Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

# Chapter 17
# Information Integration

*contributions by Jürgen Göres*



## Outline

- Information Integration Challenges
  - Distribution
  - Autonomy
  - Heterogeneity
- Schema Matching
  - Classification of Approaches
  - Example: Cupid
- Multidatabase Languages
  - SchemaSQL
  - FIRA/FISQL
- Integration Planning
  - Clio
- Dynamic Information Integration – Paladin
- Change Impact Analysis – Caro

# Integration Challenges

- Goal of Information Integration:
  *Provide a homogeneous, integrated view on multiple, distributed, autonomous and heterogeneous data sources.*
- Three fundamental challenges:
  - Distribution
  - Autonomy
  - Heterogeneity
- Orthogonal, but interrelated
- Techniques to handle distribution discussed in previous chapters
- In this chapter we focus on resolving heterogeneity



# Distribution

- Physical distribution
  - Data located on (geographically) separated systems
  - Challenges:
    - Addressing data across the globe (URLs)
    - Accessing data in different schemas (Multi-database languages, federated database systems)
    - Optimizing distributed queries (no topic of this lecture)
- Logical distribution
  - Several possible storage locations for a given data item
  - Caused by (partial) redundancy due to overlapping intension of schema elements
  - Challenges:
    - Maintaining consistency among redundant data
    - Provide metadata to enable data localization
    - Detect and resolve duplicates
    - Detect and resolve data inconsistencies and conflicts

    } Data Cleaning
- Physical and logical distribution are orthogonal:
  - Data can be logically distributed and physically on the same system (and vice versa)

# Autonomy

- Design Autonomy
  - Administrators of data sources can freely decide in which way they model data
  - Data model, formats, units, …
  - Leads to heterogeneity among sources
- Interface Autonomy
  - Freedom to decide how technical access is provided
  - Protocols (HTTP, JDBC, SOAP, …), supported query languages (SQL, XQuery, …)
- Access Autonomy
  - Freedom to decide *whom* to allow access to *what* data
  - Mode of Authentication (Certificates, Username/Password)
  - Authorization (boolean, R/W, Access Control Lists, …)
- Judicial Autonomy
  - Freedom to prohibit integration of data by others
  - Intellectual property (IP) issues

⇨ Autonomy is the major cause of integration problems



Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

# Forms of Heterogeneity

# Heterogeneity

- Translated from [LeNa07]:
  *"Two information systems that do not provide the exact same methods, models and structures to access their data are called heterogeneous."*
- Causes for heterogeneity among IS:
  - Specific requirements
  - Independent development
  - Developer preferences
  - ...

  ⇨ All aspects result from autonomy
- Heterogeneity of metadata *and* data
- Two main approaches:
  - Try to resolve heterogeneity when needed
  - Enforce homogeneity/limit heterogeneity by establishing standards (not in this lecture)
    - No real solution to the problem
    - Only creates "spheres of homogeneity", any participants that have existing systems or requirements not conforming to the standards have to resolve heterogeneity locally



# Technical Heterogeneity

- Refers to differences in the options to access data, e.g.
  - communication protocols (HTTP, SOAP, …)
  - Exchange formats (binary, text, XML, …)
  - APIs (JDBC, ODBC, proprietary)
  - Query mechanism
    - Forms, canned queries
    - Query languages
  - Query language
    - SQL, XQuery, …

## Data Model Heterogeneity

- Caused by the use of different data models among data sources
  - hierarchical, relational, XML, …
- Data models can have different expressiveness, e.g. support of
  - Inheritance
  - Types and degree of associations between entities/application concepts
  - Multi-valued attributes
  - Different atomic data types
- Mapping from semantically richer to poorer models in general results in a loss of information
- Approaches to bridge data model heterogeneity
  - SQL/XML (Chapter 13)
  - Wrappers/Mediators (Chapter 9)



## Syntactic Heterogeneity

- Differences in the representation of identical facts
  - Binary representations (little/big endian, number formats)
  - Encodings (ASCII, ISO-8859-1, EBCDIC, Unicode, …)
  - Separators (Tab-delimited vs. CSV)
  - Textual representation
- Not to be mixed up with semantic heterogeneity!
- Usually easy to resolve (if used consistently)
- Examples:
  - "20070201" vs. "Februar 1st, 2007" vs. "02-01-07"
  - "123.45" vs. "1.2345x10$^2$"

➔ *Data Fusion*

# Structural Heterogeneity

- Caused by modeling identical application concepts differently using the *same* elements in the same data model
- Example - denormalized relational schema

**Employee**

| EmpNo | Name | DoB | DeptNo |
|---|---|---|---|
| 4711 | Bob | 1978-03-20 | 11 |
| 0815 | Jane | 1975-11-05 | 7 |
| 1234 | Joe | 1954-05-26 | 11 |

**Department**

| DeptNo | Name |
|---|---|
| 7 | Sales |
| 11 | Accounting |

⟺

**EmpDept**

| EmpNo | Name | DoB | Deptname | DeptNo |
|---|---|---|---|---|
| 4711 | Bob | 1978-03-20 | Accounting | 11 |
| 0815 | Jane | 1975-11-05 | Sales | 7 |
| 1234 | Joe | 1954-05-26 | Accounting | 11 |

- Easily resolved using relational operators:

```
SELECT e.EmpNo, e.Name, e.DoB, d.name as deptname, d.deptno
FROM Employee e, Department d WHERE e.deptno = d.deptno
```



# Structural Heterogeneity (cont.)

- Example: inverted hierarchy

```
<bib>
  <book title="a">
     <author name="x"/>
     <author name="y"/>
  </book>
  <book title="b">
     <author name="x"/>
  </book>
</bib>
```

⟺

```
<bib>
  <author name="y">
    <book title="a"/>
  </author>
  <author name="x">
    <book title="a"/>
    <book title="b"/>
  </author>
</bib>
```

- Easily resolved using XQuery

```
<bib> {
   for $a in distinct-values(doc("BookAuthor.xml")//author/@name)
   return <author name="{$a}"> {
      for $b in doc("BookAuthor.xml")//book
      where $b/author/@name = $a
      return <book title="{$b/@title}"/>
      } </author>
} </bib>
```

# Schematic Heterogeneity

- Often considered a special case of structural heterogeneity
- Caused by modeling identical application concepts using *different* data model concepts of the same data model
- Example: *attribute value – relation name* conflict

*categorical attribute*

**Person**

| ID | Name | Gender |
|---|---|---|
| 1234 | Bob | male |
| 4567 | Jane | female |

**Men**

| ID | Name |
|---|---|
| 1234 | Bob |

**Women**

| ID | Name |
|---|---|
| 4567 | Jane |

- Problems of this kind cannot be resolved *generically* with SQL
  - How to handle an unknown/variable number of values for categorical attributes?

➔ *Multi-database languages*



# Semantic Heterogeneity

- "Semantics" = interpretation of data and metadata
- Different representation of identical application concepts, (e.g. synonyms)
- Identical representation of different application concepts (e.g. homonyms)
  - e.g. Lotus (the car) vs. Lotus (the flower)
- Ambiguities – unclear whether two elements refer to the same concept (are synonyms) or refer to broader/narrower terms (hypernyms)
  - hypernym or synonym?
    - car – (motor) vehicle
    - person – employee
    - product – item
  - decision depending on context
- Perhaps *the* biggest challenge in II
- Resolving semantic heterogeneity is a prerequisite for many integration tasks
- Many attempts to automate

➔ *Schema Matching*

# Bridging/Resolving Heterogeneity

- Real-world integration scenarios suffer from all kinds of heterogeneity
- Techniques and concepts already discussed in previous chapters and the primary issues they address:
  - Wrappers (data model heterogeneity, technical heterogeneity, syntactic heterogeneity)
  - Garlic (technical heterogeneity, structural heterogeneity, distribution)
  - SQL/XML (data model heterogeneity)
  - DB Gateways (technical heterogeneity)
  - ETL tools (structural heterogeneity, technical heterogeneity, syntactic heterogeneity)
  - ⇨ focus on data access/transformation infrastructure (i.e., as a runtime platform)
- Further techniques discussed in this chapter
  - Schema Matching and Integration (semantic heterogeneity, structural heterogeneity)
  - Multi-database languages (schematic heterogeneity, technical heterogeneity, distribution)
  - [Data Cleaning/Fusion (syntactic heterogeneity, semantic heterogeneity (in data))]
  - ⇨ focus on integration planning, resolving schematic heterogeneity



# Information Integration Tasks

- Information integration subsumes numerous tasks (and has numerous names for most of them...):
  1. Schema Merging/Schema Integration
  2. Design of the integrated target schema
  3. Schema Matching/Schema Mapping
  4. Integration Planning/Schema Mapping/Schema Integration/Mapping Generation/Mapping Interpretation
  5. Data Cleaning
  6. Data Fusion/Record Matching/Entity Resolution/Instance Disambiguation
  7. Wrapping/Data model transformation

## Information Integration Phases [Gö05b]

- Analysis – Determine the requirements on the integrated schema:
  - Desired data model, integration strategy (virtual or materialized)
  - Relevant data (which application concepts should be present)
- Discovery – Find/identify relevant data sources
  - In classical scenarios sources are often known implicitly
  - Challenging aspect of ➔ Dynamic information integration
- Planning – Resolve heterogeneity
  - Technical heterogeneity (enable access to sources)
  - Semantic heterogeneity ➔ Schema Matching
  - Data model, structural and schematic heterogeneity
    ➔ develop data transformation specification (integration plan)
- Deployment
  - Set up integration plan in a runtime environment that provides the integrated data
  - e.g., federated DBMS, data warehouse, stylesheets, scripts
- Runtime
  - React to changes in the data sources/requirements



## Information Integration Approaches

- Bottom-up design
  - Used to completely integrate a well-known set of data sources
  - Assumes that changes of the number and properties of the data sources are rare
  - Integrated schema is created based on the data sources (➔ *Schema Merging*)
  - No distinguished discovery and analysis phases
  - Common in enterprise integration scenarios
- Top-down design
  - Used when the available data sources are not known a priori
  - The number and properties of candidate data sources for integration are changing constantly
  - Integrated schema is designed independently from the sources, based only on the application requirements
  - Analysis phase precedes discovery phase
  - ➔ *Dynamic Information Integration*
- Hybrid design
  - Selection of data sources based on requirements
  - Design of integrated schema influenced by requirements and data source schemas
  - Analysis and discovery are intertwined

Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

# Schema Matching



## Schema Matching

- Goal: Identify semantically related elements across different schemas
- Schema element: table, column, element, attribute, class, etc.
- Result: set of *matches* or *(value) correspondences* (a *mapping*)
- Essential preparation step for most subsequent integration tasks
- Different expressiveness of correspondences
  - Match Degree (also: *local cardinality*)
    - 1:1 semantic relationship of one element of schema A with one element of schema B
    - 1:n semantic relationship of one element of schema A with a set of elements of schema B
    - n:m semantic relationship between sets of elements from schemas A and B
  - Match Semantics
    - Basic matches do not carry additional semantics, they only indicate "some relationship"
    - Advanced matches can indicate abstraction concepts (inheritance, composition, etc.) or functions (e.g., "A is equivalent to the sum of $B_1$ and $B_2$")
- "Higher order" correspondences
  - Connect different types of schema elements (e.g. a department table corresponding to a department attribute)
  - Connect metadata with data (e.g., categorical attributes)
- Does *not* refer to the relationship between the instances of the matched concepts (e.g. instances are identical/subsumed/disjoint/overlap)

## Schema Matching – Terminology Disambiguation

- Mapping
  - **A set of correspondences between two schemas**
  - The process of creating a set of correspondences (➔ schema matching, see below)
  - But also
    - A function or transformation describing how data is transformed (➔ Integration plan)
    - The process to create a function/transformation (➔ Integration planning)
- Schema Matching
  - **The process of obtaining a mapping**
  - An *automatic* process to obtain a mapping

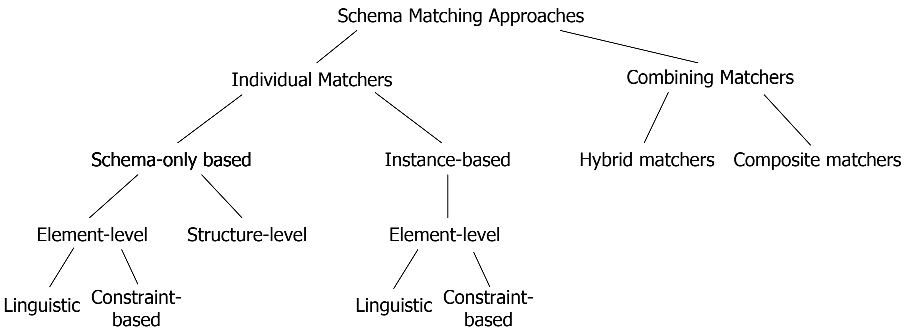


## Schema Matching – Challenges

- Identification of matches difficult
  - Very large schemas ($10^2$-$10^3$ relations, $10^3$-$10^4$ attributes)
  - Complex schemas
  - Initially unknown and undocumented schemas
  - Ambiguities (Synonyms, Hypernyms, Abbreviations, …)
  - Foreign languages
  - Cryptic identifiers
- Time-consuming and expensive
  - Element-wise "comparing" a schema A with *n* elements with a schema B with *m* elements requires $n \cdot m/2$ comparisons
  - For $n \approx m$: $O(n^2)$
  - Even higher complexity if sets of elements are compared ($O(2^{2n})$), e.g. to obtain 1:n/n:m matches ➔ practical approaches limit sets to a maximum size k

➔ Numerous approaches to automate schema matching
  - Error-prone (false-positives and false-negatives)
  - At best semi-automatic (for good results, domain experts must review, amend and revise matches)
  - ➔ Used as a preparation step for a human domain expert to reduce search space

# Schema Matching – Classification of Approaches

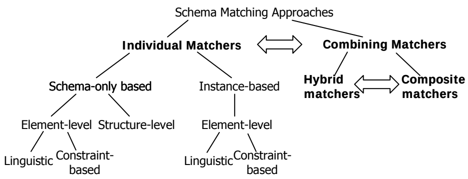



based on [RaBe01]



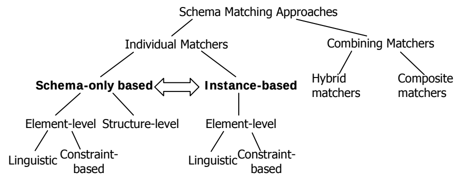
# Individual vs. Combining Matchers

- Individual matchers exploit only one kind of information for identifying matches
- Combining matchers use several:
  - Hybrid:
    - Different approaches "hard-wired" into one (parameterizable) component to create a single mapping between the schemas
    - Reuse of individual elements in combination with other matchers or extension with new concepts and approaches to matching is difficult
  - Composite
    - Retroactively combine mappings from different (individual and combining) matchers
    - Common methods: (weighted) average, max, min



# Schema-only vs. instance-based matching

- Schema-only techniques operate solely on metadata:
  - table/column/element/attribute/… identifiers and comments or annotations
  - data types
  - constraints
  - element structuring
- Instance-based techniques also consider properties of the data
  - Can only be used *among* data sources
  - In order to use with target schema, sample data can be provided
  - Uses statistical information on data values
    - Actual value ranges of attribute values (e.g., ints in the interval [0,120])
    - Enumeration of values actually present in the data
    - Histograms (Number of occurrences of individual attribute values)
    - Regular expressions describing value patterns (e.g. `[0..9]{5}` for German zip codes)






# Linguistic Matching – String Similarity

- String distance or similarity measures [CRF03]
- Based on the lexical similarity of schema element identifiers
- Often used after applying string preprocessing techniques
  - Tokenization: split identifiers based on case, punctuation, etc.
  - Stemming: reduce identifiers to word stem (e.g. "computer" ➔ "comput")
    Note: Stemming algorithms are language-dependent (for English: Porter's algorithm)
  - Stopword elimination
- Edit-distance-like functions, e.g.
  - Levenshtein distance:
    - Count the number of edit operations (insert, modify, delete) to turn string a into string b
    - Example:
      `kitten`
      `sitting`
      ➔ 2 replacements, 1 insertion LevenshteinDist("kitten","sitting") = 3
    - Weighting of operations possible (e.g. replace more expensive than delete)
    - Normalization to interval [0,1] by dividing result through max(length(String A), length(string B))
  - Other measures: Monge-Elkan, Jaro-Winkler, ...

# Linguistic Matching – String Similarity (cont.)

- Token-based functions, e.g.
  - Applied on sets of tokens of identifiers
    - Tokenization based on word separators (white space, punctuation, special characters, case)
      - e.g. "Web-of-trust" ➔ {"Web", "of", "trust"}, "CamelCaseIdentifier" ➔ {"Camel", "Case","Identifier"}
    - Tokenization based on n-grams
      - Tokens created by sliding a window of size n over the string
      - e.g. 3-grams for "Information" ➔ {"Inf", "nfo", "for", "orm", "rma", "mat", "ati", "tio", "ion"}
  - Jaccard similarity – describes the similarity of two sets

$$\text{JaccardSimilarity}(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

    - Example:
      ProductPrice ➔ A= {Product, Price}, PriceOfProduct ➔ B = {Price, Product, Of}
      JaccardSimilarity(A,B) = 2/3
  - TFIDF (Term frequency/inverse document frequency) methods
    - Measure originally developed for information retrieval
    - Here: document = (tokenized) identifier, term = token
    - Determines a weight $w_s(t)$ for each token t of a string S based on its frequency in the identifier (term frequency, $tf_S(t)$) and the inverse of its frequency in all identifiers (inverse document frequency, idf(t))
    - Idea: Tokens occurring frequently in the string S have a high weight, while tokens occurring in almost every string receive a low weight
    - Basic weight formula: $w_s(t) = tf_s(t) \cdot idf(t)$



# Linguistic Matching – String Similarity (cont.)

  - TFIDF (continued)
    - Many different approaches to calculate $tf_S(t)$ and idf(t)
    - e.g., with $n_{S,x}$ being the number of occurrences of term x in document S, T being the set of all terms in S, N being the total number of documents, and $N_t$ being the number of documents that contain term t (at least once):

$$tf_s(t) = \frac{n_{S,t}}{\max_{i \in T}(n_{S,i})} \qquad idf_s(t) = \log_e\left(1 + \frac{N}{N_t}\right)$$

    - Identifiers can be interpreted as vectors in n-dimensional space (with n being the number of different tokens), with the term weights $w_s(t)$ as vector components/elements
    - The similarity between the identifiers is the similarity of the direction (ignoring length) of their respective vectors, i.e., the greater the angle between their vectors, the smaller the similarity
    - Applying the cosine on the angle, we normalize the difference in angle to [0,1]: for an angle of 0°, the cosine is 1 (maximum similarity), for an angle of 90° the cosine is 0
    - Then the similarity function between two identifiers $S_1$ and $S_2$ is defined using the cosine measure

$$\text{cosine}(S_1, S_2) = \frac{\sum_{t=1}^{n} w_{S1} \cdot w_{S2}}{\sqrt{\sum_{t=1}^{n} w_{S1}(t)^2} \cdot \sqrt{\sum_{t=1}^{n} w_{S2}(t)^2}}$$

- Hybrid approaches
  - use a secondary similarity function to determine similarity between tokens
- Problem of all approaches based on lexical similarity:
  - Lexical similarity does not necessarily indicate semantic similarity! (and v.v.)

# Linguistic Matching – Ontology-based approaches

- Use a Dictionary/Thesaurus/Ontology[1] to store knowledge about application domain terms and concepts and their relationships, e.g.
  - Synonymy
  - Hypo/hypernymy, sub/superclasses
  - Aggregation
  - Opposite terms/concepts
- Can contain alternative forms for terms (word stem, abbreviations)
- Distance of two terms within the thesaurus is translated to similarity value
- Can be extended to handle different languages
- Ontologies can be domain-specific or generic and vary in the level of detail
  - Design of a good ontology is a daunting task
  - Depending on their specific point of view and their level of detail, ontologies will often disagree on terms and their relationships, e.g.:
    Is "car" a special type of "vehicle" (hyponym), or are the terms synonyms?

1 These and similar terms are not used consistently throughout the literature. See e.g. http://www.metamodel.com/article.php?story=20030115211223271 for an attempt at a definition of these terms.




# Structural Schema Matching

- Exploit the relationships (structure) among schema elements to improve the quality of matches
- Usually require an initial set of correspondences provided by (non-structural) schema matchers
  → Practical implementations are usually hybrid matchers (although they could be built as combining matchers)
- Examples:
  - Cupid [MBR01]
  - Similarity Flooding [MGR02]

# Cupid

- Developed by Microsoft Research [MBR01]
- Hybrid approach:
  - Element-based: linguistic and data type similarity
  - Structure-based: *TreeMatch* algorithm
- Three phases
  - Linguistic matching
    - Determine initial matches based on schema element identifiers
  - Structure matching
    - Modify initial values based on element structure
  - Creation of mappings/matches
    - Choose the matches to return as result
    - Method depends on the intended use for the matches, e.g.
      - Prune matches below a given threshold
      - Return only leaf-level matches



# Cupid Linguistic Matching

1. Normalization
   - Tokenization: split identifiers into tokens based on punctuation, case, etc.
     e.g. POBillTo ⇨ {PO, Bill, To}
     five token types: number, special symbol, common word, concept, content
   - Expansion: expand acronyms with the help of a thesaurus/dictionary
     e.g. Qty ⇨ Quantity
   - Elimination: discard prepositions, articles, etc. with the help of a stop word list
     e.g. {PO, Bill, To} ⇨ {PO, Bill}
   - Tagging: identifiers related to a known application concept are tagged with the concept
     e.g. identifiers *Price*, *Cost* and *Value* are tagged with the concept *Money*
2. Categorization
   - Clusters elements into categories (= a group of elements identified by a set of keywords)
   - Goal: reduce comparisons to only those elements within compatible categories
   - One category for each:
     - Concept tag
     - Data type (coarse grained, e.g., number, string, date, ...)
     - Container (e.g., address contains city, state, and street)
   - Elements can belong to multiple categories
   - Categories are compatible, if their respective sets of keywords are "name similar"

# Cupid Linguistic Matching (cont.)

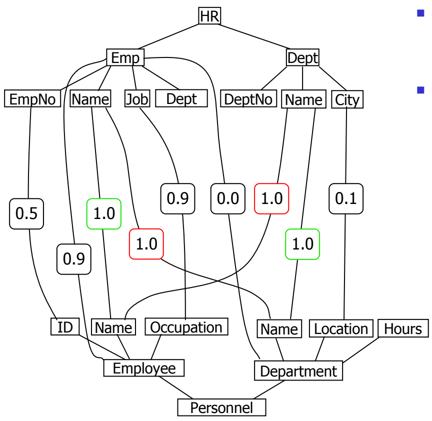
- Name similarity:
  - The *name similarity* of two token sets $T_1$ and $T_2$ is the average of the best similarity of each token in set $T_1$ with a token in set $T_2$
  - To determine the similarity of two tokens $t_1$ and $t_2$, a thesaurus lookup is performed
  - If no thesaurus entry is present for a pair of tokens, substring matching is used to identify common pre- and suffixes

3. Comparison
   - Determines the linguistic similarity coefficient $lsim(s,t)$ $s \in S$, $t \in T$, for pairs of elements of the two schemas S and T
   - For each pair of elements s, t from compatible categories
     1. Calculate the name similarity of the element tokens *per token type*
     2. Calculate the weighted mean of the per-token-type name similarity (concept and content tokens are assigned a higher weight)
     3. Calculate lsim for the pair by scaling the result of 2. with the maximum name similarity of the categories of s and t

- Result: a table of linguistic similarity coefficients $lsim(s,t)$ in the range [0,1]



# Cupid Linguistic Matching – Problems




(not all matches shown)

- Linguistic matching does not consider context: e.g., false positive: Emp/Name is as similar to Employee/Name as it is to Department/Name
- Linguistically dissimilar, but semantically related elements are underrated (caused by missing or incomplete thesaurus)
  e.g. Dept/City – Department/Location

# Cupid Structural Matching

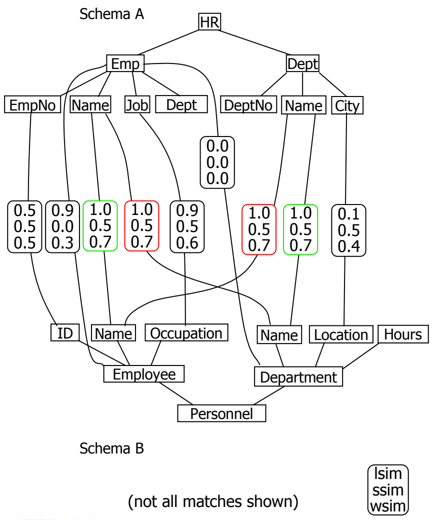
- Based on a tree representation of the structure of the schema
- *TreeMatch* algorithm
- Basic intuitions
  1. A pair of leaves from two trees is similar, if
     a) they are individually similar (linguistic, data type, …)
     b) their neighbors (ancestors and siblings) are similar
  2. A pair of non-leaves is similar, if
     a) they are linguistically similar
     b) their subtrees are similar
  3. A pair of non-leaves is structurally similar, if their respective leaves are highly similar (not necessarily their direct children)
- Initialize *ssim* for all leaves using a data type compatibility matrix (range [0,0.5])
- *Stronglink*: similarity between two leaves is above threshold $th_{accept}$
  - based on wheighted similarity (see next chart)



# Cupid Structural Matching (cont.)

- Iterate over the tree nodes in post-order (bottom-up calculation)
- For each pair s,t:
  - Calculate ssim(s,t) as the fraction of leaves in the two subtrees below s and t that have at least one stronglink to a leaf in the other subtree
  - Calculate a weighted similarity measure wsim(s,t):
    $wsim(s,t) = w_{struct} \cdot ssim(s,t) + (1-w_{struct}) \cdot lsim(s,t)$
  - If wsim(s,t) is above threshold $th_{high}$, increase the similarity of each pair of leaves in the subtrees of s and t by a factor $c_{inc}$ (not exceeding 1)
  - If wsim(s,t) is below threshold $th_{low}$, decrease the similarity of each pair of leaves in the subtrees of s and t by a factor $c_{dec}$ (but never below 0)

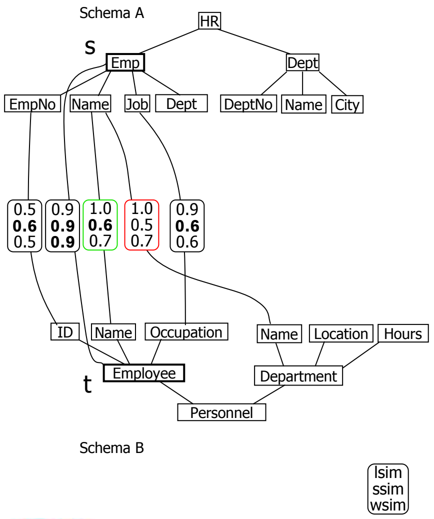
# Cupid Structural Matching – Example




- Initialization:
  - ssim set to 0.0 for all non-leaf nodes
  - ssim set to data type similarity for leaves
- Parameters:
  - $th_{accept} = 0.5$
  - $w_{struct} = 0.7$
  - $th_{high} = 0.7$, $c_{inc} = 1.2$
  - $th_{low} = 0.3$, $c_{dec} = 0.8$

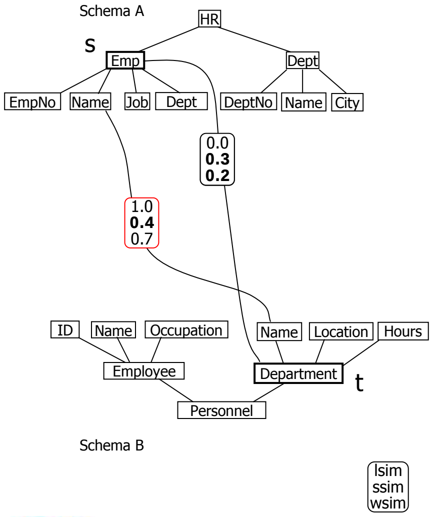


# Cupid Structural Matching – Example (cont.)




- Iteration for
  s = Emp, t = Employee:
  - Calculate ssim:
    3 out of 4 leaves of Emp have stronglinks to leaves of Employee, 3 out of 3 leaves of Employee have stronglinks to Emp
    $ssim(s,t) = 6/7 \approx 0.9$
  - Calculate wsim:
    $wsim(s,t) = w_{struct} \cdot ssim(s,t) + (1-w_{struct}) \cdot lsim(s,t)$
    $= 0.7 \cdot 0.9 + 0.3 \cdot 0.9 = 0.9$
  - Modify structural similarity for leaves of s and t:
    $wsim(s,t) = 0.9 > th_{high} = 0.7$
    ⇨ increase ssim for each pair $(l_s, l_t)$,
    $l_s \in leaves(s)$ and $l_t \in leaves(t)$:
    $ssim_{new}(l_s,l_t) = ssim_{old}(l_s,l_t) \cdot c_{inc} = 0.5 \cdot 1.2 = 0.6$
    (wsim for leaf-pairs is left unchanged)
- Result:
  - Similarity between s and t increased, because children are similar (intuitions 2b and 3)
  - Similarity between the child nodes increased, because their neighbors (here: ancestors) are similar (intuition 1b)

# Cupid Structural Matching – Example (cont.)




- Iteration for
  s = Emp, t = Department:
  - Calculate ssim:
    ssim(s,t) = 2/7 ≈ 0.3
    (1 out of 4 leaves of Emp have stronglinks to leaves of Department, 1 out of 3 leaves of Department have stronglinks to leaves of Emp)
  - Calculate wsim:
    $wsim(s,t) = w_{struct} \cdot ssim(s,t) + (1-w_{struct}) \cdot lsim(s,t)$
    $= 0.7 \cdot 0.3 + 0.3 \cdot 0.0 = 0.21 \approx 0.2$
  - Modify structural similarity for leaves of s and t:
    $wsim(s,t) = 0.2 < th_{low} = 0.3$
    ⇨ decrease ssim for each pair $(l_s,l_t)$,
    $l_s \in$ leaves(s) and $l_t \in$ leaves(t):
    $ssim_{new}(l_s,l_t) = ssim_{old}(l_s,l_t) \cdot c_{dec}$
    (wsim for leaf-pairs is left unchanged)
- Result:
  - Similarity between Emp/Name and Department/Name decreased, because their ancestors are not similar



# Cupid – Summary

- TreeMatch exploits a schema element's context to modify similarity values
- Helps to discern between pairs that were rated identical by linguistic matching:
  - Confidence of false positives reduced:
    - Match confidence between leaves with dissimilar ancestors decreases
    - Match confidence of linguistically similar non-leaves with different children decreases
  - Confidence of false negatives or uncertain matches increased
    - Match confidence of leaf-pairs with similar ancestor increases
    - Match confidence of linguistically dissimilar non-leaves with similar children increases

Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

# Multi-database languages/ Schematic Query Languages



## Limitations of SQL

- Standard SQL is unable to generically solve most forms of schematic heterogeneity
- Comp. Person – Men/Women example

Schema A

**Person**

| ID | Name | Gender |
|---|---|---|
| 1234 | Bob | male |
| 4567 | Jane | female |

⟺

Schema B

**Men**

| ID | Name |
|---|---|
| 1234 | Bob |

**Women**

| ID | Name |
|---|---|
| 4567 | Jane |

- Can be solved with relational view(s)...

A to B

```
CREATE VIEW Men AS
SELECT ID, Name
FROM Person
WHERE Gender='male'

CREATE VIEW Women AS
SELECT ID, Name
FROM Person
WHERE Gender='female'
```

⟺

B to A

```
CREATE VIEW (ID, Name, Gender)
AS
SELECT ID, Name, 'male'
FROM Men
UNION
SELECT ID, Name, 'female'
FROM Women
```

- ... but only because the number of different "categories" (here: genders) is known a priori (and fixed)

# Limitations of SQL (cont.)

- e.g., replace gender with department:

Schema A

**Person**

| ID | Name | Department |
|---|---|---|
| 1234 | Bob | Accounting |
| 4567 | Jane | Sales |
| **9876** | **Joe** | **Service** |

Schema B

**Accounting**

| ID | Name |
|---|---|
| 1234 | Bob |

**Sales**

| ID | Name |
|---|---|
| 4567 | Jane |

**Service**

| **ID** | **Name** |
|---|---|
| **9876** | **Joe** |

- Departments might change over time
- When using static views as before
  - Each new department in A requires its own view definition to transform to schema B
  - Each new department in B requires a modification of the view to transform to schema A
- ➔ Expensive maintenance

A to B

```
CREATE VIEW Accounting AS
…
CREATE VIEW Sales AS
…
CREATE VIEW Service AS
SELECT ID, Name
FROM Person
WHERE Department ='Service'
```

B to A

```
CREATE VIEW (ID, Name, Department)
AS
SELECT ID, Name, 'Accounting'
FROM Accounting
UNION
SELECT ID, Name, 'Sales'
FROM Sales
UNION
SELECT ID, Name, 'Service'
FROM Service
```



# Schematic Query Languages

- Solution: Extend SQL to be able to transform data to metadata (and v.v.)
- ➔ Schematic Query Languages (a.k.a. Multi-database QLs)
- Examples
  - SchemaSQL
  - FIRA/FISQL
- Challenge:
  - The schema of the result of a query is now dependent on the data actually present in the input relations
  - To allow such *dynamic schemas*, schematic query languages have to extend the relational model
- In addition, schematic query languages provide mechanism to access different databases in a single query

# Example Databases

**Kaiserslautern (KL)**

**Sales**

| Store | Department | AvgSales |
|---|---|---|
| Innenstadt | TV | 139000 |
| Innenstadt | Computer | 156000 |
| Innenstadt | Hifi | 118000 |
| Gewerbegbt | TV | 112000 |
| Gewerbegbt | Computer | 180000 |
| Gewerbegbt | Hifi | 57000 |

**Mannheim (MA)**

**AvgSales**

| Store | TV | Computer | Hifi |
|---|---|---|---|
| Quadrate | 205000 | 234000 | 108000 |
| Kaefertal | 90000 | 76000 | 87000 |
| Sandhofen | 73000 | 81000 | 98000 |

**Trier (TR)**

**Eisenbahnstr**

| Dept | AvgSales |
|---|---|
| TV | 67000 |
| Computer | 51000 |
| Hifi | 78000 |

**Hauptstr**

| Dept | AvgSales |
|---|---|
| TV | 74000 |
| Computer | 103000 |
| Hifi | 89000 |



# SchemaSQL

- Lakshmanan, Sadri & Subramanian [LSS96, LSS01]
- First approach addresses the issue of schematic heterogeneity with SQL
- Built on top of SQL by providing an extended FROM clause:
  - Specify range variables ("aliases") not only over tuples of relations, but also over
    - the databases of the (M)DBMS `->`
    - the relation names of a database `db->`
    - attribute names of a relation `db::rel->`
    - tuples of a relation (-> SQL) `db::rel`
    - distinct values of an attribute `db::rel.attr`
  - Elements of the FROM clause can be nested, e.g.
    `FROM xdb-> xdbtables, xdbtables-> atts`
    to iterate over the relations of database xdb and then over the relations' attributes
  - Variables in the FROM clause can be used in view definitions for dynamic result schemas

# SchemaSQL – Example

- Transform KL database to MA format:

```
CREATE VIEW KL2MA::AvgSales(Store, KD) AS
SELECT KS.Store, KS.AvgSales
FROM KL::Sales KS, KS.Department KD
```

1. Dynamic result schema: number of attributes depends on number of attribute values in the source relation's department attribute
2. Nesting of sets in FROM clause
3. A source tuple's value for AvgSales is placed in the result column depending on the value of the tuple's Department attribute (merge into one result tuple is implicit)

- Problem: Operation (the merge) is not well-defined for all source relations
  - What happens if there was an additional tuple ("Innenstadt","Hifi", 97500) in the KL database? Which value (11800 or 97500) to place into the "Hifi" column?
  - SchemaSQL does not answer this question



# SchemaSQL – Example (cont.)

- Aggregation over a variable number of columns
- e.g. "What are the average sales of the Mannheim stores, across all departments?"
- Number of departments cannot assumed to be fixed!

```
SELECT MS.Store, AVG(MSAtts)
FROM MA::AvgSales MS, MA::AvgSales-> MSAtts
WHERE MSAtts<>'Store'
GROUP BY MS.Store
```

- Use of attribute set in aggregate function

Mannheim (MA)

**AvgSales**

| Store | TV | Computer | Hifi |
|---|---|---|---|
| Quadrate | 205000 | 234000 | 108000 |
| Kaefertal | 90000 | 76000 | 87000 |
| Sandhofen | 73000 | 81000 | 98000 |

# SchemaSQL – Criticism

- Semantics of a SchemaSQL SELECT statement differs depending on context:
  - e.g., query from Example 2, placed in a view definition:

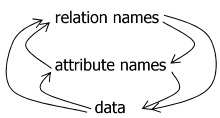
```
CREATE VIEW MA::PerDeptAvgs(Store, MSAtts) AS
    SELECT MS.Store, AVG(MSAtts)
    FROM MA::AvgSales MS, MA::AvgSales-> MSAtts
    WHERE MSAtts<>'Store'
    GROUP BY MS.Store
```

- Query now computes the averages for each department *individually*!




# FIRA/FISQL

- Presented by Wyss and Robertson [WyRo05]
- Extends the relational model to the *federated relational model*
  - Number of output relations and their attributes is fully dynamic
- Provides an extended SQL syntax (Federated Interoperable SQL, FISQL)
- Provides a sound theoretical foundation by specifying the underlying algebra operators (Federated Interoperable Relational Algebra, FIRA)
- FIRA/FISQL is *transformationally complete*:
  - Transform any form of relational metadata to data and v.v.




- FISQL allows nesting of queries

# FIRA/FISQL Data Model

- *Federated* relational data model:
  - Extends the relational model to incorporate metadata
    - A federated tuple is a mapping from a finite set of names S (=attribute names) to values; S is known as the schema of the tuple.
    - A federated relation has a name and contains a finite set of federated tuples
    - A federated database has a name and consists of a finite set of federated relations
    - A federation consists of a finite set of federated databases
    - The schema of a federated relation is the union of the schemas of the tuples
      - → Operations that add/change/delete tuples may modify the relation schema
- Defines federated counterparts of the six standard relational operators, e.g.
  - Renaming of relations (in addition to attributes)
  - Cartesian product/union/difference of databases
- Introduces six new operators
- Most operators defined on federated relations and on federated databases, i.e. operators take a relation/database as input and produce a relation/database as output



# FIRA/FISQL – Operators

- Drop-projection $\not\pi_A(R)$, $\not\pi_A(D)$
  - Two variants: one for relations, one for federated databases
  - Parameter A is the set of attributes to be *removed* from the relation/fed. DB
  - Required to generically handle relations/fed. DBs with variable schema
- Down $\downarrow_I(R)$, $\downarrow_I(D)$
  - Two variants: one for relations, one for federated databases
  - "Demotes" a table R's metadata to data by creating a relation *metadata$_i$* and forming its crossproduct with R.
  - For a relation R with name N and attributes $A_1 \dots 1_n$, the relation *metadata$_i$* is defined as:
  - Ignores metadata columns: $\downarrow_i(R) = metadata_i(R) \times \not\pi_{r_i,a_i}(R)$

metadata$_i$(R)=

| $r_i$ | $a_i$ |
|---|---|
| N | $A_1$ |
| N | $A_2$ |
| ... | ... |
| N | $A_n$ |

- Attribute Dereference $\Delta_A^B(R)$
  - The value of attribute B of the target tuple t is determined by using the value found in the attribute named equal to t's value in column A, values of all other attributes of t are equal to the respective value of those in source tuple s
  - Let t[X] denote the value of attribute X of tuple t. The attribute values of a result tuple t are obtained from the values of its respective source tuple s like this:

$$t[X] = \begin{cases} s[s[A]] \text{ if } X = B \\ s[X] \text{ otherwise} \end{cases}$$

# FIRA/FISQL – Operators (cont.)

- Generalized Union $\Sigma(D)$
  - Creates a relation holding the outer union of all relations in the database D
- Transpose $\tau_A^B$ (R)
  - For each distinct value of the parameter column B in the input relation R, create a column in the result relation (whose name is the respective value of B)
  - For each tuple t of the result relation, obtain the value of column X (denoted t[X]) from the respective source tuple s like this:

$$t[X] = \begin{cases} s[A] \text{ if } X = s[B] \\ s[X] \text{ if } X \in \text{schema(s)} \\ \text{NULL otherwise} \end{cases}$$

  - i.e.: for each new attribute $N_i$, its value is that of the source tuple's A attribute if the source tuple's B attribute value is equal to the name of attribute $N_i$, NULL otherwise
  - other attributes remain unchanged
- Partition operator $\wp_A(R)$
  - Roughly the opposite of Generalized Union
  - Creates a federated database with one relation for each distinct value in column A of input relation R




# FIRA/FISQL example – KL2TR

- Transform the Kaiserslautern database to the format of the Trier database
- Requires the Partition operator $\wp_A(R)$ and (drop) projection

Kaiserslautern (KL)

**Sales**

| Store | Department | AvgSales |
|---|---|---|
| Innenstadt | TV | 139000 |
| Innenstadt | Computer | 156000 |
| Innenstadt | Hifi | 118000 |
| Gewerbegbt | TV | 112000 |
| Gewerbegbt | Computer | 180000 |
| Gewerbegbt | Hifi | 57000 |

KL2TR'= $\wp_{Store}$(KL)

KL2TR'

**Innenstadt**

| Store | Department | AvgSales |
|---|---|---|
| Innenstadt | TV | 139000 |
| Innenstadt | Computer | 156000 |
| Innenstadt | Hifi | 118000 |

**Gewerbegbt**

| Store | Department | AvgSales |
|---|---|---|
| Gewerbegbt | TV | 112000 |
| Gewerbegbt | Computer | 180000 |
| Gewerbegbt | Hifi | 57000 |

KL2TR= $\mathcal{U}_{Store}$(KL2TR')

OR

KL2TR= $\pi_{Department,\ AvgSales}$(KL2TR')

KL2TR

**Innenstadt**

| Department | AvgSales |
|---|---|
| TV | 139000 |
| Computer | 156000 |
| Hifi | 118000 |

**Gewerbegbt**

| Department | AvgSales |
|---|---|
| TV | 112000 |
| Computer | 180000 |
| Hifi | 57000 |

- FISQL statement:

```
SELECT KS.Department AS Dept, KS.AvgSales INTO KS.Store
FROM KL.Sales KS
```

A

# FIRA/FISQL example – MA2KL

- Transform the Mannheim database to the format of the Kaiserslautern database
- Requires a combination of the down and attribute deference operator

Mannheim (MA)

**AvgSales**

| Store | TV | Computer | Hifi |
|---|---|---|---|
| Quadrate | 205000 | 234000 | 108000 |
| Kaefertal | 90000 | 76000 | 87000 |
| Sandhofen | 73000 | 81000 | 98000 |

$S = \downarrow_1 (MA.AvgSales)$

**S**

| $r_1$ | $a_1$ | Store | TV | Computer | Hifi |
|---|---|---|---|---|---|
| AvgSales | Store | Quadrate | 205000 | 234000 | 108000 |
| AvgSales | TV | Quadrate | 205000 | 234000 | 108000 |
| AvgSales | Computer | Quadrate | 205000 | 234000 | 108000 |
| AvgSales | Hifi | Quadrate | 205000 | 234000 | 108000 |
| AvgSales | Store | Kaefertal | 90000 | 76000 | 87000 |
| AvgSales | TV | Kaefertal | 90000 | 76000 | 87000 |
| … | … | … | … | … | … |
| AvgSales | Hifi | Sandhofen | 73000 | 81000 | 98000 |

$S' = \sigma_{a_1 <> \text{'Store'}}(S)$

**S'**

| $r_1$ | $a_1$ | Store | TV | Computer | Hifi |
|---|---|---|---|---|---|
| AvgSales | TV | Quadrate | 205000 | 234000 | 108000 |
| AvgSales | Computer | Quadrate | 205000 | 234000 | 108000 |
| AvgSales | Hifi | Quadrate | 205000 | 234000 | 108000 |
| AvgSales | TV | Kaefertal | 90000 | 76000 | 87000 |
| … | … | … | … | … | … |
| AvgSales | Hifi | Sandhofen | 73000 | 81000 | 98000 |



# FIRA/FISQL – MA2KL (cont.)

$S'' = \Delta_{a_1}^{AvgSales}(S')$

**S''**

| $r_1$ | $a_1$ | Store | TV | Computer | Hifi | **AvgSales** |
|---|---|---|---|---|---|---|
| AvgSales | **TV** | Quadrate | **205000** | 234000 | 108000 | **205000** |
| AvgSales | **Computer** | Quadrate | 205000 | **234000** | 108000 | **234000** |
| AvgSales | **Hifi** | Quadrate | 205000 | 234000 | **108000** | **108000** |
| AvgSales | **TV** | Kaefertal | **90000** | 76000 | 87000 | **90000** |
| … | … | … | … | … | … | |
| AvgSales | **Hifi** | Sandhofen | 73000 | 81000 | **98000** | **98000** |

- Cleanup:
  MA2KL= $\pi_{\text{Store, Department, AvgSales}}\ \rho_{\text{a1->Department}}(S'')$

**MA2KL**

| Store | Department | AvgSales |
|---|---|---|
| Quadrate | TV | 205000 |
| Quadrate | Computer | 234000 |
| Quadrate | Hifi | 108000 |
| Kaefertal | TV | 90000 |
| … | … | |
| Sandhofen | Hifi | 98000 |

# FIRA/FISQL example – TR2KL

- Use Down (on DB) with Generalized union, renaming and projection:
  TR2KL= $\rho^{\varepsilon\text{->}Sales}\pi_{Store,\ Dept,\ AvgSales}\ \rho_{r1\text{->}Store}\ \Sigma(\downarrow_1 (TR))$

Trier (TR)

| Eisenbahnstr | | Hauptstr | |
|---|---|---|---|
| Dept | AvgSales | Dept | AvgSales |
| TV | 67000 | TV | 74000 |
| Computer | 51000 | Computer | 103000 |
| Hifi | 78000 | Hifi | 89000 |

$TR' = \downarrow_1 (TR)$

TR`

**Eisenbahnstr**

| $r_1$ | $a_1$ | Dept | AvgSales |
|---|---|---|---|
| Eisenbahnstr | Dept | TV | 67000 |
| Eisenbahnstr | AvgSales | TV | 67000 |
| Eisenbahnstr | Dept | Computer | 51000 |
| ... | ... | ... | ... |
| Eisenbahnstr | AvgSales | Hifi | 78000 |

**Hauptstr**

| r1 | a |
|---|---|
| Hauptstr | D |
| Hauptstr | A |
| ... | . |

$TR'' = \Sigma\ (TR')$

TR"

**ε**

| $r_1$ | $a_1$ | Dept | AvgSales |
|---|---|---|---|
| Eisenbahnstr | Dept | TV | 67000 |
| Eisenbahnstr | AvgSales | TV | 67000 |
| Eisenbahnstr | Dept | Computer | 51000 |
| ... | ... | ... | ... |
| Hauptstr | AvgSales | Hifi | 89000 |

TR2KL= $\rho^{\varepsilon\text{->}Sales}\ \pi_{Store,\ Dept,\ AvgSales}\ \rho_{r1\text{->}Store}\ (TR'')$

TR2KL

**Sales**

| Store | Dept | AvgSales |
|---|---|---|
| Eisenbahnstr | TV | 67000 |
| Eisenbahnstr | TV | 67000 |
| Eisenbahnstr | Computer | 51000 |
| ... | ... | ... |
| Hauptstr | Hifi | 89000 |



# FIRA/FISQL example – KL2MA

- Transform the Kaiserslautern database to the format of the Mannheim database
- Requires the transpose and drop-projection operators:

Kaiserslautern (KL)

**Sales**

| Store | Department | AvgSales |
|---|---|---|
| Innenstadt | TV | 139000 |
| Innenstadt | Computer | 156000 |
| Innenstadt | Hifi | 118000 |
| Gewerbegbt | TV | 112000 |
| Gewerbegbt | Computer | 180000 |
| Gewerbegbt | Hifi | 57000 |

$S = \tau^{Department}_{AvgSales}\ (Sales)$

KL

**S**

| Store | Department | AvgSales | TV | Computer | Hifi |
|---|---|---|---|---|---|
| Innenstadt | TV | 139000 | 139000 | - | - |
| Innenstadt | Computer | 156000 | - | 156000 | - |
| Innenstadt | Hifi | 118000 | - | | 118000 |
| Gewerbegbt | TV | 112000 | 112000 | - | - |
| Gewerbegbt | Computer | 180000 | - | 180000 | - |
| Gewerbegbt | Hifi | 57000 | - | - | 57000 |

$S' = \nu_{Department,\ AvgSales}(S)$

KL

**S'**

| Store | TV | Computer | Hifi |
|---|---|---|---|
| Innenstadt | 139000 | - | - |
| Innenstadt | - | 156000 | - |
| Innenstadt | - | | 118000 |
| Gewerbegbt | 112000 | - | - |
| Gewerbegbt | - | 180000 | - |
| Gewerbegbt | - | - | 57000 |
| Innenstadt | - | - | 97500 |

Merge

**S"**

| Store | TV | Computer | Hifi |
|---|---|---|---|
| Innenstadt | 139000 | 156000 | 118000 |
| Gewerbegbt | 112000 | 180000 | 57000 |

?

# FIRA/FISQL – Merging

- Merging of tuples required
  - Merging is simple if no "conflicts" arise
  - Merge not uniquely defined if tuples conflict
  - Two tuples $t_1$, $t_2$ of a relation with n attributes are mergeable if either
    - $t_1[A_i] = t_2[A_i]$ or
    - one of $t_1[A_i]$ or $t_2[A_i]$ is a null value

    holds for $1 \le i \le n$
  - The merge t of two mergeable tuples $t_1$, $t_2$ (denoted $t = t_1 \odot t_2$) is defined as

$$t[A_i] = \begin{cases} t_1[A_i] \text{ if } t_1[A_i] \text{ not null} \\ t_2[A_i] \text{ otherwise} \end{cases} \quad \text{for } 1 \le i \le n$$

- Optimal tuple merge
  - For a relation schema R and two relations $r_1$ and $r_2$ that are instances of R, $r_2$ is a tuple merge of $r_1$, if it can be obtained from $r_1$ by a finite sequence of merge operations of mergeable tuples
  - A tuple merge $r_2$ of $r_1$ is an optimal tuple merge, if for every $r_3$ that is also a tuple merge of r1
    $|r_2| \le |r_3|$ holds



# FIRA/FISQL – Merge Operator

- (Unique optimal tuple) Merge Operator $\mu(R)$ [WyRo05b]
  - Let R be a relational schema, and r an instance of R
  - Let $\emptyset^R$ denote the empty relation of schema R
  - Then the unique optimal tuple merge of r is

$$\mu(r) := \begin{cases} \emptyset^R \text{if there is more than one optimal tuple merge of } r \\ \text{the unique optimal tuple merge otherwise} \end{cases}$$

- Merge was not part of the original FIRA/FISQL
  - ➔ No FISQL syntax specified
- FISQL statement (without merge):

```
SELECT DROP (KS1.Department, KS1.Avgsales)
FROM    (SELECT KS.*, KS.Avgsales ON KS.Department
         FROM KL.Sales AS KS)     A               B
       S
S'      AS KS1
```

# FIRA/FISQL example – KL2MA continued

KL

**S'**

| Store | TV | Computer | Hifi |
|---|---|---|---|
| Innenstadt | 139000 | - | - |
| Innenstadt | - | 156000 | - |
| Innenstadt | - | | 118000 |
| Gewerbegbt | 112000 | - | - |
| Gewerbegbt | - | 180000 | - |
| Gewerbegbt | - | - | 57000 |

$S'' = \mu_{Store}(S')$

**S''**

| Store | TV | Computer | Hifi |
|---|---|---|---|
| Innenstadt | 139000 | 156000 | 118000 |
| Gewerbegbt | 112000 | 180000 | 57000 |

KL

**S'**

| Store | TV | Computer | Hifi |
|---|---|---|---|
| Innenstadt | 139000 | - | - |
| Innenstadt | - | 156000 | - |
| Innenstadt | - | | 118000 |
| Gewerbegbt | 112000 | - | - |
| Gewerbegbt | - | 180000 | - |
| Gewerbegbt | - | - | 57000 |
| Innenstadt | - | - | 97500 |

$S'' = \mu_{Store}(S')$

**S''**

| Store | TV | Computer | Hifi |
|---|---|---|---|



# FIRA/FISQL – Summary

- Theoretically sound approach to resolve schematic heterogeneity
- Open questions:
  - How does grouping/aggregation fit into the model?
    - Group by/aggregate over an unknown set of attributes ?
    - Could allow the user to solve the merge problem for relations with conflicting tuples by explicitly specifying the desired merge semantics (using an aggregate function)
  - What does transformational completeness mean in the XML data model?

Kaiserslautern (KL)

**Sales**

| Store | Department | AvgSales |
|---|---|---|
| Innenstadt | TV | 139000 |
| Innenstadt | Computer | 156000 |
| Innenstadt | Hifi | 118000 |
| Gewerbegbt | TV | 112000 |
| Gewerbegbt | Computer | 180000 |
| Gewerbegbt | Hifi | 57000 |

$\tau + \upsilon + \mu$

$\downarrow + \Delta$

Mannheim (MA)

**AvgSales**

| Store | TV | Computer | Hifi |
|---|---|---|---|
| Quadrate | 205000 | 234000 | 108000 |
| Kaefertal | 90000 | 76000 | 87000 |
| Sandhofen | 73000 | 81000 | 98000 |

$\downarrow + \Sigma$

$\wp + \pi/\upsilon$

Trier (TR)

**Eisenbahnstr**

| Dept | AvgSales |
|---|---|
| TV | 67000 |
| Computer | 51000 |
| Hifi | 78000 |

**Hauptstr**

| Dept | AvgSales |
|---|---|
| TV | 74000 |
| Computer | 103000 |
| Hifi | 89000 |

Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

# Schema Integration



## Schema Integration

- Goal: Create an integrated schema T from a set S of schemas that is:
  - complete (contains all concepts of S)
  - minimal (contains semantically equivalent concepts only once)
  - correct (each concept must correspond to a concept of at least one source)
  - intelligible (humans can understand the schema, e.g., names of concepts and their attributes should be preserved where possible)
- Schema Integration is *not* about transforming data from one schema to another (➔ Information integration, data fusion)
- Also known as schema (or ontology) merging
- Can be separated into four phases [BLN86]:
  - Preintegration
    - Choose schemas to integrate
    - Collect additional information (e.g., documentation of data sources)
  - Comparing the schemas
    - Schema Matching
    - Identify conflicts

# Schema Integration (cont.)

- "Conforming" the schemas
  - Resolve conflicts, e.g., by renaming attributes, restructuring (e.g., (de-)normalization))
  - At the end of the phase, identical concepts are represented identically in all schemas
- Schema Merging and Restructuring
  - Superimpose schemas
  - Restructure to meet the four goals

- Two main categories:
  - Binary approaches integrate exactly two schemas
  - n-ary approaches integrate an arbitrary number of schemas in one step
- For binary approaches, the sequence in which they are applied to the n input schemas can make a difference
- Most approaches are not algorithms, but guidelines
  - Even algorithms require manual conflict resolution
  - ➔ At best semi-automatic
- Examples:
  - Rondo Merge Operator [PoBe03]
  - Generic Integration Model (GIM) [ScSa05]



# Rondo Merge Operator – Schema Representation

- A model L is a triple (*E*, *Root*, *Re*), with *E* being a set of elements, $Root \in E$ being the root element of the model, and *Re* being the set of relationships of the model
- Elements with required properties *name* and an internal *ID*
- Binary, directed relationships R(x,y) with cardinality constraints and five different kinds:
  - Associates A(x,y) – elements x and y are associated in a (not further specified) manner
  - Contains C(x,y) – element x (container) contains element y (containee) (Containment)
    - Containees cannot exist on their own (i.e., delete on the container cascades to the containees)
    - transitive and acyclic
  - Has-a H(x,y) – element x has a subelement y (Aggregation)
    - weaker than contains: no cascading of deletes, cycles allowed
  - Is-a I(x,y) – x is a specialization of y (Specialization/Generalization)
    - transitive and acyclic
  - Type-of T(x,y) – x is of type y
    - an element can be of at most one type (*one-type restriction*)

# Rondo Merge Operator (cont.)

- Metamodel-specific *relationship implication rules* to infer implicit relations based on explicit relations, e.g.
  - If T(q,r) and I(r,s), then T(q,s) – an element q of type r is implicitly also an instance of any of r's superclasses s
  - If I(p,q) and H(q,r), then H(p,r) and If I(p,q) and C(q,r), then C(p,r) – an element inherits aggregates and components from its superclasses
- Mappings (=sets of correspondences) are themselves models
  - Contain mapping elements (two kinds: equality and similarity)
  - Contain mapping relationships M(x,y), indicating that mapping element x represents element y
  - All model elements y represented by a single mapping element via M(x,y) are said to *correspond* to one another



# Rondo Merge Operator Requirements

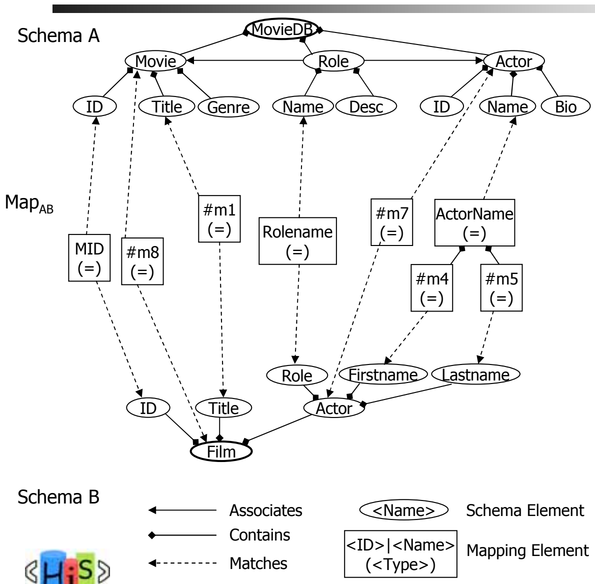
- Inputs:
  - Two models A and B
  - A mapping $Map_{AB}$ (=set of correspondences) between A and B
  - Optional: an indication which model is the preferred one
- Output: a merged model G
- Merge semantics based on *Generic Merge Requirements*
  1. Each element e with $e \in A \cup B \cup Map_{AB}$ corresponds to exactly one element e' in G (Element preservation)
  2. Two input elements are only mapped to the same element in G if the mapping indicates that they are equal (Equality preservation)
  3. Each input relationship is represented directly in G or implied by G (according to the rules of the metamodel) (Relationship preservation)
  4. Elements which are similar (but not equal) according to $Map_{AB}$, remain separate in G and are related by a relationship (Similarity preservation)
  5. No other elements besides those specified in rules 1-4 exist (Extraneous item prohibition)
  6. An element e in G has a property p if it has a corresponding element e' in A or B that has property p (Property Preservation)

# Rondo Merge Algorithm

- Form groups of elements for which an equality mapping exists (directly or transitively)
  - Groups include the mapping elements themselves
- For each group I, create an element e in G:
  - ID(e) is set to an unused ID value
  - For other properties p of e, p's value v is in order of precedence:
    1. the value of property p of a mapping element in I for which property p is defined, otherwise
    2. the value of property p of an element in I of the preferred model for which p is defined, otherwise
    3. the value of property p of any element of I for which p is defined.
    - If more than one value is possible in 1-3, one is chosen arbitrarily
    - ➔ Values of mappings take precedence over those of the preferred model over those of the other model
- For each pair of elements e' and f' in G that correspond to different groups E and F
  - if for any two e ∈ F and f ∈ F a relationship R(x,y) of kind t exists in A resp. B
  - create a relationship R(e',f') of kind t in G
  - Relationships between elements of the same group are ignored
  - Remove implied relationships until a mincover remains
- Resolve conflicts



# Merging Example

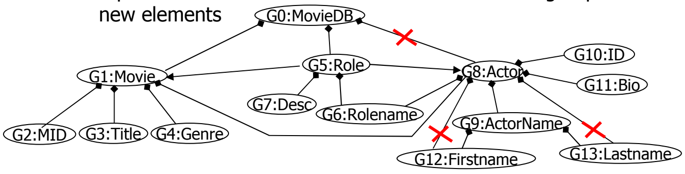



**Groups:**

| | |
|---|---|
| G0 | {MovieDB} |
| G1 | {A.Movie,B.Film, $Map_{AB}$.#m8} |
| G2 | {A.Movie.ID, B.Film.ID, $Map_{AB}$.MID} |
| G3 | {A.Movie.Title, B.Film.Title, $Map_{AB}$.#m1} |
| G4 | {A.Movie.Genre} |
| G5 | {A.Role} |
| G6 | {A.Role.Name, B.Film.Actor.Role, $Map_{AB}$.Rolename} |
| G7 | {A.Role.Desc} |
| G8 | {A.Actor, B.Film.Actor, $Map_{AB}$.#m7} |
| G9 | {A.Actor.Name, $Map_{AB}$.ActorName} |
| G10 | {A.Actor.ID} |
| G11 | {A.Actor.Bio} |
| G12 | {B.Film.Actor.Firstname, $Map_{AB}$.#m4} |
| G13 | {B.Film.Actor.Lastname, $Map_{AB}$.#m5} |

# Merging Example (cont.)

- Merge(A,B, $Map_{AB}$) with A as the preferred schema
  - One element for each group
  - replicate all associations between members of the groups as associations between the new elements

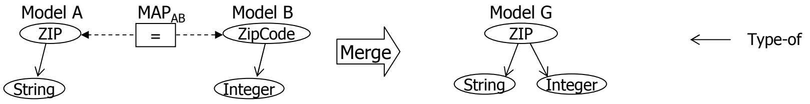



- Remove implied relationships to obtain minimum coverage of associations

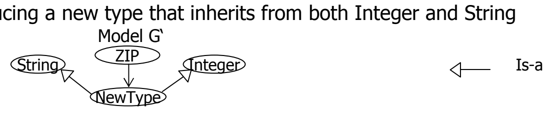


# Conflict resolution

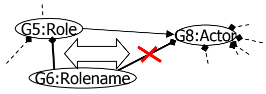
- Fundamental conflicts (shared across all metamodels)
  - e.g. One-type restriction violated




  - Resolve e.g. by introducing a new type that inherits from both Integer and String




- Metamodel conflicts
  - Metamodel-dependent resolution rules
  - e.g., in most data models, an element can be containee in at most one container
    - e.g. Rolename in the example
    - remove one containment relationship
  - SQL92 does not have the concept of subcolumn (as needed for name(firstname, lastname))



Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

# Integration Planning



## Integration Planning – Goals

- Creation of an "executable mapping", i.e., a data transformation from source to target schemas
- Inputs
  - Source schemas (and data)
  - Target schema (and sample data)
  - (Correspondences)
- Output
  - An "executable mapping", i.e., a specification for data transformation from the sources to the target schema
  - e.g. SQL(/XML) queries/views, ETL scripts, XQuery statements etc.
  - Usually created manually with tool support
- Many different approaches to partially automate the process
  - Clio Query Discovery [MHH00]
  - Tupelo [FlWy06]
  - Integration Patterns [Gö05a]

# Clio Query Discovery – Overview

- Clio is a combined tool for schema matching and mapping
- Creates executable mappings as SQL/XQuery statements for use in FDBMS
- Uses *value correspondences (VCs):*
  - Essentially complex 1:n matches
  - A value correspondence $v_i$ is a tuple $(f_i, p_i)$ with
    - *a function* $f_i$ describing how to derive a certain target attribute B from a set of source attributes $A_k$ (and possibly from source metadata):
      $f_i$: dom($A_1$) x dom($A_2$) x … dom($A_q$) $\rightarrow$ dom(B)
    - a *filter* $p_i$ indicating which source values should be used:
      $p_i$: dom($A_1$) x dom($A_2$) x … dom($A_r$) $\rightarrow$ boolean
  - Note: function and filter of a correspondence can be defined on different sets of attributes
- Idea: Divide the set of value correspondences *V* into subsets each of which determines one way to compute a given target relation $T_k$



# Clio Query Discovery – Algorithm

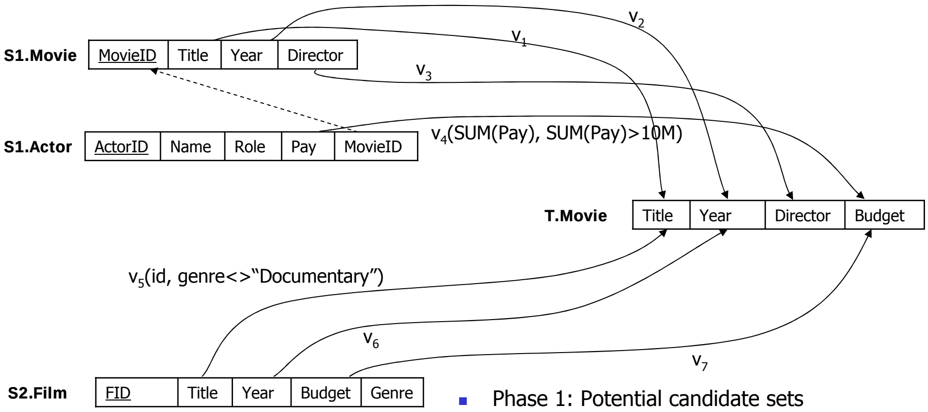
- Consists of four distinct phases
- For each target relation $T_k$
  1. Partition V into *potential candidate sets* $\{c_1, \ldots, c_p\}$ that contain *at most* one VC per attribute of $T_k$:
     - The $c_i$ need not be disjoint
     - A $c_i$ is called *complete* if it includes a VC for *every* attribute in $T_k$
     - Prefer complete potential candidate sets, and further prefer those that use the smallest set of source relations
     - Prune potential candidate sets that are subsets of another
     - Incomplete candidate sets are considered, as not every target attribute might have a VC
  2. Prune those potential candidate sets that cannot be mapped to a "good" query
     - To create a query, a way of joining the source relations of the potential candidate set is needed
     - Search for *join paths* (i.e. foreign keys) between the relations
     - If several join paths exist, use the one for which the estimated difference in size of an outer and an inner join is smallest, resulting in a minimum number of dangling tuples
     - If no join path exist, request the user to specify them
     - All potential candidate sets without a join path are removed
     - Result: *Candidate sets* for every target relation, representing different ways to obtain the values of the target relation
     - Each candidate set can be mapped to a Select-Project-Join(-Group-by-Aggregate) query

# Clio Query Discovery – Algorithm (cont.)

3. Find sets of the candidate sets (*covers*) that contain every VC at least once
   - Determine a minimum cover, i.e., eliminate all covers from which candidate sets can be removed while still containing all VCs
   - Rank the remaining covers according to the inverse number of candidate sets they contain (less candidate sets means less queries)
   - For those with an equal number of candidate sets, choose those that have the largest number of target attributes in all candidate sets (i.e., minimize null values)
   - Present ranked covers as alternative mappings to the user
4. Create the query q for target relation $T_k$ from the selected cover
   - For each candidate set $c_i$ in the cover, create a candidate query $q_i$ such that
     - All correspondence functions $f_k$ mentioned in $c_i$ appear in the SELECT clause
     - All source relations of the VCs in $c_i$ appear in the FROM clause
     - All predicates $p_i$ of the VCs in $c_i$ appear in the WHERE clause
     - All source relations needed for join paths appear in the FROM clause and the join predicates appear in the WHERE clause
     - If $c_i$ contains aggregate functions, all attributes not in the aggregate function are selected as grouping attributes. If the aggregate is in the correspondence function $f_k$, it is placed in the SELECT clause. If it is in a predicate, it is placed in a HAVING clause.
   - Combine all candidate queries $q_i$ into q by the use of UNION ALL



# Clio Query Discovery – Example




- Phase 1: Potential candidate sets
  - c1 = {v1, v2, v3, v4}
  - c2 = {v5, v6, v7}
  - c3 = {v1, v6, v3, v7}
  - c4 = {v5, v2, v3, v7}
  - …

default for $f_i$ is id, default for $p_i$ is true

## Clio Query Discovery – Example (cont.)

- Phase 2: Eliminate potential candidate sets that have no good query
  - e.g. $c_3$ and $c_4$ have no join paths, others are subsets
  - Only $c_1$ and $c_2$ remain
- Phase 3: Find all minimum cover (sets of candidate sets that contain all VCs)
  - ➔ $\{\{c_1,c_2\}\}$
- Phase 4: Create candidate querys and combined query:

$q_1$
```
SELECT Title, Year, Director, SUM(Pay)
FROM S1.Movie m, S1.Actor a
WHERE m.MovieID = a.MovieID
GROUP BY Title, Year, Director
HAVING SUM(Pay) >10M
```
```
UNION ALL
```
$q_2$
```
SELECT Title, Year, null, Budget
FROM S2.Film
WHERE genre <> "Documentary"
```




Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

# Dynamic Information Integration

The Paladin Project

http://wwwdvs.informatik.uni-kl.de/aghis/projects/paladin/

http://wwwdvs.informatik.uni-kl.de/aghis/staff/Goeres/offen.html

## Differences to Classical Integration Scenarios

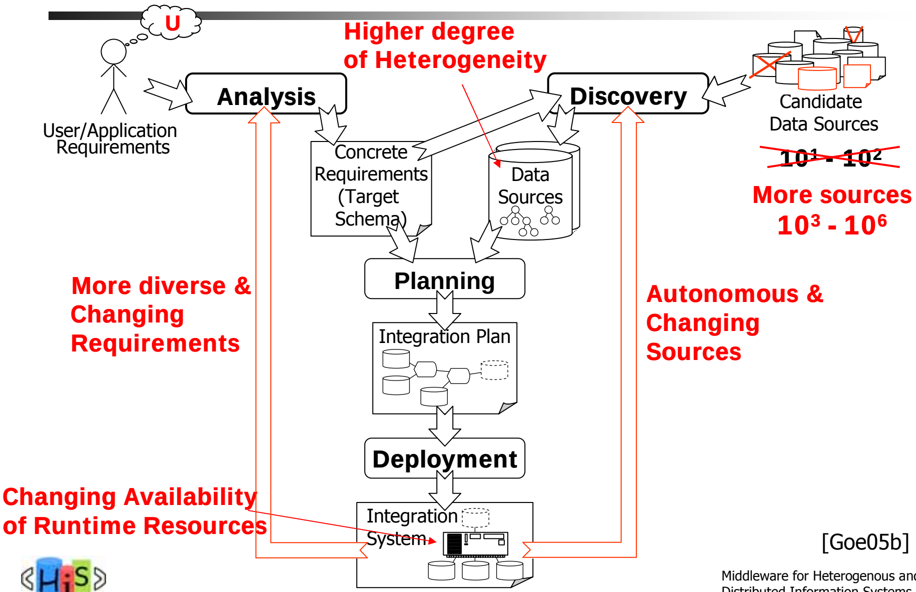
- Classical integration scenarios
  - Integrate data within organizations (department, company)
    - Limited degree of heterogeneity (as data sources come from similar context)
  - Assume a static, closed-world environment
    - Well-known data sources
    - Full administrative control over all involved data sources (i.e., very limited data source autonomy)
    - Well-defined and fixed user/application requirements
- Dynamic integration scenarios
  - Integrate data across organizational boundaries
    - Higher degree of heterogeneity
      - Technical/Data model: Choice of technology not limited by company policy
      - Semantic: data sources come from completely different backgrounds
  - Dynamic, open-world environment
    - Data sources come and go, or change their properties (export schema)
    - Several orders of magnitude more data sources
    - More and more diverse user requirements ("integration to the masses")
  - e.g. data grids, mashups



## Challenges

- Requires a top-down approach to integration
  - Methods to specify a target schema for non-expert users
- Find the best sources for a given information requirement
  - Methods to discovery the most useful sources out of a very large number of candidates
- Reduce the human interaction needed for integration planning
  - Better automatic schema matchers
  - Improved (Semi-)automatic creation of integration plans from input schemas, target schema, and mappings across all kinds of data sources and data models
- Flexibility in the deployment of integration plans to different runtime environment
  - Users might not have the resources to host their own integration system
  - Hosting provided as service
  - Mechanisms for flexible redeployment needed to allow migration to different provider (due to cost, availability)

# Dynamic Integration Process – Challenges




[Goe05b]



# Paladin Metamodel Architecture - Motivation

- II involves many different tasks that require handling of data and metadata
  - For each II task, a plethora of different mechanisms, languages and tools exist with individual advantages and drawbacks
  - No single "swiss army knife" to tackle all integration problems
  - A practical integration scenario requires several tools, but
- Interoperability between tools is limited
  - No uniform representation of metadata (and data)
  - ➔ Requires a lot of conversions (not always possible)
  - Very different mechanisms to describe data transformations
    - descriptive (query languages, algebras)
    - procedural/declarative/functional programming languages (XSLT, ETL scripts, shell scripts...)
    - parameterized black boxes (wrappers)
  - No single mechanism is powerful enough to express a complete source-to-target data transformation (*integration plan*)
  - ➔ Actual integration plans have to be described by combining several mechanisms
  - ➔ Integration plans are tightly coupled to a type of runtime system

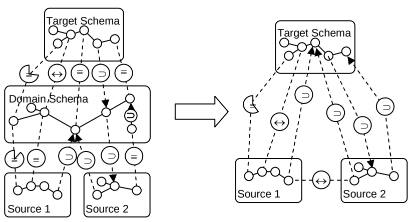
# Paladin Metamodel Architecture (PMA)

- Goal: An integrated model for data, metadata and operations [GoDe07]
- Graph-based representation of data and metadata that is
  - extensible (can support new metamodels)
  - lossless (support special features of different metamodels)
  - generic (capture common properties among different metamodels)
  - efficient (should not be overly verbose, especially for data representation)
  - natural
- Generic representation of arbitrary data and metadata management operators
  - All operations manipulate the graph structure of data and metadata represented in PMA
  - Idea: express the effects of operators as graph transformations
  - Chain transformations to obtain one consistent representation for source-to-target integration plans




# Discovery

- Begin integration planning with $10^4$-$10^6$ sources?
  - Need to preselect the *most useful* data sources
- What makes a data source useful?
  - ➔ Covers *similar concepts* as the target schema
- How are similar concepts identified (in the presence of heterogeneity)?
  - Schema matching!
  - But matching target schema with $10^4$-$10^6$ sources is infeasible
  - Idea: Indirect schema matching via *domain schemas*
    - Matches can be created offline during data source registration
    - At discovery time, only the target schema has to be matched against (few!) domain schemas
    - Direct matches can be inferred assuming transitivity
    - Use inferred matches to calculate utility measures for the sources



# Planning – Resolving semantic heterogeneity

- Schema Matching Framework (ScheMaF)
  - Idea: provide infrastructure on which to more easily implement new and existing schema matchers
    - Paladin Metamodel Architecture for the uniform and efficient representation of metadata and data
      - Generic access to metadata for schema-based matchers and
      - to data (statistics) for instance-based matchers
    - Metadata/data import/export, preprocessing (stemming, tokenization), ...
  - Individual schema matching approaches built as components on top of this infrastructure
  - Allow the flexible wiring of different matchers into *match plans*
    - Import the schemas
    - Perform preprocessing
    - Apply matchers
    - Combine match results using different composite matchers
  - Match plans can be domain-specific



# Planning – Creating a Mapping

- Existing approaches to create "executable mappings" are algorithmic
  - Can only solve problems that the programmers anticipated
  - Are difficult to extend
  - Are usually limited to one or few metamodels
- Observations
  - Integration planning is essentially solving a large number of atomic mapping problems
  - The same abstract mapping problems appear in many different manifestations in many different application domains
  - Schema matches abstract from the detailed semantics of the application domain and capture application knowledge
  - Once matches have been determined, the actual application domain is irrelevant for solving the mapping problem
- Idea: Integration patterns [Goe05a]
  - Describe a problem situation (e.g., a specific constellation of schema elements and matches) in an abstract way
  - Provide an abstract solution to the problem (i.e., a small part of a mapping)
  - Recognize the abstract problem situation in a specific integration scenario
  - Adapt the abstract solution to the specific problem and apply it

# Integration Patterns

- Problem description uses the graph representation of data and metadata
  - Describe problem situations as graph patterns
  - Describe solution as parameterized operator(s)
- Choose the "best" pattern for the given situation
  - Highest increase of similarity between input schemas and target schema (greedy approach)
  - Perform backtracking if dead end reached
- Compare to
  - Proving a theorem (the target schema)
  - starting with a set of axioms (the sources)
  - using a set of rules (the patterns)
- Result: an abstract integration plan formed by chained operators
- Open issues:
  - Decidability
  - Pattern granularity



# Deployment

- Abstract integration plan cannot reasonably be used to answer queries directly (performance)
- Idea: Deploy abstract integration plan to (a federation of) concrete runtime platform(s)
  - e.g. translate the abstract plan to a set of configured wrappers and SQL views if target platform is an FDBMS
  - or translate it to an ETL script if a materialized integration is preferred
- Compare to Model-driven Architecture:
  - Abstract plan = platform-independent model (PIM)
  - Concrete plan = platform-specific model (PSM)
- Open issues:
  - Graph transformations are very generic, allow specifying the same operation in many different ways
  - How to recognize an operation in order to map it to the operators of a given target platform?

Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

# Change Management in Large-Scale Information Infrastructures

The Caro Project

http://wwwdvs.informatik.uni-kl.de/aghis/projects/caro/index.html

http://wwwdvs.informatik.uni-kl.de/aghis/staff/Stumm/aktuell.html



## Change Management (CM) in Large-Scale Information Infrastructures

- Information infrastructures consist of dozens, hundreds or more single systems
  - Each system has *metadata* describing it
- Complex dependency structures between single systems
  - Federations, materialized views, workflows, replication, …
- Local changes can impact a large part of the whole infrastructure
  - System failure, wrong results, data corruption, …
  - Caused by schema changes, “bug fixes”, changing file system structures, …
- No central point of responsibility
  - Different departments, B2B scenarios, …
- Heterogeneous and dynamic environment
  - Systems keep changing all the time
  - Metadata is provided in many different formats and models
- How to keep everything running?

# What is System Metadata?

- Database and document schemas
  - SQL schemas, XML Schemas, DTDs, proprietary schemas for text and binary files, …
- System configuration
  - Java property files, windows registry, …
- APIs
  - Libraries APIs, web services (WSDL files), …
- Functional assertions
  - Behavior of function, methods, and software components.
- Quality assertions
  - Data quality assertions, execution time and speed, data freshness, …

⇨ *System Metadata (in CM): any fact about a system that could, if changed, have an impact on a dependent system.*



# Change Management and Change Impact Analysis (CIA)

- Organization
  - Strict Change Management Process: Adherence to a strict process ensures that no faults occur.
  - No/Ad-hoc Change Management Process: Without adherence to a CM process, probability of faults increases.
- CIA Time
  - Preventive: Analyze the impacts of changing a system before applying the changes.
  - Reactive: Impact analysis is done at some point in time after changes are applied.
- CIA Automation:
  - Automatic: The complete, global impact analysis as well as adapting systems to changes can be done automatically.
  - Semi-Automatic: At least some manual intervention is needed, either in CIA or in adaptation.
  - Manual: Everything has to be done manually.

# Caro – Basic Idea

- A *strict* change management process, with *fully automated, preventive* CIA and *automatic* adaptation of impacted systems is not realistic.
  - The "human in the loop" will bypass uncomfortable procedures and processes, and enforcing them is not practicable.
  - High heterogeneity doesn't allow for a full automation.
  - Absence of centralized management and distributed responsibilities hinder preventive CIA.
- The best we can get is an approximation of the optimal case, on a "best effort" basis:
  - No assumptions regarding CM processes
  - Do preventive CIA if possible, and try to do reactive CIA as soon as possible after a change is applied.
  - Enable the use of Caro as a distributed system, to account for different responsibilities in different areas of the information infrastructure.

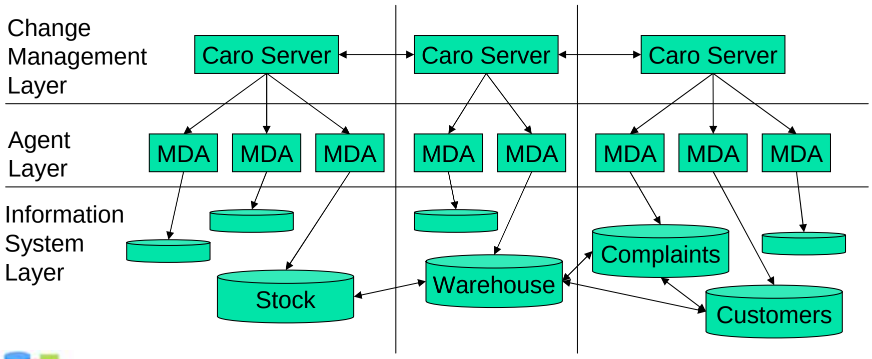



# Caro – Goals

- Provide a model
  - to represent arbitrary system metadata in a generic way
  - to represent the dependencies between systems
- Provide algorithms for change impact analysis
  - locally within a system, and globally over system boundaries
- Provide means for preventive reactive change impact analysis
  - An editor to manually edit metadata, and let the analysis component calculate what impact these changes would have
  - Metadata Agents which observe information systems for changes, so they do not go undetected for a long time
  - Directives (system shutdown, administrator notification, automatic adaptation) to execute when changes are detected
- Provide a central management component
  - to store metadata of participating systems
  - to do global change impact analysis and coordinate the various metadata agents

## Caro – Architecture

- Change Managers: change impact analysis, versioned metadata storage, communication between each other
- Metadata Agents: extracting metadata, observing systems for changes, notifying, initiating repair
- Dependent Systems: do not need to be aware of the Caro framework






## Caro – Metadata Agents (MDAs)

- Generic MDA framework
  - Provides the common functionality of an MDA
  - Wrapper approach similar to Garlic
  - Not all systems have optimal capabilities, and the MDA framework has to account for that
- System-specific plugins
  - Implement the parts needed to "connect" to the system
- Tasks
  - Monitoring systems for changes (periodic or manual scan, or direct notification by the system if supported)
  - Metadata extraction (complete on initial deployment, later only the differences, if supported)
  - Analysis of the changes (intra-model analysis)
  - Communication between users, system, and Caro server
  - Editing interface for manual data entry
  - Executing directives

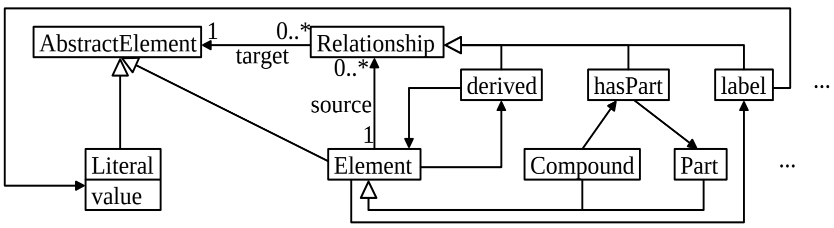
# Caro – Model

- Concepts
  - Provision: The part of the metadata of a system visible to other systems
  - Usage: The part of the metadata of a system specifying on which parts of another system it depends on
  - A usage is generally a copy of a part of a provision
- Metadata graphs
  - A labeled, bipartite digraph
  - Element nodes represent the metadata elements, like Tables, Columns, XML elements, types
  - Relationship nodes represent the correlation of element nodes, like "hasColumn", "hasType"
  - Literal nodes contain values, like table names or cardinality restrictions
- Model layers
  - The change impact system description model (CISDM) defines node types relevant for CIA
  - Under the CISDM, models for specific data models (SQL, XML, …) are defined

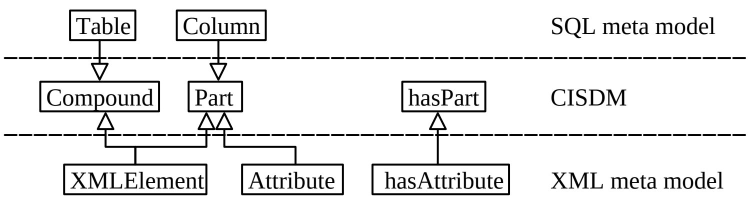


# Caro – CISDM

Core CISDM containing change-relevant model elements




SQL and XML models inheriting from the core CISDM



# Caro – Example (1)

Reporting Application

```
Report generateProjectManagers() {
   …
   query("select e.name, count(e.name)
      from employees e, projects p
      where e.id = p.manager
      group by e.name");
   …
}
```

```
Report generateUrgentProjects() {
   …
      query("select *
         from projects
         where deadline < current_date + 30");
   …
}
```

Backend Database

```
create table employees (
    id integer primary key,
    name varchar(50),
    salary integer
);
```

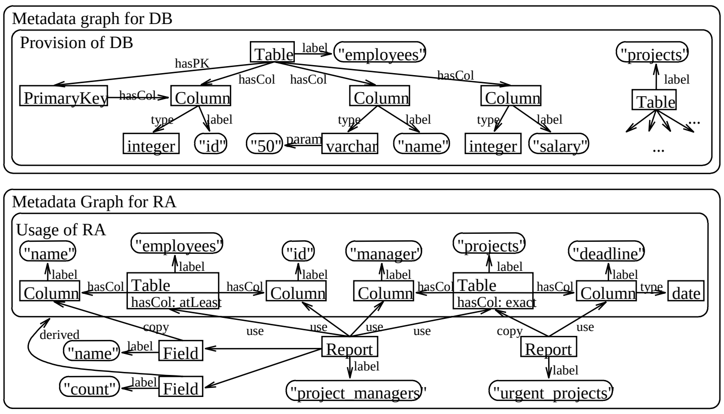
```
create table projects (
    pid integer primary key,
    p_name varchar(50),
    deadline date,
    manager integer references employees
);
```



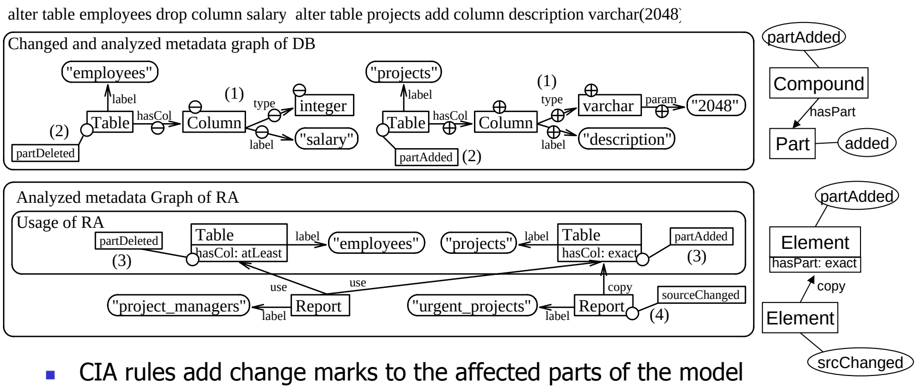


# Caro – Example (2)

- Metadata graphs consisting of provision and usage



# Caro – Intra-Model Analysis

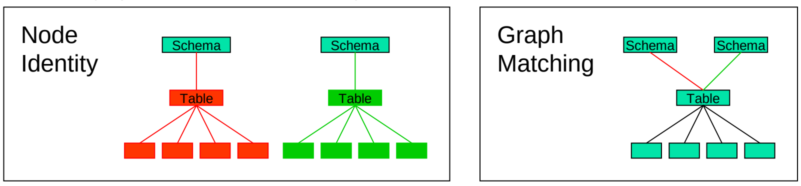



- CIA rules add change marks to the affected parts of the model
  - for provision model, mark changes for elements (1) and compounds (2)
  - copy the change status information to usage model (3) and analyse impact (4)
- Rules are repeatedly applied until no more rules fire



# Caro – Inter-Model Analysis

- Usage is a subset of the provision
  - Usage and provision are matched with a graph matching algorithm, and change marks just copied over
- Computational complexity
  - Graph-subgraph matching is a NP-hard problem
  - In practical cases, the graphs have a very regular, tree-like structure
  - Matching can be done in an efficient way
- Advantage
  - No matching per node identity
  - If parts of a graph "move", e.g., a table moves to another schema, much fewer parts of the graph will be marked as changed



# References (I)


- [BLN86] Batini, C.; Lenzerini, M. & Navathe, S.B.
  A comparative analysis of methodologies for database schema integration
  *ACM Comput. Surv., ACM Press,* 1986*, 18*, 323-364
- [CRF03] Cohen, W.W.; Ravikumar, P. & Fienberg, S.E.
  A Comparison of String Distance Metrics for Name-Matching Tasks.
  *IIWeb,* 2003, 73-78
- [FlWy05] Fletcher, G.H.L. & Wyss, C.M.
  Relational data mapping in MIQIS
  *SIGMOD 2005, ACM Press*, 912-914
- [FlWy06] Fletcher, G.H.L. & Wyss, C.M.
  Data Mapping as Search.
  EDBT, 2006, 95-111
- [GoDe07] Göres, J. & Dessloch, S.
  Towards an Integrated Model for Data, Metadata, and Operations
  BTW 2007, to appear
- [Goe05b] Göres, J.:
  Towards Dynamic Information Integration
  First VLDB Workshop on Data Management in Grids (DMG05), Trondheim, 2005, 16-29
- [Goe05a] Göres, J.:
  Pattern-based Information Integration in Dynamic Environments
  9th International Database Engineering Applications Symposium (IDEAS 2005), 125-134
- [LeNa07] Leser, U. & Naumann, F.:
  Informationsintegration
  *dpunkt Verlag,* 2007
- [LSS96] Lakshmanan, L.V.S.; Sadri, F. & Subramanian, I.N. Vijayaraman, T.M.
  SchemaSQL: A Language for Interoperability in Relational Multidatabase Systems
  VLDB 1996, 239-250




# References (II)


- [LSS01] Lakshmanan, L.V.S.; Sadri, F. & Subramanian, S.N.
  SchemaSQL: An extension to SQL for multidatabase interoperability
  Database Systems, 2001, 26, 476-519
- [RaBe01] Rahm, E. & Bernstein, P.A.
  A survey of approaches to automatic schema matching
  VLDB Journal, 2001, 10, 334-350
- [PoBe03] Pottinger, R. & Bernstein, P.A.
  Merging Models Based on Given Correspondences.
  VLDB, 2003, 826-873
- [MBR01] Madhavan, J.; Bernstein, P.A. & Rahm, E.
  Generic Schema Matching with Cupid
  The VLDB Journal, 2001, 49-58
- [MGR02] Melnik, S.; Garcia-Molina, H. & Rahm, E.
  Similarity Flooding: A Versatile Graph Matching Algorithm
  and Its Application to Schema Matching.
  ICDE 2002, 117-128
- [MHH00] Miller, R.J.; Haas, L.M. & Hernández
  Schema Mapping as Query Discovery
  VLDB 2000, Morgan Kaufmann, 2000, 77-88
- [ScSa05] Schmitt, I. & Saake, G.
  A comprehensive database schema integration method based on the theory of formal concepts.
  Acta Inf., 2005, 41, 475-524
- [WyRo05] Wyss, C.M. & Robertson, E.L.
  Relational languages for metadata integration
  ACM Trans. Database Syst., ACM Press, 2005, 30, 624-660
- [WyRo05b] Wyss, C.M. & Robertson, E.L.
  A formal characterization of PIVOT/UNPIVOT
  CIKM 2005, ACM Press, 602-608

Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

# Data Integration

- Data Quality Problems
- Causes and Consequences
- Data Cleaning Approaches



## Data Quality

- All approaches discussed so far only resolve heterogeneity regarding the schemas/metadata of the data sources
- Problems in the data itself remain to be resolved:
  - Erroneous data (values outside domain, violated constraints)
  - Data inconsistencies (Contradictions across and within a data source)
  - Duplicates (Are two tuples from different sources refering to the same real world object?)
  - Completeness (Does a data source deliver all data for a concept?)
  - Credibility (Is the source reliable, can the data be trusted?)
  - Timeliness (Is the data up-to-date?)
- Many problems are similar to those for schema integration
  - Synonyms, homonyms ~ semantic heterogeneity
    - Do the tables "Person" and "Pers" refer to the same concept? ≈
    - Do "Gottlieb-Daimler-Straße" and "Gottl.-Daiml.-Str" refer to the same object?
  - Considerable degree of uncertainty
  - Scale of the problem several orders of magnitude larger:
    - ~$10^2$-$10^3$ schema elements, but $10^2$-$10^9$++ instances
    - Resolving data quality ("Data Cleaning") problems is extremely expensive
    - Today usually only done in replicating integration systems

# Classification of Data Quality Problems

- based on [RaDo00, LeNa07]

Data Quality Problems
- Single-source Problems
  - Schema Level
  - Instance Level
- Multi-source Problems
  - (Schema Level)
  - Instance Level

- Allocation of problems to categories is not always unambiguous
- Instance level multi-source problems were previously subsumed as syntactic heterogeneity
- Schema level multi-source problems were discussed in previous sections (forms of heterogeneity)




# Single-source schema level problems

- Lack of integrity constraints: data source cannot enforce application constraints that are not made explicit using the facilities of the data model
  - No unique constraints ➔ Duplicate values
  - No enforced referential integrity ➔ inconsistent references
  - Inadequate typing (e.g. String to represent dates) ➔ invalid values
  - Unspecified dependencies ➔ dependency violations
    - e.g. Age = $today – birthdate
  - NOT NULL constraint omitted ➔ missing values
- Bad Schema Design
  - e.g., redundancies in schema caused by denormalization

  ➔ Inconsistencies due to insert/delete/update anomalies

# Single-source data level problems (I)

- Typos (e.g. "Gremany")
  - can be resolved by spellcheckers or domain experts
- Dummy values to "outwit" constraints
  - e.g. ZIP code 99999 used for "unknown value"
  - "John Doe" for an unidentified person
  - often resolvable for domain experts, but dummy values often not used consistently
- Wrong values – value does not properly represent the real world
  - e.g. Movie(Title="Lord of the Rings", Year="1928")
- Deprecated values
  - e.g. Germany(Founded="1949", Chancelor="Gerhard Schröder")
- Cryptic values
  - encoded or abbreviated data values
- Embedded values
  - values embedded in other fields to compensate for missing fields
  - e.g. Movie(Title="Fight Club, 1999")
- Wrong allocation
  - correct value entered into wrong field/swapped values
  - e.g. Actor(Name="Tyler Durden", Role="Brad Pitt")



# Single-source data level problems (II)

- Wrong reference
  - reference to an existing, but the wrong object
- Contradictory values
  - Address(Name="Kaiserslautern", ZIP="12345")
  - Student(Name="Christian Meier", Gender="f")
- Transpositions
  - different sequences used for data items within a field
  - Person("Hans Meier"), Person ("Müller, Karl")
- Duplicates
  - two or more data records representing the same real world object
  - techniques for duplicate detection and resolution
  - a problem with many names: record matching, entity resolution, instance disambiguation
  - Data Conflicts
    - Duplicates contradict each other
    - Movie(Title="Lord of the Rings", Year="1978") vs. Movie(Title="Lord of the Rings", Year="2001")
    - How to separate two duplicates with a conflict from two correct entries?

# Multi-source data level problems

- Differentiation is difficult – Therefore, multi-source data level problems
  - are new kinds of problems that *typically* occur during integration of several source (but can also be present in a single source)
  - include many of the single-source data level problems, e.g. Transpositions, Duplicates when they occur after integration
- Contradictory values
  - data from different sources contradict each other (≠Conflict!)
  - e.g. Source1.Person(ID="1234", Age="47") vs.<br>Source2.Person(ID="1234", DoB="1983-06-03")
- Differing representations
  - e.g. Source1.Emp(ID="1234", Job="Sales Mgr.") vs.<br>Source2.Emp(ID="1234", Job="S24")
- Different physical units
  - e.g. Source1.Person(Name="Herbert Meier", height="183") [cm] vs.<br>Source2.Person(Name="Herbert Meier", height="72") [inches]
- Different precision
  - e.g. Source1.Movie(Title="Fight Club", runtime="2h19min") vs.<br>Source2.Movie(Title="Fight Club", runtime="2h19min12sec")
- Different levels of details
  - e.g. "all actors" vs. "only main cast"



# Handling Data Quality Problems

Entweder diese und die nächste Folie, oder diese ohne Entity Resolution plus den Rest

- Two Phases
  - Data Scrubbing (individual records)
    - Resolve errors within individual tuples/data items
    - Normalise data
      - unify case, stemming, stopword removal, acronym expansion
      - Formating: unify date formats, person names ("H. Schmidt" vs. "Schmidt, H."), addresses
    - Conversions: convert numerical values to a single unit
      - simple for physical values (e.g.: length measures: conversion between m, cm, inch etc. is constant)
      - difficult for currencies! (which exchange rate to use? Today's? The rate at the (maybe unknown) insertion date?)
    - Remove outliers
      - test if data conforms to expectations (expressed as constraints, „sanity checks")
      - perform lookup in reference data (e.g., telephone directories)
    - Violated constraints
      - Test referential integrity
  - Entity Resolution
    - Detect and resolve duplicate records
    - Resolve inconsistencies among duplicates (Fusion)

# Handling Data Quality Problems (II)

- Entity Resolution
  - Resolve problems involving multiple records
  - Detect duplicate entries
    - Pairwise comparison of tuples, calculation of a similarity value
    - If similarity above threshold -> duplicate detected
    - False positives and negatives
    - Determine quality of duplicate detection using
      - precision (percentage of identified duplicates that are really duplicates)
      - recall (percentage of actual duplicates found)
    - Very expensive: $O(n^2)$ (possibly very complex) comparisons
    - Partition data and only compare tuples within a partion
  - Data fusion
    - Combine detected duplicates into one consistent tuple
    - 4 cases: equality, subsumption, complement, conflict
    - Subtlety of null value semantics

Kurzfassung, des Themas, die oder die folgenden Folien

wenn Kurzfassung, dann auf vorheriger Folie den Entity-Resolution-Teil streichen



# Duplicate Detection

- Basic idea: pairwise comparison of all tuples, for each pair
  - Determine similarity value *sim*
  - If value is above a given threshold *th* ➔ duplicates detected
- Two conflicting goals:
  - Effectiveness – high quality of the result
    - Result of a duplicate detection method contains four kinds of pairs of tuples:

| | | *Reality* | |
|---|---|---|---|
| | | duplicate | no duplicate |
| *Result of Method* | duplicate | true positive | false positive |
| | no duplicate | false negative | true negative |

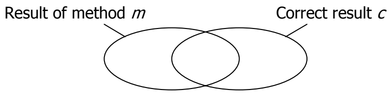
    - Minimize false positives and false negatives
  - Efficiency – effort to obtain the result
    - A duplicate detection method has to scale well with the amount of data

# Effectiveness of Duplicate Detection

- Two measures known from information retrieval to assess effectiveness
  - Precision – fraction of actual duplicates among all pairs identified as duplicates
  - Recall – fraction of correctly identified duplicates among all actual duplicates

$$\text{precision} = \frac{|\text{true-positives}|}{|\text{true-positives}| + |\text{false-positives}|} \qquad \text{recall} = \frac{|\text{true-positives}|}{|\text{true-positives}| + |\text{false-negatives}|}$$

- Precision and Recall illustrated using Venn diagrams




$$\text{precision} = \frac{|m| \cap |c|}{|m|} \qquad \text{recall} = \frac{|m| \cap |c|}{|c|}$$

- Recall and precision are themselves in conflict:
  - A cautious method will yield a high precision (only definitive duplicates are returned) but a low recall (a lot of actual duplicates will be missed)
  - A rigid method will yield a low precision (non-duplicates are returned) but a high recall (few actual duplicates will be missed)
- Combined assessment: $\text{f-measure} = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}$



# Efficiency

- Pairwise comparison of n tuples requires n·(n-1)/2 comparisons (O($n^2$))
- Expensive, even when effort for a single comparison is low
- Duplicate detection methods try to reduce the number of comparisons
- Partitioning
  - Split data items in (possibly overlapping) partitions
  - Compare only tuples within a partition
  - Increases efficiency at the cost of recall
  - A good partitioning scheme is hard to find
- Sorted-Neighborhood method [HeSt98]
  - Sort tuples based on a domain-specific key
    - e.g. the first three characters of a person's name concatenated with the year of the person's date of birth
  - Slide a window of fixed size *w* over the *n* data items (with w<<n)
    - Determine similarity of tuples within the first window (requires w·(w-1)/2 comparisons)
    - Slide window in one-tuple increments. For each step, only w-1 new comparisons are needed, as only the new record has to be compared against the other records in the window (requires a total of (n-w)·(w-1) comparisons)
    - w·(w-1)/2 + (n-w)·(w-1) = (w/2 + n-w)·(w-1) => O(w·n)
    - Total complexity (including sorting) O(w·n·log n)
  - Often the transitive closure of the is-duplicate relationship is created
    - Not without problems: FATHER ~ MATHER ~ MOTHER ➔ FATHER ~ MOTHER ?
    - Idea: cut the "transitivity chain" between the pair with the lowest similarity, declaring them to be no duplicates

# Efficiency (II)

- Sorted-Neighborhood method (continued)
  - Quality of the result depends on window size and choice of the sort key
  - Window size not critical: Window sizes of ~20 have shown good results
  - Method is highly sensitive to the sort key, especially the initial characters
- ➔ Multi-pass sorted neighborhood method
  - Use multiple sort keys and apply SNM to each of them
  - Duplicates that happen to be not in the same window in one pass, are likely to be in another
  - Experiments have shown that window size can be reduced compared to single-pass method
- Union/find method
  - For each group of duplicates, a *prime representative* is selected
  - Windowed method: for each new tuple, compare against representative of other groups
    - If the new tuple is dissimilar enough it is not considered a duplicate for this group
    - If the new tuple is similar enough it is considered a duplicate for this group
    - If similarity to representative is in an uncertain range, the new tuple is compared with other members of the group to come to a decision
  - Different strategies for the selection of the representative



# Similarity Measures for Tuples

- Should return a high similarity if two tuples are probable duplicates
- Similarity measures for individual attributes differ depending on
  - Data type
    - Edit-distance- and token-based similarity measures for string comparison already discussed (➔ Schema Matching)
  - Meaning of an attribute
    - e.g. street names (abbreviations common) vs. person names (order unknown)
  - Language
  - Source of the data
    - Manually entered values: difference between letters depends on their distance on a keyboard
    - Values received orally (e.g. via telephone): assign similarity depending on how words sound
- A similarity measure for pairs of records should be based on measures for their individual components (attributes)
  - Apply individual measures on attributes and calculate a weighted measure
  - Specific rules for a record set, e.g.

```
if ((t1.birthdate – t2.birthdate) < 5days
    OR Edit-Distance(t1.birthdate,t2.birthdate) <2 )
AND sim(t1.name,t2.name) > 0.5
THEN duplicate(t1,t2)
```

# Data Fusion

- Merge the records that were identified as duplicates so that no real-world object is represented twice in the database
- Data about a RW object missing in one source can be taken from another source
- Data source can contradict each other ➔ Data conflicts
- Four types of relationships between duplicate tuples
  - Equality – tuples agree on all attributes
  - Subsumption – a tuple $t_1$ subsumes tuple $t_2$, if it has less null values than $t_2$ and agrees with $t_2$ on all non-null values
  - Complementation – two tuples complement each other, if none subsumes the other and if for each non-null value of one tuple, the other tuple either has a null value or the tuples agree on the value
  - Conflict – all other situations represent a conflict, i.e., if two duplicate tuples do not agree on at least one attribute value
- Subtlety of the semantics of null values
  - Value can be *unknown*
  - No value was applicable (*inapplicable*)
  - Value known but *withheld*
  - Many other possible interpretations
- Assumption: duplicate detection has added an ID attribute which is identical for all tuples of a duplicate group



# Data Fusion – Example Relations

**Movies**

| ID | Title | Year | Director | Runtime |
|---|---|---|---|---|
| 1 | Fight Club | 1999 | Fincher | - |
| 1 | Fight Club | 1999 | - | 142 |
| 2 | Hunt for Red October ϟ | 1990 | - | 134 |
| 2 | The Hunt for Red October | 1990 | McTiernan | - |
| 3 | The Godfather | 1972 | Coppola | 175 |
| 4 | Alien | 1978 ϟ | Scott | 117 |

**Films**

| ID | Title | Year | Director | Genre |
|---|---|---|---|---|
| 1 | Fight Club | 1999 | Fincher | - |
| 2 | The Hunt for Red October | 1990 | McTiernan | Thriller |
| 4 | Alien | 1979 | Cameron ϟ | SciFi |
| 5 | Aliens | 1986 | Cameron | SciFi |

# Data Fusion with Union Operators

- *Union* requires identical schemas of both input relations
  - Result contains all tuples that are at least within one of the input relations
  - Equality duplicates are removed
- *Outer Union* does not require identical schemas
  - Result schema is the union of the attributes of the input relations
  - Missing attributes in a tuple are assigned null values
- *Minimum Union*
  - Performs an outer union and removes all subsumed tuples
- Conflicts and Complements remain unresolved

OuterUnion (Movies, Films)

| ID | Title | Year | Director | Runtime | Genre |
|---|---|---|---|---|---|
| 1 | Fight Club | 1999 | Fincher | - | - |
| 1 | Fight Club | 1999 | - | 142 | - |
| 2 | Hunt for Red October ϟ | 1990 | - | 134 | - |
| ~~2~~ | ~~The Hunt for Red October~~ | ~~1990~~ | ~~McTiernan~~ | ~~-~~ | ~~-~~ |
| 2 | The Hunt for Red October | 1990 | McTiernan | - | Thriller |
| 3 | The Godfather | 1972 | Coppola | 175 | - |
| 4 | Alien | 1978 ϟ | Scott | 117 | - |
| 4 | Alien | 1979 | Cameron ϟ | - | SciFi |
| 5 | Aliens | 1986 | Cameron | - | SciFi |

Equality (duplicate eliminated)

Complement

**Minimum Union**

Subsumption




# Data Fusion with Join Operators

- Join
  - Use ID as join attribute
  - If no duplicates existed within the individual input relations, every RW object is represented by one tuple in the result
  - If duplicates existed, they are now multiplied
  - Tuples without a duplicate in the other relation are missing from the result
- Full Outer Join
  - All tuples of the input relations will be represented in the result
  - For tuples without join partner, the attributes coming from the other relation are filled with null values

```
SELECT M.ID, M.Title, F.Title, ...
FROM Movies M FULL OUTER JOIN Films F ON M.ID=F.ID
```

| ID | M.Title | F.Title | M.Year | F.Year | M.Director | F.Director | Runtime | Genre |
|---|---|---|---|---|---|---|---|---|
| 1 | Fight Club | Fight Club | 1999 | 1999 | Fincher | Fincher | - | - |
| 1 | Fight Club | Fight Club | 1999 | 1999 | - | Fincher | 142 | - |
| 2 | Hunt for Red October | The Hunt for Red October | 1990 | 1990 | - | McTiernan | 134 | Thriller |
| 2 | The Hunt for Red October | The Hunt for Red October | 1990 | 1990 | McTiernan | McTiernan | - | Thriller |
| 3 | The Godfather | - | 1972 | - | Coppola | - | 175 | - |
| 4 | Alien | Alien | 1978 | 1979 | Scott | Cameron | 117 | SciFi |
| 5 | - | Aliens | - | 1986 | - | Cameron | - | SciFi |

# Data Fusion with Join Operators

- Use SELECT clause to choose attributes to use in the result
  - does not resolve conflicts, but gives a preference to one data source for each attribute
  - If preferred data source has a null value for an attribute, null is returned

```
SELECT M.ID, M.Title, F.Year,
       F.Director, Runtime, Genre
FROM   Movies M FULL OUTER JOIN Films F
       ON M.ID=F.ID
```

| ID | M.Title | F.Year | F.Director | Runtime | Genre |
|---|---|---|---|---|---|
| 1 | Fight Club | 1999 | Fincher | - | - |
| 1 | Fight Club | 1999 | Fincher | 142 | - |
| 2 | Hunt for Red October | 1990 | McTiernan | 134 | Thriller |
| 2 | The Hunt for Red October | 1990 | McTiernan | - | Thriller |
| 3 | The Godfather | - | - | 175 | - |
| 4 | Alien | 1979 | Cameron | 117 | SciFi |
| 5 | - | 1986 | Cameron | - | SciFi |

- Use of COALESCE function
  - Chooses the first non-null value of all input parameters
  - Avoids nulls from preferred source, but preferred source could be wrong

```
SELECT M.ID,
       COALESCE (M.Title, F.Title),
       COALESCE (M.Year, F.Year),
       COALESCE (F.Director, M.Director),
       Runtime, Genre
FROM   Movies M FULL OUTER JOIN Films F
       ON M.ID=F.ID
```

| ID | Title | Year | Director | Runtime | Genre |
|---|---|---|---|---|---|
| 1 | Fight Club | 1999 | Fincher | - | - |
| 1 | Fight Club | 1999 | Fincher | 142 | - |
| 2 | Hunt for Red October | 1990 | McTiernan | 134 | Thriller |
| 2 | The Hunt for Red October | 1990 | McTiernan | - | Thriller |
| 3 | The Godfather | 1972 | Coppola | 175 | - |
| 4 | Alien | 1978 | Cameron | 117 | SciFi |
| 5 | Aliens | 1986 | Cameron | - | SciFi |



# Data Fusion using Merge

- Merge Operator proposed by [GPZ01]
- Completes tuples from one input relation with data from the other (and vv.)
- Duplicates are removed
- Does not lose conflicting information from the relation that is not preferred, but ... does also not lose wrong information
- Can be interpreted as an SQL representation for multi-valued attributes, maintaining all possible values for each of a real world object's properties
- Can be simulated with SQL using COALESCE and the UNION of a LEFT and a RIGHT OUTER join:

```
SELECT M.ID, COALESCE (M.Title, F.Title),
  COALESCE (M.Year, F.Year), COALESCE (
  M.Director, F.Director), Runtime, Genre
FROM Movies M LEFT OUTER JOIN Films F
  ON M.ID=F.ID
UNION
SELECT M.ID, COALESCE (F.Title, M.Title),
  COALESCE (F.Year, M.Year), COALESCE (
  F.Director, M.Director), Runtime, Genre
FROM Movies M RIGHT OUTER JOIN Films F
ON M.ID=F.ID
```

| ID | Title | Year | Director | Runtime | Genre |
|---|---|---|---|---|---|
| 1 | Fight Club | 1999 | Fincher | - | - |
| 1 | Fight Club | 1999 | Fincher | 142 | - |
| 2 | Hunt for Red October | 1990 | McTiernan | 134 | Thriller |
| 2 | The Hunt for Red October | 1990 | McTiernan | - | Thriller |
| 3 | The Godfather | 1972 | Coppola | 175 | - |
| 4 | Alien | 1978 | Scott | 117 | SciFi |
| ~~1~~ | ~~Fight Club~~ | ~~1999~~ | ~~Fincher~~ | ~~-~~ | ~~-~~ |
| ~~1~~ | ~~Fight Club~~ | ~~1999~~ | ~~Fincher~~ | ~~142~~ | ~~-~~ |
| 2 | The Hunt for Red October | 1990 | McTiernan | 134 | Thriller |
| ~~2~~ | ~~The Hunt for Red October~~ | ~~1990~~ | ~~McTiernan~~ | ~~-~~ | ~~Thriller~~ |
| 4 | Alien | 1979 | Cameron | 117 | SciFi |
| 5 | Aliens | 1986 | Cameron | - | SciFi |

## Verification with Reference Data

- In order to detect individual errors in input data, a verification with reference data can be performed, e.g.
  - Public address directories (telephone books)
- Access to reference data for commercial purposes often limited or expensive
- Some kinds of data can be checked for validity through the use of checksums
  - Credit cards
  - Bank account numbers
  - Passport numbers
  - Social security numbers



## Data Cleaning – Summary

- Creation of data cleaning mappings requires human interaction
  - Tools can suggest reasonable mappings
- Many errors can not be resolved "in batch"
  - Either we decide for one source, possibly introducing errors and losing correct data
  - Or we do not make a decision and leave conflicting duplicates in the result
- Duplicate detection and resolution introduces uncertainties
- Actual validity of individual tuples cannot reasonably be checked for all kinds of data
  - Only limited availability of reference data for specific application concepts (e.g. addresses)

# References (I)


- [BLN86] Batini, C.; Lenzerini, M. & Navathe, S.B.: A comparative analysis of methodologies for database schema integration
*ACM Comput. Surv., ACM Press,* 1986*, 18,* 323-364
- [CRF03] Cohen, W.W.; Ravikumar, P. & Fienberg, S.E.: A Comparison of String Distance Metrics for Name-Matching Tasks.
*IIWeb,* 2003, 73-78
- [HeSt98] Hernandez, M.A. & Stolfo, S.J.: Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem
Data Mining and Knowledge Discovery, 1998, 2, 9-37
- [FlWy05] Fletcher, G.H.L. & Wyss, C.M.: Relational data mapping in MIQIS
*SIGMOD 2005, ACM Press*, 912-914
- [FlWy06] Fletcher, G.H.L. & Wyss, C.M.: Data Mapping as Search.
EDBT, 2006, 95-111
- [GoDe07] Göres, J. & Dessloch, S.: Towards an Integrated Model for Data, Metadata, and Operations
BTW 2007, to appear
- [Goe05b] Göres, J.: Towards Dynamic Information Integration
First VLDB Workshop on Data Management in Grids (DMG05), Trondheim, 2005, 16-29
- [Goe05a] Göres, J.: Pattern-based Information Integration in Dynamic Environments
9th International Database Engineering Applications Symposium (IDEAS 2005), 125-134
- [GPZ01] Greco, S.; Pontieri, L. & Zumpano, E.: Integrating and Managing Conflicting Data
PSI '02, Springer-Verlag, 2001, 349-362
- [LeNa07] Leser, U. & Naumann, F.: Informationsintegration
*dpunkt Verlag,* 2007
- [LSS96] Lakshmanan, L.V.S.; Sadri, F. & Subramanian, I.N. Vijayaraman, T.M.
SchemaSQL: A Language for Interoperability in Relational Multidatabase Systems
VLDB 1996, 239-250




# References (II)


- [LSS01] Lakshmanan, L.V.S.; Sadri, F. & Subramanian, S.N.
SchemaSQL: An extension to SQL for multidatabase interoperability
*Database Systems,* 2001*, 26*, 476-519
- [RaBe01] Rahm, E. & Bernstein, P.A.
A survey of approaches to automatic schema matching
*VLDB Journal,* 2001*, 10*, 334-350
- [RaDo00] Rahm, E. & Do, H.H.
Data Cleaning: Problems and Current Approaches.
*IEEE Data Eng. Bull.,* 2000*, 23*, 3-13
- [PoBe03] Pottinger, R. & Bernstein, P.A.
Merging Models Based on Given Correspondences.
VLDB, 2003, 826-873
- [MBR01] Madhavan, J.; Bernstein, P.A. & Rahm, E.
Generic Schema Matching with Cupid
*The VLDB Journal,* 2001, 49-58
- [MGR02] Melnik, S.; Garcia-Molina, H. & Rahm, E.
Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching.
*ICDE* 2002, 117-128
- [MHH00] Miller, R.J.; Haas, L.M. & Hernández
Schema Mapping as Query Discovery
VLDB 2000, Morgan Kaufmann, 2000, 77-88
- [ScSa05] Schmitt, I. & Saake, G.
A comprehensive database schema integration method based on the theory of formal concepts.
*Acta Inf.,* 2005*, 41*, 475-524
- [WyRo05] Wyss, C.M. & Robertson, E.L.
Relational languages for metadata integration
*ACM Trans. Database Syst., ACM Press,* 2005*, 30*, 624-660
- [WyRo05b] Wyss, C.M. & Robertson, E.L.
A formal characterization of PIVOT/UNPIVOT
CIKM 2005, ACM Press, 602-608