>(312) 567 890</td>
</tr>
<tr>
<td>Bob</td>
<td>Bob</td>
<td>New York</td>
<td>3</td>
<td>New York</td>
<td>(465) 111 222</td>
</tr>
</tbody>
</table>

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

Can we come up with a solution?

5.1 Number of Solutions

- How many solutions exist?
  - Depends on 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 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>X</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

<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>X</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 Data Exchange 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(a₁, ..., aₙ) in J is mapped to a tuple in J’:
  h(R(a₁), ..., h(aₙ)) in J’

Definition: Universal solution

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

- There exists a homomorphism from J to J’
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) 123-456</td>
<td>Y</td>
</tr>
<tr>
<td>Alice</td>
<td>Chicago</td>
<td>(312) 678-901</td>
<td>X</td>
</tr>
<tr>
<td>Bob</td>
<td>New York</td>
<td>(415) 234-567</td>
<td>C</td>
</tr>
</tbody>
</table>

Above is universal solution.

How to map to below non-universal solution?
Replace generic labelled Nulls with values:
X = Hometown, Y = 111-322-3454, C = other town,
A = A

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) 123-456</td>
<td>111-322-3454</td>
</tr>
<tr>
<td>Alice</td>
<td>Chicago</td>
<td>(312) 678-901</td>
<td>A</td>
</tr>
<tr>
<td>Bob</td>
<td>New York</td>
<td>(415) 234-567</td>
<td>D</td>
</tr>
</tbody>
</table>

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)

• Schema mappings (st-tgds) are tuple-generating dependencies
  • What other tgd’s do we know
  • Foreign keys
  • How did we solve violations to FKs?
    • The chase!
    – Chase produces universal solution!

5.2 Computing Solutions

• 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

5.2 Skolem Functions

Example: Skolem Functions
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 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

Example: Skolem Functions

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

Annotations
- Annotate each target attribute connected to a source attribute with that source attribute
- Propagate annotations according to the following rules
  - Propagate annotations from attributes to relations
  - Propagate annotations from relations to attributes
    - Only if attribute uses existentially quantified variable
  - Propagate annotations between target attributes connected by equality edges

Example: Skolem Functions

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

1) Initialize with source attribute names

Example: Skolem Functions

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

2) Propagate to parent and back to children
5.2 Skolem Functions

- Clio Schema Graph Algorithm
- Skolem functions
  - Derive skolem function arguments from the schema graph annotations of an element

Example: Skolem Functions

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

- For variable \(f\) (id, address) we assign sk1(a,b,c)
- For variable \(g\) (age) we assign sk2(a,b,c)

5.2 Executable Transformations

- SQL Code Generation Example
  - For each tgd mentioning a target relation \(R\) we generate a query fragment
  - All query fragments for \(R\) are “unioned” together
  - A query fragment is
    - A FROM and WHERE clause that is a direct translation of the LHS of a tgd into SQL
    - A SELECT clause corresponding the \(R\) atom in the RHS using attributes from the FROM clause can the skolem functions we have determined in the previous step

Example: Skolem Functions

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

- For Person atom in RHS:
  - SELECT name, address, office-phone AS address
  - SELECT age, address, office-phone AS age
  - FROM Person

- For Address atom in RHS:
  - SELECT name, address, office-phone AS address
  - FROM Person

5.3 Recap Data Exchange Steps

- Schema Matching
- Generate Schema Mappings
  - Use constraints
- Generate Executable Transformations
  - SQL, XSLT, XQuery
  - Skolems for missing value
- 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
6. Data Warehousing

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

6. What is Datawarehousing?

• Problem: Data Analysis, Prediction, Mining
  – Example: Walmart
  – Transactional databases
    • Run many “cheap” updates concurrently
    • E.g., each store has a database storing its stock and sales
  – Complex Analysis over Transactional Databases?
    • Want to analyze across several transactional databases
    • E.g., compute total Walmart sales per month
    • Distribution and heterogeneity
    • Want to run complex analysis over large datasets
    • Resource consumption of queries affects normal operations on
      transactional databases

• Solution:
• Performance
  – Store data in a different system (the datawarehouse) for analysis
  – Bulk-load data to avoid wasting performance on concurrency control during analysis
• Heterogeneity and Distribution
  – Preprocess data coming from transactional databases to clean it and translate it into a unified format before bulk-loading

6. Datawarehousing Process

• 1) Design a schema for the warehouse
• 2) Create a process for preprocessing the data
• 3) Repeat
  – A) Preprocess data from the transactional databases
  – B) Bulk-load it into the warehouse
  – C) Run analytics

6. Overview

• The multidimensional datamodel (cube)
  – Multidimensional data model
  – Relational implementations
• Preprocessing and loading (ETL)
• Query language extensions
  – ROLL UP, CUBE, …
• Query processing in datawarehouses
  – Bitmap indexes
  – Query answering with views
  – Self-tuning
6. Multidimensional Datamodel

- Analysis queries are typically aggregating lower level facts about a business
  - The revenue of Walmart in each state (country, city)
  - The amount of toy products in a warehouse of a company per week
  - The call volume per zip code for the Sprint network
- Commonality among these queries:
  - At the core are facts: a sale in a Walmart store, a toy stored in a warehouse, a call made by a certain phone
  - Data is aggregated across one or more dimensions
    - These dimensions are typically organized hierarchically: year – month – day – hour, country – state - zip
  - Example
    - The revenue (sum of sale amounts) of Walmart in each state

6. Example 2D

<table>
<thead>
<tr>
<th></th>
<th>2014</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Jan</td>
<td>Feb</td>
</tr>
<tr>
<td>Toy</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Puppet</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Fishing rod</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Model of Dinky</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Books</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Mobile device</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>King Lear</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

6. Generalization to multiple dimensions

- Given a fixed number of dimensions
  - E.g., product type, location, time
- Given some measure
  - E.g., number of sales, items in stock, …
- In the multidimensional datamodel we store facts: the values of measures for a combination of values for the dimensions

6. Data cubes

- Given $n$ dimensions
  - E.g., product type, location, time
- Given $m$ measures
  - E.g., number of sales, items in stock, …
- A datacube (datahypercube) is an $n$-dimensional datastucture that maps values in the dimensions to values for the $m$ measures
  - Schema: $D_1, \ldots, D_n, M_1, \ldots, M_m$
  - Instance: a function $dom(D_1) \times \ldots \times dom(D_n) \rightarrow dom(M_1) \times \ldots \times dom(M_m)$
6. Dimension Example

- **Location**
  - **Levels**: location, state, city

6. Dimension Schema

- **Schema of a Dimension**
  - A set $D$ of category attributes $D_1, \ldots, D_n$
    - These correspond to the levels
  - A partial order $\rightarrow$ over $D$ which represents parent-child relationships in the hierarchy
    - These correspond to upward edges in the hierarchy
    - $\text{Top}_D$ is larger than anything else
      - For every $D_i, D_i \rightarrow \text{Top}_D$
    - There exists $D_{\text{min}}$ which is smaller than anything else
      - For every $D_i, D_{\text{min}} \rightarrow D_i$

6. Dimension Schema Example

- **Schema of Location Dimension**
  - Set of categories $D = \{\text{location, state, city}\}$
  - Partial order
    - $\{\text{city} \rightarrow \text{state, city} \rightarrow \text{location, state} \rightarrow \text{location}\}$
  - $\text{Top}_D = \text{location}$
  - $D_{\text{min}} = \text{city}$

6. Cells, Facts, and Measures

- Each **cell** in the cube corresponds to a combination of elements from each dimension
  - **Facts** are non-empty cells
  - Cells store **measures**
- Cube for a combination of levels of the dimension

6. Remarks

- In principle there does not have to exist an order among the elements at one level of the hierarchy
  - E.g., cities
- Hierarchies do not have to be linear

Facts

- Targets of analytics
  - E.g., revenue, #sales, #stock
- A fact is uniquely defined by the combination of values from the dimensions
  - E.g., for dimensions time and location
    - Revenue in **Illinois** during Jan 2015
- **Granularity**: Levels in the dimension hierarchy corresponding to the fact
  - E.g., state, month
Facts (Event vs. Snapshot)

- **Event Facts**
  - Model real-world events
  - E.g., Sale of an item

- **Snapshot Facts**
  - Temporal state
  - A single object (e.g., a book) may contribute to several facts
  - E.g., number of items in stock

Measures

- A measure describes a fact
  - May be derived from other measures

- Two components
  - Numerical value
  - Formula (optional): how to derive it
    - E.g., \( \text{avg(revenue)} = \frac{\text{sum(revenue)}}{\text{count(revenue)}} \)

- We may associate multiple measures to each cell
  - E.g., number of sales and total revenue

Measures - Granularity

- Similar to facts, measures also have a granularity
- How to change granularity of a measure?
- Need algorithm to combine measures
  - Additive measures
    - Can be aggregated along any dimension
  - Semi-additive/non-additive
    - Cannot be aggregated along some/all dimensions
    - E.g., snapshot facts along time dimension
      - Number of items in stock at Jan + Feb + … + items in stock during year
    - Median of a measure

Design Process (after Kimball)

- Comparison to classical relational modeling
  - Analysis driven
    - No need to model all existing data and relationships relevant to a domain
    - Limit modeling to information that is relevant for predicted analytics
  - Redundancy
    - Tolerate redundancy for performance if reasonable
      - E.g., in dimension tables to reduce number of joins

Design Process – Steps

1) Select relevant business processes
   - E.g., order shipping, sales, support, stock management

2) Select granularity
   - E.g., track stock at level of branches or regions

3) Design dimensions
   - E.g., time, location, product, …

4) Select measures
   - E.g., revenue, cost, #sales, items in stock, #support requests

Design Process Example

- Coffee shop chain
  - Processes
    - Sell coffee to customers
    - Buy ingredients from suppliers
    - Ship supplies to branches
    - Pay employees
    - HR (hire, advertise positions, …)
  - Which process is relevant to be analysed to increase profits?
Design Process Example

1) Selecting process(es)
   - sell coffee to customers

2) Select granularity
   - Single sale?
   - Sale per branch/day?
   - Sale per city/year?

3) Determine relevant dimensions
   - Location (country, state, city, zip, shop)
   - Time (year, month, day)
   - Product (type, brand, product)

4) Select measures
   - cost, revenue, profit?
Star Schema

- A data cube is represented as a set of dimension tables and a fact table
- Dimension tables
  - For each dimension schema \( D = (D_1, \ldots, D_k, \text{Top}_D) \) we create a relation
    \( D(PK, D_1, \ldots, D_k) \)
    - Here \( PK \) is a primary key, e.g., \( D_{min} \)
- Fact table
  - \( F(FK_1, \ldots, FK_n, M_1, \ldots, M_m) \)
  - Each \( FK_i \) is a foreign key to \( D_i \)
  - Primary key is the combination of all \( FK_i \)

Star Schema - Remarks

- Dimension tables have redundancy
  - Values for higher levels are repeated
- Fact table is in 3NF
- \( \text{Top}_D \) does not have to be stored explicitly
- Primary keys for dimension tables are typically generated (surrogate keys)
  - Better query performance by using integers

Snowflake Schema

- A data cube is represented as a set of dimension tables and a fact table
- Dimension tables
  - For each dimension schema \( D = (D_1, \ldots, D_k, \text{Top}_D) \) we create a relation multiple relations connected through FKs
    \( D_i(PK, A_1, \ldots, A_l, FK_j) \)
    - \( A_l \) is a descriptive attribute
    - \( FK_j \) is foreign key to the immediate parent(s) of \( D_i \)
- Fact table
  - \( F(FK_1, \ldots, FK_n, M_1, \ldots, M_m) \)
  - Each \( FK_i \) is a foreign key to \( D_i \)
  - Primary key is the combination of all \( FK_i \)

Snowflake Schema - Remarks

- Avoids redundancy
- Results in much more joins during query processing
- Possible to find a compromise between snowflake and star schema
  - E.g., use snowflake for very fine-granular dimensions with many levels

Snowflake Schema - Example

- Coffee chain example

6. Extract-Transform-Load (ETL)

- The preprocessing and loading phase is called \textit{extract-transform-load (ETL)} in datawarehousing
- Many commercial and open-source tools available
- ETL process is modeled as a workflow of operators
  - Tools typically have a broad set of build-in operators: e.g., key generation, replacing missing values, relational operators,
  - Also support user-defined operators
6. Extract-Transform-Load (ETL)

- Some ETL tools
  - Pentaho Data Integration
  - Oracle Warehouse Builder (OWB)
  - IBM Infosphere Information Server
  - Talend Studio for Data Integration
  - CloverETL
  - Cognos Data Manager
  - Pervasive Data Integrator
  - …

6. Operators supported by ETL

- Many of the preprocessing and cleaning operators we already know
  - Surrogate key generation (like creating existentials with skolens)
  - Fixing missing values
    - With default value, using trained model (machine learning)
  - Relational queries
    - E.g., union of two tables or joining two tables
  - Extraction of structured data from semi-structured data and/or unstructured data
  - Entity resolution, data fusion

6. ETL Process

- Operators can be composed to form complex workflows

6. Typical ETL operators

- Elementizing
  - Split values into more fine-granular elements
- Standardization
- Verification
- Matching with master data
- Key generation
- Schema matching, Entity resolution/ Deduplication, Fusion

6. Typical ETL operators

- Control flow operators
  - AND/OR
  - Fork
  - Loops
  - Termination
    - Successful
    - With warning/errors

6. Typical ETL operators

- Elementizing
  - Split non 1NF data into individual elements
- Examples
  - name: “Peter Gertsen” -> firstname: “Peter”, lastname: “Gertsen”
  - Address: “10 W 31st, Chicago, IL 60616” -> street = “10 W 31st”, city = “Chicago”, state = “IL”, zip = “60616”
6. Typical ETL operators

**Standardization**
- Expand abbreviation
- Resolve synonyms
- Unified representation of, e.g., dates

**Examples**
- "IL" -> "Illinois"
- "m/w", "M/F" -> "male/female"
- "Jan", "01", "January", "january" -> "January"
- "St" -> "Street", "Dr" -> "Drive", ...

**Verification**
- Same purpose as constraint based data cleaning but typically does not rely on constraints, but, e.g., regular expression matching

**Examples**
- Phone matches "[0-9]{3}-[0-9]{3}-[0-9]{4}"
- For all t in Tokens(product description), t exists in English language dictionary

6. Matching master data (lookup)
- Check and potentially repair data based on available master data

**Examples**
- E.g., using a clean lookup table with (city,zip) replace the city in each tuple if the pair (city,zip) does not occur in the lookup table

6. Metadata management
- As part of analysis in DW data is subjected to a complex pipeline of operations
  - Sources
  - ETL
  - Analysis queries
- -> important, but hard, to keep track of what operations have been applied to data and from which sources it has been derived
- Need metadata management
  - Including provenance (later in this course)

6. Querying DW
- Targeted model (cube vs. relational)
  - Design specific language for datacubes
  - Add suitable extensions to SQL
- Support typical analytical query patterns
  - Multiple parallel grouping criteria
    - Show total sales, subtotal per state, and subtotal per city
    - -> three subqueries with different group-by in SQL
  - Windowed aggregates and ranking
    - Show 10 most successful stores
    - Show cumulative sales for months of 2016
    - E.g., the result for Feb would be the sum of the sales for Jan + Feb

6. Querying DW
- Targeted model (cube vs. relational)
  - Design specific language for datacubes
    - MDX
  - Add suitable extensions to SQL
    - GROUPING SETS, CUBE, ...
    - Windowed aggregation using OVER(), PARTITION BY, ORDER BY, window specification
    - Window functions
      - RANK, DENSE_RANK...
6. Cube operations

• Roll-up
  – Move from fine-granular to more coarse-granular in one or more dimensions of a datacube
  • E.g., sales per (city,month,product category) to Sales per (state,year, product category)

• Drill-down
  – Move from coarse-granular to more fine-granular in one of more dimensions
  • E.g., phonecalls per (city,month) to phonecalls per (zip,month)

• Drill-out
  – Add additional dimensions
    • special case of drill-down starting from TopD in dimension(s)
    • E.g., sales per (city, product category) to Sales per (state,year, product category)

• Drill-in
  – Remove dimension
    • special case for roll-up move to TopD for dimension(s)
    • E.g., phonecalls per (city,month) to phonecalls per (month)

6. SQL Extensions

• Slice
  – Select data based on restriction of the values of one dimension
  • E.g., sales per (city,month) -> sales per (city) in Jan

• Dice
  – Select data based on restrictions of the values of multiple dimensions
  • E.g., sales per (city,month) -> sales in Jan for Chicago and Washington DC

• GROUPING SETS

  • GROUP BY GROUPING SETS ((set_1), ..., (set_n))
  • Explicitly list sets of group by attributes
  • Semantics:
    – Equivalent to UNION over duplicates of the query each with a group by clause GROUP BY set
    – Schema contains all attributes listed in any set
    – For a particular set, the attribute not in this set are filled with NULL values
6. GROUPING SETS

SELECT quarter, city, product_typ, SUM(profit) AS profit
FROM facttable F, time T, location L, product P
WHERE F.TID = T.TID AND F.LID = L.LID AND F.PID = P.PID
GROUP BY GROUPING SETS
( (quarter, city),
 (quarter, product_typ))

<table>
<thead>
<tr>
<th>quarter</th>
<th>city</th>
<th>product_typ</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010 Q1</td>
<td>Books</td>
<td>8347</td>
<td></td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Books</td>
<td>7836</td>
<td></td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Gardening</td>
<td>12300</td>
<td></td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Chicago</td>
<td>12344</td>
<td></td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Seattle</td>
<td>124345</td>
<td></td>
</tr>
</tbody>
</table>

6. GROUPING SETS

• Problem:
  – How to distinguish between NULLs based on grouping sets and NULL values in a group by column?

GROUP BY GROUPING SETS
( (quarter, city), (quarter, product_typ), (quarter, product_typ, city))

<table>
<thead>
<tr>
<th>quarter</th>
<th>city</th>
<th>product_typ</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010 Q1</td>
<td></td>
<td>Books</td>
<td>8347</td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Books</td>
<td>7836</td>
<td></td>
</tr>
<tr>
<td>2012 Q2</td>
<td></td>
<td>Gardening</td>
<td>12300</td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Chicago</td>
<td>12344</td>
<td></td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Seattle</td>
<td>124345</td>
<td></td>
</tr>
</tbody>
</table>

6. GROUPING SETS

6. CUBE

GROUP BY CUBE (set)
• Group by all 2^n subsets of set

GROUP BY CUBE (A,B,C)
  = GROUP BY GROUPING SETS
      ( (A,B), (A,C), (A) )
      ( (B), (C) )
GROUP BY GROUPING SETS ( (A,B), (B,C)
GROUP BY GROUPING SETS ( (D,E), (D) )
GROUP BY GROUPING SETS
( (A,B,D,E), (A,B,D), (B,C,D,E), (B,C,D) )
6. CUBE

- **GROUP BY ROLLUP** \((A_1, \ldots, A_n)\)
  - Group by all prefixes
  - Typically different granularity levels from single dimension hierarchy, e.g., year-month-day
    - Database can often find better evaluation strategy

\[
\text{GROUP BY ROLLUP} (A, B, C) = \text{GROUP BY GROUPING SETS} \{ \\
(A,B,C), \\
(A,B), \\
(A), \\
() \\
\}
\]

6. OVER clause

- **Agg OVER** (partition-clause, order-by,window-specification)
- New type of aggregation and grouping where
  - Each input tuple is paired with the aggregation result for the group it belongs to
  - More flexible grouping based on order and windowing
  - New aggregation functions for ranking queries
    - e.g., RANK, DENSE_RANK

6. OVER clause

- **Agg OVER** (partition-clause, order-by,window-specification)
- New type of aggregation and grouping where

\[
\text{window frame preceding} ::= \{ \\
\text{UNBOUNDED PRECEDING} | n \text{ PRECEDING} | \text{CURRENT ROW} \} \\
\text{window frame following} ::= \{ \\
\text{UNBOUNDED FOLLOWING} | n \text{ FOLLOWING} | \text{CURRENT ROW} \}
\]

6. GROUPING SETS

\[
\text{SELECT year, month, city, profit} \sum(\text{profit}) \text{ OVER} () \text{ AS ttl} \\
\text{FROM sales} \\
\]

- For each tuple build a set of tuples belonging to the same window
  - Compute aggregation function over window
  - Return each input tuple paired with the aggregation result for its window
- OVER() = one window containing all tuples

6. GROUPING SETS

\[
\text{SELECT year, month, city, profit} \sum(\text{profit}) \text{ OVER (PARTITION BY year) AS ttl} \\
\text{FROM sales} \\
\]

- **PARTITION BY**
  - only tuples with same partition-by attributes belong to the same window
- **LIKE GROUP BY**

<table>
<thead>
<tr>
<th>Year</th>
<th>Month</th>
<th>City</th>
<th>Profit</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>Chicago</td>
<td>10</td>
<td>92</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>Chicago</td>
<td>5</td>
<td>92</td>
</tr>
<tr>
<td>2010</td>
<td>3</td>
<td>Chicago</td>
<td>20</td>
<td>92</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>Chicago</td>
<td>45</td>
<td>92</td>
</tr>
<tr>
<td>2010</td>
<td>1</td>
<td>New York</td>
<td>12</td>
<td>47</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>New York</td>
<td>12</td>
<td>47</td>
</tr>
<tr>
<td>2012</td>
<td>1</td>
<td>New York</td>
<td>12</td>
<td>47</td>
</tr>
</tbody>
</table>
6. GROUPING SETS

SELECT year, month, city
SUM(profit) OVER (ORDER BY year, month) AS ttl
FROM sales

• ORDER BY
  – Order tuples on these expressions
  – Only tuples which are <= to the order as the current tuple belong to the same window

• E.g., can be used to compute an accumulate total

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>city</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>Chicago</td>
<td>10</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>Chicago</td>
<td>5</td>
</tr>
<tr>
<td>2010</td>
<td>3</td>
<td>Chicago</td>
<td>20</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>Chicago</td>
<td>45</td>
</tr>
<tr>
<td>2010</td>
<td>1</td>
<td>New York</td>
<td>12</td>
</tr>
</tbody>
</table>

6. GROUPING SETS

SELECT year, month, city
SUM(profit) OVER (ORDER BY year, month) AS ttl
FROM sales

• ORDER BY
  – Order tuples on these expressions
  – Only tuples which are <= to the order as the current tuple belong to the same window

• E.g., can be used to compute an accumulate total

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>city</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>Chicago</td>
<td>10</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>Chicago</td>
<td>5</td>
</tr>
<tr>
<td>2010</td>
<td>3</td>
<td>Chicago</td>
<td>20</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>Chicago</td>
<td>45</td>
</tr>
<tr>
<td>2010</td>
<td>1</td>
<td>New York</td>
<td>12</td>
</tr>
</tbody>
</table>

6. GROUPING SETS

SELECT year, month, city
SUM(profit) OVER (PARTITION BY year ORDER BY month) AS ttl
FROM sales

• ORDER BY
  – Order tuples on these expressions
  – Only tuples which are <= to the order as the current tuple belong to the same window

• E.g., can be used to compute an accumulate total

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>city</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>Chicago</td>
<td>10</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>Chicago</td>
<td>5</td>
</tr>
<tr>
<td>2010</td>
<td>3</td>
<td>Chicago</td>
<td>20</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>Chicago</td>
<td>45</td>
</tr>
<tr>
<td>2010</td>
<td>1</td>
<td>New York</td>
<td>12</td>
</tr>
</tbody>
</table>

6. GROUPING SETS

SELECT year, month, city
SUM(profit) OVER (PARTITION BY year ORDER BY month) AS ttl
FROM sales

• ORDER BY
  – Order tuples on these expressions
  – Only tuples which are <= to the order as the current tuple belong to the same window

• E.g., can be used to compute an accumulate total

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>city</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>Chicago</td>
<td>10</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>Chicago</td>
<td>5</td>
</tr>
<tr>
<td>2010</td>
<td>3</td>
<td>Chicago</td>
<td>20</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>Chicago</td>
<td>45</td>
</tr>
<tr>
<td>2010</td>
<td>1</td>
<td>New York</td>
<td>12</td>
</tr>
</tbody>
</table>
6. GROUPING SETS

SELECT year, month, city
    SUM(profit) OVER (ORDER BY year, month
    RANGE BETWEEN 1 PRECEDING
    AND 1 FOLLOWING) AS ttl
FROM sales

• Explicit window specification
  – Requires ORDER BY
  – Determines which tuples “surrounding” the tuple according to the sort order to
    include in the window

6. MDX

- Multidimensional expressions (MDX)
  – Introduced by Microsoft
  – Query language for the cube data model
  – SQL-like syntax
    • Keywords have different meaning
    • MDX queries return a multi-dimensional report
      • 2D = spreadsheet
      • 3D or higher, e.g., multiple spreadsheets

6. MDX Query

• Basic Query Structure

SELECT <axis-spec>, ... FROM <cube-spec>, ... WHERE ( <select-spec> )

• Note!
  – Semantics of SELECT, FROM, WHERE not what
    you would expect knowing SQL
6. MXD - SELECT

- Specify concepts from dimensions
  - List all values as set, e.g., { [2010], [2011] }
  - Not necessarily from same level of hierarchy (e.g., mix years and months)

- Language constructs for accessing parents and children or members of a level in the hierarchy
  - CHILDREN: all direct children
    - E.g., [2010].CHILDREN = {[2010 Jan], …, [2010 Dec]}
  - PARENT: the direct parent
    - E.g., [2010 Jan].PARENT = [2010]
  - MEMBERS: all direct children
    - E.g., Time.Years.MEMBERS = {[1990], [1991], …, [2016]}
  - LASTCHILD: last child (according to order of children)
    - E.g., [2010].LASTCHILD = [2010 Dec]
  - NEXTMEMBER: right sibling on same level
    - E.g., [2010].NEXTMEMBER = [2011]
  - [a]:[b]: all members in interval between a and b

- Nesting of sets: CROSSJOIN
  - Project two dimensions into one
  - Forming all possible combinations

SELECT CROSSJOIN ( { Chicago, Schaumburg }, { [2010], [2011] } )
ON ROWS ( [2010], [2011].CHILDREN )
ON COLUMNS FROM PhoneCallsCube
WHERE ( Measures.numCalls )

<table>
<thead>
<tr>
<th></th>
<th>2010</th>
<th>2011</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chicago</td>
<td>1231</td>
<td>3231</td>
</tr>
<tr>
<td>Schaumburg</td>
<td>3232</td>
<td>1235</td>
</tr>
</tbody>
</table>

- Conditional selection of members: FILTER
  - One use members that fulfill condition
  - E.g., condition over aggregation result

SELECT FILTER ([2010].CHILDREN, (Sprint, numCalls) > (ATT, numCalls))
ON ROWS ( Chicago )
ON COLUMNS FROM PhoneCallsCube
WHERE ( Measures.numCalls )

6. Query Processing in DW

- Large topic, here we focus on two aspects
  - Partitioning
  - Query answering with materialized views

6. Partitioning

- Partitioning splits a table into multiple fragments that are stored independently
  - E.g., split across X disks, across Y servers

- Vertical partitioning
  - Split columns across fragments
    - E.g., R = {A,B,C,D}, fragment F1 = {A,B}, F2 = {C,D}
    - Either add a row id to each fragment or the primary key to be able to reconstruct

- Horizontal partitioning
  - Split rows
  - Hash vs. range partitioning
6. Partitioning

- Why partitioning?
  - Parallel/distributed query processing
  - Read/write fragments in parallel
  - Distribute storage load across disks/servers
  - Avoid reading data that is not needed to answer a query
    - Vertical
      - Only read columns that are accessed by query
    - Horizontal
      - Only read tuples that may match queries selection conditions

---

**Vertical Partitioning**

- Fragments $F_1$ to $F_n$ of relation $R$ such that
  - $\text{Sch}(F_1) \cup \text{Sch}(F_2) \cup \ldots \cup \text{Sch}(F_n) = \text{Sch}(R)$
  - Store row id or PK of $R$ with every fragment
  - Restore relation $R$ through natural joins

---

**Horizontal Partitioning**

- **Range** partitioning on attribute $A$
  - Split domain of $A$ into intervals representing fragments
  - E.g., tuples with $A = 15$ belong to fragment $[0,20]$  
  - Fragments $F_1$ to $F_n$ of relation $R$ such that
    - $\text{Sch}(F_1) = \ldots = \text{Sch}(F_n) = \text{Sch}(R)$
    - $R = F_1 \cup \ldots \cup F_n$

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Age</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>12,000</td>
<td>45</td>
<td>M</td>
</tr>
<tr>
<td>Alice</td>
<td>24,000</td>
<td>34</td>
<td>F</td>
</tr>
<tr>
<td>Bob</td>
<td>20,000</td>
<td>22</td>
<td>M</td>
</tr>
<tr>
<td>Gertrud</td>
<td>50,000</td>
<td>55</td>
<td>F</td>
</tr>
<tr>
<td>Pferdegert</td>
<td>14,000</td>
<td>23</td>
<td>M</td>
</tr>
</tbody>
</table>

- **Hash** partitioning on attribute $A$
  - Split domain of $A$ into $x$ buckets using hash function
  - E.g., tuples with $h(A) = 3$ belong to fragment $F_3$
  - $\text{Sch}(F_1) = \ldots = \text{Sch}(F_n) = \text{Sch}(R)$
  - $R = F_1 \cup \ldots \cup F_n$

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Age</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>12,000</td>
<td>45</td>
<td>M</td>
</tr>
<tr>
<td>Alice</td>
<td>24,000</td>
<td>34</td>
<td>F</td>
</tr>
<tr>
<td>Bob</td>
<td>20,000</td>
<td>22</td>
<td>M</td>
</tr>
<tr>
<td>Gertrud</td>
<td>50,000</td>
<td>55</td>
<td>F</td>
</tr>
<tr>
<td>Pferdegert</td>
<td>14,000</td>
<td>23</td>
<td>M</td>
</tr>
</tbody>
</table>

---

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