Chapter 4
Information Integration

Outline

- Information Integration Tasks
- Schema Matching
  - Classification of Approaches
  - Example: Cupid
- Schema Merging
  - Example: Rondo
- Integration Planning
  - Example: Clio
- Deployment
  - Example: Orchid
  - Incremental loading of DW
- Data Integration
  - Data Quality Problems
  - Causes and Consequences
  - Data Cleaning Approaches
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)
  - Multi-database languages (schematic heterogeneity, technical 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)
  - Data Cleaning/Fusion (syntactic heterogeneity, semantic heterogeneity (in data))
  - focus on integration planning

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
  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
  8. Deployment/Integration Plan Implementation
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
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 (chema 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 $n = m$: $O(n^2)$
  - Even higher complexity if sets of elements are compared ($O(2^n)$), e.g. to obtain 1:n/n:m matches

- 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

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”, “oInf”, “nfor”, “orm”, “rma”, “mat”, “ati”, “tio”, “tion”)
    - Jaccard similarity – describes the similarity of two sets
      - JaccardSimilarity(A, B) = \|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: 
      \[
      w_s(t) = tf_s(t) \cdot idf(t)
      \]
    - 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
      \[
      \cosine(S_1, S_2) = \frac{\sum t \in S_1 \cdot w_{s_1}(t) \cdot w_{s_2}(t)}{\sqrt{\sum t \cdot w_{s_1}(t)^2 \cdot \sum t \cdot w_{s_2}(t)^2}}
      \]
  - **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.metamodell.com/article.php?story=20030115211223271](http://www.metamodell.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 $\text{lsim}(s,t)$ 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 $\text{lsim}$ 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 $\text{lsim}(s,t)$ in the range $[0,1]$.

Cupid Linguistic Matching – Problems

- 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.

(not all matches shown)
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 \( th_{\text{accept}} \)
  - based on weighted 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 \( \text{wsim}(s,t) \):
    \[
    \text{wsim}(s,t) = w_{\text{struct}} \cdot ssim(s,t) + (1-w_{\text{struct}}) \cdot \text{lsim}(s,t)
    \]
  - If \( \text{wsim}(s,t) \) is above threshold \( th_{\text{high}} \), increase the similarity of each pair of leaves in the subtrees of \( s \) and \( t \) by a factor \( c_{\text{inc}} \) (not exceeding \( 1 \))
  - If \( \text{wsim}(s,t) \) is below threshold \( th_{\text{low}} \), decrease the similarity of each pair of leaves in the subtrees of \( s \) and \( t \) by a factor \( c_{\text{dec}} \) (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:**
  - $\theta_{th_{eq}} = 0.5$
  - $w_{struct} = 0.7$
  - $\theta_{th_{eq}} = 0.7$, $c_{inc} = 1.2$
  - $\theta_{low} = 0.3$, $c_{dec} = 0.8$

Cupid Structural Matching – Example (cont.)

- **Iteration for $s = \text{Emp}$, $t = \text{Employee}$:**
  - 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
    - $\text{ssim}(s,t) = \frac{6}{7} = 0.9$
  - Calculate wsim:
    - $\text{wsim}(s,t) = w_{struct} \cdot \text{ssim}(s,t) + (1-w_{struct}) \cdot \text{lsim}(s,t)$
    - $= 0.7 \cdot 0.9 + 0.3 \cdot 0.9 = 0.9$
  - Modify structural similarity for leaves of $s$ and $t$:
    - $\text{wsim}(s,t) = 0.9 > \theta_{th_{eq}} = 0.7$
    - Increase ssim for each pair $(l_s, l_t)$,
      $l_s \in \text{leaves}(s)$ and $l_t \in \text{leaves}(t)$:
      $\text{ssim}_{new}(l_s, l_t) = \text{ssim}_{old}(l_s, l_t) \cdot c_{inc} = 0.5 \cdot 1.2 = 0.6$
      (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)
Iteration for
s = Emp, t = Department:
- Calculate ssim:
  \[ ssim(s,t) = \frac{2}{7} = 0.3 \]  
  (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:
  \[ wsim(s,t) = w_{\text{struct}} \cdot ssim(s,t) + (1-w_{\text{struct}}) \cdot lsim(s,t) = 0.7 \cdot 0.3 + 0.3 \cdot 0.0 = 0.21 = 0.2 \]
- Modify structural similarity for leaves of s and t:
  \[ wsim(s,t) = 0.2 < th_{\text{low}} = 0.3 \rightarrow \text{decrease ssim for each pair (l}_s, l}_t), l}_s \in \text{leaves(s)} \text{ and } l}_t \in \text{leaves(t)}: \]  
  \[ ssim_{\text{new}}(l}_s, l}_t) = ssim_{\text{old}}(l}_s, l}_t) \cdot c_{\text{dec}} \]  
  (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
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
- Collecting 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 $(E, \text{Root}, \text{Re})$, with $E$ being a set of elements, $\text{Root} \in E$ being the root element of the model, and $\text{Re}$ 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 ∈ E 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.ID, B.Film.ID, MapAB.#m8)
G2 (A.Movie.ID, B.Film.ID, MapAB.MID)
G3 (A.Movie.Title, B.Film.Title, MapAB.#m1)
G4 (A.Movie.Genre)
G5 (A.Role)
G6 (A.Role.Name, B.Film.Actor.Role, MapAB.Rolename)
G7 (A.Role.Desc)
G8 (A.Actor, B.Film.Actor, MapAB.#m7)
G9 (A.Actor.Name, MapAB.ActorName)
G10 (A.Actor.ID)
G11 (A.Actor.Bio)
G12 (B.Film.Actor.Firstname, MapAB.#m4)
G13 (B.Film.Actor.Lastname, MapAB.#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 container 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))
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 [FIWy06]
  - 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 \( v = (f, p) \) with
    - a function \( f \) describing how to derive a certain target attribute \( B \) from a set of source attributes \( A \) (and possibly from source metadata):
      \[ f: \text{dom}(A_1) \times \text{dom}(A_2) \times \ldots \times \text{dom}(A_q) \rightarrow \text{dom}(B) \]
    - a filter \( p \) indicating which source values should be used:
      \[ p: \text{dom}(A_1) \times \text{dom}(A_2) \times \ldots \times \text{dom}(A_r) \rightarrow \text{boolean} \]
  - 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 paths 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
     - 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 queries and combined query:
  - \( Q_1 \) selects titles, year, director, and sum of pay from movies and actors where movie ID equals actor ID, grouped by title, year, and director, having sum of pay greater than 10M and union all
  - \( Q_2 \) selects title, year, null, and budget from films where genre is not documentary

```
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
```

```
SELECT Title, Year, null, Budget
FROM S2.Film
WHERE genre <> "Documentary"
```

WS 2011/2012
Information Integration Middleware

- Multitude of middleware systems and architectures
  - Major approaches:
    - logical (virtual) integration
    - federated DBMS, multi-database systems
    - data processing specified using SQL, XQuery, ...
    - physical (materialized) integration
    - data replication, data warehousing, ETL (extract-transform-load), XML transformations, message brokering
    - utilizes ETL "scripts" based on (product-specific) dataset processing operators

- Technologies
  - differ in terms of
    - functional properties (data processing specification, expressive power)
    - non-functional properties (target response times, data currency)
  - are often used in combination, involving several product platforms

- Complex development /deployment tasks!
  No common language for platform-independent integration plan!

An Abstract Data Set Processing Model

- Idea: provide a generic model for describing data set processing
  - abstract data set model
    - structural properties (schema): flat & nested relations, XML
    - data access properties: associative vs. sequential, persistent vs. transient, sorting/grouping properties, update properties...
    - should also cover data streams, XML feeds
  - abstract processing model
    - platform-independent data processing operators
    - starting point: extended relational algebra
    - should also cover XML processing, data cleansing operations, propagation of source updates
    - used to specify an integration plan in a platform-independent manner
Major Advantages

- **Modeling, visualizing, and reasoning about data processing independent of a deployment platform**
- **Top-down development**
  - choice of platform often based on non-functional requirements
  - suggested by system, or determined by user
  - automatic generation of target platform artifacts during deployment
    - ETL scripts, queries and view definitions, replication setup, ...
    - initial load vs. incremental load (considering updates, insertions, deletions on data sources)
- **Optimization opportunities**
  - logical (algebraic) optimization
  - choice of deployment platform(s) for operator subgraphs
    - e.g., push part of processing into the DBMS at the source or target
  - platform-dependent optimization
    - e.g., chose the most suitable ETL operator
- **Active area of research**

Orchid

- **Research project at IBM Almaden [HDWRZ08]**
- **Links different phases, levels of abstraction in information integration**
  - Mappings, mapping interpretations (→ Clio)
  - Abstract data set processing model (OHM – Operator Hub Model)
  - Deployment platforms
    - main focus initially on ETL
- **In parts already reflected in IBM products**
  - IBM Information Server v8.0.1
Orchid Architecture

ETL System

Mapping System

External Layer (Layer 1)

Intermediate Layer (Layer 2)

Primitives Layer (Layer 3)

ETL Product Level

dsx DataStageX

ETL Representation Level

Stage 1 → Stage 2 → Stage 3

Flow Mappings

A → B, B → C, ...

A → C, ...

Stages Graph Model

Operators Graph

Operator Hub Model (OHM)

Mapping Product Level (e.g., Clio, RDA)

MSL

OHM Operators

- Based on Relational Algebra operators
  - Initial focus was relational data transformation
  - Simple and well-known semantics (30+ years of history)
  - Plenty of well-known query graph representations, query optimizations, query rewrite techniques.
- Main OHM operators:

<table>
<thead>
<tr>
<th>OPERATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILTER</td>
</tr>
<tr>
<td>JOIN</td>
</tr>
<tr>
<td>UNION</td>
</tr>
<tr>
<td>GROUP</td>
</tr>
<tr>
<td>PROJECT</td>
</tr>
<tr>
<td>UNNEST</td>
</tr>
<tr>
<td>BASIC PROJECT</td>
</tr>
<tr>
<td>KEYGEN</td>
</tr>
<tr>
<td>COLUMN SPLIT</td>
</tr>
<tr>
<td>COLUMN MERGE</td>
</tr>
<tr>
<td>NEST</td>
</tr>
</tbody>
</table>
Deployment: Multiple-runtime deployment

- OHM plan can be deployed into multiple runtimes
  - Optimization is an issue
Supporting Incremental Loading [JoDe08]

- OHM instance as starting point
- Replace basic OHM operators with *incremental* variants
- Incremental operators are composed of basic OHM operators
- Leverage Orchid’s optimization and deployment facilities

Change Data Propagation

- Interface between Change Data Capture and Change Data Application
- Given CDC limitations, what CDA requirements are satisfiable?
- Given CDA requirements, what CDC limitations are acceptable?
- What data transformations are to be performed for change data propagation?
Change Data Model

- Given dataset $D$
- change data is $(\triangle D, \triangledown D, \boxplus D, \boxminus D)$
  - $\triangle D$ denotes insertions
  - $\triangledown D$ denotes deletions
  - $\boxplus D$ denotes updates (current state)
  - $\boxminus D$ denotes updates (initial state)

- CDC limitations
  - Partial change data results from CDC limitations
  - Missing change data
  - Indistinguishable changes

- Audit columns: $(\triangle D \cup \boxplus D)$ or $\triangle D, \boxplus D$
- Snapshot differentials: $\triangle D, \triangledown D, \boxplus D$
- Log-based CDC: $\triangle D, \triangledown D, \boxplus D, \boxminus D$

Incremental OHM Instance

(\triangle P \cup \boxplus P) \rightarrow Incremental Join

$P_{new}$

$Q_{new}$ \rightarrow \boxminus Q

Incremental Join

(\triangle D \cup \boxplus D)
Summary - Deployment

- Challenge: complexity of implementing an integration solution
  - approaches: virtual vs. materialized – or combinations thereof
  - different middleware platforms
  - complex to use
  - no common language for platform-independent integration plans
- Goal: support an abstract data and transformation model
  - platform-independent, top-down development
  - (cross-platform) optimization
- Orchid
  - Links mapping tools and transformation (ETL) platforms using operator hub model, OHM
  - Generates ETL scripts from mapping specifications (and vice versa)
  - Can deploy to combination of multiple platforms (e.g., DBMS pushdown + ETL)
- Incremental operators
  - Model for (partial) change data
  - Generation of incremental load processes based on
    - CDC limitations, CDA requirements, Source properties and schema constraints
  - Leverage Orchid’s deployment facility

Data Integration

- Data Quality Problems
- Causes and Consequences
- Data Cleaning
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 referring 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:
    - ~10^1 - 10^9 schema elements, but 10^2 - 10^9+ instances
    - Resolving data quality ("Data Cleaning") problems is extremely expensive
    - Today usually only done in replicating/materialized integration systems

Classification of Data Quality Problems

- based on [RaDo00, LeNa07]
- 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", Chancellor="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(City="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. Source2.Person(ID="1234", DoB="1983-06-03")
- Differing representations
  - e.g. Source1.Emp(ID="1234", Job="Sales Mgr.") vs. Source2.Emp(ID="1234", Job="S24")
- Different physical units
  - e.g. Source1.Person(Name="Herbert Meier", height="183") [cm] vs. Source2.Person(Name="Herbert Meier", height="72") [inches]
- Different precision
  - e.g. Source1.Movie(Title="Fight Club", runtime="2h19min") vs. Source2.Movie(Title="Fight Club", runtime="2h19min12sec")
- Different levels of details
  - e.g. "all actors" vs. "only main cast"
Handling Data Quality Problems

- **Phase 1: Data Scrubbing (individual records)**
  - Resolve errors within individual tuples/data items
  - Normalise data
    - unify case, stemming, stopword removal, acronym expansion
    - Formatting: 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

- **Phase 2: 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 partition
  - Data Fusion
    - Combine detected duplicates into one consistent tuple
      - 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 null value semantics (unknown, inapplicable, withheld ...)

Handling Data Quality Problems (II)
Data Cleaning – Summary

- Creation of data cleaning mappings requires human interaction
  - Tools can suggest reasonable mappings
- Many errors cannot 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)

  SIGMOD Record, ACM Press, 2000, 29, 55-64
  SIGMOD, 2003, 73-78
- [FHYv04] Fletcher, G.H.L. & Wyss, C.M.: Relational data mapping in MIQIS
  SIGMOD-2005, ACM Press, 913-914
- [FHYv06] Fletcher, G.H.L. & Wyss, C.M.: Data Mapping as Search.
  EDBT, 2006, 95-111
  BTW, 2007, to appear
- [GoNe06b] Göres, J.: Towards Dynamic Information Integration
  First VLDB Workshop on Data Management in Grids (OMG25), Trondheim, 2005, 16-29
- [GoNe06a] Göres, J.: Pattern-based Information Integration in Dynamic Environments
  9th International Database Engineering Applications Symposium (IDEAS 2005), 125-134
  PSI 102, Springer-Verlag, 2001, 349-362
  Orchid: Integrating Schemat Mapping and ETL.
  Proc. 24th International Conference on Data Engineering, April 7-12, 2008, Cancún, México
- [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
- [JoDe08] Jörg, T; Dessloch, S.: Towards Generating ETL Processes for Incremental Loading
  12th Int. Database Engineering & Applications Symposium (IDEAS 2008), 2008
- [LeNa07] Leon, U. & Naumann, F.: Information Integration
  dpunkt Verlag, 2007
- [SSS06] Lakhdar, M.; Sadi, F. & Subramanian, J.N. Vijayararaman, T.M.
  SchemaSQL: A Language for Interoperability in Relational Multidatabase Systems
  VLDB 1996, 235-250
References (II)


