
Principles of Distributed Database Systems

M. Tamer Özsu
Patrick Valduriez

Outline

- Introduction
- Distributed and Parallel Database Design
- Distributed Data Control
- Distributed Query Processing
- Distributed Transaction Processing
- Data Replication
- Database Integration – Multidatabase Systems
- Parallel Database Systems
- Peer-to-Peer Data Management
- Big Data Processing
- NoSQL, NewSQL and Polystores
- Web Data Management

Outline

- Database Integration – Multidatabase Systems
 - Schema Matching
 - Schema Integration
 - Schema Mapping
 - Query Rewriting
 - Optimization Issues

Problem Definition

- 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*
- Bottom-up design process

Integration Alternatives

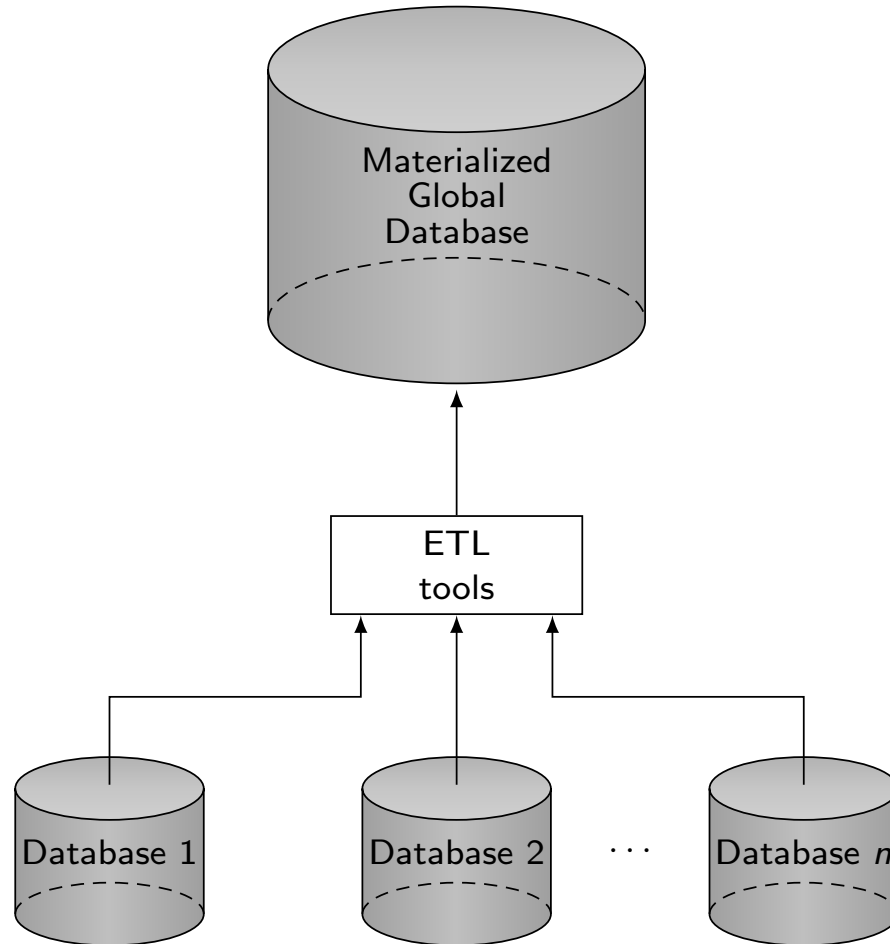
■ Physical integration

- ❑ 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



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

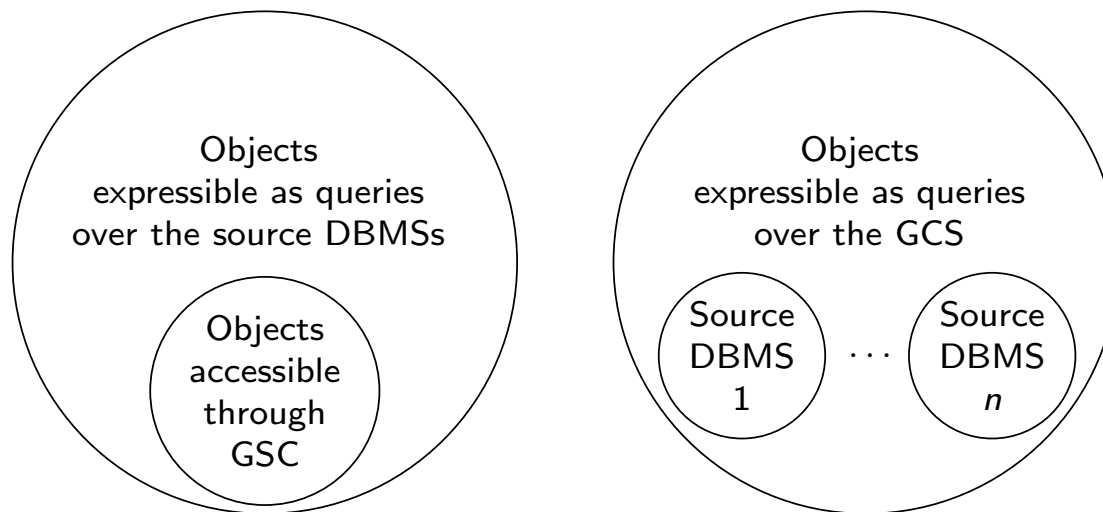
GCS/LCS Relationship

■ Local-as-view

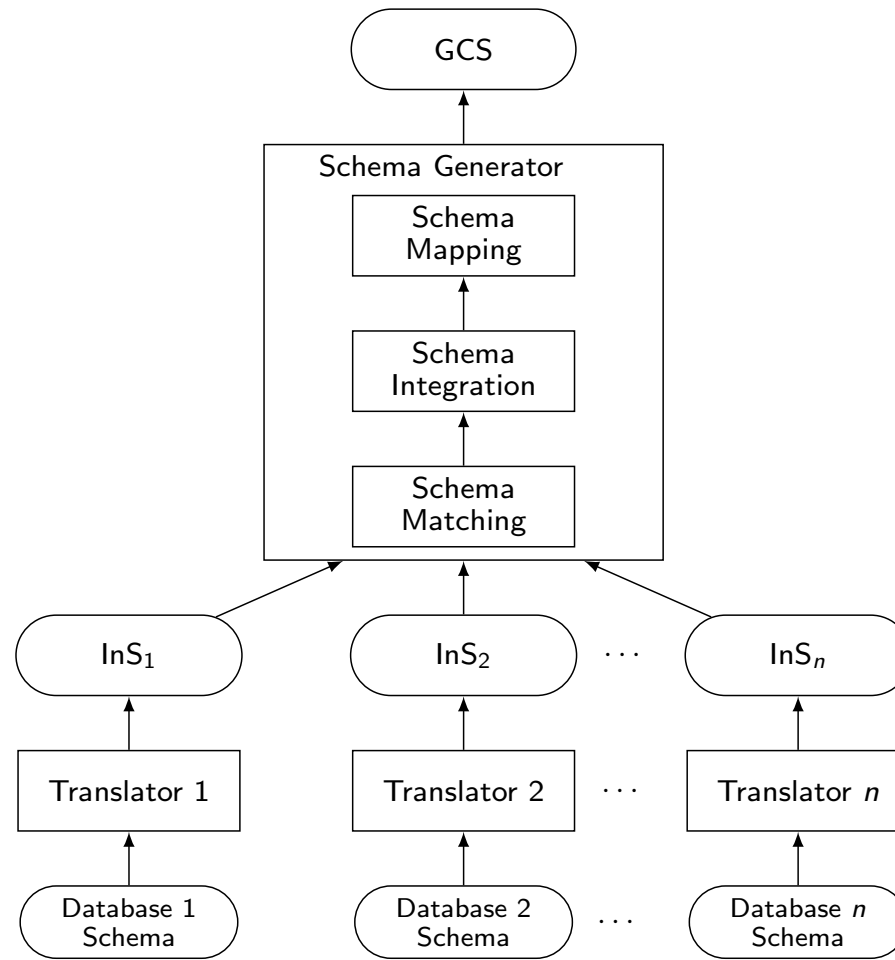
- ▣ The GCS definition is assumed to exist, and each LCS is treated as a view definition over it

■ Global-as-view

- ▣ The GCS is defined as a set of views over the LCSs



Database Integration Process



Database Integration Issues – Schema Translation

- Component database schemas translated to a common intermediate canonical representation
- What is the canonical data model?
 - Relational
 - Entity-relationship
 - DIKE
 - Object-oriented
 - ARTEMIS
 - Graph-oriented
 - DIPE, TranScm, COMA, Cupid
- Translation algorithms
 - These are well-known

Database Integration Issues – Schema Generation

- Intermediate schemas are used to create a global conceptual schema
- 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

Outline

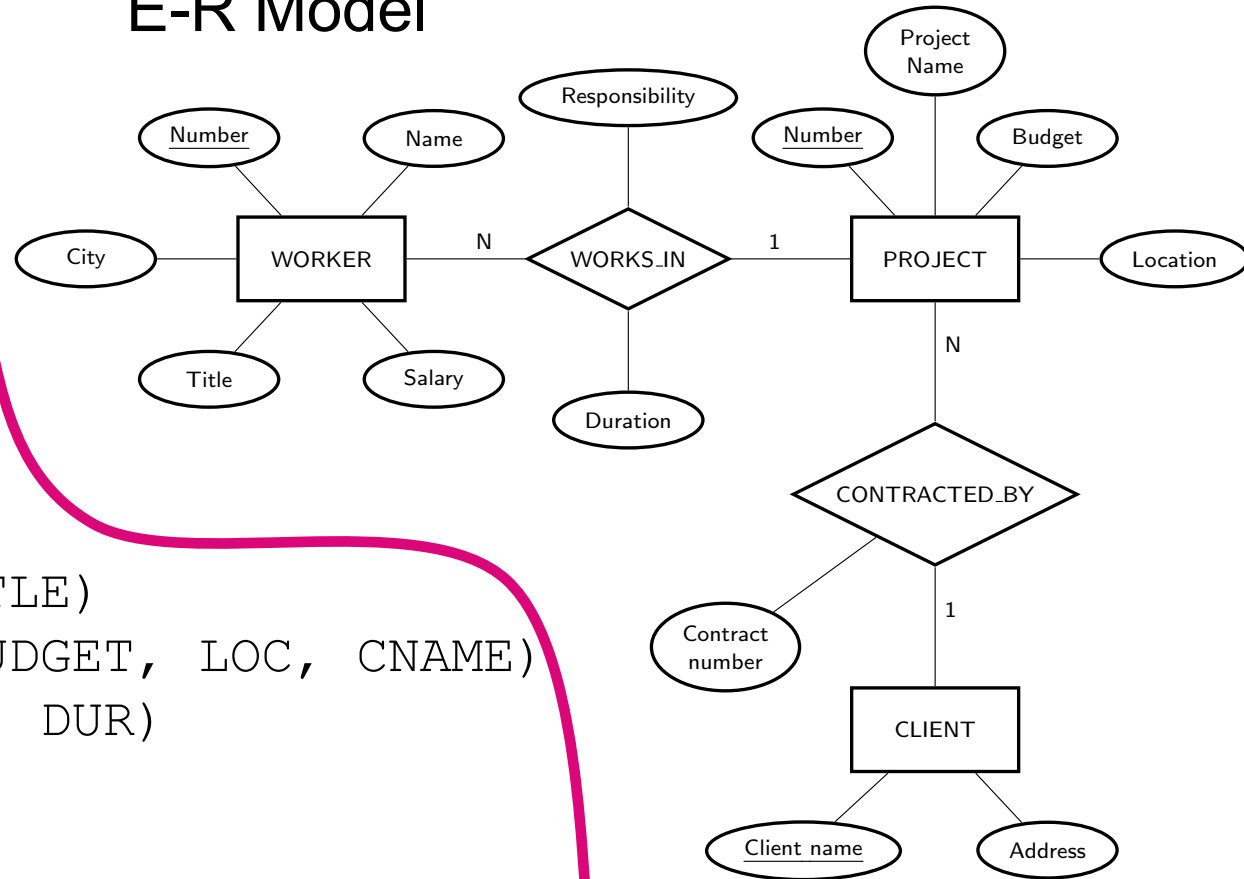
- Database Integration – Multidatabase Systems
 - Schema Matching
 - Schema Integration
 - Schema Mapping
 - Query Rewriting
 - Optimization Issues

Running Example

Relational

EMP(ENO, ENAME, TITLE)
PROJ(PNO, PNAME, BUDGET, LOC, CNAME)
ASG(ENO, PNO, RESP, DUR)
PAY(TITLE, SAL)

E-R Model



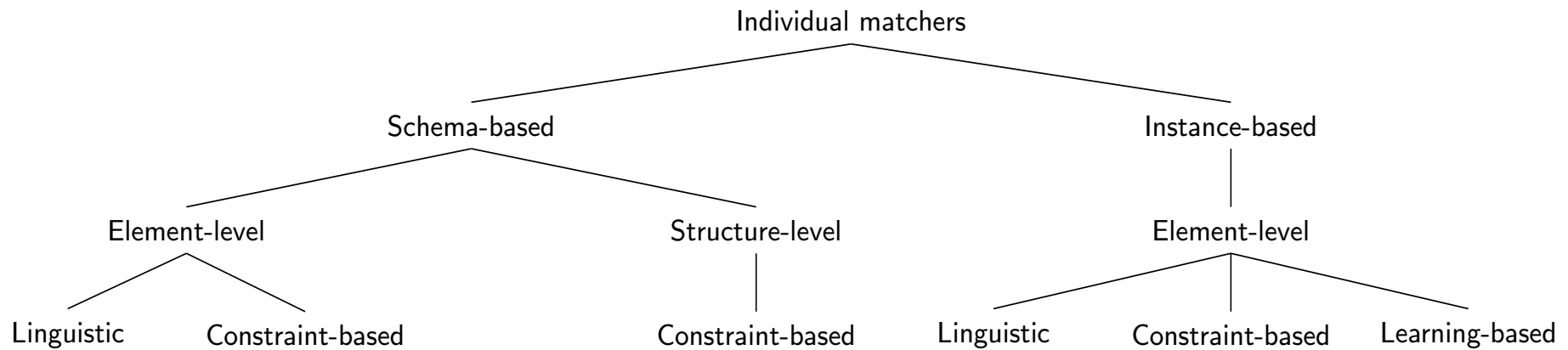
Schema Matching

- 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



Linguistic Schema Matching

- Use element names and other textual information (textual descriptions, annotations)
- May use external sources (e.g., Thesauri)
- $\langle \text{SC1.element-1} \approx \text{SC2.element-2}, p, s \rangle$
 - Element-1 in schema SC1 is similar to element-2 in schema SC2 if predicate p holds with a similarity value of s
- Schema level
 - Deal with names of schema elements
 - Handle cases such as synonyms, homonyms, hypernyms, data type similarities
- Instance level
 - Focus on information retrieval techniques (e.g., word frequencies, key terms)
 - “Deduce” similarities from these

Linguistic Matchers

- Use a set of linguistic (terminological) rules
- Basic rules can be hand-crafted or may be discovered from outside sources (e.g., WordNet)
- Predicate p and similarity value s
 - ❑ hand-crafted \Rightarrow specified,
 - ❑ discovered \Rightarrow may be computed or specified by an expert after discovery
- Examples
 - ❑ $\langle \text{uppercase names} \approx \text{lower case names}, \text{true}, 1.0 \rangle$
 - ❑ $\langle \text{uppercase names} \approx \text{capitalized names}, \text{true}, 1.0 \rangle$
 - ❑ $\langle \text{capitalized names} \approx \text{lower case names}, \text{true}, 1.0 \rangle$
 - ❑ $\langle \text{DB1.ASG} \approx \text{DB2.WORKS_IN}, \text{true}, 0.8 \rangle$

Automatic Discovery of Name Similarities

- Affixes
 - Common prefixes and suffixes between two element name strings
- N-grams
 - Comparing how many substrings of length n are common between the two name strings
- Edit distance
 - Number of character modifications (additions, deletions, insertions) that needs to be performed to convert one string into the other
- Soundex code
 - Phonetic similarity between names based on their soundex codes
- Also look at data types
 - Data type similarity may suggest stronger relationship than the computed similarity using these methods or to differentiate between multiple strings with same value

N-gram Example

- 3-grams of string “Responsibility” are the following:

● Res	● sib
● ibi	● esp
● bip	● spo
● ili	● pon
● lit	● ons
● ity	● nsi

- 3-grams of string “Resp” are

- Res
- esp

- 3-gram similarity: $2/12 = 0.17$

Edit Distance Example

- Again consider “Responsibility” and “Resp”
- To convert “Responsibility” to “Resp”
 - Delete characters “o”, “n”, “s”, “i”, “b”, “i”, “l”, “i”, “t”, “y”
- To convert “Resp” to “Responsibility”
 - Add characters “o”, “n”, “s”, “i”, “b”, “i”, “l”, “i”, “t”, “y”
- The number of edit operations required is 10
- Similarity is $1 - (10/14) = 0.29$

Constraint-based Matchers

- Data always have constraints – use them
 - ❑ Data type information
 - ❑ Value ranges
 - ❑ ...
- Examples
 - ❑ RESP and RESPONSIBILITY: n-gram similarity = 0.17, edit distance similarity = 0.19 (low)
 - ❑ If they come from the same domain, this may increase their similarity value
 - ❑ ENO in relational, WORKER.NUMBER and PROJECT.NUMBER in E-R
 - ❑ ENO and WORKER.NUMBER may have type INTEGER while PROJECT.NUMBER may have STRING

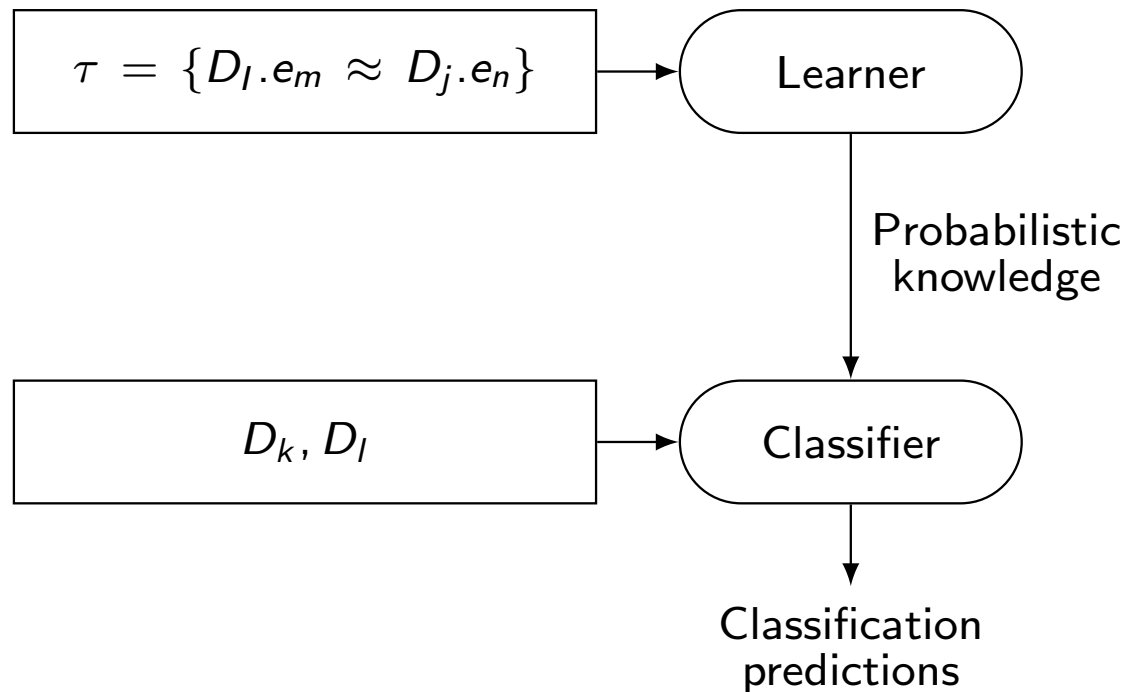
Constraint-based Structural Matching

- If two schema elements are structurally similar, then there is a higher likelihood that they represent the same concept
- Structural similarity:
 - Same properties (attributes)
 - “Neighborhood” similarity
 - Using graph representation
 - The set of nodes that can be reached within a particular path length from a node are the neighbors of that node
 - If two concepts (nodes) have similar set of neighbors, they are likely to represent the same concept

Learning-based Schema Matching

- Use machine learning techniques to determine schema matches
- Classification problem: classify concepts from various schemas into classes according to their similarity. Those that fall into the same class represent similar concepts
- Similarity is defined according to features of *data instances*
- Classification is “learned” from a training set

Learning-based Schema Matching



Combined Schema Matching Approaches

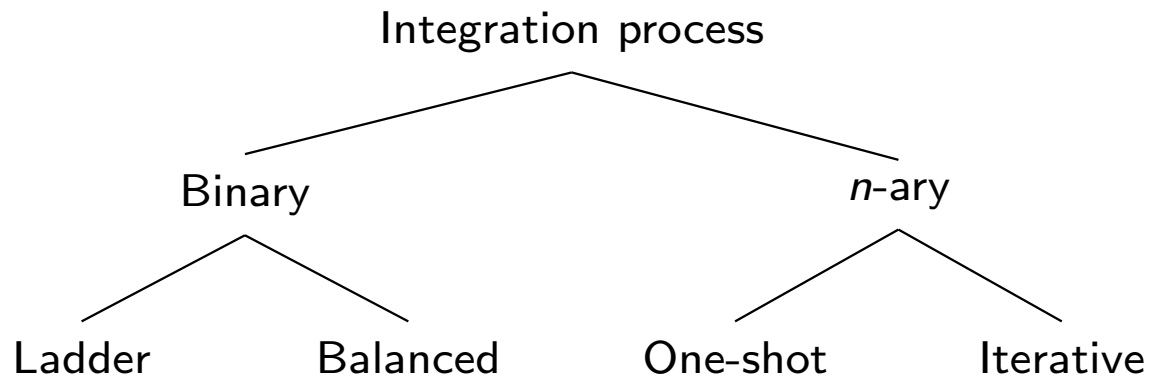
- Use multiple matchers
 - ▣ Each matcher focuses on one area (name, etc)
- Meta-matcher integrates these into one prediction
- Integration may be simple (take average of similarity values) or more complex (see Fagin's work)

Outline

- Database Integration – Multidatabase Systems
 - Schema Matching
 - Schema Integration
 - Schema Mapping
 - Query Rewriting
 - Optimization Issues

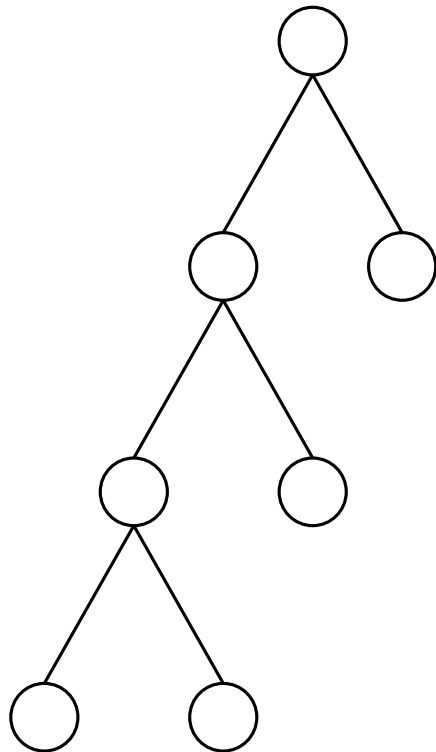
Schema Integration

- Use the correspondences to create a GCS
- Mainly a manual process, although rules can help

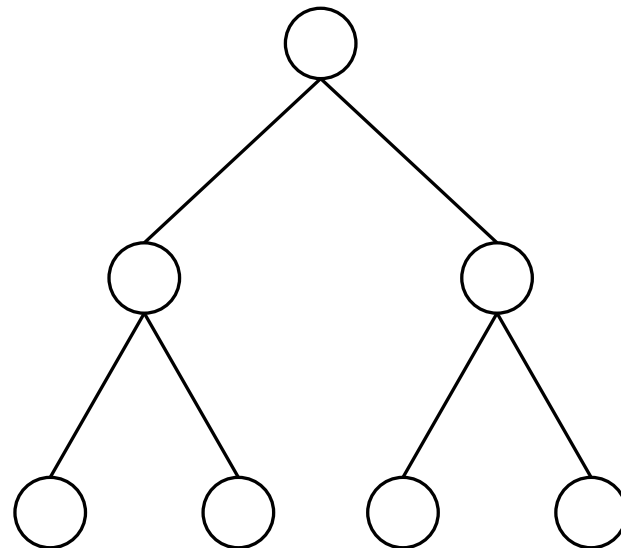


Binary Integration Methods

Stepwise

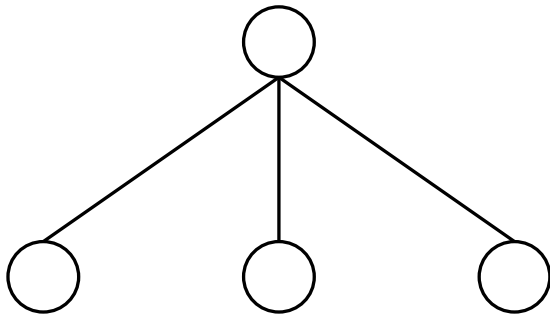


Pure binary

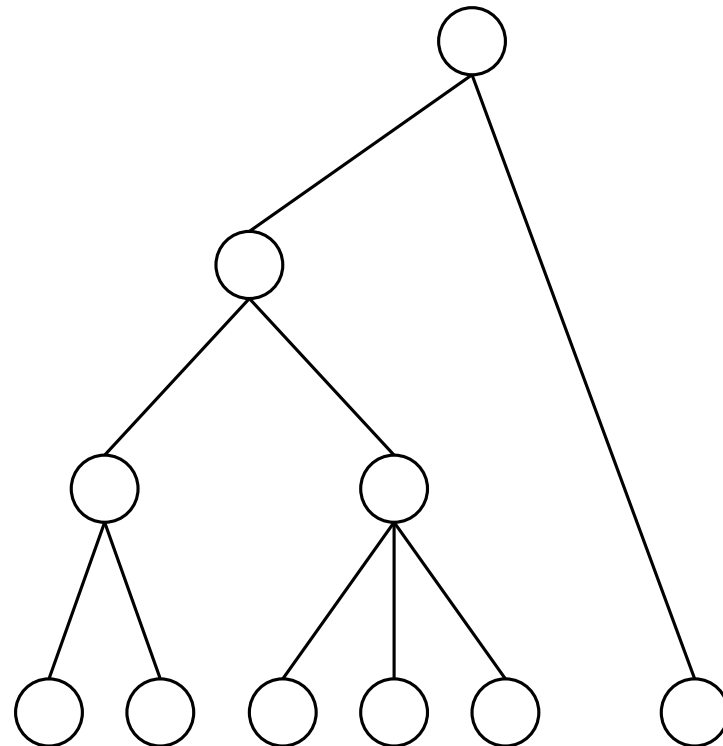


N-ary Integration Methods

One-pass



Iterative



Outline

- Database Integration – Multidatabase Systems
 - Schema Matching
 - Schema Integration
 - Schema Mapping
 - Query Rewriting
 - Optimization Issues

Schema Mapping

- Mapping data from each local database (source) to GCS (target) while preserving semantic consistency as defined in both source and target.
- Data warehouses \Rightarrow actual translation
- Data integration systems \Rightarrow discover mappings that can be used in the query processing phase
- Mapping creation
- Mapping maintenance

Mapping Creation

Given

- ❑ A source LCS: $\mathcal{S} = \{S_i\}$
- ❑ A target GCS: $\mathcal{T} = \{T_i\}$
- ❑ A set of value correspondences discovered during schema matching phase: $\mathcal{V} = \{V_i\}$

Produce a set of queries that, when executed, will create GCS data instances from the source data.

We are looking, for each T_k , a query Q_k that is defined on a (possibly proper) subset of the relations in \mathcal{S} such that, when executed, will generate data for T_i from the source relations

Mapping Creation Algorithm

General idea:

- Consider each T_k in turn.
 - ▣ Divide V_k into subsets $\{V_k^1, \dots, V_k^n\}$ such that each V_k^j specifies one possible way that values of T_k can be computed
- Each V_k^j can be mapped to a query q_k^j that, when executed, would generate *some* of T_k 's data.
- Union of these queries gives $Q_k (= \cup_j q_k^j)$

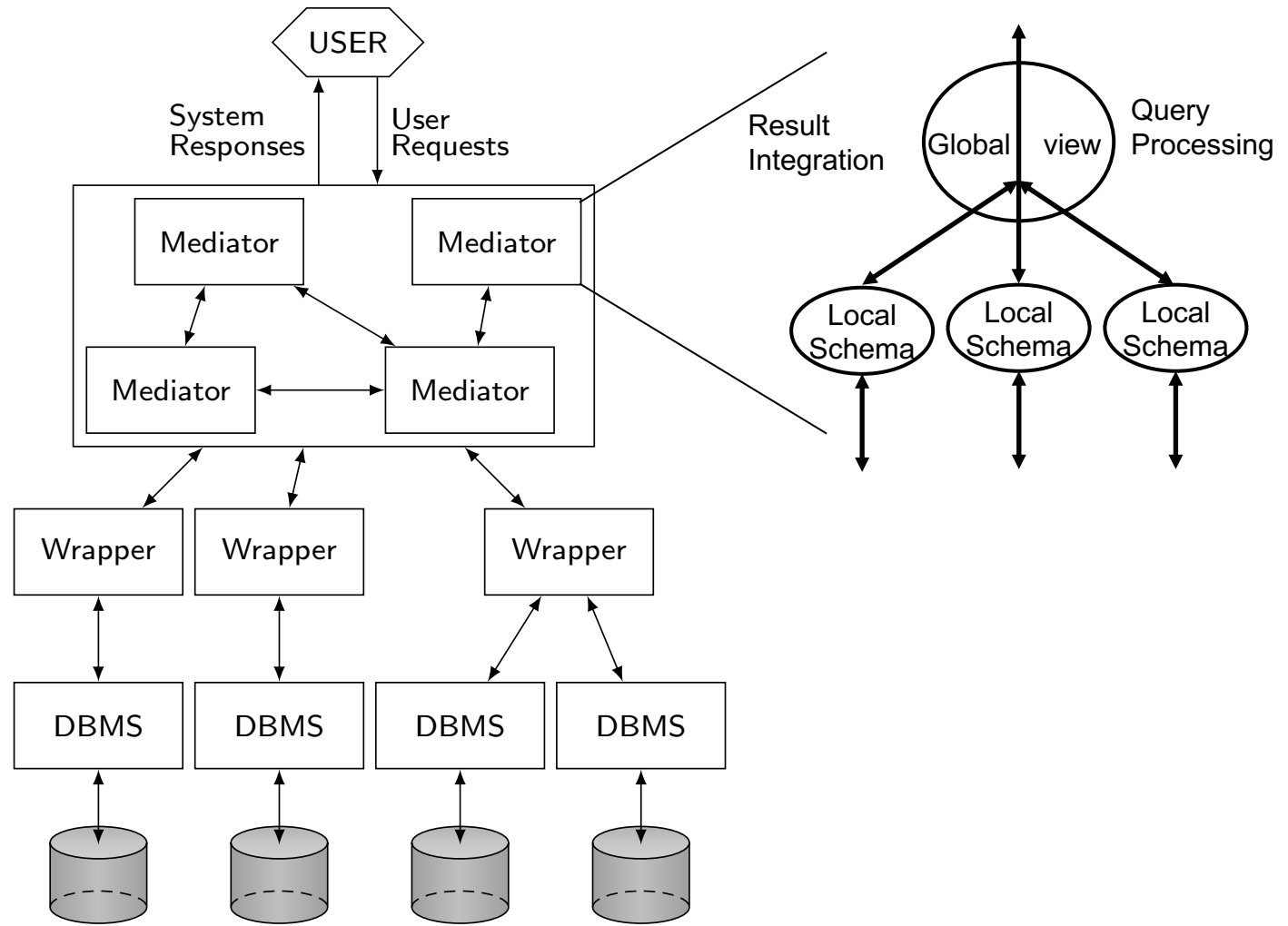
Outline

- Database Integration – Multidatabase Systems
 - Schema Matching
 - Schema Integration
 - Schema Mapping
 - Query Rewriting
 - Optimization Issues

Multidatabase Query Processing

- Mediator/wrapper architecture
- MDB query processing architecture
- Query rewriting using views
- Query optimization and execution
- Query translation and execution

Recall Mediator/Wrapper Architecture



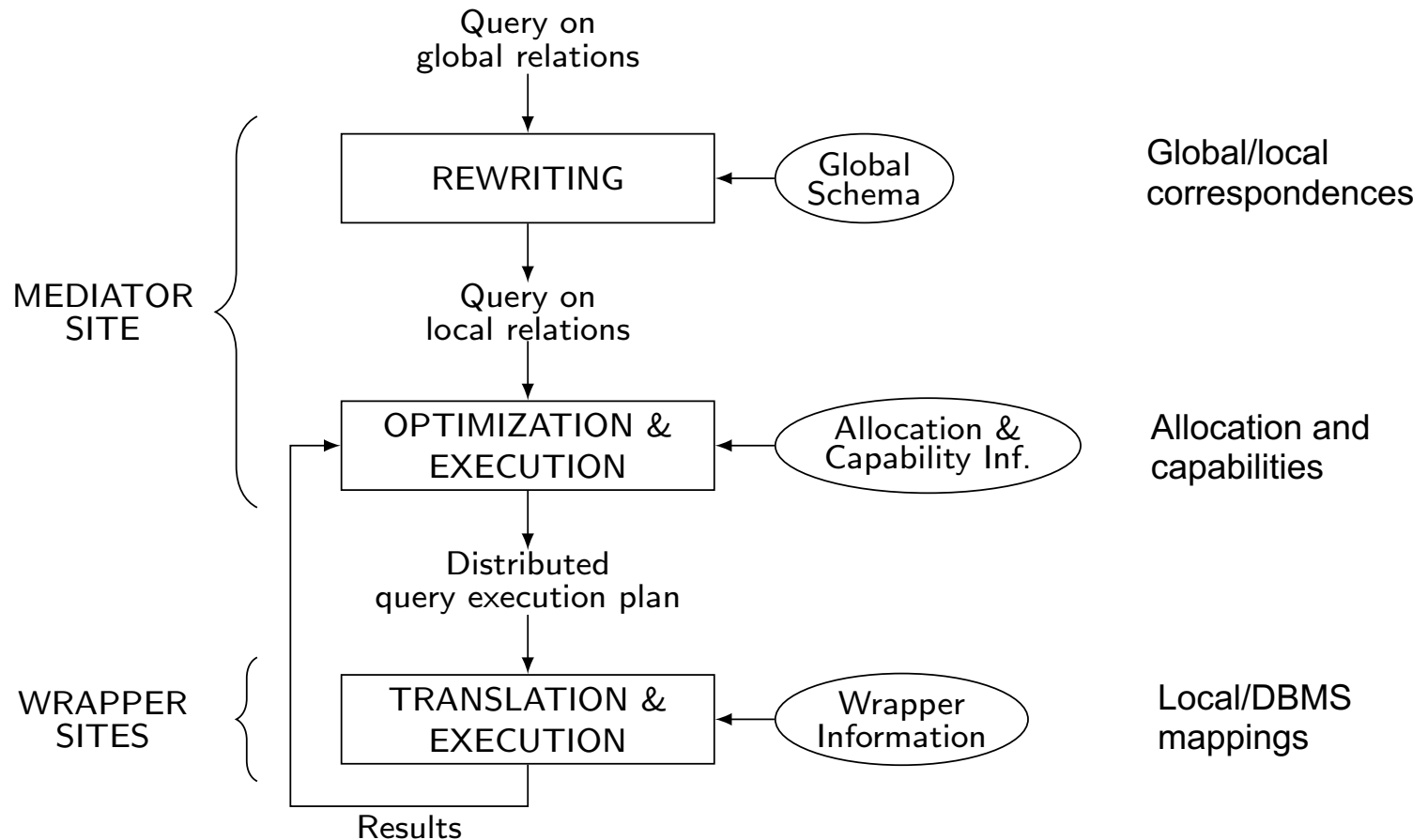
Issues in MDB Query Processing

- Component DBMSs are autonomous and may range from full-fledge relational DBMS to flat file systems
 - Different computing capabilities
 - Prevents uniform treatment of queries across DBMSs
 - Different processing cost and optimization capabilities
 - Makes cost modeling difficult
 - Different data models and query languages
 - Makes query translation and result integration difficult
 - Different runtime performance and unpredictable behavior
 - Makes query execution difficult

Component DBMS Autonomy

- Communication autonomy
 - ❑ The ability to terminate services at any time
 - ❑ How to answer queries completely?
- Design autonomy
 - ❑ The ability to restrict the availability and accuracy of information needed for query optimization
 - ❑ How to obtain cost information?
- Execution autonomy
 - ❑ The ability to execute queries in unpredictable ways
 - ❑ How to adapt to this?

MDB Query Processing Architecture



Query Rewriting Using Views

- Views used to describe the correspondences between global and local relations
 - ❑ **Global As View**: the global schema is integrated from the local databases and each global relation is a view over the local relations
 - ❑ **Local As View**: the global schema is defined independently of the local databases and each local relation is a view over the global relations
- Query rewriting best done with Datalog, a logic-based language
 - ❑ More expressive power than relational calculus
 - ❑ Inline version of relational domain calculus

Datalog Terminology

- Conjunctive (SPJ) query: a rule of the form
 - ❑ $Q(T) :- R_1(T_1), \dots R_n(T_n)$
 - ❑ $Q(T)$: head of the query denoting the result relation
 - ❑ $R_1(T_1), \dots R_n(T_n)$: subgoals in the body of the query
 - ❑ $R_1, \dots R_n$: predicate names corresponding to relation names
 - ❑ $T_1, \dots T_n$: refer to tuples with variables and constants
 - ❑ Variables correspond to attributes (as in domain calculus)
 - ❑ “-” means unnamed variable
- Disjunctive query = n conjunctive queries with same head predicate

Datalog Example

EMP (E#, ENAME, TITLE, CITY)
WORKS (E#, P#, RESP, DUR)

```
SELECT    E#, TITLE, P#  
FROM      EMP NATURAL JOIN WORKS  
WHERE      TITLE = "Programmer" OR DUR=24
```

$Q(E\#, TITLE, P\#)$:- EMP (E#, ENAME, "Programmer", CITY),
WORKS (E#, P#, RESP, DUR) .

$Q(E\#, TITLE, P\#)$:- EMP (E#, ENAME, TITLE, CITY),
WORKS (E#, P#, RESP, 24) .

Rewriting in GAV

- Global schema similar to that of homogeneous distributed DBMS
 - Local relations can be fragments
 - But no completeness: a tuple in the global relation may not exist in local relations
 - Yields incomplete answers
 - And no disjointness: the same tuple may exist in different local databases
 - Yields duplicate answers
- Rewriting (*unfolding*)
 - Similar to query modification
 - Apply view definition rules to the query and produce a union of conjunctive queries, one per rule application
 - Eliminate redundant queries

GAV Example Schema

Global relations

EMP (E#, ENAME, CITY)

WORKS (E#, P#, TITLE, DUR)

Local relations

EMP1 (E#, ENAME, TITLE, CITY)

EMP2 (E#, ENAME, TITLE, CITY)

WORKS (E#, P#, DUR)

EMP (E#, ENAME, CITY) :- EMP1 (E#, ENAME, TITLE, CITY) . (d_1)

EMP (E#, ENAME, CITY) :- EMP2 (E#, ENAME, TITLE, CITY) . (d_2)

WORKS (E#, P#, TITLE, DUR) :- EMP1 (E#, ENAME, TITLE, CITY) ,
WORKS (E#, P#, DUR) . (d_3)

WORKS (E#, P#, TITLE, DUR) :- EMP2 (E#, ENAME, TITLE, CITY) ,
WORKS (E#, P#, DUR) . (d_4)

GAV Example Query

Let Q: project for employees in Paris

$$Q(e, p) \text{ :- EMP}(e, \text{ENAME}, \text{"Paris"}), \text{WORKS}(e, p, \text{TITLE}, \text{DUR}) .$$

Unfolding produces Q'

$$\begin{aligned} Q'(e, p) \text{ :- EMP1}(e, \text{ENAME}, \text{"Paris"}), \\ \text{WORKS}(e, p, \text{TITLE}, \text{DUR}) . \end{aligned} \quad (q_1)$$
$$\begin{aligned} Q'(e, p) \text{ :- EMP2}(e, \text{ENAME}, \text{"Paris"}), \\ \text{WORKS}(e, p, \text{TITLE}, \text{DUR}) . \end{aligned} \quad (q_2)$$

where

q_1 is obtained by applying d_3 only or both d_1 and d_3

In the latter case, there are redundant queries

same for q_2 with d_2 only or both d_2 and d_4

Rewriting in LAV

- More difficult than in GAV
 - No direct correspondence between the terms in GS (EMP, ENAME) and those in the views (EMP1, EMP2, ENAME)
 - There may be many more views than global relations
 - Views may contain complex predicates to reflect the content of the local relations
 - e.g. a view EMP3 for only programmers
- Often not possible to find an equivalent rewriting
 - Best is to find a *maximally-contained query* which produces a maximum subset of the answer
 - e.g. EMP3 can only return a subset of the employees

Rewriting Algorithms

- The problem to find an equivalent query is NP-complete in the number of views and number of subgoals of the query
- Thus, algorithms try to reduce the numbers of rewritings to be considered
- Three main algorithms
 - Bucket
 - Inverse rule
 - MiniCon

LAV Example Schema

Local relations

EMP1 (E#, ENAME, TITLE, CITY)

EMP2 (E#, ENAME, TITLE, CITY)

WORKS1 (E#, P#, DUR)

Global relations

EMP (E#, ENAME, CITY)

WORKS (E#, P#, TITLE, DUR)

$$\text{EMP1 (E\#,ENAME,TITLE,CITY)} :- \text{EMP (E\#,ENAME,CITY)} \quad (d_5)$$

$$\text{WORKS (E\#,P\#,TITLE,DUR)} .$$
$$\text{EMP1 (E\#,ENAME,TITLE,CITY)} :- \text{EMP (E\#,ENAME,CITY)} \quad (d_6)$$

$$\text{WORKS (E\#,P\#,TITLE,DUR)} .$$
$$\text{WORKS}(E\#, P\#, \text{DUR}) :- \text{WORKS}(E\#, P\#, \text{TITLE}, \text{DUR}) . \quad (d_7)$$

Bucket Algorithm

- Considers each predicate of the query Q independently to select only the relevant views

Step 1

- Build a bucket b for each subgoal q of Q that is not a comparison predicate
- Insert in b the heads of the views which are relevant to answer q

Step 2

- For each view V of the Cartesian product of the buckets, produce a conjunctive query
 - If it is contained in Q , keep it
- The rewritten query is a union of conjunctive queries

LAV Example Query

$Q(e, p) \text{ :- EMP}(e, \text{ENAME}, \text{"Paris"}), \text{WORKS}(e, p, \text{TITLE}, \text{DUR}).$

Step1: we obtain 2 buckets (one for each subgoal of Q)

$b_1 = \{ \text{EMP1}(E\#, \text{ENAME}, \text{TITLE}', \text{CITY}),$
 $\text{EMP2}(E\#, \text{ENAME}, \text{TITLE}', \text{CITY}) \}$

$b_2 = \{ \text{WORKS1}(E\#, P\#, \text{DUR}') \}$

(the prime variables (TITLE' and DUR') are not useful)

Step2: produces

$Q'(e, p) \text{ :- EMP1}(e, \text{ENAME}, \text{TITLE}, \text{"Paris"}),$
 $\text{WORKS1}(e, p, \text{DUR}). \quad (q_1)$

$Q'(e, p) \text{ :- EMP2}(e, \text{ENAME}, \text{TITLE}, \text{"Paris"}),$
 $\text{WORKS1}(e, p, \text{DUR}). \quad (q_2)$

Outline

- Database Integration – Multidatabase Systems
 - Schema Matching
 - Schema Integration
 - Schema Mapping
 - Query Rewriting
 - Optimization Issues

Query Optimization and Execution

- Takes a query expressed on local relations and produces a distributed QEP to be executed by the wrappers and mediator
- Three main problems
 - Heterogeneous cost modeling
 - To produce a global cost model from component DBMS
 - Heterogeneous query optimization
 - To deal with different query computing capabilities
 - Adaptive query processing
 - To deal with strong variations in the execution environment

Heterogeneous Cost Modeling

- Goal: determine the cost of executing the subqueries at component DBMS
- Three approaches
 - ❑ Black-box: treats each component DBMS as a black-box and determines costs by running test queries
 - ❑ Customized: customizes an initial cost model
 - ❑ Dynamic: monitors the run-time behavior of the component DBMS and dynamically collect cost information

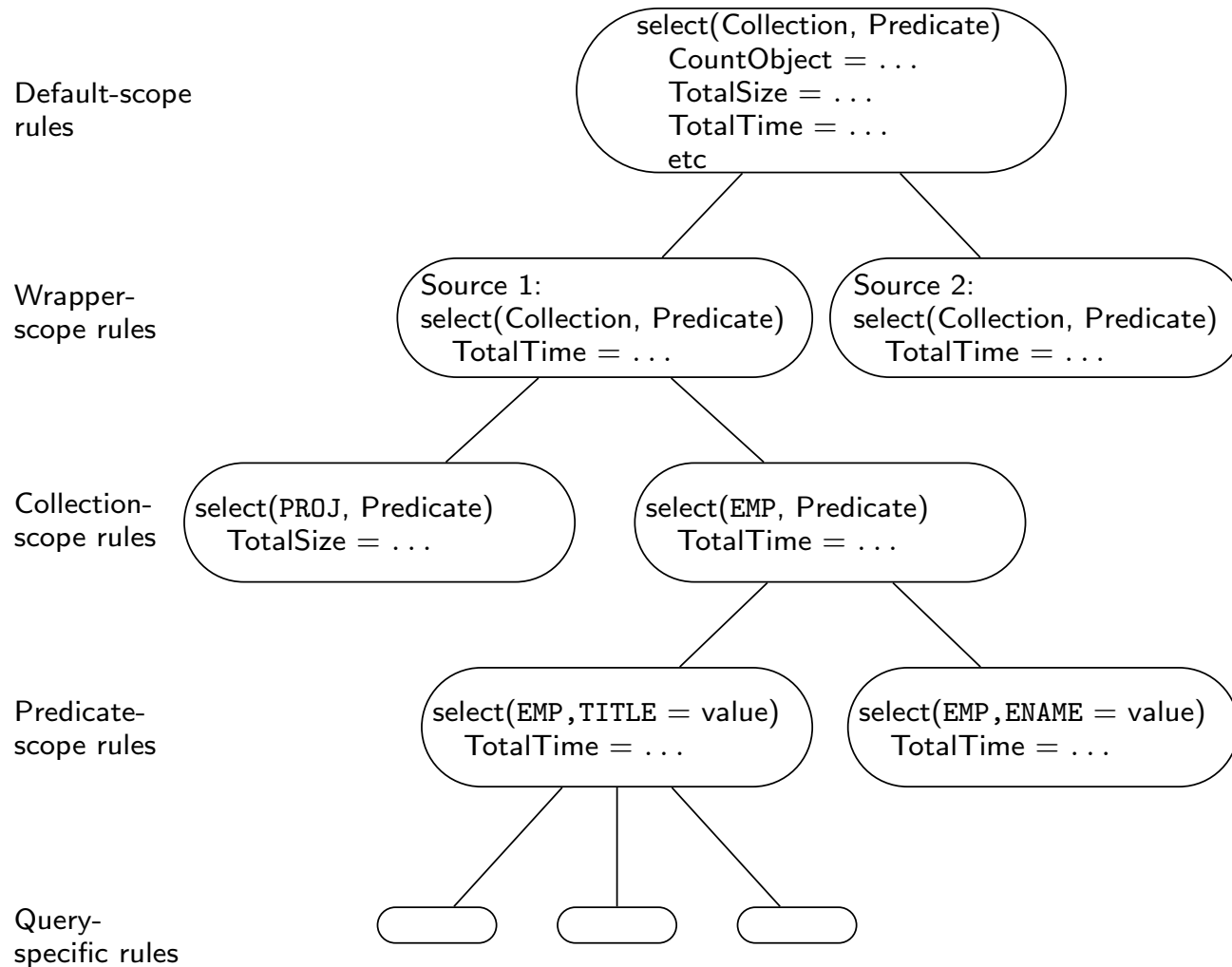
Black-box Approach

- Define a logical cost expression
 - $Cost = \text{init cost} + \text{cost to find qualifying tuples} + \text{cost to process selected tuples}$
 - The terms will differ much with different DBMS
- Run probing queries on component DBMS to compute cost coefficients
 - Count the numbers of tuples, measure cost, etc.
 - Special case: sample queries for each class of important queries
 - Use of classification to identify the classes
- Problems
 - The instantiated cost model (by probing or sampling) may change over time
 - The logical cost function may not capture important details of component DBMS

Customized Approach

- Relies on the wrapper (i.e. developer) to provide cost information to the mediator
- Two solutions
 - Wrapper provides the logic to compute cost estimates
 - $\text{Access_cost} = \text{reset} + (\text{card}-1) * \text{advance}$
 - reset = time to initiate the query and receive a first tuple
 - advance = time to get the next tuple (advance)
 - card = result cardinality
 - Hierarchical cost model
 - Each node associates a query pattern with a cost function
 - The wrapper developer can give cost information at various levels of details, depending on knowledge of the component DBMS

Hierarchical Cost Model



Dynamic Approach

- Deals with execution environment factors which may change
 - Frequently: load, throughput, network contention, etc.
 - Slowly: physical data organization, DB schemas, etc.
- Two main solutions
 - Extend the sampling method to consider some new queries as samples and correct the cost model on a regular basis
 - Use adaptive query processing which computes cost during query execution to make optimization decisions

Heterogeneous Query Optimization

- Deals with heterogeneous capabilities of component DBMS
 - One DBMS may support complex SQL queries while another only simple select on one fixed attribute
- Two approaches, depending on the M/W interface level
 - Query-based
 - All wrappers support the same query-based interface (e.g. ODBC or SQL/MED) so they appear homogeneous to the mediator
 - Capabilities not provided by the DBMS must be supported by the wrappers
 - Operator-based
 - Wrappers export capabilities as compositions of operators
 - Specific capabilities are available to mediator
 - More flexibility in defining the level of M/W interface

Query-based Approach

- We can use 2-step query optimization with a heterogeneous cost model
 - ▣ But centralized query optimizers produce left-linear join trees whereas in MDB, we want to push as much processing in the wrappers, i.e. exploit bushy trees
- Solution: convert a left-linear join tree into a bushy tree such that
 - ▣ The initial total cost of the QEP is maintained
 - ▣ The response time is improved
- Algorithm
 - ▣ Iterative improvement of the initial left-linear tree by moving down subtrees while response time is improved

Operator-based Approach

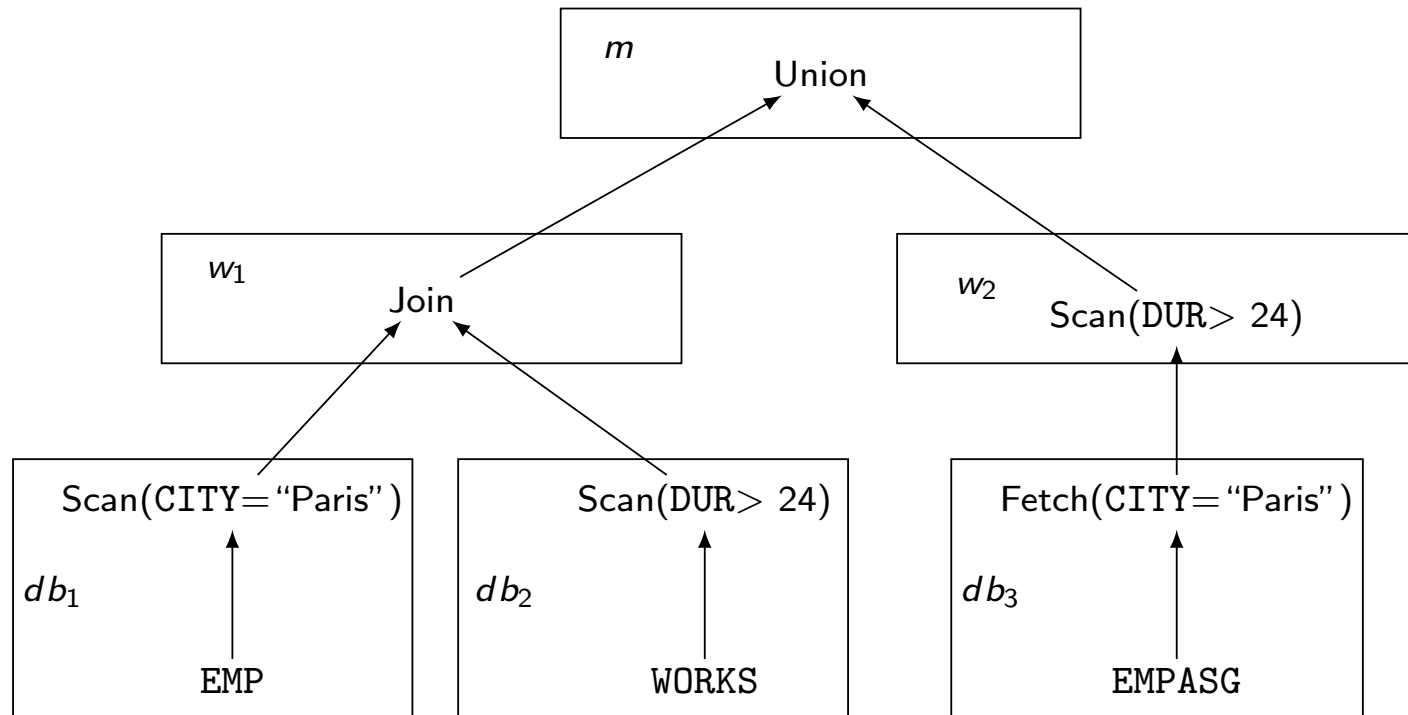
- M/W communication in terms of subplans
- Use of planning functions (Garlic)
 - Extension of cost-based centralized optimizer with new operators
 - Create temporary relations
 - Retrieve locally stored data
 - Push down operators in wrappers
 - accessPlan and joinPlan rules
 - Operator nodes annotated with
 - Location of operands, materialization, etc.

Planning Functions Example

- Consider 3 component databases with 2 wrappers:
 - ❑ $w_1.db_1$: EMP (ENO, ENAME, CITY)
 - ❑ $w_1.db_2$: ASG (ENO, PNAME, DUR)
 - ❑ $w_2.db_3$: EMPASG (ENAME, CITY, PNAME, DUR)
- Planning functions of w_1
 - ❑ $accessPlan(R: rel, A: attlist, P: pred) = scan(R, A, P, db(R))$
 - ❑ $joinPlan(R_1, R_2: rel, A: attlist, P: joinpred) = join(R_1, R_2, A, P)$
 - condition: $db(R_1) \neq db(R_2)$
 - implemented by w_1
- Planning functions of w_2
 - ❑ $accessPlan(R: rel, A: attlist, P: pred) = fetch(city=c)$
 - condition: (city=c) included in P
 - ❑ $accessPlan(R: rel, A: attlist, P: pred) = scan(R, A, P, db(R))$
 - implemented by w_2

Heterogenous QEP

SELECT ENAME, PNAME, DUR
FROM EMPASG
WHERE CITY = "Paris" AND DUR>24

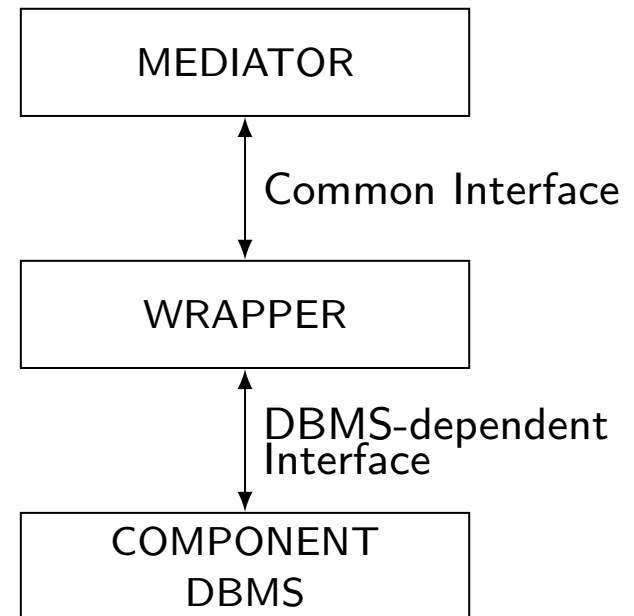


Query Translation and Execution

- Performed by wrappers using the component DBMS
 - Conversion between common interface of mediator and DBMS-dependent interface
 - Query translation from wrapper to DBMS
 - Result format translation from DBMS to wrapper
 - Wrapper has the local schema exported to the mediator (in common interface) and the mapping to the DBMS schema
 - Common interface can be query-based (e.g. ODBC or SQL/MED) or operator-based
- In addition, wrappers can implement operators not supported by the component DBMS, e.g. join

Wrapper Placement

- Depends on the level of autonomy of component DB
- Cooperative DB
 - ❑ May place wrapper at component DBMS site
 - ❑ Efficient wrapper-DBMS com.
- Uncooperative DB
 - ❑ May place wrapper at mediator
 - ❑ Efficient mediator-wrapper com.
- Impact on cost functions



SQL Wrapper for Text Files

- Consider EMP (ENO, ENAME, CITY) stored in a Unix text file in componentDB

- ▣ Each EMP tuple is a line in the file, with attributes separated by “:”

- SQL/MED definition of EMP

```
CREATE FOREIGN TABLE EMP
```

```
    ENO INTEGER, ENAME VARCHAR(30), CITY CHAR(30)
```

```
SERVER componentDB
```

```
OPTIONS (Filename '/usr/EngDB/emp.txt', Delimiter ':')
```

- The query

```
SELECT ENAME FROM EMP
```

Can be translated by the wrapper using a Unix shell command

```
cut -d: -f2/ usr/EngDB/emp
```

Wrapper Management Issues

- Wrappers mostly used for read-only queries
 - Makes query translation and wrapper construction easy
 - DBMS vendors provide standard wrappers
 - ODBC, JDBC, ADO, etc.
- Updating makes wrapper construction harder
 - Problem: heterogeneity of integrity constraints
 - Implicit in some legacy DB
 - Solution: reverse engineering of legacy DB to identify implicit constraints and translate in validation code in the wrapper
- Wrapper maintenance
 - schema mappings can become invalid as a result of changes in component DB schemas
 - Use detection and correction, using mapping maintenance techniques