ILLINOIS INSTITUTE
OF TECHNOLOGY

CS520
Data Integration, Warehousing, and Provenance

Course Info

IIT DBGroup

Boris Glavic

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

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

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

Outline

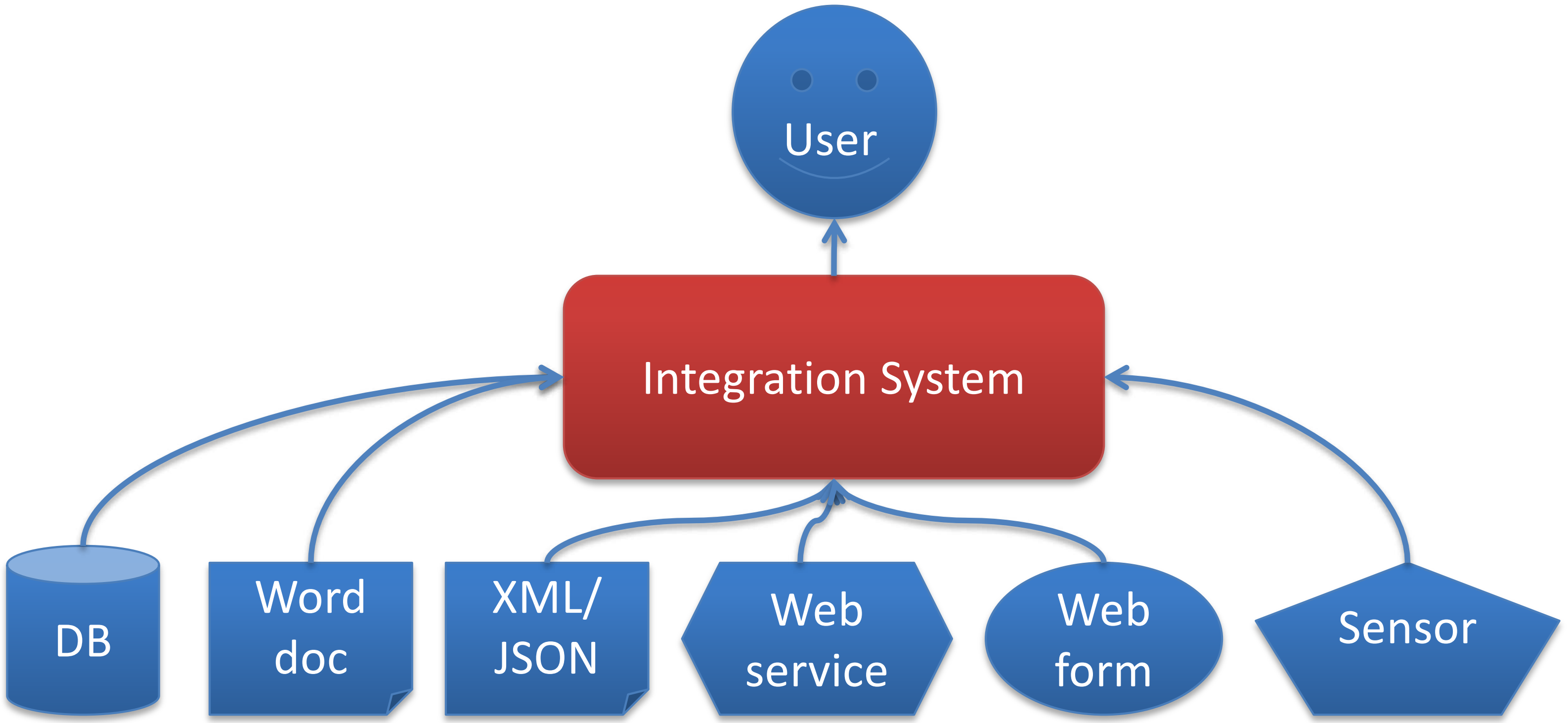
0) Course Info

1) Introduction
2) Data Preparation and Cleaning
3) Data Translation: Schema mappings, Virtual Data Integration, and Data Exchange
4) Data Warehousing
5) Big Data Analytics
6) Data Provenance

What is information integration?

ILLINOIS INSTITUTE OF TECHNOLOGY

- Combination of data and content from multiple sources into a common format
  - Completeness
  - Correctness
  - Efficiency



Why Information Integration?

- Data is already available, right?
- …, but
- **Heterogeneity**
  - Structural
    - Data model (relational, XML, unstructured)
    - Schema (if exists)
  - Semantic
    - Naming and identity conflicts
    - Data conflicts
  - Syntactic
    - Interfaces (web form, query language, binary file)

Why Information Integration?

- **Autonomy**
  - Sources may not give you unlimited access
    - Web form only support a fixed format of queries
    - Does not allow access to unlimited amounts of data
  - Source may not be available all the time
  - 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 sources
- **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)

Steps

1) Interfaces
2) Schema integration
3) Translate queries
4) Optimization
5) Send queries to sources
6) Gather query results
7) Entity resolution
8) Fusion
9) Curation / Cleaning
10) Return final results

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>
```

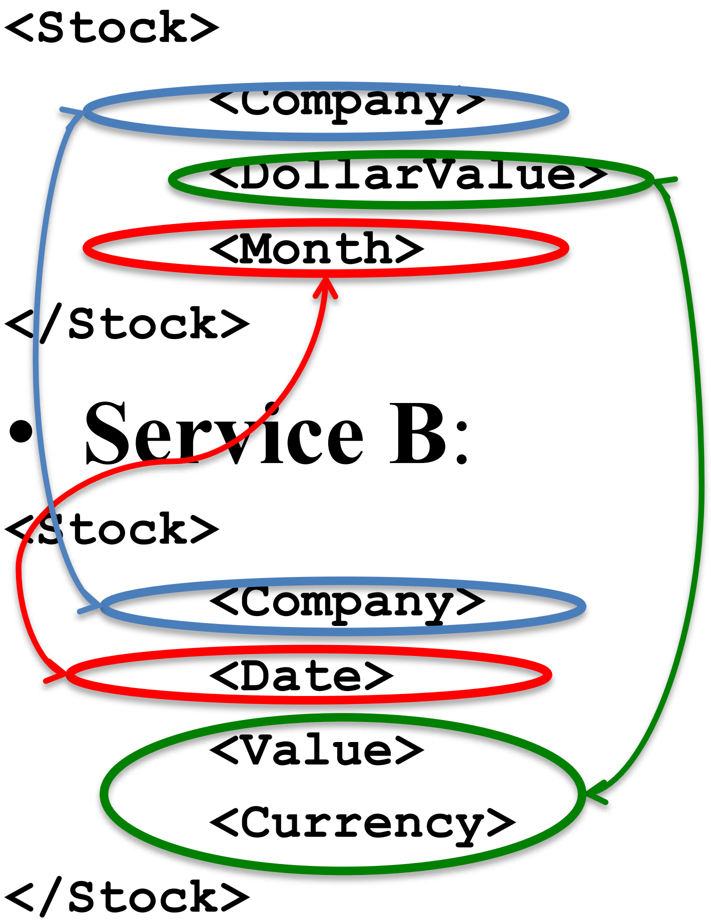
Steps

1) Interfaces
2) Schema integration
3) Translate queries
4) Optimization
5) Send queries to sources
6) Gather query results
7) Entity resolution
8) Fusion
9) Curation / Cleaning
10) Return final results

Example Integration Problem [2]

ILLINOIS INSTITUTE OF TECHNOLOGY

- **Service A:**



```
<Stock>
    <Company>
    <DollarValue>
    <Month>
</Stock>
```

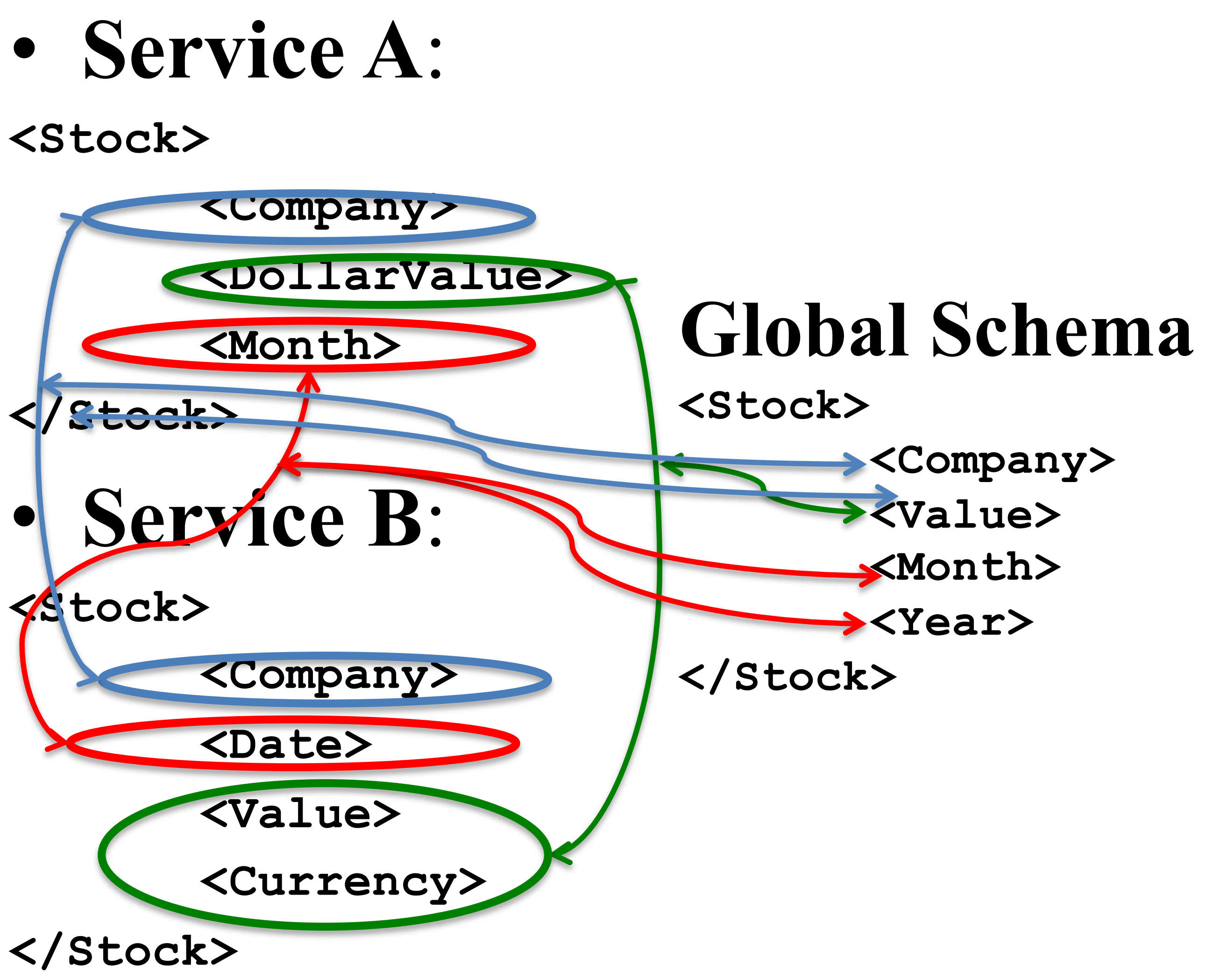
- **Service B:**

```
<Stock>
    <Company>
    <Date>
    <Value>
    <Currency>
</Stock>
```

Steps

1) Interfaces
2) Schema integration
3) Translate queries
4) Optimization
5) Send queries to sources
6) Gather query results
7) Entity resolution
8) Fusion
9) Curation / Cleaning
10) Return final results

Example Integration Problem [2]



- **Service A:**

```
<Stock>
    <Company>
    <DollarValue>
    <Month>
</Stock>
```

- **Service B:**

```
<Stock>
    <Company>
    <Date>
    <Value>
    <Currency>
</Stock>
```

Global Schema

```
<Stock>
    <Company>
    <Value>
    <Month>
    <Year>
</Stock>
```

Steps

1) Interfaces
2) Schema integration
3) Translate queries
4) Optimization
5) Send queries to sources
6) Gather query results
7) Entity resolution
8) Fusion
9) Curation / Cleaning
10) Return final results

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)**

<u>Steps</u>

1) Interfaces
2) Schema integration
3) Translate queries
4) Optimization
5) Send queries to sources
6) Gather query results
7) Entity resolution
8) Fusion
9) Curation / Cleaning
10) Return final results

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, …

Steps

1) Interfaces
2) Schema integration
3) Translate queries
4) Optimization
5) Send queries to sources
6) Gather query results
7) Entity resolution
8) Fusion
9) Curation / Cleaning
10) Return final results

Example Integration Problem [5]

- **Service A**: (IBM, 2014)
- **Service B**: (2014)

Steps

1) Interfaces
2) Schema integration
3) Translate queries
4) Optimization
5) Send queries to sources
6) Gather query results
7) Entity resolution
8) Fusion
9) Curation / Cleaning
10) Return final results

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>
…
```

Steps

1) Interfaces
2) Schema integration
3) Translate queries
4) Optimization
5) Send queries to sources
6) Gather query results
7) Entity resolution
8) Fusion
9) Curation / Cleaning
10) Return final results

Example Integration Problem [7]

- IBM vs. Integrated Business Machines

Steps

1) Interfaces
2) Schema integration
3) Translate queries
4) Optimization
5) Send queries to sources
6) Gather query results
7) Entity resolution
8) Fusion
9) Curation / Cleaning
10) Return final results

Example Integration Problem [8]

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

Steps

1) Interfaces
2) Schema integration
3) Translate queries
4) Optimization
5) Send queries to sources
6) Gather query results
7) Entity resolution
8) Fusion
9) Curation / Cleaning
10) Return final results

Example Integration Problem [9]

- **“Dirty Data”**
  - **Outliers**
    - E.g., $10M / unit not realistic
  - **Violations of constraints**
    - E.g., stock value has to be positive
  - **Format and type errors**
    - E.g., include $ in value or not
    - Value has to be a number
- **Service A:**

```
<DollarValue>-15</DollarValue>
<DollarValue>10000000.8</DollarValue>
<DollarValue>$24</DollarValue>
<DollarValue>five dollar</DollarValue>
<DollarValue>fad23e19hasd</DollarValue>
```

…

Steps

1) Interfaces
2) Schema integration
3) Translate queries
4) Optimization
5) Send queries to sources
6) Gather query results
7) Entity resolution
8) Fusion
9) Curation / Cleaning
10) Return final results

Example Integration Problem [10]

- Return final results:

```
<Stock>
        <Month>01</Month>
        <Value>105</Value>
</Stock>
…
<Stock>
        <Month>12</Month>
        <Value>107</Value>
</Stock>
```

Steps

1) Interfaces
2) Schema integration
3) Translate queries
4) Optimization
5) Send queries to sources
6) Gather query results
7) Entity resolution
8) Fusion
9) Curation / Cleaning
10) Return final results

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 exist
- So why do you 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 machine interpretable information on what type of information is stored in the attribute
  - **Undecidable computational problems**: e.g., 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 Extraction**
  - Extract data from unstructured sources / text
- **Data cleaning**:
  - Clean dirty data before integration
  - Conformance with a set of constraints
  - Deal with missing and outlier values
- **Entity resolution**
  - Determine which objects from multiple dataset represent the same real world entity
- **Data fusion**
  - Merge (potentially conflicting) data for the same entity

Relevant less general problems

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

Relevant less general problems

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

Relevant less general problems

- **Integration in Big Data Analytics**
  - Often "pay-as-you-go":
    - No or limited schema
    - Engines support wide variety of data formats
- **Provenance**
  - Information about the origin and creation process of data
  - Very important for integrated data
    - E.g., "from which data source is this part of my query result"

Webpage and Faculty

- **Course Info**
  - **Course Webpage**: http://cs.iit.edu/~glavic/cs520
  - **Discord**:
    - Used for announcements
    - Use it to discuss with me, TA, and fellow students
  - **Syllabus: http://www.cs.iit.edu/~glavic/cs520/2023-fall/syllabus/**
- **Faculty**
  - **Boris Glavic** (http://cs.iit.edu/~glavic)
  - **Email:** bglavic@iit.edu
  - **Phone**: 312.567.5205
  - **Office**: SB 206B

Workload and Grading

- **Exams (60%)**
  - Final (30%), Midterm (30%)
- **Homework Assignments** (preparation for exams!)
  - **Theory part**: Practice theory for final exam
  - **Lab part**: Practice the tools we discuss in class
- **Literature Review (20%)**
  - In groups of 3 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)

Workload and Grading

- **Data Curation Project(20%)**
  - In groups of 3 students (same groups as for literature review)
  - You will acquire and curate (clean, integrate, …) a real world dataset
  - This is open-ended, you can choose whatever tools you need, whatever domain you think is interesting, …
    - Only limitation is that you need to document your cleaning workflow using a **Vizier notebook** (so at lease some python is required)
    - https://vizierdb.info/
  - Steps:
    - Acquire or extract one or more real world datasets for a domain of choice
    - Gain an understanding of the data and identify data quality issues
    - Research tools that are suited for the data cleaning, integration, extraction tasks that you need to apply to create a correct and clean output dataset
    - Apply the tools and produce an output
  - Work will be submitted through git repositories on bitbucket.org that we will create for each group

Workload and Grading

- **Timeline:**
  - See course webpage for detailed dates
    - You are required to meet with the TA/Prof. several times for discussing the progress for the literature review and data curation project
  - Literature reviews and project presentations will be done in a block seminar towards the end of the semester (1-2 days)

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 notions of data provenance and know how to compute provenance

Fraud Policies

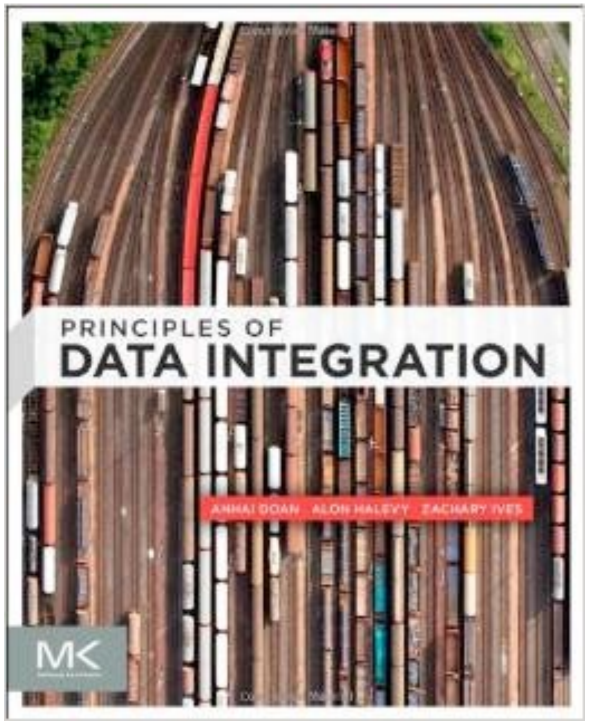
- All work has to be original!
  - Cheating = 0 points for review/exam
  - Possibly E in course and further administrative sanctions
  - Every dishonesty will be reported to office of academic honesty
- Late policy:
  - -20% per day
  - You have to give your presentation to pass the course!
  - No exceptions!

Fraud Policies cont.

- Literature Review:
  - Every student has to contribute in the presentation, report, and data curation project!
  - **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
  - ISBN-13: 978-0124160446
  - Prerequisites:
    - CS 425



Additional Reading

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

Outline

ILLINOIS INSTITUTE OF TECHNOLOGY

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

ILLINOIS INSTITUTE OF TECHNOLOGY

CS520
Data Integration, Warehousing, and Provenance

1. Introduction

IIT DBGroup

Boris Glavic

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

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

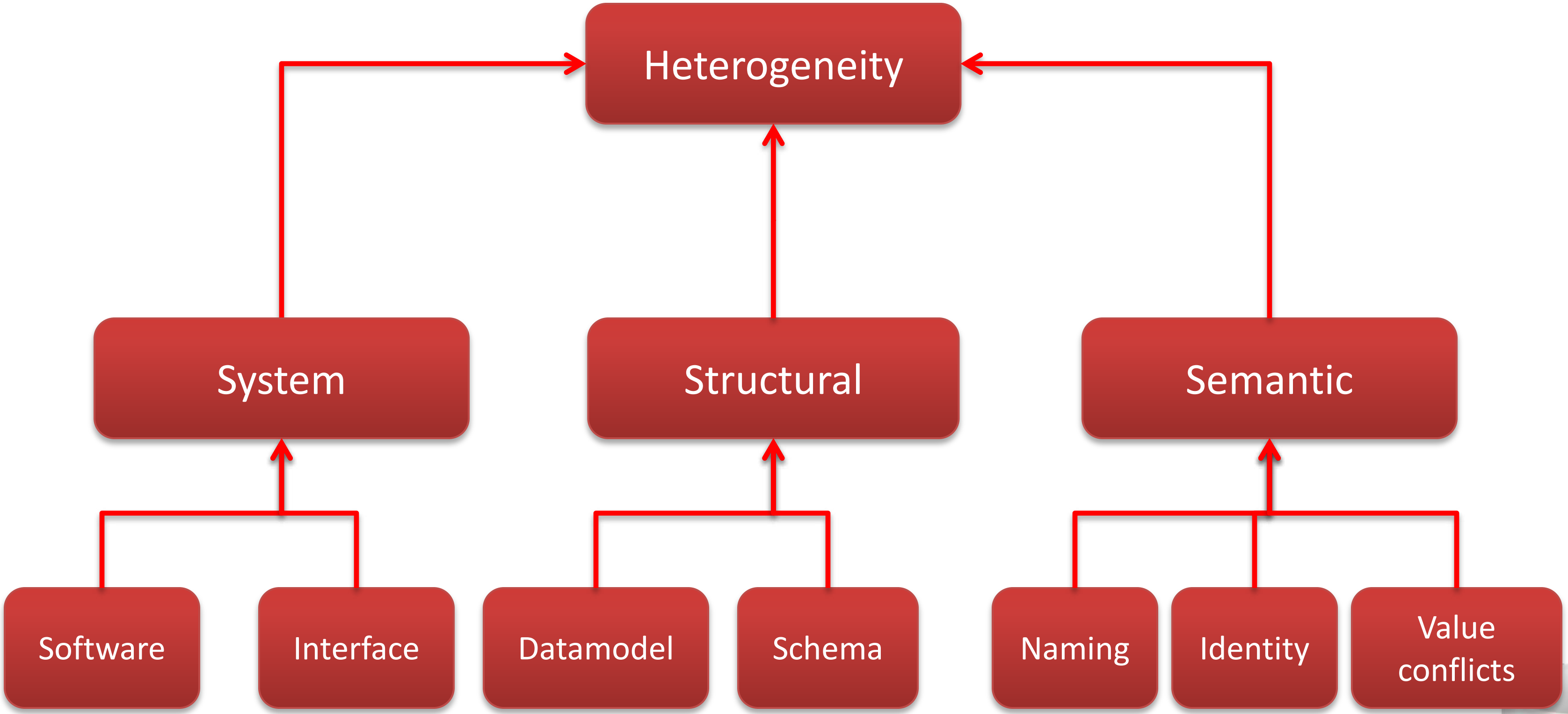
8) Data Provenance

Overview

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

1.1 Heterogeneity +Autonomy

- Taxonomy of Heterogeneity



Heterogeneity

System

Structural

Semantic

Software

Interface

Datamodel

Schema

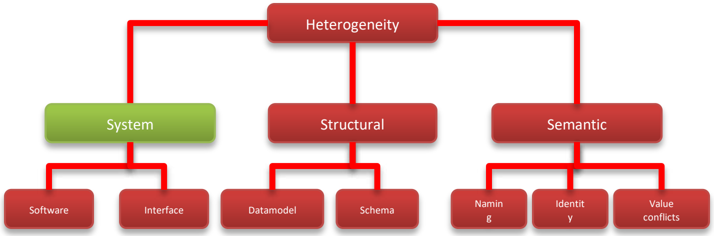
Naming

Identity

Value conflicts

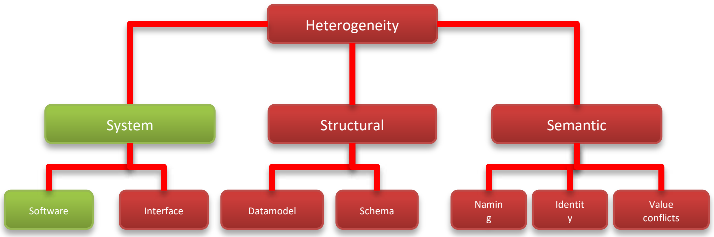
1.1 System Heterogeneity

- 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



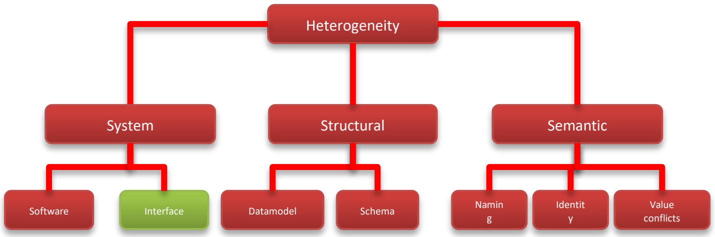
1.1 System Heterogeneity

- Hardware/Software
  - Different hardware capabilities of sources
    - **Mobile phone vs. server:** Cannot evaluate cross-product of two 1GB relations on a mobile phone
  - Different protocols, binary file formats, …
    - **Order information stored in text files:** line ending differs between Mac/Window/Linux, character encoding
  - Different access control mechanism
    - **FTP-access to files:** public, ssh authentication, ..



1.1 System Heterogeneity

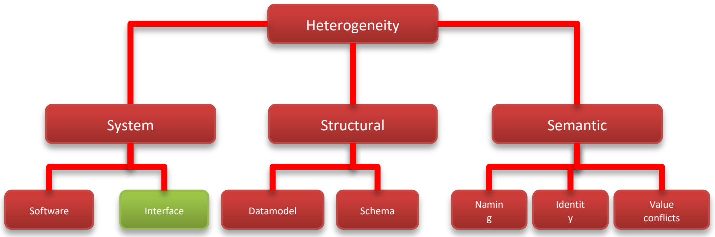
- Interface Heterogeneity
  - Different interfaces for accessing data from a source
    - HTML forms
    - Services (SOA)
    - Declarative language
    - Files
    - Proprietary network protocol
    - …



1.1 System Heterogeneity

ILLINOIS INSTITUTE OF TECHNOLOGY

- 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, …



Heterogeneity

System

Structural

Semantic

Software

Interface

Datamodel

Schema

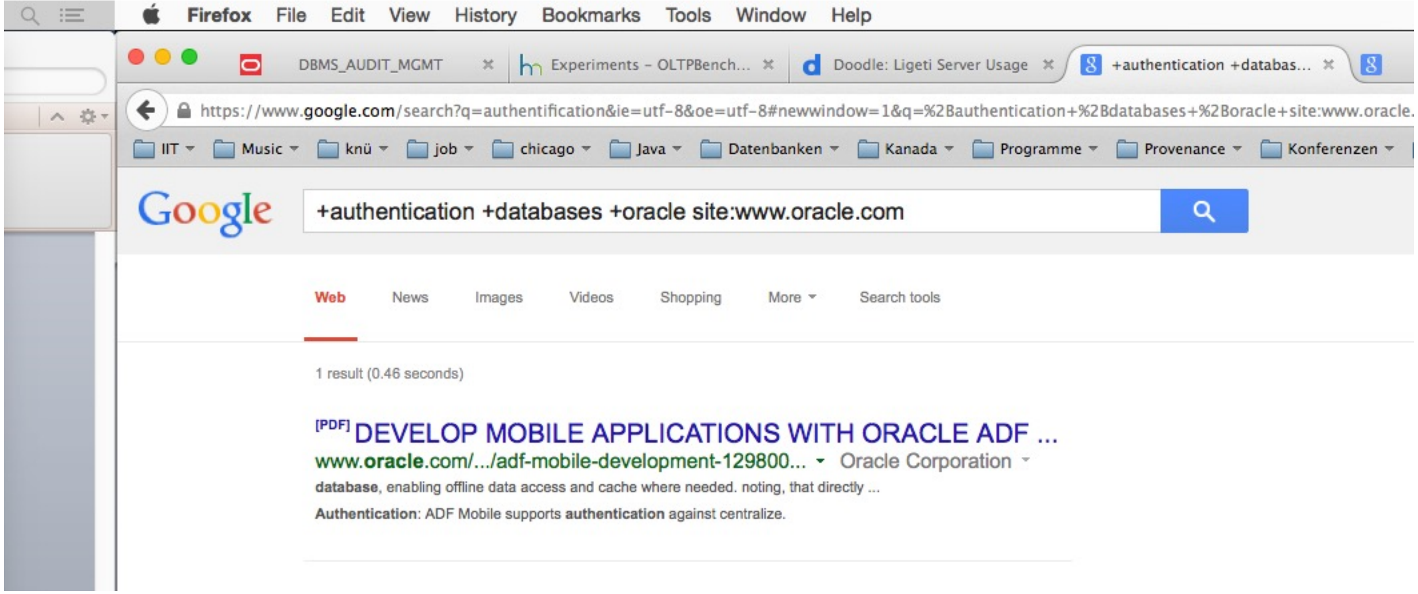
Namin g

Identit y

Value conflicts

1.1 System Heterogeneity

- Interface Heterogeneity – Examples
  - Google search (+/-, site:, intitle:, filetype:

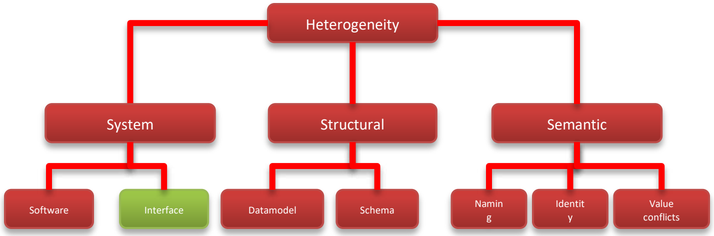


https://www.google.com/search?q=authentification&ie=utf-8&oe=utf-8#newwindow=1&q=%2Bauthentication+%2Bdatabases+%2Boracle+site:www.oracle.

+authentication +databases +oracle site:www.oracle.com

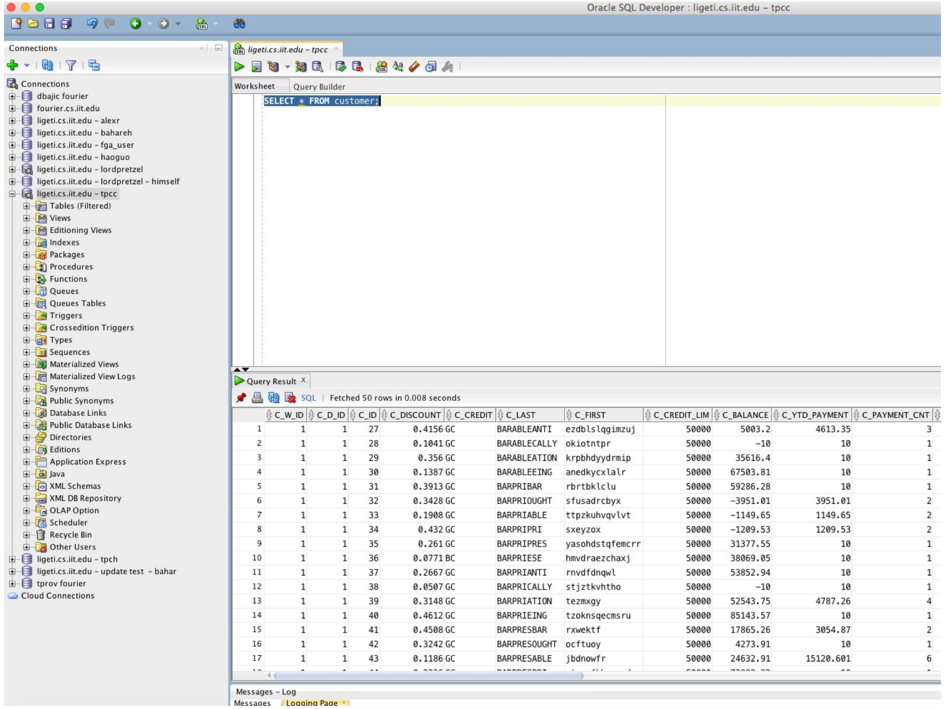
1 result (0.46 seconds)

[PDF] DEVELOP MOBILE APPLICATIONS WITH ORACLE ADF ...
www.oracle.com/.../adf-mobile-development-129800... Oracle Corporation
database, enabling offline data access and cache where needed. noting, that directly ...
Authentication: ADF Mobile supports **authentication** against centralize.



1.1 System Heterogeneity

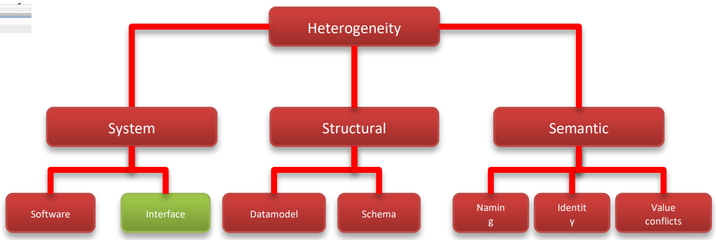
- Interface Heterogeneity – Examples
  - SQL



```
SELECT * FROM customer;
```

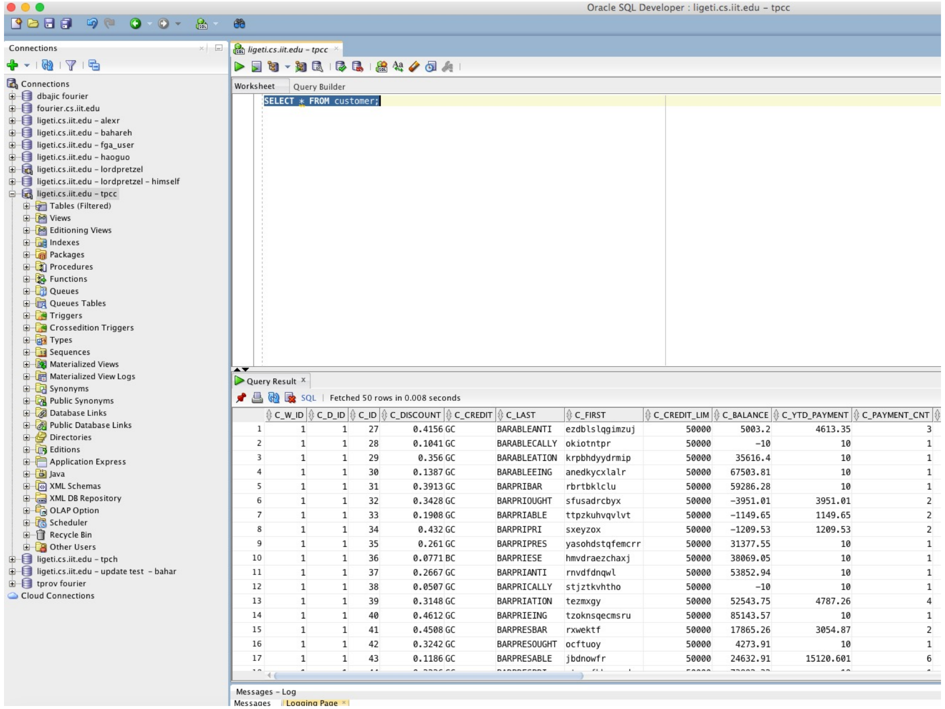
Fetched 50 rows in 0.008 seconds

	C_W_ID	C_D_ID	C_ID	C_DISCOUNT	C_CREDIT	C_LAST	C_FIRST	C_CREDIT_LIM	C_BALANCE	C_YTD_PAYMENT	C_PAYMENT_CNT
1	1	1	27	0.4156	GC	BARABLEANTI	ezdblslqgimzuj	50000	5003.2	4613.35	3
2	1	1	28	0.1041	GC	BARABLECALLY	okiotntpr	50000	-10	10	1
3	1	1	29	0.356	GC	BARABLEATION	krpbhdyydrmip	50000	35616.4	10	1
4	1	1	30	0.1387	GC	BARABLEEING	anedkycxlalr	50000	67503.81	10	1
5	1	1	31	0.3913	GC	BARPRIBAR	rbrtbklclu	50000	59286.28	10	1
6	1	1	32	0.3428	GC	BARPRIOUGHT	sfusadrcbyx	50000	-3951.01	3951.01	2
7	1	1	33	0.1908	GC	BARPRIABLE	ttpzkuhvqvlvt	50000	-1149.65	1149.65	2
8	1	1	34	0.432	GC	BARPRIPRI	sxeyzox	50000	-1209.53	1209.53	2
9	1	1	35	0.261	GC	BARPRIPRES	yasohdstqfemcrr	50000	31377.55	10	1
10	1	1	36	0.0771	BC	BARPRIESE	hmvdraezchaxj	50000	38069.05	10	1
11	1	1	37	0.2667	GC	BARPRIANTI	rnvdfdnqwl	50000	53852.94	10	1
12	1	1	38	0.0507	GC	BARPRICALLY	stjztkvhtho	50000	-10	10	1
13	1	1	39	0.3148	GC	BARPRIATION	tezmxgy	50000	52543.75	4787.26	4
14	1	1	40	0.4612	GC	BARPRIEING	tzoknsqecmsru	50000	85143.57	10	1
15	1	1	41	0.4508	GC	BARPRESBAR	rxwektf	50000	17865.26	3054.87	2
16	1	1	42	0.3242	GC	BARPRESOUGHT	ocftuoy	50000	4273.91	10	1
17	1	1	43	0.1186	GC	BARPRESABLE	jbdnowfr	50000	24632.91	15120.601	6



1.1 System Heterogeneity

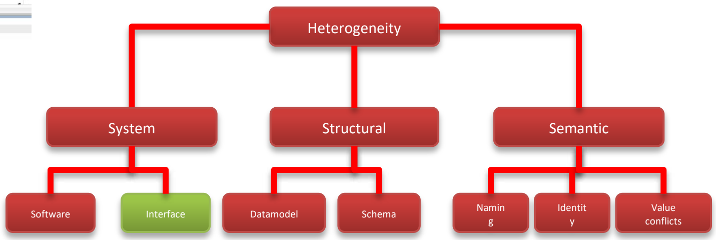
- Interface Heterogeneity – Examples
  - SQL



```
SELECT * FROM customer;
```

Fetched 50 rows in 0.008 seconds

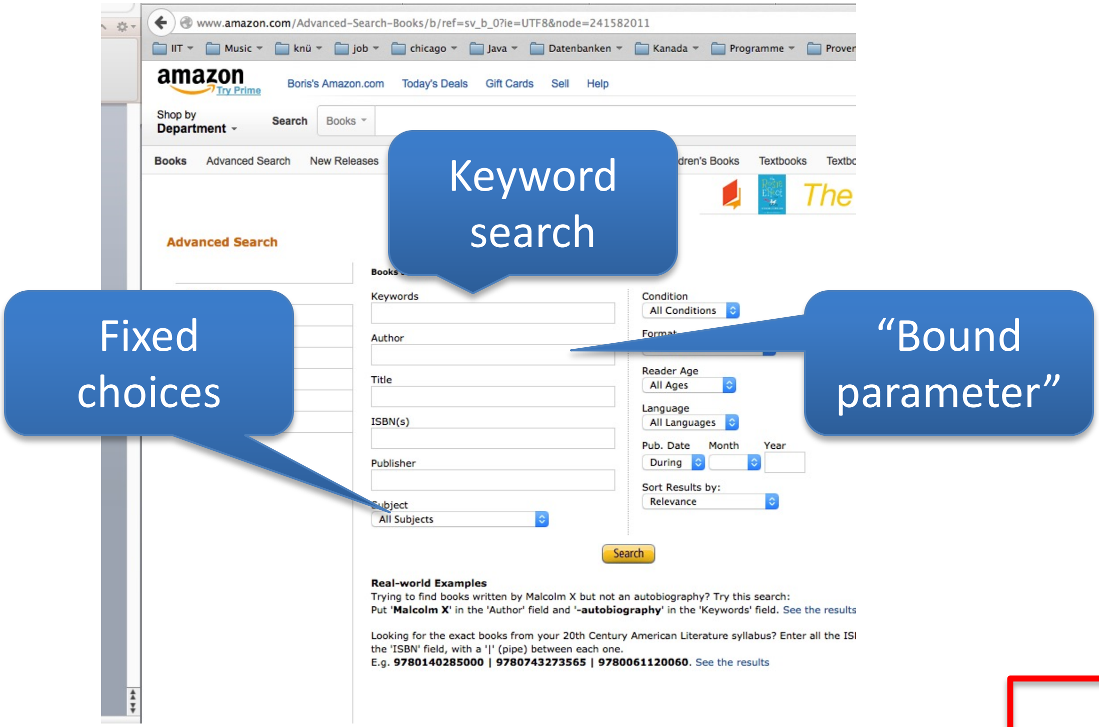
	C_W_ID	C_D_ID	C_ID	C_DISCOUNT	C_CREDIT	C_LAST	C_FIRST	C_CREDIT_LIM	C_BALANCE	C_YTD_PAYMENT	C_PAYMENT_CNT
1	1	1	27	0.4156	GC	BARABLEANTI	ezdblslqgimzuj	50000	5003.2	4613.35	3
2	1	1	28	0.1041	GC	BARABLECALLY	okiotntpr	50000	-10	10	1
3	1	1	29	0.356	GC	BARABLEATION	krpbhdyydrmip	50000	35616.4	10	1
4	1	1	30	0.1387	GC	BARABLEEING	anedkycxlalr	50000	67503.81	10	1
5	1	1	31	0.3913	GC	BARPRIBAR	rbrtbklclu	50000	59286.28	10	1
6	1	1	32	0.3428	GC	BARPRIOUGHT	sfusadrcbyx	50000	-3951.01	3951.01	2
7	1	1	33	0.1908	GC	BARPRIABLE	ttpzkuhvqvlvt	50000	-1149.65	1149.65	2
8	1	1	34	0.432	GC	BARPRIPRI	sxeyzox	50000	-1209.53	1209.53	2
9	1	1	35	0.261	GC	BARPRIPRES	yasohdstqfemcrr	50000	31377.55	10	1
10	1	1	36	0.0771	BC	BARPRIESE	hmvdraezchaxj	50000	38069.05	10	1
11	1	1	37	0.2667	GC	BARPRIANTI	rnvdfdnqwl	50000	53852.94	10	1
12	1	1	38	0.0507	GC	BARPRICALLY	stjztkvhtho	50000	-10	10	1
13	1	1	39	0.3148	GC	BARPRIATION	tezmxgy	50000	52543.75	4787.26	4
14	1	1	40	0.4612	GC	BARPRIEING	tzoknsqecmsru	50000	85143.57	10	1
15	1	1	41	0.4508	GC	BARPRESBAR	rxwektf	50000	17865.26	3054.87	2
16	1	1	42	0.3242	GC	BARPRESOUGHT	ocftuoy	50000	4273.91	10	1
17	1	1	43	0.1186	GC	BARPRESABLE	jbdnowfr	50000	24632.91	15120.601	6



1.1 System Heterogeneity

ILLINOIS INSTITUTE OF TECHNOLOGY

- Interface Heterogeneity – Examples
  - Web-form (with DB backend?)



www.amazon.com/Advanced-Search-Books/b/ref=sv_b_0?ie=UTF8&node=241582011

IIT Music knü job chicago Java Datenbanken Kanada Programme Prover

amazon Try Prime   Boris's Amazon.com   Today's Deals   Gift Cards   Sell   Help

Shop by Department   Search   Books

Books   Advanced Search   New Releases   dren's Books   Textbooks   Textbc

Advanced Search

Books

Keywords

Author

Title

ISBN(s)

Publisher

Subject: All Subjects

Condition: All Conditions

Format

Reader Age: All Ages

Language: All Languages

Pub. Date: During   Month   Year

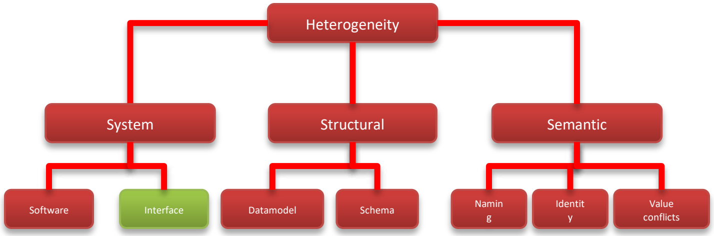
Sort Results by: Relevance

Search

Real-world Examples

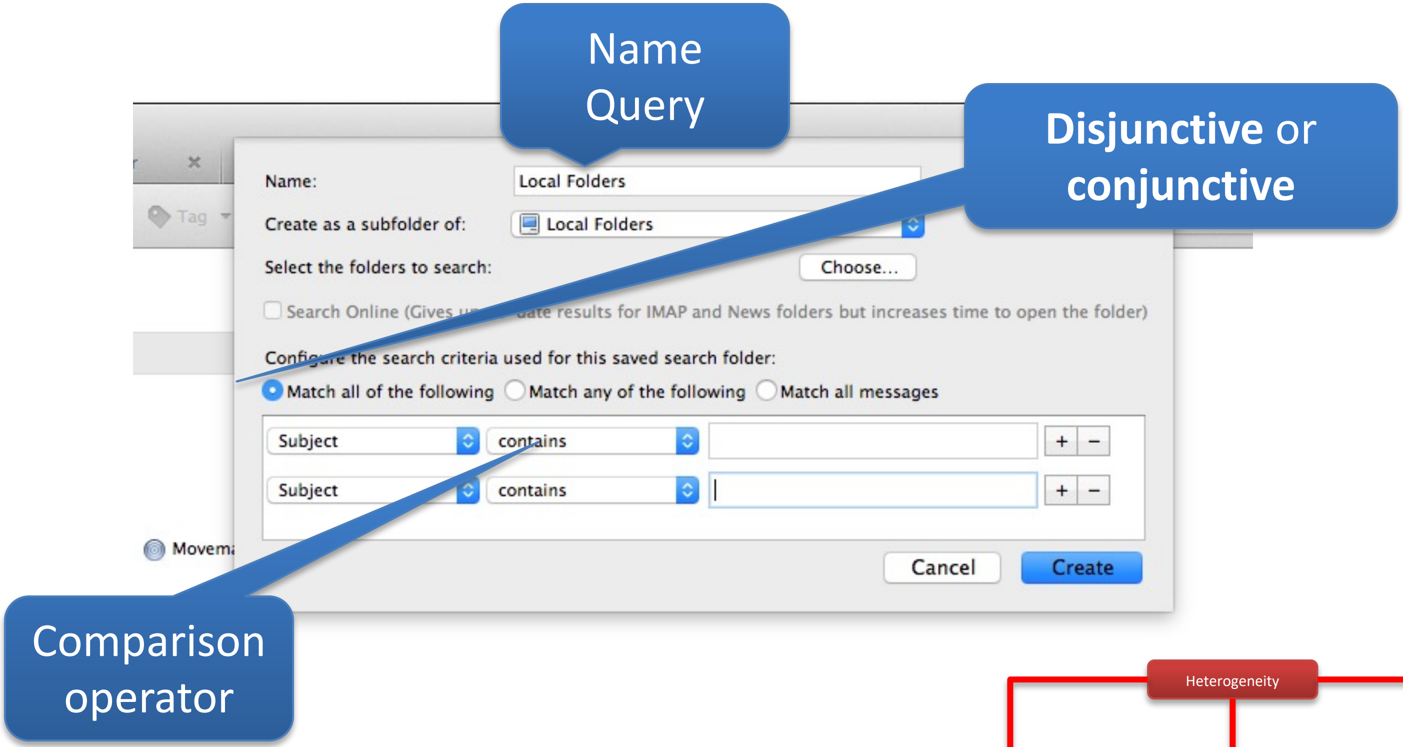
Trying to find books written by Malcolm X but not an autobiography? Try this search:
Put '**Malcolm X**' in the 'Author' field and '**-autobiography**' in the 'Keywords' field. See the results

Looking for the exact books from your 20th Century American Literature syllabus? Enter all the ISI
the 'ISBN' field, with a '|' (pipe) between each one.
E.g. **9780140285000 | 9780743273565 | 9780061120060**. See the results



1.1 System Heterogeneity

- Interface Heterogeneity – Examples
  - Email-client



Name: Local Folders

Create as a subfolder of: Local Folders

Select the folders to search: Choose...

☐ Search Online (Gives up-to-date results for IMAP and News folders but increases time to open the folder)

Configure the search criteria used for this saved search folder:

● Match all of the following ○ Match any of the following ○ Match all messages

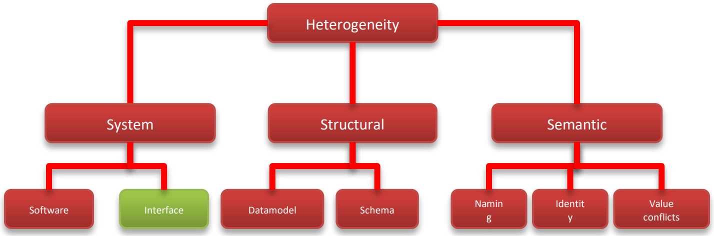
Subject contains

Subject contains

Name Query

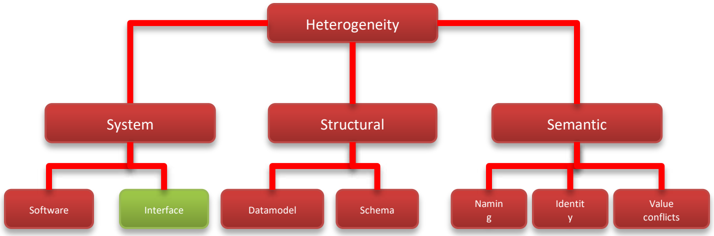
Disjunctive or conjunctive

Comparison operator



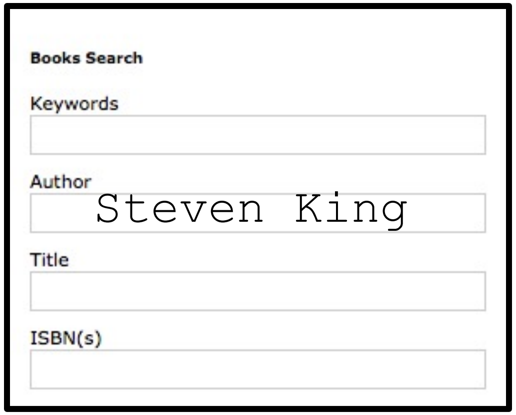
1.1 System Heterogeneity

- 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



1.1 System Heterogeneity

- 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?



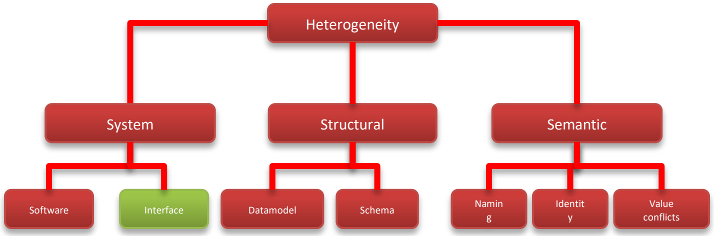
Books Search

Keywords

Author
Steven King

Title

ISBN(s)



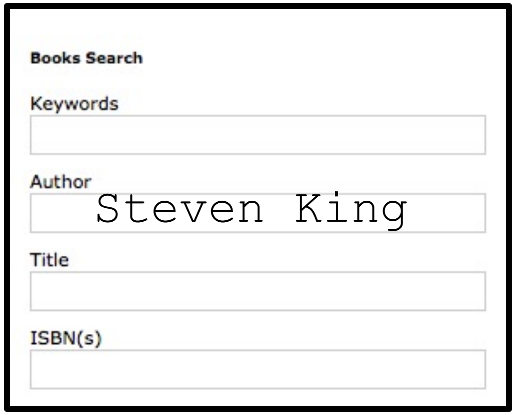
1.1 System Heterogeneity

- Integration system has to process part of the query

```
SELECT title
FROM books
WHERE author = ‘Steven King’
      AND year = 2012;
```

Stephen King, 2012, Misery

Stephen King, 2012, Misery
Stephen King, 2014, …
Stephen Kine, 1990, …



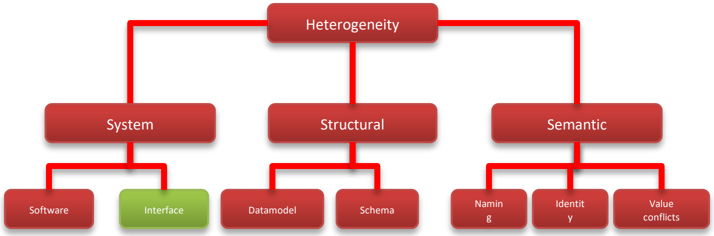
Books Search

Keywords

Author
Steven King

Title

ISBN(s)

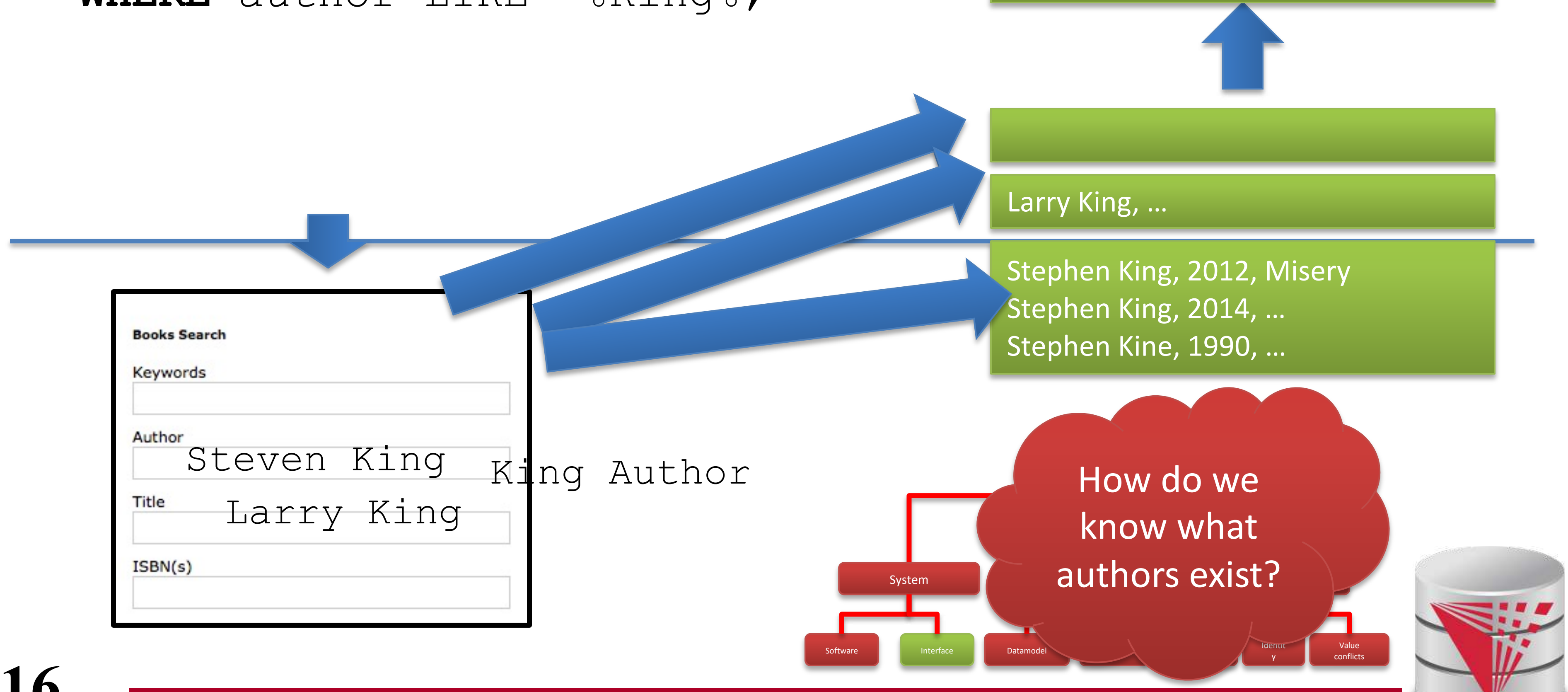


1.1 System Heterogeneity

- Query requires multiple requests

```
SELECT title
FROM books
WHERE author LIKE '%King%;
```

Larry Kin, …
Stephen King, 2012, Misery
Stephen King, 2012, Misery
…



Larry King, …

Stephen King, 2012, Misery
Stephen King, 2014, …
Stephen Kine, 1990, …

Books Search

Keywords

Author Steven King

Title Larry King

ISBN(s)

King Author

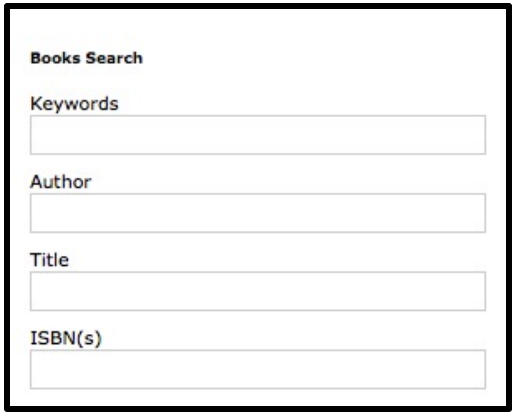
How do we know what authors exist?

1.1 System Heterogeneity

- Query cannot be answered

```
SELECT title
FROM books
WHERE genre = 'SciFi';
```

Web form is for history book only!



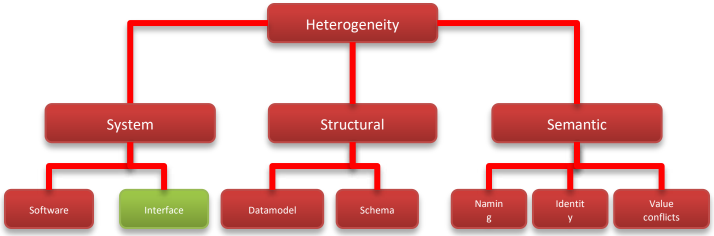
Books Search

Keywords

Author

Title

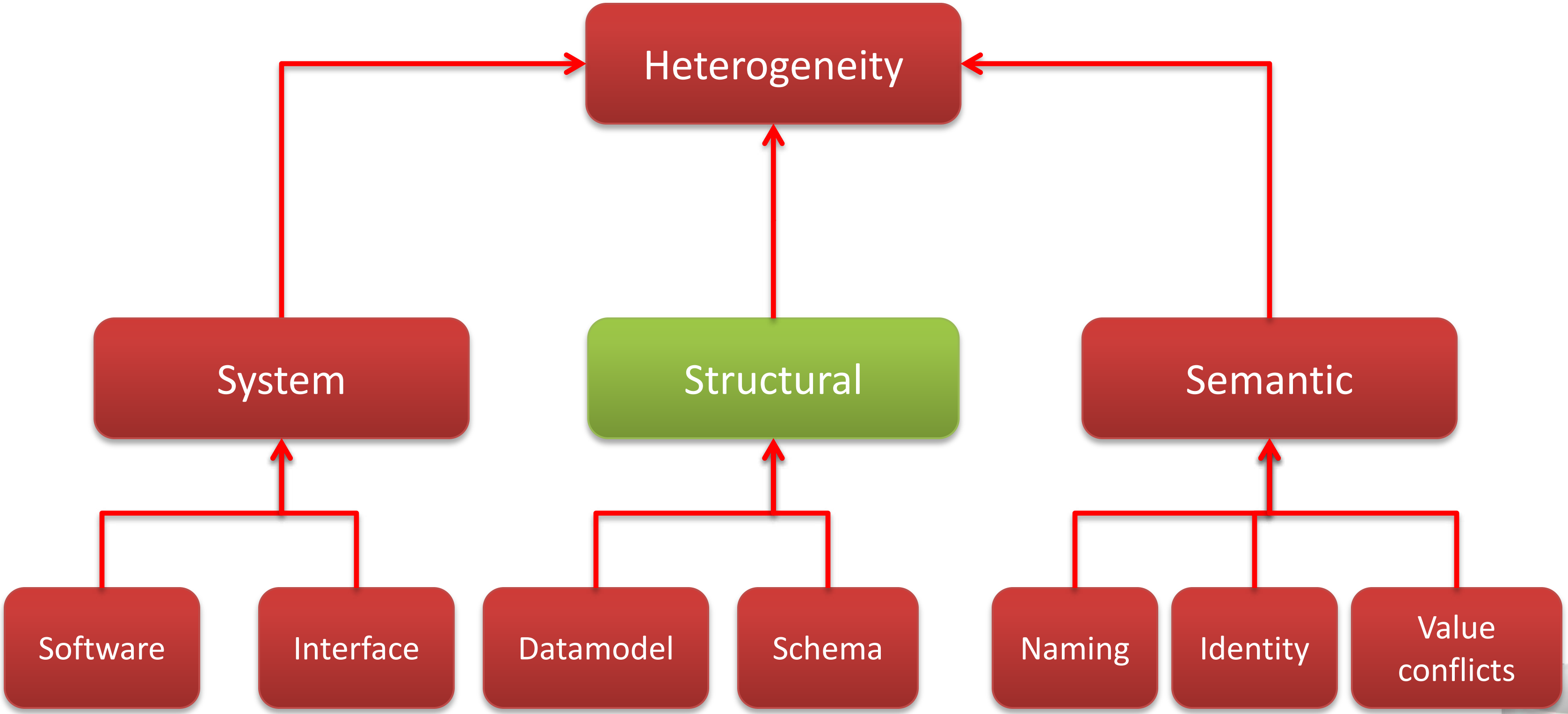
ISBN(s)



1.1 Heterogeneity +Autonomy

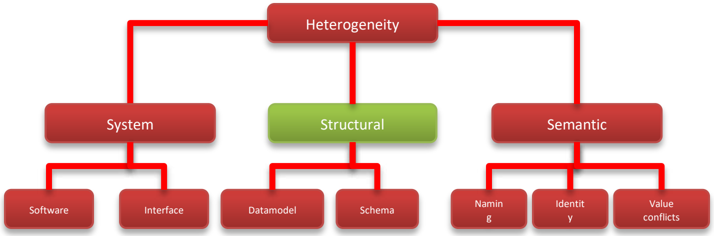
ILLINOIS INSTITUTE
OF TECHNOLOGY

- Taxonomy of Heterogeneity



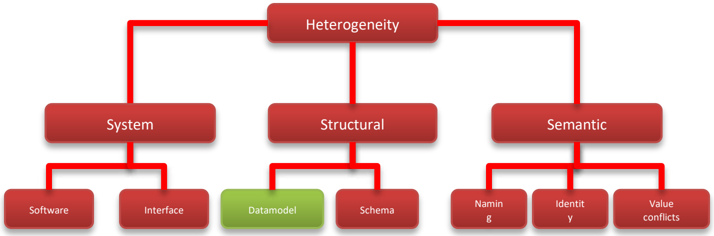
1.1 Structural Heterogeneity

- **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

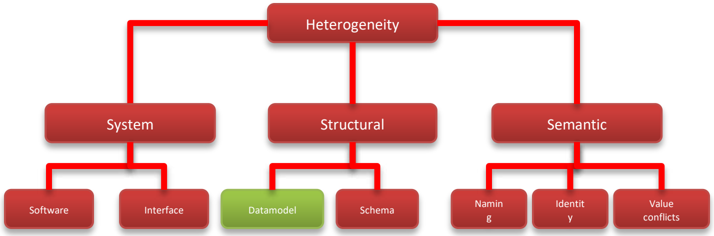
- **Data model**
  - Relational model
  - XML model
  - Object-oriented model
  - Ontological model
  - JSON
  - …



1.1 Structural Heterogeneity

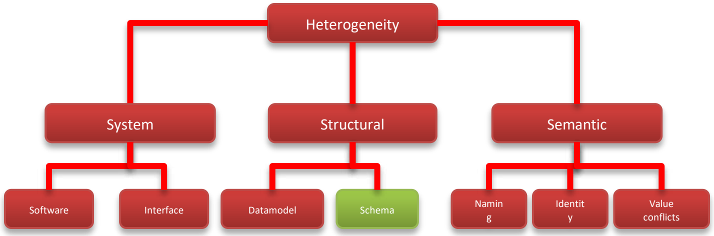
- Example: data model
  - Relational model
  - XML model
  - JSON
  - OO

- Person and their addresses



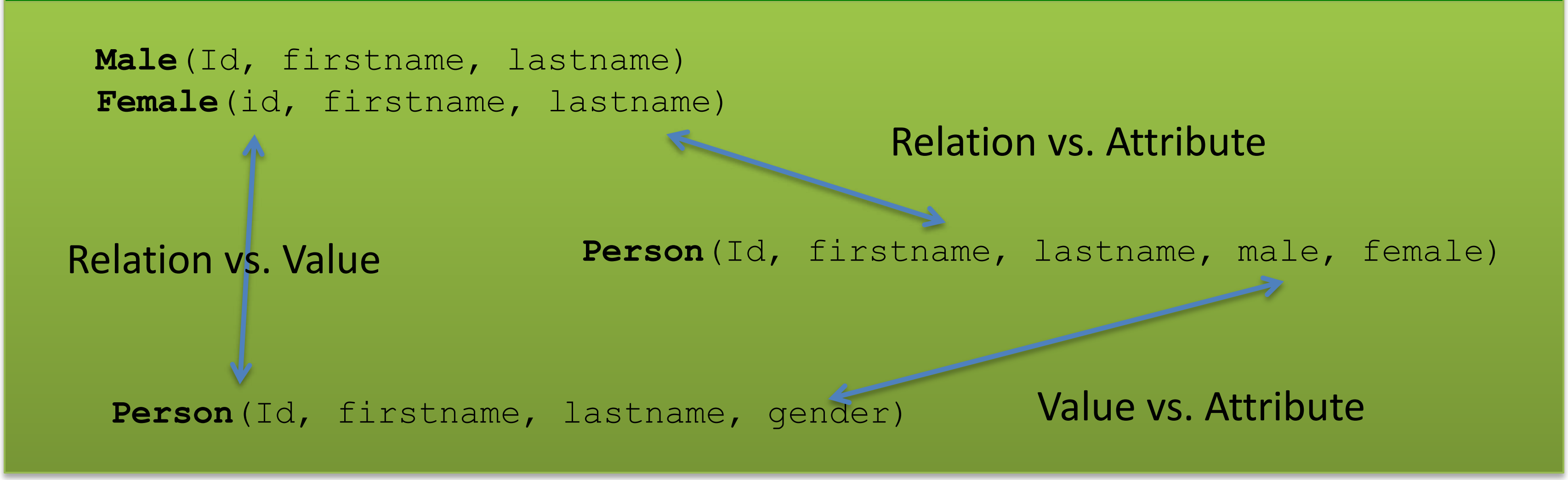
1.1 Structural Heterogeneity

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



1.1 Structural Heterogeneity

Example: Modeling choices



```
Male(Id, firstname, lastname)
Female(id, firstname, lastname)
```

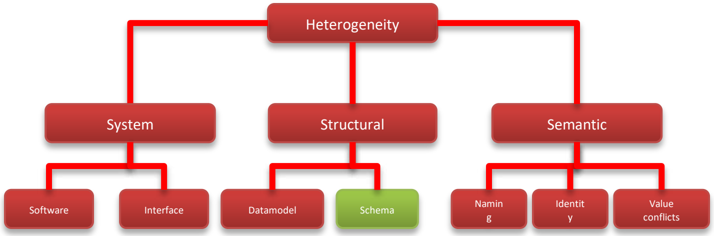
Relation vs. Attribute

```
Person(Id, firstname, lastname, male, female)
```

Relation vs. Value

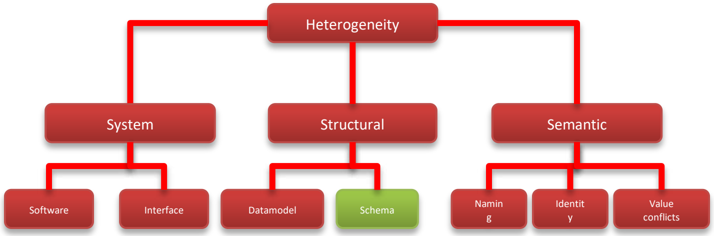
```
Person(Id, firstname, lastname, gender)
```

Value vs. Attribute



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



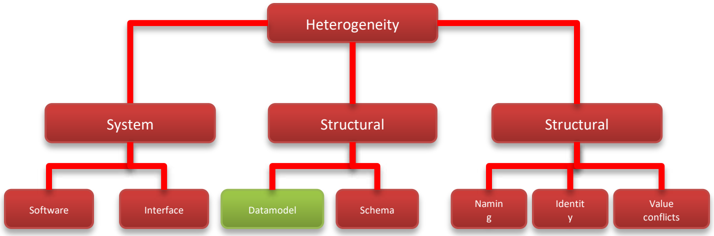
1.1 Structural Heterogeneity

Example: Conflicts between relations

Person(Id, firstname, lastname, male, female)

Person(Id, name, gender, birthday)

Manager(Id, name, gender, age)



Heterogeneity

System

Structural

Structural

Software

Interface

Datamodel

Schema

Namin g

Identit y

Value conflicts

1.1 Structural Heterogeneity

Mutliple attribtue vs one attribute

Example: Conflict between relations

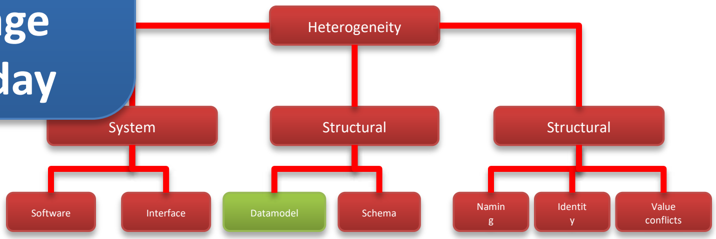
```
Person(Id, firstname, lastname, male, female)

Person(Id, name, gender, birthday)

Manager(Id, name, gender, age)
```

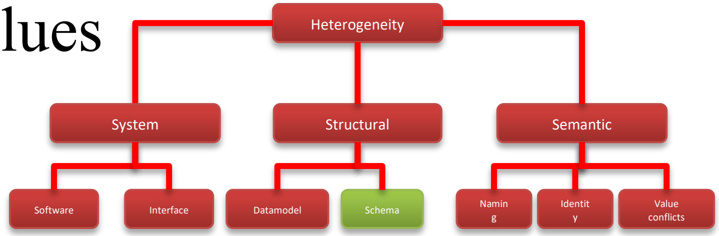
Missing derivable attribute: Role

Derivable attribute: Compute age from birthday



1.1 Structural Heterogeneity

- **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



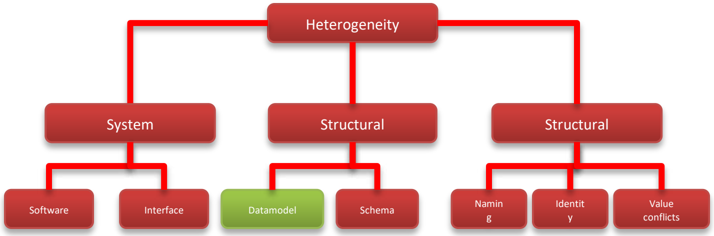
1.1 Structural Heterogeneity

ILLINOIS INSTITUTE OF TECHNOLOGY

Example: Conflicts between attributes and attributes

SSN	FirstName VARCHAR(40)	LastName	Age CHECK(Age > 18)
333-333-3333	Peter	Schmeter	30
333-333-9999	Hans	Glanz	NULL

SSN	FirstName VARCHAR(25)	SurName	Age
3333333333	Peter	Schmeter	30
3333339999	Hans	Glanz	-1



1.1 Structural Heterogeneity

Example: Conflicts between attributes and attributes

SSN	FirstName VARCHAR(40)	LastName	Age CHECK(Age > 18)
333-333-3333	Peter	Schmeter	30
333-333-9999	Hans	Glanz	NULL

SSN	FirstName VARCHAR(25)	SurName	Age
3333333333	Peter	Schmeter	30
3333339999	Hans	Glanz	-1

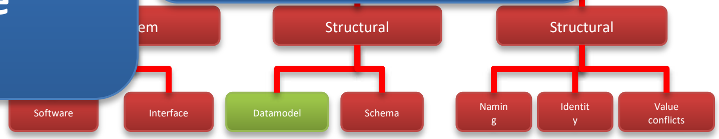
Conflicting constraint

Conflicting default value

Conflicting format

Conflicting datatype

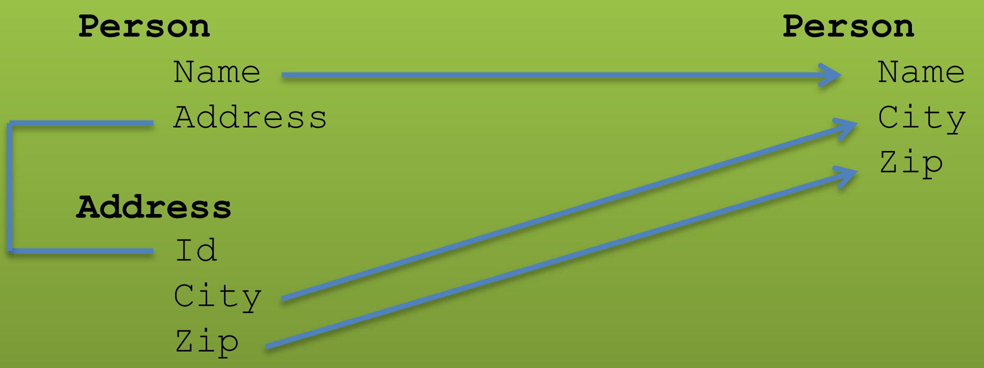
synonym



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

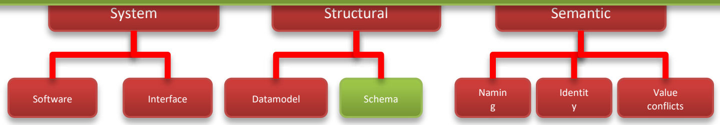
Example



Person
Name
Address

Address
Id
City
Zip

Person
Name
City
Zip

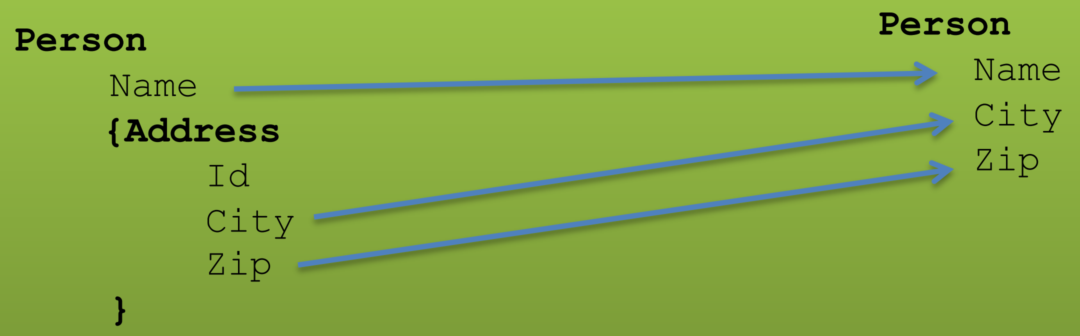


System
Structural
Semantic
Software
Interface
Datamodel
Schema
Namin
g
Identit
y
Value
conflicts

1.1 Structural Heterogeneity

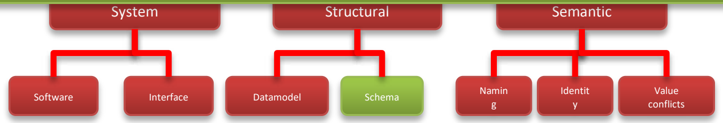
- **Nested vs. flat**
  - Association between entities can be represented using nesting or references (previous slides)

Example



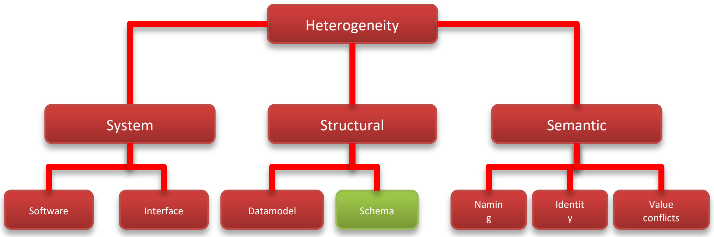
```
Person
        Name
        {Address
                Id
                City
                Zip
        }
```

```
Person
        Name
        City
        Zip
```



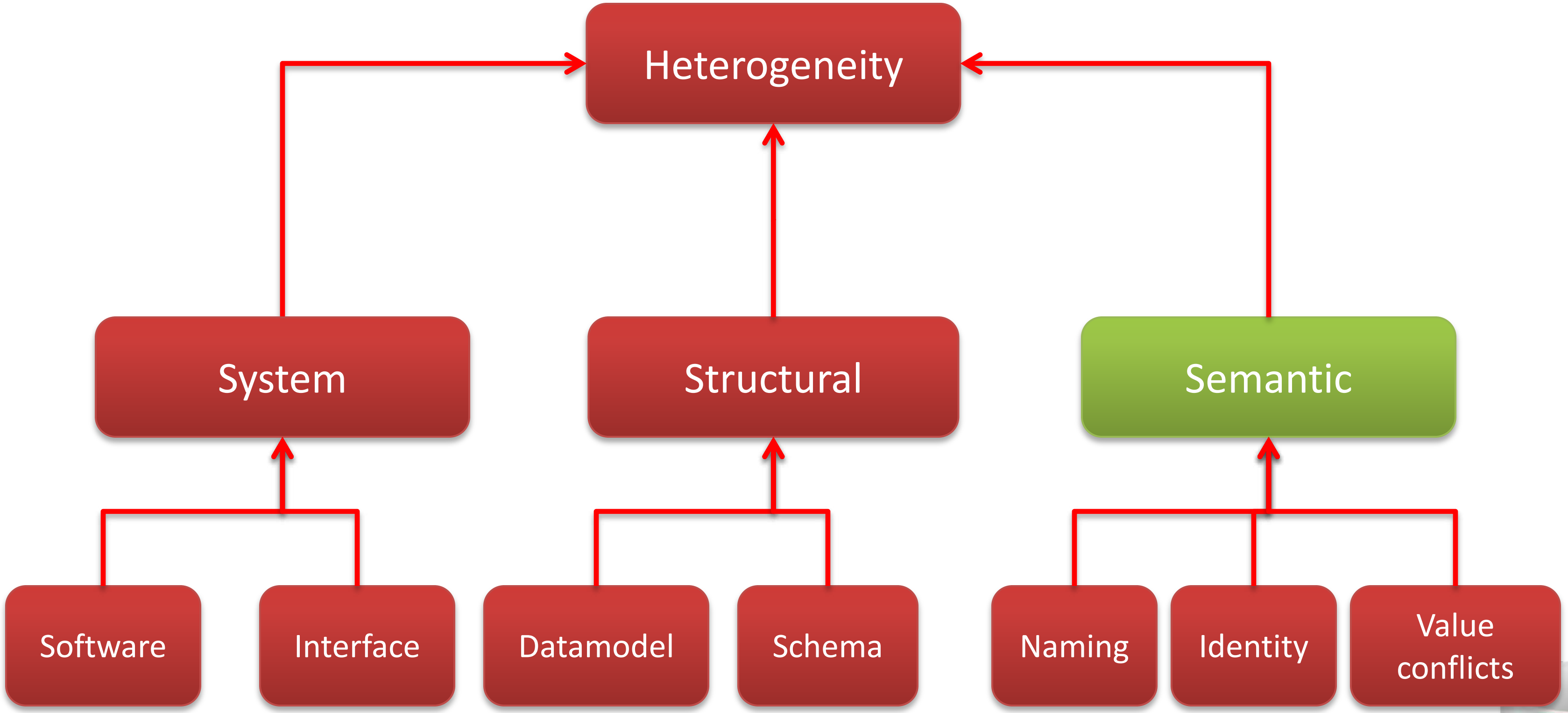
1.1 Structural Heterogeneity

- **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)



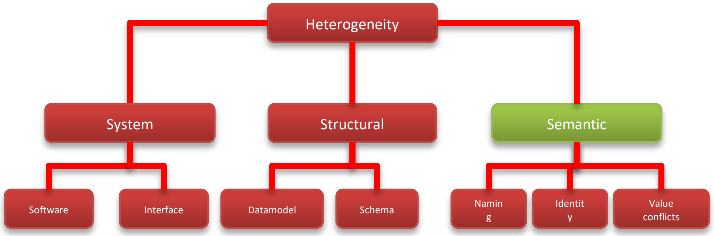
1.1 Heterogeneity +Autonomy

- Taxonomy of Heterogeneity



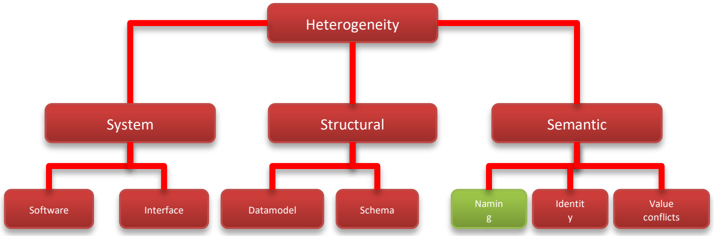
1.1 Semantic Heterogeneity

- **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



Heterogeneity

System

Structural

Semantic

Software

Interface

Datamodel

Schema

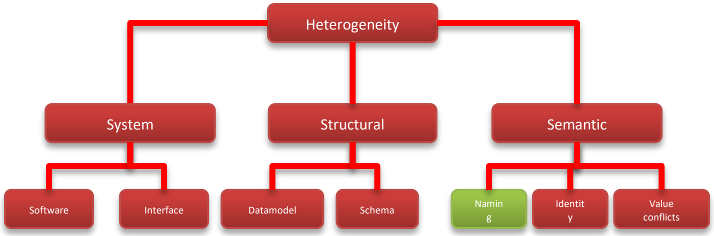
Namin
g

Identit
y

Value
conflicts

1.1 Semantic Heterogeneity

- **Ontological concepts**
  - Relationships between concepts
    - **A = B** - Equivalence
    - **A ⊆B** - Inclusion
    - **A ∩ B** - Overlap
    - **A ≠ B** - Disjunction



Heterogeneity

System

Structural

Semantic

Software

Interface

Datamodel

Schema

Naming

Identity

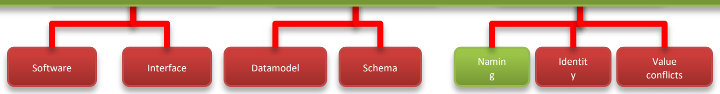
Value conflicts

1.1 Semantic Heterogeneity

- **Ontological concepts**
  - Relationships between concepts
    - **A = B** - Equivalence
    - **A ⊆B** - Inclusion
    - **A ∩ B** - Overlap
    - **A ≠ B** - Disjunction

Example

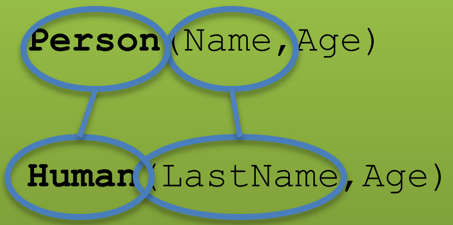
Equivalence:	Human vs Homo sapiens
Inclusion:	Bird vs Animal
Overlap:	Animal vs aquatic lifeform
Disjunction:	Fish vs Mamal



1.1 Semantic Heterogeneity

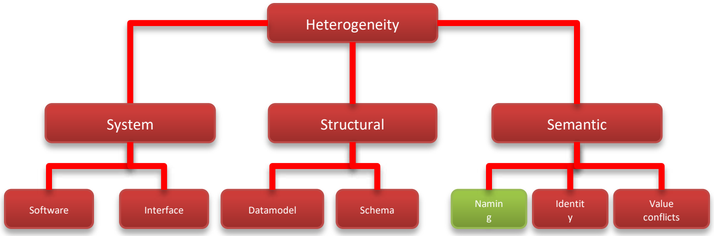
- **Naming concepts (synonyms)**
  - **Different** words with **same meaning**

Example



```
Person(Name, Age)

Human(LastName, Age)
```



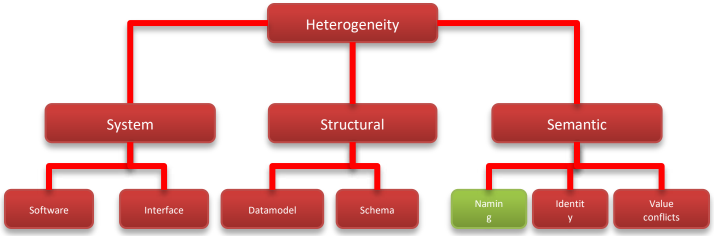
1.1 Semantic Heterogeneity

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

Example

Person (Title, Name)

Movie (Title, Year)



Heterogeneity

System

Structural

Semantic

Software

Interface

Datamodel

Schema

Namin g

Identit y

Value conflicts

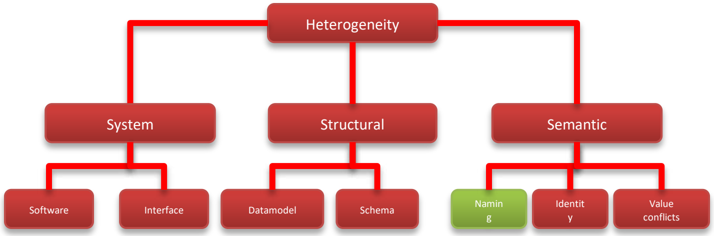
1.1 Semantic Heterogeneity

- **Naming concepts (units)**

Example

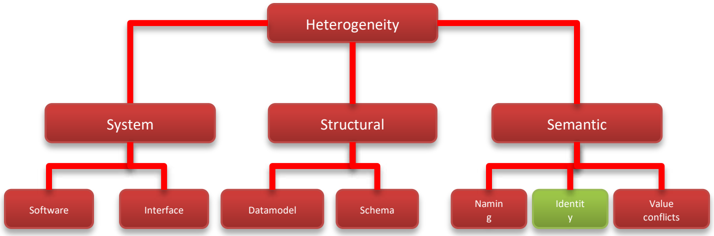
`Person(Title,Name,Salary)` — $

`Person(Title,Name,Salary)` — CAD



1.1 Semantic Heterogeneity

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



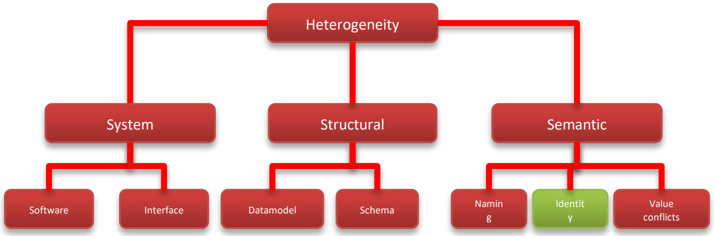
1.1 Semantic Heterogeneity

- Identity Conflicts

Example

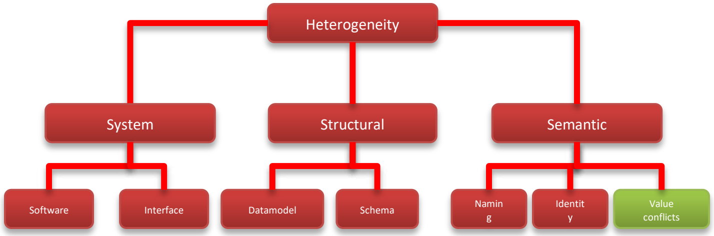
```
(IBM,300000000,USA)

(International Business Machines Corporation,50000)
```



1.1 Semantic Heterogeneity

- **Value Conflicts**
  - Objects representing the same entities have conflicting values for semantically equivalent attributes
    - We have to identified that these objects are represent the same entitity 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 Boolean Logic

- **Boolean Logic (syntax)**
  - **Atomic formulas:**
    - Boolean constants (**true, false)**
    - Boolean Variables (can take Boolean constants as values)
  - **Formulas:**
    - Any atomic formula is also a formula
    - If ϕ, ψ are formulas then the following are also valid formulas:

$\neg \phi$ $\qquad\qquad$ $\phi \wedge \psi$

$\phi \vee \psi$ $\qquad\qquad$ $\phi \rightarrow \psi$

1.4 Boolean Logic

- **Boolean Logic (semantics)**
  - **Valuation:**
    - Assign truth values to the variables of a formula
    - Under a valuation a formula evaluates to a Boolean value (true or false)
    - If there exists a valuation that makes the formula ψ true then the formula ψ is called **satisfiable**
  - **Semantics:**
    - Expected semantics of Boolean operators:

$$\top \wedge \bot = \bot$$

$$\top \wedge \top = \top$$

$$\bot \vee \top = \top$$

$$\ldots$$

1.4 Boolean Logic

Example

Formula:

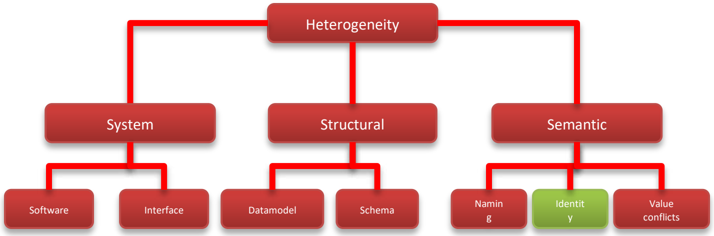
$$(x \vee y) \wedge \neg z$$

A possible valuation:

$$\nu : x = \top, y = \bot, z = \top$$

Evaluating the formula:

$$(\top \vee \bot) \wedge \neg \top = \top \wedge \bot = \bot$$



1.4 First-order logic (FO)

- Concepts
  - **Domain of discourse** $\mathbb{D}$
    - These are the values that we can bind variables to
    - Values from the domain can also be used as constants in formulas
  - **A set of predicate symbols (each with an arity)**

$$R_1, \ldots, R_n$$

    - These represent relations (in the mathematical sense)
  - **An infinite set of variables** $\mathcal{X}$

1.4 FO Syntax

- **Terms**
  - **Variables:** any variable from $\mathcal{X}$ is a term
  - **Constants:** any constant from $\mathbb{D}$ is a term
- **Atomic formulas:**
  - For any n-ary predicate *R* and terms $t_1, \ldots, t_n$ $R(t_1, \ldots, t_n)$ is an atomic formula
- **Formulas:**
  - If ϕ, ψ are formulas then the following are also valid formulas:

$\psi \wedge \phi$ $\psi \vee \phi$ $\neg \phi$

$\psi \rightarrow \phi$ $\exists x : \psi$ $\forall x : \psi$

1.4 Free / Bound Variables

ILLINOIS INSTITUTE OF TECHNOLOGY

– Free variables of a formula

- All variables not bound by quantifiers

$$free(\neg\psi) = free(\psi)$$
$$free(\psi \land \phi) = free(\psi) \cup free(\phi)$$
$$free(\psi \lor \phi) = free(\psi) \cup free(\phi)$$
$$free(\forall x : \psi) = free(\psi) - \{x\}$$
$$free(\exists x : \psi) = free(\psi) - \{x\}$$
$$free(R(t_1, \ldots, t_n)) = free(t_1) \cup \ldots \cup free(t_n)$$
$$free(x) = \{x\}$$
$$free(c) = \emptyset$$

1.4 FO Semantics

- **Model $\mathcal{M}$**
  - an interpretation of the predicates, i.e., we assign each predicate to a concrete relation
  - We select a domain of discourse $\mathbb{D}$
- **Valuations μ for a formula ψ**
  - Assigns free variables of ψ to values from $\mathbb{D}$
- **Substitutions**
  - Replace all free occurrences of variable *x* with *c*

$$\psi[x \leftarrow c]$$

1.4 FO Semantics

- **Given a model $\mathcal{M}$ and valuation μ**
  - The “result” of a formula $[\![\psi]\!]_{\mathcal{M},\mu}$

$$[\![c]\!]_{\mathcal{M},\mu} = c$$

$$[\![x]\!]_{\mathcal{M},\mu} = \mu(x)$$

$$[\![R(t_1,\ldots,t_n)]\!]_{\mathcal{M},\mu} = \begin{cases} \top & \text{if } ([\![t_1]\!]_{\mathcal{M},\mu},\ldots,[\![t_n]\!]_{\mathcal{M},\mu}) \in R \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\psi \wedge \phi]\!]_{\mathcal{M},\mu} = [\![\psi]\!]_{\mathcal{M},\mu} \wedge [\![\phi]\!]_{\mathcal{M},\mu}$$

$$[\![\psi \vee \phi]\!]_{\mathcal{M},\mu} = [\![\psi]\!]_{\mathcal{M},\mu} \vee [\![\phi]\!]_{\mathcal{M},\mu}$$

$$[\![\neg\, \psi]\!]_{\mathcal{M},\mu} = \neg\, [\![\psi]\!]_{\mathcal{M},\mu}$$

$$[\![\exists x : \psi]\!]_{\mathcal{M},\mu} = \bigvee_{c \in \mathbb{D}} : [\![\psi[x \leftarrow c]]\!]_{\mathcal{M},\mu}$$

$$[\![\forall x : \psi]\!]_{\mathcal{M},\mu} = \bigwedge_{c \in \mathbb{D}} : [\![\psi[x \leftarrow c]]\!]_{\mathcal{M},\mu}$$

1.4 FO semantics

ILLINOIS INSTITUTE OF TECHNOLOGY

Example

Formula: $$\psi = \forall y : R(x,y)$$

Model: $$\mathcal{M} = \{R = \{(1,1),(1,2),(1,3)\}$$

$$\mathbb{D} = \{1,2,3\}\}$$

Valuation: $$\mu(x) = 1$$

Result:

$$\begin{aligned}
&\llbracket \forall y : R(x,y) \rrbracket_{\mathcal{M},\mu} \\
=&\llbracket R(x,1) \rrbracket_{\mathcal{M},\mu} \wedge \llbracket R(x,2) \rrbracket_{\mathcal{M},\mu} \wedge \llbracket R(x,3) \rrbracket_{\mathcal{M},\mu} \\
=&\llbracket (x,1) \rrbracket_{\mathcal{M},\mu} \in R \wedge \llbracket (x,2) \rrbracket_{\mathcal{M},\mu} \in R \wedge \llbracket (x,3) \rrbracket_{\mathcal{M},\mu} \in R \\
=&(\mu(x),1) \in R \wedge (\mu(x),2) \in R \wedge (\mu(x),3) \in R \\
=&(1,1) \in R \wedge (1,2) \in R \wedge (1,3) \in R \\
=&\top \wedge \top \wedge \top \\
=&\top
\end{aligned}$$

1.4 FO Problems

- **Model checking**
  - Given a model $\mathcal{M}$ and formula ψ without free variables
  - Is $[\![\psi]\!]_{\mathcal{M},\mu}$ true?
- **Satisfiability**
  - Given a formula ψ does there exist a model $\mathcal{M}$ and valuation μ such that $[\![\psi]\!]_{\mathcal{M},\mu}$ is true?

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**

1.4 Integrity constraints

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

1.4 Integrity 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

ILLINOIS INSTITUTE OF TECHNOLOGY

Example

Primary Key R($\underline{\text{A}}$,B):

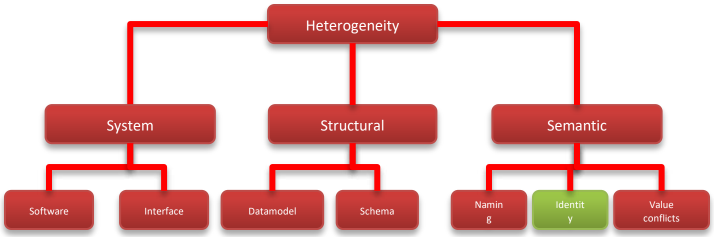
$$\forall x, y, z : R(x, y) \land R(x, z) \rightarrow y = z$$

Functional Dependency R(A,B) with A->B:

$$\forall x, y, z, a : R(x, y) \land R(z, a) \land x = z \rightarrow y = a$$

Foreign Key R($\underline{\text{A}}$,B), S(C,D) where D is FK to R:

$$\forall x, y : S(x, y) \rightarrow \exists z : R(y, z)$$



1.4 Integrity constraints

- Types of constraints we will use a lot
  - Tuple-generating dependencies (**tgds**)
    - Implication with conjunction of relational atoms
    - Foreign keys and schema mappings (later)

$$\forall \vec{x} : \phi(\vec{x}) \rightarrow \exists \vec{y} : \psi(\vec{x}, \vec{y})$$

  - Equality-generating dependencies (**egds**)
    - Generalizes keys, FDs

$$\forall \vec{x} : \phi(\vec{x}) \rightarrow \wedge_{k=1}^{n} x_{i_k} = x_{j_k}$$

1.4 Datalog

- What is Datalog?
  - Prolog for databases (syntax very similar)
  - A logic-based query language
- Queries (Program) expressed as set of rules

$$Q(\vec{x}) :- R_1(\vec{x_1}), \ldots, R_n(\vec{x_n}).$$

- One Q is specified as the answer relation (the relation returned by the query)

1.4 Datalog - Intuition

- **A Datalog rule**

$$Q(\vec{x}) :- R_1(\vec{x_1}), \ldots, R_n(\vec{x_n}).$$

- **Procedural Interpretation:** For all bindings of variables 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** adom(I) of an instance I is the set of all constants that occur in I

$$Q(\vec{x}) :- R_1(\vec{x_1}), \ldots, R_n(\vec{x_n}).$$

1.4 Datalog - Terminology

ILLINOIS INSTITUTE OF TECHNOLOGY

Example

```
Q(N) :- Person(N,A).
```

N, A are **variables**

Q(N), Person(N,A) are **atoms**

Person and Q are **predicates**

Name	Age
peter	34
bob	45

Activate domain

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

1.4 Datalog - Terminology

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

$$Q(\vec{x}) :- R_1(\vec{x_1}), \ldots, R_n(\vec{x_n}).$$

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

$$Q(\vec{x}) :- R_1(\vec{x_1}), \ldots, R_n(\vec{x_n}).$$

1.4 Datalog - Semantics

Example

```
Q(N):- Person(N,A).
```

```
N=peter,A=peter: Q(peter):- Person(peter,peter).
N=peter,A=bob: Q(peter):- Person(peter,bob).
N=peter,A=34: Q(peter):- Person(peter,34).
N=bob,A=peter: Q(bob):- Person(peter,peter).
N=bob,A=bob: Q(bob):- Person(peter,bob).
N=bob,A=34: Q(bob):- Person(bob,34).
N=34,A=peter: Q(34):- Person(34,peter).
N=34,A=bob: Q(34):- Person(34,bob).
N=34,A=34: Q(34):- Person(34,34).
```

N
peter
bob

Active domain

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

Name	Age
peter	34
bob	34

1.4 Datalog

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

1.4 Datalog

- Different flavors of datalog
  - **Recursion**
    - Rules may have recursion:
      - E.g., head predicate in the body
    - Fixpoint semantics based on immediate consequence operator
  - **Negation (first-order queries)**
    - Negated relational atoms allowed
    - Require that every variable used in a negated atom also occurs in at least on positive atom (**safety**)
  - **Combined Negation + recursion**
    - Stronger requirements (e.g., stratification)

1.4 Datalog – Semantics (Negation)

- 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
- A negated atom not R(X) is true if R(X) is not part of the instance

$$Q(\vec{x}) : -R_1(\vec{x_1}), \ldots, R_n(\vec{x_n}).$$

1.4 Datalog - Semantics

Example

```
Q(N) :- Person(N,A), not Lives(N).
```

```
N=peter,A=peter: Q(peter):- Person(peter,peter),
                              not Lives(peter).
N=peter,A=bob: Q(peter):- Person(peter,bob),
                              not Lives(peter).

…
N=bob,A=34: Q(bob):- Person(bob,34),
                              not Lives(bob).
…
```

Result

N
bob

Lives

Name
peter

Person

Name	Age
peter	34
bob	34

Active domain

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

1.4 Datalog

Example

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

Q_{2hop}(x,z): hop(x,y),hop(y,z).

Q_{reach}(x,y): hop(x,y).
Q_{reach}(x,z): Q_{reach}(x,y),Q_{reach}(y,z).

Q_{node}(x): hop(x,y).
Q_{node}(x): hop(y,x).

1.4 Datalog

ILLINOIS INSTITUTE
OF TECHNOLOGY

Example

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

$Q_{node}(x)$: hop(x,y).
$Q_{node}(x)$: hop(y,x).

$Q_{notReach}(x,y)$: $Q_{node}(x)$, $Q_{node}(y)$,
not $Q_{reach}(x,y)$.

1.4 Datalog versus FO

- A Datalog rule is a FO implication:

$$Q(X,Y) :- R(X,Z), R(Z,Y).$$

Means

$$\forall x, y : \exists z : R(x,z) \wedge R(z,y) \rightarrow Q(x,y)$$

- Databases can be expressed as rules!

$$R = \{(Peter, Bob), (Bob, Alice)\}$$

$$R(Peter, Bob) :-$$

$$R(Bob, Alice) :-$$

1.4 Model-theoretic semantics

- The result of a Datalog program P is the smallest model $\mathcal{M}$ for the program if interpreted as a logical formula
  - Only facts that are justified by the program are included in the query result!

1.4 Free Datalog Systems

- Datalog Education System (DES)
  - http://des.sourceforge.net/
- DLV
  - http://www.dlvsystem.com/dlv/

1.4 Containment and Equivalence

Definition: Query Equivalence

Query Q is equivalent to Q’ iff for every database instance I both queries return the same result

$$Q \equiv Q' \Leftrightarrow \forall I : Q(I) = Q'(I)$$

Definition: Query Containment

Query Q is contained in query Q’ iff for every database instance I the result of Q is contained in the result of Q’

$$Q \sqsubseteq Q' \Leftrightarrow \forall I : Q(I) \subseteq Q'(I)$$

1.4 Equivalence

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

1.4 Containment and Equiv.

Example

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

$Q_2(x,y): R(x,y).$

$Q_3(x,x): R(x,x).$

$Q_4(x,y): R(x,y).$

$Q_5(x,x): R(x,y), R(x,x).$

$Q_6(x,z): R(x,y), R(y,z).$

1.4 Containment and Equiv.

Example

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

Q_{2hop}(x,z): hop(x,y),hop(x,z).

Q_{up2Hop}(x,z): hop(x,y),hop(x,z).
Q_{up2Hop}(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.

Set semantics	Relational Algebra	Conjunctive Queries (CQ)	Union of Conjunctive Queries (UCQ)	Monotone Queries/ CQ≠
Query Evaluation (Combined Complexity)	PSPACE-complete	NP-complete	NP-complete	NP-complete
Query Evaluation (Data Complexity)	LOGSPACE (that means in P)	LOGSPACE (that means in P)	LOGSPACE (that means in P)	LOGSPACE (that means in P)
Query Equivalence	Undecidable	NP-complete	NP-complete	Π_2^p-complete
Query Containment	Undecidable	NP-complete	NP-complete	Π_2^p-complete

1.4 Complexity of Eq. and Cont.

Bag semantics	Relational Algebra	Conjunctive Queries (CQ)	Union of Conjunctive Queries (UCQ)
Query Equivalence	Undecidable	Equivalent to graph isomorphism	Undecidable
Query Containment	Undecidable	**Open Problem**	Undecidable

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

```
Hop relation: Hop(A,B)

Q :- hop(x,y)

SELECT EXISTS (SELECT * FROM hop)

Note: in Oracle and DB2 we need a
from clause
```

1.4 Boolean Conjunctive Queries

Example

```
SELECT
     CASE WHEN EXISTS (SELECT *
                       FROM hop)
     THEN 1 ELSE 0
     END AS x
FROM dual;


Notes:
- Oracle and DB2 FROM not optional
- Oracle has no boolean datatype
```

1.4 Boolean Conjunctive Queries

- BCQ in SQL

Example

```
Q :- hop(x,y), hop(y,z)

SELECT EXISTS
    (SELECT *
    FROM hop l, hop r
    WHERE l.B = r.A)
```

1.4 Containment Mappings

- How to check for containment of CQs (set)

Definition: Variable Mapping

A variable mapping ψ 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 ψ such that:

$$\Psi(head(Q)) = head(Q')$$

$$\forall R(\vec{x_i}) \in body(Q) : \Psi(R(\vec{x_i})) \in body(Q')$$

1.4 Containment Mappings

ILLINOIS INSTITUTE OF TECHNOLOGY

Theorem: Containment Mappings and Query Containment

Query Q is contained in query Q’ iff there exists a containment mapping ψ from Q’ to Q

$$Q \sqsubseteq Q' \Leftrightarrow \exists \Psi : \Psi \text{ is a containment mapping } Q' \rightarrow Q$$

Example

Q_1(u,z): R(u,z).

Q_2(x,y): R(x,y).

Can we find a containment mapping?

1.4 Containment Mappings

Theorem: Containment Mapping and Query Containment

Query Q is contained in query Q’ iff there exists a containment mapping ψ from Q’ to Q

Example

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

Q_1 -> Q_2 :$\Psi(u)=x$, $\Psi(z)=y$
Q_2 -> Q_1 :$\Psi(x)=u$, $\Psi(y)=z$

1.4 Containment Mappings

Example

$$Q_1(a,b) :- R(a,b), R(b,c).$$

$$Q_2(x,y) :- R(x,y).$$

1.4 Containment Mappings

Example

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

Do containment mappings exist?

Q_1 -> Q_2: none exists
Q_2 -> Q_1: Ψ(x)=a, Ψ(y)=b

1.4 Containment Mappings

Example

Q_1(a,b): R(a,b), R(c,b).

Q_2(x,y): R(x,y).

Q_1 -> Q_2 :Ψ(a)=x, Ψ(b)=y, Ψ(c)=x

Q_2 -> Q_1 :Ψ(x)=a, Ψ(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 = {hop(a,b), hop(b,c)} -> Q^I :- hop(a,b), hop(b,c)**
  - Canonical instance I^Q for query Q
    - Interpret each conjunct as a tuple
    - Interpret variables as constants
    - **Q :- hop(a,a) -> I^Q = {hop(a,a)}**

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 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

Example

$Q_1()$: $R(a,b)$, $R(c,b)$.

$Q_2()$: $R(x,y)$.

$Q_2 \rightarrow Q_1$: $\Psi(x)=a$, $\Psi(y)=b$

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

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

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

$\Psi\,\varphi(x)=1$, $\Psi\,\varphi(y)=2$

1.4 Containment Background

- Containment Mapping <-> Containment
- Proof idea (boolean queries)
  - (only-if direction)
    - Assume Q_2 contained in Q_1
    - Consider canonical (frozen) database I^{Q2}
    - Evaluating Q_1 over I^{Q2} and taking a variable mapping that is produced as a side-effect gives us a containment mapping

1.4 Containment Mappings

Example

Q_1(): R(a,b), R(c,b).

Q_2(): R(x,y).

Q_2 -> Q_1 :Ψ(x)=a, Ψ(y)=b

I^{Q1} = {(a,b),(c,b)}

$Q_2(I^{Q1})$={()}

φ(x)=a, φ(y)=b

φ is our containment mapping Ψ

1.4 Containment Background

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

ILLINOIS INSTITUTE OF TECHNOLOGY

Example

Q_1(): **R(a,b)**, R(c,b).

Q_2(): **R(x,y)**.

Q_2 -> Q_1 :Ψ(x)=a, Ψ(y)=b

If there exists tuples

R(a,b) and R(c,b)

in R that make Q_1 true, then we take

R(a,b)

to fulfill Q_2

1.4 Containment Background

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

1.4 Containment Mappings

Example

Q_1(**a**): **R(a,b)**, R(c,b).

Q_2(**x**): **R(x,y)**.

Q_2 -> Q_1 :$\Psi(x)=a$, $\Psi(y)=b$

For every

R(a,b) and R(c,b)

Q_1 returns (a) and for every

R(a,b)

Q_2 returns (a)

1.4 Similarity Measures

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

1.4 Similarity Measures

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

1.4 Similarity Measures

Definition: Similarity Measure

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

- 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]**

1.4 Similarity Measures

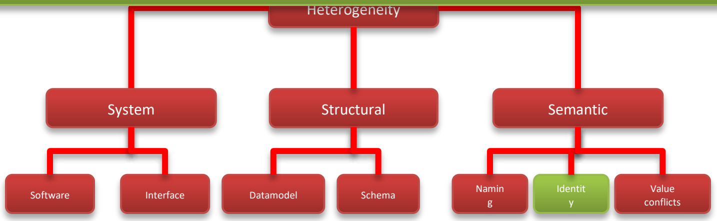
ILLINOIS INSTITUTE OF TECHNOLOGY

Example

String equality: d(p,q) = 0 if p=q
strings d(p,q) = 1 else

Euclidian distance: d(p,q) = $\sqrt{\sum_{i=1}^{n}(p[i]-q[i])^2}$
N-dimensional space

Edit distance: d(p,q) = minimum number of single character insertions, deletions, replacements to transform p into q
strings



1.4 Similarity Measures

Definition: Metric

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

• **Non-negative**	$d(p,q) >= 0$
• **Symmetry**	$d(p,q) = d(q,p)$
• **Identity of indiscernibles**	$d(p,q) = 0$ iff $p=q$
• **Triangle inequality**	$d(p,q) + d(q,r) >= d(p,r)$

1.4 Similarity Measures

Definition: Metric

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

• **Non-negative**	$d(p,q) >= 0$
• **Symmetry**	$d(p,q) = d(q,p)$
• **Identity of indiscernibles**	$d(p,q) = 0$ iff $p=q$
• **Triangle inequality**	$d(p,q) + d(q,r) >= d(p,r)$

1.4 Similarity Measures

- **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 Neighboorhood-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

ILLINOIS INSTITUTE OF TECHNOLOGY

CS520
Data Integration, Warehousing, and Provenance

2. Data Preparation and Cleaning

IIT DBGroup

Boris Glavic

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

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

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

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

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

2. Cleaning Methods

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

2. Overview

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

2.1 Cleaning Methods

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

2.1 Constraint Languages

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

2.1 Constraint Languages (cont.)

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

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

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

2.1 Example Constraints

Example: Constraints Languages

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

C_1: The zip code uniquely determines the city

C_2: Nobody should earn more than their direct superior

C_3: Salaries are non-negative

2.1 Example Constraints

Example: Constraints Languages

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

C_1: The zip code uniquely determines the city
- expressible as functional dependency

C_2: Nobody should earn more than their direct superior
- e.g., denial constraint

C_3: Salaries are non-negative
- e.g., denial constraint

2.1 Example Constraints

Example: Constraints Languages

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

C$_1$: The zip code uniquely determines the city

FD$_1$: zip -> city

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

C$_2$: Nobody should earn more than their direct superior

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

C$_3$: Salaries are non-negative

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

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

Definition: Constraint Repair Problem

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

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

2.1 Constraint based Cleaning Overview

- 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

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

FD$_1$: zip -> city

2.1 Example Constraints

Example: Constraint Violations

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

FD$_1$: `zip -> city`

2.1 Example Constraints

Example: Constraint Violations

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

How to repair?

Deletion:

- remove some conflicting tuples
- quite destructive

Update:

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

2.1 Constraint based Cleaning Overview

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

2.1 Example Constraints

Example: Constraint Repair

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

Deletion:

Delete Chicago or Schaumburg?

Delete New York or the two Chicago tuples?

- one tuple deleted vs. two tuples deleted

2.1 Example Constraints

Example: Constraint Repair

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

Update equate RHS:

Update Chicago->Schaumburg or Schaumburg->Chicago

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

Update disequate LHS:

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

2.1 Constraint based Cleaning Overview

- **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 …**

2.1 Detecting Violations

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

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

2.1 Example Constraints

Example: SQL Violation Detection

Relation: Person(name,city,zip)

FD1: zip -> city

Violation Detection Query

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

To know which tuples caused the conflict:

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

2.1 Fixing Violations

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

2.1 Constraint Repair Problem

Definition: Constraint Repair Problem (restated)

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

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

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

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

2.1 Cost Metrics

- **Deletion + Insertion**

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

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

$$\sum_{t \in R} \sum_{A \in Schema(R)} d(t.A, t'.A)$$

2.1 Cost Metrics

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

2.1 Naïve FD Repair Algorithm

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

2.1 Constraint Repair

ILLINOIS INSTITUTE OF TECHNOLOGY

Example: Constraint Repair

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

t_1 and t_4: set t_1.city = Chicago

t_1 and t_5: set t_1.city = Chicago

t_2 and t_3: set t_2.city = Schaumburg

2.1 Problems with the Algorithm

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

2.1 Constraint Repair

Example: Constraint Repair

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

t_4 and t_1: set t_4.city = New York

t_1 and t_5: set t_1.city = Chicago

t_2 and t_3: set t_2.city = Schaumburg

Now t_1 and t_4 and t_4 and t_5 in violation!

2.1 Problems with the Algorithm

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

2.1 Problems with the Algorithm

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

2.1 Constraint Repair

Example: Constraint Repair

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

t_4 and t_1: set t_4.city = New York
t_1 and t_5: set t_1.city = Chicago

Now t_1 and t_4 and t_4 and t_5 in violation!

t_4 and t_1: set t_1.city = New York
T_5 and t_4: set t_4.city = Chicago

repeat

2.1 Problems with the Algorithm

ILLINOIS INSTITUTE OF TECHNOLOGY

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

2.1 Constraint Repair

Example: Constraint Repair

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

Cheaper: t_1.city = Chicago

Not so cheap: set t_4.city and t_5.city = New York

2.1 Problems with the Algorithm

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

2.1 Problems with the Algorithm

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

2.1 Problems with the Algorithm

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

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

2.1 Consistent Query Answering

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

2.1 Constraint Repair

Example: Constraint Repair

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

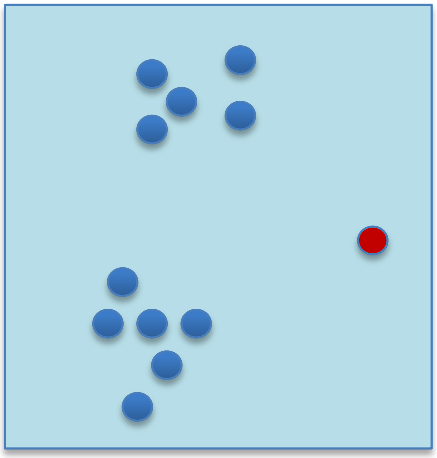
Cheaper: t_1.city = Chicago

Not so cheap: set t_4.city and t_5.city = New York

2. Overview

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

2.2 Statistical and Outlier



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

2. Overview

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

2.3 Entity Resolution

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

2.3 Entity Resolution

Definition: Entity Resolution Problem

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

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

2.3 Entity Resolution

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

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

SSN	zip	city	name
3333333333	IL 60616		Petre

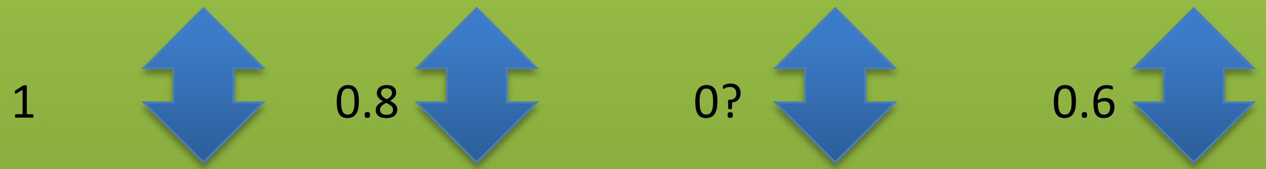
2.3 Entity Resolution

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

2.3 Entity Resolution

Example: Per attribute similarity

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



SSN	zip	city	name
3333333333	IL 60616		Petre

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?

2.3 Entity Resolution

Definition: Edit Distance

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

Example:

```
NEED -> STREET

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

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

2.3 Entity Resolution

Example:

```
NEED -> STREET

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

d(NEED,STREET) = 4
```

2.3 Entity Resolution

- **Principal of optimality**
  - Best solution of a subproblem is part of the best solution for the whole problem

- **Dynamic programming algorithm**
  - D(i,j) is the edit distance between prefix of len i of s and prefix of len j of s’
  - D(len(s),len(s’)) is the solution
  - Represented as matrix
  - Populate based on rules shown on the next slide

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

    }

2.3 Entity Resolution

Example:

`NEED -> STREET`

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

2.3 Entity Resolution

ILLINOIS INSTITUTE OF TECHNOLOGY

Example:

NEED -> STREET

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

2.3 Entity Resolution

Example:

NEED -> STREET

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

2.3 Entity Resolution

Example:

NEED -> STREET

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

2.3 Entity Resolution

Example:

`NEED -> STREET`

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

2.3 Entity Resolution

Example:

```
NEED -> STREET
```

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

2.3 Entity Resolution

Example:

`NEED -> STREET`

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

2.3 Entity Resolution

Example:

`NEED -> STREET`

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

2.3 Entity Resolution – Distance Measures

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

2.3 Entity Resolution – Distance Measures

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

2.3 Entity Resolution – Distance Measures

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

2.3 Entity Resolution

Example: Tokenization

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

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

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

2.3 Entity Resolution – Distance Measures

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

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

2.3 Entity Resolution

Example: Tokenization

Input string:

S = “nanotubes are used in these experiments to…”
S’= “we consider nanotubes in our experiments…”
S’’= “we prove that P=NP, thus solving …”

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

$d_{jacc}(S,S')=$

$d_{jacc}(S,S'')=$

$d_{jacc}(S',S'')=$

2.3 Entity Resolution

Example: Tokenization

Input string:

S = “nanotubes are used in these experiments to…”
S’= “we consider nanotubes in our experiments…”
S’’= “we prove that P=NP, thus solving …”

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

$d_{jacc}(S,S')$ = 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]**

      -> "Gender:male,Salary:9000"

      or -> "male,9000"
  - Combine distance measures for single attributes
    - Weighted sum or more complex combinations
      - E.g., $d(t, t') = w_1 \times d_A(t.A, t'.A) + w_2 \times d_B(t.B, t'.B)$
  - Use quadratic distance measure
    - E.g., earth-movers distance

2.3 Entity Resolution

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

2.3 Entity Resolution

- **Weighted linear combination**
  - Say tuples have **n** attributes
  - **w_i**: predetermined weight of an attribute
  - **d_i(t,t')**: similarity measure for the **i**th attribute

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

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

2.3 Entity Resolution

Example: Weighted sum of attribute similarities

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

1 0.8 0? 0.6

SSN	zip	city	name
3333333333	IL 60616		Petre

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

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

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

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

2.3 Entity Resolution

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

2.3 Entity Resolution

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

2.3 Entity Resolution

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

2.3 Entity Resolution

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

2.3 Entity Resolution

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

2.3 Entity Resolution

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

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. Overview

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

2.4 Data Fusion

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

Outline

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

ILLINOIS INSTITUTE OF TECHNOLOGY

CS520 Data Integration, Warehousing, and Provenance

3. Schema Matching and Mapping

IIT DBGroup

Boris Glavic

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

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

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

Outline

ILLINOIS INSTITUTE OF TECHNOLOGY

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

3. Why matching and mapping?

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

3. Why matching and mapping?

ILLINOIS INSTITUTE OF TECHNOLOGY

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

3. Why matching and mapping?

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

3. Overview

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

3.1 Schema Matching

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

3.1 Schema Matching

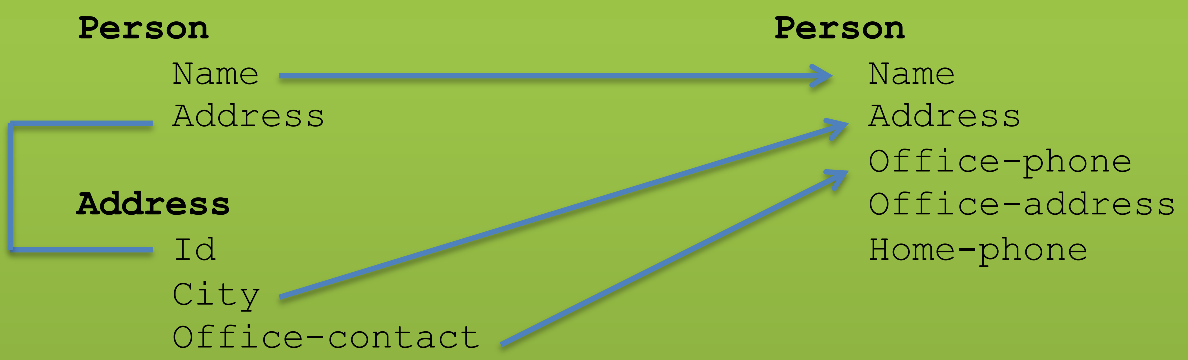
- **Why is this hard?**
  - **Insufficient information**: schema does not capture full semantics of a domain
  - **Schemas can be misleading**:
    - E,g., attributes are not necessarily descriptive
    - E.g., finding the right way to translate attributes not obvious

3.1 Schema Matching

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

3.1 Schema Matching

Example: Types of Matching



Person
Name
Address

Address
Id
City
Office-contact

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

Name	Address
Peter	1
Alice	3
Bob	3

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343	Chicago, IL 60655	(333) 323 3344
Alice	Chicago	(312) 555 7777	Chicago, IL 60633	(123) 323 3344
Bob	New York	(465) 123 1234	New York, NY 55443	(888) 323 3344

3.1 Schema Matching

ILLINOIS INSTITUTE OF TECHNOLOGY

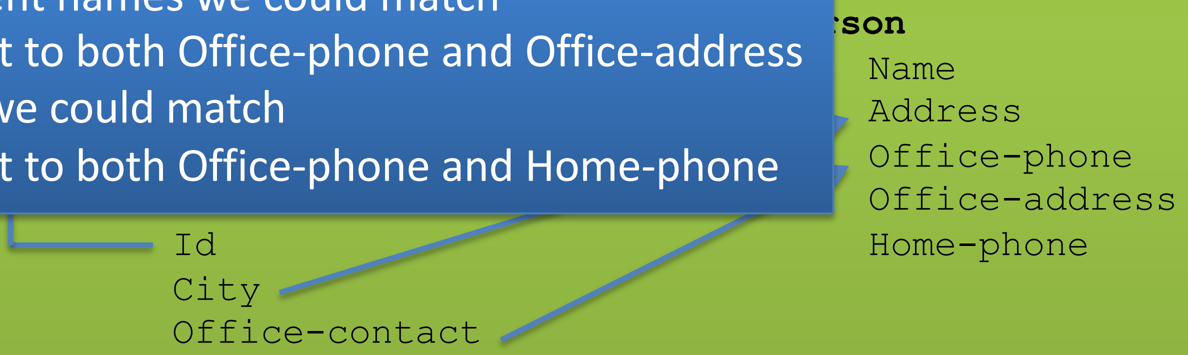
Example: Types of Matching

Based on element names we could match

- Office-contact to both Office-phone and Office-address

Based on data we could match

- Office-contact to both Office-phone and Home-phone

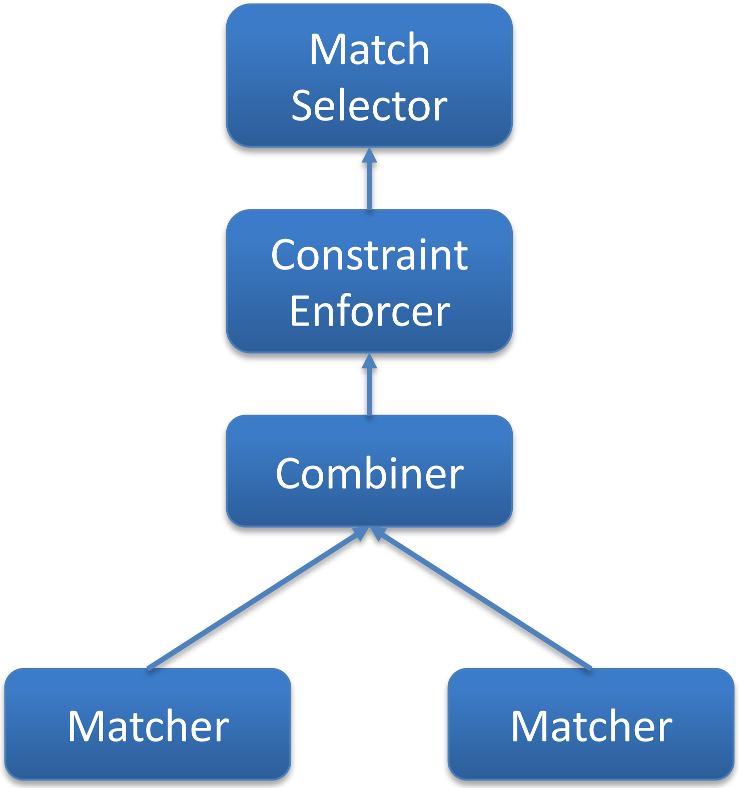


Name	Address
Peter	1
Alice	3
Bob	3

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343	Chicago, IL 60655	(333) 323 3344
Alice	Chicago	(312) 555 7777	Chicago, IL 60633	(123) 323 3344
Bob	New York	(465) 123 1234	New York, NY 55443	(888) 323 3344

- **Typical Matching System Architecture**



Determine actual matches

Use constraints to modify similarity matrix

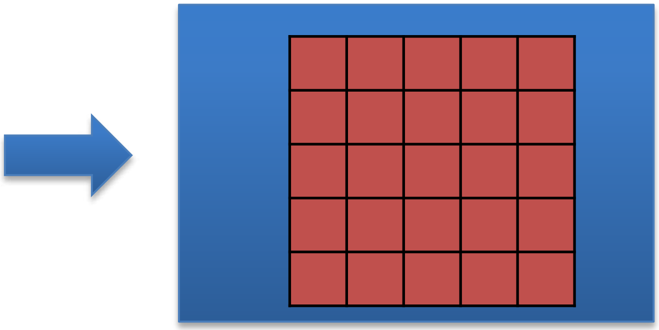
Combine individual similarity matrices

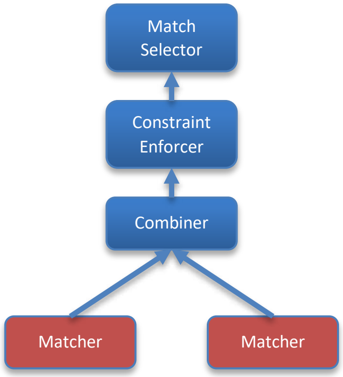
Each matcher uses one type of information to compute similarity matrix

3.1 Schema Matching

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

```
Person                     Person
      Name                       Name
      Address                    Address
                                 Office-phone
Address                          Office-address
      Id                         Home-phone
      City
      Office-contact
```





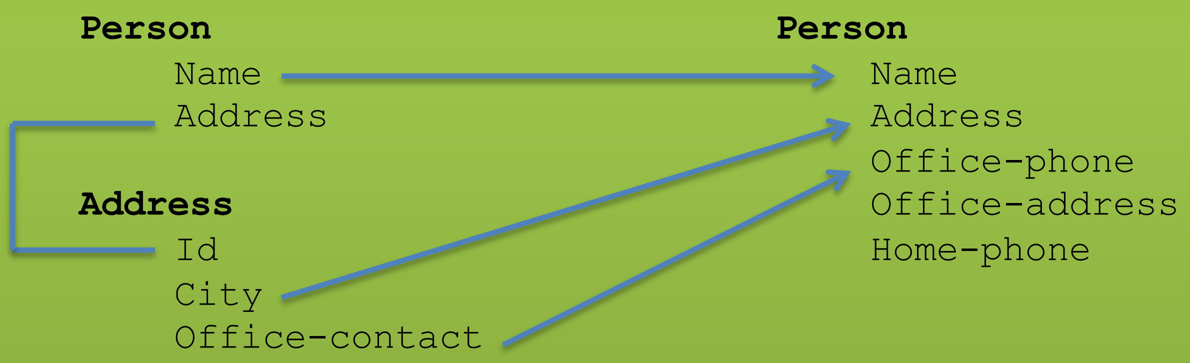
3.1 Schema Matching

ILLINOIS INSTITUTE OF TECHNOLOGY

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

3.1 Schema Matching

Example: Types of Matching



```
Person                                   Person
      Name                                     Name
      Address                                  Address
                                               Office-phone
Address                                        Office-address
      Id                                       Home-phone
      City
      Office-contact
```

	Name	Address	Office-phone	Office-address	Home-phone
Name	**1**	0	0	0	0
Address	0	**1**	0	**0.4**	0
Id	0	0	0	0	0
City	0	0	0	0	0
Office-contact	0	0	**0.5**	**0.5**	0

3.1 Schema Matching

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

3.1 Schema Matching

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

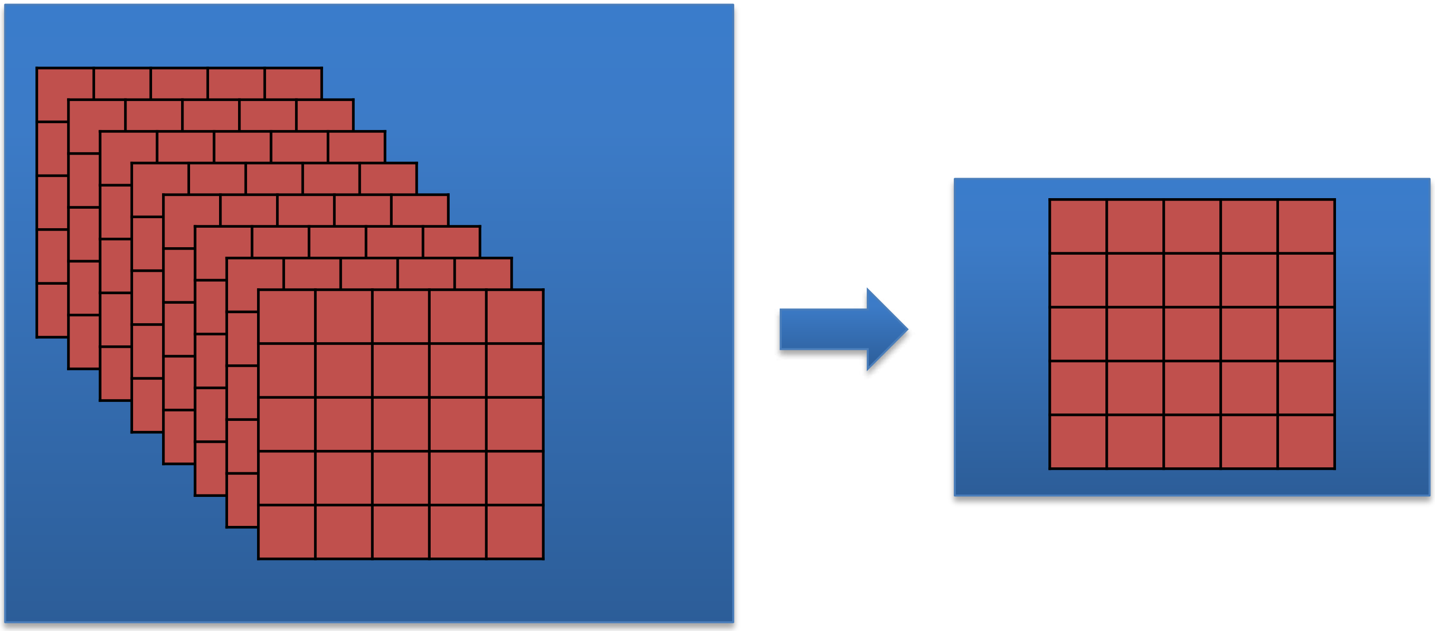
3.1 Schema Matching

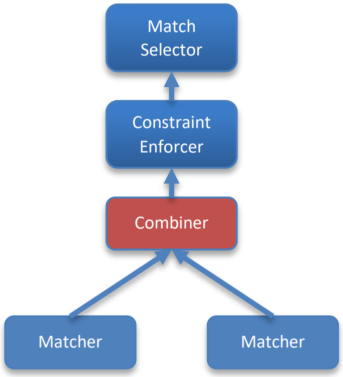
- **Overlap of attribute domains**
  - Each attribute value is a token
  - Use set-based similarity measure such as Jaccard
- **Classifier**
  - Train classifier to identify values of one attribute **A** from the source
    - Training set are values from **A** as positive examples and values of other attributes as negative examples
  - Apply classifier to all values of attributes from target schema
    - Aggregate into similarity score

3.1 Schema Matching

ILLINOIS INSTITUTE OF TECHNOLOGY

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



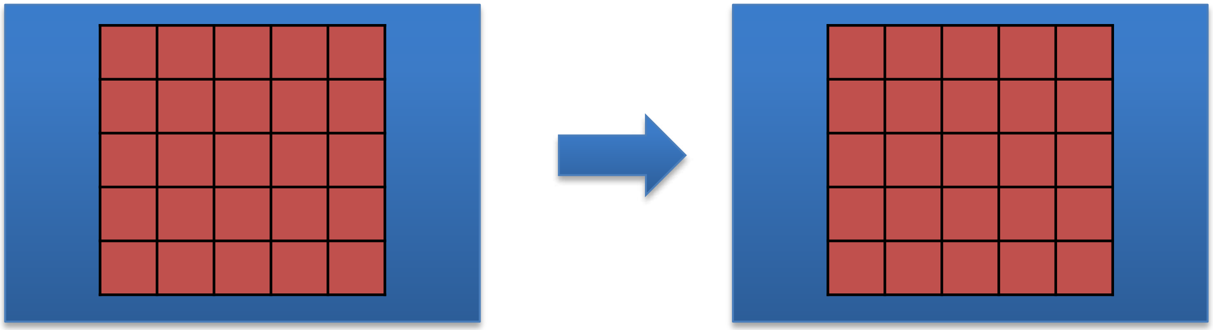


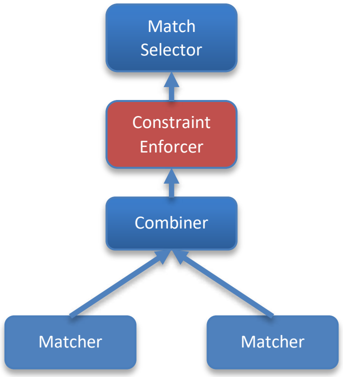
3.1 Schema Matching

- **Combiner**
  - Merge similarity matrices produced by the matchers into single matrix
  - Typical strategies
    - Average, Minimum, Max
    - Weighted combinations
    - Some script

3.1 Schema Matching

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





3.1 Schema Matching

- **Constraint Enforcer**
  - Determine most probably match by assigning each attribute from source to one target attribute
    - Multiple similarity scores to get likelihood of match combination to be true
  - 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

Example: Constraints

Constraint 1: An attribute matched to **source.cust-phone** has to get a score of 1 from the phone regexpr matcher

Constraint 2: Any attribute matched to **source.fax** has to have fax in its name

Constraint 3: If an attribute is matched to **source.firstname** with score > 0.9 then there has to be another attribute from the same target table that is matched to **source.lastname** with score > 0.9

3.1 Schema Matching

- **How to search match combinations**
  - Full search
    - Exponentially many combinations potentially
  - Informed search approaches
    - A* search
  - Local propagation
    - Only local optimizations

3.1 Schema Matching

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

3.1 Schema Matching

- **A* search**
  - Estimated cost of a state **f(n) = g(n) + h(n)**
    - **g(n)** = cost of path from start state to **n**
    - **h(n)** = lower bound for path from **n** to goal state
  - No path reaching the goal state from **n** can have a total cost lower than **f(n)**

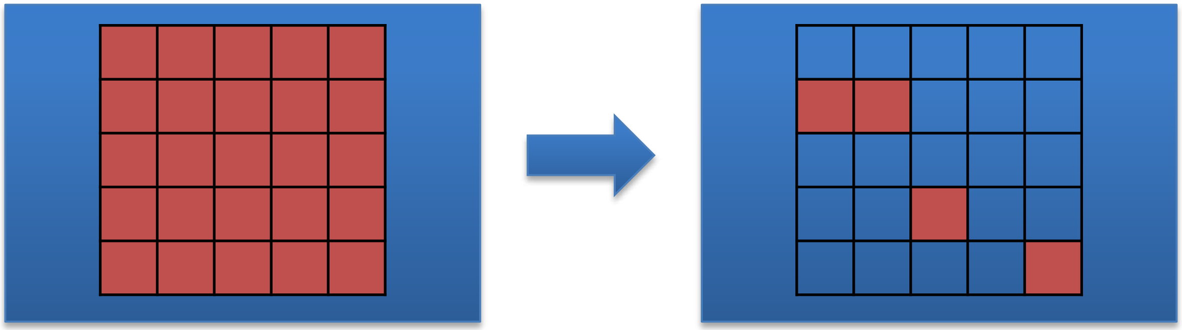
3.1 Schema Matching

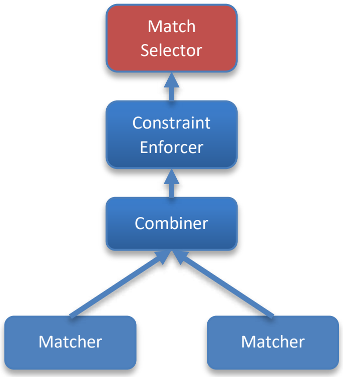
- **Algorithm**
  - Data structures
    - Keep a priority queue **q** of states sorted on f(n)
      - Initialize with start state
    - Keep set **v** of already visited nodes
      - Initially empty
  - While **q** is not empty
    - pop state **s** from head of **q**
    - If **s** is goal state return
    - 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
    - $[B_1, *, *, B_3]$
  - Initial state
    - [*, *, *, *]
  - Goal states
    - All states without *

3.1 Schema Matching

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





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

3.1 Schema Matching

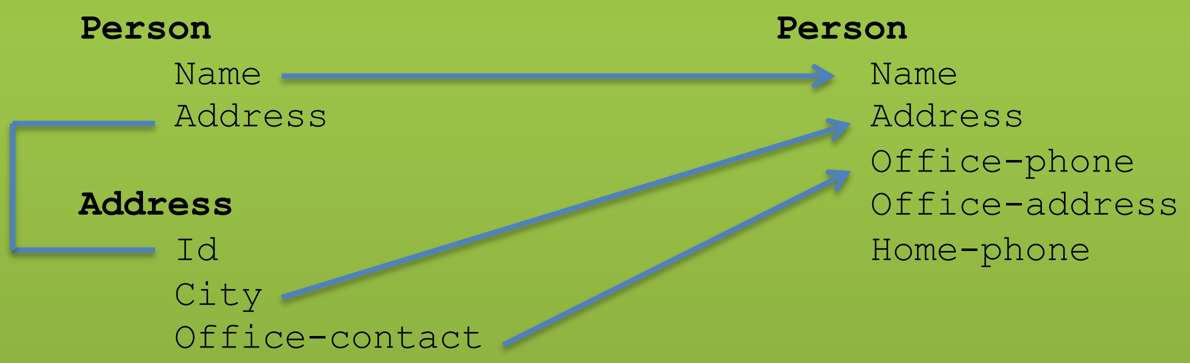
- **Many-to-many matchers**
  - Combine multiple columns using a set of functions
    - **E.g., concat, +, currency exchange, unit exchange**
  - Large or even unlimited search space
  - -> need method that explores interesting part of the search space
  - Specific searchers
    - Only concatenation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

3. Overview

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

3.2 Schema Mapping

Example: Matching Result



Person
Name
Address

Address
Id
City
Office-contact

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

Name	Address
Peter	1
Alice	3
Bob	3

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

Assume: We have data in the source as shown above

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

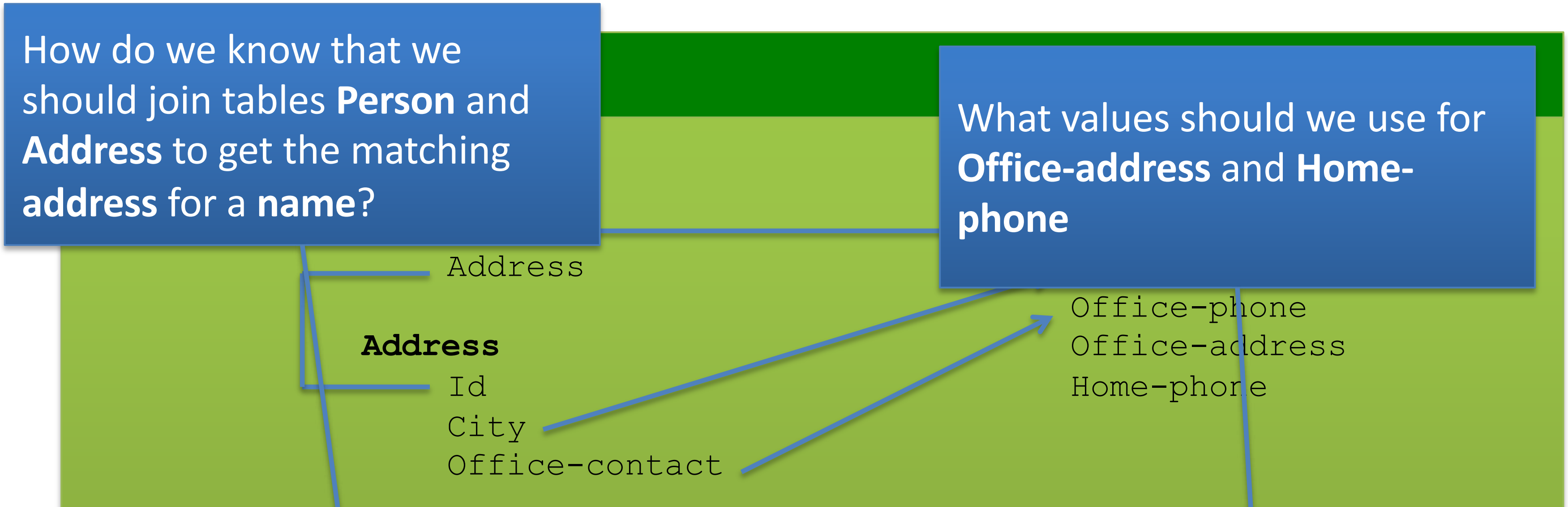
3.2 Schema Mapping

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

3.2 Schema Mapping

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**



Name	Address
Peter	1
Alice	3
Bob	3

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343	🟥	🟥
Alice	Chicago	(312) 555 7777	🟥	🟥
Bob	New York	(465) 123 1234	🟥	🟥

3.2 Schema Mapping

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

3.2 Schema Mapping

- **Instance-based definition of mappings**
  - Global schema **G**
  - Local schemas $\mathbf{S_1}$ to $\mathbf{S_n}$
  - 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 $(\mathbf{I_G} \times \mathbf{I_1} \times \ldots \times \mathbf{I_n})$
  - Useful as a different way to think about mappings, but not a practical way to define mappings

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

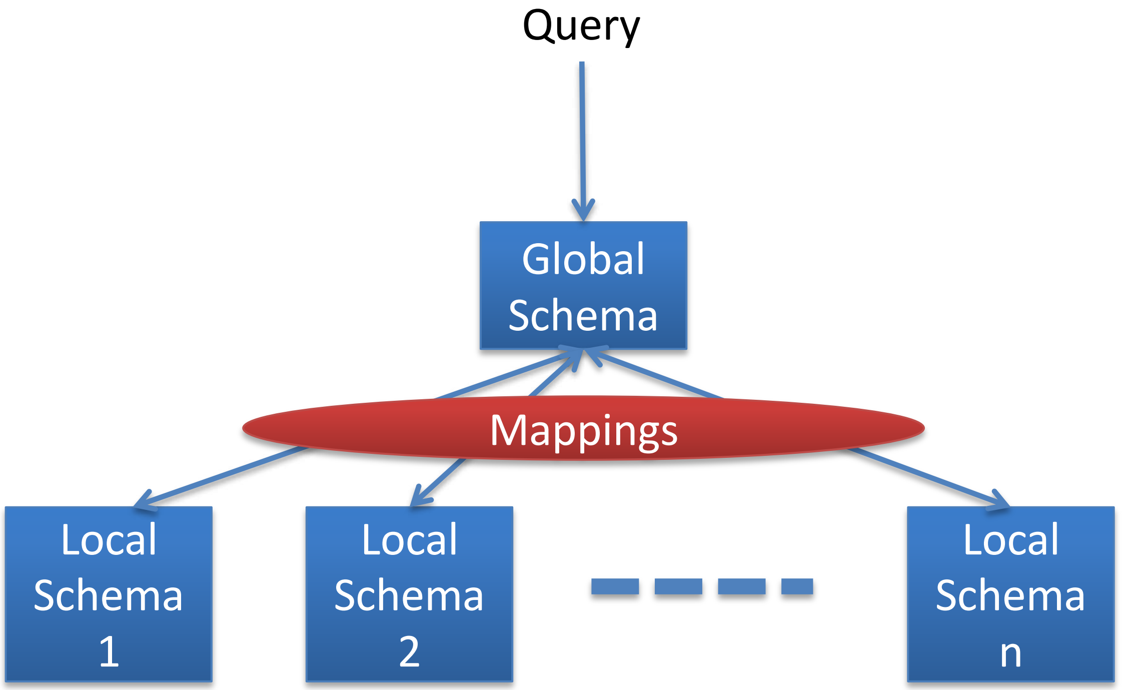
3.2 Schema Mapping

- **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

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

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

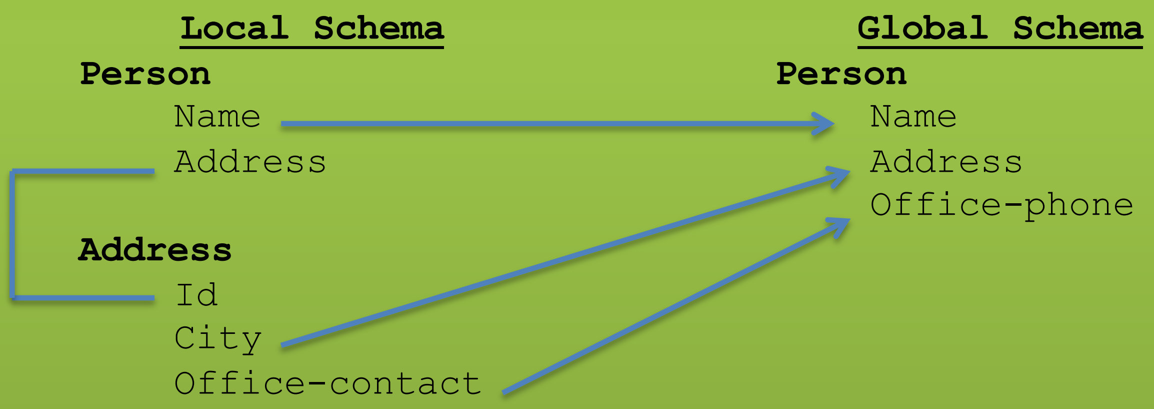


3.2 Schema Mapping

- **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: GAV



Local Schema

Person
Name
Address

Address
Id
City
Office-contact

Global Schema

Person
Name
Address
Office-phone

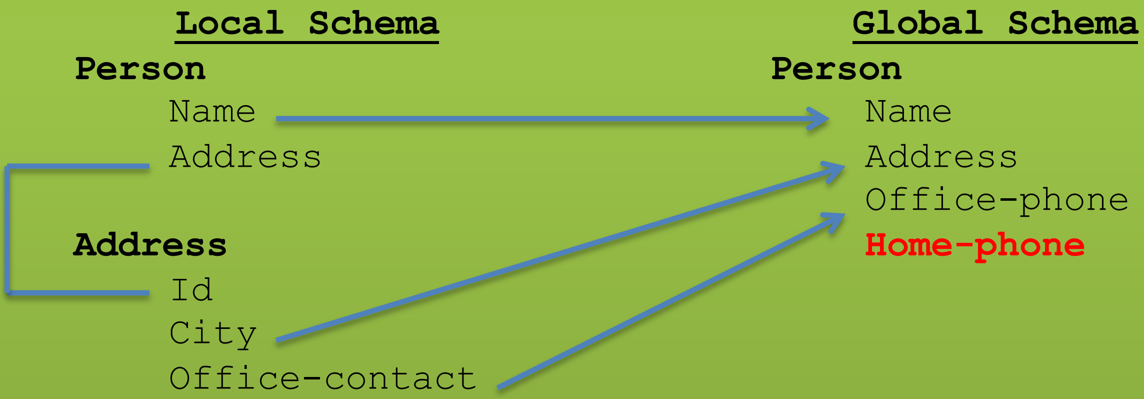
```
Q(X,Z,A) :- Person(X,Z,A)
= Q(X,Z,A) :- Person(X,Y), Address(Y,Z,A)
```

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

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

3.2 Schema Mapping

Example: GAV not possible



```
Q(X',Y',Z',A') :- Person(X',Y',Z',A')
= Q(X,Z,A, ????) :- Person(X,Y), Address(Y,Z,A)
```

Cannot be expressed as GAV mapping! No way to compute the Home-phone attribute values since there is no correspondence with a source attribute!

3.2 Schema Mapping

- **Global-as-view (GAV)**
- **Solutions (mapping M)**
  - Unique data exchange solution (later)
  - Intuitively, execute queries over local instance that produced global instance

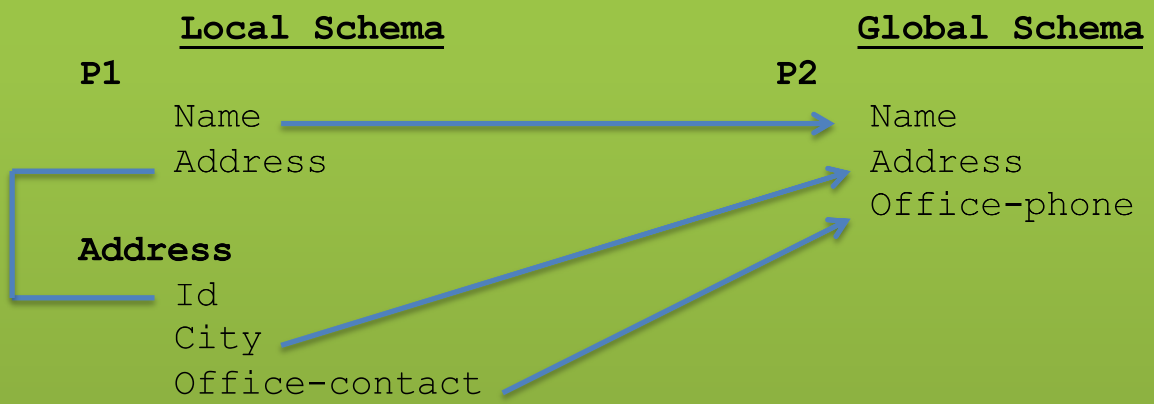
3.2 Schema Mapping

- **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

Example: GAV – query answering



Local Schema

P1
Name
Address

Address
Id
City
Office-contact

Global Schema

P2
Name
Address
Office-phone

GAV mapping:

```
P2(X,Z,A) = P1(X,Y), Address(Y,Z,A)
```

Query – Select Name from Persons

```
Q(A) :- P2(A,B,C)
```

View unfolding: Replace P2 with its definition

```
Q(A) :- P1(A,Y), Address(Y,B,C)
```

3.2 Schema Mapping

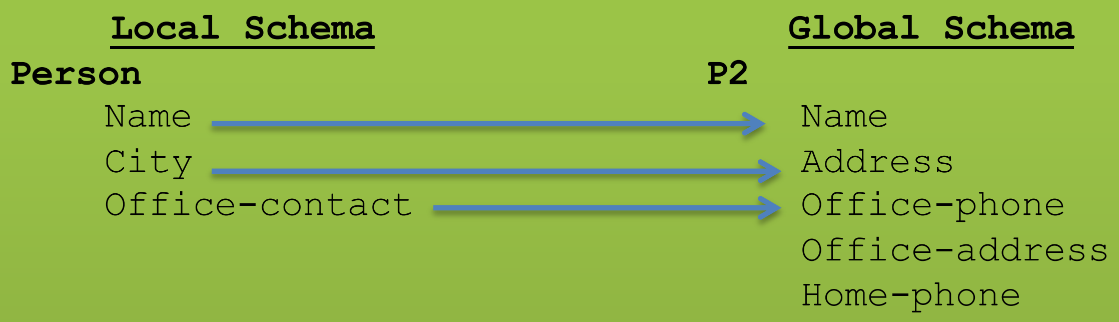
- **Global-as-view (GAV) Discussion**
  - Hard to add new source
    - -> have to rewrite the view definitions
  - Does not deal with missing values
  - Easy query processing
    - -> view unfolding

3.2 Schema Mapping

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

3.2 Schema Mapping

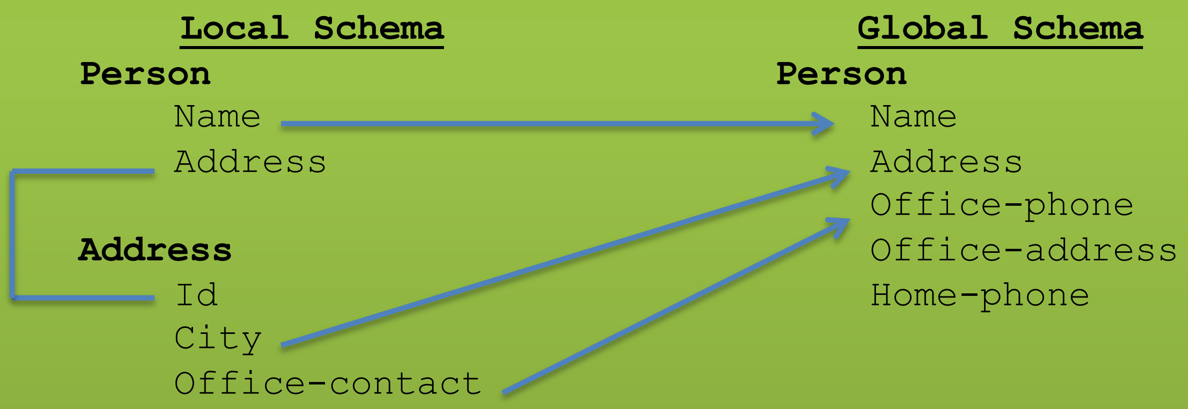
Example: LAV



```
Person(X,Y,Z) = P2(X,Y,Z,A,B)
```

3.2 Schema Mapping

Example: LAV not possible



Local Schema

Person
Name
Address

Address
Id
City
Office-contact

Global Schema

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

Cannot deal with attributes from the local schema that do not have a correspondence with attributes in the global schema

```
Person(X,???) = Person(X,Y,Z,A,B)
Address(???,Y,Z) = Person(X,Y,Z,A,B)
```

3.2 Schema Mapping

- **Local-as-view (LAV)**
- **Solutions (mapping M)**
  - Incompleteness possible

  => There may exist many solutions

3.2 Schema Mapping

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

3.2 Schema Mapping

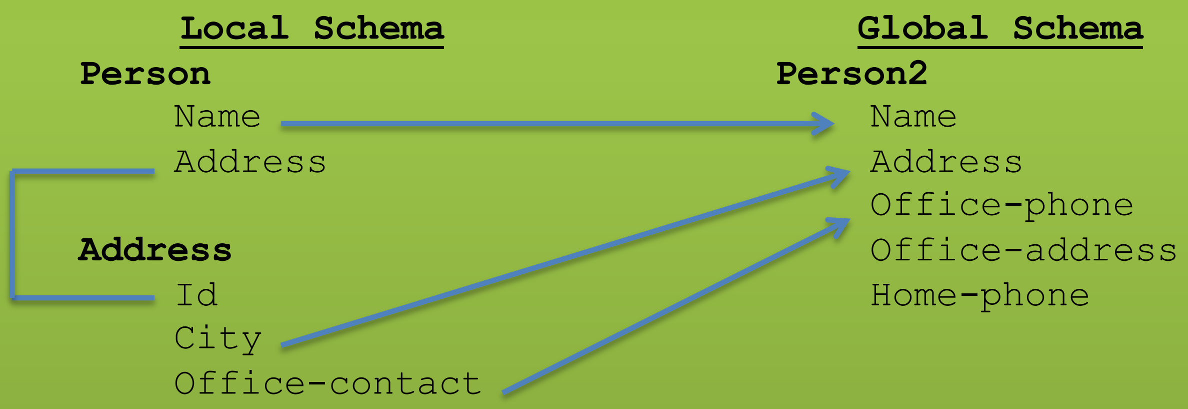
- **Local-as-view (LAV) Discussion**
  - Easy to add new sources
    - -> have to write a new view definition
    - May take some time to get used to expressing sources like that
  - Still does not deal gracefully with all cases of missing values
    - Loosing correlation
  - Hard query processing
    - Equivalent rewriting using views only
    - Later: give up equivalence

3.2 Schema Mapping

- **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) ⊇Q(S)

3.2 Schema Mapping

Example: GLAV



Local Schema

Person
Name
Address

Address
Id
City
Office-contact

Global Schema

Person2
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’) :- Person2(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 LAV
  - 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)**
  - Logical way of expressing GLAV mappings
    - LHS formula is a conjunction of source (local) relation atoms (and comparisons
    - RHS formula is a conjunction of target (global) relation atoms and comparisons

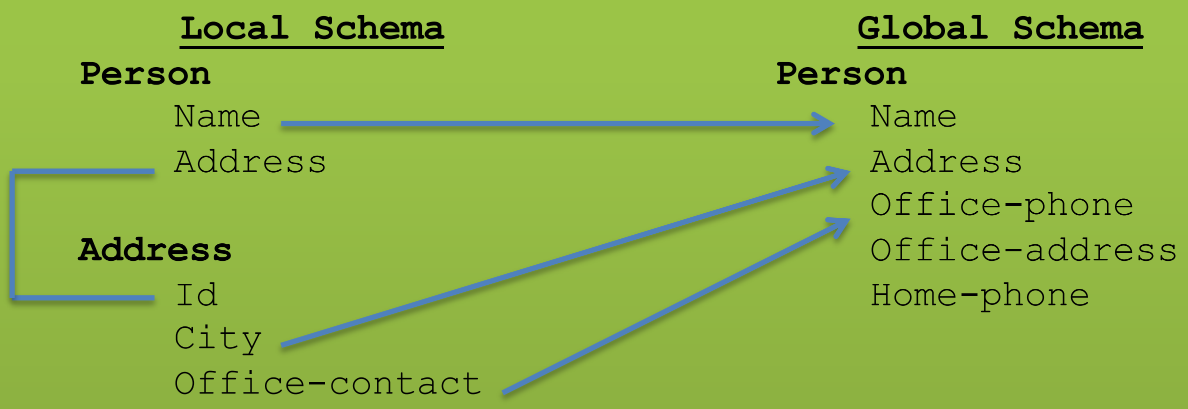
$$\forall \vec{x} : \phi(\vec{x}) \rightarrow \exists \vec{y} : \psi(\vec{x}, \vec{y})$$

  - Equivalence to a containment constraint:

$$Q'(G) \supseteq Q(S)$$

3.2 Schema Mapping

Example: Types of Matching



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

```
Source: Q(X,Y,Z) :- Person(X′,Y′), Address(Y′,Z′,A′)
=
Target: Q(X′,Y′,Z′) :- Person(X′,Y′,Z′,A′,B′)
```

- **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 loose 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 Mapping Generation Algorithm**
  - **Inputs**: Source and Target schemas, matches
  - **Output**: Mapping from source to target schema
  - Note, Clio works for nested schemas such as XML too not just for relational data.
    - Here we will look at the relational model part only

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** (tgds)
    - There are also chase procedures that consider **edgs** (e.g., PKs)
  - Starting point are all single relational atoms
    - E.g., R(X,Y)

3.2 Schema Mapping

- **Chase step**
  - Works on **tabelau**: set of relational atoms
  - A chase step takes one tgd t where the LHS is fulfilled and the RHS is not fulfilled
    - We fulfill the tgd t by adding new atoms to the tableau and mapping variables from t to the actually occuring variables from the current tablau
- **Chase**
  - Applying the chase until no more changes
  - Note: if there are cyclic constraints this may not terminate

- **Clio Algorithm: 1) Find logical associations**
  - Compute chase R(X) for each atom R in source and target
  - Each chase result is a logical association
  - Intuitively, each such logical association is a possible way to join relations in a schema based on the FK constraints

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

Outline

0) Course Info

1) Introduction

2) Data Preparation and Cleaning

3) Schema matching and mapping

4) Virtual Data Integration

5) Data Exchange

6) Data Warehousing

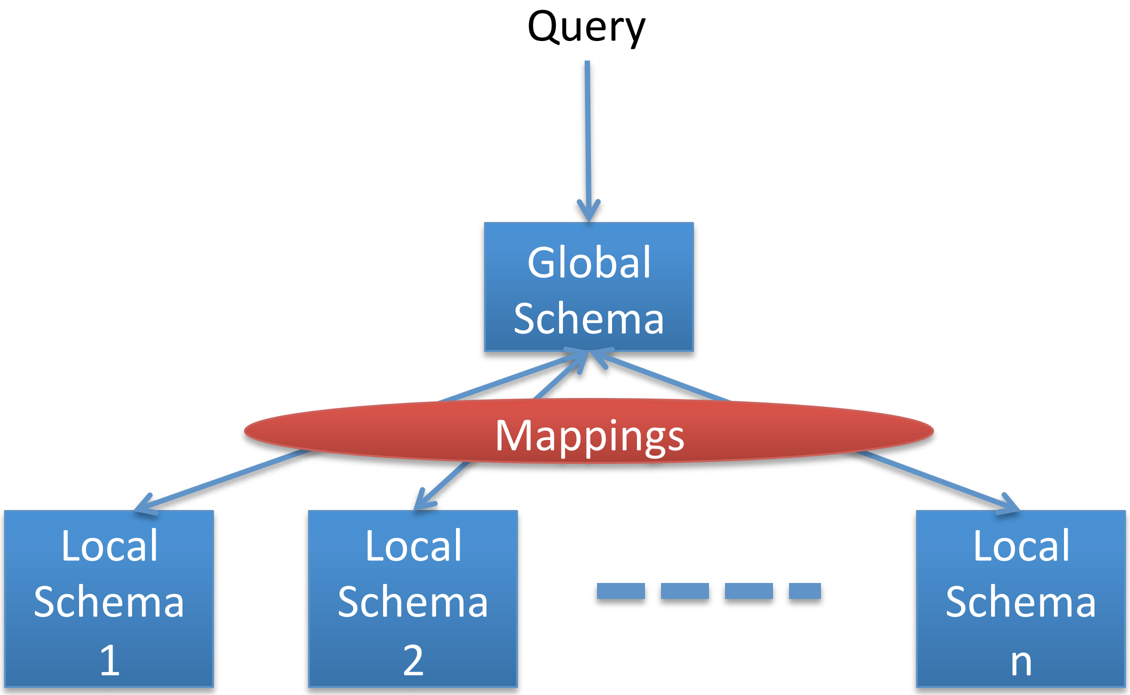
7) Big Data Analytics

8) Data Provenance

4. Virtual Data Integration

ILLINOIS INSTITUTE OF TECHNOLOGY

- **Virtual Data Integration**



4. Virtual Data Integration

Problems:

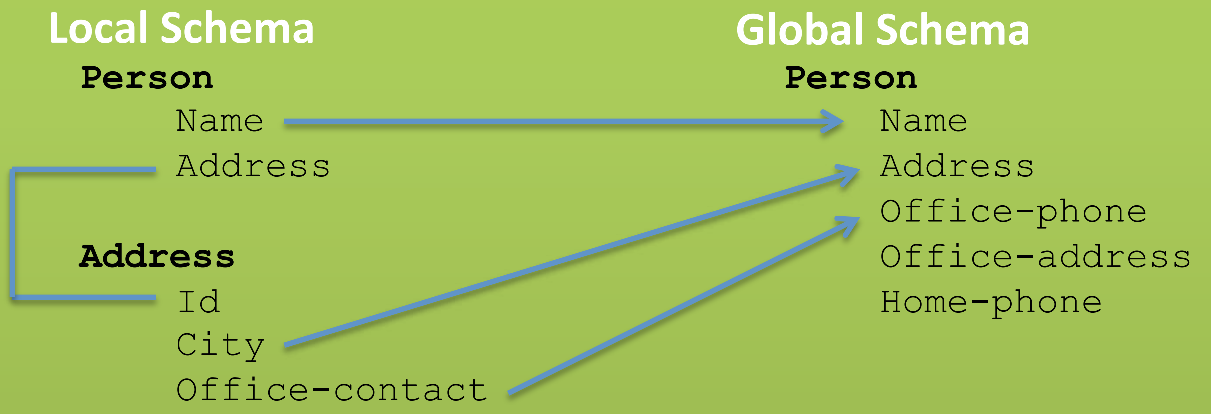
- How to create mappings?
  - – Discussed in previous part of the course
- How to compute query Q
  - – This is the main focus of this part

4. Query Answering with Views

- **How to compute query Q over global schema based on source schemas only?**
  - What language is used to express mappings?
  - What language 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?

4.1 Query Answering with Views

Example: Solutions



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

Query: `Q(Name) :- Person(Name, A, OP, OA, HP).`

Name	Address
Peter	1
Alice	2
Bob	3

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

4.1 Query Answering with Views

Example: Solutions

Local Schema

Name	Address
Peter	1
Alice	2
Bob	3

Global Schema

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

```
City
Office-contact
```

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

Query: **`Q(Name) :- Person(Name, A, OP, OA, HP).`**

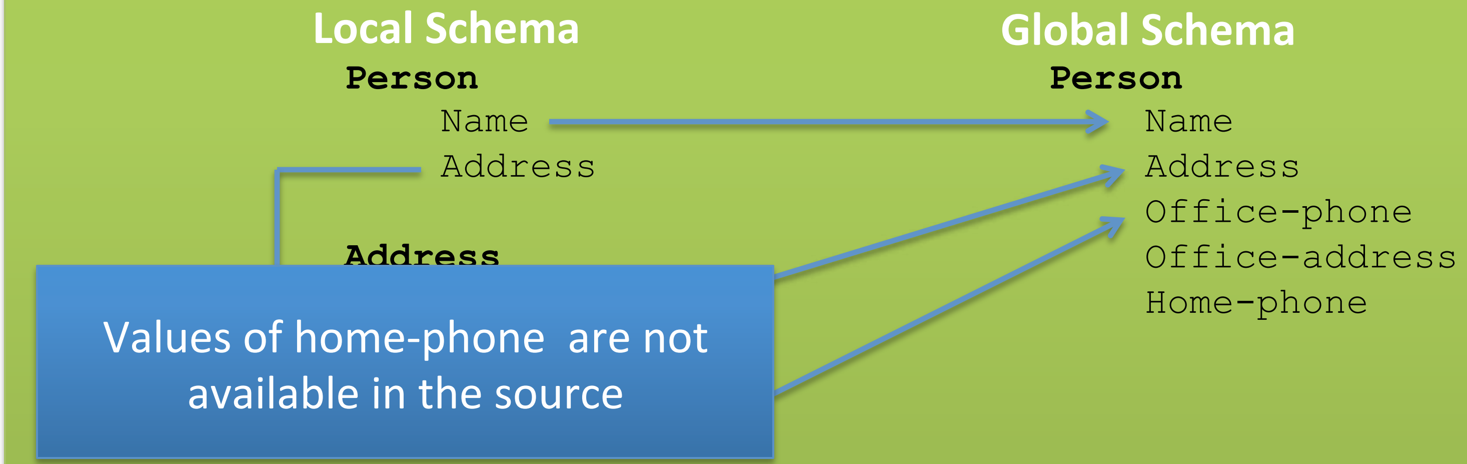
Rewritten query over the source:

```
Q(Name) :- Person(Name, AI),
           Address(AI,A,OP).
```

Name
Peter
Alice
Bob

4.1 Query Answering with Views

Example: Solutions



Values of home-phone are not available in the source

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

Query: `Q(Home-ph) :- Person(N, A, OP, OA, Home-ph).`

Name	Address
Peter	1
Alice	2
Bob	3

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

4. Query Answering with Views

- **Problems**
  - How to determine whether 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)

$$Q(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_1(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1940, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_1 \supseteq Q \qquad \Rightarrow \qquad Q'(T,Y,D) :- V_1(T,Y,D), Y \geq 1950$$

Containment is enough to show that V_1 can be used to answer Q.

Motivating Example (Part 2)

$$Q(T,Y,D):-Movie(I,T,Y,G),Y\geq 1950,G="comedy"$$
$$Director(I,D),Actor(I,D)$$

$$V_2(I,T,Y):-Movie(I,T,Y,G),Y\geq 1950,G="comedy"$$

$$V_3(I,D):-Director(I,D),Actor(ID,D)$$

Containment does not hold, but intuitively, *V_2* and *V_3* are useful for answering *Q*.

$$Q''(T,Y,D):-V_2(I,T,Y),V_3(I,D)$$

How do we express that intuition?

Answering queries using views!

Problem Definition

Input: Query Q

View definitions: $V_1, \ldots, V_n$

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

An equivalent rewriting of Q using $V_1, \ldots, V_n$:

a rewriting Q', such that $Q' \Leftrightarrow Q$

Naïve approach

- **Given Q and views**
  - Randomly combine views into a query Q’
  - Check equivalence of Q’ and Q
  - If Q’ is equivalent we are done
  - Else repeat
- **Why is this not good?**
  - There are infinitely many ways of combining views
    - E.g., V, V x V, V x V x V, …
  - We are not using any information in the query

Motivating Example (Part 3)

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

$$Q(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_4(I,T,Y) :- Movie(I,T,Y,G), \underline{Y \geq 1960}, G = "comedy"$$

$$V_3(I,D) :- Director(I,D), Actor(ID,D)$$

$$Q'''(T,Y,D) :- \underline{V_4(I,T,Y)}, V_3(I,D)$$

maximally-contained rewriting

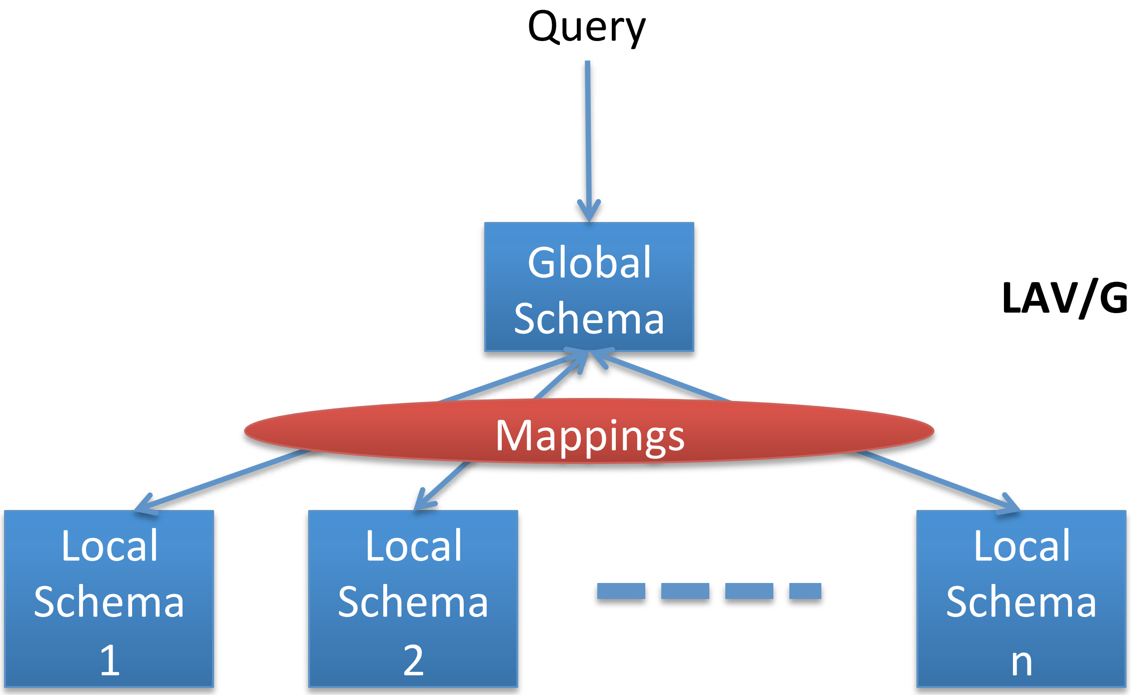
Maximally-Contained Rewritings

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

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

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

Why again?



LAV/GLAV!

Other use-cases

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

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

$$Q(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_2(I,T,Y) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$

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

$$V_6(T,Y) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$

$$V_7(I,T,Y) :- Movie(I,T,Y,G), Y \geq 1950,$$
$$G = "comedy", Award(I,W)$$

$$V_8(I,T) :- Movie(I,T,Y,G), Y \geq 1940, G = "comedy"$$

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: $Q(\overline{X}) :- p_1(\overline{X_1}),\ldots,p_n(\overline{X_n})$

Rewriting: $Q'(\overline{X}) :- V_1(\overline{X_1}),\ldots,V_m(\overline{X_m})$

φ

Expansion: $Q''(\overline{X}) :- \underbrace{g_1^1,\ldots g_k^1}, \ldots, \underbrace{g_1^m,\ldots,g_j^m}$

Proof: Only *n* subgoals in Q can contribute to the image of the containment mapping φ

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:

- 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

The Bucket Algorithm

Step 1 Intuition

- A view can only be used to provide information about a goal R(X) if it has a goal R(Y)
  - `Q(X) :- R(X,Y)`
  - `V(X) :- S(X,Y)`
- If the query goal contains variables that are in the head of the query, then the view is only useful if it gives access to these values (they are in the head)
  - `Q(X) :- R(X,Y)`
  - `V(X) :- S(X,Y), R(Y,Z)`

Bucket Algorithm in Action

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

$$V_1(I, Y) :- Movie(I, T, Y, G), Revenues(I, A), I \geq 5000, A \geq \$200M$$
$$V_2(I, A) :- Movie(I, T, Y, G), Revenues(I, A)$$
$$V_3(I, A) :- Revenues(I, A), A \leq \$50M$$
$$V_4(I, D, Y) :- Movie(I, T, Y, G), Director(I, D), I \leq 3000$$

View atoms that can contribute to *Movie*:

V_1(**ID**,year'), V_2(**ID**,A'), V_4(**ID**,D' ,year'')

Buckets and Cartesian product

Movie(ID,title, year,genre)	**Revenues(ID, amount)**	**Director(ID,**dir**)**
V_1(**ID**,year)	V_1(**ID**,Y')	V_4(**ID**,**Dir**,Y')
V_2(**ID**,A')	V_2(**ID**,amount)	
V_4(**ID**,D' ,year)		

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

$$q_1'(ID,dir) :- V_1(ID,\cancel{year}),V_1(ID,y'),V_4(ID,dir,y')$$

Next Candidate Rewriting

ILLINOIS INSTITUTE OF TECHNOLOGY

Movie(ID,title,year,genre)	Revenues(ID,amount)	Director(ID,dir)
V_1(**ID**,year)	V_1(**ID**,Y')	V_4(**ID**,Dir,Y')
V_2(**ID**,A')	V_2(**ID**,amount)	
V_4(**ID**,D' ,year)		

$$q_2'(ID,dir) :- V_2(ID,A'),V_2(ID,amount),V_4(ID,dir,y')$$

$$q_2'(ID,dir) :- V_2(ID,amount),V_4(ID,dir,y'),$$
$$amount \geq \$100M$$

The Bucket Algorithm

Step 2:

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

The 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 MiniCon Algorithm

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

$$\begin{tikzcd}[column sep=small]
Director(ID,dir) \arrow[d] & Actor(ID,dir) \arrow[d] \\
Director(I,D) & Actor(I,A)
\end{tikzcd}$$

$$V_5(D,A) :- Director(I,D), Actor(I,A)$$

Intuition: The variable *I* is not in the head of V_5, hence V_5 cannot be used in a rewriting.
MiniCon discards this option early on, while the Bucket algorithm does not notice the interaction.

MinCon Algorithm Steps

- **1) Create MiniCon descriptions (MCDs)**:
  - Homomorphism on view heads
  - Each MCD covers a set of subgoals in the query with a set of subgoals in a view
- **2) Combination step**:
  - Any set of MCDs that covers the query subgoals (without overlap) is a rewriting
  - No need for an additional containment check!

MiniCon Descriptions (MCDs)

An atomic fragment of the ultimate containment mapping

$$Q(title, act, dir) :- Movie(ID, title, year, genre),$$
$$Director(ID, dir), Actor(ID, act)$$

$$V(I, D, A) :- Director(I, D), Actor(I, A)$$

MCD:

mapping:

$$ID \rightarrow I$$
$$dir \rightarrow D$$
$$act \rightarrow A$$

covered subgoals of *Q: {2,3}*

MCDs: Detail 1

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

$$V(I, D, A) :- Director(I, D), Actor(I, A)$$

Need to specialize the view first:

$$V'(I, D, D) :- Director(I, D), Actor(I, D)$$

MCD:

mapping:

$$ID \rightarrow I$$

$$dir \rightarrow D$$

covered subgoals of Q: {2,3}

MCDs: Detail 2

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

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

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

MCD:

mapping:

$$ID \rightarrow I$$

$$dir \rightarrow D$$

covered subgoals of Q *still:* {2,3}

Inverse-Rules Algorithm

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

Inverse Rules by Example

ILLINOIS INSTITUTE OF TECHNOLOGY

Given the following view:

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

And the following tuple in *V_7*:

V_7(79,Manhattan,1979,Comedy)

Then we can infer the tuple:

Movie(79,Manhattan,1979,Comedy)

Hence, the following ‘rule’ is sound:

IN$_1$: *Movie(I,T,Y,G) :- V_7(I,T,Y,G)*

Skolem Functions

ILLINOIS INSTITUTE OF TECHNOLOGY

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

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
Rewriting = Inverse Rules + Query

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

Given Q2, 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)

(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

$$Query \quad and \quad view:$$

$$Q(X,Y) :- e_1(X,Z), e_2(Z,Y)$$

$$V(A,B) :- e_1(A,C), e_2(C,B)$$

$$Inverse \quad rules:$$

$$e_1(A, f_1(A,B)) :- V(A,B)$$

$$e_2(f_1(A,B), B) :- V(A,B)$$

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:

$$V_8(dir) :- Movie(ID, dir, actor)$$

$$V_9(actor) :- Movie(ID, dir, actor)$$

And suppose the extensions of the views are:

V_8: {Allen, Copolla}
V_9: {Keaton, Pacino}

Possible Databases

ILLINOIS INSTITUTE OF TECHNOLOGY

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

DB1. {(Allen, Keaton), (Coppola, Pacino)}
DB2. {(Allen, Pacino), (Coppola, Keaton)}

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

Certain answers are those true in *all* databases that are consistent with the views and their extensions.

Certain Answers: Formal Definition [Open-world Assumption]

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

Certain Answers [Closed-world Assumption]

- Given:
  - Views: $V_1,\dots,V_n$
  - View extensions $v_1,\dots v_n$
  - 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

$V_8(dir) :- Director(ID, dir)$

$V_9(actor) :- Actor(ID, actor)$

V8: {Allen}

V9: {Keaton}

$Q(dir, actor) :- Director(ID, dir), Actor(ID, actor)$

Under closed-world assumption:

single DB possible ⇒ (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

In Other News…

ILLINOIS INSTITUTE OF TECHNOLOGY

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

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

$$v_1(X) :- color(X,Y) \qquad I(V_1) = V$$

$$v_2(Y) :- color(X,Y) \qquad I(V_2) = \{red, green, blue\}$$

$$v_3(X,Y) :- edge(X,Y) \qquad I(V_3) = E$$

$$q() :- edge(X,Y), color(X,Z), color(Y,Z)$$

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

Interpreted Predicates

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

Outline

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

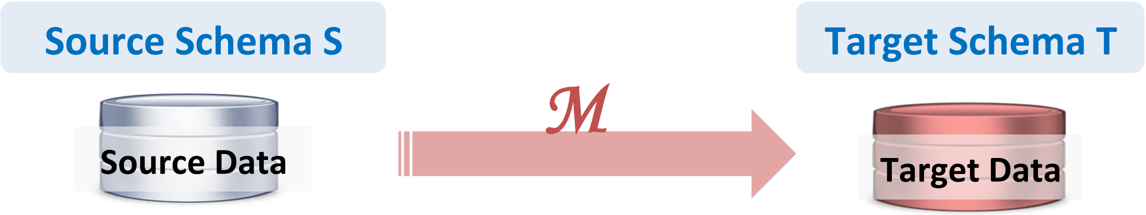
Outline

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

- **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"

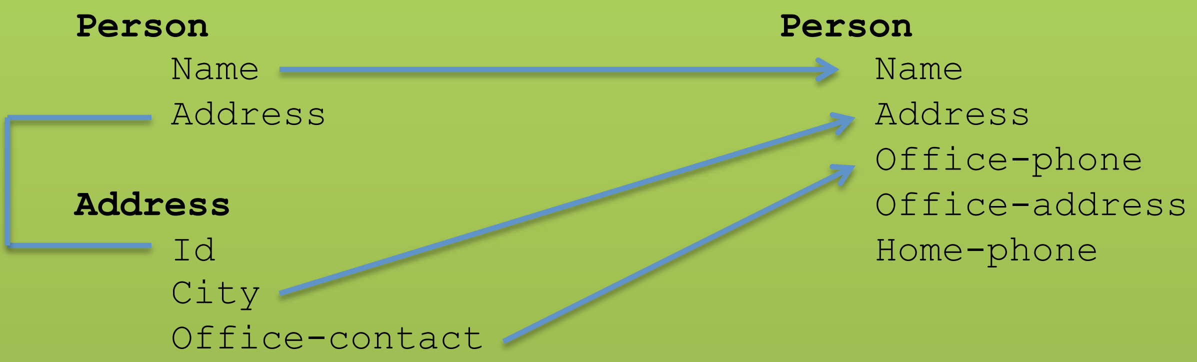
5. Data Exchange

- **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



5. Data Exchange

Example: Types of Matching



Person
Name
Address

Address
Id
City
Office-contact

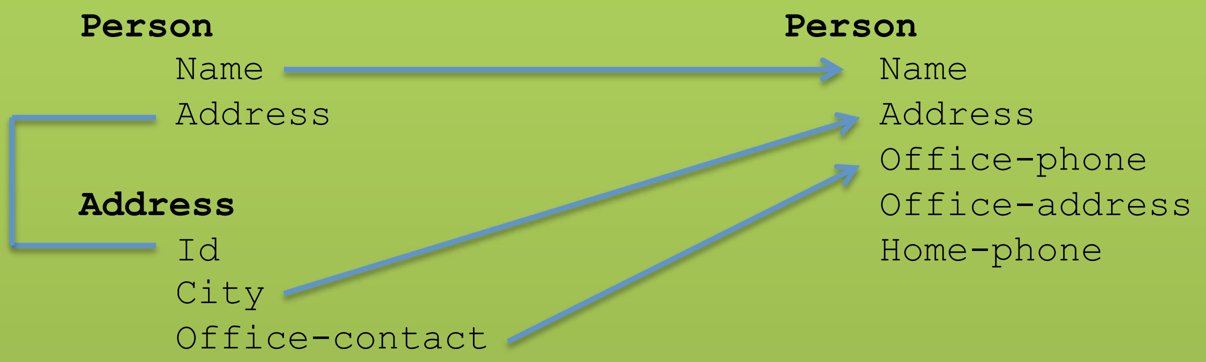
Person
Name
Address
Office-phone
Office-address
Home-phone

Name	Address
Peter	1
Alice	3
Bob	3

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

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

Example: Types of Matching



Person
Name
Address

Address
Id
City
Office-contact

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

Name	Address
Peter	1
Alice	2
Bob	3

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

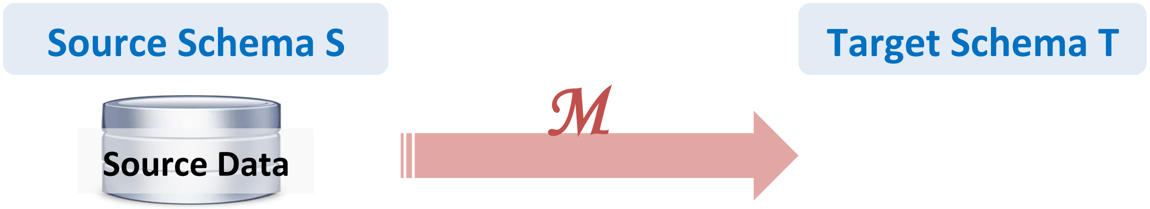
Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343		
Alice	Chicago	(312) 555 7777		
Bob	New York	(465) 123 1234		

5.1 Data Exchange Setting

Definition: Data Exchange Setting

Data Exchange setting is a tuple (**S**,**T**,**I**,**Σ**)

- Schema **S**
- Schema **T**
- Instance **I** of **S**
- Mappings **Σ** from **S** to **T**

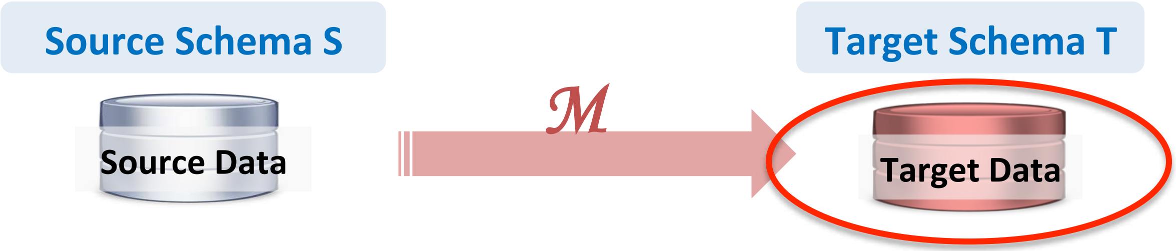


5.1 Data Exchange Solutions

Definition: Data Exchange Solution

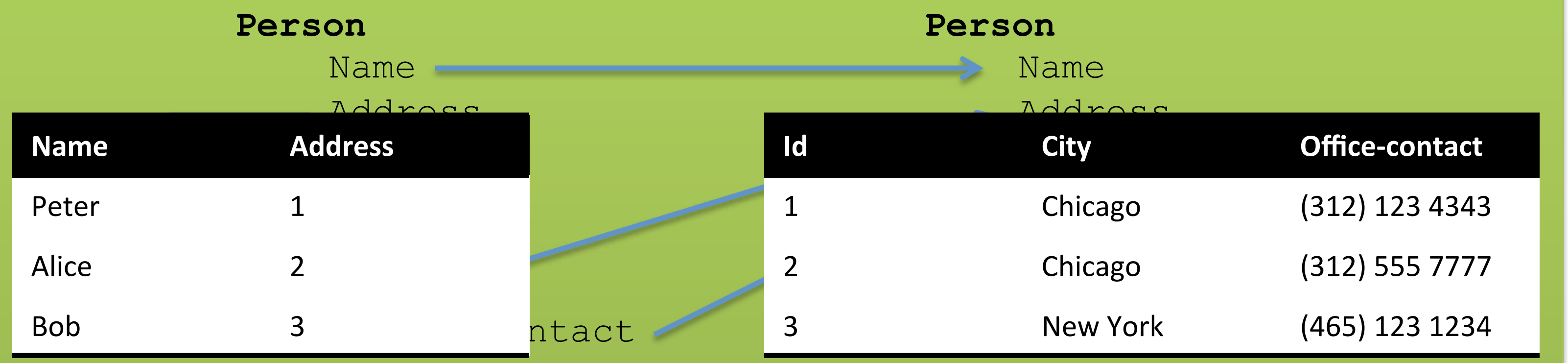
Given data exchange setting is a tuple (**S,T,I,Σ**)

- Find instance **J** of **T** so that **(I,J)** fulfills mappings **Σ**
- **J** uses values from a **universe U** and set of **labeled nulls N**



5.1 Data Exchange Solutions

Example: Solutions



Name	Address
Peter	1
Alice	2
Bob	3

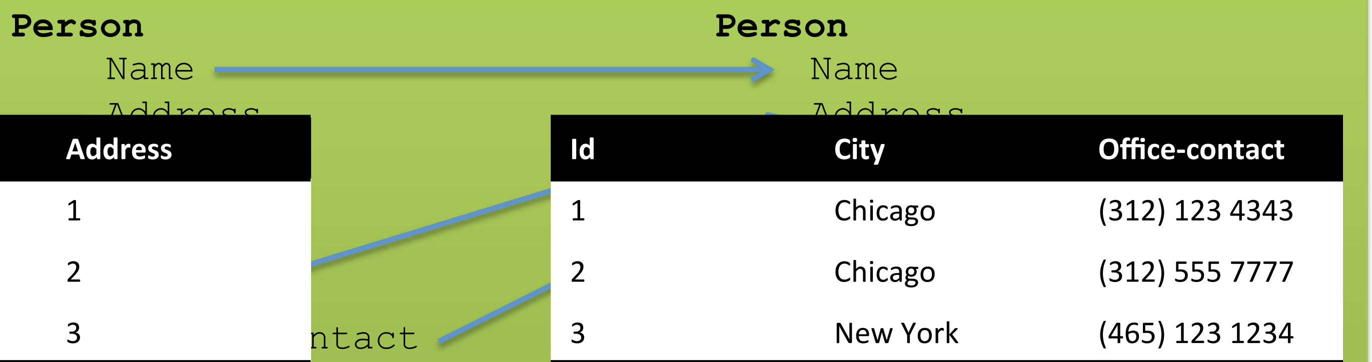
Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

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

Can we come up with a solution?

5.1 Data Exchange Solutions

Example: Solutions



Name	Address
Peter	1
Alice	2
Bob	3

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

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

Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343	NULL	NULL
Alice	Chicago	(312) 555 7777	NULL	NULL
Bob	New York	(465) 123 1234	NULL	NULL

5.1 Number of Solutions

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

5.1 Number of Solutions

- **Target instance domain**
  - Consider a **universe U**
    - Source instance can only use values from U
  - Consider an infinite **set N of labeled nulls**
    - Target instance can use these as placeholders for missing values

5.1 Data Exchange Solutions

Example: Multiple Solutions

Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343	X	Y
Alice	Chicago	(312) 555 7777	A	A
Bob	New York	(465) 123 1234	C	D

Id
City

Home-phone

Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343	X	Y
Alice	Chicago	(312) 555 7777	A	A
Bob	New York	(465) 123 1234	C	D
Heinzbert	Pferdegert	111-222-3798	E	

Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343	Hometown	111-322-3454
Alice	Chicago	(312) 555 7777	A	A
Bob	New York	(465) 123 1234	Other town	D

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 Data Exchange Solutions

Example: Solution generality

Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343	X	Y
Alice	Chicago	(312) 555 7777	A	A
Bob	New York	(465) 123 1234	C	D

How general is solution (in terms of certain answers)?

Consider query

```
Q(n) :- P(n,a,op,oa,hp), oa = Hometown
```

Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343	Hometown	111-322-3454
Alice	Chicago	(312) 555 7777	A	A
Bob	New York	(465) 123 1234	Other town	D

5.1 Universal solutions

- **Universal solution**
  - Want a solution that is as general as possible
  - We call such most general solutions universal solutions
  - How do we know whether it is most general
    - We can map the tuples in this solution to any other less general solution by replacing unspecified values (labelled nulls) with actual data values
- **Query answering with universal solutions**
  - For UCQs: run query over universal instance
  - Remove tuples with labelled nulls
  - Result are the certain answers!

5.1 Universal Solutions

ILLINOIS INSTITUTE OF TECHNOLOGY

Definition: Homomorphism

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

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

Definition: Universal solution

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

- There exists a homomorphism from **J** to **J'**

5.1 Data Exchange Solutions

Example: Solution generality

Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343	X	Y
Alice	Chicago	(312) 555 7777	A	A
Bob	New York	(465) 123 1234	C	D

How general is solution (in terms of certain answers)?

Consider query

```
Q(n) :- P(n,a,op,oa,hp), oa = Hometown
```

5.1 Data Exchange Solutions

Example: Solution generality

Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343	X	Y
Alice	Chicago	(312) 555 7777	A	A
Bob	New York	(465) 123 1234	C	D

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,

Name	Address	Office-phone	Office-address	Home-phone
Peter	Chicago	(312) 123 4343	Hometown	111-322-3454
Alice	Chicago	(312) 555 7777	A	A
Bob	New York	(465) 123 1234	Other town	D

5.2 Computing Solutions

- **Note**
  - 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!

Source Schema S
Source Data
$\mathcal{M}$
Target Schema T
Target Data

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)

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

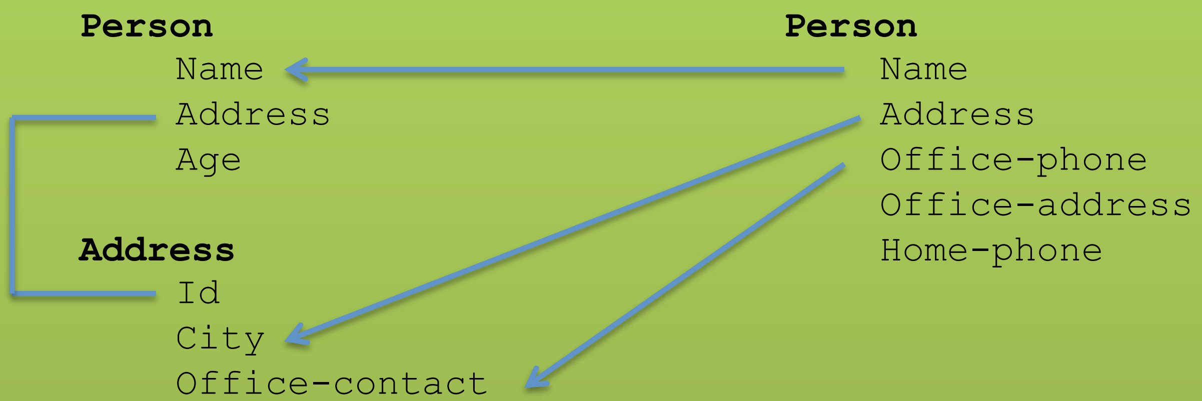
5.2 Skolem Functions

- **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?
    - Naïve: all universally quantified variables
    - Better: only relevant ones

5.2 Skolem Functions

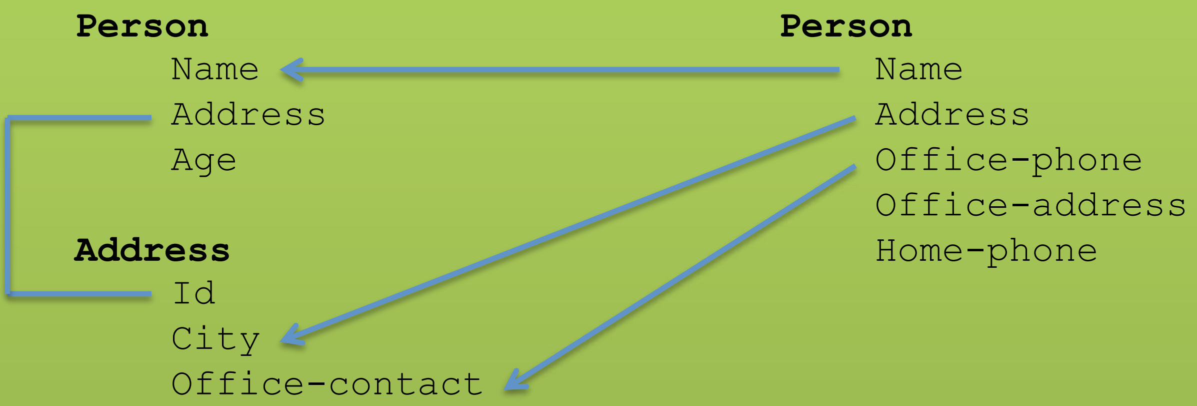
ILLINOIS INSTITUTE OF TECHNOLOGY

Example: Skolem Functions



5.2 Skolem Functions

Example: Skolem Functions



$$\forall a, b, c, d, e : Person(a, b, c, d, e) \rightarrow \exists f, g Person(a, f, g) \wedge Address(f, b, c)$$

Introduce skolem function **sk1** and **sk2** for **f** and **g**.

What arguments to choose for **sk1** and **sk2**?

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

5.2 Skolem Functions

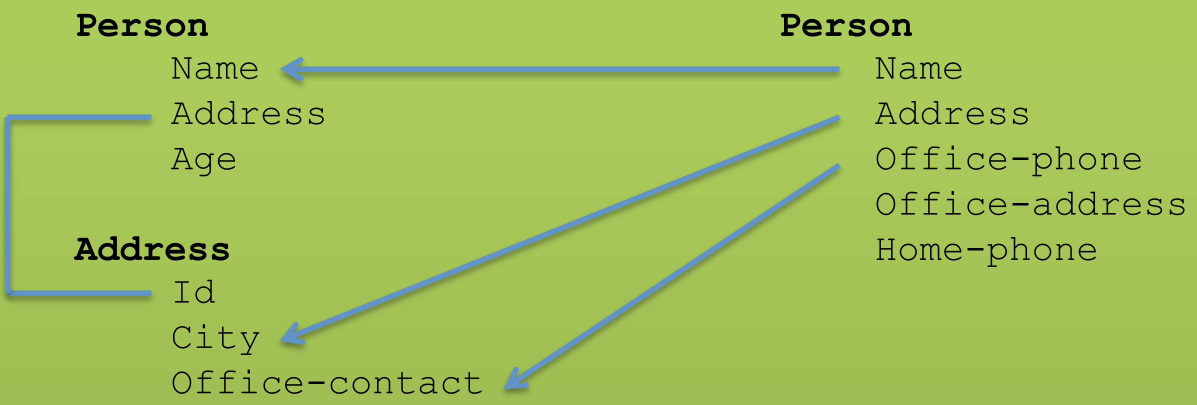
- **Clio Schema Graph Algorithm**
- **Nodes**
  - – Create a graph with one node for every target attribute and one node for every target relation
  - – Also add nodes for source attribute if they are copied to the target according to the mapping
- **Edges**
  - – Edges between a relation and its attributes
  - – Edges between target attributes that use the same variable
  - – Edges between source attributes and target attributes if they use the same variable

5.2 Skolem Functions

- **Clio Schema Graph Algorithm**
- **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

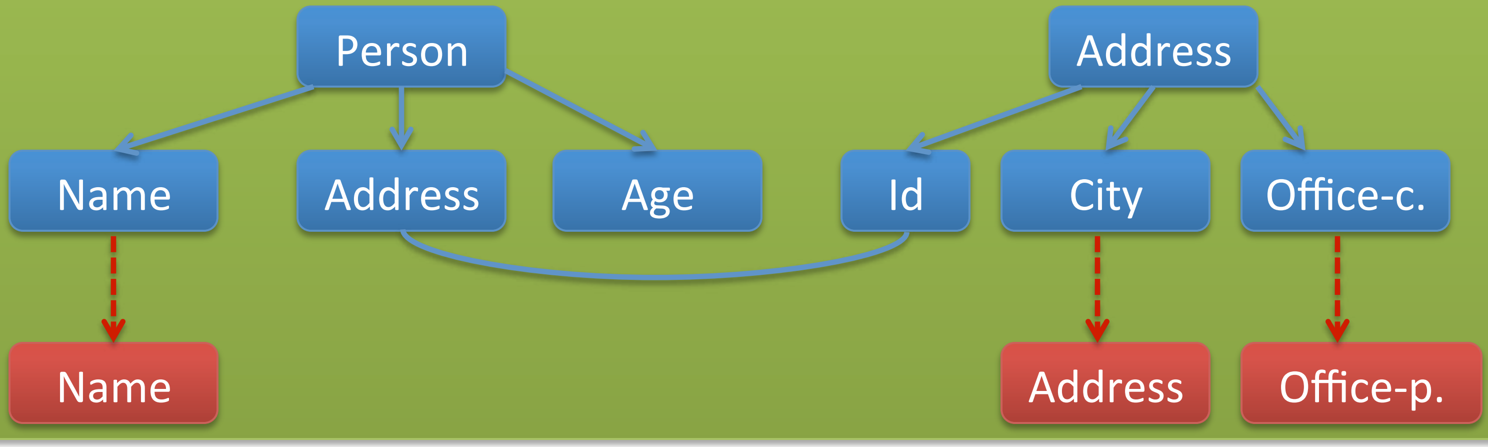
5.2 Skolem Functions

Example: Skolem Functions



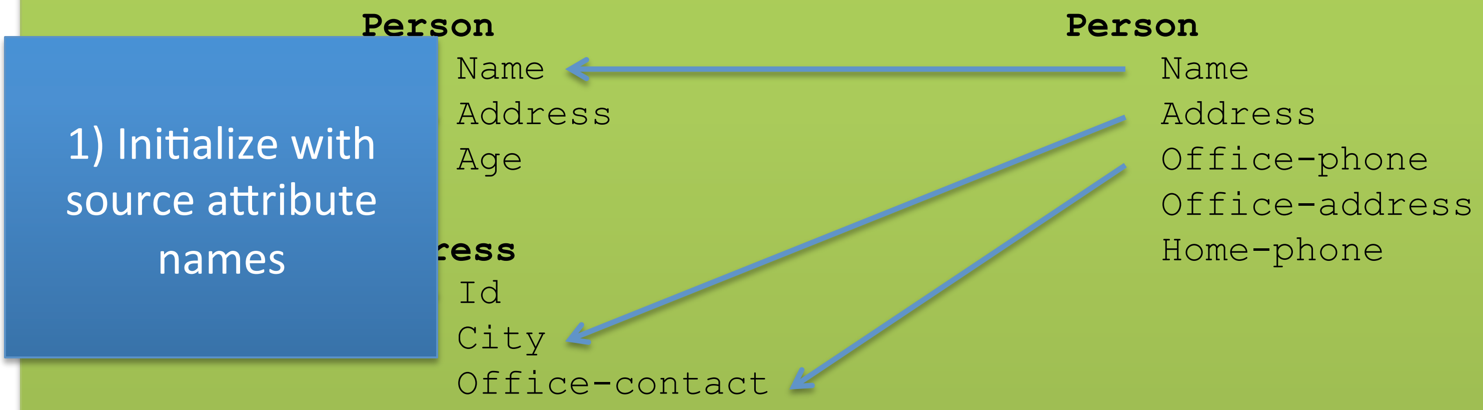
```
Person                              Person
    Name                                Name
    Address                             Address
    Age                                 Office-phone
                                        Office-address
Address                                 Home-phone
    Id
    City
    Office-contact
```

$$\forall a, b, c, d, e : Person(a, b, c, d, e) \rightarrow \exists f, g Person(a, f, g) \wedge Address(f, b, c)$$



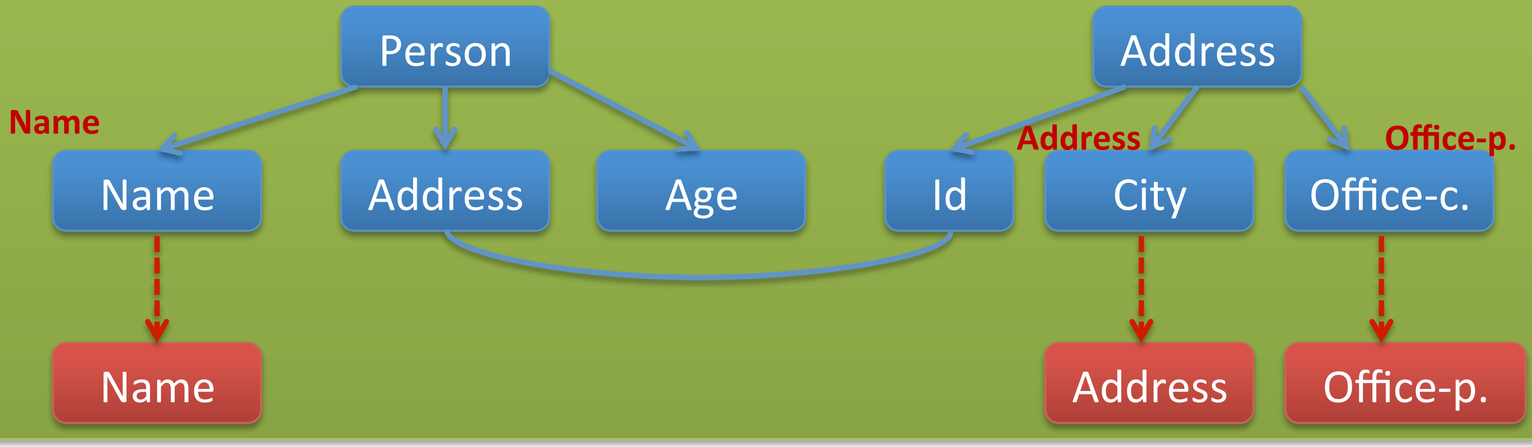
5.2 Skolem Functions

Example: Skolem Functions



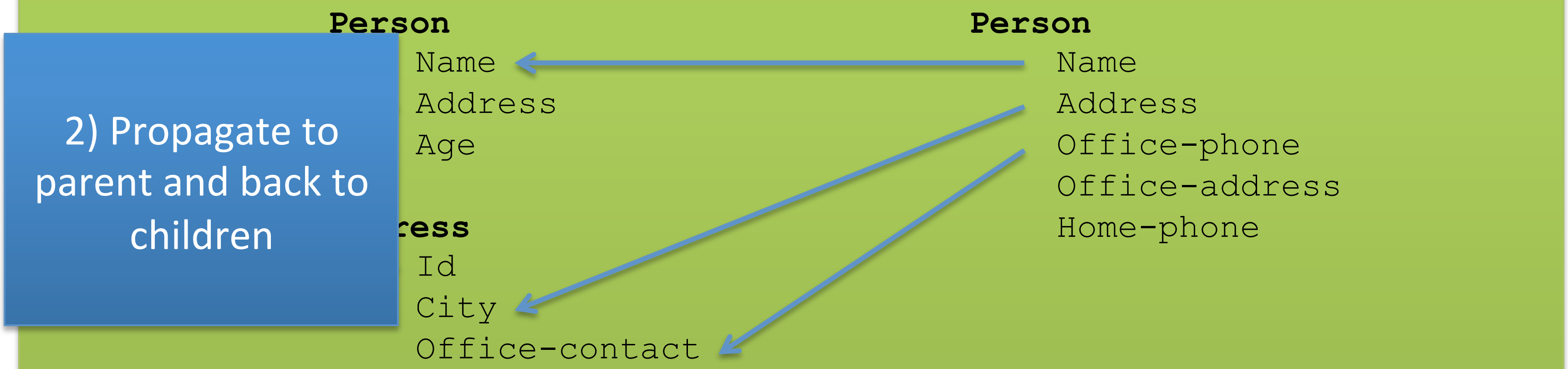
1) Initialize with source attribute names

$$\forall a, b, c, d, e : Person(a, b, c, d, e) \rightarrow \exists f, g Person(a, f, g) \wedge Address(f, b, c)$$



5.2 Skolem Functions

Example: Skolem Functions



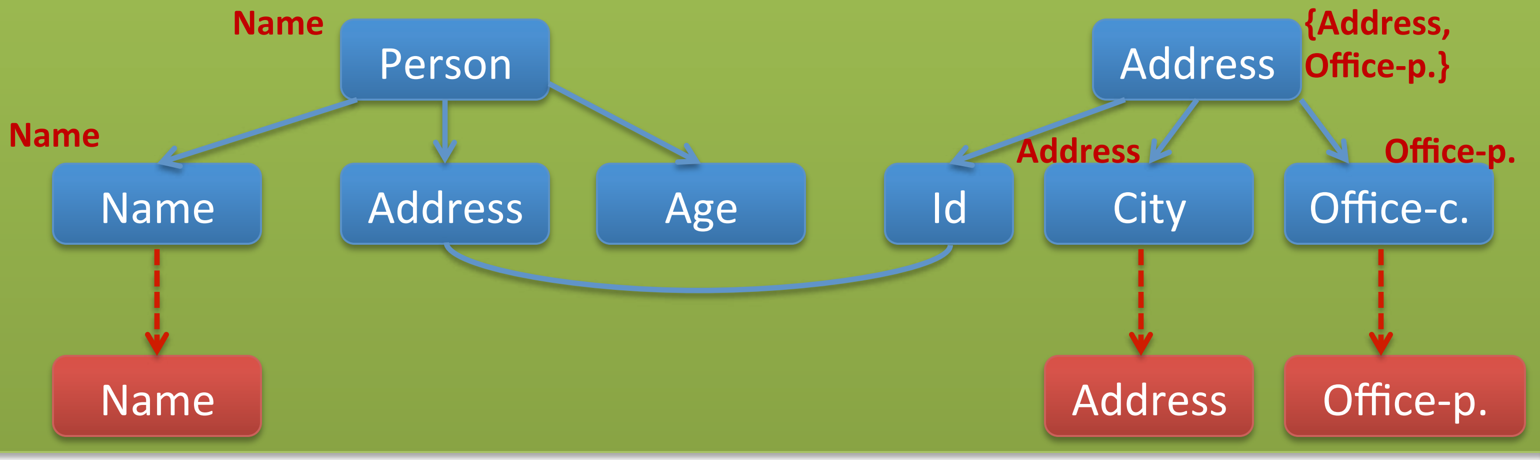
2) Propagate to parent and back to children

Person
Name
Address
Age

ress
Id
City
Office-contact

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

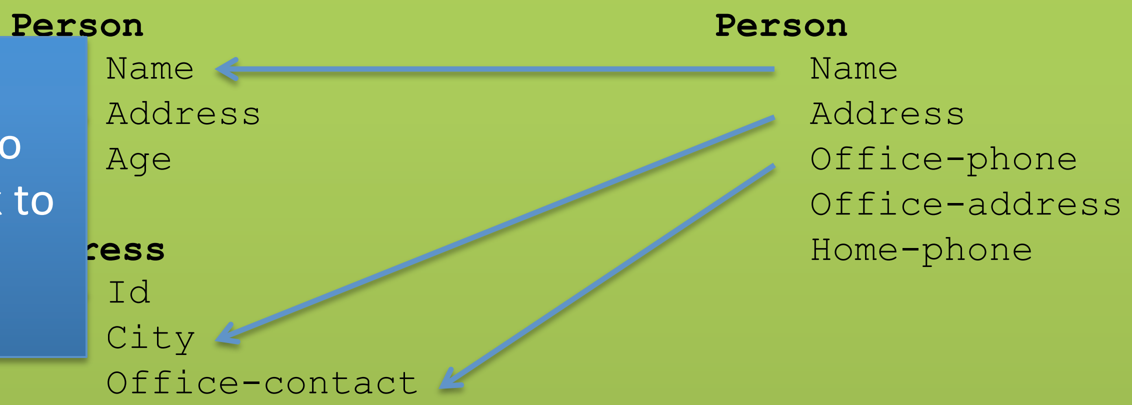
$$\forall a, b, c, d, e : Person(a, b, c, d, e) \rightarrow \exists f, g Person(a, f, g) \wedge Address(f, b, c)$$



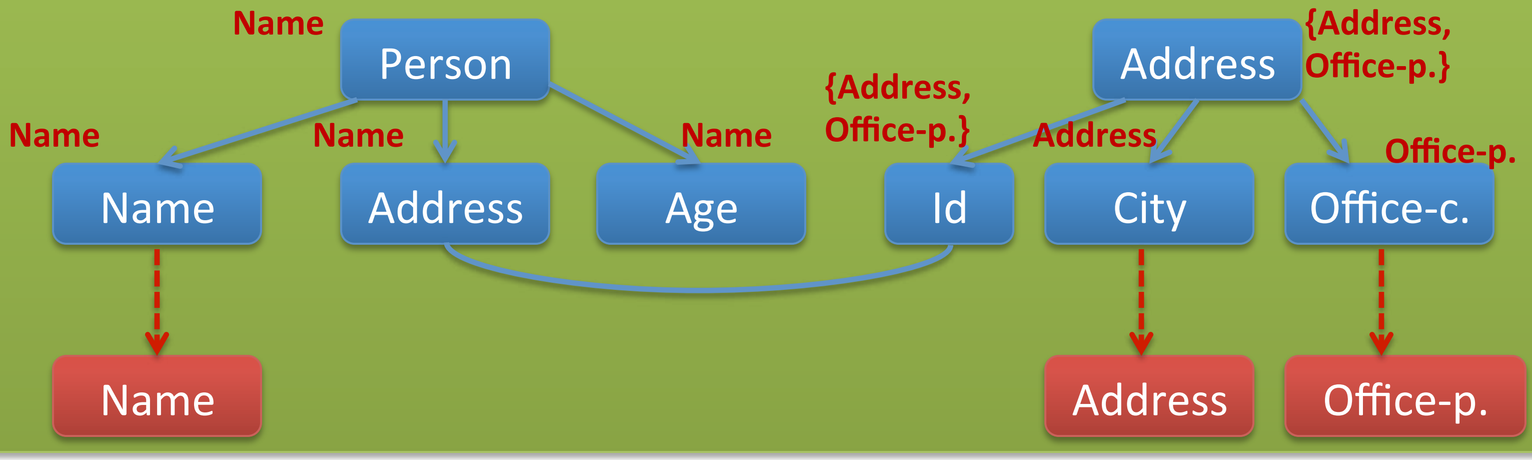
5.2 Skolem Functions

Example: Skolem Functions

2) Propagate to parent and back to children

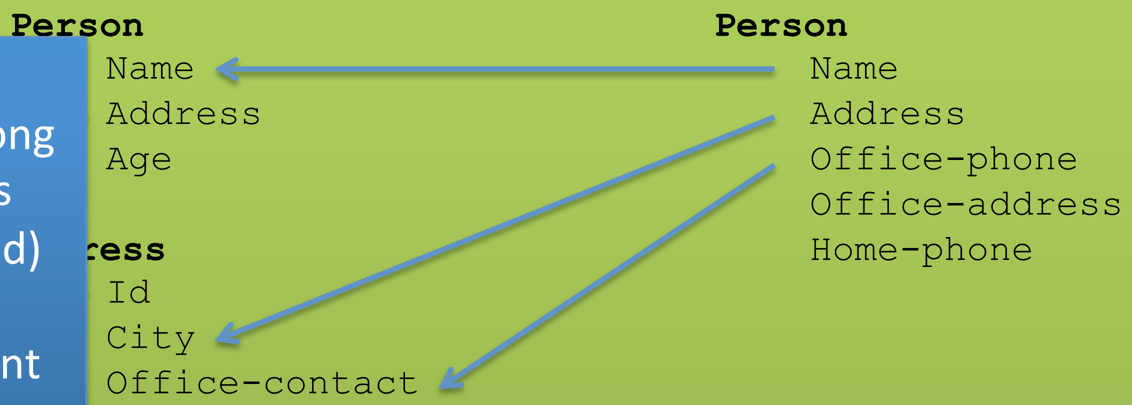


$$\forall a, b, c, d, e : Person(a, b, c, d, e) \rightarrow \exists f, g Person(a, f, g) \wedge Address(f, b, c)$$



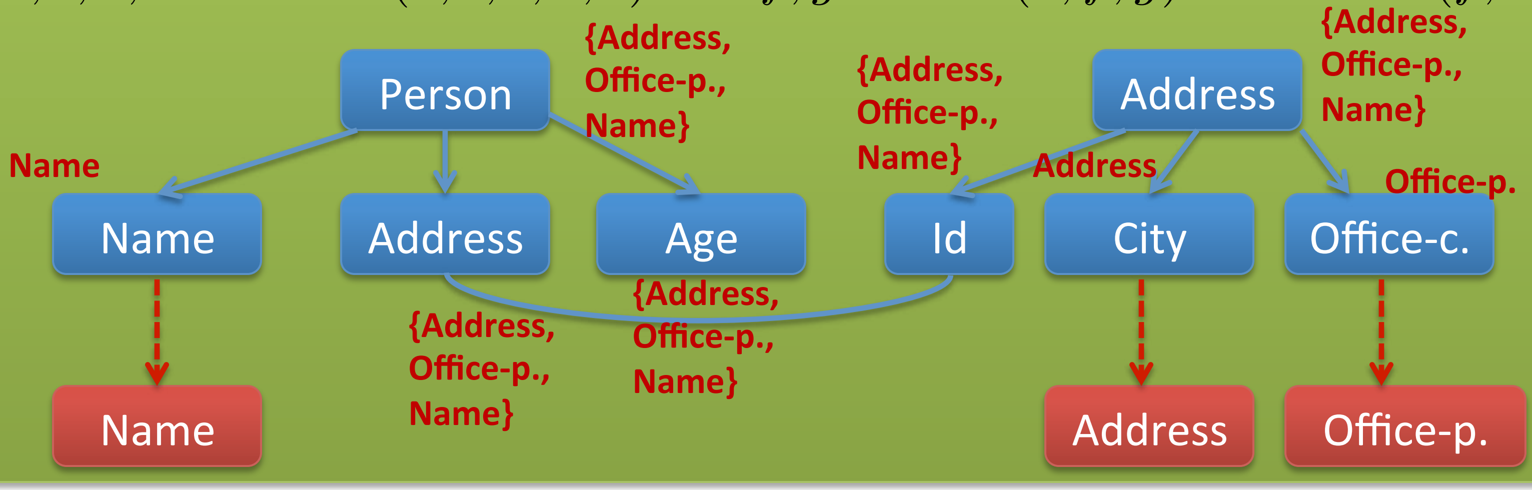
5.1 Data Exchange Solutions

Example: Skolem Functions



3) Propagate along equality edges (here address=id)
…
Compute fixpoint

$$\forall a, b, c, d, e : Person(a, b, c, d, e) \rightarrow \exists f, g Person(a, f, g) \wedge Address(f, b, c)$$


{Address, Office-p., Name}
{Address, Office-p., Name}
{Address, Office-p., Name}
Person
Address
Name
Address
Office-p.
Name
Address
Age
Id
City
Office-c.
{Address, Office-p., Name}
{Address, Office-p., Name}
Name
Address
Office-p.

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

$$\forall a, b, c, d, e : Person(a, b, c, d, e) \rightarrow \exists f, g Person(a, f, g) \wedge Address(f, b, c)$$

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

5.2 Executable Transformations

Example: Skolem Functions

$$\forall a, b, c, d, e : Person(a, b, c, d, e) \rightarrow \exists f, gPerson(a, f, g) \wedge Address(f, b, c)$$

For Person atom in RHS:

```
SELECT name,
        'SK1' || name || address || office-phone AS address,
        'SK2' || name || address || office-phone AS age
FROM Person
```

For Address atom in RHS:

```
SELECT  'SK1' || name || address || office-phone AS address,
         address AS city,
         office-phone AS office-contact
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

ILLINOIS INSTITUTE OF TECHNOLOGY

CS520
Data Integration, Warehousing, and Provenance

6. Data Warehousing

IIT DBGroup

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

6. What is Datawarehousing?

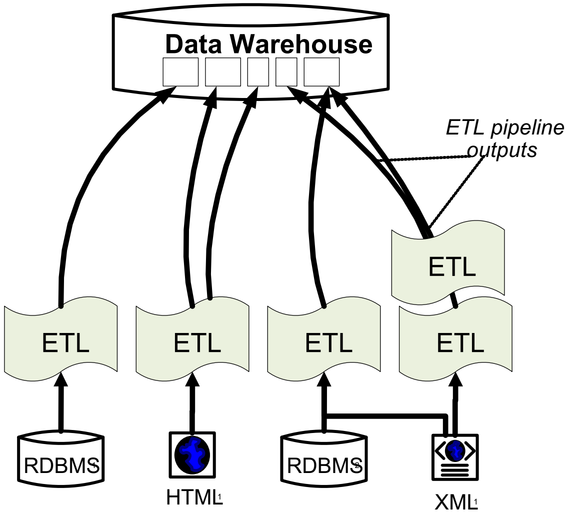
- **Problem: Data Analysis, Prediction, Mining**
  - **Example: Walmart**
  - Transactional databases
    - Run many “cheap” updates concurrently
    - E.g., each store has a database storing its stock and sales
  - Complex Analysis over Transactional Databases?
    - Want to analyze across several transactional 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

6. What is Datawarehousing?

- **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
  - …

6. Multidimensional Datamodel

- 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

		2014												2015				
		1. Quarter			2. Quarter			3. Quarter			4. Quarter			1. Quarter			2. Qu…	
		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May
Toy	car	3	7	6	37	7	92	37	7	92	37	7	92	37	7	92	2	…
	puppet	9	4	5	31	1	1	1	1	1	1	1	1	1	2	2	2	…
	Fishing rod	11	12	22	22	22	22	22	22	7	6	6	6	6	65	4	33	…
Books	Moby Dick	3	40	39	37	7	92	81	6	51	7	48	51	5	7	3	3	…
	Mobile devel.	3	2	5	43	7	0	81	6	51	7	48	51	5	7	3	3	…
	King Lear	3	9	6	37	7	92	5	6	51	7	48	51	5	7	3	3	…

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 datastructure that maps values in the dimensions to values for the m measures
  - **Schema**: D_1, …, D_n, M_1, …, M_m
  - **Instance**: a function

$\texttt{dom}(D_1) \times \ldots \times \texttt{dom}(D_n) \rightarrow \texttt{dom}(M_1) \times \ldots \times \texttt{dom}(M_m)$

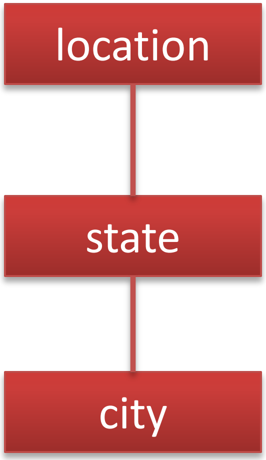
6. Dimensions

- **Purpose**
  - Selection of descriptive data
  - Grouping with desired level of granularity
- A dimension is define through a **containment-hierarchy**
- Hierarchies typically have several **levels**
- The **root level** represents the whole dimensions
- We may associate additional descriptive information with a elements in the hierarchy (e.g., number of residents in a city)

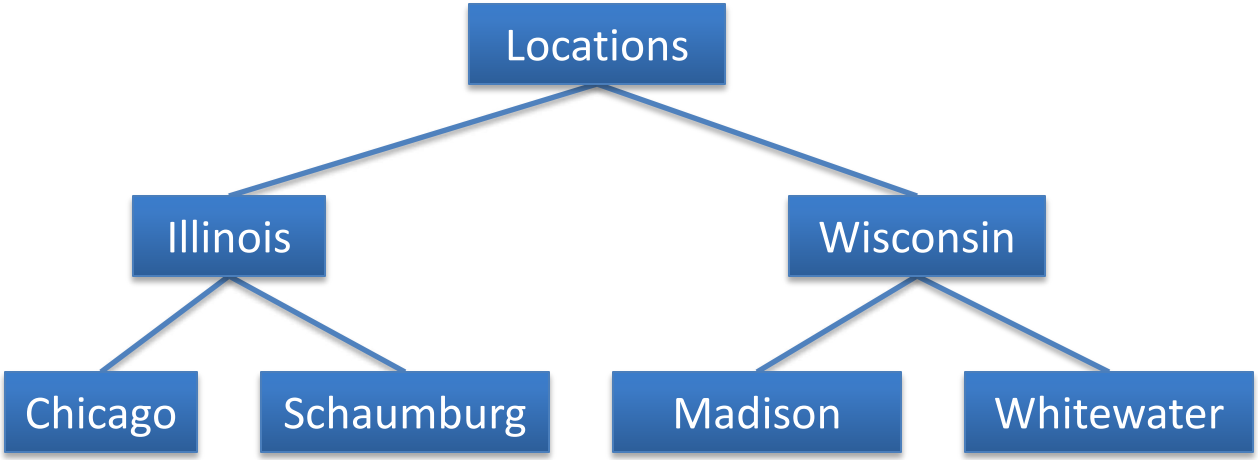
6. Dimension Example

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

Schema



Instance



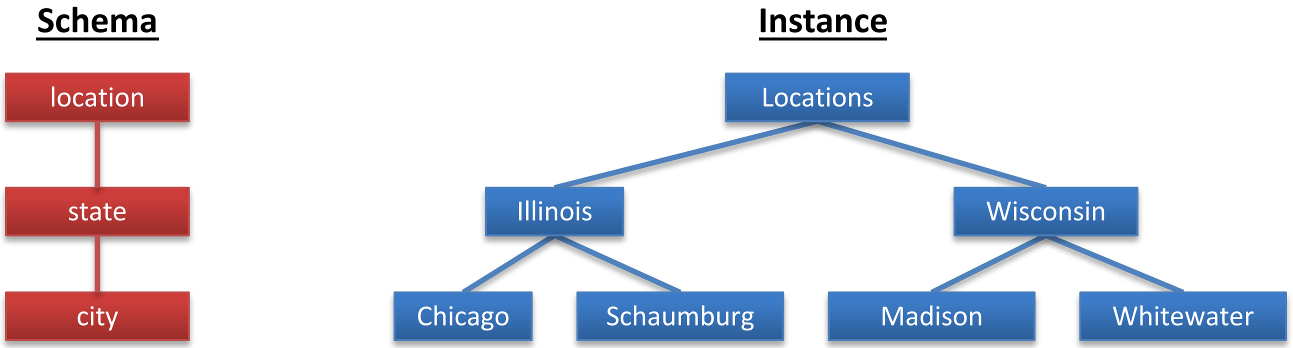
6. Dimension Schema

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

6. Dimension Schema Example

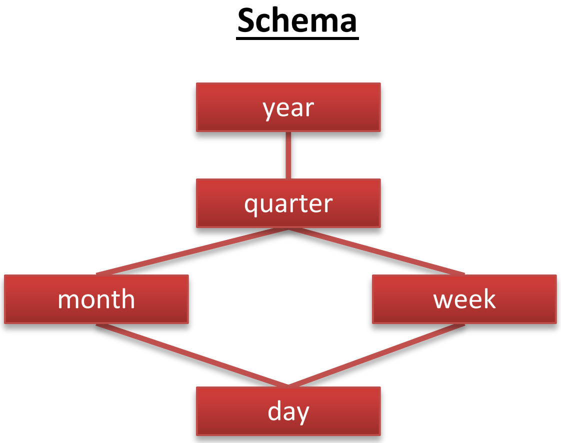
- **Schema of Location Dimension**
  - Set of categories D = {location, state, city}
  - Partial order

    { city → state, city → location, state → location }
  - Top_D = location
  - D_{min} = city



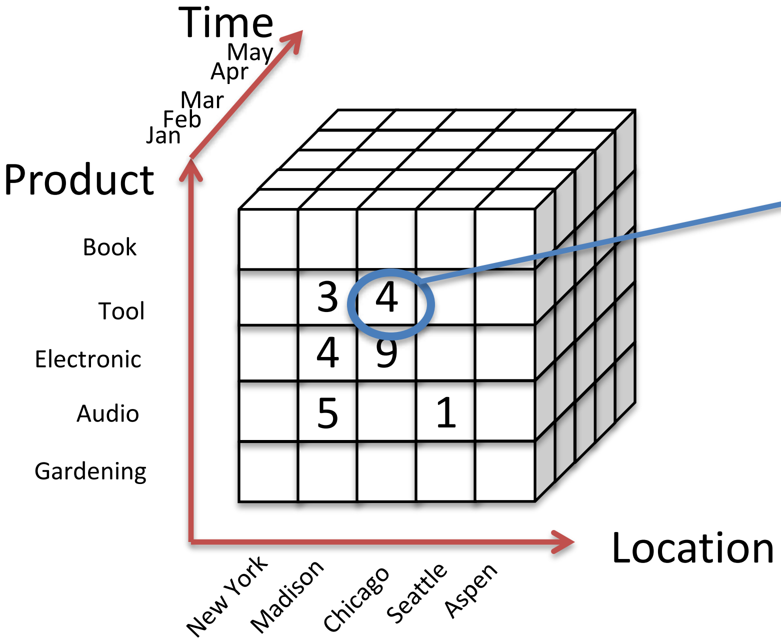
6. Remarks

- 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



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

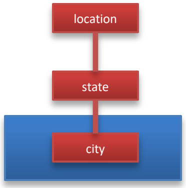


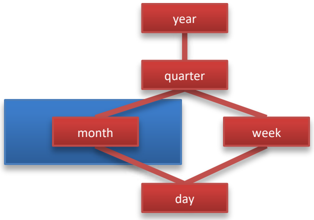
Fact:
Items in stock in **Jan** at **Chicago** that belong to category **Tool**

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 and location

    Revenue in **Illinois** during **Jan 2015**
- **Granularity:** Levels in the dimension hierarchy corresponding to the fact
  - E.g., city, 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., avg(revenue) = sum(revenue) / 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 granuarity**
  - 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?

Design Process Example

- **1) Selecting process(es)**
  - – sell coffee to customers
- **2) Select granularity**
  - – Sale of type of coffee per branch per day
  - – Sufficient for analysis
    - Save storage
- **3) Determine relevant dimensions**
  - – Location
  - – Time
  - – Product, …

Design Process Example

- **1) Selecting process(es)**
  - sell coffee to customers
- **2) Select granularity**
  - Sale of type of coffee per branch per day
- **3) Determine relevant dimensions**
  - Location (country, state, city, zip, shop)
  - Time (year, month, day)
  - Product (type, brand, product)

Design Process Example

- **1) Selecting process(es)**
  - sell coffee to customers
- **2) Select granularity**
  - Sale of type of coffee per branch per day
- **3) Determine relevant dimensions**
  - Location (country, state, city, zip, shop)
  - Time (year, month, day)
  - Product (type, brand, product)
- **4) Select measures**

Design Process Example

- **1) Selecting process(es)**
  - sell coffee to customers
- **2) Select granularity**
  - Sale of type of coffee per branch per day
- **3) Determine relevant dimensions**
  - Location (country, state, city, zip, shop)
  - Time (year, month, day)
  - Product (type, brand, product)
- **4) Select measures**
  - cost, revenue, profit?

Relational representation

- How to model a **datacube** using the **relational datamodel**
- We start from
  - Dimension schemas
  - Set of measures

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,…,D_k,Top_D) we create a relation
  - D ($\underline{PK}$, D_1,…,D_k)
  - Here PK is a primary key, e.g., D_{min}
- Fact table
  - F($\underline{FK_1}$, …, $\underline{FK_n}$, M_1, ..., M_m)
  - Each $\mathbf{FK_i}$ is a foreign key to $\mathbf{D_i}$
  - 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
- 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,…,D_k,Top_D) we create a relation multiple relations connected through FKs
  - – D_i ($\underline{PK}$, A_1, …, A_l, FK_j)
  - – A_l is a descriptive attribute
  - – FKj is foreign key to the immediate parent(s) of D_i
- Fact table
  - – F($\underline{FK_1}$, …, $\underline{FK_n}$, M_1, ..., M_m)
  - – Each **FK**$_{\mathbf{i}}$ is a foreign key to **D**$_{\mathbf{i}}$
  - – 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 **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

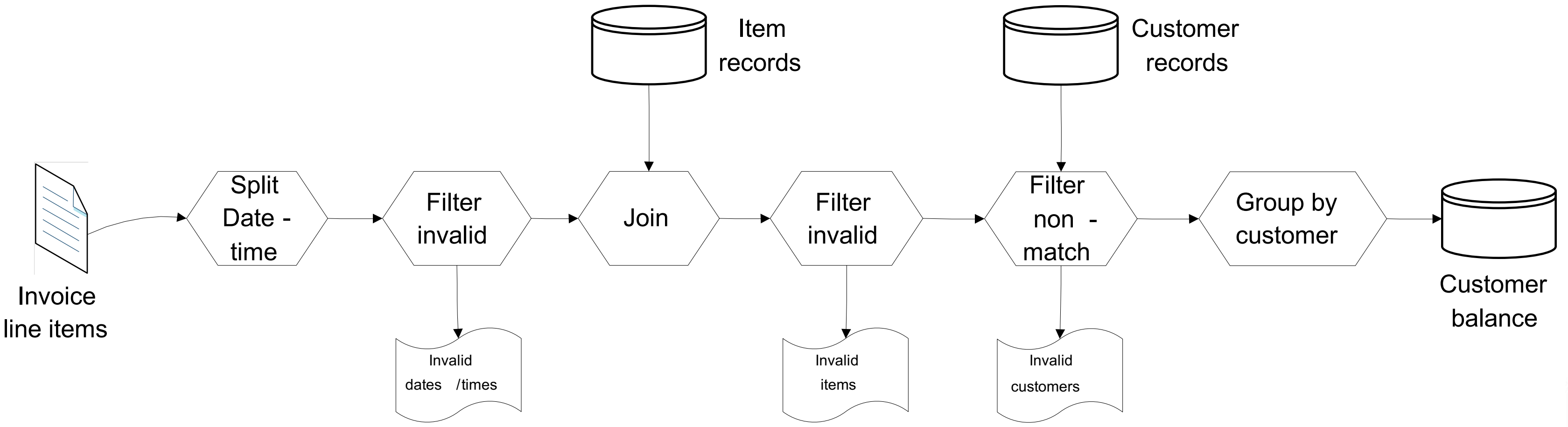
6. Extract-Transform-Load (ETL)

- **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
  - …

- **Operators supported by ETL**
  - Many of the preprocessing and cleaning operators we already know
    - **Surrogate key generation** (like creating existentials with skolems)
    - **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”**
  - **date: “12.12.2015” -> year: 2002, month: 12, day :12**
  - **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”, …**

6. Typical ETL operators

- **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. Typical ETL operators

- **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)**

6. Cube operations

- **Drill-out**
  - Add additional dimensions
    - special case of drill-down starting from Top_D in dimension(s)
    - **E.g., sales per (city, product category) to Sales per (city,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. Cube operations

- **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**

6. SQL Extensions

- Recall that grouping on multiple sets of attributes is hard to express in SQL
  - E.g., give me the total sales, the sales per year, and the sales per month
    - Practice

6. SQL Extensions

- Syntactic Sugar for multiple grouping
  - **GROUPING SETS**
  - **CUBE**
  - **ROLLUP**
- These constructs are allowed as expressions in the GROUP BY clause

6. 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$_i$
  - 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))
```

quarter	city	product_typ	profit
2010 Q1		Books	8347
2012 Q2		Books	7836
2012 Q2		Gardening	12300
2012 Q2	Chicago		12344
2012 Q2	Seattle		124345

6. GROUPING SETS

```
SELECT quarter, city, NULL AS 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 quarter, city
UNION
SELECT quarter, NULL AS 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 quarter, product_type
```

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)
```

quarter	city	product_typ	profit
2010 Q1			8347
2012 Q2			7836
2012 Q2			12300
2012 Q2	Chicago		12344
2012 Q2	Seattle		124345
2012 Q2	Seattle	Gardening	12343

Did not group on **product_typ** or this is the group for all NULL values in **product_typ**?

6. GROUPING SETS

- Solution:
  - GROUPING predicate
  - GOUPING(A) = 1 if grouped on attribute A, 0 else

```
SELECT … GROUPING(product_typ) AS grp_prd
…
GROUP BY GROUPING SETS
    ( (quarter, city), (quarter, product_typ), (quarter, product_typ, city)
```

quarter	city	product_typ	profit	grp_prd
2010 Q1		Books	8347	1
2012 Q2		Books	7836	1
2		Gardening	12300	1
2			12344	0
2012 Q2	Seattle		124345	1
2012 Q2	Seattle	Gardening	12343	1

Now it's clear!

6. GROUPING SETS

- Combining GROUPING SETS

```
GROUP BY A, B
= GROUP BY GROUPING SETS ((A,B))

GROUP BY GROUPING SETS ((A,B), (A,C), (A))
= GROUP BY A, GROUPING SETS ((B), (C), ())

GROUP BY GROUPING SETS ((A,B), (B,C),
         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` **`CUBE (set)`**
- Group by all 2^n subsets of **`set`**

```
GROUP BY CUBE (A,B,C)
= GROUP BY GROUPING SETS (
    (),
    (A), (B), (C),
    (A,B), (A,C), (B,C),
    (A,B,C)
  )
```

6. CUBE

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

```
GROUP BY ROLLUP (A,B,C)
= 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 too
  - 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

```
SELECT shop, sum(profit) OVER()
```

- aggregation over full table

```
SELECT shop, sum(profit) OVER(PARTITION BY state)
```

- like group-by

```
SELECT shop, sum(profit) OVER(ORDER BY month)
```

- rolling sum including everything with smaller month

```
SELECT shop, sum(profit) OVER(ORDER BY month 6
PRECEDING 3 FOLLOWING)
```

6. OVER clause

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

```
<window frame preceding> ::= {
   UNBOUNDED PRECEDING
       | n PRECEDING
       | CURRENT ROW }

<window frame following> ::= {
       UNBOUNDED FOLLOWING
         | n FOLLOWING
     | CURRENT ROW
}
```

6. OVER clause

ILLINOIS INSTITUTE
OF TECHNOLOGY

```
SELECT year, month, city, profit
       SUM(profit) OVER () AS ttl
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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	92
2010	2	Chicago	5	92
2010	3	Chicago	20	92
2011	1	Chicago	45	92
2010	1	New York	12	92

6. OVER clause

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

- **PARITION BY**
  - only tuples with same partition-by attributes belong to the same window
- Like **GROUP BY**

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	47
2010	2	Chicago	5	47
2010	3	Chicago	20	47
2011	1	Chicago	45	45
2010	1	New York	12	47

6. OVER clause

```
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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	22
2010	2	Chicago	5	27
2010	3	Chicago	20	47
2011	1	Chicago	45	92
2010	1	New York	12	22

6. OVER clause

```
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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	22
2010	2	Chicago	5	27
2010	3	Chicago	20	47
2011	1	Chicago	45	45
2010	1	New York	12	22

6. OVER clause

```
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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	22
2010	2	Chicago	5	27
2010	3	Chicago	20	47
2011	1	Chicago	45	45
2010	1	New York	12	22

6. OVER clause

```
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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	22
2010	2	Chicago	5	27
2010	3	Chicago	20	47
2011	1	Chicago	45	92
2010	1	New York	12	22

6. OVER clause

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

- Combining **PARTITION BY** and **ORDER BY**
  - First partition, then order tuples within each partition

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	22
2010	2	Chicago	5	27
2010	3	Chicago	20	47
2011	1	Chicago	45	45
2010	1	New York	12	22

6. OVER clause

```
SELECT year, month, city
      SUM(profit) OVER (PARTITION BY year ORDER BY 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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	27
2010	2	Chicago	5	47
2010	3	Chicago	20	25
2011	1	Chicago	45	45
2010	1	New York	12	27

6. OVER clause

```
SELECT year, month, city
      SUM(profit) OVER (ORDER BY year, month
                        ROWS 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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	22
2010	2	Chicago	5	37
2010	3	Chicago	20	70
2011	1	Chicago	45	65
2010	1	New York	12	27

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-spec₁>, …
FROM <cube-spec₁>, …
WHERE ( <select-spec> )
```

- Note!
  - Semantics of SELECT, FROM, WHERE not what you would expect knowing SQL

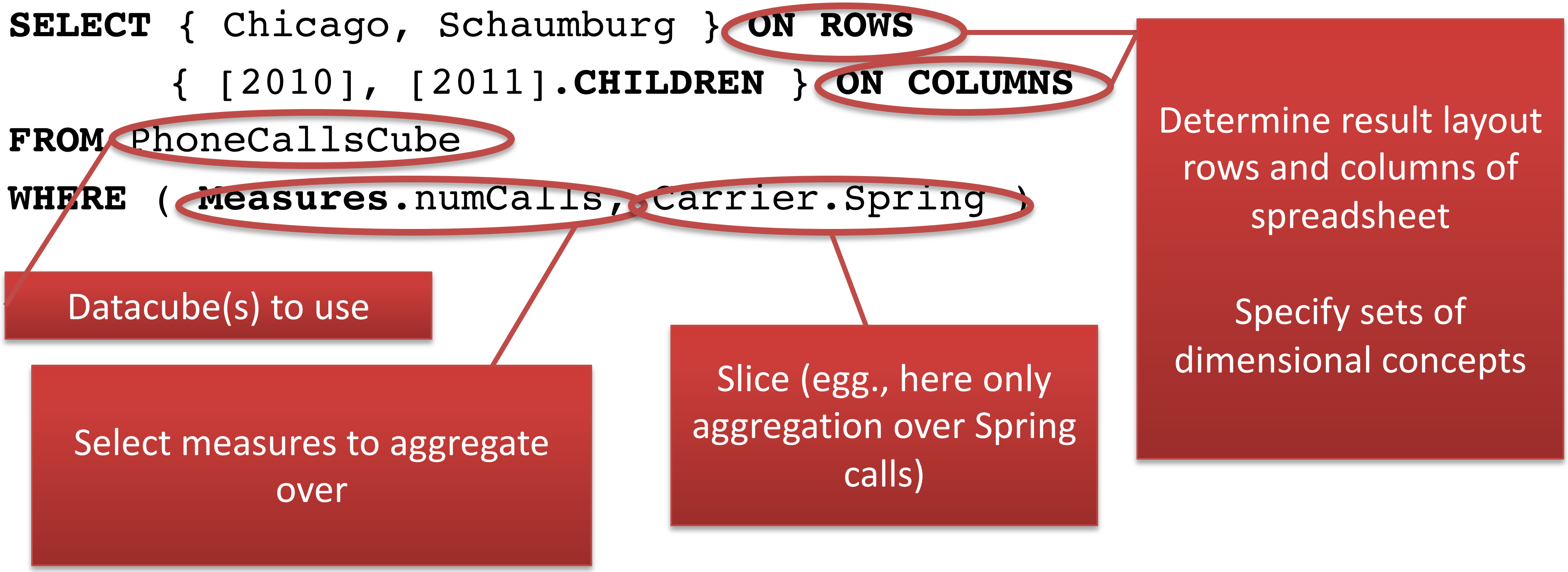
6. MXD

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

- Meaning of
  - [] interpret number as name
  - {} set notation
  - () tuple in where clause

	2010	2011 Jan	2011 Feb	2011 Mar	…	2011 Dec
Chicago	23423	5425234523	432	43243434	…	12231
Schaumburg	32132	12315	213333	123213	….	123153425

6. MXD



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

Determine result layout rows and columns of spreadsheet

Specify sets of dimensional concepts

Datacube(s) to use

Select measures to aggregate over

Slice (egg., here only aggregation over Spring calls)

	2010	2011 Jan	2011 Feb	2011 Mar	…	2011 Dec
Chicago	23423	5425234523	432	43243434	…	12231
Schaumburg	32132	12315	213333	123213	….	123153425

6. MXD - SELECT

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

- Select specifies dimensions in result and how to visualize
  - **ON COLUMNS, ON ROWS, ON PAGES, ON SECTIONS, ON CHAPTERS**
- Every dimension in result corresponds to one dimension in the cube
  - Set of concepts from this dimensions which may be from different levels of granularity
  - **E.g., {2010, 2011 Jan, 2012 Jan, 2012 Feb, 2010 Jan 1st}**

	2010	2011 Jan	2011 Feb	2011 Mar	…	2011 Dec
Chicago	23423	5425234523	432	43243434	…	12231
Schaumburg	32132	12315	213333	123213	….	123153425

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
    - E.g., `[1990]:[1993] = {[1990], [1991], [1992], [1993]}`

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
    - E.g., `[1990]:[1993] = {[1990], [1991], [1992], [1993]}`

6. MXD - SELECT

- 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 )
```

Chicago	2010	123411
	2011	3231
Schaumburg	2010	32321132
	2011	12355

6. MXD - SELECT

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

- **Show results for all month of 2010 where there are more Sprint calls than ATT calls**

```
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

6. Partitioning

- **Vertical Partitioning**
  - Fragments F_1 to Fn of relation R such that
    - Sch(F_1) u Sch(F_2) u … u Sch(F_n) = Sch(R)
    - Store row id or PK of R with every fragment
    - Restore relation R through natural joins

Name	Salary	Age	Gender
Peter	12,000	45	M
Alice	24,000	34	F
Bob	20,000	22	M
Gertrud	50,000	55	F
Pferdegert	14,000	23	M

Rowid	Name	Salary
1	Peter	12,000
2	Alice	24,000
3	Bob	20,000
4	Gertrud	50,000
5	Pferdegert	14,000

Rowid	Age	Gender
1	45	M
2	34	F
3	22	M
4	55	F
5	23	M

6. Partitioning

- **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 Fn of relation R such that
    - $Sch(F_1) = Sch(F_2) = \dots = Sch(F_n) = Sch(R)$
    - $R = F_1 \text{ u } \dots \text{ u } F_n$

Name	Salary	Age	Gender
Peter	12,000	45	M
Alice	24,000	34	F
Bob	20,000	22	M
Gertrud	50,000	55	F
Pferdegert	14,000	23	M

Name	Salary	Age	Gender
Peter	12,000	45	M
Pferdegert	14,000	23	M

Salary
[0,15000]

Name	Salary	Age	Gender
Alice	24,000	34	F
Bob	20,000	22	M
Gertrud	50,000	55	F

Salary
[15001,100000]

6. Partitioning

- **Horizontal Partitioning**
  - **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**
    - $Sch(F_1) = Sch(F_2) = … = Sch(F_n) = Sch(R)$
    - $R = F_1$ u … u F_n

Name	Salary	Age	Gender
Peter	12,000	45	M
Alice	24,000	34	F
Bob	20,000	22	M
Gertrud	50,000	55	F
Pferdegert	14,000	23	M

Name	Salary	Age	Gender
Alice	24,000	34	F
Pferdegert	14,000	23	M

Salary
h(24,000) = 0
H(14,000) = 0

Name	Salary	Age	Gender
Peter	12,000	45	M
Bob	20,000	22	M
Gertrud	50,000	55	F

Salary
h(12,000) = 1
H(20,000) = 1
H(50,000) = 1

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

ILLINOIS INSTITUTE
OF TECHNOLOGY

CS520
Data Integration, Warehousing, and Provenance

7. Big Data Systems and Integration

IIT DBGroup

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

3. Big Data Analytics

- **Big Topic, big Buzzwords ;-)**
- Here
  - **Overview of two types of systems**
    - Key-value/document stores
    - **Mainly**: Bulk processing (MR, graph, …)
  - **What is new compared to single node systems?**
  - **How do these systems change our approach to integration/analytics**
    - Schema first vs. Schema later
    - **Pay-as-you-go**

3. Big Data Overview

- **1) How does data processing at scale (read using many machines) differ from what we had before?**
  - Load-balancing
  - Fault tolerance
  - Communication
  - New abstractions
    - Distributed file systems/storage

3. Big Data Overview

- **2) Overview of systems and how they achieve scalability**
  - **Bulk processing**
    - MapReduce, Shark, Flink, Hyracks, …
    - Graph: e.g., Giraph, Pregel, …
  - **Key-value/document stores = NoSQL**
    - Cassandra, MongoDB, Memcached, Dynamo, …

3. Big Data Overview

- **2) Overview of systems and how they achieve scalability**
  - **Bulk processing**
    - MapReduce, Shark, Flink,
  - **Fault tolerance**
    - Replication
    - Handling stragglers
  - **Load balancing**
    - Partitioning
    - Shuffle

3. Big Data Overview

- **3) New approach towards integration**
  - **Large clusters enable directly running queries over semi-structured data (within feasible time)**
    - Take a click-stream log and run a query
  - One of the reasons why **pay-as-you-go** is now feasible
    - **Previously:** designing a database schema upfront and designing a process (e.g., ETL) for cleaning and transforming data to match this schema, then query
    - **Now:** start analysis directly, clean and transform data if needed for the analysis

3. Big Data Overview

- **3) New approach towards integration**
  - **Advantage of pay-as-you-go**
    - More timely data (direct access)
    - More applicable if characteristics of data change dramatically (e.g., yesterdays ETL process no longer applicable)
  - **Disadvantages of pay-as-you-go**
    - Potentially repeated efforts (everybody cleans the click-log before running the analysis)
    - Lack of meta-data may make it hard to
      - Determine what data to use for analysis
      - Hard to understand semantics of data

3. Big Data Overview

- **Scalable systems**
  - Performance of the system scales in the number of nodes
    - Ideally the per node performance is constant independent of how many nodes there are in the system
    - This means: having twice the number of nodes would give us twice the performance
  - Why scaling is important?
    - If a system scales well we can "throw" more resources at it to improve performance and this is cost effective

3. Big Data Overview

- **What impacts scaling?**
  - Basically how parallelizable is my algorithm
    - **Positive example**: problem can be divided into subproblems that can be solved independently without requiring communication
      - E.g., array of 1-billion integers [i_1, …, $i_{1,000,000,000}$] add 3 to each integer. Compute on n nodes, split input into n equally sized chunks and let each node process one chunk
    - **Negative example**: problem where subproblems are strongly intercorrelated
      - E.g., Context Free Grammar Membership: given a string and a context free grammar, does the string belong to the language defined by the grammar.

3. Big Data – Processing at Scale

- **New problems at scale**
  - DBMS
    - running on 1 or 10's of machines
    - running on 1000's of machines
- Each machine has low probability of failure
  - If you have many machines, failures are the norm
  - Need mechanisms for the system to cope with failures
    - Do not loose data
    - Do not use progress of computation when node fails
  - This is called **fault-tolerance**

3. Big Data – Processing at Scale

- **New problems at scale**
  - DBMS
    - running on 1 or 10's of machines
    - running on 1000's of machines
- Each machine has limited storage and computational capabilities
  - Need to **evenly** distribute data and computation across nodes
    - Often most overloaded node determine processing speed
  - This is called **load-balancing**

3. Big Data – Processing at Scale

- **Building distributed systems is hard**
  - Many pitfalls
    - Maintaining distributed state
    - Fault tolerance
    - Load balancing
  - Requires a lot of background in
    - OS
    - Networking
    - Algorithm design
    - Parallel programming

3. Big Data – Processing at Scale

- **Building distributed systems is hard**
  - Hard to debug
    - Even debugging a parallel program on a single machine is already hard
      - Non-determinism because of scheduling: Race conditions
      - In general hard to reason over behavior of parallel threads of execution
    - Even harder when across machines
  - Just think about how hard it was for you to first program with threads/processes

3. Big Data – Why large scale?

- **Datasets are too large**
  - **Storing a 1 Petabyte dataset requires 1 PB storage**
    - **Not possible on single machine even with RAID storage**
- **Processing power/bandwidth of single machine is not sufficient**
  - **Run a query over the facebook social network graph**
    - **Only possible within feasible time if distributed across many nodes**

3. Big Data – User's Point of View

- **How to improve the efficiency of distributed systems experts**
  - Building a distributed system from scratch for every store and analysis task is obviously not feasible!
- **How to support analysis over large datasets for non distributed systems experts**
  - How to enable somebody with some programming but limited/no distributed systems background to run distributed computations

3. Big Data – Abstractions

- **Solution**
  - Provide higher level abstractions
- **Examples**
  - **MPI** (message passing interface)
    - Widely applied in HPC
    - Still quite low-level
  - **Distributed file systems**
    - Make distribution of storage transparent
  - **Key-value storage**
    - Distributed store/retrieval of data by identifier (key)

3. Big Data – Abstractions

- **More Examples**
  - **Distributed table storage**
    - Store relations, but no SQL interface
  - **Distributed programming frameworks**
    - Provide a, typically, limited programming model with automated distribution
  - **Distributed databases, scripting languages**
    - Provide a high-level language, e.g., SQL-like with an execution engine that is distributed

3. Distributed File Systems

- **Transparent distribution of storage**
  - Fault tolerance
  - Load balancing?
- **Examples**
  - **HPC distributed filesystems**
    - Typically assume a limited number of dedicated storage servers
    - GPFS, Lustre, PVFS
  - **"Big Data" filesystems**
    - Google file system, HDFS

3. HDFS

- **Hadoop Distributed Filesystem (HDFS)**
- Architecture
  - – One nodes storing metadata (name node)
  - – Many nodes storing file content (data nodes)
- Filestructure
  - – Files consist of blocks (e.g., 64MB size)
- Limitations
  - – Files are append only

3. HDFS

- **Name node**
- Stores the directory structure
- Stores which blocks belong to which files
- Stores which nodes store copies of which block
- Detects when data nodes are down
  - Heartbeat mechanism

- Clients communicate with the name node to gather FS metadata

3. HDFS

- **Data nodes**
- Store blocks
- Send/receive file data from clients
- Send heart-beat messages to name node to indicate that they are still alive

- Clients communicate with data nodes for reading/writing files

3. HDFS

- **Fault tolerance**
  - n-way replication
  - Name node detects failed nodes based on heart-beats
  - If a node if down, then the name node schedules additional copies of the blocks stored by this node to be copied from nodes storing the remaining copies

3. Distributed FS Discussion

- **What do we get?**
  - Can store files that do not fit onto single nodes
  - Get fault tolerance
  - Improved read speed (caused by replication)
  - Decreased write speed (caused by replication)
- **What is missing?**
  - Computations
  - Locality (horizontal partitioning)
  - Updates
- **What is not working properly?**
  - Large number of files (name nodes would be overloaded)

3. Frameworks for Distributed Computations

- **Problems**
  - Not all algorithms do parallelize well
  - How to simplify distributed programming?
- **Solution**
  - Fix a reasonable powerful, but simple enough model of computation for which scalable algorithms are known
  - Implement distributed execution engine for this model and make it fault tolerant and load-balanced

3. MapReduce

- **Data Model**
  - Sets of key-value pairs $\{(k_1,v_1), \ldots, (k_n,v_n)\}$
  - **Key** is an identifier for a piece data
  - **Value** is the data associaed with a key
- **Programming Model**
  - We have two higher-level functions **map** and **reduce**
    - Take as input a user-defined function that is applied to elements in the input key-value pair set
  - Complex computations can be achieved by chaining map-reduce computations

3. MapReduce Datamodel

- **Data Model**
  - Sets of key-value pairs $\{(k_1,v_1), \ldots, (k_n,v_n)\}$
  - **Key** is an identifier for a piece data
  - **Value** is the data associaed with a key
- **Examples**
  - Document **d** with an **id**
    - (id, d)
  - Person with name, salary, and SSN
    - (SSN, “name, salary”)

3. MapReduce Computational Model

- **Map**
  - Takes as input a set of key-value pairs and a user-defined function `f:(k,v) -> {(k,v)}`
  - Map applies f to every input key-value pair and returns the union of the outputs produced by f

$$\{(k_1,v_1),\ldots,(k_n,v_n)\}$$
$$\rightarrow$$
$$f((k_1,v_1)) \cup \ldots \cup f((k_n,v_n))$$

3. MapReduce Computational Model

- **Example**
  - **Input**: Set of (city,population) pairs
  - **Task**: multiply population by 1.05
- **Map function**
  -
    ```
    f: (city,population) ->
    {(city,population*1.05)}
    ```
- Application of f through map
  - Input: `{(chicago, 3), (nashville, 1)}`
  - Output:
    ```
    {(chicago, 3.15)} ∪ {(nashville, 1.05)}
             = {(chicago, 3.15), (nashville, 1.05)}
    ```

3. MapReduce Computational Model

- **Reduce**
  - Takes as input a key with a list of associated values and a user-defined function

    `g: (k,list(v)) -> {(k,v)}`
  - Reduce groups all values with the same key in the input key-value set and passes each key and its list of values to **g** and returns the union of the outputs produced by **g**

$\{(k_1,v_{11}),\ldots,(k_1,v_{1n1}),\ \ldots\ (k_m,v_{m1}),\ldots,(k_m,v_{mnm})\}$

$->$

$g((k_1,(v_{11},\ldots,v_{1n1}))\ \cup\ \ldots\ \cup\ g((k_m,(v_{m1},\ldots,v_{mnm}))$

3. MapReduce Computational Model

- **Example**
  - **Input**: Set of (state, population) pairs one for each city in the state
  - **Task**: compute the total population per state
- **Reduce function**
  - $\texttt{g: (state,[}p_1\texttt{, …, }p_n\texttt{]) ->}$ $\texttt{\{(state,SUM([}p_1\texttt{,…,}p_n\texttt{]))\}}$
- Application of g through reduce
  - Input: `{(illinois, 3), (illinois, 1), (oregon, 15)}`
  - Output: `{(illinois, 4), (oregon, 15)}`

3. MapReduce Workflows

- **Workflows**
  - Computations in MapReduce consists of map phases followed by reduce phases
    - The input to the reduce phase is the output of the map phase
  - Complex computations may require multiple map-reduce phases to be chained together

3. MapReduce Implementations

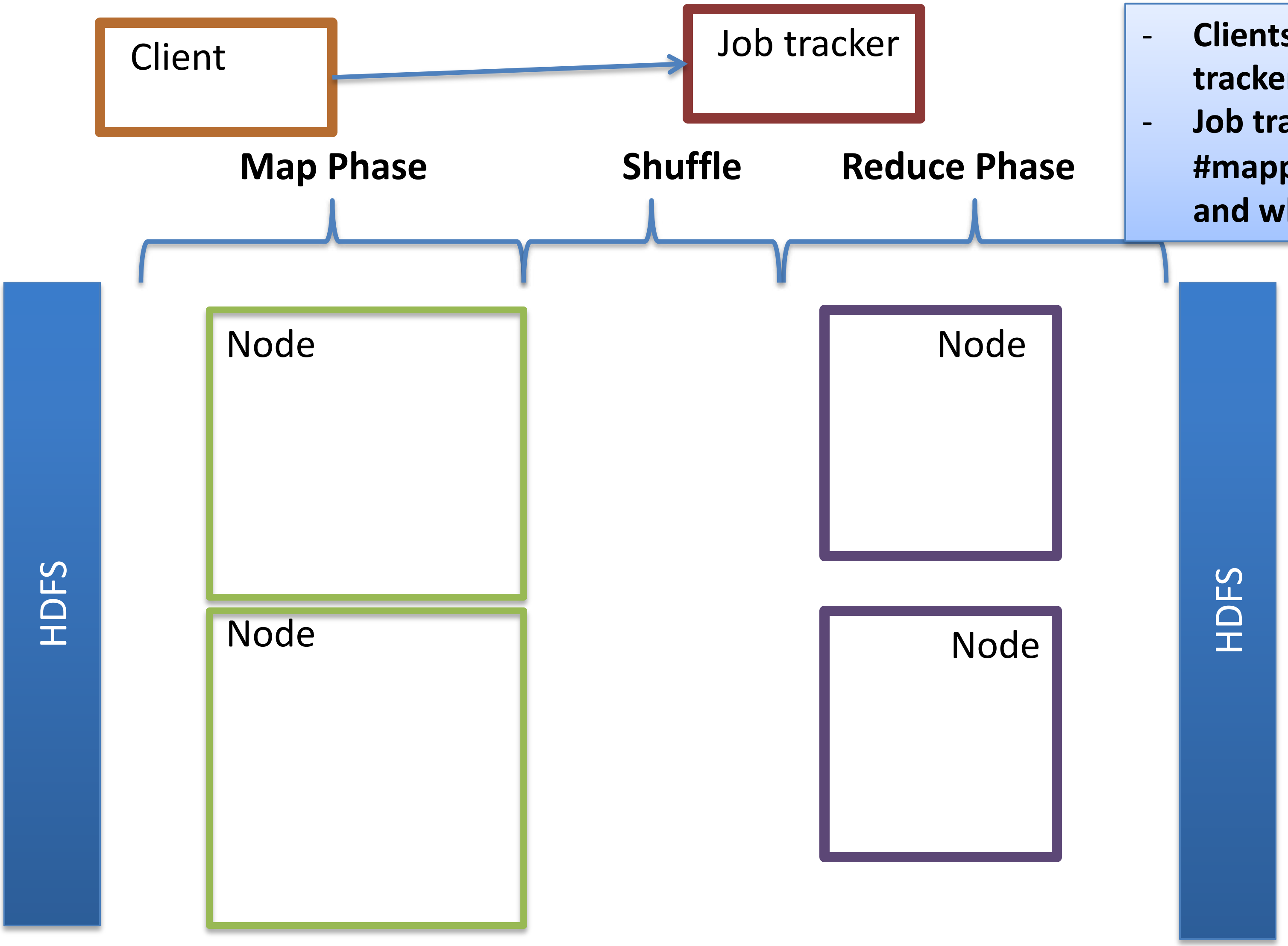
- **MapReduce**
  - Developed by google
  - Written in C
  - Runs on top of GFS (Google's distributed filesystem)
- **Hadoop**
  - Open source Apache project
  - Written in Java
  - Runs on-top of HDFS

- **Anatomy of a Hadoop cluster**
  - **Job tracker**
    - Clients submit MR jobs to the job tracker
    - Job tracker monitors progress
  - **Task tracker aka workers**
    - Execute map and reduce jobs
- **Job**
  - **Input**: files from HDFS
  - **Output**: written to HDFS
  - Map/Reduce UDFs

3. Hadoop

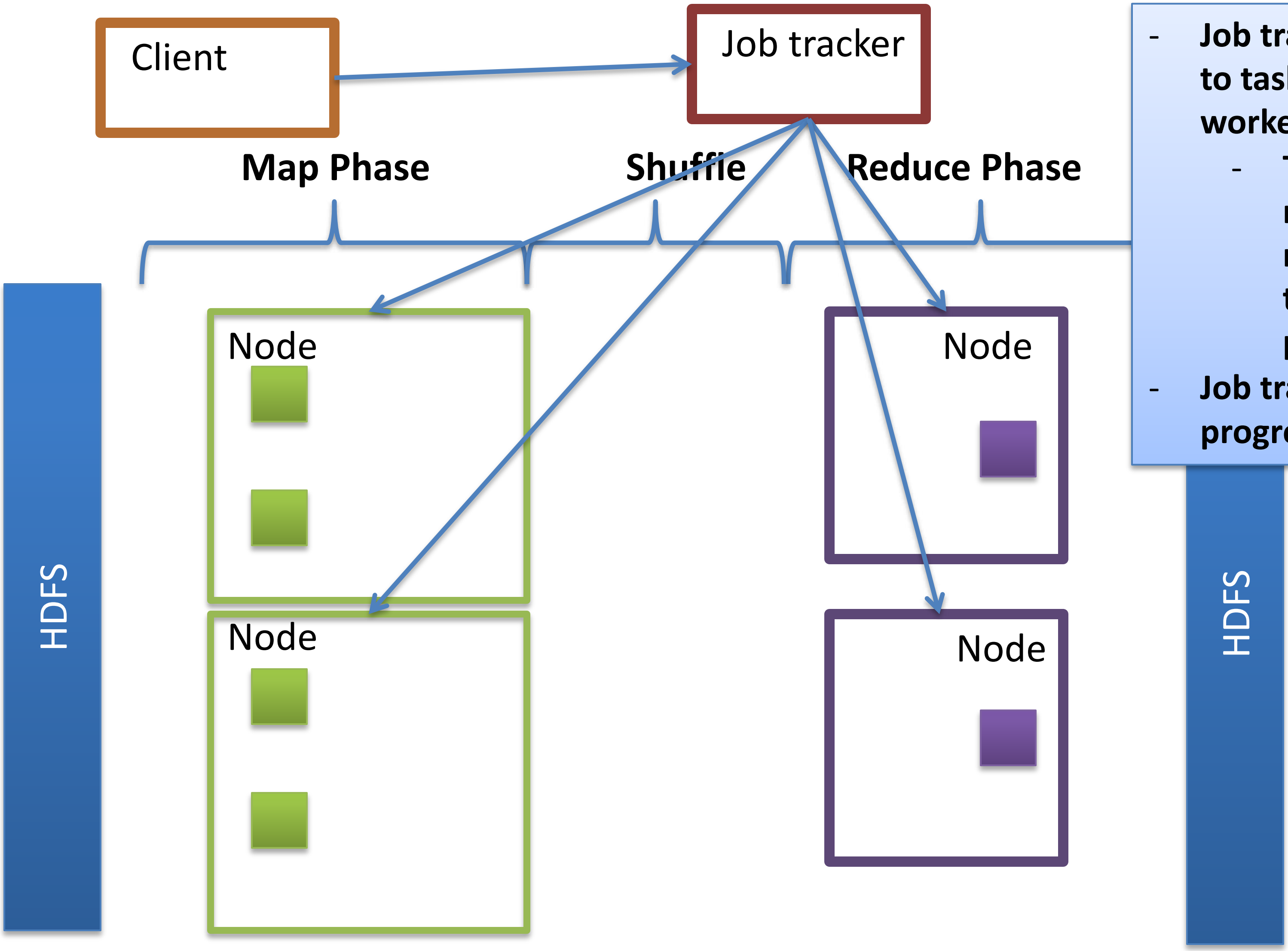
- **Fault tolerance**
  - **Handling stragglers**
    - Job tracker will reschedule jobs to a different worker if the worker falls behind too much with processing
  - **Materialization**
    - Inputs are read from HDFS
    - Workers write results of map jobs assigned to them to local disk
    - Workers write results of reduce jobs to HDFS for persistence

3. Hadoop – MR Job



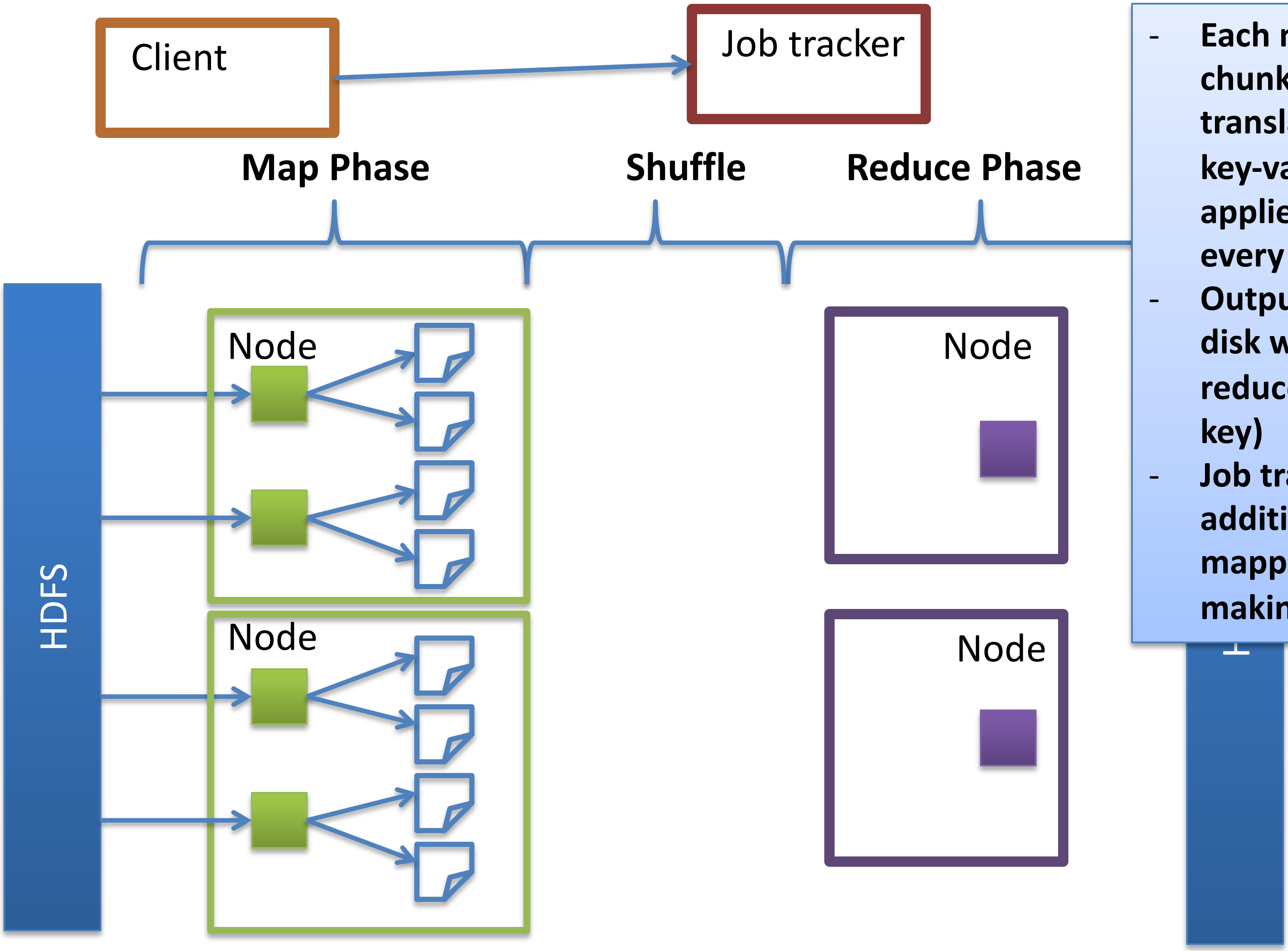
- Clients sends job to job tracker
- Job tracker decides #mappers, #reducers and which nodes to use

3. Hadoop – MR Job



- Job tracker sends jobs to task tracker on worker nodes
    - Try to schedule map jobs on nodes that store the chunk processed by a job
- Job tracker monitors progress

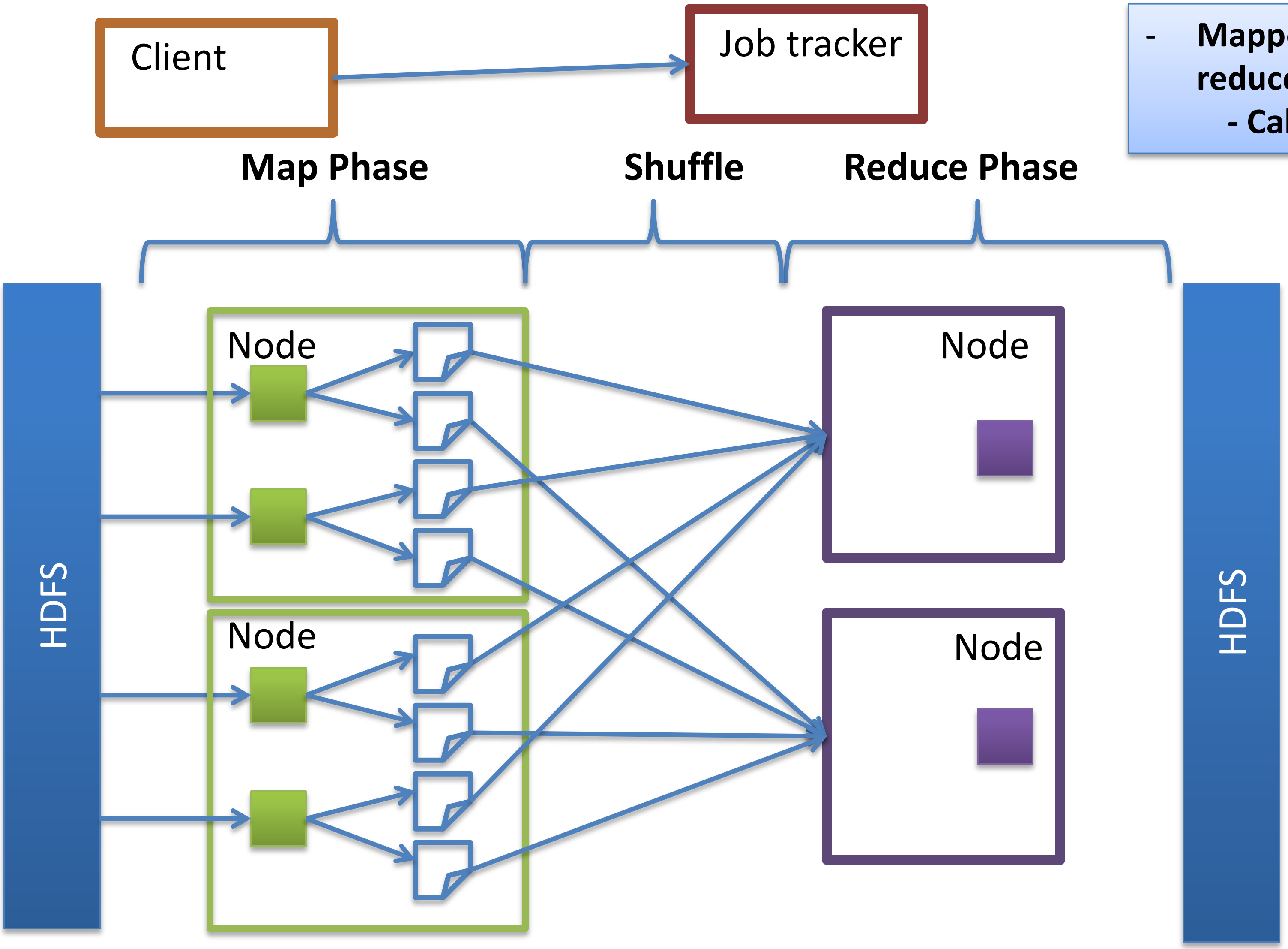
3. Hadoop – MR Job



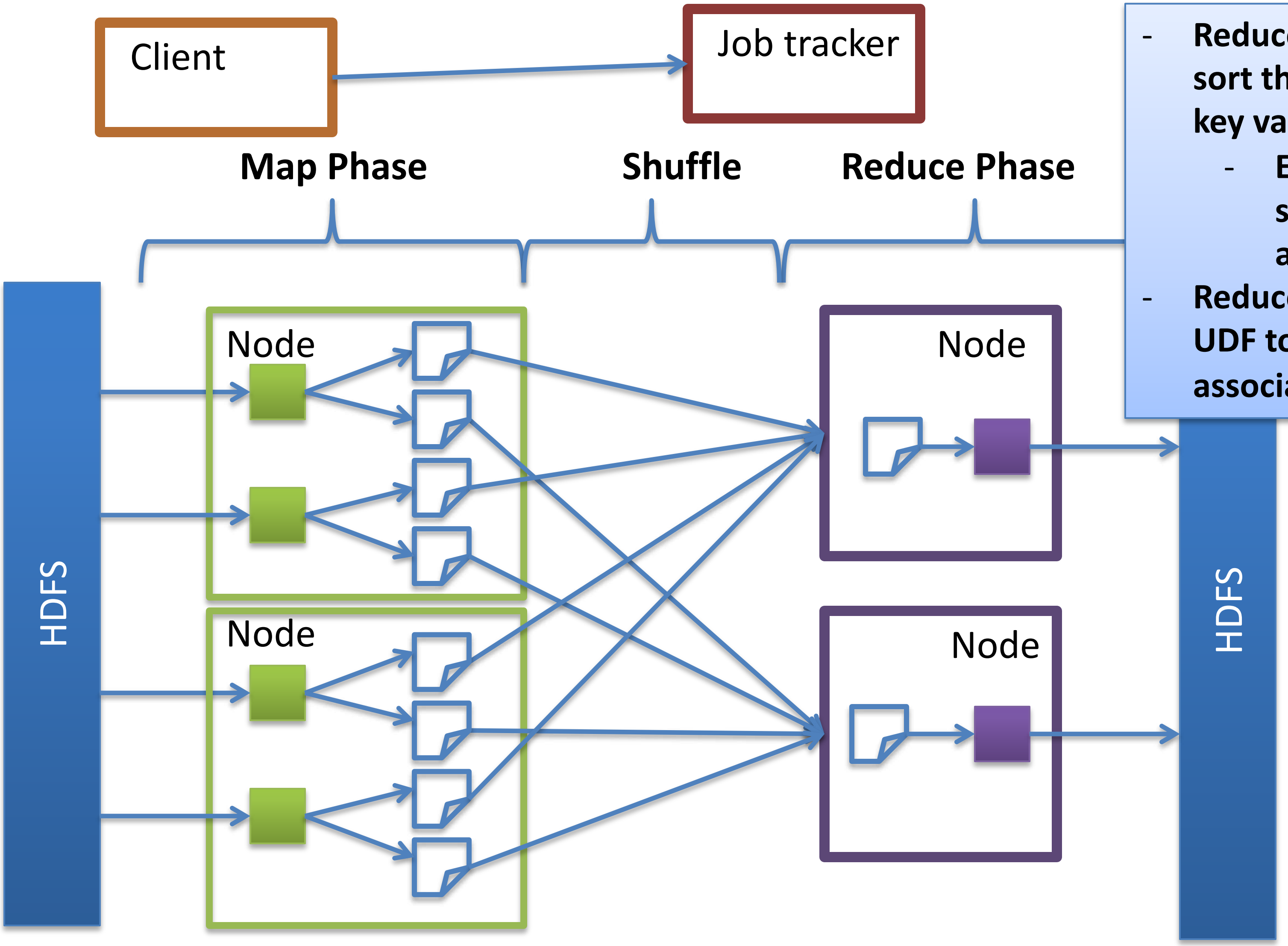
- Each mapper reads its chunk from HDFS, translates the input into key-value pairs and applies the map UDF to every (k,v)
- Outputs are written to disk with one file per reducer (hashing on key)
- Job tracker may spawn additional mappers if mappers are not making progress

3. Hadoop – MR Job

- Mappers send files to reducers (scp)
  - Called shuffle



3. Hadoop – MR Job



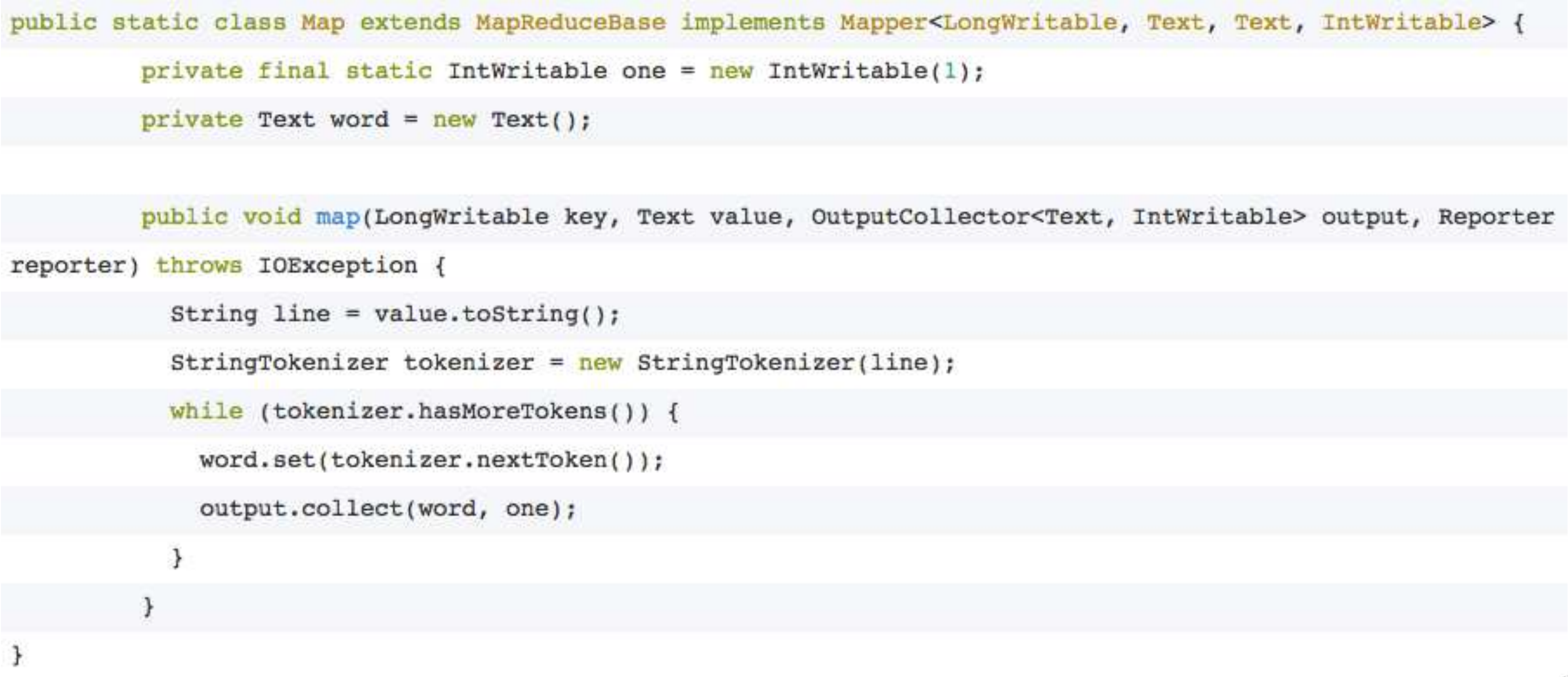
- Reducers merge and sort these input files on key values
  - External merge sort where runs already exists
- Reducer applies reduce UDF to each key and associated list of values

3. Combiners

- Certain reduce functions lend themselves to pre-aggregation
  - E.g., SUM(revenue) group by state
    - Can compute partial sums over incomplete groups and then sum up the pre-aggregated results
  - This can be done at the mappers to reduce amount of data send to the reducers
- Supported in Hadoop through a user provided combiner function
  - The combiner function is applied before writing the mapper results to local disk

3. Example code – Word count

- https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

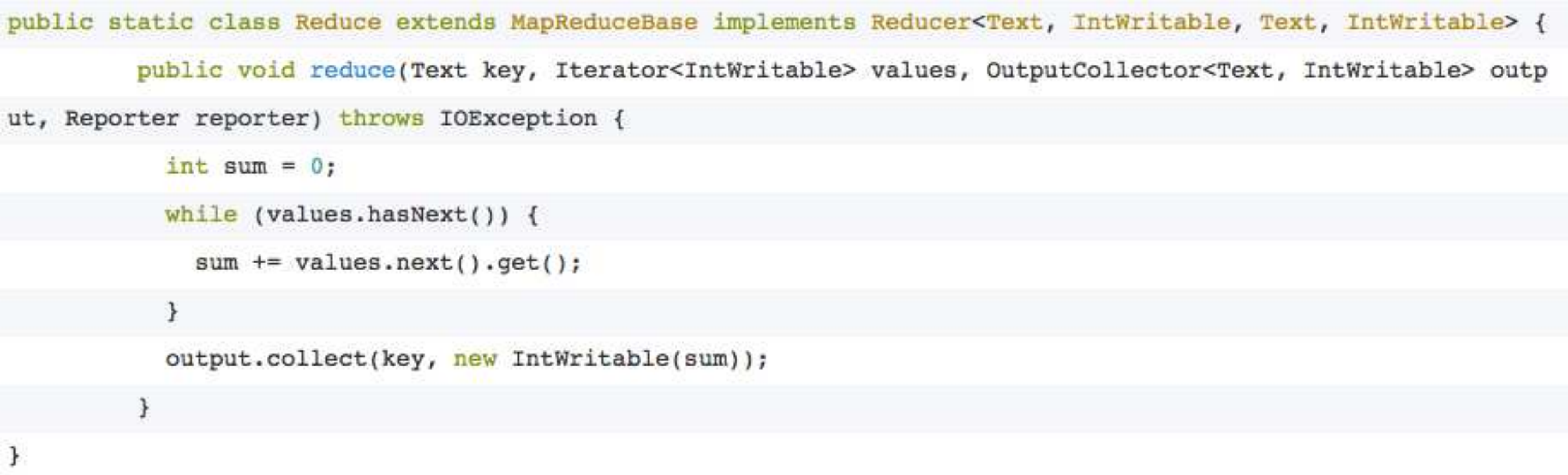


```
public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter
reporter) throws IOException {
          String line = value.toString();
          StringTokenizer tokenizer = new StringTokenizer(line);
          while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            output.collect(word, one);
          }
        }
}
```

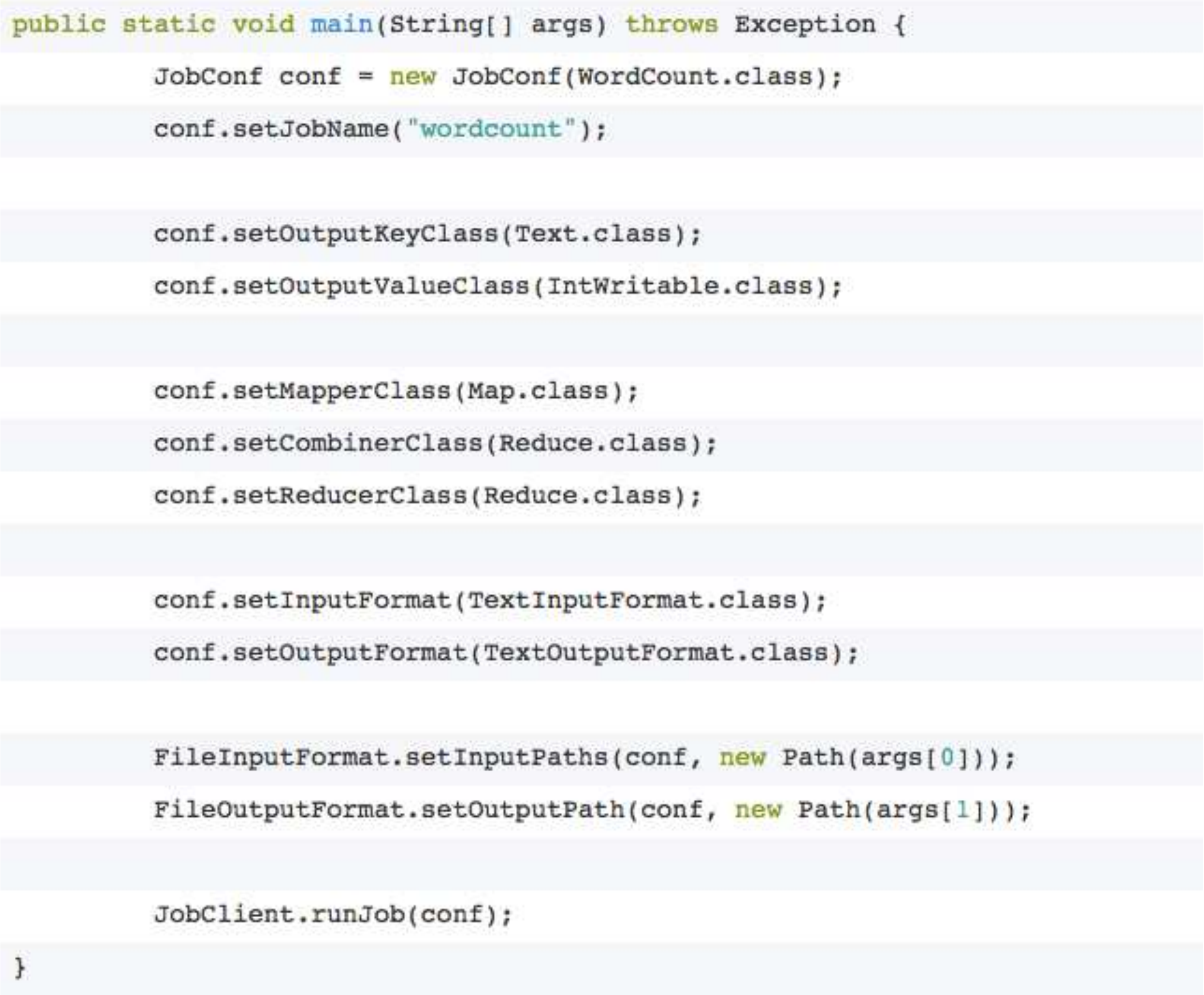
3. Example code – Word count

- https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html



```
public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
         public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> outp
ut, Reporter reporter) throws IOException {
           int sum = 0;
           while (values.hasNext()) {
             sum += values.next().get();
           }
           output.collect(key, new IntWritable(sum));
         }
}
```

3. Example code – Word count



```
public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);

        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);

        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        JobClient.runJob(conf);
}
```

3. Systems/Languages on top of MapReduce

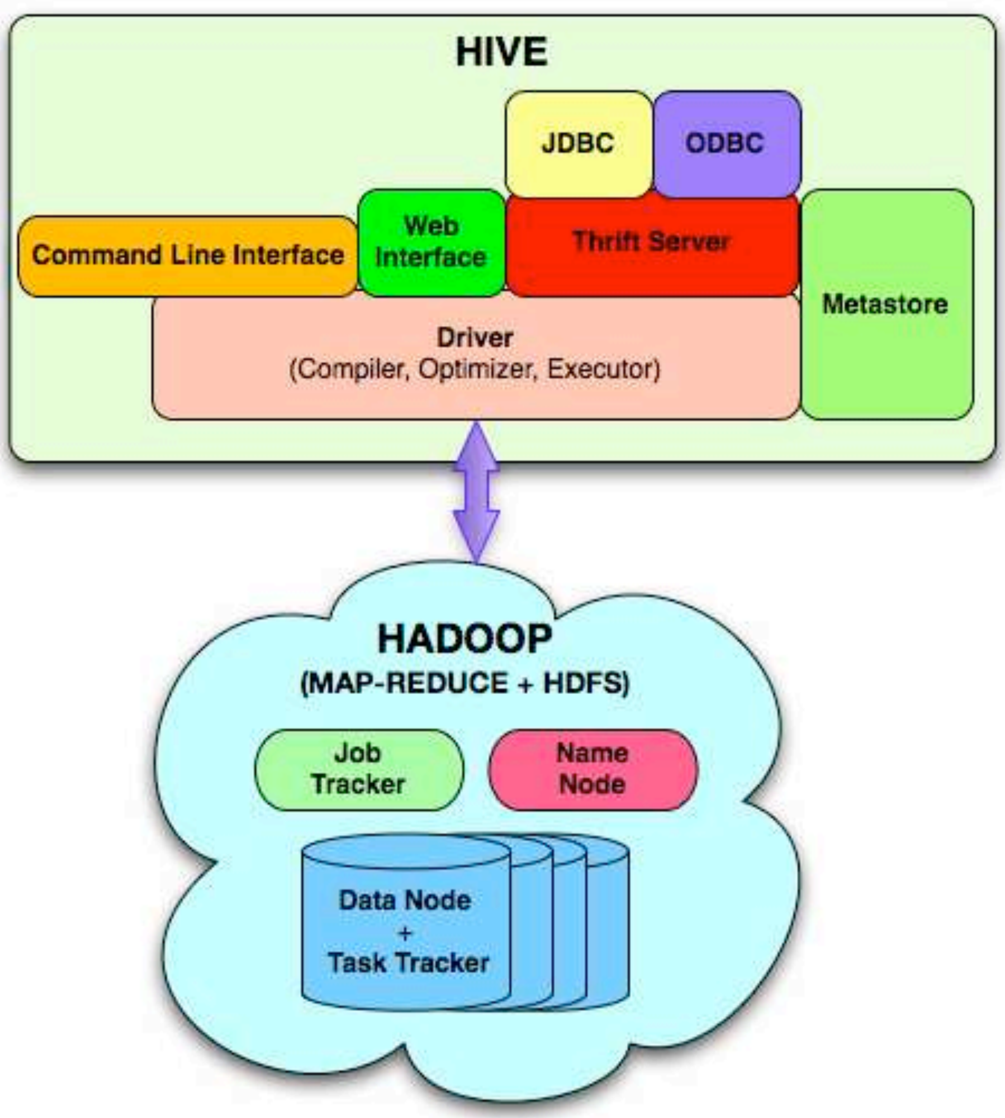
- Pig
  - Scripting language, compiled into MR
  - Akin to nested relational algebra
- Hive
  - SQL interface for warehousing
  - Compiled into MR
- …

3. Hive

- **Hive**
  - HiveQL: SQL dialect with support for directly applying given Map+Reduce functions as part of a query
  - HiveQL is compiled into MR jobs
  - Executed of Hadoop cluster

```
FROM (
        MAP doctext USING 'python wc_mapper.py' AS (word, cnt)
        FROM docs
        CLUSTER BY word
) a
REDUCE word, cnt USING 'python wc_reduce.py';
```

3. Hive Architecture



3. Hive Datamodel

- **Tables**
  - – Attribute-DataType pairs
  - – User can instruct Hive to partition the table in a certain way
- **Datatypes**
  - – Primitive: integer, float, string
  - – Complex types
    - Map: Key->Value
    - List
    - Struct
  - – Complex types can be nested
- **Example**:

  CREATE TABLE t1(st string, fl float, li list<map<string, struct<p1:int, p2:int>>);
- **Implementation**:
  - – Tables are stored in HDFS
  - – Serializer/Deserializer - transform for querying

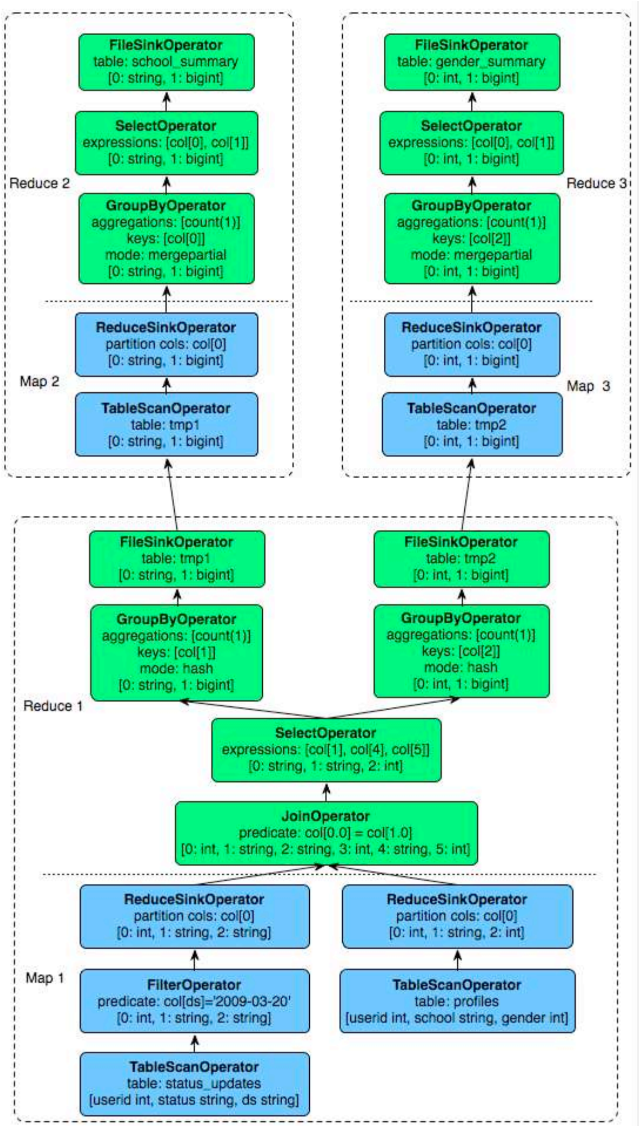
3. Hive - Query Processing

- Compile HiveQL query into DAG of map and reduce functions.
  - A single map/reduce may implement several traditional query operators
    - E.g., filtering out tuples that do not match a condition (selection) and filtering out certain columns (projection)
  - Hive tries to use the partition information to avoid reading partitions that are not needed to answer the query
    - For example
      - table **instructor**(name,department) is partitioned on department
      - **SELECT** name **FROM** instructor **WHERE** department = 'CS'
      - This query would only access the partition of the table for department 'CS'

3. Operator implementations

- **Join implementations**
  - **Broadcast join**
    - Send the smaller table to all nodes
    - Process the other table partitioned
      - Each node finds all the join partners for a partition of the larger table and the whole smaller table
  - **Reduce join (partition join)**
    - Use a map job to create key-value pairs where the key is the join attributes
    - Reducer output joined rows

3. Example plan



Spark

- MR uses heavy materialization to achieve fault tolerance
  - A lot of I/O
- **Spark**
  - Works in main memory (where possible)
  - Inputs and final outputs stored in HDFS
  - Recomputes partial results instead of materializing them - **resilient distributed datasets** (*RDD*)
    - **Lineage**: Need to know from which chunk a chunk was derived from and by which computation

Summary

- Big data storage systems
- Big data computation platforms
- Big data "databases"
- How to achieve scalability
  - Fault tolerance
  - Load balancing
- Big data integration
  - Pay-as-you-go
  - Schema later

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

ILLINOIS INSTITUTE
OF TECHNOLOGY

CS520
Data Integration, Warehousing, and Provenance

8. Provenance

IIT DBGroup

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**

8.What is Data Provenance?

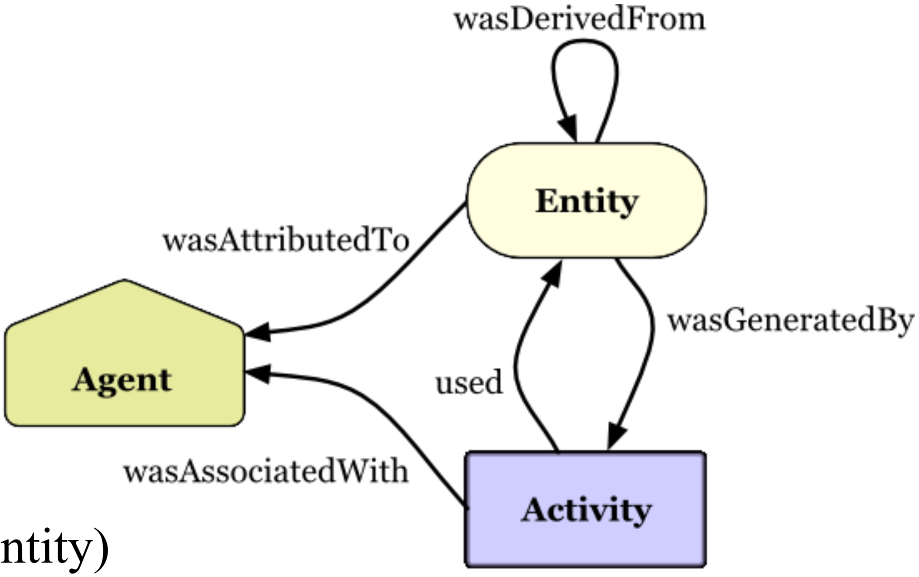
- **Metadata** describing the **origin** and **creation process** of **data**
  - **Data items**
    - Data item **granularity**
      - **A File**
      - **A Database**
      - **An Attribute value**
      - **A Row**
  - **Transformations**
    - Transformation **granularity**
      - **A program**
      - **A query**
      - **An operator in a query**
      - **A line in a program**

8.What is Data Provenance?

- **Provenance** records **dependencies**
  - **Data dependencies**
    - Data item x was used to generate data item y
  - **Dependencies between transformations and data**
    - Transformations generated a data item
    - Transformations used a data item

8. Provenance as graphs

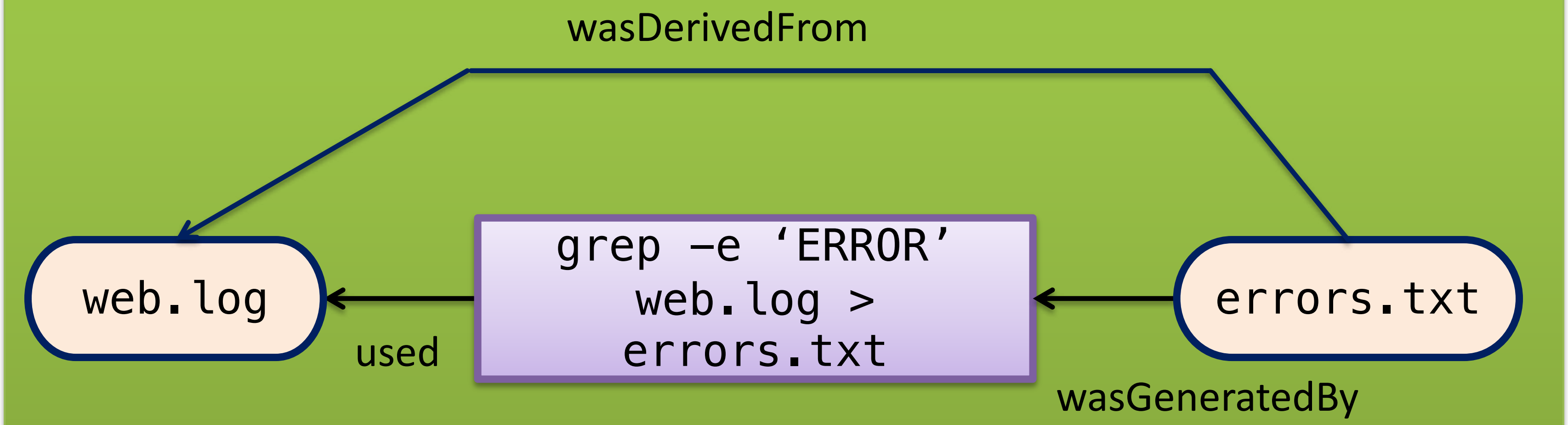
- **Provenance graphs** (W3C PROV standard)
  - https://www.w3.org/TR/2013/NOTE-prov-primer-20130430/
  - **Nodes**
    - **Entities**
      - what we call data items
    - **Activities**
      - what we call transformations
    - **Agents**
      - Trigger / control activities
      - E.g., users and machines



  - **Edges**
    - **wasDerivedFrom** (entity – entity)
      - Data dependencies
    - **wasGeneratedBy** (activity – entity)
      - Transformation generated an output data item
    - **used** (entity – activity)
      - Transformation read and input data item

8. PROV example

Example: find errors in a weblog with grep



wasDerivedFrom

web.log

used

```
grep –e 'ERROR'
web.log >
errors.txt
```

errors.txt

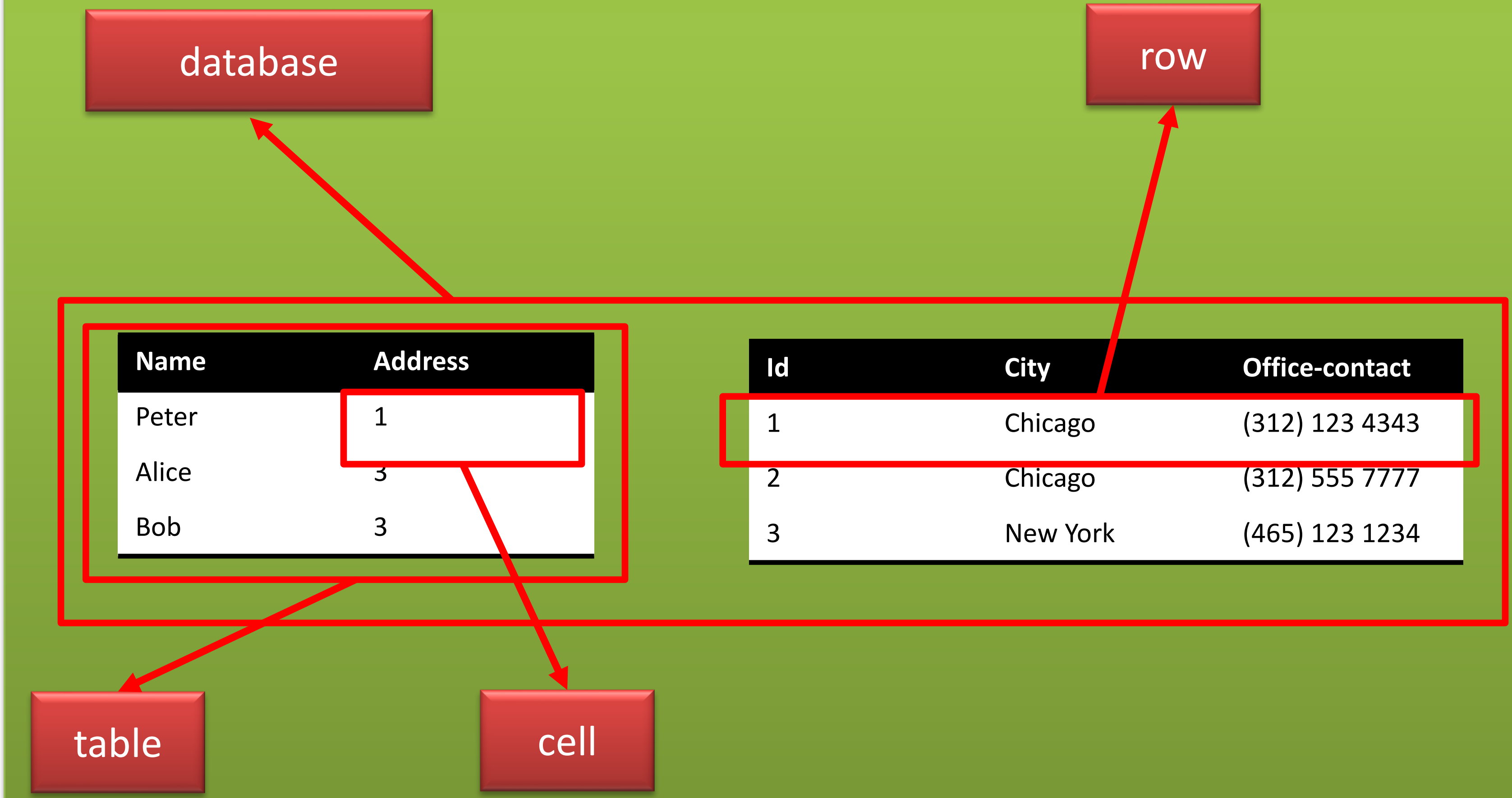
wasGeneratedBy

8. Provenance for Databases

- **Transformations**
  - **SQL queries**
  - **Updates and transactions**
  - **Procedural code**
- **Data items**
  - **Databases**
  - **Tables**
  - **Rows**
  - **Cells** (attribute value of a row)

8. Databases Prov. – Data items

Example: data item granularity



Name	Address
Peter	1
Alice	3
Bob	3

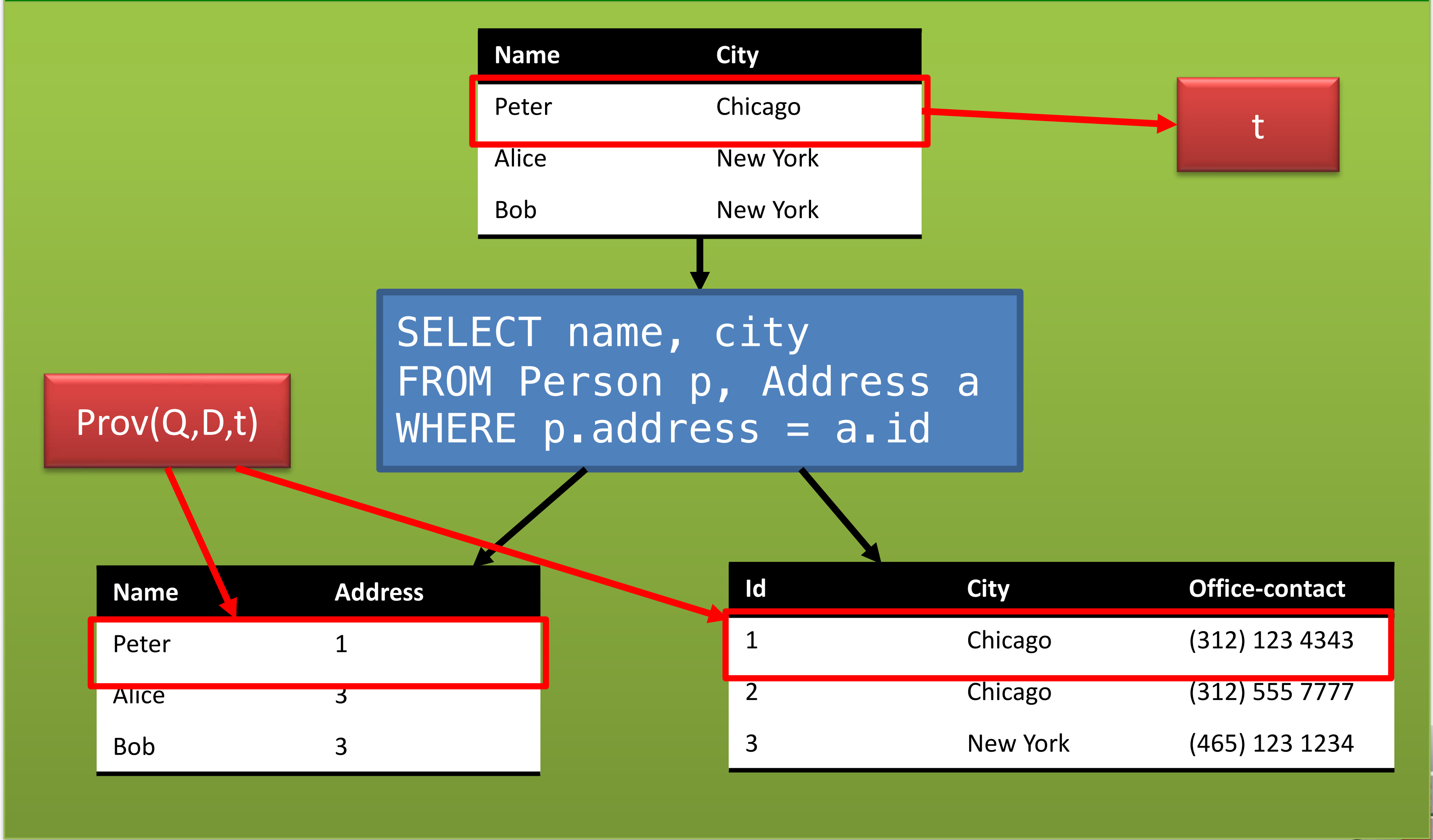
Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

8. Provenance for Queries

- **Data dependencies**
  - For each **output tuple** (**cell**) of the query determine which **input tuples** (**cells**) of the query it depends on
- **Formally (kind of)**
  - Given database **D** and query **Q** and tuple **t** in **Q(D)**
    - **Prov(Q,D,t)** = the subset of **D** that was used to derive **t** through **Q**

8. Databases Prov. – Data items

Example: data item granularity



Name	City
Peter	Chicago
Alice	New York
Bob	New York

```
SELECT name, city
FROM Person p, Address a
WHERE p.address = a.id
```

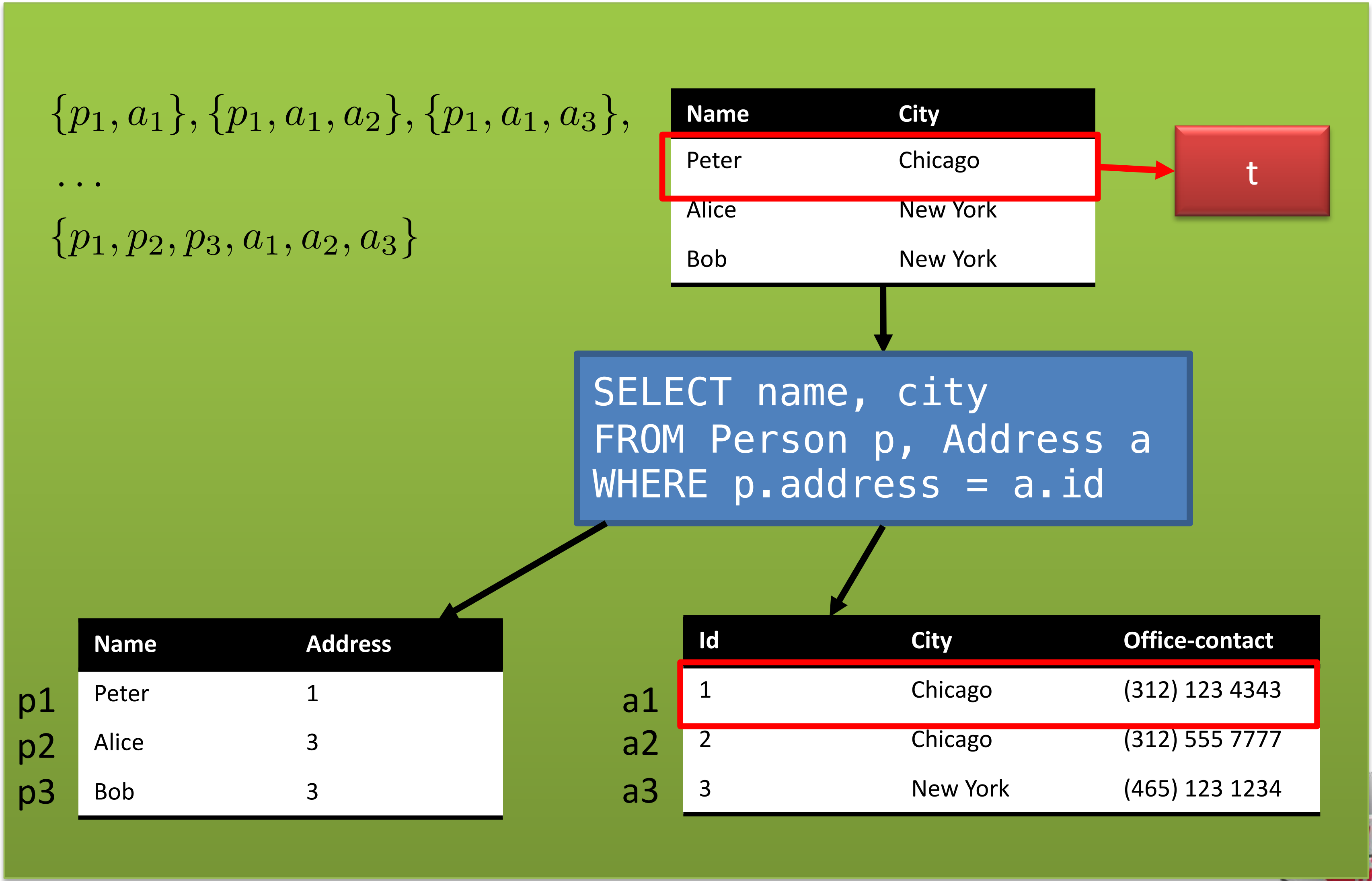
Name	Address
Peter	1
Alice	3
Bob	3

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

8. Formalizing data dependencies

- **How to formalize data dependencies?**
  - **Access**: query did read the data
    - No! Everything depends on everything!
  - **Sufficiency**: the provenance is enough to produce the result tuple **t**
    - **t** is in **Q(Prov(Q,D,t))**
    - Guarantees that everything that was needed to produce **t** is in the provenance

8. Sufficiency - Example



$$\{p_1, a_1\}, \{p_1, a_1, a_2\}, \{p_1, a_1, a_3\},$$

$$\ldots$$

$$\{p_1, p_2, p_3, a_1, a_2, a_3\}$$

Name	City
Peter	Chicago
Alice	New York
Bob	New York

```
SELECT name, city
FROM Person p, Address a
WHERE p.address = a.id
```

	Name	Address
p1	Peter	1
p2	Alice	3
p3	Bob	3

	Id	City	Office-contact
a1	1	Chicago	(312) 123 4343
a2	2	Chicago	(312) 555 7777
a3	3	New York	(465) 123 1234

8. Sufficiency cont.

- **Is sufficiency enough?**
  - No, sufficiency does not prevent irrelevant inputs to be included in the provenance!
  - Sufficiency does not uniquely define provenance
- **Monotone Queries**
  - A query **Q** is monotone if

$$\forall D, D' : D \subseteq D' \Rightarrow Q(D) \subseteq Q(D')$$

- **For all monotone queries Q:**
  - If D is sufficient then so is any superset of D
  - in particular the input database D is sufficient

8. Why provenance

ILLINOIS INSTITUTE OF TECHNOLOGY

- **Rationale:** define provenance as the set of all sufficient subsets of the input
  - this uniquely defines provenance
  - this does not solve the redundancy issue!
- **Why provenance**:

$$Why(Q, D, t) = \{D' \mid D' \subseteq D \wedge t \in Q(D')\}$$

- Each sufficient subset of D in the why provenance is called a witness

8. Minimality

- **Rationale:**
  - Remove tuples that do not contribute to the result
  - If a subset of a witness is already sufficient then everything not in the subset is unnecessary and should be removed
- **Definition**

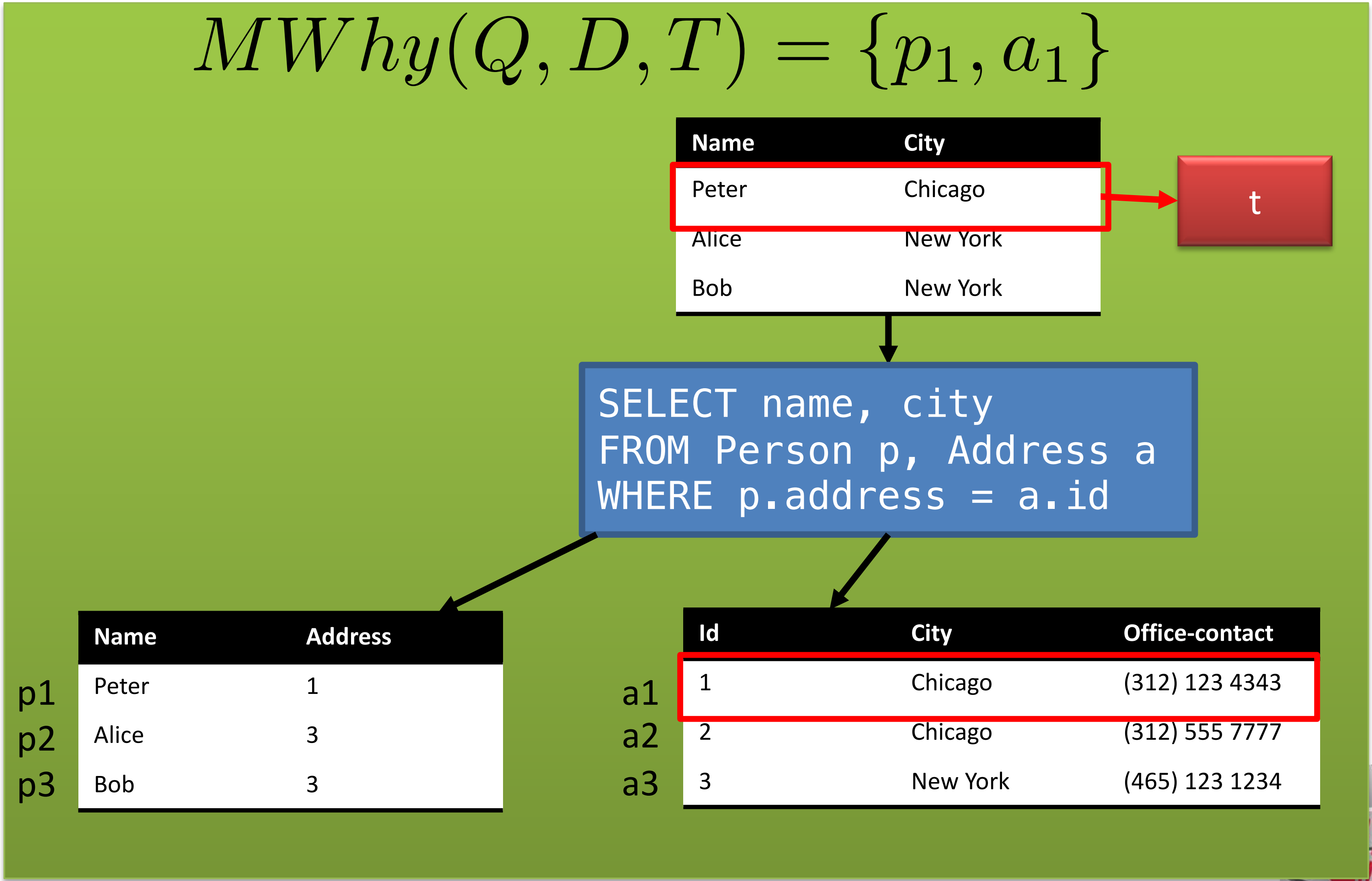
$$D' \text{ is a minimal witness for } t \text{ if } \forall D' \subset D'' : t \notin Q(D'')$$

8. Minimal Why provenance

- **Minimal Why provenance**:
- Only include minimal witnesses

$$MWhy(Q, D, t) = \{D' \mid D' \in Why(Q, D, t) \land \nexists D'' \subset D' : D'' \in Why(Q, D, t)\}$$

8. Sufficiency - Example



$$MWhy(Q,D,T) = \{p_1, a_1\}$$

Name	City
Peter	Chicago
Alice	New York
Bob	New York

t

```
SELECT name, city
FROM Person p, Address a
WHERE p.address = a.id
```

	Name	Address
p1	Peter	1
p2	Alice	3
p3	Bob	3

	Id	City	Office-contact
a1	1	Chicago	(312) 123 4343
a2	2	Chicago	(312) 555 7777
a3	3	New York	(465) 123 1234

8. Why provenance - discussion

- **Independent of query syntax**
  - Queries are treated as blackbox functions
  - Equivalent queries have the same provenance!
- How to compute this efficiently?
  - The discussion so far only gives a brute force exponential time algorithm
    - For each subset D’ of D test whether it is a witness
    - Then for every witness test whether it is minimal by testing for a subset relationship with all other witnesses
  - Top-down rules that calculate MWhy in a syntax driven manner

8. MWhy – top-down recursion

- Define top-down syntax-driven rules
  - calculate a set of witnesses
  - Minimizing the result of these rules returns MWhy

$$W(R, t, I) = \{\{t\}\}$$

$$W(\sigma_\theta(Q), t, I) = W(Q, t, I)$$

$$W(\pi_A(Q), t, I) = \bigcup_{u \in Q(I) : u.A = t} W(Q, u, I)$$

$$W(Q_1 \bowtie_\theta Q_2, t, I) = \{(w_1 \cup w_2) \mid w_1 \in W(Q_1, t_1, I) \\ \wedge w_2 \in W(Q_2, t_2, I) \wedge t = (t_1, t_2)\}$$

$$W(Q_1 \cup Q_2, t, I) = W(Q_1, t, I) \cup W(Q_2, t, I)$$

8. Why provenance – discussion 2

- **This works well for set semantics, but not bag semantics**
  - Minimization can lead to incorrect results with bag semantics
  - Treating the provenance as sets of tuples does not align well with bags
- **This only encodes data dependencies**
  - We know from which tuples we have derived a result, but not how the tuples were combined to produce the result

8. Semiring annotations - Agenda

- **We will now discuss a model that …**
  - Provides provenance for both sets and bags
  - Allows us to track how tuples where combined
  - Can express many other provenance models including MWhy
  - Can also express bag and set semantics and other extensions of the relational model such as the incomplete databases we discussed earlier

8. Annotations on Data

- **Annotations**
  - Allow data to be associated with additional metadata
    - Comments from users
    - Trust annotations
    - Provenance
    - …
  - Here we are interested in annotations on the tuples of a table

8. K-relations

- **Annotation domain**
  - We fix a set K of possible annotations
  - Examples
    - Powerset(Powerset(D)) = all possible sets of witnesses
      - We can annotate each tuple with its Why or MWhy provenance
    - Natural numbers
      - We can simulate bag semantics by annotating each tuple with its multiplicity
    - A set of possible world identifiers D1 to Dn
      - Incomplete databases

8. K-relations

- **K-relations**
  - We fix a set **K** of possible annotations
  - **K** has to have a distinguished element $\mathbf{0_K}$
  - Assume some data domain **U**
  - An n-ary K-relation is a function

$$\mathcal{U}^n \rightarrow K$$

    - We associate an annotation with every possible n-ary tuple
    - $\mathbf{0_k}$ is used to annotate tuples that are not in the relation
    - Only finitely many tuples are allowed to be mapped to a non-zero annotation

8. Example – bag semantics

Bag Semantics

Name	Address
Peter	1
Peter	1
Peter	1
Alice	3
Alice	3
Bob	3

N-relation

Name	Address	Annotation
Peter	1	3
Alice	3	2
Bob	3	1

8. Example – set semantics

Bag Semantics

Name	Address
Peter	1
Peter	1
Peter	1
Alice	3
Alice	3
Bob	3

B-relation

Name	Address	Annotation
Peter	1	true
Alice	3	true
Bob	3	true

$$\mathbb{B} = \{false, true\}$$

8. Example – incomplete DBs

ILLINOIS INSTITUTE OF TECHNOLOGY

Incomplet Database

D_1

Name	Address
Peter	1
Peter	2
Bob	3

D_2

Name	Address
Peter	1
Alice	2
Bob	3

Ω -relation

Name	Address	Annotation
Peter	1	{D1,D2}
Peter	2	{D1}
Alice	2	{D2}
Bob	3	{D1,D2}

$$\Omega = \mathcal{P}(\{D_1, D_2\})$$
$$= \{\emptyset, \{D_1\}, \{D_2\}, \{D_1, D_2\}\}$$

8. Example – MWhy

MWhy

Name	Address
Peter	1
Peter	2
Bob	3

MWhy(p1) = {{x1}}
MWhy(p2) = {{x2,a1},{x3}}
Mwhy(p3) = {{x4,a1},{x4,a2}}

PosBool[X]-relation

Name	Address	Annotation
Peter	1	{{x1}}
Peter	2	{{x2,a1},{x3}}
Bob	3	{{x4,a1},{x4,a2}}

$$X = D$$

$$PosBool[X] = \mathcal{P}(\mathcal{P}(X))$$

8. K-relations – Query semantics

- **Annotated Databases are powerful**
  - We can many different types of information
  - However, what is the right query semantics?
    - e.g., bag and set semantics queries do not have the same semantics, let along queries over incomplete databases or calculating provenance
- **Query Semantics**
  - Split the query semantics into two parts
    - One part is generic and independent of the choice of K
    - One part is specific to the choice of K
  - => every K has to be paired with operations that define how annotations propagate through queries
    - The generic semantics uses these operations to calculate query result annotations

8. Semirings

- **A semiring** $\mathcal{K} = (K, \oplus_{\mathcal{K}}, \otimes_{\mathcal{K}}, 0_{\mathcal{K}}, 1_{\mathcal{K}})$
  - K is the set of elements of semiring
    - We use them as annotations
  - There are two binary operations

    $$\oplus_{\mathcal{K}}, \otimes_{\mathcal{K}} : K \times K \rightarrow K$$

    - We will use them to combine annotations of input tuples
      - Addition will be used to model operations that are disjunctive in nature (union, projection)
      - Multiplication will be used to model operations that are conjunctive (join)
  - Two distinguished elements $0_{\mathcal{K}}, 1_{\mathcal{K}}$

8. Semiring Laws

- **A semiring** $\mathcal{K} = (K, \oplus_{\mathcal{K}}, \otimes_{\mathcal{K}}, 0_{\mathcal{K}}, 1_{\mathcal{K}})$

$$
\begin{aligned}
k_1 \oplus_{\mathcal{K}} k_2 &= k_2 \oplus_{\mathcal{K}} k_1 && \text{(commutativity)}\\
k_1 \oplus_{\mathcal{K}} (k_2 \oplus_{\mathcal{K}} k_3) &= (k_1 \oplus_{\mathcal{K}} k_2) \oplus_{\mathcal{K}} k_3 && \text{(associativity)}\\
k_1 \otimes_{\mathcal{K}} k_2 &= k_2 \otimes_{\mathcal{K}} k_1 && \text{(commutativity)}\\
k_1 \otimes_{\mathcal{K}} (k_2 \otimes_{\mathcal{K}} k_3) &= (k_1 \otimes_{\mathcal{K}} k_2) \otimes_{\mathcal{K}} k_3 && \text{(associativity)}\\
k \oplus_{\mathcal{K}} 0_{\mathcal{K}} &= k && \text{(neutral element)}\\
k \otimes_{\mathcal{K}} 1_{\mathcal{K}} &= k && \text{(neutral element)}\\
k \otimes_{\mathcal{K}} 0_{\mathcal{K}} &= 0_{\mathcal{K}} && \text{(annihilation by zero)}\\
k_1 \otimes_{\mathcal{K}} (k_2 \oplus_{\mathcal{K}} k_3) &= (k_1 \otimes_{\mathcal{K}} k_2) \oplus (k_1 \otimes_{\mathcal{K}} k_3) && \text{(distributivity)}
\end{aligned}
$$

8. Semirings - Examples

$$\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$$

$$\mathbb{B} = (\mathbb{B}, \vee, \wedge, false, true)$$

$$\mathcal{K}_{MWhy}[X] = (\mathcal{P}(\mathcal{P}(X)), \cup, \uplus, \emptyset, \{\emptyset\})$$

$$\mathcal{K}_{\Omega}[X] = (\mathcal{P}(\Omega), \cup, \cap, \emptyset, \Omega)$$

$$\mathbb{N}[X] = (\mathbb{N}[X], +, \cdot, 0, 1)$$

8. Provenance Polynomials

- **Semiring** $\mathbb{N}[X] = (\mathbb{N}[X], +, \cdot, 0, 1)$
  - N[X] is the set of all polynomials over variables X
    - Intuitively X are tuple identifiers
  - **Provenance polynomials** are used to track provenance for **bag semantics**!
  - Provenance polynomials record how a result has been derived by combining input tuples
    - Multiplication means conjunctive use (as in join)
    - Addition means disjunctive use

8. K-relations – Query semantics

- **Positive relational algebra (RA$^+$)**
  - Selection, projection, cross-product, renaming, union

$$\textbf{Union:}\quad (R_1 \cup R_2)(t) = R_1(t) \oplus_{\mathcal{K}} R_2(t)$$

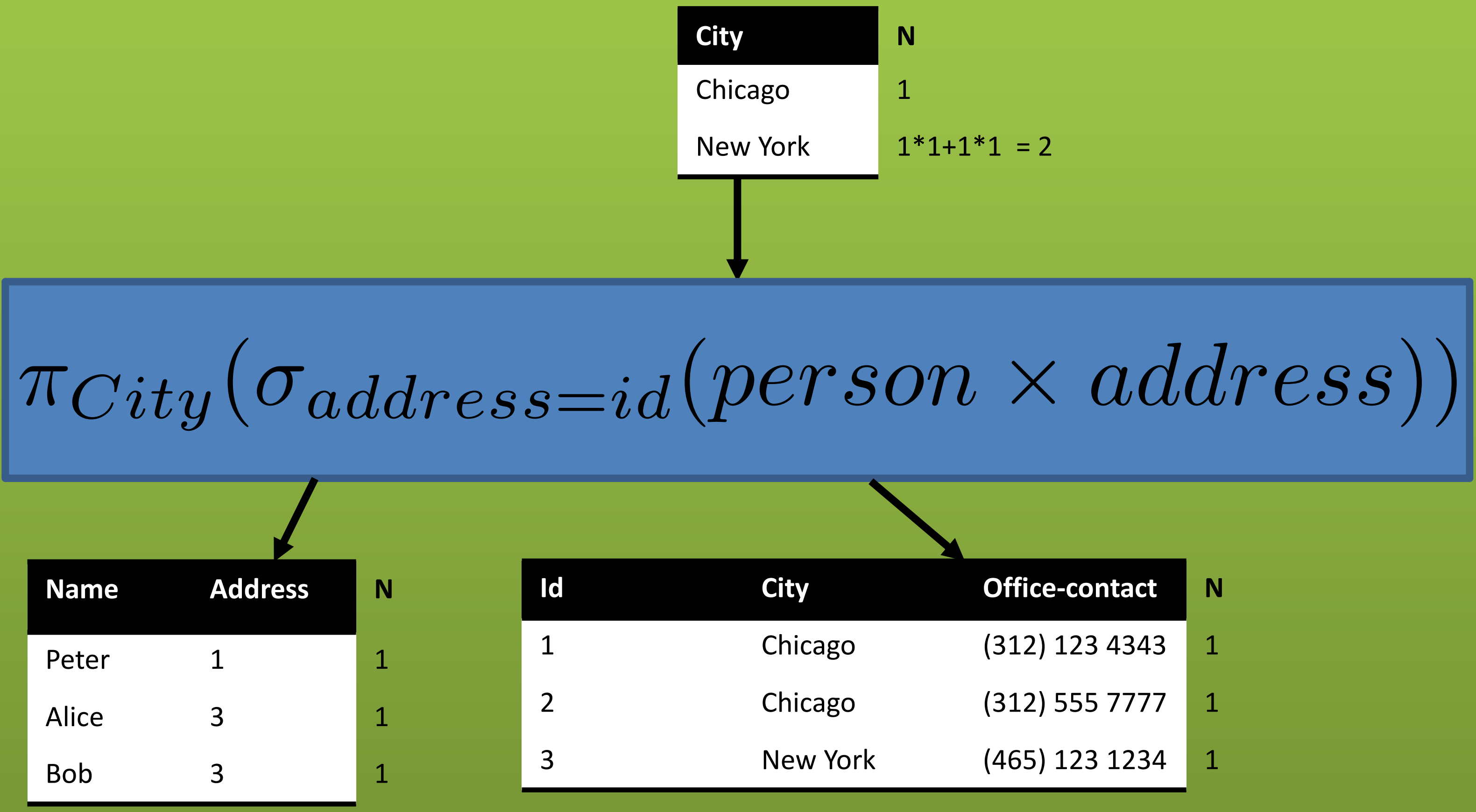
$$\textbf{Join:}\quad (R_1 \bowtie R_2)(t) = R_1(t[R_1]) \otimes_{\mathcal{K}} R_2(t[R_2])$$

$$\textbf{Projection:}\quad (\pi_A(R))(t) = \bigoplus_{t=t'[A]} R(t')$$

$$\textbf{Selection:}\quad (\sigma_\theta(R))(t) = R(t) \otimes_{\mathcal{K}} \theta(t)$$

$$\theta(t) = \begin{cases} 0_{\mathcal{K}} & \text{if } t \models \theta \\ 1_{\mathcal{K}} & \text{otherwise} \end{cases}$$

8. Query Semantics - Bags



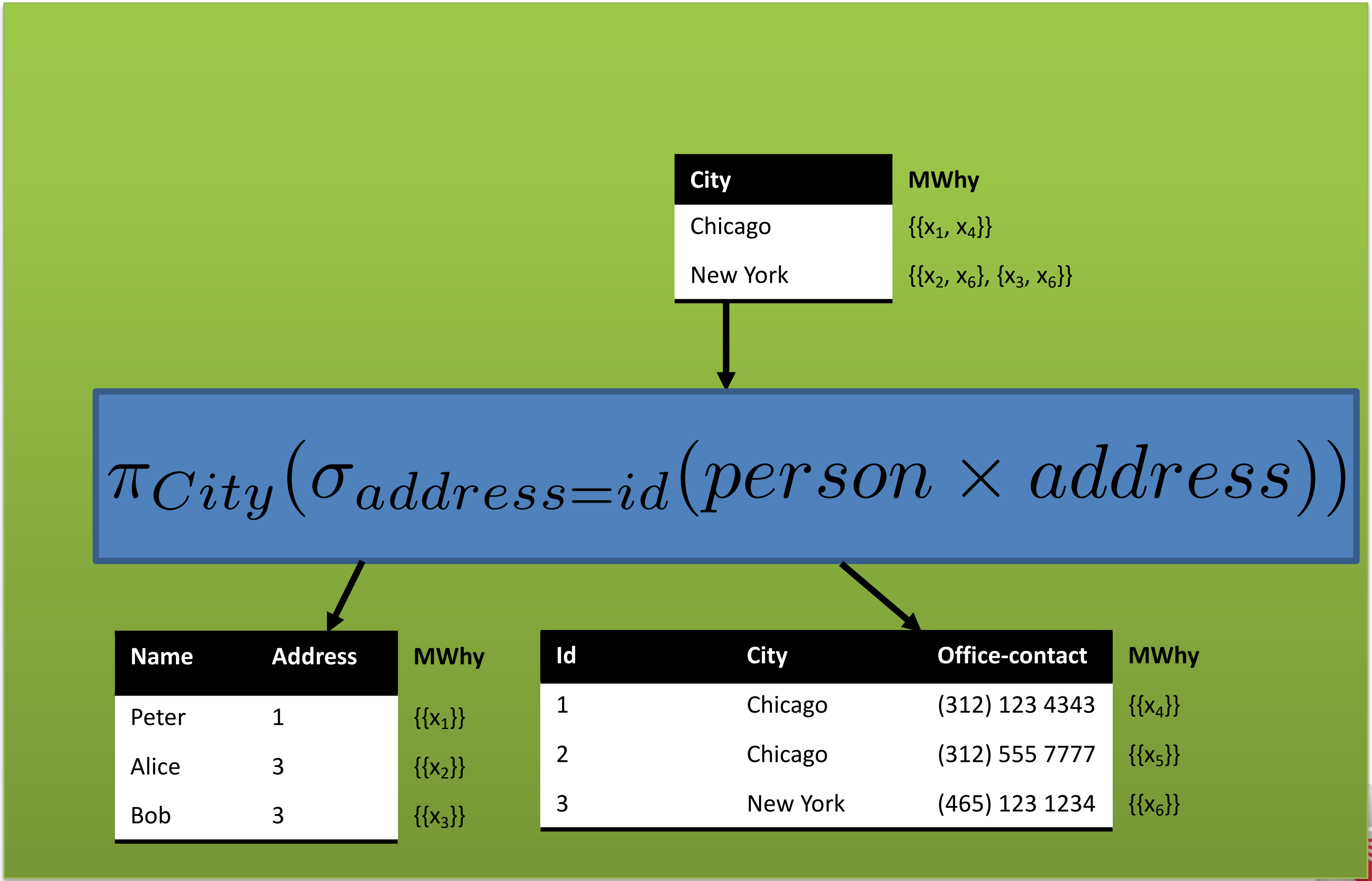
City	N
Chicago	1
New York	1*1+1*1 = 2

$$\pi_{City}(\sigma_{address=id}(person \times address))$$

Name	Address	N
Peter	1	1
Alice	3	1
Bob	3	1

Id	City	Office-contact	N
1	Chicago	(312) 123 4343	1
2	Chicago	(312) 555 7777	1
3	New York	(465) 123 1234	1

8. Query Semantics - MWhy



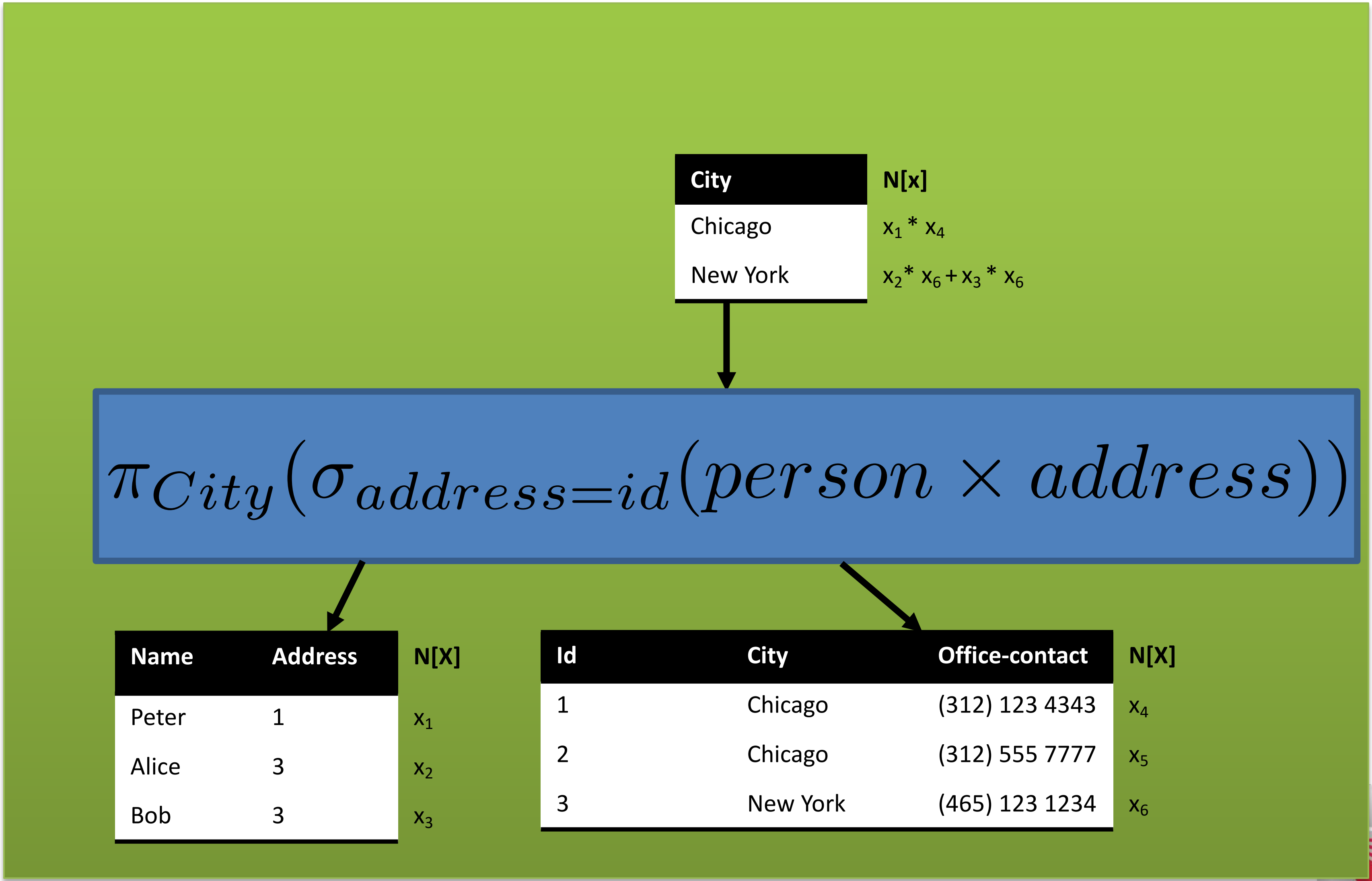
City	MWhy
Chicago	$\{\{x_1, x_4\}\}$
New York	$\{\{x_2, x_6\}, \{x_3, x_6\}\}$

$$\pi_{City}(\sigma_{address=id}(person \times address))$$

Name	Address	MWhy
Peter	1	$\{\{x_1\}\}$
Alice	3	$\{\{x_2\}\}$
Bob	3	$\{\{x_3\}\}$

Id	City	Office-contact	MWhy
1	Chicago	(312) 123 4343	$\{\{x_4\}\}$
2	Chicago	(312) 555 7777	$\{\{x_5\}\}$
3	New York	(465) 123 1234	$\{\{x_6\}\}$

8. Query Semantics - PP



City	N[x]
Chicago	$x_1 * x_4$
New York	$x_2 * x_6 + x_3 * x_6$

$$\pi_{City}(\sigma_{address=id}(person \times address))$$

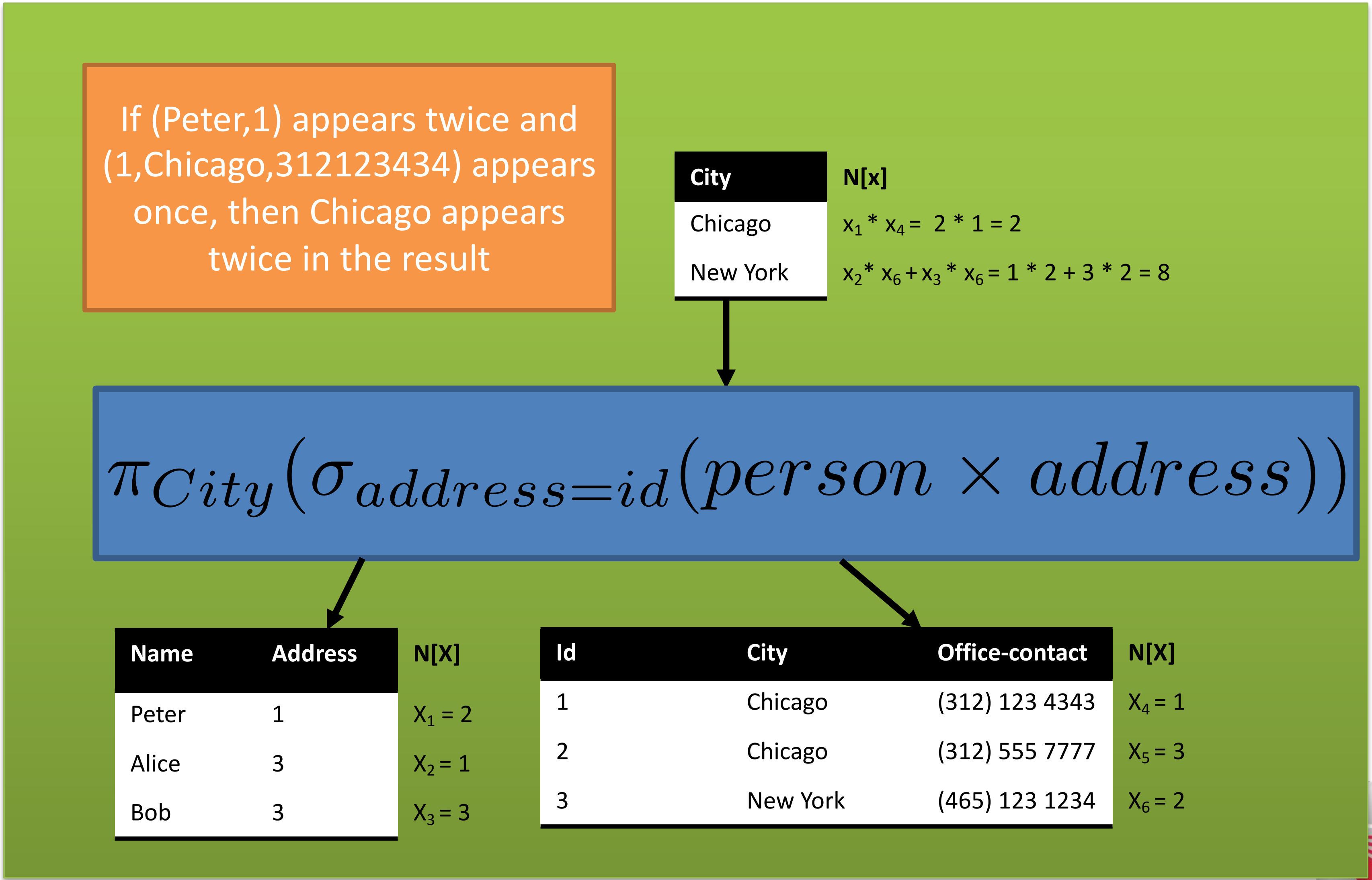
Name	Address	N[X]
Peter	1	x_1
Alice	3	x_2
Bob	3	x_3

Id	City	Office-contact	N[X]
1	Chicago	(312) 123 4343	x_4
2	Chicago	(312) 555 7777	x_5
3	New York	(465) 123 1234	x_6

8. Provenance Polynomials - Computability

- Recall our requirements of sufficiency and minimality
- Provenance polynomials fulfill a stronger requirement: **computability**
  - Given the result of a query in N[X], we can compute the query result in any other semiring K under a given assignment of input tuples (variables of the polynomials) to annotations from K

8. Query Semantics - PP



If (Peter,1) appears twice and (1,Chicago,312123434) appears once, then Chicago appears twice in the result

City	N[x]
Chicago	$x_1 * x_4 = 2 * 1 = 2$
New York	$x_2 * x_6 + x_3 * x_6 = 1 * 2 + 3 * 2 = 8$

$$\pi_{City}(\sigma_{address=id}(person \times address))$$

Name	Address	N[X]
Peter	1	$X_1 = 2$
Alice	3	$X_2 = 1$
Bob	3	$X_3 = 3$

Id	City	Office-contact	N[X]
1	Chicago	(312) 123 4343	$X_4 = 1$
2	Chicago	(312) 555 7777	$X_5 = 3$
3	New York	(465) 123 1234	$X_6 = 2$

8. Homomorphisms

- A function h from semiring K1 to K2 is a homomorphism if

$$h(k_1 \oplus_{\mathcal{K}_1} k_2) = h(k_1) \oplus_{\mathcal{K}_2} h(k_2)$$

$$h(k_1 \otimes_{\mathcal{K}_1} k_2) = h(k_1) \otimes_{\mathcal{K}_2} h(k_2)$$

$$h(0_{\mathcal{K}_1}) = 0_{\mathcal{K}_2}$$

$$h(1_{\mathcal{K}_1}) = 1_{\mathcal{K}_2}$$

- **Theorem**: Homomorphism commute with queries

$$Q(h(D)) = h(Q(D))$$

- **Proof Sketch**: queries are defined using semiring operations which commute with homomorphisms

8. Fundamental theorem

- **Theorem**: Homomorphism commute with queries

$$Q(h(D)) = h(Q(D))$$

- **Proof Sketch**: queries are defined using semiring operations which commute with homomorphisms
- **Theorem**: Any assignment X -> K induces a semiring homomorphism N[X] -> K

8. Summary

- **Provenance is information about the origin and creation process of data**
  - Data dependencies
  - Dependencies between data and the transformations that generated it
- **Provenance for Queries**
  - **Correctness criteria**:
    - sufficiency, minimality, computability
  - **Provenance models:**
    - Why, MWhy, Provenance polynomials