Distributed DB design

Data Management for Big Data

2019-2020 (spring semester)

Dario Della Monica

These slides are a modified version of the slides provided with the book

Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

The original version of the slides is available at: extras.springer.com

Outline (distributed DB)

- Introduction (Ch. 1) *
- Distributed Database Design (Ch. 3) *
  - ➡ Fragmentation
  - ➡ Data distribution (allocation)
- Distributed Query Processing (Ch. 6-8) *
- Distributed Transaction Management (Ch. 10-12) *

* Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

Outline (today)

- Distributed DB design (Ch. 3) *
  - ➡ Introduction
  - ➡ Top-down (vs. bottom-up) design
  - ➡ Distribution design issues
    - ✦ Fragmentation
    - ✦ Allocation
  - ➡ Fragmentation
    - ✦ Horizontal Fragmentation (HF)
      - ✓ Primary Horizontal Fragmentation (PHF)
      - ✓ Derived Horizontal Fragmentation (DHF)
    - ✦ Vertical Fragmentation (VF)
    - ✦ Hybrid Fragmentation (HyF)
  - ➡ Allocation
  - ➡ Data directory

* Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

Design Problem

- In the general setting:

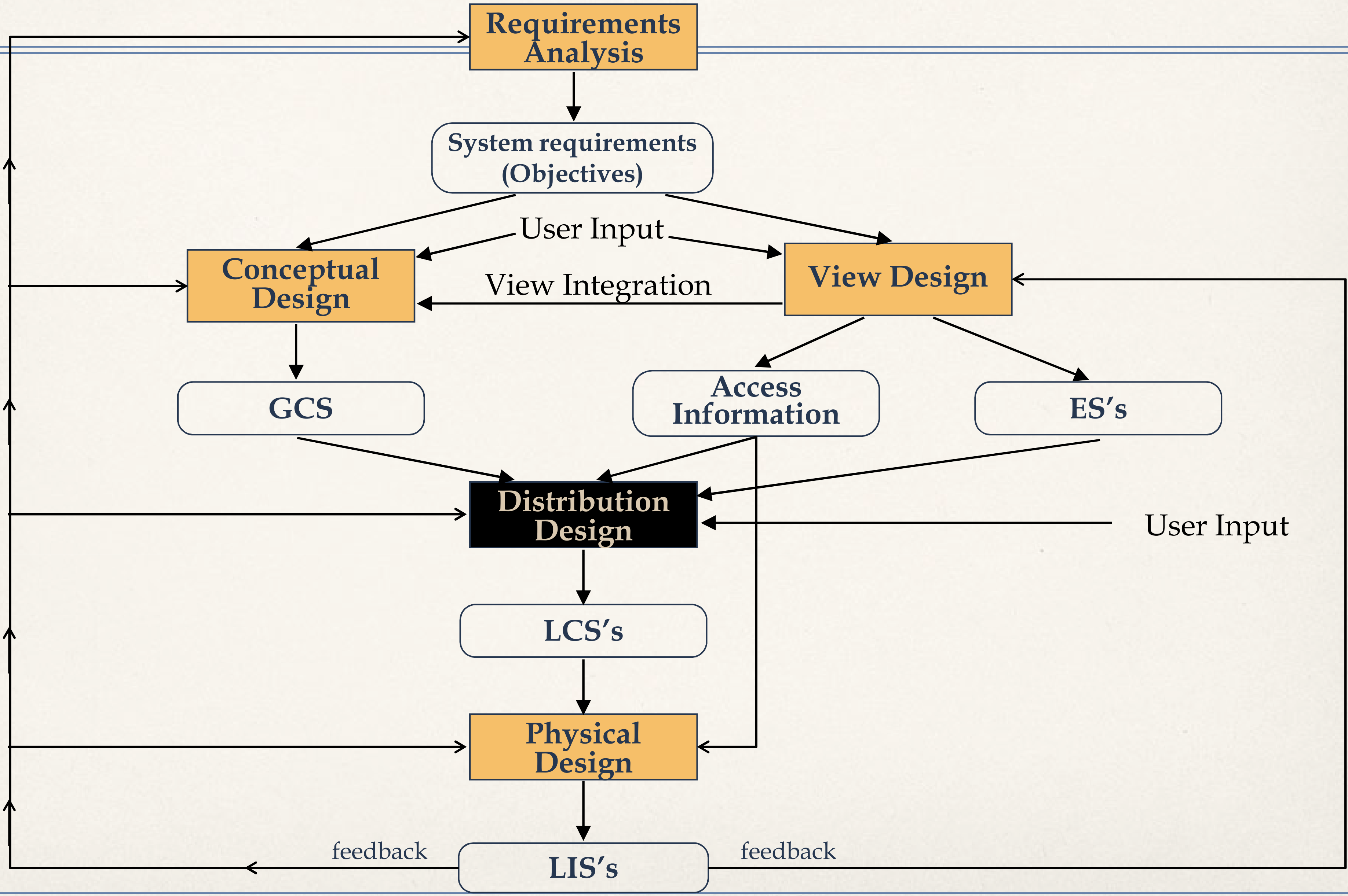
  Making decisions about the placement of data and programs (control) across the sites of a computer network as well as possibly designing the network itself

- In Distributed DBMS, the placement of applications entails
  - ➡ placement of the distributed DBMS software; and
  - ➡ placement of the applications that run on the database

Distribution Design

- Top-down
  - ➡ mostly in designing systems from scratch
  - ➡ mostly in homogeneous systems
  - ➡ applies to fully distributed DBMS (a logical view of the whole DB exists)
- Bottom-up
  - ➡ when the databases already exist at a number of sites
  - ➡ applies to MDBS (we will not treat them)

Top-Down Design



Distribution Design Issues

Distribution design activity boils down to *fragmentation* and *allocation*

❶ Why fragment at all?	[reasons for fragmentation]
❷ How to fragment?	[fragmentation alternatives]
❸ How much to fragment?	[degree of fragmentation]
❹ How to test correctness?	[correctness rules of fragmentation]
❺ How to allocate?	[allocation alternatives]
❻ Information requirements?	[for both fragmentation and allocation]

1. Reasons for Fragmentation

- Can't we just distribute relations (no intrinsic reason to fragment)?
  - ➡ distributed file systems are not fragmented (i.e., distr. unit is the file)
- What is a reasonable unit of distribution?
  - ➡ advantages of fragmentation (why isn't relation the best choice?)
    - ✦ application views are subsets of relations ➔ **locality** allows for finer accesses (applications only access to relevant subsets of relations)
      - ✓ 2 applications accessing different portion of a relation: without fragmentation, either unnecessary data replication or loss of locality (extra communication)
    - ✦ without fragmentation, no **intra-query parallelism**
  - ➡ disadvantages of fragmentation
    - ✦ might cause queries to be executed on more than one fragment (performance degradation, especially when fragments are not disjoint)
    - ✦ semantic data control (especially integrity enforcement) more difficult and costly

2. Fragmentation Alternatives

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

Horizontal fragmentation

- $PROJ_1$: projects with budget less than $200,000
- $PROJ_2$: projects with budget greater than or equal to $200,000

Vertical fragmentation

- PROJ1: information about project budgets
- PROJ2:information about project names and locations

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

$PROJ_1$

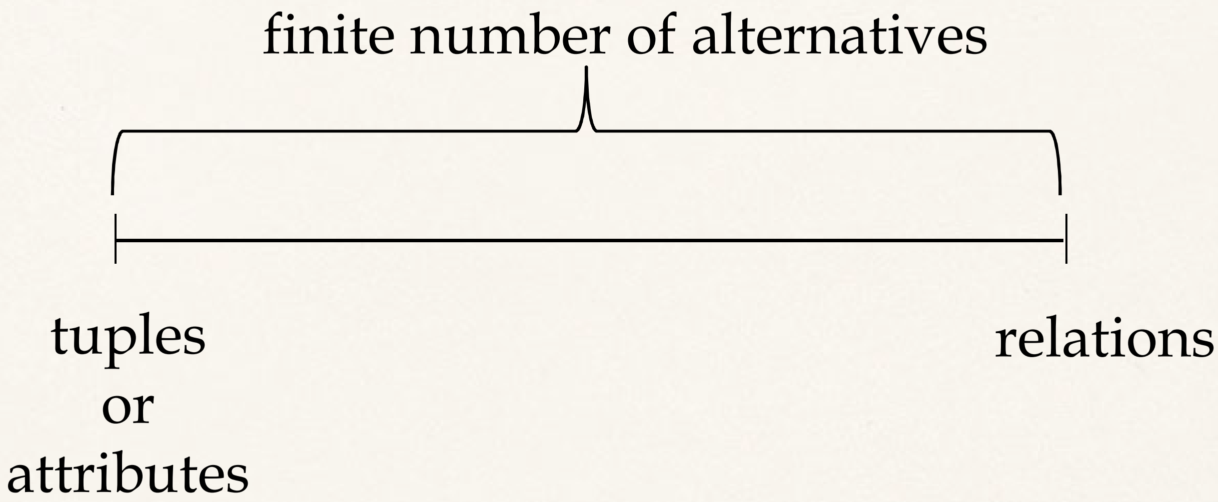
PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000

$PROJ_2$

PNO	PNAME	LOC
P1	Instrumentation	Montreal
P2	Database Develop.	New York
P3	CAD/CAM	New York
P4	Maintenance	Paris

Hybrid fragmentation: obtained by nesting horizontal and vertical fragmentation

3. Degree of Fragmentation



- Finding the suitable level of partitioning within this range
- It depends especially on the applications that will use the DB
- This is the real difficulty of fragmentation

4. Correctness of Fragmentation

- Completeness
  - ➡ Decomposition of relation R into fragments $R_1, R_2, \ldots, R_n$ is complete if and only if each data item in R can also be found in some R_i
- Reconstruction
  - ➡ If relation R is decomposed into fragments $R_1, R_2, \ldots, R_n$, then there should exist some relational operator ∇ such that

$$R = \nabla_{1 \leq i \leq n} R_i$$

- Disjointness
  - ➡ If relation R is decomposed into fragments $R_1, R_2, \ldots, R_n$, and data item d_i is in R_j, then d_i should not be in any other fragment R_k ($k \neq j$).

5. Allocation Alternatives

- Assigning fragments to sites and deciding **whether or not to replicate a fragment**
  - ➡ *partitioned* (aka *non-replicated*): each fragment resides at only one site
  - ➡ *fully replicated*: each fragment at each site
  - ➡ *partially replicated*: each fragment at some of the sites
- Rule of thumb:

  If $\frac{\text{read-only queries}}{\text{update queries}} >> 1$, replication is advantageous,

  otherwise replication may cause problems
- In case of partially replicated DDBS, the number of copies of replicated fragments can either be an input to the allocation algorithm or a decision variable to be computed by the algorithm

6. Information Requirements

- The difficulty of the distributed DB design problem is that too many factor affect the choices towards an optimal design
  - ➡ Logical organization of the DB
  - ➡ Location of DBMS applications
  - ➡ Characteristics of user applications (how they access the DB)
  - ➡ Properties of (computers at) network nodes
  - ➡ …
- Those can be grouped into four categories:
  - ➡ Database information
  - ➡ Application information
  - ➡ Communication network information
  - ➡ Computer system information

} quantitative information, mostly used for allocation, we will not treat them

Fragmentation

- Horizontal Fragmentation (HF)
  - ➡ Primary Horizontal Fragmentation (PHF)
  - ➡ Derived Horizontal Fragmentation (DHF)
- Vertical Fragmentation (VF)
- Hybrid Fragmentation (HyF)

PHF – Information Requirements

- application information needed for horizontal fragmentation
  - ➡ Predicates used in queries
    - ✦ 80/20 rule: the most active 20% of user applications account for 80% of accesses
    - ✦ **simple predicates**: Given $R[A_1, A_2, \ldots, A_n]$, a **simple predicate** p_j over R is

      $$A_i \;\; \theta \;\; Value$$

      where $\theta \in \{=,<,\leq,>,\geq,\neq\}$, $Value \in D_i$ and D_i is the domain of A_i.

      Example:

      PNAME = "Maintenance"

      BUDGET ≤ 200 000
    - ✦ **minterms**: Given a set $Pr = \{p_1, p_2, \ldots, p_m\}$ of simple predicates over a relation R, a **minterm** (induced by Pr) is a conjunction

      $$\bigwedge_{p_j \in Pr} p_r^*$$

      where $p_j^* \in \{ p_j , \neg p_j \}$, for all $p_j \in Pr$

      We let $M_{Pr} = \{m_1, m_2, \ldots, m_r\}$ be the set of all minterms induced by a set of simple predicates Pr

PHF – Information Requirements Example

Example

Pr = { PNAME="Maintenance" , BUDGET < 200000 }

$M_{Pr} = \{ m_1 , m_2 , m_3 , m_4 \}$

Where

- m_1: PNAME="Maintenance" $\wedge$ BUDGET < 200000
- m_2: $\neg$(PNAME="Maintenance") $\wedge$ BUDGET < 200000
- m_3: PNAME= "Maintenance" $\wedge$ $\neg$(BUDGET < 200000)
- m_4: $\neg$(PNAME="Maintenance") $\wedge$ $\neg$(BUDGET < 200000)

PHF – Extra Information Requirements

- Database Information
  - ➡ minterm selectivity *(quantitative)*
- Application Information
  - ➡ predicates used in queries (simple predicates, minterms) *(qualitative)*
  - ➡ access frequency of queries *(quantitative)*

Primary Horizontal Fragmentation

- Primary horizontal fragmentation (**PHF**) is induced by a set of minterm.
- **Definition:** A set $M = \{ m_1, m_2, \ldots, m_n \}$ of minterm induces the fragmentation

$$F = \{ R_i \mid R_i = \sigma_{m_i}(R),\ m_i \in M \}$$

- Therefore, a horizontal fragment R_i of relation R consists of all the tuples of R which satisfy a minterm predicate m_i

$$\Downarrow$$

Given a set of minterm predicates M, there are as many horizontal fragments of relation R as there are minterm predicates (some fragments might be empty)

PHF – Example (1)

- Assume there is an application **Q: find projects with budget less than 200 000 €**
- Then, it makes sense to consider the set of simple predicates S = { BUDGET < 200000 }

  which induces the set of minterms M_S = { BUDGET < 200000, ¬(BUDGET < 200000) }

  which, in turn, induces fragmentation F = { $PROJ_1$, $PROJ_2$ }
- $PROJ_1$ and $PROJ_2$ are the **fragments induced by *S***

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

PHF – Example (2)

- Consider now another application **Q′: find projects at any given location**
  Then, it makes sense to consider the set of simple predicates

$$S' = \{\ LOC = \text{“Montreal”},\quad LOC = \text{“New York”},\quad LOC = \text{“Paris”}\ \}$$

which induces the set of minterms (use abbreviations $\mathbf{L_M}$: LOC = “Montreal”, $\mathbf{L_N}$: LOC = “New York”, $\mathbf{L_P}$: LOC = “Paris”)

$M_{S'} = \{$ ~~$L_M \wedge L_N \wedge L_P$,~~ ~~$L_M \wedge L_N \wedge \neg L_P$,~~ ~~$L_M \wedge \neg L_N \wedge L_P$,~~ $L_M \wedge \neg L_N \wedge \neg L_P$,
~~$\neg L_M \wedge L_N \wedge L_P$,~~ $\neg L_M \wedge L_N \wedge \neg L_P$, $\neg L_M \wedge \neg L_N \wedge L_P$, ~~$\neg L_M \wedge \neg L_N \wedge \neg L_P$~~ $\}$

which reduces to $\{\ L_M \wedge \neg L_N \wedge \neg L_P,\quad \neg L_M \wedge L_N \wedge \neg L_P,\quad \neg L_M \wedge \neg L_N \wedge L_P\ \}$

or, even more succinctly, $\{\ L_M\ ,\ L_N\ ,\ L_P\ \}$

which, in turn, induces fragmentation $F' = \{\ PROJ'_1\ ,\ PROJ'_2\ ,\ PROJ'_3\ \}$

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

$PROJ'_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

$PROJ'_2$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York

$PROJ'_3$

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

Completeness of the Set of Simple Predicates

- Set of simple predicates (and thus sets of minterms) should be complete and minimal
- Intuitively, *complete* means that all applications (queries) are taken into account
- Definition: a set of simple predicates *Pr* is said to be complete if and only if any two tuples in a fragment induced by *Pr* have the same probability of being accessed by any application

Informal definition (completeness): **in other words**, we have that

Q access either **all** or **none** of the tuples in *F*

for every application *Q* and every fragment *F* induced by *Pr*

Completeness – Examples

Informal definition (completeness): Q and Q' access either **all** or **none** of the tuples in each fragment

- Q: find projects with budget less than 200 000 €
- Q': find projects based in New York
- Is S' = { LOC = "New York" } complete wrt. appl. Q and Q' ?
  - **NO!**
    - it produces F = { $PROJ_1$, $PROJ_2$ }
    - Q only accesses project P2 in fragment $PROJ_1$

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P4	Maintenance	310000	Paris

- S'' = {BUDGET < 200000 , LOC = "New York" } is **complete** wrt. appl. Q and Q'
  - it produces the minterm set (L_N stands for LOC = "New York")

$M_{S''}$ = { BUDGET < 200000 $\wedge \neg L_N$, BUDGET $\geq$ 200000 $\wedge \neg L_N$, BUDGET < 200000 $\wedge L_N$, BUDGET $\geq$ 200000 $\wedge L_N$, }

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

Minimality of the Set of Simple Predicates

- Set of simple predicates (and thus sets of minterms) should be complete and minimal
- Intuitively, *minimal* means that all predicates should be relevant in the set:
  - relevant wrt. to final fragmentation (every predicate produces some fragments *not produced by other predicates in Pr*)
  - relevant wrt. to applications (there is at least one application that benefits from the predicate)
- Definition: a set of simple predicates *Pr* is said to be minimal if and only if every predicates $p \in Pr$ creates a new fragment (i.e., *p* divides fragment *F* into F_1 and F_2) and F_1 and F_2 are accessed differently by at least one application

Minimality – Example 1

- Intuitively, *minimal* means that all predicates should be relevant in the set wrt.:
  - final fragmentation (every predicate produces some fragments *not produced by other predicates*)
  - applications (there is at least one application that benefit from the predicate)

- *Q*: find projects with budget less than 200 000 €
- *Q′*: find projects based in New York
- *Q′′*: find "Database Develop." projects

L_N **stands for** LOC = "New York"

$PNAME_{DBdevel}$ **stands for** PNAME = "Database Develop."

- Is *S′′* = { BUDGET < 200000 , L_N , $PNAME_{DBdevel}$ } minimal wrt. applications *Q*, *Q′*, *Q′′* ?
  - NO!
    - $PNAME_{DBdevel}$ is not relvant wrt. final fragmentation
    - *S′′* = { BUDGET < 200000 , L_N } produces the same fragmentation

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

$PROJ'_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

$PROJ''_2$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York

$PROJ'''_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York

$PROJ'_3$

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

Minimality – Example 2

- Intuitively, *minimal* means that all predicates should be relevant in the set wrt.:
  - final fragmentation (every predicate produces some fragments *not produced by other predicates*)
  - applications (there is at least one application that benefit from the predicate)

- Q': find projects based in New York
- Is S'' = { BUDGET < 200000 , L_N } minimal wrt. application Q' ?
  - it produces F = { $PROJ'_1$, $PROJ''_2$, $PROJ'''_2$, $PROJ'_3$ }
  - BUDGET < 200000 is the reason of dividing $PROJ''_2$ and $PROJ'''_2$
  - Q' cannot distinguish between $PROJ''_2$ and $PROJ'''_2$:

    Q' accesses $PROJ''_2$ iff Q' accesses $PROJ'''_2$

$PROJ'_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

$PROJ''_2$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York

$PROJ'''_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York

$PROJ'_3$

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

PHF – Algorithm (Intuition)

Input: a relation R and a set of simple predicates Pr over attributes of R

Output: a *complete* and *minimal* set of simple predicates Pr' over R

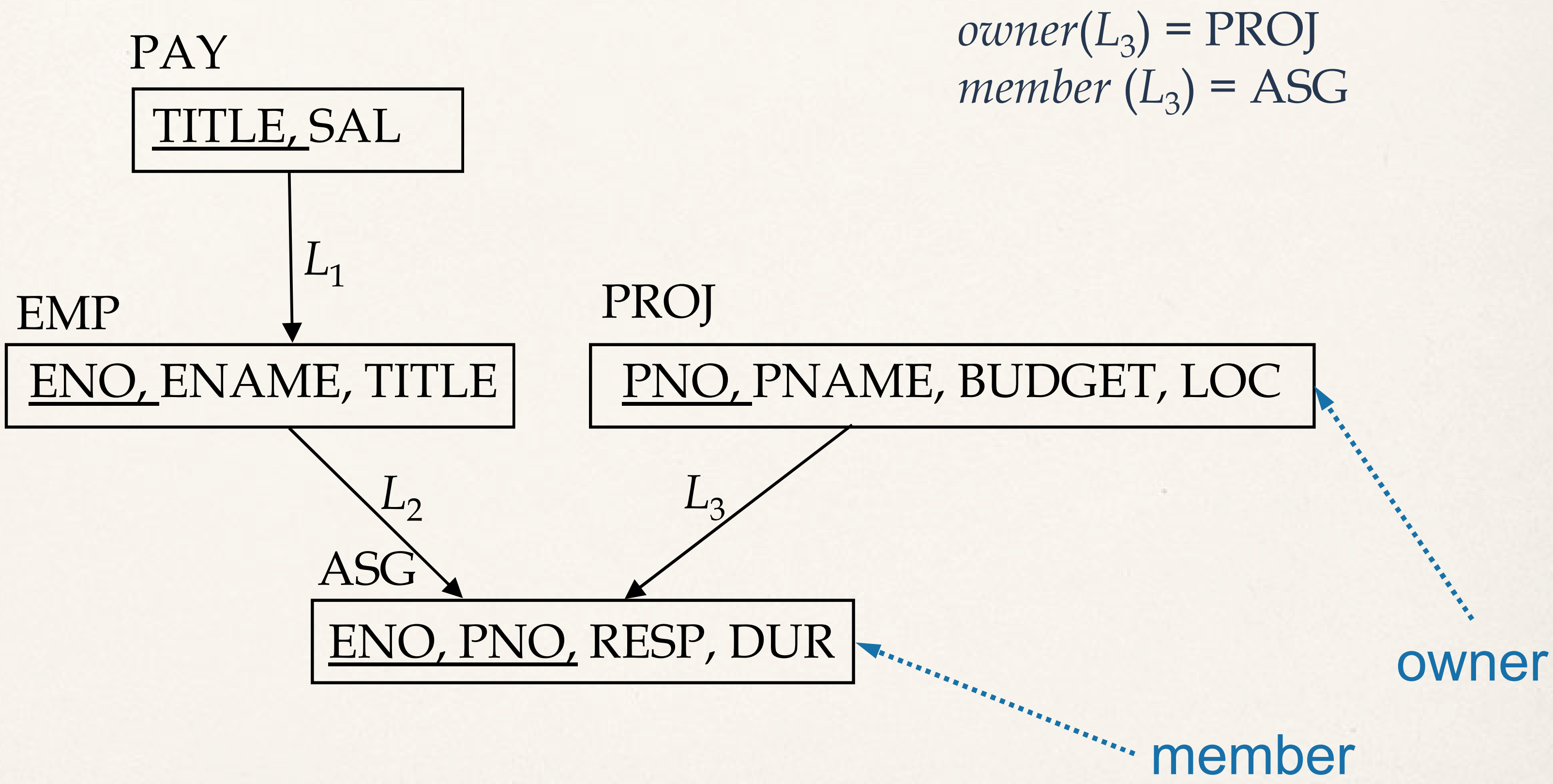
Minimality rule (relevant predicates): a predicate $p \in Pr$ is **relevant in *Pr*** if and only if

- ➡ produces some fragments which is not produced by any other predicate in Pr
- ➡ there is at least one application that benefit from p

```
repeat
     select a relevant predicate p ∈ Pr
     Pr := Pr \ { p }
     P′ := P′ ∪ { p }
     P′ := P′ \ {p ∈ P′ | p is not relevant in P′ }
until P′ is complete
compute set M of minterms induced by Pr
eliminate contradictory minterms from M    // i.e., minterms that
                                           // produce empty fragments
return fragmentation F = { R_m = σ_m(R) | m ∈ M }
```

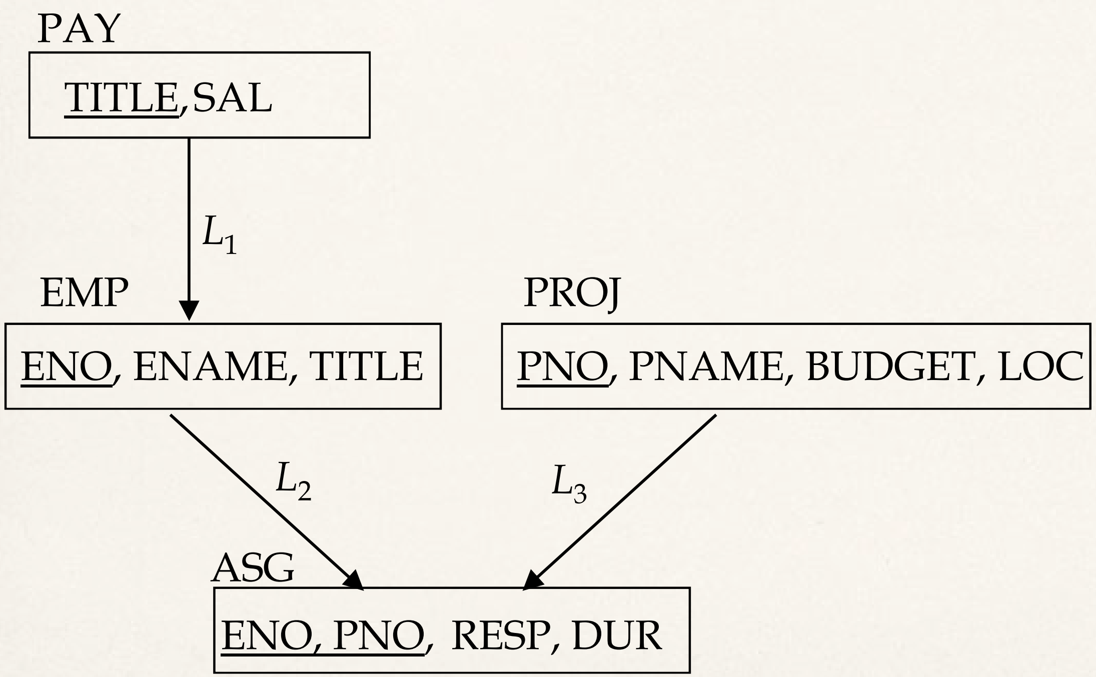
DHF – Information Requirements

- qualitative Database Information
  - ➡ relationship



Derived Horizontal Fragmentation

- Derived Horizontal Fragmentation (DHF) is defined on a member relation of a link according to a selection operation specified on its owner (propagated from owner to member)



owner(L_1) = PAY
member (L_1) = EMP

owner(L_2) = EMP
member (L_2) = ASG

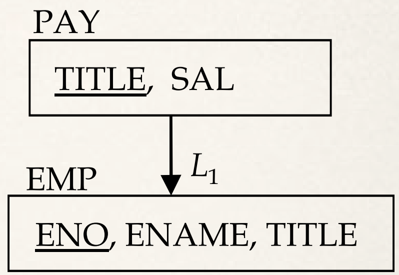
owner(L_3) = PROJ
member (L_3) = ASG

DHF – Definition

Given

- a relation S fragmented into $F_S = \{ S_1, S_2, \ldots, S_w \}$ and
- a link L where $owner(L)=S$ and $member(L)=R$,

the derived horizontal fragments of R are defined as $R_i = R \ltimes S_i \; (S_i \in F_S)$



PAY$_1$

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000

PAY$_2$

TITLE	SAL
Mech. Eng.	27000
Programmer	24000

EMP$_1$

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E8	J. Jones	Syst. Anal.

EMP$_2$

ENO	ENAME	TITLE
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E7	R. Davis	Mech. Eng.

PAY

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

$PAY_1 = \sigma_{SAL \geq 30000}(PAY)$
$PAY_2 = \sigma_{SAL < 30000}(PAY)$

$EMP_1 = EMP \ltimes PAY_1$
$EMP_2 = EMP \ltimes PAY_2$

ASG could be fragmented into (the choice depends on applications)

either $ASG_i = ASG \ltimes EMP_i$

or $ASG_i = ASG \ltimes PROJ_i$

HF – Correctness

- **Completeness** (info is entirely preserved) for **primary** horizontal fragmentation
  - ➡ PHF: completeness follows from the way minterms are built (**exhaustively**)
    - ✦ NOTICE: The textbook says something slightly different
- **Reconstruction** for both **primary** and **derived** horizontal fragmentation
  - ➡ Assume R is fragmented into $F = \{R_1, R_2, \ldots, R_r\}$

$$R = \bigcup_{\forall R_i \in F} R_i$$

- **Disjointness** for **primary** horizontal fragmentation
  - ➡ PHF: minterms are **mutually exclusive** by construction (assuming the set of simple predicates to be minimal)
- **Completeness** and **disjointness** for **derived** horizontal fragmentation
  - ➡ Both come from **integrity constraints** of foreign keys and from completeness/disjointness of PHF
    - ✦ fragmentation propagates from *owner* to *member* following one-to-many associations; thus, each tuple of *member* is associated with exactly 1 tuple of *owner* (a NOT NULL constraint must be defined on the foreign key in the *member* relation that refer to the *owner* relation); by disjointness and completeness of PHF, such tuple of owner appears in exactly 1 fragment of owner

Vertical Fragmentation

- Has been studied within the centralized context
  - ➡ design methodology
  - ➡ physical clustering
- Choose a partition $P = \{ P_1, P_2, \ldots, P_n \}$ of the set of attribute of relation. Then,

$$F = \{ R_i \mid R_i = \Pi_{P_i \cup key}(R) \text{ and } P_i \in P \}$$

  where *key* is the (set of) key attribute(s): they are replicated in each fragment
- The problems boils down to finding the best partition
  - ➡ Number of elements of the partition
  - ➡ Distribution of attributes among elements of the partition
- More difficult than horizontal, because more alternatives exist
  - ➡ Number of possible partitions of a set of size n is the Bell's number B_n (its growth rate is more than exponential)
- Two approaches :
  - ➡ Grouping (bottom-up) – from single attributes to fragments
  - ➡ Splitting (top-down) – from relation to fragments
    - ✦ preferable for 2 reasons
      - ✓ close to the design approach
      - ✓ optimal solution is more likely to be close to the full relation than to the fully fragmented situation

VF – The General Idea

- Partition is guided by a measure of affinity ("togetherness")
- Affinity measures how much attributes that are accessed together by queries

VF – Information Requirements (Qualitative Application Info)

- The matrix ***use(q, A)*** for attribute usage values
  - ➡ R relation over attributes $A_1, A_2, \ldots, A_n$
  - ➡ $Q = \{q_1, q_2, \ldots, q_q\}$: set of queries that will run on R
    - ✦ (the 80/20 rule can be used here, too: select the most active 20% of queries only)

$$use(q_i, A_j) = \begin{cases} 1 \text{ if attribute } A_j \text{ is referenced by query } q_i \\ 0 \text{ otherwise} \end{cases}$$

VF – Example of $use(q_i, A_j)$

Consider the following 4 queries for relation PROJ

q_1: **SELECT** BUDGET
FROM PROJ
WHERE PNO=Value

q_2: **SELECT** PNAME,BUDGET
FROM PROJ

q_3: **SELECT** PNAME
FROM PROJ
WHERE LOC=Value

q_4: **SELECT** **SUM**(BUDGET)
FROM PROJ
WHERE LOC=Value

use(q, A)

	PNO	*PNAME*	*BUDGET*	*LOC*
q_1	1	0	1	0
q_2	0	1	1	0
q_3	0	1	0	1
q_4	0	0	1	1

VF – Information Requirements (Quantitative Application Info)

- matrix $acc_s(q)$ for the number of execution of q at s in a given period
- attribute affinity measure $aff(A_i, A_j)$ between any two attributes A_i and A_j of a relation R with respect to a set of applications Q

$$aff(A_i, A_j) = \sum_{\substack{\text{all queries } q \\ \text{that access} \\ \text{both } A_i \text{ and } A_j}} \; \sum_{\text{all sites } s} acc_s(q)$$

according to matrix $use(q,A)$:
all queries q such that
$use(q, A_i) = use(q, A_j) = 1$

$use(q, A)$	PNO	PNAME	BUDGET	LOC
q_1	1	0	1	0
q_2	0	1	1	0
q_3	0	1	0	1
q_4	0	0	1	1

VF – Computation of $aff(A_i, A_j)$

$$aff(A_i, A_j) = \sum_{\substack{\text{all queries } q \text{ that} \\ \text{access both } A_i \\ \text{and } A_j}} \; \sum_{\text{all sites } s} acc_s(q)$$

- Example: affinity between *PNO* and *BUDGET*
- q_1 is the only query that access both *PNO* and *BUDGET*
- Also consider the access frequencies: $acc_s(q)$
- Then, ***aff*(*PNO, BUDGET*)** = 15 + 20 + 10 = 45
- *aff*(. , .) is stored in the **attribute affinity matrix** *AA*
- Any clustering algorithm based on the attribute affinity values
  - ➡ Bond energy algorithm
  - ➡ Neural network
  - ➡ Machine learning
  - ➡ **(no details here)**

use(q, A)

	PNO	*PNAME*	*BUDGET*	*LOC*
q_1	1	0	1	0
q_2	0	1	1	0
q_3	0	1	0	1
q_4	0	0	1	1

$acc_s(q)$

	S_1	S_2	S_3
q_1	15	20	10
q_2	5	0	0
q_3	25	25	25
q_4	3	0	0

$aff(A_i, A_j)$

	PNO	*PNAME*	*BUDGET*	*LOC*
PNO	45	0	45	0
PNAME	0	80	5	75
BUDGET	45	5	53	3
LOC	0	75	3	78

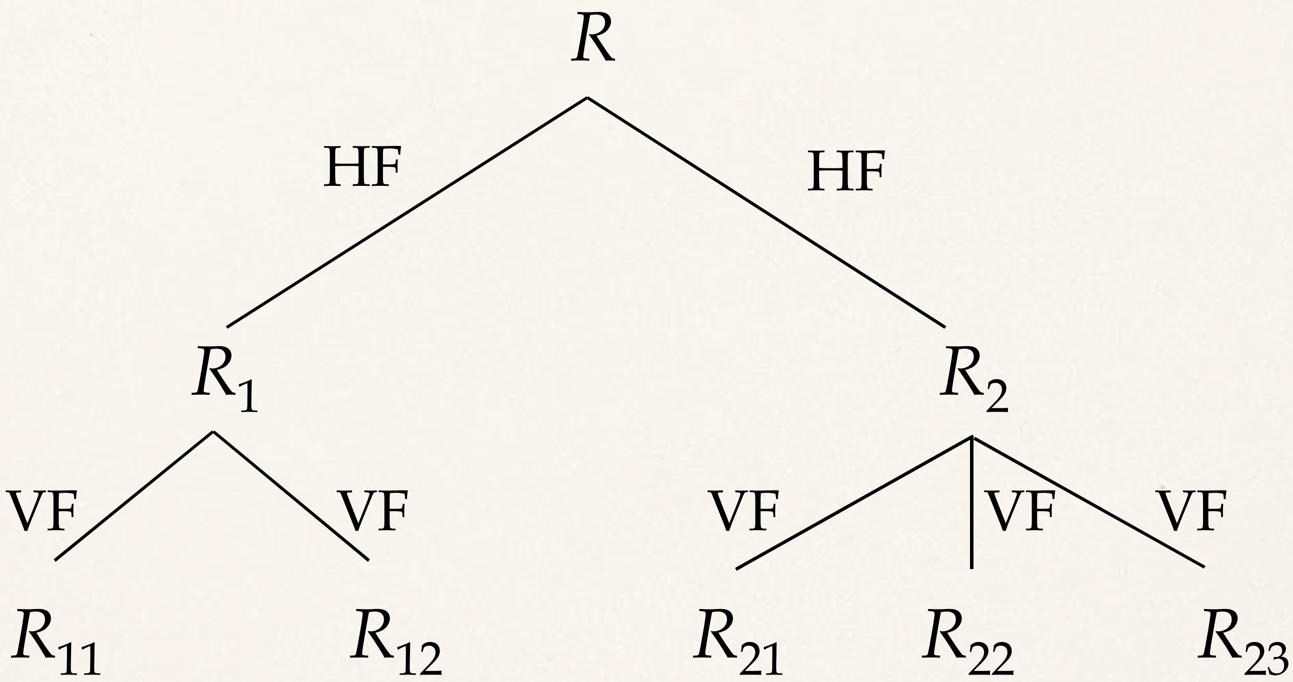
VF – Correctness

- Completeness and disjointness follow from properties (completeness and disjointness) intrinsic of a partition (returned by the clustering algorithm)
- Reconstruction
  - ➡ Let $F_R = \{R_1, R_2, \ldots, R_n\}$ be the vertical fragmentation obtained for R
  - ➡ R is recovered by joining the fragments

$$R = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$$

Hybrid Fragmentation

Hybrid fragmentation, aka *mixed* or *nested fragmentation*



To reconstruct R: start from the leaves and move upward applying fragmentation reconstruction methods depending on fragmentation types

Fragment Allocation

- Fragment allocation concerns distribution of resources across network nodes
  - ➡ Assignment (possibly with replications) of fragments to sites
- Problem formalization
  - ➡ Given

    $F = \{F_1, F_2, \ldots, F_n\}$ fragments

    $S = \{S_1, S_2, \ldots, S_m\}$ network sites

    $Q = \{q_1, q_2, \ldots, q_q\}$ application information (frequencies, access patterns, ecc.)

    Find the best ("optimal") distribution of fragments in F among sites in S according to info in Q
- Optimality factors
  - ➡ Minimal cost
    - ✦ Communication, Storage (of F_i at site s_j), Querying (F_i at site s_j , from site s_k), Updating (F_i at all sites where it is replicated, from site s_k)
  - ➡ Performance
    - ✦ Response time and/or total time
  - ➡ Can be formulated as an operations research problem
    - ✦ one of the above optimality factors is the cost function to minimize, the others are constraint to satisfy)

```
min (cost function)
s.t. constraints (response time, storage, ...)
```

    - ✦ techniques and heuristics from the field of operations research apply (no optimal solution, NP-hard)

Data directory

- Data directory (aka. data dictionary or catalog)
- Both in classic (centralized) and distributed DB, it stores metadata about DB
  - ➡ Centralized context
    - ✦ Schema (relation metadata) definitions
    - ✦ Usage statistics
    - ✦ Memory usage
    - ✦ ...
  - ➡ Distributed context
    - ✦ Info to reconstruct global view of whole DB
    - ✦ What relation/fragment is stored at which site
    - ✦ ...
- It is itself part of the DB, so considerations about fragmentation and allocation issues apply