Data Integration

Problem:
• *Global Data Model*: given existing databases with their Local Conceptual Schemas (LCSs), how to integrate the LCSs into a Global Conceptual Schema (GCS) - GCS is also called *mediated schema* 
• *Global Query Processing & Optimization*
• *Global Transaction Management* ?
• *Bottom-up design process*

Vera Goebel
Department of Informatics, University of Oslo

Fall 2013
Integration Alternatives

• Physical integration
  – Source databases integrated and the integrated database is materialized
  – Data Warehouses

• Logical integration
  – Global conceptual schema is virtual and not materialized
  – Enterprise Information Integration (EII)
Data Warehouse Approach (OLAP)
Bottom-up Design

• GCS (also called mediated schema) is defined first
  – Map LCSs to this schema
  – As in data warehouses

• GCS is defined as an integration of parts of LCSs
  – Generate GCS and map LCSs to this GCS
GCS/LCS Relationship

- **Local-as-view**
  - The GCS definition is assumed to exist, and each LCS is treated as a view definition over it

- **Global-as-view**
  - The GCS is defined as a set of views over the LCSs
Database Integration Process
Recall Access Architecture
Heterogeneous / Federated / Multi-Database Systems

• Why Heterogeneous Database Systems (HDBS)?
  – Applications, …
• Architectures for HDBS
• Main Problems:
  - Global Data Model
  - Query Processing & Optimization
  - Transaction Management
Applications

- Multitude of extensive, isolated data agglomerations managed by different DBMSs or file systems
  - Similar data
    • Ex: 3 Customer Info Systems
  - Dissimilar data
    • Ex: Extended CAD Application

- Extension of data and management software because of new and/or extended applications

- Heterogeneous application domains (e.g., CIM, CAD, Biz-mgmt, …)
Heterogeneous Database Systems (HDBS)
Requirements for HDBS

• Properties known from homogeneous DBS:
  - global data model, transactions, recovery, dist transparency, ...

• Integration of Heterogeneous Data Stores
  -> queries across HDBs (combine heterogeneous data)
  -> heterogeneous information structures
  -> avoid redundancy
  -> access (query) language transparency

• “Open” system
  support for integration of existing data models and DBSs, as well as their schemas and DBs

• Constraints
  -> retain autonomy of DBS to be integrated
  -> avoid modifications of existing local applications
  -> define a viable global data model for global applications
Definition - Heterogeneous DBS (HDBS)

A HDBS comprises a software layer (integration layer) and multiple DBSs and/or file systems to be integrated.

Users can transparently access the integrated DBSs and/or file systems via the interface provided by the integration layer.

- Defines a global data model
- Supports a Data Definition Language (DDL)
- Supports a Data Manipulation Language (DML)
- Distributed Transaction Management
- Transparent integration of the underlying, disparate DBSs

The integrated, local DBSs are autonomous and can also be used as stand-alone systems.

Local applications are unchanged and unknown to the HDBS.
Abstract Component Architecture of HDBS

Coupling software can be partitioned into processes (or agents) that execute on HDBMS hosts and on local DB hosts.
Heterogeneous Database Systems (fully auton. HDBS)

HDBS Server or HDBS Proxy
- Runs on the local DB site
- Typically includes some code that is specific to the local DB type
Information Integration Architecture

“Multiple, legacy data sources”
Concepts in the Integration Layer

- Global data model
- Global schema and meta data management
- Distributed query processing and optimization
- Distributed transaction management
- Extensible software construction
  (to allow the “easy” integration of additional system components)
Data Model

- **Local data models**: any kind of data model possible, e.g., object-oriented, relational, entity-relationship, hierarchical, network-oriented, flat files, ...  

- **Global data model**: must comprise modeling concepts and mechanisms to express the features of the local data models  
  - When integrating N local data models, use the “richest” model of the N models you are integrating  
  - Object-oriented data models  
    - Provide user-defined data types and methods  
    - Are often used as the global (integration) data model  

Goals - To define a data model that:  
1) Is a complete, minimal, and understandable data model for the union of the data stored in the set of local data bases (*application development time*)  
2) Support application queries that can be satisfied by retrieving data from the set of local data bases (*application runtime*)
Database Integration Issues

• Schema translation
  – Component database schemas translated to a common intermediate canonical representation

• Schema generation
  – Intermediate schemas are used to create a global conceptual schema
Schema Translation

• What is the canonical data model?
  – Relational
  – Entity-relationship
  – Object-oriented
  – Graph-oriented
    • Preferable with emergence of XML
    • No common graph formalism

• Mapping algorithms
  – These are well-known
Schema Generation

• Schema matching
  – Finding the correspondences between multiple schemas

• Schema integration
  – Creation of the GCS (or mediated schema) using the correspondences

• Schema mapping
  – How to map data from local databases to the GCS

• Important: sometimes the GCS is defined first and schema matching and schema mapping is done against this target GCS
Schema Architecture of HDBS

- Global trick: federated schema
- Domain-specific NHL schemas
- Meta-model for domain-specific NHL schemas

- Homogenization
- Schema integration
- Global data model
- Local data models
Schema Architecture of HDBS - 2

5-layer schema architecture

- external schema
- local schema
- auxiliary schema
- export schema
- global data model
- local data models
- federated schema
- component schema
- component schema
- local schema
- local schema
- external schema
- auxiliary schema

Multi-lingual
Multi-Use
Multiple Views

App View Defn
Integration
Global View Defn
Translation
Schema Homogenization

• Schema Translation
  – Map each local schema to the language of the global data model
    • Ex: a Relational schema to an Object-oriented schema

• Schema Integration
  – For $N$ translated, local schemas
    • Pairwise integration, X-at-a-time integration, One-step integration
  – Determine ”common semantics” of the schemas
  – Make the ”same things” be ”one thing” in the integrated schema
  – Resolve conflicts
    • structural and semantic
Schema Matching

- Schema heterogeneity
  - Structural heterogeneity
    - Type conflicts
    - Dependency conflicts
    - Key conflicts
    - Behavioral conflicts
  - Semantic heterogeneity
    - More important and harder to deal with
    - Synonyms, homonyms, hypernyms
    - Different ontology
    - Imprecise wording
Schema Matching (cont’d)

• Other complications
  – Insufficient schema and instance information
  – Unavailability of schema documentation
  – Subjectivity of matching

• Issues that affect schema matching
  – Schema versus instance matching
  – Element versus structure level matching
  – Matching cardinality
Schema Matching Approaches

Individual Matchers

Schema-based

Element-level
  - Linguistic
  - Constraint-based

Structure-level
  - Constraint-based

Instance-based

Element-level
  - Linguistic
  - Constraint-based
  - Learning-based
Schema Conflicts

• Name
  – Different names for equivalent entities, attributes, relationships, etc.
  – Same name for different entities, attributes, …

• Structure
  – Missing attributes
  – Missing but implicit attributes

• Relationship
  – One-to-many, many-to-many

• Entity versus Attribute (inclusion)
  – One attribute or several attributes

• Behavior
  – Different integrity constraints
    • Ex: automatic update, delete a project when the last engineer is moved to another project
Data Representation Conflicts

• Different representation for equivalent data
  – Different units
    • Celsius ↔ Farenheit; Kilograms ↔ Pounds; Liters ↔ Gallons;
  – Different levels of precision
    • 4 decimal digits versus 2 decimal digits
    • Floating point versus integer
  – Different expression denoting same information
    • Enumerated Value sets that are not one-to-one
      – {good, ok, bad} versus {one, two, three, four, five}

How to Resolve Schema Conflicts? Can Object-Oriented Models Help?
Suitability of OO Data Models as Global Data Models

• Rich set of type constructors
  -> easy representation of other data models

• Extensibility (user-defined types + type specific operators) & Encapsulation
  -> representation of “foreign” types/systems
  -> hiding heterogeneity (concrete storage) in a natural way

• Inheritance (generalization) & computational completeness
  -> schema integration
    - factor out common properties of similar types
    - thereby “arbitrary” computations possible
Use of Generalization & Comp. Completeness (Example)

class Person (  
  name: string,  
  address: Address)  
method net-income(): float;

is_a

class Employee (  
  salary: float,  
  course-given: set (Courses),  
  tax-rate: float)  
method net-income (): float  
return (self->salary * (1-self->tax-rate));

is_a

class Student (  
  grant: float,  
  course-enroll: set (Courses))  
method net-income (): float  
return (self->grant);

global data model

local data models
Conflict Resolution

• Renaming entities and attributes
  – Pick one name for the same things
  – Use unique prefixes for different things

• Homogenizing representations
  – Use conversions and mappings
    • stored programs in relational systems
    • methods in OO systems
    • auxiliary schemas to store conversion rules/code

• Homogenizing attributes
  – Use type coercion (e.g., integer to float)
  – Attribute concatenation (e.g., first name || last name)
  – For missing attributes, assign default values

• Homogenizing an attribute and an entity
  – Extract an attribute from the entity
    • Ex: Project department name from the Dept entity
even create a virtual attribute (e.g., Emp->Dept.name)
  – Create an entity from the attribute
    • Ex: Define default values and behavior for all other
      attributes of the Dept entity
Conflicts Resolution

- **Horizontal joins**
  - Union compatible
    - For missing attributes, assign default values or compute implicit values
  - Extended union compatible
    - Use generalization
      - Define a virtual class containing common attributes
    - Subclasses of the generalization
      - Provide specialized values and compute attribute values for generalized attributes
    - See earlier example
      - class Person generalizes class Student and class Employee

- **Vertical joins**
  - Many and many to one

- **Mixed Joins**
  - Vertical and horizontal joins in combination
Conflict Resolution involving a Database Key

• Entity-Attribute Conflicts where the Attribute is a DB key in one local schema

• Example:
  – The global schema defines Attr1 as an entity
  – Attr1 is a DB key for instances of LDB2-E

• If Attr1 is a complete DB key in LDB2, then in the global schema
  – Define entities E and D and relationship Rel
  – Define a new DB key attribute that will be used to uniquely identify instances of LDB2-E when they are accessed through GDB-E and GDB-D
Conflict Resolution involving a Partial Database Key

- **Entity-Attribute Conflicts** where the Attribute is a partial DB key in one local schema

**Example:**
- The global schema defines Attr1 as an entity
- Attr1 is a partial DB key for instances of LDB2-E

- **If Attr1 is a partial DB key in LDB2**
  - Define the entities E and D, and relationship Rel
  - Define a new attribute as a partial DB key
  - Add the other partial key attributes from LDB2 as partial keys
  - Add partial DB key LDB2-Attr1 as an attribute only
Global Schema Management

• HDBS manages the global schema $= \sum$ (all local exported schema)

• Global schema definition facilities provide mechanisms for handling the full spectrum of schematic differences that may exist among the heterogeneous local schemata.
  – Can use an Auxiliary Schema to store mappers, translators, and converters.

• Data is stored in the local component systems.

• Global dictionary information is used to query and manipulate the data. The global language statements are translated into equivalent statements of the local languages supported by the local systems.
Query Processing and Optimization

• The HDBMS has
  – A global Data Definition Language (DDL)
  – A global Data Manipulation Language (DML)
  – A set of local DMLs

• The HDBMS Query Processing Goal:
  – Given a query stated in the global query language (DML),
    execute that query, in an optimal manner, using the local database management systems
Query Planning and Optimization in a Distributed Multi-DBMS

Localized multi-DB query 1

query fragmentation and global optimization

SQ 1 → TQ 1 → DB 1
SQ 2 → TQ 2 → DB 2
SQ 3 → TQ 3 → DB 3
...
SQ n → TQ n → DB n

Another Multi-DBMS

Localized multi-DB query m

query localization

global query

PQ 1 → TQ n

Query translator 1
Query translator 2
Query translator 3
...
Query translator n

Joining intermediate results

Sorting and unioning result data

...
Information Supporting Query Planning & Optimization

Control Site

Global Query on Multiple Databases at Multiple Sites

Localization

{ Subqueries, each on a single Multi-DB }

{ Post-processing Queries }

Data Allocation

Multi-DB Manager

Fragmentation & Global Opt

{ Subqueries, each on a single local DBMS }

{ Post-processing Queries }

Data Directory

Translation

{ Queries, that can be processed by local DBMS }

Export & Aux Schema

Local DBMS

Decomposition & Local Optimization

Local Schema & Access Paths

Optimized Local Execution Plan
Query Fragmentation

• Similar to query fragmentation problem for homogeneous distributed DBSs

• But … Complicating factors:
  – Autonomy
    • Little information about “how” the subquery will be executed by the Local DBS
  – Heterogeneous Data Definition Languages
    • Weaker modeling languages do not support the same manipulation “features”
    • Must use multiple techniques in order to define a consistent global data model
    • Query fragmentation must produce a set of subqueries that reverse the operations used to create/define the global schema

• Processing Steps:
  (1) Replace names from the global schema with “fullnames” from the export schemas
  (2) If a subquery involves multiple export schemas, then break the query into queries that operate on one export schema and insert data communication operators to exchange intermediate results between local database systems
Global Query Optimization

• Similar to global query optimization for homogeneous distributed DBSs (many algorithms can be used directly)
• But only possible under the following assumptions:
  – No data inconsistency (the global schema correctly represents the semantics of disjoint, overlapping, and conflicting data)
  – Know the characteristics of local DBSs
    • e.g., statistical info on data cardinalities and selectivities are available
  – Can transfer partial data results between different local DBSs
    • Major impact on post-processing plans

• Primary Considerations:
  – Post-processing Strategy
  – Parallel Execution Possibilities
  – Global Cost Function/Estimation
Post-Processing Strategies

• Three Strategies:
  
  1) Control site performs all intermediate and post-processing operations (I&PP-ops)
     • Heavy work load; minimal parallelism
  
  2) Control site performs I&PP-ops for multi-DB results; Multi-DB managers, and HDBMS agents on the local database sites perform I&PP-ops for DBSs within one multi-DB environment
     • Better work load balance; more parallelism
  
  3) Use strategy #2 and use “pushdown” to get the local database systems to perform I&PP-ops
     • Possible if local DBMS can read intermediate results from external sources, and sort, join, etc. can be directly invoked
Parallel Execution Strategies

- Join operations are slow → speedup with parallel execution?
- Traditional query plans use left linear join trees
  - One of the operands is always a base relation
    - Have good info on cardinality and selectivity for the base
  - Used even in homogeneous distributed DBSs because cooperative nodes can pipeline the sequence of joins
- Bushy join trees provide parallel execution in heterogeneous multi-DB environments
  - Convert a left linear join tree into a (balanced?) bushy join tree
Global Cost Estimation

- Differs from cost estimation in homogeneous distributed DBSs
  - Little (or no) info on QP algorithms and data statistics in local DBS

- Cost Estimation Function
  - Cost to execute each subquery on the local DBMSs
  - Cost to execute all I&PP-ops
    - via pushdown or by any HDBMS agent/service

- Use a simplified cost function

\[
\text{Cost} = \text{Initialization cost} + \text{cost to retrieve a set of objects} + \text{cost to process a set of objects}
\]

- Run test queries on the local DBSs to get time estimates for ops
  - Selection, with and without an index
  - Join (testing for different algorithms: sort, hash, or indexed based algorithms)
Query Translation

When a query language of a local DBS is different from the global query language, each export schema subquery for the local DB needs to be translated from the global language to the target language.

Object-oriented (global) → Object-oriented (local)
Relational (global) → Relational (local)
Hierarchical (local) → Network-oriented (local)

Weaker target languages do not support the same operations, so emulate required operations in post-processing

Ex: retrieve more data than requested by the query and then post-process that data to compute the correct response to the query

Reduce the number of language mappings using the Entity-Relationship Query Language as an intermediary language
Query Translation - 2

(a) global query

"select all car company presidents that are 52 years old and own a car that is built in their hometown"

(b) relational predicate graph

Join Predicates:
(1) Company-OID (2) City-OID
(3) People-OID (4) Car-OID
(5) City1.name = City2.name

(c) object-oriented local schema <4 classes>

(d) object-oriented predicate graph

Object References (implicit & explicit joins):
(1) manufacturer (2) headquarter
(3) president (4) car
(5) hometown (6) City1.name = City2.name
HDBS Transaction Model

Global transactions

\[ \text{GT}_i \quad \text{GT}_j \]

\[ \text{GTM} - \text{global transaction manager} \]

\{ \text{GST}_{i1}, \text{GST}_{j1}, \text{GST}_{i2}, \text{GST}_{j2} \}
### Transaction Management

- **Local transactions**: access data at a single site outside of the global HDBS control.

- **Global transactions**: are executed under the HDBS control.

**Local DBMSs have three types of autonomy:**

<table>
<thead>
<tr>
<th>Autonomy Type</th>
<th>Definition</th>
<th>Resulting Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>No changes can be made to the local DBMS software to support the HDBMS</td>
<td>Non-serializable schedule for global transactions</td>
</tr>
<tr>
<td>Execution</td>
<td>Each local DBMS controls execution of global subtransactions and local transactions (the commit/abort decision)</td>
<td>Non-atomic &amp; non-durable global transactions</td>
</tr>
<tr>
<td>Communication</td>
<td>Local DBMS do not communicate with each other and they do not exchange execution control information</td>
<td>Distributed deadlock can not be detected</td>
</tr>
</tbody>
</table>
Global Serializability Problem

- GTM is responsible for
  - A serializable schedule for the set of global transactions
  - Coordination of submission and execution of global subtransactions among the local DBMSs

- Serializing the global schedule?

\[
\begin{align*}
\text{GST}_{11} \prec \& \text{GST}_{22} \text{ at site DBMS-1,} \\
\text{Then it must be the case that } \text{GST}_{12} \prec \& \text{GST}_{23} \text{ at site DBMS-2} \\
\text{GST}_{23} \prec \& \text{GST}_{12} \text{ at site DBMS-2} \Rightarrow \text{GT}_2 \prec \& \text{GT}_1 \\
\end{align*}
\]

A non-serializable schedule!
Local Transactions and the Global Serializable Schedule

- Local transactions execute outside the control of the GTM
- Local transactions create indirect conflicts with global transactions
- GTM is not aware of local transactions and these indirect conflicts
- In general, the GTM cannot ensure global serializability
Controlling the Execution Order of Global Subtransactions

• Three Strategies:

1) Execute global transactions serially
   • No concurrent execution for global transactions!
   • Does not solve indirect conflicts with local transactions

2) Relax the serializability/consistency requirement
   • Use “strong correctness” instead
   • Most indirect conflicts have no effect on correctness

3) Define a specific order over the global transactions and use the concurrency control mechanism of each local DBMS to enforce that order
   • Use a local database “ticket”
Alternative Consistency Notions

Constraint-based strategies

• Local serializability: In some HDBS applications there may be no global constraints because each DBS is quite independent from others and may wish to remain that way. => no global concurrency control mechanism needed
  That is, local serializability is sufficient to ensure strong correctness of global executions.
  – Example application: travel reservation service for planes, trains, ferries, hotels, etc.

• Handling global constraints: In some applications we need global constraints. However, it may still be possible to enforce them without the full generality of globally serializable schedules (two-level serializability, 2LSR). The data that can be involved in global constraints are limited. Two types of data: global and local data. Global constraints may only span global data, and local transactions may not write to global data.
  – Artificial solution: local site has no autonomy over global data; master-slave relationship.

Non-constraint-based strategies

• Other approaches: extend the allowable schedules beyond global serializability, e.g., epsilon serializability (schedule can have a limited number of nonserializable conflicts), or define sets of compatible transactions that are known to be interleavable.
Global Serializability Schemes

Failure-free environment where the local DBMSs cannot unilaterally abort transactions (unrealistic case, but we can relax some of these conditions later).

- **Unknown DBMSs**: the GTM ensures that all global transactions will conflict at every site where they execute together. If a pair of transactions does not naturally conflict, then the GTM modifies them so that they do conflict. Each local site has a special data item (called a **ticket**). Every subtransaction reads and writes the ticket:

  GT1: r1(a) w1(a)
  GT2: r2(b) w2(b)

  newGT1: r1(ticketS1) r1(a) w1(a) w1(ticketS1) c1
  newGT2: r2(ticketS1) r2(b) w2(b) w2(ticketS2) c2

  - **Means GT1 and GT2 will be correctly serialized with respect to all global transactions and all local transaction executed by the local DBMS at S1**

- **Rigorous DBMSs**: scenario where the GTM knows that all local DBMSs use the rigorous (strict) two-phase locking protocol (R2PL). With local R2PL, global serializability can be ensured as long as the GTM does not issue any commits for a transaction until all its actions have been completed.

  Severe performance issues with these approaches
Global Atomicity and Recovery Problem

- The GTM must guarantee that a global transaction commits at all sites or aborts at all sites.
- Local DBMSs wish to preserve their execution autonomy:
  - May not implement or export a prepare-to-commit interface.

A local DBMS can unilaterally abort a subtransaction anytime:
- Results in non-atomic global transactions and incorrect global schedules.
- Local transactions and global subtransactions see committed partial results.

Note: The first heterogeneous systems did not support update transactions!
Approaches to Achieve Atomicity and Durability

- If all LDBMSs export a “prepare-to-commit” interface, then use 2PC between the proxy and the LDBMS
- If some LDBMSs do not export “prepare-to-commit”, then three approaches:
  1) Modify each global subtransaction to “callback to the proxy” just before local commit
     - Blocks the global subtransaction until GTM completes 2PC with proxies
     - Possibly only if the LDBMS supports a client callback service
     - Fails if the LDBMS is running optimistic concurrency control
       - If any global subtransaction aborts
         2) Attempt to REDO that global subtransaction
            - Other transactions see inconsistent data until the redo is successful
         3) Execute compensating transactions to UNDO the committed global subtransactions
            - Other transaction see inconsistent data until the undo is completed
Global Deadlock Problem

• Same problem as in distributed homogeneous DBMSs

![Diagram of global deadlock]

Site X
T1 x needs a 
waits for T1 y 
to complete
holds lock Lx

T1 x waits for T1 y to release Lx

T2 x
holds lock Lb
T2 x waits for T2 y to complete

Site Y
T1 y
holds lock La

T1 y waits for T1 y to release Ly

T2 y
holds lock Ly
T2 y needs b 
waits for T2 x 
to complete

• We solved the problem by exchanging lock information to construct the global “waits-for” graph
  – This violates design autonomy and communication autonomy

• Therefore the GTM will be unaware of a global deadlock.

• There are no complete solutions to the global deadlock problem for autonomous multi-database systems.
Status: Transaction Management for HDBS

- Transaction management for HDBSs is a very active research area.
- Distributed transactions over the Internet define new semantic possibilities, allowing development of new solutions.

Open issues:
- What can be done if some of the local subsystems (e.g., file systems) do not support transaction management?
- Performance implications of transaction management strategy?
- Handling of different degrees of consistency?
Conclusions

HDBS allows
a uniform view on the combination of data maintained by different autonomous database systems.

- **available**: prototypes & commercial products with a set of fixed / specific drivers (so-called gateways) for existing, widely used data management systems (conventional DBS and file systems)

- **missing**: systematic support for individual integration of arbitrary data management systems (especially modern DBS)