POLITECNICO
MILANO 1863

Data integration

(second part)

Cinzia Cappiello
A.A. 2023-2024

These slides are based on Prof. Tanca slides.

Designing data integration in the Multidatabase (with a unique data model)

1. Source schema identification (when present)
2. Source schema reverse engineering (data source conceptual schemata)
3. Conceptual schemata integration and restructuring: conflict resolution and restructuring
4. Conceptual to logical translation (of the obtained global schema)
5. Mapping between the global logical schema and the single schemata (logical view definition)
6. After integration: query-answering through data views

Mapping between the global logical (mediated) schema and the single source schemata.

In general:

- Formally, a data integration system is a triple
  (G, S, M)
- where
  - G is the global pattern,
  - S is the heterogeneous set of sources, and
  - M is the mapping associating queries between the sources and the global schema
- A query to the integrated system is posed in terms of G and specifies which data of the virtual database we are interested in
- The problem is understanding *which real data (in the data sources) correspond to those virtual data*

Mapping between the global logical (mediated) schema and the single source schemata (logical view definition)

- Two basic techniques
  - **GAV (Global As View)**
  - **LAV (Local As View)**
- The same approach can be used also in case of different data models
- In that case also a **model transformation** is required (we'll see it later)

GAV (Global As View)

- Up to now we supposed that the global schema *be derived* from the integration process of the data source schemata
- Thus the global schema is *expressed in terms of the data source schemata*
- Such approach is called the *Global As View approach*

5

The other possible ways

LAV (Local As View)

- The global schema has been designed independently of the data source schemata
- The relationship (mapping) between sources and global schema is obtained by defining each data source as a view over the global schema

GLAV (Global and Local As View)

- The relationship (mapping) between sources and global schema is obtained by defining a set of views, some over the global schema and some over the data sources

6

GAV

- A GAV mapping is a set of assertions, one for each element *g* of ***G***

$$g \rightarrow q_S$$

That is, the mapping specifies g as a query q_S over the data sources. This means that the mapping (view) tells us exactly how the element g is computed.

- OK for stable data sources
- Difficult to extend with a new data source

8

GAV example

SOURCE 1

Product(Code, Name, Description, Warnings, Notes, CatID)
Category(ID, Name, Description)
Version(ProductCode, VersionCode, Size, Color, Name, Description, Stock, Price)

SOURCE 2

Product(Code, Name, Size, Color, Description, Type, Price, Q.ty)
TYpe(TypeCode, Name, Description)

note: here we do not care about data types…

9

SOURCE 1

Product(Code, Name, Description, Warnings, Notes, CatID)

Version(ProductCode, VersionCode, Size, Color, Name, Description, Stock, Price)

SOURCE 2

Product(Code, Name, Size, Color, Description, Type, Price, Q.ty)

GLOBAL SCHEMA

```
CREATE VIEW GLOB-PROD AS
SELECT Code AS PCode, VersionCode
    as VCode, Version.Name AS Name,
    Size, Color, Version.Description as
    Description, CatID, Price, Stock
FROM SOURCE1.Product,
    SOURCE1.Version
WHERE Code = ProductCode
UNION
SELECT Code AS PCode, null as
    VCode, Name, Size, Color,
    Description,Type as CatID, Price,
    Q.ty AS Stock
FROM SOURCE2.Product
```

10

Query processing in GAV

QUERY OVER THE GLOBAL SCHEMA

```
SELECT PCode, VCode, Price, Stock
FROM GLOB-PROD
WHERE Size = "V" AND Color = "Red"
```

In this particular case the transformation is easy, since the combination operator is a UNION → push selections through union!!

```
SELECT Code, VersionCode, Price, Stock
FROM SOURCE1.Product, SOURCE1.Version
WHERE Code = ProductCode AND Size = "V" AND Color = "Red"
UNION
SELECT Code, null, Price, Q.ty
FROM SOURCE2.Product
WHERE Size = "V" AND Color = "Red"
```

11

11

Further considerations on GAV

- Suppose we introduce a new source
- The simple views we have just created must be modified
- In the simplest case (like in the GLOB_PROD case) we only need to add a union with a new SELECT-FROM-WHERE clause
- This is not true in general, view definitions may be much more complex
- In LAV this problem does not occur

LAV

A mapping LAV is a set of assertions, one for each element *s* of each source S

$$s \rightarrow q_G$$

Thus the content of each source is characterized in terms of a view q_G over the global schema

- OK if the global schema is stable, e.g. based on a domain description (e.g. an ontology) or an enterprise model
- It favours extensibility
- Query processing much more complex

A LAV example

SOURCE 1

Product(Code, Name, Description, Warnings, Notes, CatID)

Version(ProductCode, VersionCode, Size, Color, Name, Description, Stock, Price)

SOURCE 2

Product(Code, Name, Size, Color, Description, Type, Price, Q.ty)

GLOBAL SCHEMA

GLOB-PROD (PCode,VCode, Name, Size, Color, Description, CatID, Price, Stock)

- In this case we have to express the source tables as views over the global schema

14

GLOBAL SCHEMA

GLOB-PROD (PCode, VCode, Name, Size, Color, Description, CatID, Price, Stock)

SOURCE 2

Product (Code, Name, Size, Color, Description, Type, Price, Q.ty)

```
CREATE VIEW SOURCE2.Product AS
SELECT PCode AS Code, Name, Size, Color, Description, CatID
   as Type, Price, Stock AS Q.ty
FROM GLOB-PROD
```

15

GLOBAL SCHEMA

GLOB-PROD (PCode, VCode, Name, Size, Color, Description, CatID, Price, Stock)

SOURCE 1

Product(Code, Name, Description, Warnings, Notes, CatID)

Version(ProductCode, VersionCode, Size, Color, Name, Description, Stock, Price)

CREATE VIEW SOURCE1.Product AS
SELECT Pcode AS Code,
Name?,Description?,
Warnings ?, Notes?, CatID
FROM GLOB-PROD

% in the global schema the attributes with a question mark are taken from the VERSION table, so where do we find the ones belonging to the PRODUCT table ?

CREATE VIEW SOURCE1. Version AS
SELECT Pcode AS ProductCode,
VCode as VersionCode, Size, Color, Name, Description , Stock, Price)
FROM GLOB-PROD

16

Notes

- Some information is lacking: there is no corresponding value (e.g. Warnings, Notes, etc..).
- On the other hand, no query coming from the global schema will ever require those attributes, since they are not present in the global schema

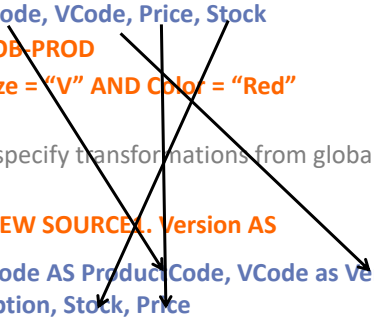
17

17

Query processing in LAV

- Queries are expressed in terms of the global schema, therefore the needed transformation is the inverse of the defined views: some **REASONING** is needed

SELECT PCode, VCode, Price, Stock
FROM GLOB-PROD
WHERE Size = "V" AND Color = "Red"



- Views specify transformations from global to sources, thus:

CREATE VIEW SOURCE1. Version AS

SELECT Pcode AS ProductCode, VCode as VersionCode, Size, Color, Name, Description, Stock, Price
FROM GLOB-PROD

- Do the same for **SOURCE 2.Product**

18

Query processing in LAV

- Information is in the sources, while the mapping, i.e. the view definition, specifies the opposite transformation w.r.t. the one we need!
- Not possible to obtain answers by a simple, obvious or direct computation of the query definition
- No simple rule or view unfolding for the relations in the body as in GAV
- A *plan* is a rewriting of the query as a set of queries to the sources and a prescription of how to combine their answers
- Both sources are implicitly needed: the system must extract the necessary information from SOURCE1.VERSION and from SOURCE2.PRODUCTS

19

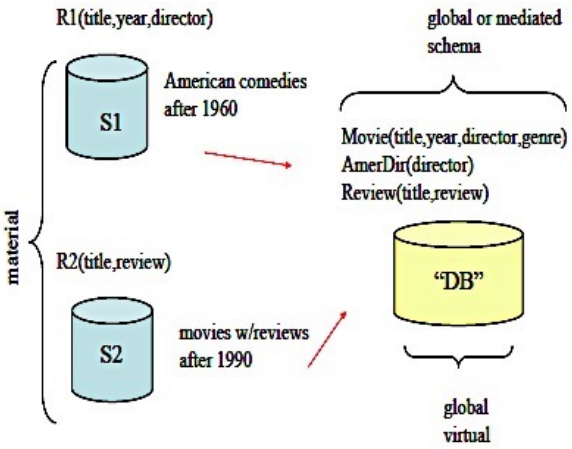
GAV

- Mapping quality depends on how well we have compiled the sources into the global schema through the mapping
- Whenever a source changes or a new one is added, the global schema needs to be reconsidered

LAV

- Mapping quality depends on how well we have characterized the sources
- High modularity and extensibility (if the global schema is well designed, when a source changes or is added, only its definition is to be updated)
- Query processing needs *reasoning*

Example (Bertossi)



Example (Bertossi)

Global schema G:

Movie(Title, Year , Director , Genre), AmerDir (Director),
Review(Title, Rev)

With LAV, the sources S1, S2 have their local relations defined as views.

S1: has R1, containing comedies, filmed after 1960, with American directors and the year they came out:

R1(Title, Year , Director) ← Movie(Title, Year , Director ,Genre),
AmerDir (Director), Genre = comedy, Year ≥ 1960.

S2: has R2, containing movies filmed after 1990 with their reviews, but no directors

R2(Title, Rev) ← Movie(Title, Year, Director, Genre),
Review(Title, Rev), Year ≥ 1990.

23

Query processing in LAV

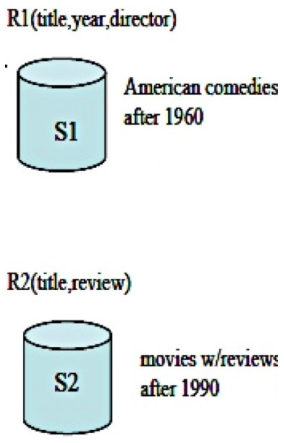
Query over **G**: ?Comedies with their reviews filmed since 1950?

Ans (Title, Rev) ← Movie(Title, Year , Director, "comedy"),
Review(Title, Rev), Year ≥ 1950

• The query is rewritten in terms of the views and can be computed:

1. Extract values for Title from V1 (it contains *the comedies*)
2. Extract the tuples from V2 (it contains *the reviews*)
3. At the global level, compute the join via Title

Due to the limited contents of the sources, we only obtain comedies by American directors filmed after 1990 with their reviews



24

Source incompleteness

- Besides the info in **S2**, there could be other information about movies after 1990 with their reviews, not contained in **S2**
- Thus, the information provided by **S2** is somehow "incomplete" wrt what G "expresses" (and might contain)
- In general, a data source could contain only a part of the information of the same kind "expected" in the global system
- This is the most common scenario: sources are assumed to be *incomplete* w.r.t. the global schema

The most useful (and used) integration operators to write relational GAV views

- Union
- Outerunion
- Join
- Outerjoin
- Generalization

OPERATORS: EXAMPLES

R1

SSN	NAME	AGE	SALARY
123456789	JOHN	34	30K
234567891	KETTY	27	25K
345678912	WANG	39	32K

SSN	NAME	SALARY	PHONE
234567891	KETTY	20K	1234567
345678912	WANG	22K	2345678
456789123	MARY	34K	3456789

R2

R1 OU R2

SSN	NAME	AGE	SALARY	PHONE
123456789	JOHN	34	30K	NULL
234567891	KETTY	27	25K	NULL
345678912	WANG	39	32K	NULL
234567891	KETTY	NULL	20K	1234567
345678912	WANG	NULL	22K	2345678
456789123	MARY	NULL	34K	3456789

R1 JOIN R2

SSN,NAME

SSN	NAME	AGE	SALARY1	SALARY2	PHONE
234567891	KETTY	27	25K	20K	1234567
345678912	WANG	39	39K	22K	2345678

OPERATORS: EXAMPLES (2)

R1

SSN	NAME	AGE	SALARY
123456789	JOHN	34	30K
234567891	KETTY	27	25K
345678912	WANG	39	32K

SSN	NAME	SALARY	PHONE
234567891	KETTY	20K	1234567
345678912	WANG	22K	2345678
456789123	MARY	34K	3456789

R2

R1 OJ R2

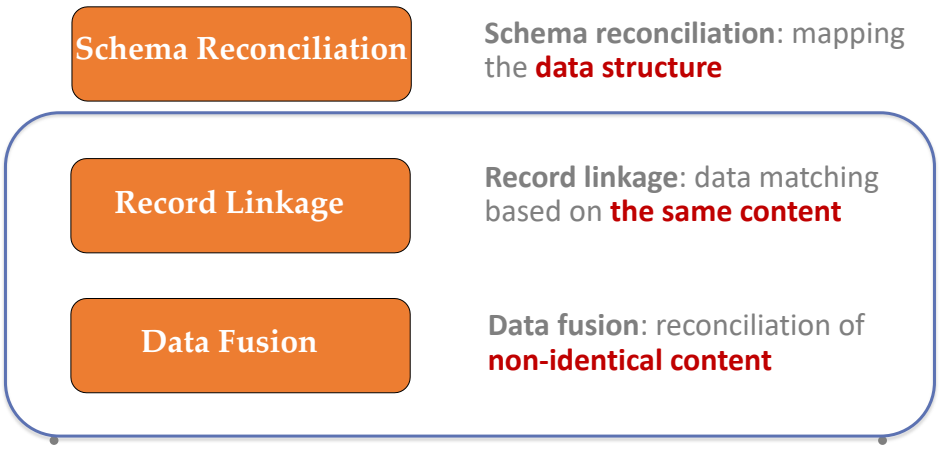
SSN	NAME	AGE	SALARY	PHONE
123456789	JOHN	34	30K	NULL
234567891	KETTY	27	??	1234567
345678912	WANG	39	??	2345678
456789123	MARY	NULL	34K	3456789

R1 Ge R2

SSN	NAME	SALARY
234567891	KETTY	??
345678912	WANG	??
123456789	JOHN	30K
456789123	MARY	34K

What shall we do about these "uncertain" values ?

The steps of Data Integration: how is it done?



34

Be there a schema or not, we may have inconsistencies in the data

At query processing time, when a real-world object is represented by instances in different databases, they may have different values.

- Record Linkage (aka Entity Resolution): finding the info that refer to same real-world entities.
- Data Fusion: once recognized that two items refer to the same entity, how do we reconcile inconsistent information?

We have understood already that data integration often needs finding strings that refer to the same real-world entities:

- "Politecnico di Milano" and "Politecnico Milano"
- "Via ponzio 34" and "Via Ponzio 34/5"

35

Record Linkage
(aka Entity Resolution)

- Whatever the data model, we have to recognize when two datasets contain the same information
- We refer again to the relational data model because it is the most common case but also because the problem is pretty much the same
- Consider the case of relational databases. Two tuples could be compared as if they were two strings of characters:

Tanca	Letizia	DEIB	Leonardo

Tanca Letizia DEIB Leonardo

String matching vs. tuple or (more generally) structured-data matching

Tanca	Letizia	DEIB	Leonardo

Tanca Letizia DEIB Leonardo vs. Tanca Letizia DEIS Lonardo

- If the fields are distinct, it is much easier to spot similarities ! E.g. that a word like Lonardo is a wrong representation of Leonardo, because we know that Leonardo is a campus of Politecnico. The same would happen with DEIB.
- We wouldn't be able to teach this knowledge to the system if the tuple were treated as a single string

Let us start by matching strings

- Typing errors, e.g., Giovanni Rossi vs. Givanni Rossi
- Different representation conventions, e.g., Sept 20th vs. 20/9
- Nicknames or abbreviations, e.g., Giuseppe vs. Beppe
- String inversion, e.g., Giovanni Rossi vs Rossi, Giovanni
- Abbreviations, e.g. Politecnico di Milano vs PoliMI

38

String similarity and similarity measures

- Given two sets of strings, find all pairs from the two sets that refer to the same real-world entity.
- Each of these pairs is called a match
- In order to find matches, we can use a similarity measure: *a value s(x,y) in [0,1]: s(x,y)=1 iff x=y,* and the closer s(x,y) is to 0, the lower is the similarity.
- We can decide that x and y match if s(x,y) >= t , with 0 <= t <= 1

  Types of similarity measures:
  - ✓ Sequence-based: edit distance, Needleman-Wunch, affine gap, Smith-Waterman, Jaro, Jaro-Winkler
  - ✓ Set-based: overlap, Jaccard, TF/IDF
  - ✓ Hybrid: generalized Jaccard, soft TF/IDF, Monge-Elkan
  - ✓ Phonetic: Soundex
- Distance measures: they are the opposite, the lower the similarity, the higher the distance

39

Edit (or Levenshtein) Distance

- The edit distance is based on the minimal number of operations that are needed to transform string *a* into string *b*:

  character insertion, character deletion, character replacement

- Example:

  a = Politecnico di Milano, *b* = Politecnico Milano

  d(*a*,*b*) = 2, using the following sequence of ops:

  1.Delete the character d of *a*

  2.Delete the character i of *a*

- Similarity: s(*a*,*b*) = 1 - d(*a*,*b*) / [max(length(*a*), length(*b*))]
- Example: s(Politecnico di Milano, Politecnico Milano) = 1 – 2 / max(19, 17) = 1 - 0,105... = 0,894...

40

40

Jaro - Winkler

- The **Jaro-Winkler** string comparator counts
  - the **number c of common characters** between two strings (limited to the greatest integer of half the length of the longer string)
  - the number of transpositions that are the number of pairs of common characters that are out of order
    - S= 1/3*(c/m + c/n + (c-t)/c)
- Example:
  - Compare two surnames “Barnes” and “Anderson”. Common characters are “arnes”, one transposition “rne” → “ner”
  - S=1/3*(5/6+5/8+(5-1)/5)=0.75

41

Set-based Similarity Measures

- View strings as *multisets of tokens*
- E.g. the Jaccard measure: $J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$
- For strings, this corresponds to dividing the strings into tokens, and computing the measure on the two sets of tokens.
- The tokens are pieces of the strings, like for instance, we can choose tokens of lenght 2:

  E.g., for strings *pino* and *pin*

  A = {#p, pi, in, no, o#}, B = {#p, pi, in, n#}

  J(pino, pin) = 3/(5+4)-3 = 3/6 = 1/2

Phonetic Similarity Measures

- Match strings based on their *sound*
- Soundex is the most common one. Soundex calculates a *four-character code* from a word based on the pronunciation and considers two words as similar if their codes are equal.
- Similar sounding letters are assigned the same soundex code.
- Used in archives, e.g., searching ancestors, since often the way the names of the families evolve is according to sound, like for instance Legoff and Legough.
- What is an advantage for the previous use can be a disadvantage for common words, like e.g., “coughing” and “coffin”, that are pronounced almost the same in English.
- Strongly based on the language, since the way words are pronounced is fundamental, e.g., the string “gn” in English and Italian is pronounced in different ways.

American Soundex

- Soundex: it has been defined for problems with names. Names can be spelt in different ways.
- To solve this problem, we need phonetically oriented algorithms which can find similar sounding terms and names.
- Algorithm
  - 1. The first character of the word is retained as the first character of the Soundex code.
  - 2.The following letters are discarded: a,e,i,o,u,h,w, and y.
  - 3.Remaining consonants are given a code number.
  - 4.If consonants having the same code number appear consecutively, the number will only be coded once. (e.g. "B233" becomes "B23")
  - The resulting code is modified so that it becomes exactly four characters long:
    If it is less than 4 characters, zeroes are added to the end (e.g. "B2" becomes "B200")
    If it is more than 4 characters, the code is truncated (e.g. "B2435" becomes "B243")

Soundex: code numbers

b, p, f, and v	1
c, s, k, g, j, q, x, z	2
d, t	3
l	4
m,n	5
r	6

Examples

AC/DC	Ay See Dee Ci
A232	A232

Our	Hour
0600	H600

Robert	Rupert
R161	R161

Bejing	Pecking
B252	P252

Philadelphia	Filadelfia
P434	F434

46

Efficiency

- Applying s(x,y) to all pairs is quadratic in the size of the datasets
- Many solutions have been proposed to choose the pairs of words that most probably match.

47

47

Now record matching

Types of record matching

- Rule-based
- Learning- based (supervised or unsupervised)
- Probabilistic

Rule-based Matching

SSN	NAME	AGE	SALARY	POSITION
123456789	JOHN	34	30K	ENGINEER
234567891	KETTY	27	25K	ENGINEER
345678912	WANG	39	32K	MANAGER

Many types of rules exist.
E.g.
sim(x,y) =
$0.5s_{SSN}(x,y) + 0.2s_{name}(x,y) + 0.1s_{age}(x,y) + 0.1s_{salary}(x,y) + 0.1s_{position}(x,y)$

Note: possibly using different similarity measures for different attributes !

Manually written rules that specify when two tuples match. E.g. two tuples refer to the same person if they have the same SSN

SSN	NAME	AGE	SALARY	PHONE
234567891	KETTY	25	20K	1234567
345678912	WANG	38	22K	2345678
456789123	MARY	42	34K	3456789

- Rules are manually specified, it requires a lot of time and effort
- A variant is to learn rules

Learning Matching Rules

- It can be supervised or unsupervised
- Supervised (e.g. classification): learn how to match from training data, then apply it to match new tuple pairs:
  - Learn how to match each attribute of the tuples (the training phase): each (x_i,y_i) is a pair of elements and l_i is a label: "yes" if x_i matches y_i and "no" otherwise
  - Define the weight of each attribute in the final matching of the whole records
  - Apply the learned model to the new tuple pairs
- Supervised learning requires a lot of training data
- Unsupervised learning (typically clustering, based on clusterizing similar values) may solve this problem

Probabilistic Matching

- Model the matching domain using a probability distribution
- Reason with the distribution to make matching decisions
- Key benefits
  - provide a principled framework that can naturally incorporate a variety of domain knowledge
  - can leverage the wealth of probabilistic representation and reasoning techniques already developed in the AI and DB communities
  - provide a frame of reference for comparing and explaining other matching approaches
- Disadvantages
  - computationally expensive
  - often hard to understand and debug matching decisions

Data Fusion

SSN	NAME	AGE	SALARY	POSITION
123456789	JOHN	34	30K	ENGINEER
234567891	KETTY	27	25K	ENGINEER
345678912	WANG	39	32K	MANAGER

SSN	NAME	AGE	SALARY	PHONE
234567891	KETTY	25	20K	1234567
345678912	WANG	38	22K	2345678
456789123	MARY	42	34K	3456789

Once you have understood that some data clearly represent the same entity (same person, in this case), you still have to cope with the problem of what to do when other parts of the info do not match

Resolution function

Inconsistency may depend on different reasons:

- One (or both) of the sources are incorrect
- Each source has a correct but partial view, e.g. databases from different workplaces, e.g.:
  - the full salary is the sum of the two
  - the list of authors of a book is incomplete
  - one of two full names only contains the first letter of the middle name
- Often, the correct value may be obtained as a function of the original ones, e.g. : $value_1 + value_2$, or $1*value_1 + 0*value_2$ or $0{,}5*value_1 + 0{,}5*value_2$)

RESOLUTION FUNCTION: EXAMPLE

SSN	NAME	AGE	SALARY	POSITION
123456789	JOHN	34	30K	ENGINEER
234567891	KETTY	27	25K	ENGINEER
345678912	WANG	39	32K	MANAGER

SSN	NAME	AGE	SALARY	PHONE
234567891	KETTY	25	20K	1234567
345678912	WANG	38	22K	2345678
456789123	MARY	42	34K	3456789

SSN	NAME	AGE	SALARY	POSITION	PHONE
123456789	JOHN	34	30K	ENGINEER	NULL
234567891	KETTY	27	45K	ENGINEER	1234567
345678912	WANG	39	54K	MANAGER	2345678
456789123	MARY	42	34K	NULL	3456789

R=*MAX_AGE*, *SUM_SALARY* (R1 OuterJoin R2)

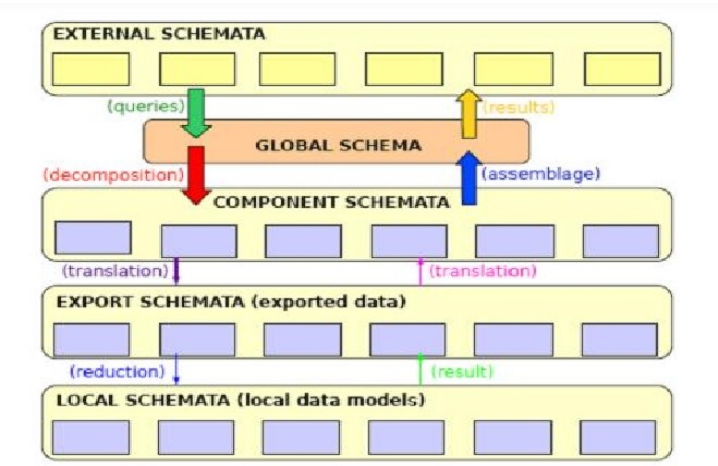
We can now start reasoning on the new application context (recall)

- A (possibly large) number of data sources
- ➢ Heterogeneous data sources
- Different levels of data structure
  - Databases (relational, OO…)
  - Semi-structured data sources (XML, HTML, more markups …)
  - Unstructured data (text, multimedia etc…)
- Different terminologies and different operational contexts
- Time-variant data (e.g. WEB)
- Mobile, transient data sources

Levels of source heterogeneity

- Schemata (already seen)
- Data Models
- Systems (easily overcome if we reconcile data models and schemata)

Data integration in the MULTIDATABASE



A new element in this figure: WRAPPERS (translators)

- Wrappers basically implement the data layer of the global figure shown in the last lecture: they convert queries into queries/commands which are understandable for the specific data source, possibly extending the query possibilities of a data source (see e.g. type conversions)
- They convert query results from the source format to a format which is understandable for the query/application
- Easy to produce in the case of structured data: e.g. for the relational model and the Object Oriented model, we can translate the queries and the results between the two models
- More difficult is the problem if the data is not structured (later)

Design steps

1. Reverse engineering (i.e. production of the conceptual schema)
2. Conceptual schemata integration
3. Choice of the target logical data model and translation of the global conceptual schema
4. Definition of the language translation (wrapping)
5. Definition of the data views (as usual)

Bibliography

- A. Doan, A. Halevy and Z. Ives, Principles of Data Integration, Morgan Kaufmann, 2012
- L. Dong, D. Srivastava, Big Data Integration, Morgan & Claypool Publishers, 2015
- Roberto De Virgilio, Fausto Giunchiglia, Letizia Tanca (Eds.): Semantic Web Information Management – A Model-Based Perspective. Springer 2009, ISBN 978-3-642-04328-4
- M. Lenzerini, Data Integration: A Theoretical Perspective, Proceedings of ACM PODS, pp. 233-246, ACM, 2002, ISBN: 1-58113-507-6
- Clement T. Yu, Weiyi Meng, Principles of Database Query Processing for Advanced Applications , Morgan Kaufmann, 1998, ISBN: 1558604340