Part VIII

Data Integration

Data Modeling Conflicts

- Conflicts of the Modeling Paradigm: OO,relational,XML,...
- Schema Conflicts
    - Extensional Conflicts
    - Structural Conflicts
    - Description Conflicts
- Data Conflicts

Schema Integration

- combining local schemata to a federated schema in order to overcome heterogeneity
- Requirements of the Integration:
  - completeness: all elements of the local schemata are mapped
  - correctness: every element of a federated schema has a corresponding element in at least one local schema
  - minimality: no real world concept is represented several times in a federated schema
  - comprehensibility: by preserving the notions of the local schemata

Ways to Reach Schema Integration

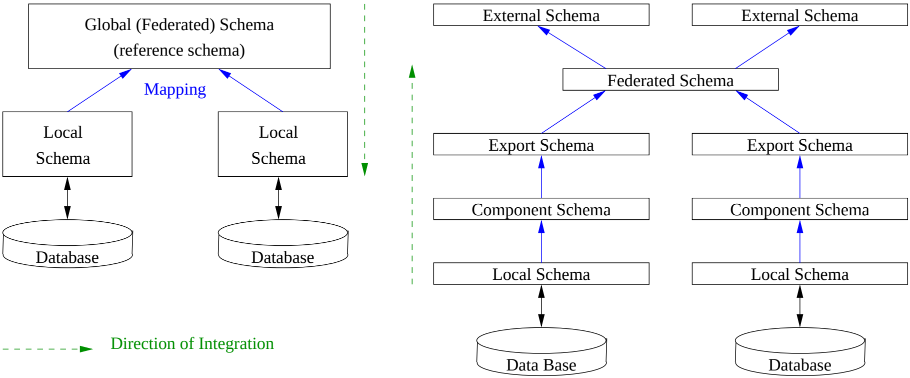
a) top down:

- Steps:
  1) specify a federated schema
  2) map the local schemata to it
- (in general) incomplete and incorrect

b) bottom up:

- combine the local schemata (possibly successively)

Top Down vs. Bottom Up Schema Integration



Extensional Schema Conflicts

- possible relationships between object sets A and B and their corresponding classes K_A and K_B:
  - equivalent sets, $A = B$: no conflict
  - subset relationsship, $A \subseteq B$:
    - extensional conflict, e.g. Employee, Manager
    - solution: K_A inherits from K_B
  - overlapping sets, $A \cap B \neq \emptyset$ and $A - B \neq \emptyset$ and $B - A \neq \emptyset$:
    - extensional conflict, e.g. Employee, Client
    - solution: K_A and K_B inherit from new class $K_{A \cup B}$
  - disjoint sets: $A \cap B = \emptyset$ (but same type!)
    - extensional conflict, e.g. EmployeeMS, EmployeeDO
    - solution: K_A and K_B inherit from new class $K_{A \cup B}$

Structural Schema Conflicts

- certain real world aspects can be modeled in different ways
- e.g. by inheritance or delegation
- this introduces structural conflicts
- solution: choose one approach and map the other to it

Description Conflicts

- different attributes, e.g. point (x, y) or (ω, r) ($\rightarrow$ mapping)
- homonyms and synonyms ($\rightarrow$ dictionary, ontology)
- different data types for semantically same attributes ($\rightarrow$ mapping)
- representation conflicts
  - e.g. sex represented by m/f or 0/1
  - e.g. Y2K Problem: year in YYYY or YY representation (e.g. 1979 or 79)
  - solution: by transformation function or table
- scaling conflicts ($\rightarrow$ transformation function, e.g. $l_m = 100 * l_{cm}$)
- precision conflicts ($\rightarrow$ round?)
- conflicts of integrity constraints
- conflicts of manipulation operations:

  local systems do not provide the same manipulation operations

Example: Description Conflicts

Homonyms:	(business) process	(legal) process
Synonyms:	employee	staff
Data Types:	int	String
Scaling:	1.75 m	175 cm
Precision:	0.5276 kg	0.53 kg
Integrity Constraints:	salary $<$ 8000	salary $<$ 9000

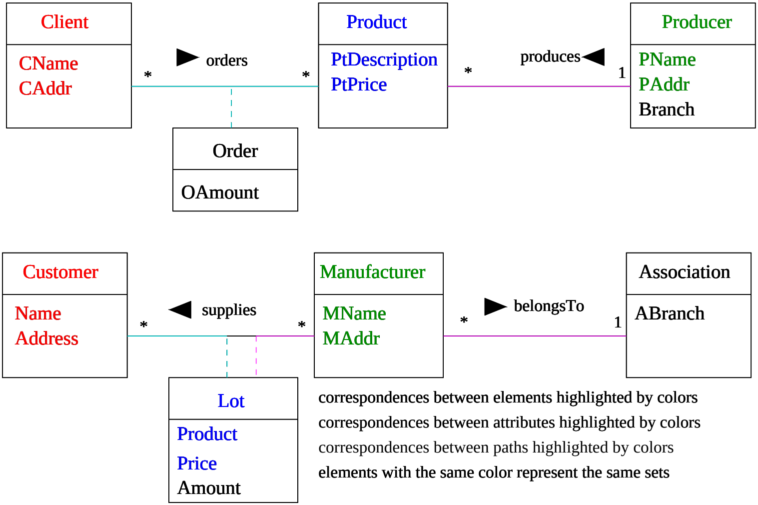
Data Conflicts

- incorrect entries:
  - e.g. due to typos or programming errors
  - solution: by similarity measure
    (problematic; possibly interactively handled)
- obsolete entries:
  - e.g. due to different update interval or missing update
  - solution: trust a) best updated system (if ∃) or b) local system
- different spellings:
  - e.g. Weseler Strasse, Weselerstr., Weseler Straße, . . .
  - solution: by similarity measure (see above),
    by background knowledge (e.g. spelling of names)

Principles of Schema Integration

1) Determine Correspondences between Schemata
   - Element Correspondences:
     - semantic relationsships between schema elements
     - e.g. classes, relations,. . .
   - Attribute Correspondences:
     - semantic relationships between attributes of the schema elements
   - Path Correspondences:
     - relate simple and compound connections between elements of the considered schemata
2) Fix Integration Rules

Example: Correspondences during Schema Integration

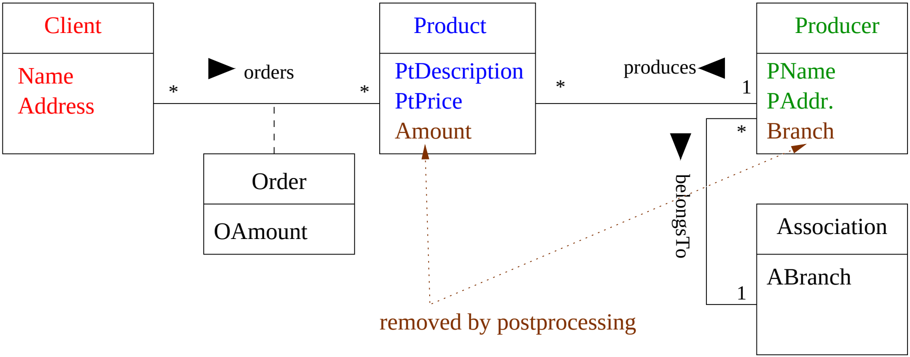


correspondences between elements highlighted by colors
correspondences between attributes highlighted by colors
correspondences between paths highlighted by colors
elements with the same color represent the same sets

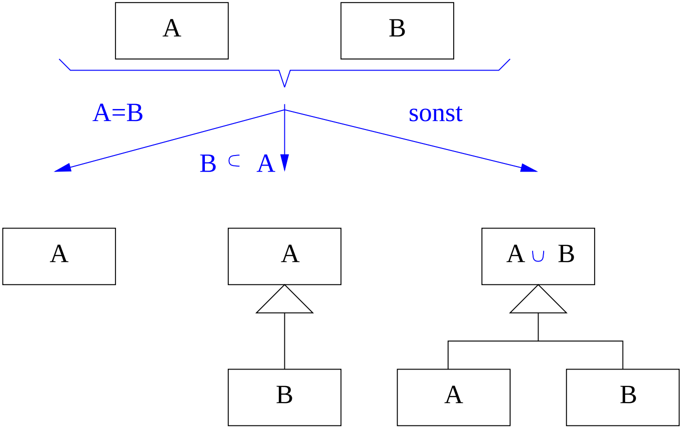
Examples of Integration Rules

1) each schema element without corresponding element in the other schema is propagated to the federated schema
2) equivalent schema elements have a unique representation in the federated schema
   a) their attributes without correspondences are adopted
   b) equivalent attributes are joined
   c) in case of subset correspondences between attributes, the superset attribute is propagated
   d) in case of attributes corresponding to overlapping or disjoint sets, a new attribute representing the union (sum, average, . . .) is used in the federated schema
3) corresponding paths are represented by a semantically equivalent path in the federated schema ($\rightarrow$ possibly integrity constraints)

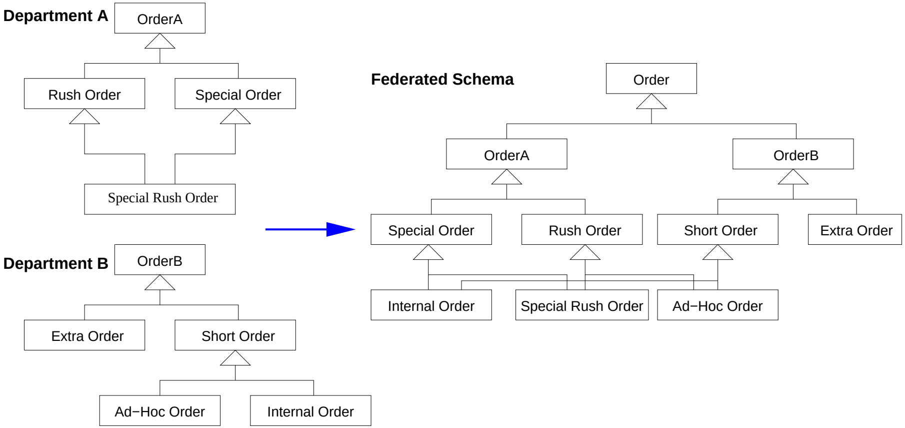
Example: Result of the Schema Integration



Integration Rules for Corresponding Schema Elements



Example: Integration of Corresponding Schema Elements



- ad-hoc orders are specific rush orders
- internal orders are always special orders

Data Cleansing

- high quality of data is essential for successful data integration
- goal: remove data conflicts, description conflicts, duplicates, inconsistencies, ...
- idea: apply data clean(s)ing (known from data warehouses)
- tools: e.g. WizRule
- single-source problem: if single data source
- multiple-source problem: if several data sources

Data Cleaning Process

- repeat the following steps successively

1) data analysis: detect problems (in general: by hand)
2) definition of transformations: for solution of problems
3) “verification”:
    - check (by random sampling), whether transformations work properly
4) apply transformation
5) store and use cleaned data

Integration of Instances

- i.e. detect and remove duplicates
- in particular for several data sources
- approach depends on extensional correspondence (equivalent: 1:1, disjoint: 1:0)
- use common keys (if available; not necessarily primary keys)
- otherwise no matching is possible

Example: Integration of Instances

Employee

Id	Name	Salary
ThMi201169K17	Miller	2734
ThMi170684R42	Miller	2734
...	...	...

Personal

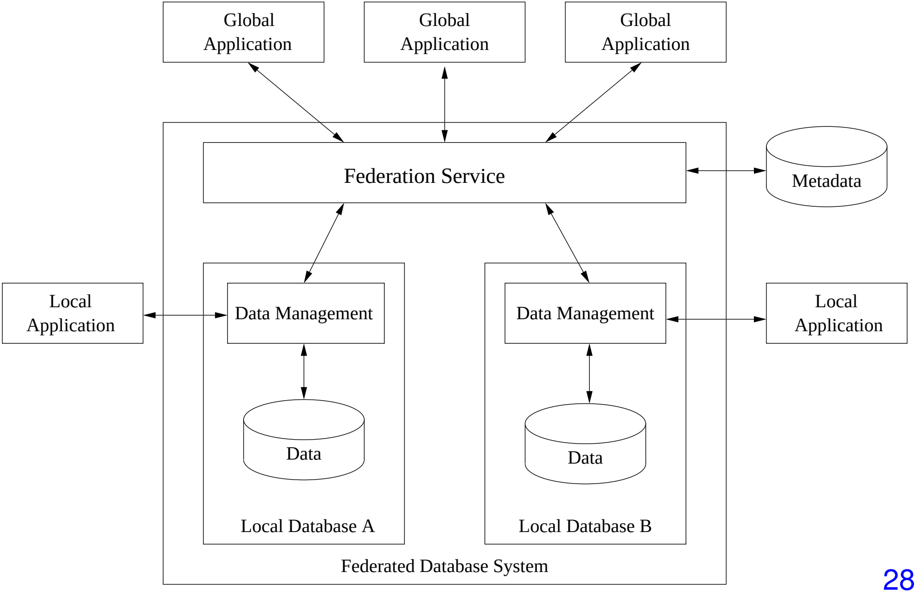
Name	Born	Salary
Miller	20.11.1969	2734
Miller	17.06.1984	2734
...	...	...

- no common keys
- but: `Id` contains `Born`; hence key can be inferred

Integrated Table

Id	Name	Salary	Born
ThMi201169K17	Miller	2734	20.11.1969
ThMi170684R42	Miller	2734	17.06.1984
...	...	...	...

Architecture of Federated Information Systems

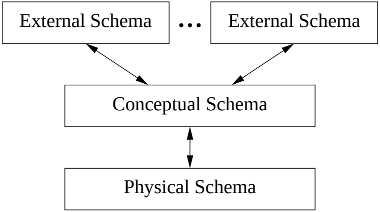


Federation Core

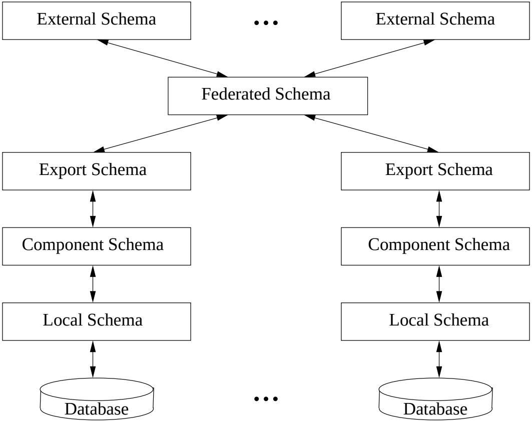
- takes care of (some) typical database functions:
  - transaction handling
  - query processing and optimization
  - integrity control
  - recovery
- in practice, these functions are only partly offered due to the enormous complexity and costs
- only the essential functions are provided

Schema Architecture

conventional DBMS: 3 Levels



federated DBMS: 5 Levels



5 Level Schema Architecture of a Federated DBMS

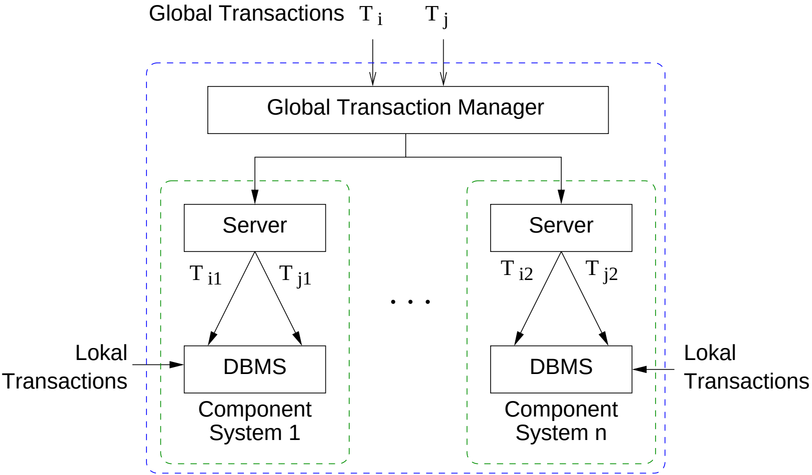
- Local Schema: corresponds to conceptual schema of a conventional DBMS
- Component Schema: translates the local schemata to a common language
- Export Schema: externally relevant part of the component schema
- Federated Schema: represents conceptual schema of the overall system
- External Schema: corresponds to a view of the federated system
- lower 3 levels correspond to physical schema of the overall system
- the restriction to the externally relevant aspects is often performed before translating to the common language

Transactions in Federated Systems

- if only reads in federated system and no absolute consistency required: abstain from global transaction processing
- otherwise aim for: ACID properties
  - Atomicity: transaction executed “all or nothing”
  - Consistency: transaction preserves consistency of data
  - Isolation: user has the impression to work alone on DB
  - Durability: successful transactions cause persistent changes
- typical implementation: 2-phase commit protocol
  - at the end of an overall transaction, all participating transactions are asked, whether they are ready for commit
  - if so: perform commit everywhere
  - else: rollback everywhere
- 2-phase commit is supported by typical DBMS

Problems with Transactions in Federated Systems

- besides global transactions, there are also local ones
- this complicates the detection of deadlocks



Example: Deadlock in Federated System

- component system 1 stores a and b
- component systen 2 stores c and d
- transactions:
  - global: GT_1: $r_1(a), r_1(d)$, GT_2: $r_2(c), r_2(b)$
  - local: LT_3: $w_3(b), w_3(a)$, LT_4: $w_4(d), w_4(c)$
- possible order: $r_1(a), w_3(b), r_2(c), w_4(d)$ deadlock
- nobody can detect the deadlock:
  - in system 1: GT_2 waits for LT_3; LT_3 waits for GT_1; GT_1 is not waiting here
  - in system 2: GT_1 waits for LT_4; LT_4 waits for GT_2; GT_2 is not waiting here
  - the global transaction manager does not know about the local transactions
- solution? (possibly rollback after Timeout)

Java Database Connectivity (JDBC)

- API for using SQL statements in Java (→ ODBC)
- results of queries are processed tuple by tuple
- in class java.sql.Statement: ResultSet executeQuery(String sql)
- also updates possible (INSERT, UPDATE, DELETE)
- in class java.sql.Statement: int executeUpdate(String sql)
- Java compiler cannot check SQL Statements (Strings!)
  (→ SQL syntax errors cause runtime errors)

Example: JDBC

```
import java.net.URL; import java.sql.*;

class JDBCbsp{
  public static void main(String argv []){
    try { // generate URL of ODBC data source
      String url  = "jdbc:odbc:mySource";
      // connect to data source
      Connection con = DriverManager.getConnection(url,"user","passwd");
      //  generate SELECT statement and execute it
      Statement stmt = con.createStatement();
      ResultSet rs  = stmt.executeQuery("SELECT no, name FROM Staff");
      while (rs.next()){ // process result collection successively
        // extract and process attribute values of current tuple
        int no = rs.getInt(1);   String name = rs.getString(2);
        System.out.println("Number: "+no+", Name: " + name);}
      stmt.close();  con.close();}
    catch (java.lang.Exception e){e.printStackTrace();}
  }
}
```

Multi-Database Languages

- Status: in development
- Motivation:
  - a SQL statement can only access a single DB
  - auxiliary solution: combination by program logic
  - convincing solution: multi-database language,
    e.g. SchemaSQL, SQL/MED (in development)
- Properties of Multi-Database Languages
  - local schemata are visible
  - each user defines own federated schema and external schemata
- Product: (e.g.) IBM DB2 Information Integrator
- for details: see literature