UNIVERSITAS FRIBURGENSIS

Distributed Databases in a Nutshell

Marc Pouly

Marc.Pouly@unifr.ch

Department of Informatics

University of Fribourg, Switzerland

Priciples of Distributed Database Systems

M. T. Özsu, P. Valduriez

Prentice Hall

February 2005

Why distributed applications at all ?

- Correspond to today's enterprise structures.
- Distributed computer technology
- Reliability
- Divide-and-Conquer, Parallelism
- Economical reasons

Why distributed applications at all ?

- Correspond to today's enterprise structures.
- Distributed computer technology
- Reliability
- Divide-and-Conquer, Parallelism
- Economical reasons

What is distributed ?

- Processing logic
- Functions (special functions delegated to special HW)
- Data
- Control of execution

$\Rightarrow$ All points are unified in the case of distributed databases.

Definitions

Distributed Database System (DDBS)

A collection of multiple, logically interrelated databases distributed over a network.

Distributed Database Management System (DDBMS)

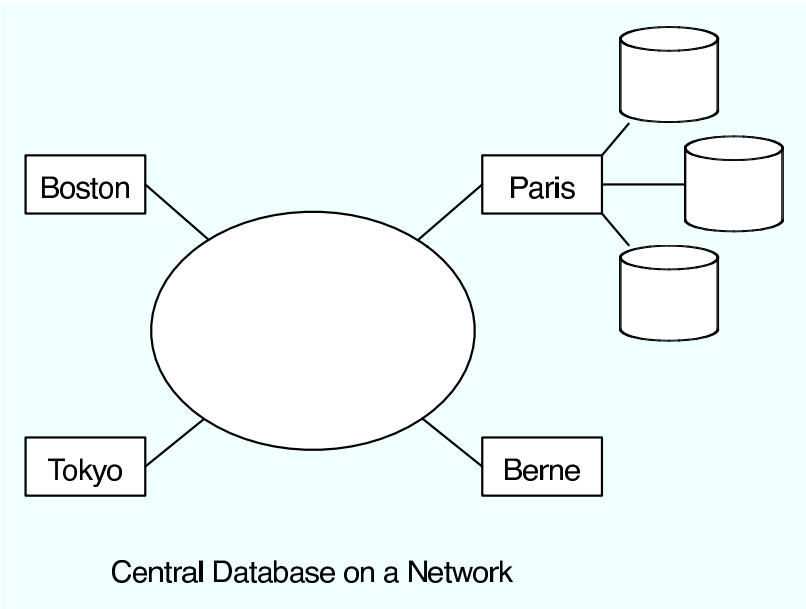
Software system that permits the management of the DDBS and makes the distribution transparent to the user.

Distributed Database System (DDBS)

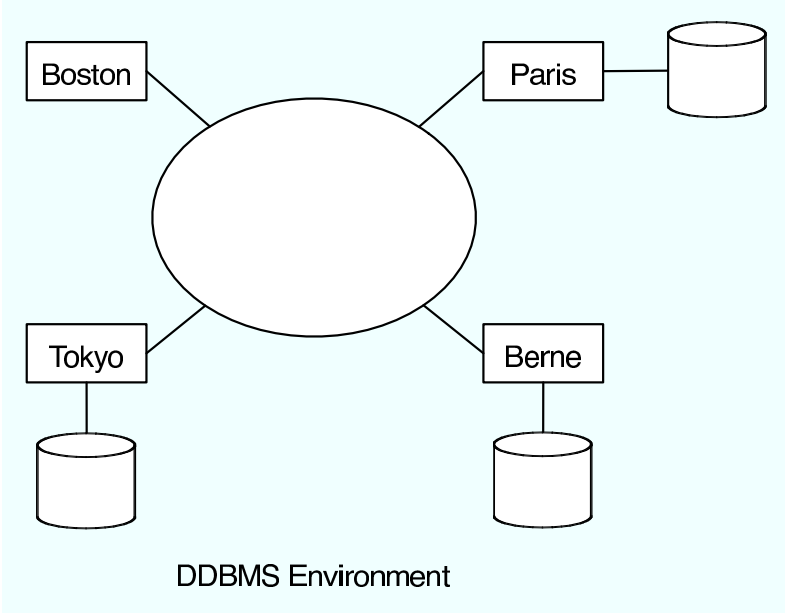
A collection of multiple, logically interrelated databases distributed over a network.

Distributed Database Management System (DDBMS)

Software system that permits the management of the DDBS and makes the distribution transparent to the user.



Central Database on a Network



DDBMS Environment

Outlook I

This talk deals with:

- **Promises and New Problem Areas of a DDBMS**
- **Architectures and Design of DDBMSs**
- **Distribution design issues**
  - Fragmentation
  - Allocation
- **Distributed query processing**
  - Query decomposition
  - Data location
  - Global & local query optimization

What is expected to know:

- Relation algebra, normal forms & relational calculus (SQL)

Outlook II

This talk does NOT deal with:

- **Relational algebra & relational calculus**
- **Semantic data control**
  - View management
  - Data security
  - Integrity control
- **Transaction management**
- **Distribution concurrency control**
- **Distributed DB reliability**
- **Distributed non-relational DB & interoperability**

Promises of DDBMSs I

Key-promises of a DDSMS

1. Improved performance.
2. Transparent management of distributed and replicated data.
3. *Reliability through distributed transactions.*

Promises of DDBMSs I

Key-promises of a DDSMS

1. Improved performance.
2. Transparent management of distributed and replicated data.
3. *Reliability through distributed transactions.*

Performance gain by:

- Proximity of data location.
- Parallelism in query processing:
  - Inter-Query: executes multiple query simultaneously.
  - Intra-Query: breaks-up queries into sub-queries and executes them at different sites (according to data location).

Promises of DDBMSs II

Types of transparency:

- Data transparency
  - Logical data independence
    → immunity of user app. to changes in logical DB structure.
  - Physical data independence
    → hiding details of storage structure.

Promises of DDBMSs II

Types of transparency:

- Data transparency
  - Logical data independence
    $\rightarrow$ immunity of user app. to changes in logical DB structure.
  - Physical data independence
    $\rightarrow$ hiding details of storage structure.

- Network transparency $=$ Distribution transparency
  $\rightarrow$ hiding existence of the network.

Promises of DDBMSs II

Types of transparency:

- Data transparency
  - Logical data independence
    $\rightarrow$ immunity of user app. to changes in logical DB structure.
  - Physical data independence
    $\rightarrow$ hiding details of storage structure.

- Network transparency $=$ Distribution transparency
  $\rightarrow$ hiding existence of the network.

- Replication transparency
  $\rightarrow$ hiding existence & handling of copies.

Types of transparency:

- Data transparency
  - Logical data independence
    → immunity of user app. to changes in logical DB structure.
  - Physical data independence
    → hiding details of storage structure.

- Network transparency = Distribution transparency
  → hiding existence of the network.

- Replication transparency
  → hiding existence & handling of copies.

- Fragmentation transparency
  → hiding existence & handling of fragmented data.

New Problem Areas & Challenges

New problem areas implied by distribution:

- Full replication vs. partial replication vs. no replication
- Degree & strategy of fragmentation
- Data distribution and relocation
- Distributed query processing
- *Distributed directory management (metadata)*
- *Distributed concurrency control*
- *Reliability, crash recovery*
- *Network problems*
- *Heterogeneous DB*
- *OS support problems*

$\Rightarrow$ These areas are not isolated one from another ...

Architectures and design of DDBMSs I

Architectures are classified by:

- Autonomity
- Distribution
- Heterogeneity

Architectures are classified by:

- Autonomity
- Distribution
- Heterogeneity

Autonomity of DDBMSs:

Distribution of control (not data). Degree of which individual DBMSs can operate independently $\rightarrow$ 3 dimensions of autonomity:

- tight integration
- semiautonomous system
- total isolation

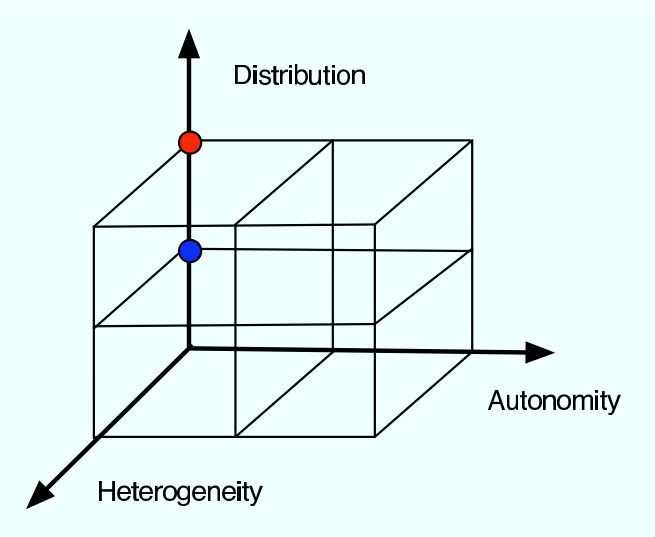
Distribution of DDBMSs:

Distribution of data $\rightarrow$ 3 dimensions of distribution:

- no distribution
- client / server (only servers have DB functionality)
- peer-to-peer (full distribution)

Heterogeneity of DDBMSs:

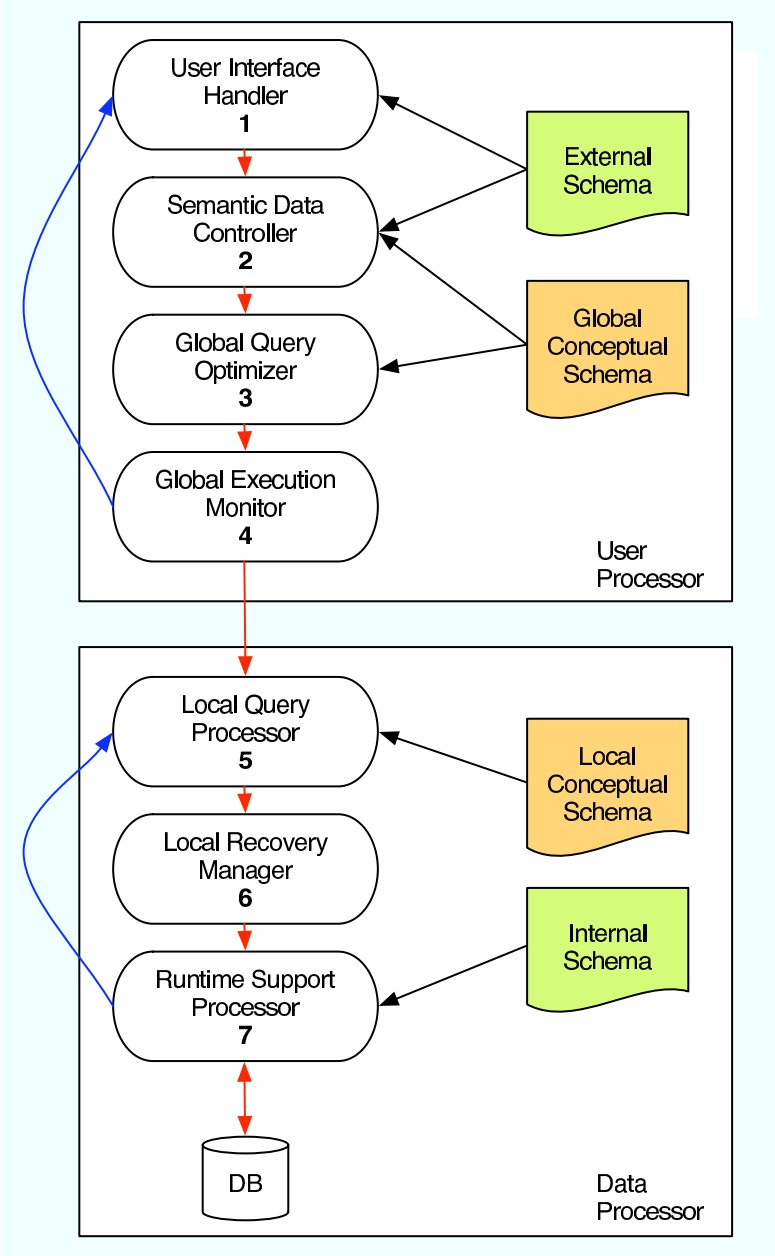
Heterogeneity of data model and query language $\rightarrow$ 2 dimensions ...



peer-to-peer distributed homogeneous DBMS

client-server distributed homogeneous DBMS

ANSI Component Design Model



Component description:

1. Interpretation of user commands and output formatting.
2. Tests integrity constraints, authorization. Solvability of user query.
3. Determines execution strategy (minimize cost function). Translates global query into local ones.
4. Coordinates execution of distributed user queries. Communicates with other 4s.
5. Chooses best data access path.
6. Ensures consistency of DB.
7. Physical access to DB. Buffering.

Fragmentation: Motivation

Relation is not a suitable unit of distribution.

Fragmentation: Motivation

Relation is not a suitable unit of distribution.

- Application views are subsets of relations.
- If multiple, distributed application access the same relation:
  - **a)** Relation is not replicated.
    $\Rightarrow$ High volume of remote data access.
  - **b)** Relation is replicated.
    $\Rightarrow$ High storage use & update problems.
- Allow multiple transactions and parallel query execution.
  $\Rightarrow$ Increases level of concurrency.

Fragmentation: Motivation

Relation is not a suitable unit of distribution.

- Application views are subsets of relations.
- If multiple, distributed application access the same relation:
  - **a)** Relation is not replicated.
    $\Rightarrow$ High volume of remote data access.
  - **b)** Relation is replicated.
    $\Rightarrow$ High storage use & update problems.
- Allow multiple transactions and parallel query execution.
  $\Rightarrow$ Increases level of concurrency.

Alternatives of fragmentation:

- horizontal fragmentation & vertical fragmentation
- hybrid fragmentation

Fragmentation: A first look

Key	Name	Population	Life
$c1$	France	58973000	78.5
$c2$	Germany	82037000	77.5
$c3$	Switzerland	7124000	79.5
$c4$	Italy	57613000	78.4

Key	Name	Population	Life
$c1$	France	58973000	78.5
$c2$	Germany	82037000	77.5

Key	Name	Population	Life
$c3$	Switzerland	7124000	79.5
$c4$	Italy	57613000	78.4

Key	Name
$c1$	France
$c2$	Germany
$c3$	Switzerland
$c4$	Italy

Key	Population	Life
$c1$	58973000	78.5
$c2$	82037000	77.5
$c3$	7124000	79.5
$c4$	57613000	78.4

Fragmentation: A first look

Key	Name	Population	Life
$c1$	France	58973000	78.5
$c2$	Germany	82037000	77.5
$c3$	Switzerland	7124000	79.5
$c4$	Italy	57613000	78.4

Key	Name	Population	Life
$c1$	France	58973000	78.5
$c2$	Germany	82037000	77.5

$\cup$

Key	Name	Population	Life
$c3$	Switzerland	7124000	79.5
$c4$	Italy	57613000	78.4

Key	Name
$c1$	France
$c2$	Germany
$c3$	Switzerland
$c4$	Italy

$\bowtie_{Key}$

Key	Population	Life
$c1$	58973000	78.5
$c2$	82037000	77.5
$c3$	7124000	79.5
$c4$	57613000	78.4

Disadvantages of fragmentation:

- Performance problem if an application uses multiple non-mutual exclusive fragments.
- *Integrity control over multiple sites.*

Fragmentation II

Disadvantages of fragmentation:

- Performance problem if an application uses multiple non-mutual exclusive fragments.
- *Integrity control over multiple sites.*

Degree of fragmentation:

- no fragmentation vs. full fragmentation (atomic tuples or columns).
- $\Rightarrow$ compromise with respect to some parameters ...

Disadvantages of fragmentation:

- Performance problem if an application uses multiple non-mutual exclusive fragments.
- *Integrity control over multiple sites.*

Degree of fragmentation:

- no fragmentation vs. full fragmentation (atomic tuples or columns).
- $\Rightarrow$ compromise with respect to some parameters ...

Correctness rules of fragmentation:

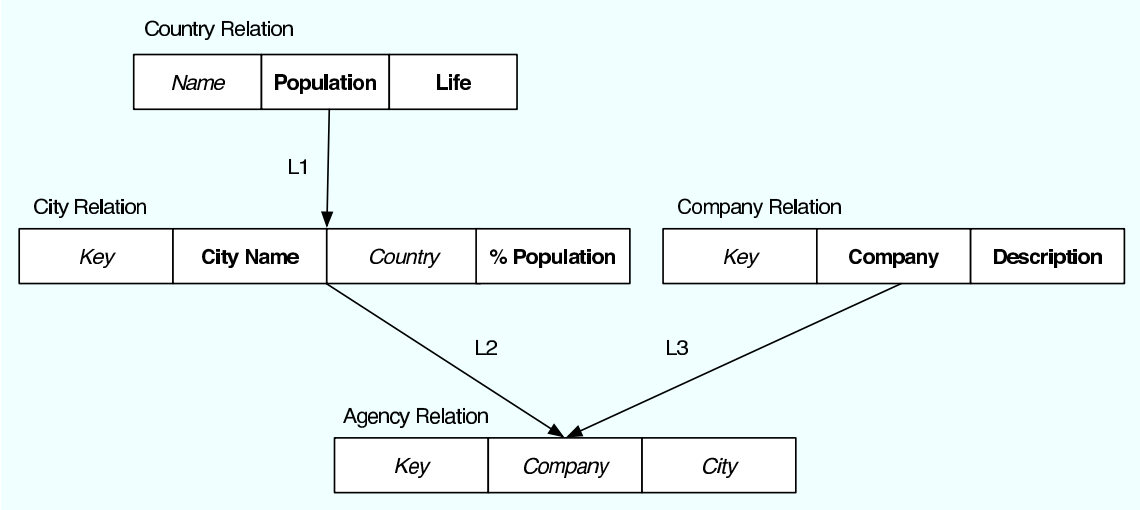
Averts semantic changes during fragmentation.

- Completeness (lost-less decomposition)
- Reconstruction: $R = \bigcup R_i$ vs. $R = \bowtie_{Key} R_i$
- Disjointness (for vertical fragmentation on non-key attributes).

Horizontal Fragmentation I

Fragmentation requires database informations (metadata).

There are two types of relations: OWNER and MEMBER.



- L_i are called LINKS.
- Ex: $owner(L_1) = Country\ Relation$, $member(L_1) = City\ Relation$
- Important are those $owner$ which are not $member$ of any link.

2 Ways of Horizontal Fragmentation (HF)

- *primary HF:* for $owner$ - Relations
- *derived HF:* for $member$ - Relations

2 Ways of Horizontal Fragmentation (HF)

- *primary HF:* for $owner$ - Relations
- *derived HF:* for $member$ - Relations

An important note:

- Links are *equi-joins* $\sim$ Joins over attributes with only "$=$" predicates.

Name	P $\cdot 10^3$
F	58973
DE	82037
CH	7124
I	57613

$\bowtie$

City	Country	% P
Paris	F	3.6
Berlin	DE	4.1
Rome	I	4.6

$=$

Country	City	P $\cdot 10^3$	% P
F	Paris	58973	3.6
DE	Berlin	82037	4.1
I	Rome	57613	4.6

$$Country \bowtie_{Country.Name = City.Country} City$$

Primary Horizontal Fragmentation I

Definition Simple Predicate:

For a relation $R(A_1, \ldots, A_n)$, $d(A_i) = d_i$, a simple predicat p_j has the form $p_j : A_i \ \theta \ value$, where $\theta \in \{=, \neq, >, <, \leq, \geq\}$ and $value \in d_i$.

Primary Horizontal Fragmentation I

Definition Simple Predicate:

For a relation $R(A_1, \ldots, A_n)$, $d(A_i) = d_i$, a simple predicat p_j has the form $p_j : A_i \ \theta \ value$, where $\theta \in \{=, \neq, >, <, \leq, \geq\}$ and $value \in d_i$.

Definition Minterm Predicate:

CNF of simple predicates.

Primary Horizontal Fragmentation I

Definition Simple Predicate:

For a relation $R(A_1, \ldots, A_n)$, $d(A_i) = d_i$, a simple predicat p_j has the form $p_j : A_i \; \theta \; value$, where $\theta \in \{=, \neq, >, <, \leq, \geq\}$ and $value \in d_i$.

Definition Minterm Predicate:

CNF of simple predicates.

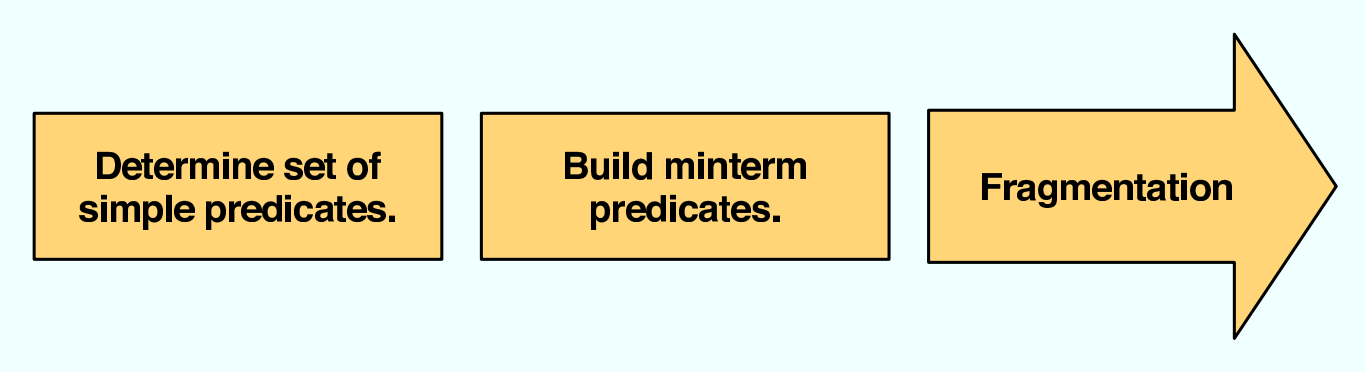
Horizontal fragmentation of R consists of all tuples in R satisfying a given minterm predicate.

$$R_i = \sigma_{m_i}(R)$$

where m_i is the minterm predicate to obtain R_i.

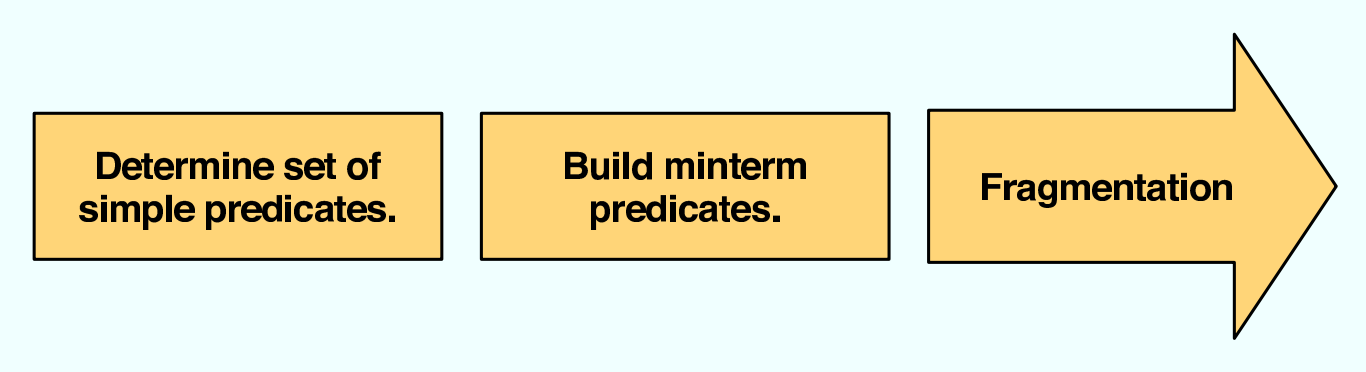
$\Rightarrow$ Given a set of minterm predicates, there are as many horizontal fragments as minterms in this set $\Rightarrow$ minterm fragments.

Primary Horizontal Fragmentation II



2 important aspects of simple predicates: completeness & minimality

Primary Horizontal Fragmentation II



2 important aspects of simple predicates: completeness & minimality

Completeness of simple predicates:

A set of simple predicates Pr is said to be complete if and only if there is an equal probability of access by every application to any tuple belonging to any minterm fragment defined by Pr.

Primary Horizontal Fragmentation III

Key	Name	Population	Life
$c1$	France	58973000	78.5
$c2$	Germany	82037000	77.5

Key	Name	Population	Life
$c3$	Switzerland	7124000	79.5
$c4$	Italy	57613000	78.4

$$R = \sigma_{Population \geq 58'000'000}(R) \cup \sigma_{Population < 58'000'000}(R)$$

- One application, access by `Population` $\gtreqless 58 \cdot 10^6 \Rightarrow$ complete !
- One application, access by `Life` $\gtreqless 79 \Rightarrow$ not complete !
- Solution: New fragmentation according to `Life` $\gtreqless 79$.

Primary Horizontal Fragmentation III

Key	Name	Population	Life
$c1$	France	58973000	78.5
$c2$	Germany	82037000	77.5

$\cup$

Key	Name	Population	Life
$c3$	Switzerland	7124000	79.5
$c4$	Italy	57613000	78.4

$$R = \sigma_{Population \geq 58'000'000}(R) \cup \sigma_{Population < 58'000'000}(R)$$

- One application, access by `Population` $\gtreqless 58 \cdot 10^6 \Rightarrow$ complete !
- One application, access by `Life` $\gtreqless 79 \Rightarrow$ not complete !
- Solution: New fragmentation according to `Life` $\gtreqless 79$.

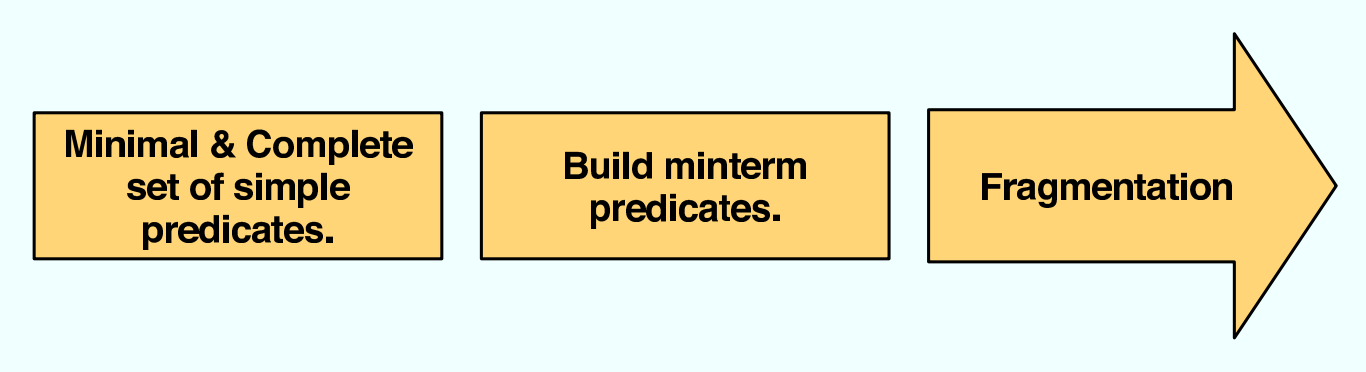
Minimality of simple predicates:

For all fragments R_i, R_j, there is at least one application accessing R_i and R_j differently.

Primary Horizontal Fragmentation IV

Consequence of completeness & minimality:

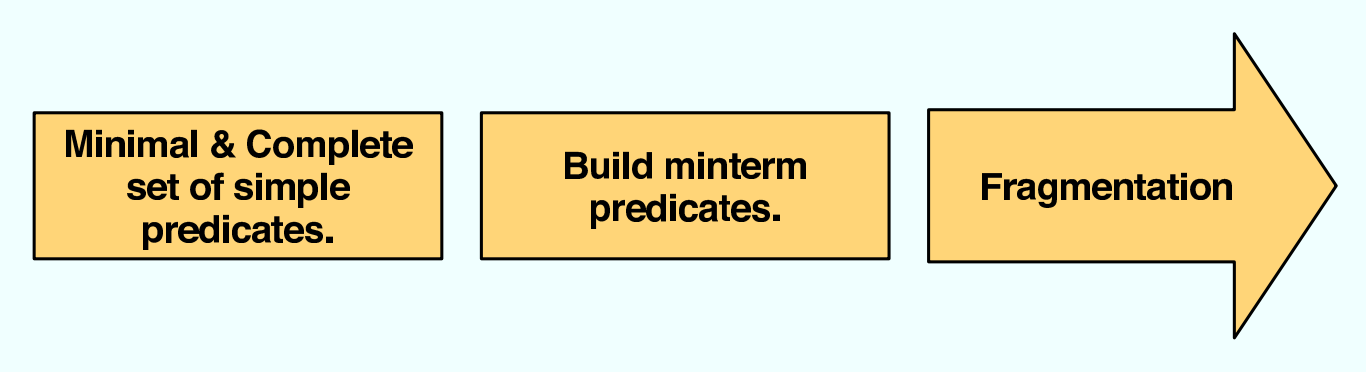
- completeness $\Rightarrow$ fragments are logically uniform.
- minimality $\Rightarrow$ simple predicates do not produce unused fragments.



Primary Horizontal Fragmentation IV

Consequence of completeness & minimality:

- completeness $\Rightarrow$ fragments are logically uniform.
- minimality $\Rightarrow$ simple predicates do not produce unused fragments.



Determining the minterms is trivial ! But this set may be quite large ...

Fortunately, the set of minterms can be reduced.

This elimination is performed by identifying those minterms that might by contradictory to a set of implications I.

Primary Horizontal Fragmentation V

Example: Attribute A with domain $d_A = \{v_1, v_2\}$. Let $P = \{p_1, p_2\}$ be a set of simple predicates,

$p_1 : \quad A = v_1$

$p_2 : \quad A = v_2$

We build the set of possible minterms:

$m_1 : (A = v_1) \wedge (A = v_2) \qquad m_3 : \neg(A = v_1) \wedge (A = v_2)$

$m_2 : (A = v_1) \wedge \neg(A = v_2) \qquad m_4 : \neg(A = v_1) \wedge \neg(A = v_2)$

Clearly, m_1 and m_2 can be eliminated; they contradict to I.

The implication $i \in I$ eliminating m_1 is:

$i : (A = v_1) \Rightarrow \neg(A = v_2)$

Derived Horizontal Fragmentation I

- $owner$-relations are fragmented according to the user queries.
- $member$-relations depend on their $owner(s)$.

Let L be a link. $owner(L) = S$ and $member(L) = R$.
We want to fragment R.

$$R_i = R \ltimes S_i$$

Derived Horizontal Fragmentation I

- $owner$-relations are fragmented according to the user queries.
- $member$-relations depend on their $owner(s)$.

Let L be a link. $owner(L) = S$ and $member(L) = R$.
We want to fragment R.

$$R_i = R \ltimes S_i$$

This is a semi-join. It results in the subset of tuples in R that participate in the join of R and S_i. Formally,

$$R_i = R \ltimes S_i = \pi_A(R \bowtie S_i)$$

where A is the domain of R.

$\Rightarrow$ Given the primary fragmentation, derived fragmentation is simple !

Derived Horizontal Fragmentation II

Often, a $member$ has multiple $owners \Rightarrow$ there are multiple possibilities for derived fragmentation (according to each of the $owners$).

Choose the fragmentation

1. with better join-characteristics.
2. that is used in more applications.

$\Rightarrow$ Optimization problem.

Derived Horizontal Fragmentation II

Often, a $member$ has multiple $owners \Rightarrow$ there are multiple possibilities for derived fragmentation (according to each of the $owners$).

Choose the fragmentation

1. with better join-characteristics.
2. that is used in more applications.

$\Rightarrow$ Optimization problem.

- Criteria 1 has 2 motivations:
  - Join on fragments is faster than on main relation itself.
  - Fragments allow parallel query execution.
- Criteria 2 facilitates the access of heavy users such that their total impact on the system is minimized.

Vertical Fragmentation

Produces fragments $R_1, R_2, \ldots, R_n$ of R, each one containing a subset of R's attributes and it's primary key.

Vertical Fragmentation

Produces fragments $R_1, R_2, \ldots, R_n$ of R, each one containing a subset of R's attributes and it's primary key.

More complicated than horizontal fragmentation:

- HF: n simple predicates $\Rightarrow 2^n$ possible minterms $=$ #fragments. A lot of them can be eliminated.
- VF: m non-primary key attributes $\Rightarrow B(m) \cong m^m$ fragments (m-th Bell number).

Vertical Fragmentation

Produces fragments $R_1, R_2, \ldots, R_n$ of R, each one containing a subset of R's attributes and it's primary key.

More complicated than horizontal fragmentation:

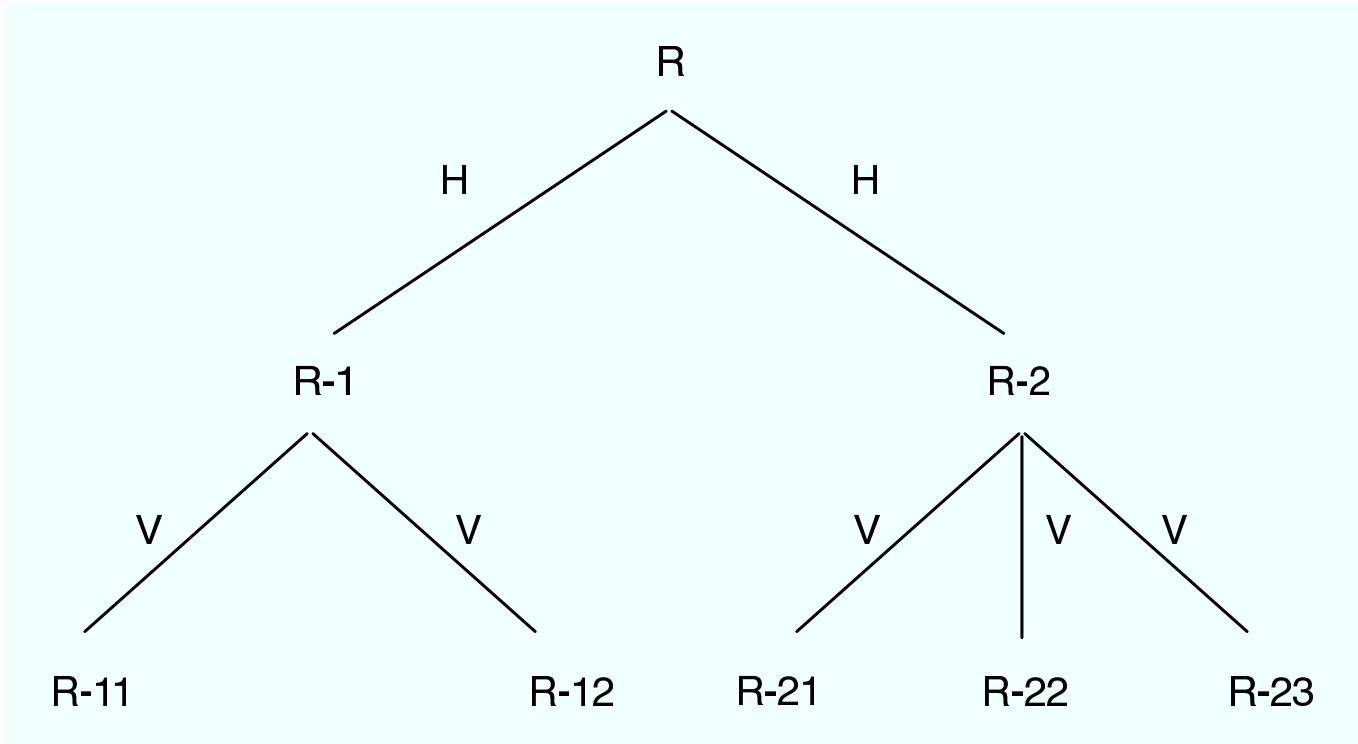
- HF: n simple predicates $\Rightarrow 2^n$ possible minterms $=$ #fragments. A lot of them can be eliminated.
- VF: m non-primary key attributes $\Rightarrow B(m) \cong m^m$ fragments (m-th Bell number).

No optimal solution $\Rightarrow$ heuristics.

- **Grouping:** Starts by assigning each attribute to one fragment and at each step, joins some fragments.
- **Splitting:** Starts with a relations and splits according to the access behavior of applications for each attribute.

Hybrid Fragmentation

Tree-structured partitioning ...



Level of nesting can be large - but in practice, levels do rarely exceed 2, because normalized relations are already small.

The Allocation Problem

Set of fragments $F = \{F_1, F_2, \ldots, F_n\}$, network consisting of sites $S = \{S_1, S_2, \ldots, S_m\}$ and applications $Q = \{q_1, q_2, \ldots, q_q\}$.

$\Rightarrow$ Optimal distribution of F to S regarding Q.

The Allocation Problem

Set of fragments $F = \{F_1, F_2, \ldots, F_n\}$, network consisting of sites $S = \{S_1, S_2, \ldots, S_m\}$ and applications $Q = \{q_1, q_2, \ldots, q_q\}$.

$\Rightarrow$ Optimal distribution of F to S regarding Q.

Optimality is defined with respect to:

- **Minimal costs** of
  - storing F_i at S_j
  - querying F_i at S_j
  - updating F_j
  - data communication
- **Performance**
  - minimize response time of S_i
  - maximize system throughput

$\Rightarrow$ Problem is NP-Complete

The Allocation Problem

Set of fragments $F = \{F_1, F_2, \ldots, F_n\}$, network consisting of sites $S = \{S_1, S_2, \ldots, S_m\}$ and applications $Q = \{q_1, q_2, \ldots, q_q\}$.

$\Rightarrow$ Optimal distribution of F to S regarding Q.

Optimality is defined with respect to:

- **Minimal costs** of
  - storing F_i at S_j
  - querying F_i at S_j
  - updating F_j
  - data communication
- **Performance**
  - minimize response time of S_i
  - maximize system throughput

$\Rightarrow$ Problem is NP-Complete

There are no general models which take a set of fragments as input and produce a near optimal allocation. All existing models make some assumptions to simplify the problem and are applicable only to certain specific formulations.

Query Processing Overview

- Transformation high-level query $\rightarrow$ low-level query.
- High-level queries typically in relation calculus (SQL-Expression).
- Low-level queries in relation algebra + communication primitives.

Query Processing Overview

- Transformation high-level query $\rightarrow$ low-level query.
- High-level queries typically in relation calculus (SQL-Expression).
- Low-level queries in relation algebra + communication primitives.

In centralized databases ...

Find best relational algebra query among all equivalent transformations $\Rightarrow$ computationally intractable $\Rightarrow$ Heuristics

In distributed databases ...

Additionally select the best site to process data and minimize network traffic $\Rightarrow$ Increases solution space and problem complexity.

Query Processing Example I

2 Relations:

`EMP(ENO, ENAME, TITLE)` $\longrightarrow$ 400 tuples

`ASG(ENO, PNO, RESP, DUR)` $\longrightarrow$ 1000 tuples

Query Processing Example I

2 Relations:

`EMP(ENO, ENAME, TITLE)` $\rightarrow$ 400 tuples

`ASG(ENO, PNO, RESP, DUR)` $\rightarrow$ 1000 tuples

High-level query:

Find the names of employees who are managing a project.

Query Processing Example I

2 Relations:

`EMP(ENO, ENAME, TITLE)` $\rightarrow$ 400 tuples

`ASG(ENO, PNO, RESP, DUR)` $\rightarrow$ 1000 tuples

High-level query:

Find the names of employees who are managing a project.

In SQL (Relational Calculus):

```
SELECT    ENAME
FROM      EMP, ASG
WHERE     EMP.ENO = ASG.ENO AND RESP = "Manager"
```

Query Processing Example I

2 Relations:

`EMP(ENO, ENAME, TITLE)` $\rightarrow$ 400 tuples

`ASG(ENO, PNO, RESP, DUR)` $\rightarrow$ 1000 tuples

High-level query:

Find the names of employees who are managing a project.

In SQL (Relational Calculus):

```
SELECT    ENAME
FROM      EMP, ASG
WHERE     EMP.ENO = ASG.ENO AND RESP = "Manager"
```

2 equivalent low-level queries:

1. $\prod_{\texttt{ENAME}}(\sigma_{\texttt{RESP} = ''Manager'' \wedge \texttt{EMP.ENO = ASG.ENO}}(\texttt{EMP} \times \texttt{ASG})$
2. $\prod_{\texttt{ENAME}}(\texttt{EMP} \bowtie_{\texttt{ENO}} (\sigma_{\texttt{RESP} = ''Manager''}(\texttt{ASG})))$

Query Processing Example II

Clearly, the second transformation is better (avoids Carthesian product).

We assume that both relations are horizontally fragmented.

$\texttt{EMP}_1 = \sigma_{\texttt{ENO} \leq ''E3''}(\texttt{EMP})$ stored at site 1.

$\texttt{EMP}_2 = \sigma_{\texttt{ENO} > ''E3''}(\texttt{EMP})$ stored at site 2.

$\texttt{ASG}_1 = \sigma_{\texttt{ENO} \leq ''E3''}(\texttt{ASG})$ stored at site 3.

$\texttt{ASG}_2 = \sigma_{\texttt{ENO} > ''E3''}(\texttt{ASG})$ stored at site 4.

The result of the query is expected at site 5.

Query Processing Example II

Clearly, the second transformation is better (avoids Carthesian product).

We assume that both relations are horizontally fragmented.

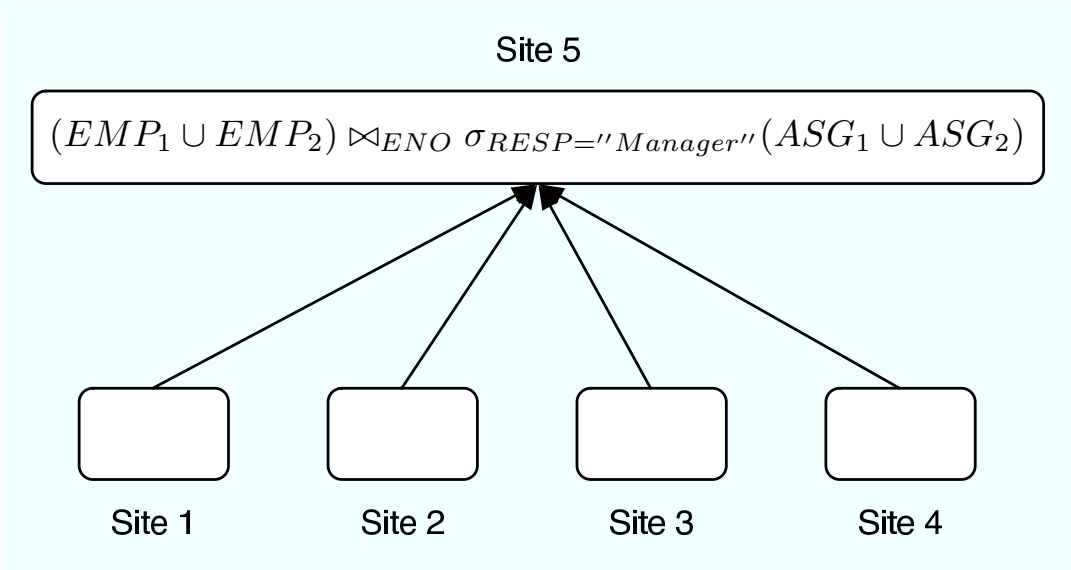
$$\begin{array}{lcll} \texttt{EMP}_1 & = & \sigma_{\texttt{ENO} \leq ''E3''}(\texttt{EMP}) & \text{stored at site } 1. \\ \texttt{EMP}_2 & = & \sigma_{\texttt{ENO} > ''E3''}(\texttt{EMP}) & \text{stored at site } 2. \\ \texttt{ASG}_1 & = & \sigma_{\texttt{ENO} \leq ''E3''}(\texttt{ASG}) & \text{stored at site } 3. \\ \texttt{ASG}_2 & = & \sigma_{\texttt{ENO} > ''E3''}(\texttt{ASG}) & \text{stored at site } 4. \end{array}$$

The result of the query is expected at site 5.

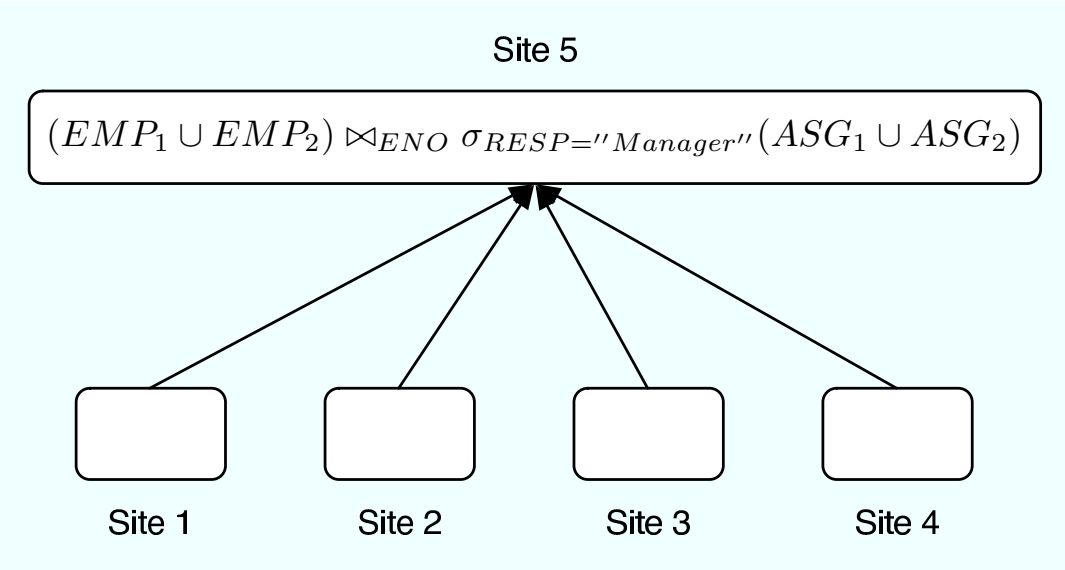
2 equivalent distributed execution strategies are considered. The costs for each strategy is computed using a total-cost function:

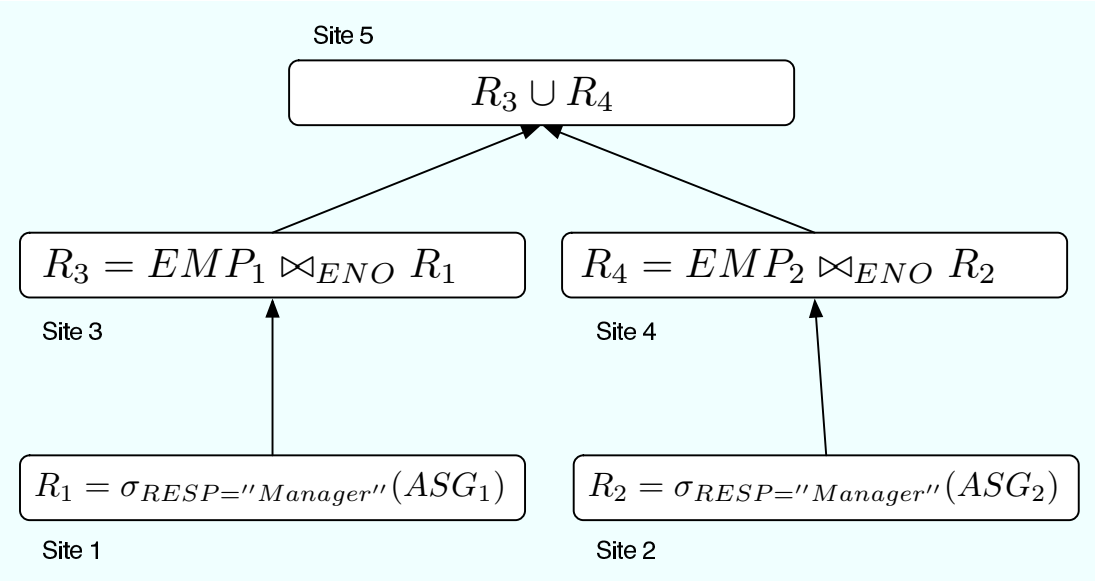
- 1 tuple acces = 1 unit
- 1 tuple transfer = 10 units

Query Processing Example III

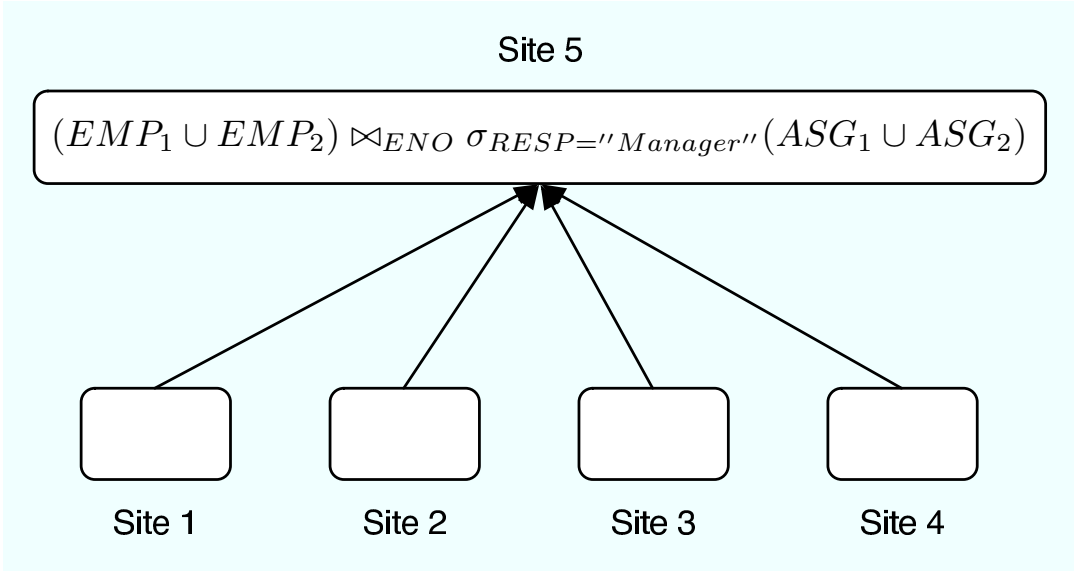


Query Processing Example III

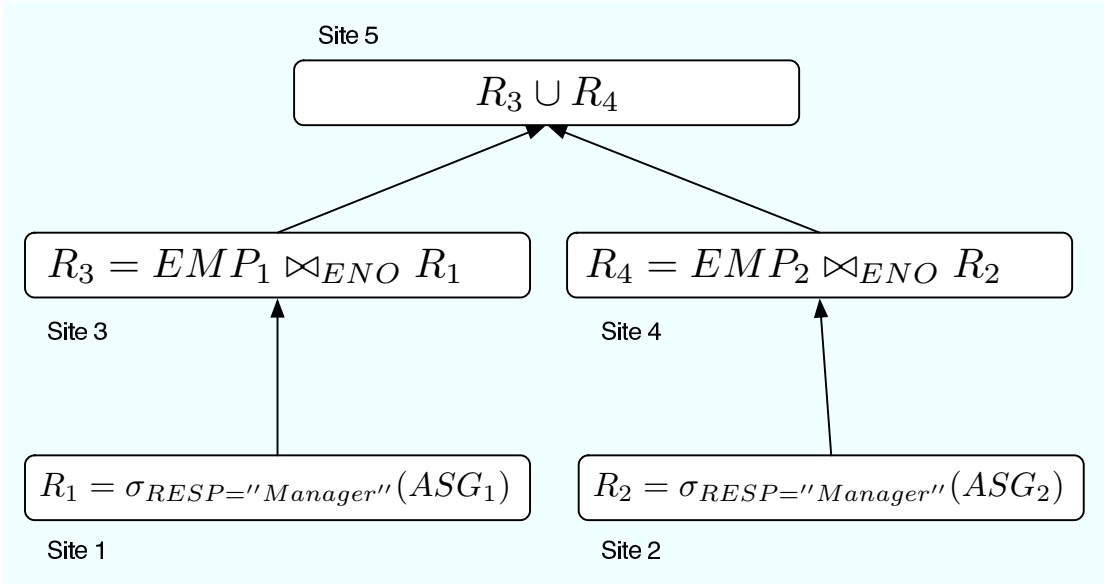




Query Processing Example III

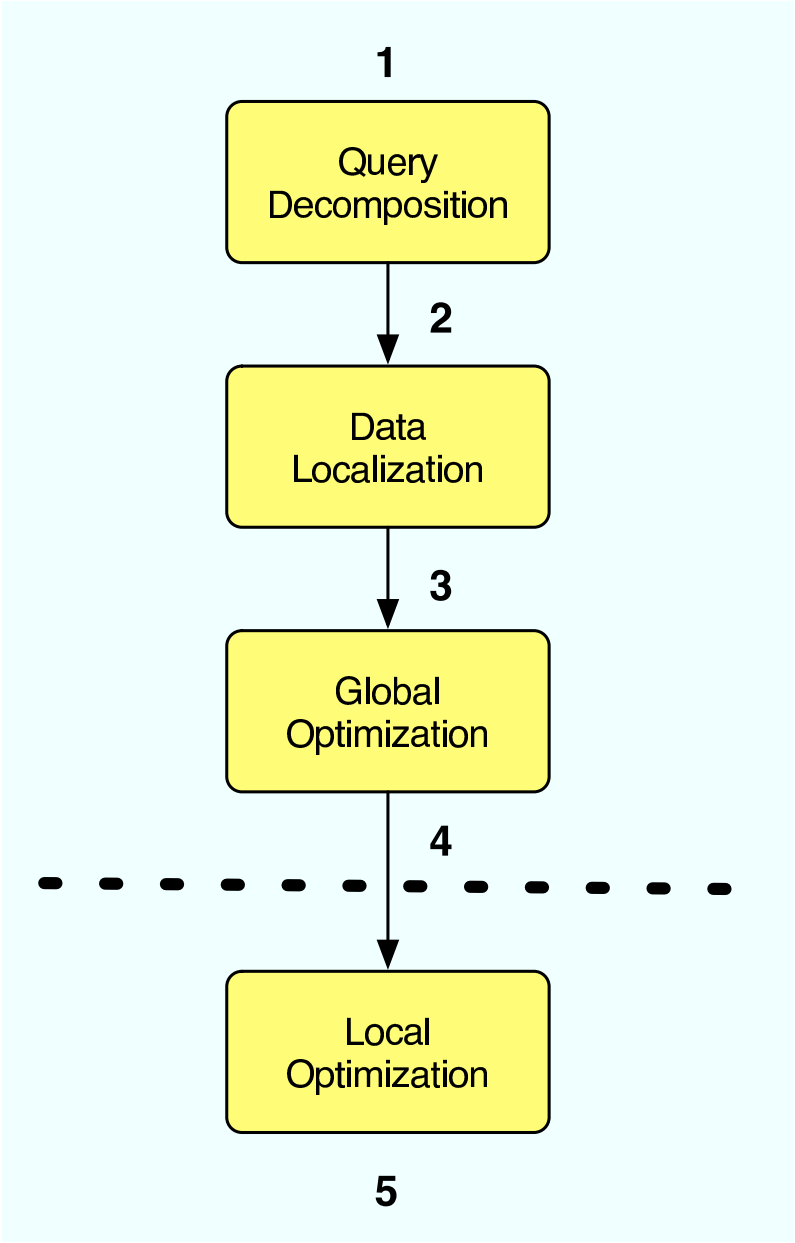


Costs: 23000 units



Costs: 460 units

Query Processing Architecture

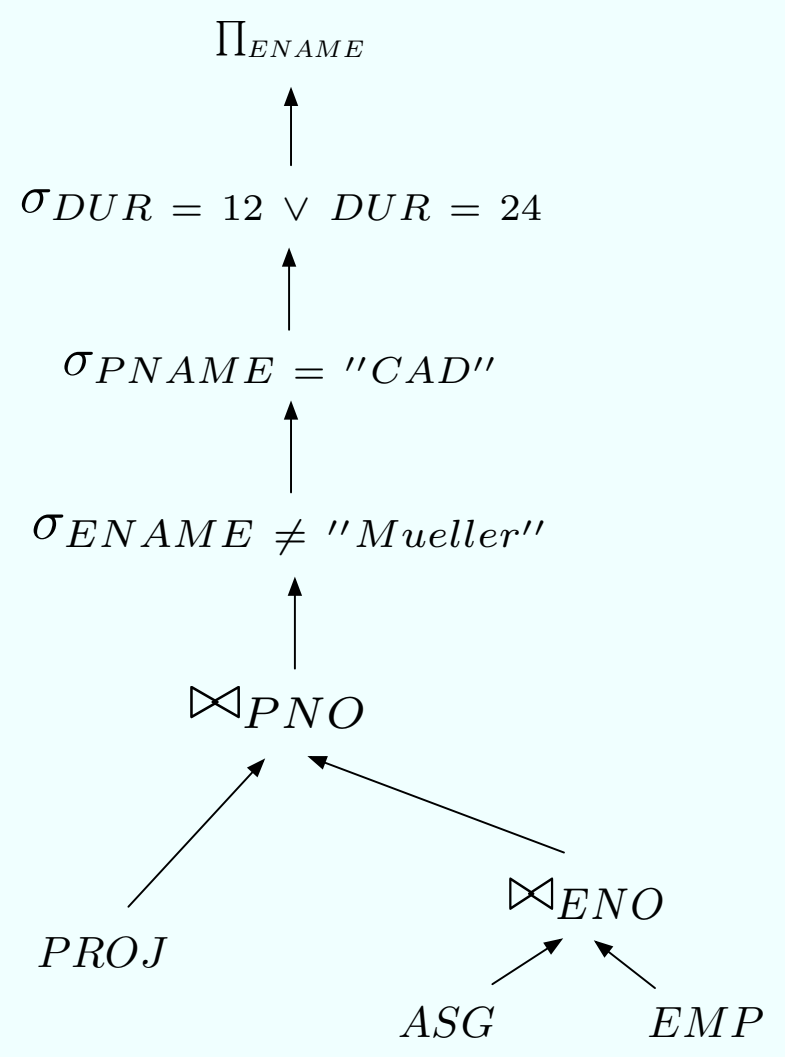


1. Calculus query on distributed relations.
2. Algebraic query on distributed relations.
3. Algebraic query on distributed fragments.
4. Optimized fragment query with communication primitives.
5. Optimized local query.

Query Decomposition I

- Validates sematics (ex. types of operations, existance of relations, ...)
- Transforms `WHERE`-clause to CNF.
- Eliminates redundancy in `WHERE`-clause (logical idempotency rules).
- Construction of operator tree.
- Optimizes operator tree by applying transformation rules.
- Rewrites operator tree in relation algebra.

Query Decomposition II



Optimization rules:

1. Separate unary operations.
2. Regroup unary operations on same relation.
3. Commute binary with unary operations $\Rightarrow$ Unary operations are pushed down.
4. Order unary operations, perform first selections, then projections.
5. ...

Data Localization

- Adds distribution to query.
- Transforms algebraic query on global relations to algebraic query on physical fragments.
- Applies fragment reconstruction rules ($\cup$ or $\bowtie$) to query.
- Eliminates selections / projections if they contradict fragment predicats.

Global Optimization I

After data localization, the global query can be executed by adding communication primitives in a systematic way. But the ordering of these primitives again creates many equivalent strategies.

Global Optimization I

After data localization, the global query can be executed by adding communication primitives in a systematic way. But the ordering of these primitives again creates many equivalent strategies.

The problem of finding a good strategy is so difficult that in most cases, all effort is concentrated rather on avoiding bad strategies that on finding a good one.

After data localization, the global query can be executed by adding communication primitives in a systematic way. But the ordering of these primitives again creates many equivalent strategies.

The problem of finding a good strategy is so difficult that in most cases, all effort is concentrated rather on avoiding bad strategies that on finding a good one.

Again, the heuristics are all based on cost-functions which use database statistics. Because this optimization step concerns communication primitives, communication cost is a central point.

Expressing the costs to transmit relation R from site S_1 to site S_2:

$$size(R) = card(R) * length(R)$$

where $length(R)$ is the length (in bytes) of a single tuple.

Global Optimization II

Optimization in 3 steps:

1. Build search space (all possible execution strategies represented as operator trees) $\sim O(n!)$ for n relations.
2. Apply cost function to each QEP = Query Execution Plan
3. Choose the best !

Global Optimization II

Optimization in 3 steps:

1. Build search space (all possible execution strategies represented as operator trees) $\sim O(n!)$ for n relations.
2. Apply cost function to each QEP = Query Execution Plan
3. Choose the best !

Carthesian products and joins have most influence on total costs. $\Rightarrow$ Optimization is concentrated on join trees = operator trees whose nodes are joins or Carthesian products.

Global Optimization II

Optimization in 3 steps:

1. Build search space (all possible execution strategies represented as operator trees) $\sim O(n!)$ for n relations.
2. Apply cost function to each QEP = Query Execution Plan
3. Choose the best !

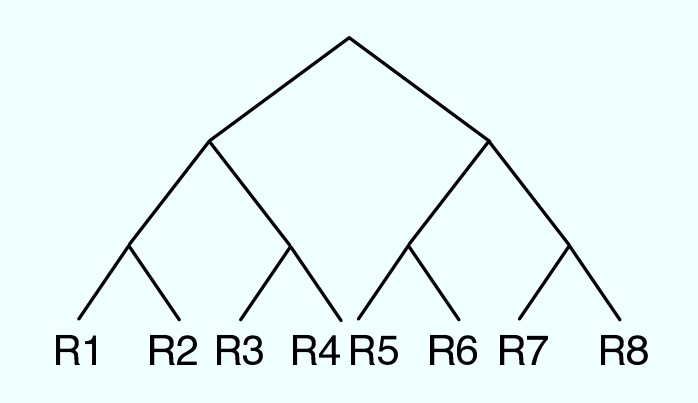
Carthesian products and joins have most influence on total costs. $\Rightarrow$ Optimization is concentrated on join trees = operator trees whose nodes are joins or Carthesian products.

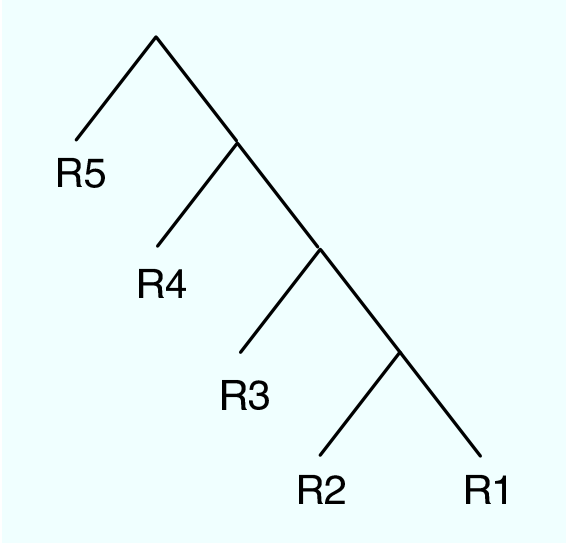
Building the search space:

- *dynamic programming:* builds all possible plans. Partial plans which are not likely to lead to a good plan are pruned.
- *randomized strategies:* apply dynamic prog. to obtain ”not-so-bad-solution” and investigate neighboring trees (by exchanging relations).

Global Optimization III

The join tree shape is a useful indicator: linear trees $\Rightarrow$ no parallelism, bushy trees $\Rightarrow$ full parallelism.





Discussing the cost functions goes beyond the scope of this presentation (case studies)...

Local Optimization

After global optimization, each sub-query is delegated to the appropriate site (due to the communication primitives). Clearly, all sites can themselves optimize their local queries according to their additional knowledge about their data.

This is local optimization ...

Questions ?