PART II - Integration of Heterogeneous Databases

1. Introduction
   1. Integration of Database Systems
   2. Important types of Integrated Database Systems

2. Wrappers: Transparent Access to Data Sources
   1. Overview of Wrapping Approaches
   2. Example: Informix VTI Interface
   3. Example: Tamino SQLNode
   4. Example: IBM DB2 XML Extender

3. Schema Transformation
   1. Schema Integration Method
   2. Identification of Correspondences
   3. Resolution of Conflicts
   4. Implementation of an Integrated View
   5. Processing of Queries and Updates

4. Summary

5. References
1 Introduction

• 1.1 Integration of Database Systems
• 1.2 Important Types of Integrated Database Systems
1.1 Integration of Databases

- A frequent problem in information systems development is the integration of existing information sources
- Example: Integration of publication data
  - Web resources: e.g. DB Server in Trier, CiteSeer citation index
  - Local bibliography: e.g. MS Access application for lab library
  - University library server: e.g. Oracle database application
  - Private bibliography: e.g. BibTex file, MS Word document

- Goals
  - Maintain consistent data
  - Perform queries over all databases at once
  - Perform updates only once

- Problems
  - Establishing communication
  - Different data models
  - Different data schemas and representation
  - Different processing capabilities
Fundamental Dimensions in Distributed Information Systems

- Distribution
  - Distributed DBMS
  - WWW-Mirrors
  - Federated databases (Multidatabases)
- Heterogeneity
- Autonomy
Heterogeneity

- Autonomous development of information systems leads naturally to different solutions for the same problems
  - A main problem in information systems integration
- Levels of heterogeneity
  - syntactic
  - semantic
- Syntactical heterogeneity
  - Communication interface
    - Access protocols: http, CORBA, SQL, ODBC, ...
    - Stateless- or stateful connections
    - Security and admission procedures
  - Data models
    - Relational, XML, object-oriented, ...
  - Functionality of access methods
    - Different scope and expressibility of query languages
    - Restrictions on query types (e.g. joins, binding of variables)
- Semantic heterogeneity
  - The semantics of the models do not coincide
  - The semantics coincides but the structural representation differs
  - The representation is the same but the extensions differ
Autonomy

- Autonomy requirement
  - The local operation of a system and system consistency is not affected by its participation in a global system
- 1. Design autonomy
  - Each system can model its data and processes using the preferred models and in the preferred way
  - This may lead to **heterogeneity** in data processing models that need to be overcome by intermediate layers
- 2. Execution autonomy
  - Each system can execute operations that are requested in the preferred way
  - This may lead to inconsistent executions of operations and requires means to recognize and compensate (→ later: distributed transactions)
- 3. Communication autonomy
  - Each system is free to decide when and to what provide access to its resources
  - This may lead to incomplete access to resources and thus limit the capabilities of the global system (→ later: distributed objects)
- Semiautonomous systems
  - Give up some of their autonomy to participate in a global system (contract)
1.2 Important Types of Integrated Database Systems

- Multidatabases
- Federated Databases
- Mediator Systems
- Data Warehouses
Multidatabases

- Enable transparent access to multiple databases, loose coupling
  - Hide distribution, different database language variants
  - Process queries and updates against multiple databases
  - Do not hide the different database schemas

Source: IBM
Federated Databases

- **Export Schema**
  - Unifies data model
  - Enriches access functions

- **Import Schema**
  - view on export schema

- **Integrated schema**
  - Homogenizes and unions import schemas
  - Virtual vs. materialized
  - Predefined vs. constructed

- **Views**
  - as in centralized DBMS
Mediator Systems

- Main differences to federated databases
  - Read-only access
  - Non-database resources
  - Light-weight and flexible

Source: Kutsche
Data Warehouses

- Are a specialized form of federated database system, where the integrated database is materialized
  - perform efficient data analysis
  - OLAP (on-line analytical processing)
  - Data mining
- Uses typically limited forms of database schemas (e.g. relational star schemas)
  - Supports multidimensional data model
- Main problems
  - Updates and data consistency
  - Wrapper development
  - Efficient processing of OLAP and data mining queries
Aspects to Consider for Integration

- General issues
  - Bottom-up vs. top-down engineering
    - from existing schema to integrated and vice versa
  - A priori vs. a posteriori integration
  - Virtual vs. materialized integration
  - Read-only vs. read-write access

- Data model related issues
  - Type of sources
    - structured, semi-structured, unstructured
  - Data model of integrated system
    - canonical data model
  - Tight vs. loose integration
    - global schema
  - Support for semantic integration
    - collection, fusion, abstraction, supplementation
  - Transparency
    - language, schema, location
  - Query model
## Properties of Integrated Information Systems (typical)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Multidatabases</th>
<th>Federated databases</th>
<th>Mediator Systems</th>
<th>Data Warehouses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom-up vs. top-down</td>
<td>none</td>
<td>bottom-up</td>
<td>top-down</td>
<td>bottom-up</td>
</tr>
<tr>
<td>Virtual vs. materialized</td>
<td>virtual</td>
<td>virtual</td>
<td>virtual</td>
<td>materialized</td>
</tr>
<tr>
<td>Read-only vs. read-write</td>
<td>read-write</td>
<td>read-write</td>
<td>read-only</td>
<td>read-only</td>
</tr>
<tr>
<td>Type of sources</td>
<td>structured</td>
<td>structured</td>
<td>all</td>
<td>all</td>
</tr>
<tr>
<td>Canonical data model</td>
<td>relational</td>
<td>relational, object-oriented</td>
<td>semi-structured, XML</td>
<td>multi-dimensional</td>
</tr>
<tr>
<td>Global schema</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Support for semantic integration</td>
<td>collection</td>
<td>collection, fusion</td>
<td>collection, fusion, abstraction</td>
<td>collection, fusion, aggregation</td>
</tr>
<tr>
<td>Transparency</td>
<td>(location), language</td>
<td>location, schema, language</td>
<td>location, schema, language</td>
<td>location, schema, language</td>
</tr>
<tr>
<td>Query model</td>
<td>SQL</td>
<td>SQL</td>
<td>Rule-based</td>
<td>OLAP</td>
</tr>
</tbody>
</table>

©2001, Karl Aberer, EPFL-DSC, Laboratoire de systèmes d'informations répartis
2 Wrappers

- 2.1 Overview of Wrapping Approaches
- 2.2 Example: Informix VTI Interface
- 2.3 Example: Tamino SQLNode
- 2.4 Example: Extensible Databases (IBM DB2)
2. 1 Wrappers: Transparent Access to Data Sources

- Wrappers are software to overcome syntactic heterogeneity (communication, data model, functionality)
Mappings between Different Data Models

- Different data models support different data types and type constructors
- Mappings from a "simple" model (fewer, less powerful constructors) to a "complex" model are straightforward
  - e.g. from relational to object-oriented or XML-Schema
- The other direction is always possible, but comes with a loss in
  - constraints: not all constraints that are modelled are maintained
    - i.e. the new model may fit inconsistent data
  - performance: Since the structure of a schema is usually exploited for optimization
- Also the differences in processing capabilities, e.g. constraints, queries, need to be considered
  - e.g. relational is more powerful than XML
Example: Generic Mapping of Relational Data into XML

- The mapping of a relational table instance $R(A_1, \ldots, A_n)$ into an XML structure is straightforward.
- The problem is:
  - Which constraints can be maintained?
  - What query processing capabilities are available?
  - E.g. joins are missing.
- XML Schema + XML Query
  - Should solve many of these problems!
Example: Generic Mapping of XML Data into Relational

- Sample source document
Example: Generic Mapping of XML Data into Relational

- Also this can be done in several straightforward ways, e.g.

<table>
<thead>
<tr>
<th>FlatElemTable</th>
<th>DocElemTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>own_id</td>
<td>doc</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DocTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>own_id</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>attrRecTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>element</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DocumentTypeTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>own_id</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

- Problems: structural constraints are lost
  - need to be dynamically reconstructed (efficiency!)
  - Consistency of data needs to be checked in addition

©2001, Karl Aberer, EPFL-DSC, Laboratoire de systèmes d'informations répartis
Categories of Wrappers

• There exists no standard approach to build wrappers
• Functionality
  – One-way: only transformation of data (e.g. for data warehouses)
  – Two-way: transformation of requests and data
• Development
  – Hard-wired wrappers, for specific data sources
  – Semi-automatic generation: wrapper development tools
  – Automatically generated wrappers
• Availability
  – Standalone programs for data conversion
  – Modules within a federated DBMS
  – Exploiting integral functionality of DBMS (extensible DBMS)
Data Conversion Tools

- In practice data conversion tools are specific software that
  - support many different formats
  - perform standard conversions semi-automatically
  - allow definition of complex conversion logic
  - maintain a repository of conversion methods

- Examples
  - Microsoft data import tools
  - XMLJunction
  - XSLT is also considered as a possible solution to convert XML data, e.g. conversion of XML databases into relational
Relational Database Middleware

- A number of products (IBM DataJoiner, Miracle, EDS) allow to access multiple relational databases (from different vendors)
  - Thus a multidatabase system
  - Tables in foreign databases can be accessed like local tables
  - 2 phase commit for transactions (updates)
  - Optimization of queries, pushing down of subqueries
- Address fully syntactic heterogeneity for relational DBs
  - Location transparency
  - SQL dialect transparency
  - Error code transparency
  - Data type transparency
Database Interfaces for Foreign Data

• Some database products allow to define user-defined mappings to data from arbitrary sources
  – Data appears like local data
  – Mapping includes (some) compensation for missing functions at the source

• Examples
  – Informix' Virtual Table Interface (VTI)
    • Any data to relational data
  – Tamino's SQLNode
    • relational data to XML data
  – IBM DB2 XML Extender
    • XML data to relational data
2.2 Informix Virtual Table Interface

- When a relational query is processed it is transformed into a sequence of calls to the storage system, for example
  - `SELECT * FROM publications`

  translates to

```plaintext
TABLEDESCRIPTION* Table;
SCANDESCRIPTION* Scan;
ROW* Record;
Table := OpenTable(Publications);
Scan := BeginScan(Table);
while (( record := NextScan(Table)) != END-OF-SCAN) {process record};
EndScan(Scan);
CloseTable(Table);
```

- If external data should appear like relational it has to support VTI functions (see table)
  - By providing and registering those functions any data can be integrated
  - Already a partial implementation can be sufficient (with less performance)
VTI functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get_Next</td>
<td>Mandatory</td>
<td>Primary scan function. Returns reference to next record.</td>
</tr>
<tr>
<td>Open_Table</td>
<td>Setup</td>
<td>Called at beginning of query to perform any initialization.</td>
</tr>
<tr>
<td>Begin_Scan</td>
<td>Setup</td>
<td>Called at beginning of each scan if query requires it (as when virtual table is part of nested loop join)</td>
</tr>
<tr>
<td>End_Scan</td>
<td>Teardown</td>
<td>Called to perform end-of-scan cleanup.</td>
</tr>
<tr>
<td>Close_Table</td>
<td>Teardown</td>
<td>Called to perform end-of-query cleanup.</td>
</tr>
<tr>
<td>Rescan</td>
<td>Teardown/setup</td>
<td>If this function is defined, query processor will call it instead of an End_Scan/Begin_Scan sequence.</td>
</tr>
<tr>
<td>Insert</td>
<td>Table modification</td>
<td>Called by SQL insert.</td>
</tr>
<tr>
<td>Update</td>
<td>Table modification</td>
<td>Called by SQL update.</td>
</tr>
<tr>
<td>Delete</td>
<td>Table modification</td>
<td>Called by SQL delete.</td>
</tr>
<tr>
<td>Scan_Cost</td>
<td>Optimization</td>
<td>Provides optimizer with information about query expense.</td>
</tr>
<tr>
<td>Drop_Table</td>
<td>Table modification</td>
<td>Called to drop virtual table.</td>
</tr>
<tr>
<td>Stats</td>
<td>Statistics maintenance</td>
<td>Called to build statistics about virtual table for optimizer.</td>
</tr>
<tr>
<td>Check</td>
<td>Table verification</td>
<td>Called to perform any of several validity checks.</td>
</tr>
<tr>
<td>Connect</td>
<td>Subquery propagation</td>
<td>Establish association with an external, SQL-aware data source.</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Subquery propagation</td>
<td>Terminate association with external data source.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Subquery propagation</td>
<td>Notify external data source of relevant subquery or other SQL.</td>
</tr>
<tr>
<td>Free</td>
<td>Subquery propagation</td>
<td>Called to deallocate resource associated with the external query.</td>
</tr>
<tr>
<td>Execute</td>
<td>Subquery propagation</td>
<td>Request external data source to execute prepared SQL.</td>
</tr>
<tr>
<td>Transact</td>
<td>Subquery propagation</td>
<td>Called at transaction demarcation points to alert the external data source of transaction begin, commit or abort request.</td>
</tr>
</tbody>
</table>

Source: DE Bulletin Vol 21
2.3 Tamino

Source: SAG

©2001, Karl Aberer, EPFL-DSC, Laboratoire de systèmes d'informations répartis
Tamino Schema Language

- Proprietary, XML-based Schema Language
  - Hierarchical XML structure
  - Processing-related attributes
  - No recursive definition possible
  - Integration of external data sources

```xml
<?xml version="1.0"?> <!DOCTYPE Collection SYSTEM "inorep.dtd">
<ino:collection id="ID001">
  <ino:doctype id="book" key="book1" options="READ INSERT UPDATE DELETE"/>
  <ino:node id="book2" key="book2" obj-type="SEQ">
    ino:parent="book1"
    ino:search-type="no" ino:map-type="Native"/>
  <ino:node id="book3" key="book3" obj-type="CDATA">
    ino:parent="book2"
    ino:search-type="text" ino:map-type="No"/>
  <ino:node id="book4" key="book4" obj-type="SEQ">
    ino:parent="book2"
    ino:search-type="no" ino:map-type="No"/>
  <ino:node id="book5" key="book5" obj-type="CDATA">
    ino:parent="book4"
    ino:search-type="text" ino:map-type="No"/>
  <ino:node id="book6" key="book6" obj-type="SEQ">
    ino:parent="book4"
    ino:search-type="no" ino:map-type="No"/>
</ino:collection>
</?xml version="1.0"?>
```

Source: SAG
Integrating Relational Data Into Tamino

- Tamino allows to populate an XML document from a relational databases
  - Definition of a DTD
  - Transformation to Tamino's schema language
  - Setting of attribute values that specify of how the data is accessed in the RDBMS
Integrating Relational Data Into Tamino
2.4 Wrapping with Extensible Databases

- Extensible databases (object-oriented, object-relational), allow to define new datatypes within the data model
  - Can be used to directly integrate external data
  - Integration is not transparent, i.e. integrated data does not appear like native data
- Example: integration of XML data into relational database
  - IBM DB2 XML extender
- Storage options
  - As character string (XMLCLOB, XMLVARCHAR)
  - As external file (XMLFILE)
  - As set of relational tables that are mapped to an XML document
- Indexing
  - When the document is stored as a whole, selected elements and attributes can be extracted into tables and indexed
  - Textual indexing using the text extender
- Mappings
  - From/to relational tables are specified using XPath and XSLT
Example: XML DTD

Location path: "/Order/Part/Shipment/ShipDate"
Example: Access using IBM DB2 Extender

- **Relational schema**

  ```
  TABLE sales_tab
  invoice_nr CHAR(6) NOT NULL PRIMARY KEY
  sales_person VARCHAR(20)
  order XMLVARCHAR
  ```

- **Indexing element** `extendedprice` **using table** `part_side_table`

- **Querying**

  ```
  SELECT sales_person FROM sales_tab
  WHERE invoice_nr in
  (SELECT invoice_nr FROM part_sidetable
   WHERE extendedprice>2500)
  ```

- **Querying a "hidden" XML element using XPath (expensive)**

  ```
  SELECT sales_person FROM sales_tab
  WHERE extractvarchar(order, '/Order/Customer') like '%IBM%'
  AND invoice_number>100
  ```
3 Schema Transformation

- 3.1 Schema Integration Method
- 3.2 Identification of Correspondences
- 3.3 Resolution of Conflicts
- 3.4 Implementation of an Integrated View
- 3.5 Processing of Queries and Updates
3.1 Schema Transformation Method

- Creation of an integrated view of the data
  - Useful to make heterogeneity transparent for applications
  - Data from the different sources can be complementary or overlapping
  - Problem: the same data can be represented in various ways
  - Requires schema mappings that overcome the differences (schema integration)
  - Requires the capability to process query against the integrated schema

- Integration Method

```
Schema translation (wrapper)
------------------------
Correspondence investigation
-----------------------------
Conflict resolution and schema integration
```
Criteria for the Quality of an Integration Method

- **Completeness**
  - Integrated schema contains all concepts of component schemas
  - no loss of information: data and constraints
- **Minimality**
  - The same real world concept represented in multiple component schemas must be represented only once
- **Correctness**
  - All information in the integrated schema must occur in one of the component schemas
  - New relations among different schema concepts (inter-schema-relations) must not contradict information from component schemas
- **Understandability**
  - Integrated schema must be easily understandable
3.2 Identification of Correspondences

- Find out which two concepts in two schemas correspond to each other
  - no automatic solution possible
  - Manual or semi-automatic approach
- Semi-automatic approaches
  - analyse semantic correspondences (names)
  - analyse structural correspondences (e.g. reachability by paths)
  - analyse data (e.g. distribution of values)
- Sources for missing information
  - database administrator, developer
  - analysis of the source schemas
  - analysis of the source databases
  - analysis of the source applications
Example: Integration of Two Relational Schemata

• SCHEMA 1

PUBLICATIONS(PNR, TITLE, AUTHOR, JOURNAL, PAGES)
AUTHORS(ANR, TITLE, NAME, AFFILIATION)
JOURNAL(JNR, NAME, PUBLISHER, ISSUE, MONTH, YEAR)

• SCHEMA 2

PAPERS(NUMBER, TITLE, WRITER, PUBLISHED)
WRITER-FIRSTNAME, LASTNAME, NROFPUBLICATIONS
PUBLISHED(TYPE, PROCEEDINGS, PP, DATE)
Example: Detecting Correspondences

- **Analysis of relation and attribute names (semantic correspondences)**
  - C1: S1.PUBLICATIONS related to S2.PAPERS (relevant)
  - C2: S1.AUTHORS.TITLE related to S2.PAPERS.TITLE (irrelevant)
  - C3: S1.PUBLICATIONS.TITLE related to S2.PAPERS.TITLE (relevant)

- **Analysis of structural correspondences**
  - Shows that C3 is more relevant than C2
  - S1.PUBLICATIONS.TITLE related to S2.PAPERS.TITLE
    - same path leads to TITLE
  - S1.AUTHORS.TITLE not related to S2.PAPERS.TITLE
    - different path leads to TITLE
  - Shows a further relationship: S1.JOURNALS.YEAR related to S2.PUBLISHED.YR since both share the data type DATE

- **Analysis of data values**
  - S1.PUBLICATIONS.PAGES related to S2.PUBLISHED.PP
  - Both attributes contain always two substrings that are up to 3 digit numbers
3.3 Resolution of Conflicts

- Types of conflicts encountered at schema level
  - Naming conflicts
  - Structural conflicts
  - Classification conflicts
  - Constraint conflicts
  - Behavioral conflicts
  - Data conflicts

- Types of conflicts encountered at data level
  - Naming conflicts
  - Representational conflicts
  - Errors
Naming Conflicts: Homonyms and Synonyms

- **Homonyms**
  - same term used for different concepts,
  - e.g. `S1.AUTHORS.TITLE` vs. `S2.PAPERS.TITLE`
  - Resolution: introduce prefixes to distinguish the names,
  - e.g. `A_TITLE` and `P_TITLE`

- **Synonyms**
  - different terms for the same concepts, e.g.
  - `S1.PUBLICATIONS.AUTHOR` vs. `S2.PAPERS.WRITER`
  - Resolution: introduce a mapping to a common attribute

- Use of Ontologies and Thesauri to detect and resolve naming conflicts
Structural Conflicts

- Same concept represented by different data structures
  - More probable the "richer" the data model is (e.g. object-oriented vs. relational)
  - Kind of conflicts depends on data model
- Example 1: Different attributes
  - \texttt{S1.AUTHORS.AFFILIATION} and \texttt{S2.WRITER.NROFPUBLICATIONS} have no corresponding attributes in the other schema
  - Resolution: create a relation with the union of the attributes
- Example 2: Different datatypes
  - \texttt{S1.JOURNAL.MONTH} and \texttt{S1.JOURNAL.YEAR} contain together the same information as \texttt{S2.PUBLISHED.DATE}
  - Resolution: build a mapping function, e.g. \texttt{DATE(MONTH, YEAR)}
- Example 3: Different data model constructs
  - attribute vs. relation: \texttt{S2.PUBLISHED.TYPE="Journal"} means that the publication is a journal, while \texttt{S1.JOURNALS} contains only journals
  - Resolution: requires higher order mapping languages that can manipulate database schemas directly
Classification Conflicts

- Extensions of concept are different
  - Modelling constructs A and B that are recognized as corresponding can cover sets with different scopes
  - All possible set relationships
    - $A=B$: equivalent sets
    - $A \subseteq B$: A is a subset of B
    - $A \cap B$: A and B have some elements in common
    - $A \neq B$: A and B have no elements in common

- Example
  - $\text{S1.AUTHORS}$ are authors of journal papers only
  - $\text{S2.WRITERS}$ contains also authors of other types of publications

- Resolution
  - build generalization hierarchies
  - requires data model with inheritance, e.g. object-oriented

- Additional problem
  - Identification of corresponding data instances
  - "real world" correspondence is application dependent
  - Example: check the reliability of the bibliographic references in the two databases, and in case of error to trace back to the source of error
Constraint and Behavioral Conflicts

- Depend on the expressivity of the data model
  - Relational model: key constraints, update and delete behavior
  - Object-oriented model: cardinality constraints, methods
- Example 1: Primary key conflicts
  - S1.PUBLICATIONS.PNR is another numbering scheme than S2.PAPERS.NUMBER
  - Resolution: introduce a mapping table
- Example 2: Cardinality conflicts
  - different types of cardinality constraints on relationships
  - Resolution: use the more general constraint
- Example 3: Behavioral conflicts for relation update
  - E.g. cascading delete vs. non-cascading
  - Resolution: add missing behavior at global level
- Example 4: Method conflicts
  - Different methods or user-defined functions supported
  - Resolution: implement methods at global level
Data Conflicts and Resolution Strategies

- Even if all conflicts at the schematic level are resolved, the question remains which data values correspond to each other
  - E.g. two authors are the same if they share the same name?
  - Problems: similar to naming conflicts at schema level
    - Same author may have different names in the databases, e.g. "Aberer, K.H." in DB1 and "Karl Aberer" in DB2
    - Different authors may share the same names, e.g. "Xu, Y." in DB1 (uniquely identified by ANR) may be "Yi Xu", "Yang Xu", "Ying Xu", ... in DB2
  - Applies to all atomic data types
    - E.g. representation of dates: e.g. "2000" vs. "00"

- Resolution
  - Provide an extensional mapping table for all data values
    - Possible for restricted data domains, e.g. "Vaud" -> "VD", "Zürich" -> "ZH", ...
  - Provide a mapping function
    - If computable, e.g. four digit year to two digit year
  - Use a similarity function
    - Compensate for typing errors or variations, e.g. "Chemin d. Monteiron" vs. "Chemin du Monteron"
  - Prefer data from one data source
    - The one that is more trusted

- Usually domain specific knowledge is required!
3.4 Implementation of an Integrated View

- **Use of a DML (e.g. SQL)**
  - approach used in multidatabases, like Data Joiner
  - problem: only sufficient to create the integrated view for simple cases
- **Procedural programming language**
  - with access to the DDL
  - problem: general, but no support for developing consistent transformations
- **Extensible databases (e.g. object-relational or OODBs)**
  - provide more complex DML constructs, DDL instructions and can in addition integrate functions to compute any mapping
  - object-oriented models (including object-relational) are the preferred canonical models for database integration
  - problems: complexity, consistency, design methodology, ...
- **Declarative languages (e.g. predicate logic based, rule based)**
  - most promising approach
  - problem: are currently available in research prototypes only
Example: Implementation of an Integrated View in SQL

- Integrated view on AUTHORS and WRITERS
  - Assume identical instances in AUTHORS and WRITERS are identified through a mapping table
    mapping(firstname, lastname, anr)
  - Implementation in MS ACCESS SQL (barely !) doable

```sql
SELECT [s2_writer.firstname] & " " & [s2_writer.lastname] AS name,
       Null AS title, Null AS affiliation, Null AS anr,
       s2_writer.nrofpublications
FROM s2_writer, mapping WHERE
  [s2_writer.firstname] & " " & [s2_writer.lastname] NOT IN
    (SELECT [firstname] & " " & [lastname] FROM mapping)
UNION
SELECT  s1_authors.name, s1_authors.title, s1_authors.affiliation,
        s1_authors.anr, Null AS nrofpublications
FROM s1_authors LEFT JOIN mapping ON s1_authors.anr = mapping.anr;
```

©2001, Karl Aberer, EPFL-DSC, Laboratoire de systèmes d'informations répartis
Example: Explanations

• Comments
  1. Resolution of the use of different data structures by combining textual fields
  2. Resolution of the use of different attributes by including all attributes into the integrated view
  3. Exclusion of the instances with occurrence in both tables
  4. Union of the both tables, resolution of different coverage
  5. Identification of corresponding instances
3.5 Processing of Queries

- Issues
  - Automatic resolution of schema mapping
  - Optimization of access
Propagation of Updates

- If the mappings are not injective updates cannot be propagated
  - For the same reason SQL restricts view updates
  - No problem in multidatabases, as each external table is integrated separately
- Example
  - The resulting table of the previous example has the structure
    \[\text{integrated}\_\text{authors}(\text{name}, \text{title}, \text{affiliation}, \text{anr}, \text{nrofpublications})\]
  - The following updates on the integrated view do not have always a well-defined semantics (meaning)
  - Why (apart from the fact that updates are principally not possible in SQL for this view) ?
    - Inserting a tuple
    - Updating the name
    - Updating the title (of the author !)
4. Summary

• Syntactic heterogeneity, communication and transaction problems have many solutions
  – Database Middleware
  – XML technology
  – Distributed objects
  – Transaction monitors
• Semantic heterogeneity remains a difficult problem
• Current developments
  – Use of more sophisticated knowledge-based tools, e.g. ontologies, declarative approaches
  – Logic language based approaches
  – Light-weight mediation on the Web based on XML technology
• Many of the heterogeneity problems illustrated for DBs also hold for application objects (except, e.g., the extensional aspects)
5. References

• Books

• Articles

• Websites
  – ibm.com/software/data/db2/extenders/xmlext
  – www.softwareag.com/germany/products/xmlstarterkit.htm