Data Integration and Analysis

04 Schema Matching and Mapping

Matthias Boehm

Graz University of Technology, Austria
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMVIT endowed chair for Data Management

Last update: Oct 25, 2019
Announcements/Org

- **#1 Video Recording**
  - Link in TeachCenter & TUbé (lectures will be public)
  - Since 2nd lecture, missing microphone

- **#2 Project Ideas**
  - **Oct 25:** published list of projects (end of this presentation)
  - **Nov 08:** exercise/project selection

- **#3 Bachelor or Master Theses**
Agenda

- Motivation and Terminology
- Schema Detection
- Schema Matching
- Schema Mapping
- Projects and Exercises
Motivation and Terminology
Recap: ETL/EAI Schema Transformations

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:element name="suppliers">
      <xsl:for-each select="/resultsets/resultset[@Tablename='Supplier']/row">
        <xsl:element name="supplier">
          <xsl:attribute name="ID"><xsl:value-of select="Suppkey"/></xsl:attribute>
          <xsl:element name="Name"><xsl:value-of select="SuppName"/></xsl:element>
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Are you kidding me? I have 100s of systems/apps and 1000s* of tables/attributes

---

Motivation and Terminology

---

* [Rudolf Munz: Datenmanagement für SAP Applikationen, BTW, 2007: 67,000 tables, 700,000 columns, 10,000 views, 13,000 indexes]
Schema Matching and Mapping

- **Schema Matching**
  - **Given:** two or more relational/hierarchical schemas (and data)
  - Find schema mappings in terms of logical attribute correspondences

- **Schema Mapping**
  - **Given:** logical attribute correspondences between schemas
  - Compile physical schema mapping programs (SQL, XSLT, XQuery, etc)

Well, I don’t even have a schema
Schema Mapping vs. Schema Integration

- **Schema Integration**
  - **Use case:** DWH schema (lecture 02), virtual/federated DBMS (lecture 03)
  - **Schema integration:** map existing schemas into consolidated schema
  - **Schema mapping** orthogonal, but both deal with semantic and structural heterogeneity
Recap: Types of Heterogeneity

→ Scope

Heterogeneity

Semantic Heterogeneity

Structural Heterogeneity

Attribute Heterogeneity

1. Synonyms/homonyms
2. Simple mapping (mathematical)
3. Different data types
4. Complex mappings
5. Language expressions

6. Nulls (Missing Values)
7. Virtual columns
8. Semantic incompatibility

9. Same attribute in different structure
10. Handling Sets
11. Attribute name w/o semantics
12. Attribute composition

Schema Detection

```
INSERT INTO Orders
SELECT BELNR, SUMME, DATUM
FROM Bestellungen
```
Atomic Data Type Detection

- **Overview**
  - **Problem:** Given CSV, JSON, XML files, detect data types of attributes
  - **Approach:** Basic extraction rules, regular expressions over data sample

- **Example: Schema Inference**
  - Infer schema for dataframe/dataset columns from sample
  - **Basic types:** Decimal, Boolean, Double, Integer, Long, String
  - Infer schema from java objects via class reflection

```scala
StructType(
    StructField(pid,IntegerType,true),
    StructField(name,StringType,true),
    StructField(pos,StringType,true),
    StructField(jnum,IntegerType,true),
    StructField(ncid,IntegerType,true),
    StructField(tid,IntegerType,true))
```

```scala
val ds = spark.read
    .format("csv")
    .option("header", true)
    .option("inferSchema", true)
    .option("samplingRatio", 0.001)
    .load("./data/players.csv");
```

`./data/players.csv:
pid,name,pos,jnum,ncid,tid
4614,Hannes Reinmayr,FW,14,1313,258
5435,Miroslav Klose,FW,11,789,144
6909,Manuel Neuer,GK,1,163,308`
Structure Extraction from Nested Documents

- **Structure Extraction Overview**
  - **Problem:** JSON/JSONL, XML documents with optional attributes, subtrees
  - **Approach:** Scan data, build maximum tree w/ attached meta data
  - Meta data := counts or appearances (e.g., document/line IDs)

- **Example JSON Schema Extraction**
  - Structure Identification Graph (appearances, data types, etc)
  - Reduced Structure Identification Graph

[Meike Klettke, Uta Störl, Stefanie Scherzinger: Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores. BTW 2015]
Data Profiling

- **#1 Inclusion Dependency (ID) Discovery**
  - ID from $R[X]$ to $S[Y]$ indicates that all values in $R[X]$ must appear in $S[Y] \rightarrow$ potential indication of *FK relationship* from $R[X]$ to $S[Y]$
  - **Robust inclusion dependency:**
    \[ |R[X] \in S[Y]| / |R[X]| > \delta \]
  - PK-FK candidate refinement
    (e.g., coverage, similarity, value ranges)

- **#2 Functional Dependency (FD) Discovery**
  - FDs indicative of relational schema
    *(key candidates, normalization)*
  - Discover minimal, non-trivial dependencies
  - Extend candidates along *set containment lattice*
    - $A \rightarrow C$ allows pruning $\{AB\} \rightarrow C$ (non-minimal)
    - NOT($A \rightarrow B$) allows pruning $A \rightarrow BC$
Semantic Data Type Detection

- **Problem**
  - Detect semantic types of columns (e.g., location, date, name)
  - **Use cases:** improved data cleaning, schema matching via semantic types

- **Sherlock: DNN-based Type Detection**
  - 78 semantic types from T2Dv2 → VizNet extraction
  - 1,588 features from each column, incl global stats
  - Formulation of type detection as multi-class classification problem

(Madelon Hulsebos et al: Sherlock: A Deep Learning Approach to Semantic Data Type Detection. *KDD 2019*)

http://webdatacommons.org/webtables/goldstandardV2.html
(DBPedia, Webtables)

(cardinality, uniqueness, avg #numerical chars)
Schema Matching

- Schema Detection
- Schema Matching
- Schema Mapping

```
INSERT INTO Orders
SELECT BELNR, SUMME, DATUM
FROM Bestellungen
```
Overview Schema Matching

**Motivation**
- Large and convoluted schemas (# tables, # attributes, structure)
- **Goal**: generation of matching candidates (refined by user)

**Problem Definition**
- **Given**: two schemas $S_1$ and $S_2$
- **Goal**: Generate correspondences between $S_1$ and $S_2$
- **Correspondence**: relationship between $M$ elements in $S_1$ and $N$ elements in $S_2$
- **Mapping expression**: function how elements are related (directional or non-directional)
System Architecture and Matching Process

External Schemas

Model Pool

Matcher Configs

Mapping Pool

Matcher 1

Matcher 2

Matcher 3

{S_{11}, S_{12}, \ldots}

{S_{21}, S_{22}, \ldots}

Matched Schemas

Component Identification

Matcher Execution

Similarity Combination

Correspondences

Erhard Rahm, Hong-Hai Do David Aumueller, Sabine Massmann: Matching Large Schemas with COMA++, 2005
Classification of Matching Techniques

- **Schema-based**
  - **Element**
    - Linguistic: Name sim, description sim
  - **Structure**
    - Constraints: Type sim, key properties
    - Constraints: parent-child, graph structure

- **Instance-based**
  - **Element**
    - Linguistic: Word freq, key terms
    - Constraints: Value patterns and ranges

- **Cardinalities**: 1:1, 1:N, N:M
- **Combined Matchers** (hybrid, composite)

[Erhard Rahm, Philip A. Bernstein: A survey of approaches to automatic schema matching. VLDB J. 2001]
Selected Matchers

- **Linguistic Approaches**
  - Syntactic
    - Affix (suffix, prefix)
  - N-grams
  - EditDistance
  - Semantic
    - Synonyms
    - Hierarchy → taxonomies
    - Language → dictionaries

- **Example Trigram**

  \[
  \text{Similarity} = \frac{2|S_1 \cap S_1|}{(|S_1| + |S_2|)} = \frac{8}{16} = 0.5
  \]
Selected Matchers, cont.

- **Constraint-Based Approaches**
  - **Elements:** data types, domains, key attributes, constraints
  - **Structure:** relationships between elements, combinations of data types, neighborhood, specialization and composition

- **Example Similarity Flooding**
  - Create map pairs A x B (PCG)
  - Propagation coefficients: $1/|\text{out}(v)|$ (IPG)
  - Initial sim, adjusted by neighborhood sim until fixpoint (converged)

Selected Matchers, cont.

- **Problem Schema-based**
  - Attribute names might differ vastly (CustNa vs Name, CustSt vs Street)
  - **Instance-based matching**, but need for data

- **Linguistic Approaches**
  - Word frequencies, bigrams, trigrams, etc
  - Keywords, abbreviations

<table>
<thead>
<tr>
<th>Street</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luisenstrasse</td>
<td>Munich, 80333</td>
</tr>
<tr>
<td>Koenigsstrasse</td>
<td>Dresden, 01199</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Street</th>
<th>Num</th>
<th>ZIP</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weststrasse</td>
<td>2</td>
<td>01187</td>
<td>Dresden</td>
</tr>
<tr>
<td>Dammweg</td>
<td>45</td>
<td>12437</td>
<td>Berlin</td>
</tr>
</tbody>
</table>

- **Constraint-based Approaches**
  - **Elements**: data types and lengths, value domains, patterns
  - **Structure**: combinations of element constraints

<table>
<thead>
<tr>
<th>City</th>
<th>ZIP</th>
<th>5-letter numeric codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Munich, 80333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dresden, 01199</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Combination of Matchers

- **#1 Reuse**
  - Exploitation of transitive similarity
  - Fast and efficient matching, but potential for lost mappings (e.g., S3 misses attributes)

- **#2 Refinement**
  - **Chaining:** matcher $M_{n+1}$ works on mappings of $M_n$ → efficiency but potential for lost mappings
  - **Context-sensitive matching:** Matching context of schema components, matching elements within components

- **#3 Composite Matchers**
  - Select combination of complementary matchers in form of **ensemble**
  - **Aggregation** of similarity (e.g., trigram, synonym → avg/min/max)

- $S_1 \cong S_3 \land S_3 \cong S_2 \rightarrow S_1 \cong S_2$
Efficiency and Scalability

- **Problem**
  - Large schemas and matching complexity; all-pair problem $O(n \times m)$

- **#1 Early Search Space Pruning**
  - Faster matchers used to eliminate unlikely matches
  - Reduce schema sizes for expensive matchers

- **#2 Partition-based Matching**
  - Blocking into independent fragments
  - Match elements of similar fragment

- **#3 Parallel Matching**
  - Process different steps/fragments in parallel

- **#4 Custom Similarity Matrices**
  - Sparse matrices (adjacency lists) w/ nesting

[Philip A. Bernstein, Jayant Madhavan, Erhard Rahm: Generic Schema Matching, Ten Years Later. PVLDB 2011]

[Philip A. Bernstein, Sergey Melnik, Michalis Petropoulos, Christoph Quix: Industrial-Strength Schema Matching. SIGMOD Record 2004]
Exкурсус: Стабильная бракосочетание проблема

- **Проблема определение**
  - Для общего случая 1:1 соответствий, глобальная схема мержинга относится к стабильной бракосочетанию и проблемам больниц/местожительства

- **Пример стабильной бракосочетанию**
  - **Стабильное мержинг**: нет соответствия (A, B), предпочтительное для обеих A и B их текущие соответствия
  - **Входе би-безусловность сходство (предпочтение ранжирование)**
  - **Дефереди Акцептанс алгоритм**

while (!converged)
  #1 unmatched A propose to highest-ranked, unasked Bs
  #2 Bs accept highest-ranked proposal (even if matched)
Schema Matching Tools

- **Commercial Tools**
  - Most *message-oriented middleware, EAI, ETL* tools provide mapping UIs (w/ basic string similarity for matching)
  - Many data modeling tools also support matching/mapping

- **Academic Prototypes**

  ![Table]

  [Erhard Rahm: Towards Large-Scale Schema and Ontology Matching. Schema Matching and Mapping 2011]
Schema Matching Tools, cont.

- **COMA ++**
  - [Hong Hai Do, Erhard Rahm: COMA - A System for Flexible Combination of Schema Matching Approaches. *VLDB 2002*]
  - [David Aumueller, Hong Hai Do, Sabine Massmann, Erhard Rahm: Schema and ontology matching with COMA++. *SIGMOD 2005*]

- **COMA 3.0**
  - 2011/2012
  - Ontology Merging
  - Workflow Management

[Credit: https://dbs.uni-leipzig.de/de/Research/coma.html]
Schema Mapping

[Schema Detection]  [Schema Matching]  [Schema Mapping]

```sql
INSERT INTO Orders
SELECT BELNR, SUMME, DATUM
FROM Bestellungen
```
Schema Mapping Overview

- **Problem**
  - **Given:** two schemas w/ high-level mapping
  - Generate concrete transformation program/query (SQL, XSLT, XQuery)

- **Schema Mapping Process** (systematic lowering)
  - **High-level mapping:** intra- and inter-schema correspondences
  - **Low-level mapping:** mapping that ensures consistency with constraints of target schema and user intend (interpretation)
  - **Transformation query:** transformation program from one into the other schema, backend-specific (query generation)

- **Example**

  **Article**
  - artPK
  - title

  **Publication**
  - pubID
  - title
  - date

  **Transformation Query:**
  ```sql
  INSERT INTO Publication
  SELECT artPK, title, NULL
  FROM Article
  ```
Mapping Interpretations

- Without Intra-Schema Correspondences

<table>
<thead>
<tr>
<th>Article</th>
<th>Publication</th>
</tr>
</thead>
<tbody>
<tr>
<td>artPK</td>
<td>pubID</td>
</tr>
<tr>
<td>title</td>
<td>title</td>
</tr>
<tr>
<td>pages</td>
<td>date</td>
</tr>
</tbody>
</table>

```
INSERT INTO Publication
(SELECT artPK AS pubID, title AS title, NULL AS date, NULL AS author
FROM Article)
UNION ALL
(SELECT NULL AS pubID, NULL AS title, NULL AS date, name AS author
FROM Author)
```
Mapping Interpretations, cont.

- With Intra-Schema Correspondences

```
INSERT INTO Publication
SELECT artPK, title, NULL, name
FROM Article, Author
WHERE artPK = artFK

Note: Inner join acts as filter (no articles w/o authors, NOT NULL constraints)

INSERT INTO Publication
SELECT artPK, title, NULL, name
FROM Article LEFT OUTER JOIN Author
ON artPK = artFK
```
Mapping Interpretations, cont.

- From Denormalized to Normalized Form

**Skolem Function (SK):** unique key generation from tuple values $t[X]$ of an attribute set $X$

```
INSERT INTO Article
SELECT DISTINCT
  SK(title), title, NULL
FROM Publication

INSERT INTO Author
SELECT SK(title), author
FROM Publication
```
Schema Mapping Tools

- **Clio** (IBM Research – Almaden)


[Laura M. Haas, Mauricio A. Hernández, Howard Ho, Lucian Popa, Mary Roth: Clio grows up: from research prototype to industrial tool. SIGMOD 2005]
Summary and Q&A

- Schema Detection
- Schema Matching
- Schema Mapping

“Given the existence of all these tools, why is it still so labor-intensive to develop engineered mappings? To some extent, it is an unavoidable consequence of ambiguity in the meaning of the data to be integrated. If there is a specification of the schemas, it often says little about integrity constraints, units of measure, data quality, intended usage, data lineage, etc. Given that the specification of meaning is weak and the mapping must be precisely engineered, it seems hopeless to fully automate the process anytime soon. A human must be in the loop.”

[Philip A. Bernstein, Sergey Melnik: Model Management 2.0: Manipulating Richer Mappings. SIGMOD 2007]

Next Lectures (Data Integration Architectures)

- 05 Entity Linking and Deduplication [Nov 08]
- 06 Data Cleaning and Data Fusion [Nov 15]
- 07 Data Provenance and Blockchain [Nov 22]
Projects

Scripts, Algorithms, and Language APIs
- #1 Scripts for Cloud Deployment (AWS EMR, Azure HDInsight) → Florijan
- #2 2x Python Language Bindings (lazy eval, builtins, packaging)
- #3 Bayesian Optimization for Hyper-Parameter Optimization
- #4 Stable Marriage Algorithms in Linear Algebra
- #5 XSLT or JSON mapping UDFs (local, distributed)
- #6 Large-Scale Slice Finding for ML Model Debugging → Svetlana

Data Cleaning and Augmentation
- #7 Hidden Markov Models for Missing Value Imputation in NLP
- #8 Missing Value Imputation for Continuous/Categorical Columns
- #9 Time Series Outlier Removal and Preprocessing
- #10 Reconstruction of Aggregated Time Series
- #11 Data Augmentation for ML-based Data Cleaning (data corruption)
Projects, cont.

Schema Detection and Data Preparation

- #12 Inclusion and Functional Dependency Discovery (local and distributed)
- #13 Schema Detection from JSON and XML
- #14 Semantic Schema Detection (see Sherlock)
- #15 Feature Transform: Feature Hashing (local, distributed)
- #16 Feature Transform: Equi-Height/Custom Binning (local, distributed)

Compiler and Runtime (many more available)

- #17 Consolidated Cost Model for HOPs and Instructions (for lineage)
- #18 4x Basic Distributed Tensor Operations (distributed, federated)
- #19 Basic Sparse Tensor Representations (homogeneous/heterogeneous)
- #20 JSON/JSONL reader/writer into Data Tensor (local, distributed)
- #21 Protobuf reader/writer into Data Tensor (local, distributed)