Data integration

Data sources:
- any format/data model

Wrappers:
- relational or XML
- data/query translation, data publishing
- using source query interfaces

Mediators:
- restructuring, merging, conflict resolution,...
- eager or lazy
- schematic discrepancies?

Relational data integration

Data Integration System:
- target (integrated) schema
- source schema (maybe more than one)
- assertions relating elements of the global schema to elements of the source schema(s)

Target and source schemas may contain integrity constraints.
Assertions

Source-to-target dependencies:
\[ \forall t. \phi_S(t) \Rightarrow \phi_T(t). \]

Slide 3  Local-as-view:
\[ \forall t. R(t) \Rightarrow \phi_T(t). \]

Global-as-view:
\[ \forall t. \phi_S(t) \Rightarrow R(t). \]

Data integration vs. exchange

Data integration:
- source schema given
- target schema and/or assertions to be constructed
- target instance corresponding to the given source instance may or may not be materialized

Slide 4  Data exchange:
- source and target schemas given
- assertions to be constructed
- target instance needs to be materialized
Problems

Schema matching.

Generation of assertions.

Data reconciliation:
- underspecification: selecting the target instance (uniqueness, nulls)
- overspecification: what if the target constraints cannot be satisfied?
- ambiguity: object identification (record linkage)

Schematic discrepancies:
- mixed schema-instance mappings
- cannot be formulated using first-order formulas

Heuristics for schema matching and data cleaning

Using names:
- name matching, similarity
- synonyms
- homonyms

Using constraints:
- data types and ranges
- key/uniqueness
- keys/foreign keys: for combining different relations

Using instances:
- overlapping sets of values
Schema matching

Finding a "best" mapping between the schemas:
- using thesauri and topological adjacency (CUPID)
- neural networks and machine learning (LSD)
- fixpoint computation (Similarity Flooding)

Similarity Flooding [Melnik et al., 2002]

Schemas:
- represented as weighted directed graphs
- connectivity and propagation edges
- relational, XML, ontologies,...

Algorithm:
1. construct an initial mapping $\sigma^0$, consisting of weighted pairs of nodes in two schemas
2. adjust the mapping based on neighborhood information
3. repeat Step 2 if necessary
4. filter the result
Adjustment and termination

\[ \sigma^{i+1}(x, y) = \sigma^i(x, y) \\
+ \sum_{(a_u, p, x) \in A, (b_u, p, y) \in B} \sigma^i(a_u, b_u) \cdot w((a_u, b_u), (x, y)) \\
+ \sum_{(x, p, a_v) \in A, (y, p, b_v) \in B} \sigma^i(a_v, b_v) \cdot w((a_v, b_v), (x, y)). \]

Slide 9  The new values are normalized to \([0, 1]\).

Termination:
- when the changes to the mapping are below a threshold.
- after a fixed number of iterations.
- guaranteed for strongly connected graphs.

Schematic discrepancies

The information in the schema of one database may correspond to the information in the instance of another database.

Postulates:

Slide 10

1. the same constant may be a relation name, a column name and an attribute value
2. schema elements should be first-class objects
3. view definitions may define more than one relation, with varying number of columns.
SchemaSQL

[Lakshmanan et al, 1996-2001].

Variables ranging over:
- relation labels
- column labels
- tuples
- attribute values
- database labels.

Variables can appear where the constants can.

Brokerage schema:

ibm(date,xge,open,close,low,high)
mst(date,xge,open,close,low,high) ... stock(ticker,busType)

SchemaSQL restructuring view definition:

CREATE VIEW X(date, priceType, S) AS
SELECT T.date, PT, T.PT
FROM ->S, S T, S->PT, T.xge X, stock U
WHERE PT <> date AND PT <> xge
AND S <> stock AND S = U.ticker
AND U.busType = 'tech'

The number of relations defined by this view and the number of columns in each relation are not fixed!
Implementation

Translation to relational algebra extended with new operators: UNFOLD, FOLD, SPLIT, and UNITE.

<table>
<thead>
<tr>
<th>stock</th>
<th>xge</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>att</td>
<td>nyse</td>
<td>25.80</td>
</tr>
<tr>
<td>lucent</td>
<td>nyse</td>
<td>15.67</td>
</tr>
<tr>
<td>att</td>
<td>tse</td>
<td>27.04</td>
</tr>
<tr>
<td>lucent</td>
<td>tse</td>
<td>13.40</td>
</tr>
</tbody>
</table>

UNFOLD\(xge, price\)

<table>
<thead>
<tr>
<th>stock</th>
<th>nyse</th>
<th>tse</th>
</tr>
</thead>
<tbody>
<tr>
<td>att</td>
<td>25.80</td>
<td>27.04</td>
</tr>
<tr>
<td>lucent</td>
<td>15.67</td>
<td>13.40</td>
</tr>
</tbody>
</table>

The reverse operation is FOLD by price on xge.

<table>
<thead>
<tr>
<th>stock</th>
<th>xge</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>att</td>
<td>nyse</td>
<td>25.80</td>
</tr>
<tr>
<td>lucent</td>
<td>nyse</td>
<td>15.67</td>
</tr>
<tr>
<td>att</td>
<td>tse</td>
<td>27.04</td>
</tr>
<tr>
<td>lucent</td>
<td>tse</td>
<td>13.40</td>
</tr>
</tbody>
</table>

SPLIT\(xge\)

<table>
<thead>
<tr>
<th>nyse</th>
<th>att</th>
<th>25.80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>lucent</td>
<td>15.67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tse</th>
<th>att</th>
<th>27.04</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>lucent</td>
<td>13.40</td>
</tr>
</tbody>
</table>

The reverse operation is UNITE by xge.
Handling schematic discrepancies in XSB

Variables and terms may be used instead of predicate names, as long as they are “hilogized”.

Only relations of fixed arity can be defined.

\[
\text{SPLIT by } xge \text{ of the table close:}
\]

\[
:- \text{hilog } xge.
\]

\[
xge(X)(\text{Stock,Price}) :- \text{close(Stock,X,Price)}.
\]

\[
\text{UNITE by } xge:
\]

\[
:- \text{hilog } nyse.
\]

\[
:- \text{hilog } tse.
\]

\[
\text{close(Stock,X,Price)} :- X(\text{Stock,Price}).
\]

Data quality problems

Context: building a data warehouse.

Single-source:

- schema-level:
  - integrity constraints not enforced
  - poor schema design
- instance-level (cannot be prevented at the schema level):
  - missing values
  - misfielded values
  - text: misspellings, abbreviations, transpositions, embedded values
  - duplicated records
Additional **multiple-source** problems:

- schema-level:
  - naming conflicts
  - structural conflicts
- instance-level:
  - different representations or interpretations of values
  - different aggregation levels or reference points
  - duplicates
  - contradictory values

**Conflict resolution**

Schema-driven:

- definition of appropriate views
- domain tools (e.g., addresses)
- specialized transformations (AJAX) formulated in an extension of SQL:
  - mapping
  - matching (approximate join)
  - clustering
  - merging
- workflows of transformations
Duplicate detection

Matching:
- a matching rule
- multi-pass sorting on different attributes
- examining tuples in a fixed window
- combining the results of different passes using transitive closure of the “is-duplicate-of” relationship

Degree of similarity:
- attributes: domain-dependent similarity functions
- tuples: combination of weighted attribute similarity functions
- “similar” ≡ degree of similarity above a threshold
- implementation: how to avoid computing Cartesian product

Edit distance

Input: two strings \( v \) and \( w \).
Output: a matrix \( D[m,n] \), \( m = |v| \), \( n = |w| \), such that \( D[i,j] \) is equal to the minimum number of character operations (change, insert, delete) necessary to transform \( v[1..i] \) into \( w[1..j] \).

Algorithm [Levenshtein, 1966]:

\[
\begin{align*}
D[0, 0] &= 0 \\
D[i, 0] &= i, \quad i=1..|v| \\
D[0, j] &= j, \quad j=1..|w| \\
\end{align*}
\]

\[
D[i,j] = \min( D[i-1,j-1] \\
+ \text{if } v[i]=w[j] \text{ then 0 else 1 fi}, \\
D[i-1, j] + 1, \\
D[i, j-1] + 1 ), \quad i=1..|v|, \quad j=1..|w|
\]