Chapter 3: Distributed Database Design

- Design problem
- Design strategies(top-down, bottom-up)
- Fragmentation
- Allocation and replication of fragments, optimality, heuristics

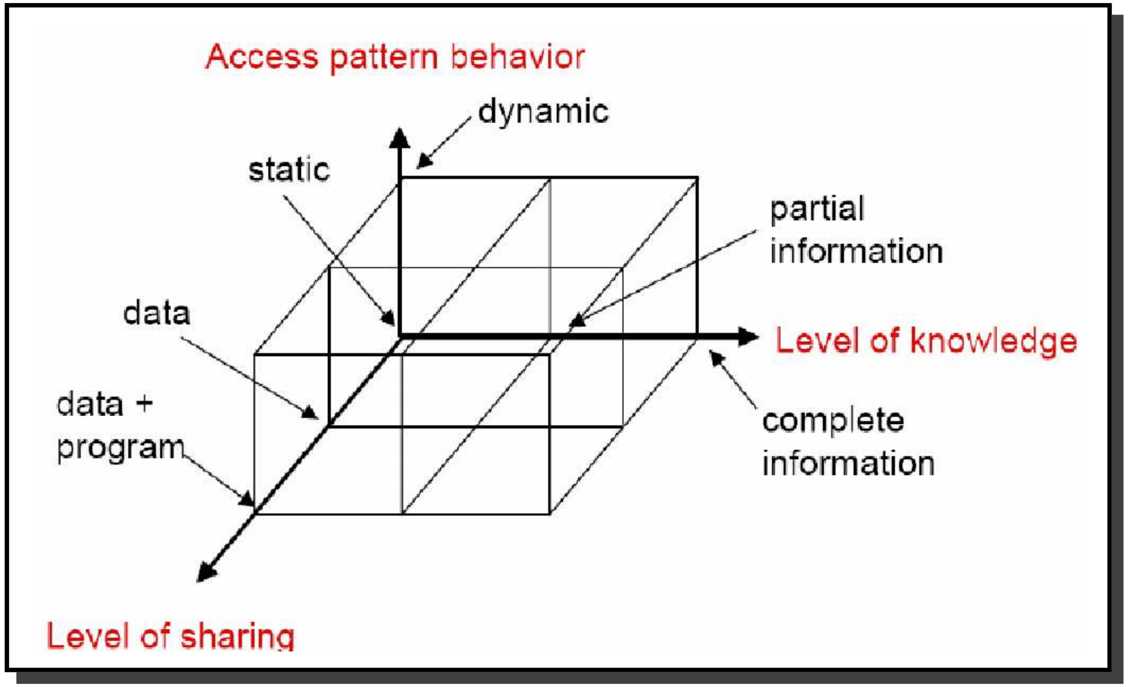
Acknowledgements: I am indebted to Arturas Mazeika for providing me his slides of this course.

Design Problem

- **Design problem of distributed systems**: Making decisions about the placement of **data** and **programs** across the sites of a computer network as well as possibly designing the network itself.
- In DDBMS, the distribution of applications involves
  - Distribution of the DDBMS software
  - Distribution of applications that run on the database
- Distribution of applications will not be considered in the following; instead the distribution of data is studied.

Framework of Distribution

- Dimension for the analysis of distributed systems
  - Level of sharing: no sharing, data sharing, data + program sharing
  - Behavior of access patterns: static, dynamic
  - Level of knowledge on access pattern behavior: no information, partial information, complete information



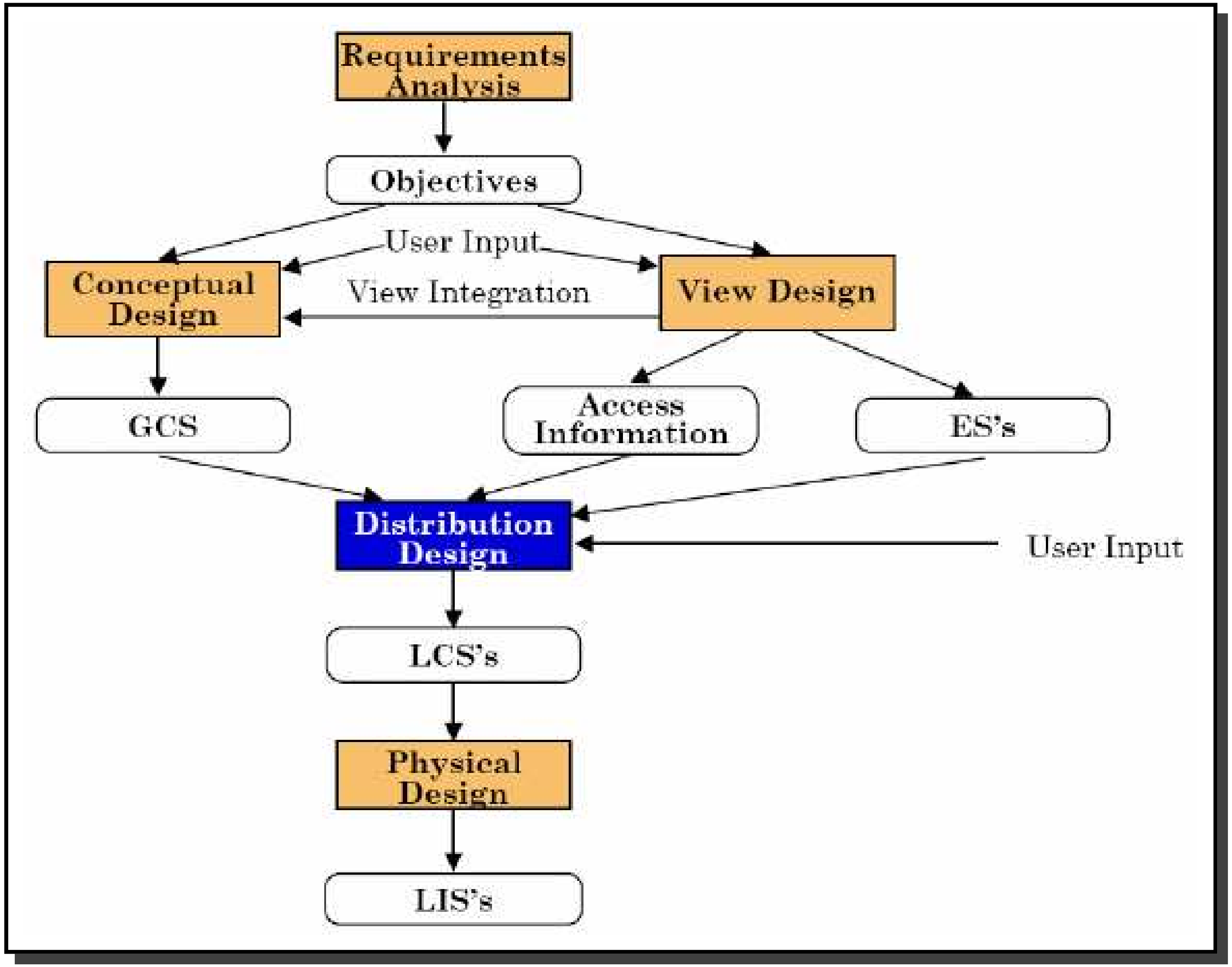
- Distributed database design should be considered within this general framework.

Design Strategies

- Top-down approach
  - Designing systems from scratch
  - Homogeneous systems
- Bottom-up approach
  - The databases already exist at a number of sites
  - The databases should be connected to solve common tasks

Design Strategies . . .

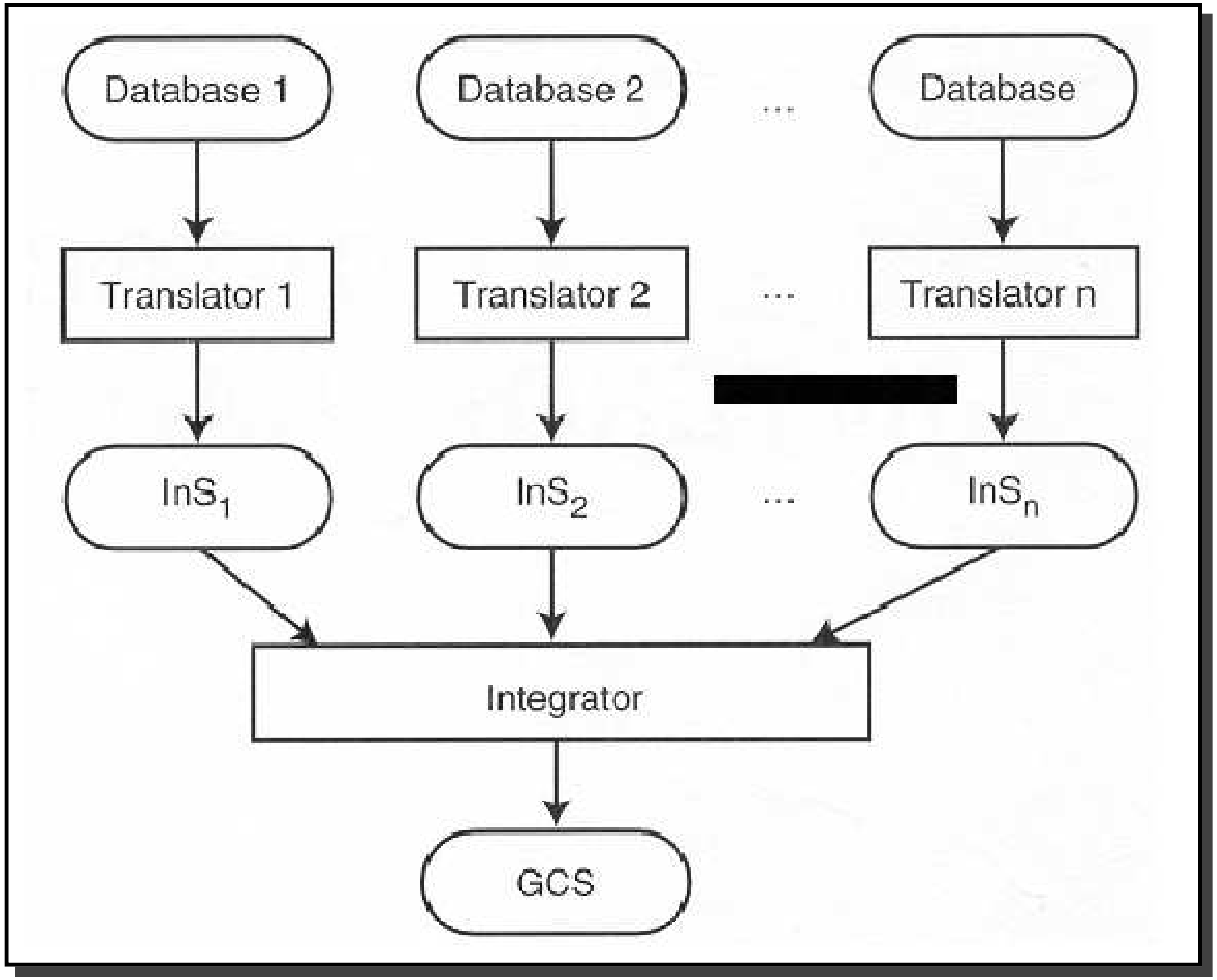
- **Top-down design strategy**



Design Strategies . . .

- **Distribution design** is the central part of the design in DDBMSs (the other tasks are similar to traditional databases)
  - **Objective**: Design the LCSs by distributing the entities (relations) over the sites
  - Two main aspects have to be designed carefully
    - **Fragmentation**
      - Relation may be divided into a number of sub-relations, which are distributed
    - **Allocation** and **replication**
      - Each fragment is stored at site with ”optimal” distribution
      - Copy of fragment may be maintained at several sites
- In this chapter we mainly concentrate on these two aspects
- Distribution design issues
  - Why fragment at all?
  - How to fragment?
  - How much to fragment?
  - How to test correctness?
  - How to allocate?

- **Bottom-up design strategy**



Fragmentation

- What is a reasonable unit of distribution? Relation or fragment of relation?
- **Relations** as unit of distribution:
  - If the relation is not replicated, we get a high volume of remote data accesses.
  - If the relation is replicated, we get unnecessary replications, which cause problems in executing updates and waste disk space
  - Might be an Ok solution, if queries need all the data in the relation and data stays at the only sites that uses the data
- **Fragments** of relationas as unit of distribution:
  - Application views are usually subsets of relations
  - Thus, locality of accesses of applications is defined on subsets of relations
  - Permits a number of transactions to execute concurrently, since they will access different portions of a relation
  - Parallel execution of a single query (intra-query concurrency)
  - However, semantic data control (especially integrity enforcement) is more difficult

$\Rightarrow$ Fragments of relations are (usually) the appropriate unit of distribution.

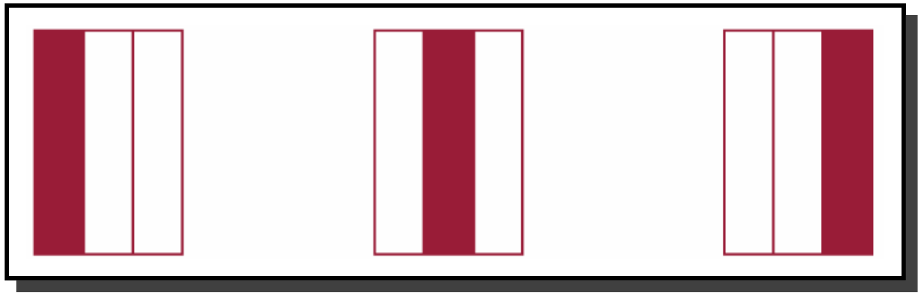
Fragmentation . . .

- Fragmentation aims to improve:
  - Reliability
  - Performance
  - Balanced storage capacity and costs
  - Communication costs
  - Security
- The following information is used to decide fragmentation:
  - Quantitative information: frequency of queries, site, where query is run, selectivity of the queries, etc.
  - Qualitative information: types of access of data, read/write, etc.

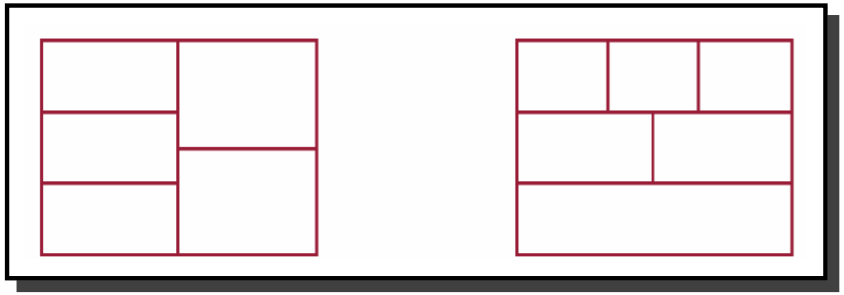
Fragmentation ...

- Types of Fragmentation
  - Horizontal: partitions a relation along its tuples
  - Vertical: partitions a relation along its attributes
  - Mixed/hybrid: a combination of horizontal and vertical fragmentation

(a) Horizontal Fragmentation



(b) Vertical Fragmentation



(c) Mixed Fragmentation

Fragmentation ...

- **Exampe**

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.

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E8	P3	Manager	40

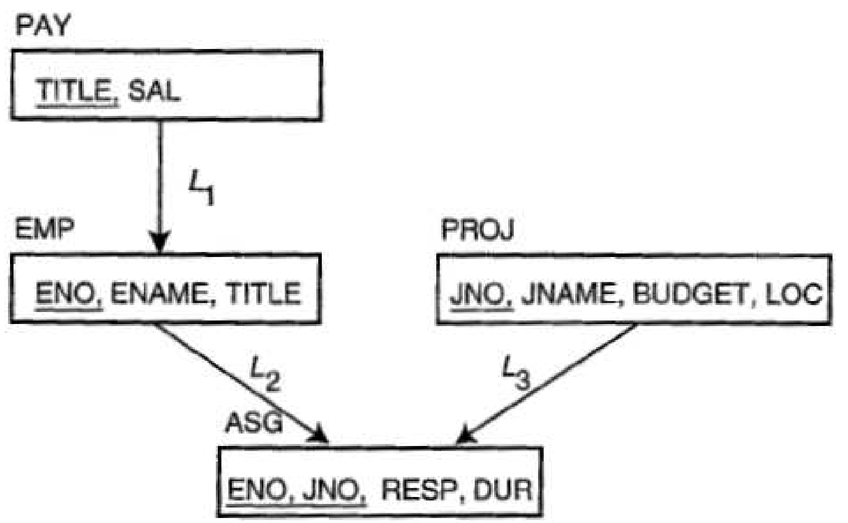
PROJ

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

PAY

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

Data



E-R Diagram

- **Example (contd.):** Horizontal fragmentation of PROJ relation
  - PROJ1: projects with budgets less than $200,000$
  - PROJ2: projects with budgets greater than or equal to $200,000$

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
P5	CAD/CAM	500000	Boston

$PROJ_1$

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
P5	CAD/CAM	500000	Boston

Fragmentation . . .

- **Example (contd.):** Vertical fragmentation of PROJ relation
  - PROJ1: information about project budgets
  - PROJ2: information about project names and locations

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
P5	CAD/CAM	500000	Boston

$PROJ_1$

PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000
P5	500000

$PROJ_2$

PNO	PNAME	LOC
P1	Instrumentation	Montreal
P2	Database Develop.	New York
P3	CAD/CAM	New York
P4	Maintenance	Paris
P5	CAD/CAM	Boston

Correctness Rules of Fragmentation

- **Completeness**
  - Decomposition of relation R into fragments $R_1, R_2, \ldots, R_n$ is complete iff 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 ∇ that reconstructs R from its fragments, i.e., $R = R_1 \nabla \ldots \nabla R_n$
    * Union to combine horizontal fragments
    * Join to combine vertical fragments
- **Disjointness**
  - If relation R is decomposed into fragments $R_1, R_2, \ldots, R_n$ and data item d_i appears in fragment R_j, then d_i should not appear in any other fragment R_k, $k \neq j$ (exception: primary key attribute for vertical fragmentation)
    * For horizontal fragmentation, data item is a tuple
    * For vertical fragmentation, data item is an attribute

Horizontal Fragmentation

- **Intuition** behind horizontal fragmentation
  - Every site should hold all information that is used to query at the site
  - The information at the site should be fragmented so the queries of the site run faster
- Horizontal fragmentation is **defined as selection operation**, $\sigma_p(R)$
- **Example**:

$$\sigma_{\text{BUDGET}<200000}(PROJ)$$
$$\sigma_{\text{BUDGET}\geq 200000}(PROJ)$$

- **Computing** horizontal fragmentation (idea)
  - Compute the **frequency** of the individual queries of the site $q_1, \ldots, q_Q$
  - Rewrite the queries of the site in the conjunctive normal form (disjunction of conjunctions); the conjunctions are called **minterms**.
  - Compute the **selectivity** of the minterms
  - Find the **minimal** and **complete** set of minterms (predicates)
    * The set of predicates is **complete** if and only if any two tuples in the same fragment are referenced with the same probability by any application
    * The set of predicates is **minimal** if and only if there is at least one query that accesses the fragment
  - There is an algorithm how to find these fragments algorithmically (the algorithm `CON_MIN` and `PHORIZONTAL` (pp 120-122) of the textbook of the course)

- **Example:** Fragmentation of the $PROJ$ relation
  - Consider the following query: *Find the name and budget of projects given their PNO.*
  - The query is issued at all three sites
  - Fragmentation based on LOC, using the set of predicates/minterms $\{LOC =' Montreal', LOC =' NewYork', LOC =' Paris'\}$

$PROJ_1 = \sigma_{LOC='Montreal'}(PROJ)$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

$PROJ_2 = \sigma_{LOC='NewYork'}(PROJ)$

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

$PROJ_3 = \sigma_{LOC='Paris'}(PROJ)$

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

- If access is only according to the location, the above set of predicates is complete
  - i.e., each tuple of each fragment $PROJ_i$ has the same probability of being accessed
- If there is a second query/application to access only those project tuples where the budget is less than $200000, the set of predicates is not complete.
  - $P2$ in $PROJ_2$ has higher probability to be accessed

- **Example (contd.):**
  - Add $BUDGET \leq 200000$ and $BUDGET > 200000$ to the set of predicates to make it complete.
    $\Rightarrow \{LOC =' Montreal', LOC =' NewYork', LOC =' Paris', BUDGET \geq 200000, BUDGET < 200000\}$ is a complete set
  - Minterms to fragment the relation are given as follows:

$$(LOC =' Montreal') \wedge (BUDGET \leq 200000)$$

$$(LOC =' Montreal') \wedge (BUDGET > 200000)$$

$$(LOC =' NewYork') \wedge (BUDGET \leq 200000)$$

$$(LOC =' NewYork') \wedge (BUDGET > 200000)$$

$$(LOC =' Paris') \wedge (BUDGET \leq 200000)$$

$$(LOC =' Paris') \wedge (BUDGET > 200000)$$

Horizontal Fragmentation . . .

- **Example (contd.):** Now, $PROJ_2$ will be split in two fragments

$PROJ_1 = \sigma_{LOC='Montreal'}(PROJ)$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

$PROJ_2 = \sigma_{LOC='NY' \wedge BUDGET < 200000}(PROJ)$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York

$PROJ_3 = \sigma_{LOC='Paris'}(PROJ)$

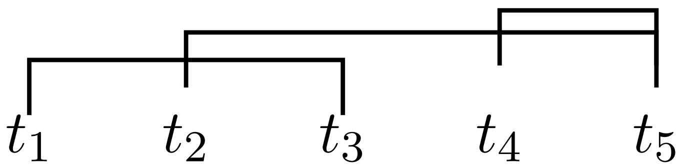
PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

$PROJ_2' = \sigma_{LOC='NY' \wedge BUDGET \geq 200000}(PROJ)$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York

  - $PROJ_1$ and $PROJ_2$ would have been split in a similar way if tuples with budgets smaller and greater than 200.000 would be stored

- In most cases intuition can be used to build horizontal partitions. Let $\{t_1, t_2, t_3\}$, $\{t_4, t_5\}$, and $\{t_2, t_3, t_4, t_5\}$ be query results. Then tuples would be fragmented in the following way:



Vertical Fragmentation

- **Objective** of vertical fragmentation is to partition a relation into a set of smaller relations so that many of the applications will run on only one fragment.
- Vertical fragmentation of a relation R produces fragments $R_1, R_2, \ldots$, each of which contains a **subset of** R's **attributes**.
- Vertical fragmentation is defined using the **projection operation** of the relational algebra:

$$\Pi_{A_1, A_2, \ldots, A_n}(R)$$

- **Example**:

$$PROJ_1 = \Pi_{PNO, BUDGET}(PROJ)$$
$$PROJ_2 = \Pi_{PNO, PNAME, LOC}(PROJ)$$

- Vertical fragmentation has also been studied for (centralized) DBMS
  - Smaller relations, and hence less page accesses
  - e.g., MONET system

Vertical Fragmentation . . .

- Vertical fragmentation is **inherently more complicated** than horizontal fragmentation
  - In horizontal partitioning: for n simple predicates, the number of possible minterms is 2^n; some of them can be ruled out by existing implications/constraints.
  - In vertical partitioning: for m non-primary key attributes, the number of possible fragments is equal to $B(m)$ (= the mth Bell number), i.e., the number of partitions of a set with m members.
    - For large numbers, $B(m) \approx m^m$ (e.g., $B(15) = 10^9$)
- Optimal solutions are not feasible, and heuristics need to be applied.

Vertical Fragmentation . . .

- Two types of heuristics for vertical fragmentation exist:
  - **Grouping**: assign each attribute to one fragment, and at each step, join some of the fragments until some criteria is satisfied.
    - Bottom-up approach
  - **Splitting**: starts with a relation and decides on beneficial partitionings based on the access behaviour of applications to the attributes.
    - Top-down approach
    - Results in non-overlapping fragments
    - “Optimal” solution is probably closer to the full relation than to a set of small relations with only one attribute
    - Only vertical fragmentation is considered here

- **Application information:** The major information required as input for vertical fragmentation is related to applications
  - Since vertical fragmentation places in one fragment those attributes usually accessed together, there is a need for some measure that would define more precisely the notion of “togetherness”, i.e., how closely related the attributes are.
  - This information is obtained from queries and collected in the *Attribute Usage Matrix* and *Attribute Affinity Matrix*.

- Given are the user queries/applications $Q = (q_1, \dots, q_q)$ that will run on relation $R(A_1, \dots, A_n)$
- **Attribute Usage Matrix**: Denotes which query uses which attribute:

$$use(q_i, A_j) = \begin{cases} 1 & \text{iff } q_i \text{ uses } A_j \\ 0 & \text{otherwise} \end{cases}$$

  - The $use(q_i, \bullet)$ vectors for each application are easy to define if the designer knows the applications that willl run on the DB (consider also the 80-20 rule)

- **Example**: Consider the following relation:

$$PROJ(PNO, PNAME, BUDGET, LOC)$$

and the following queries:

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

- Lets abbreviate $A_1 = PNO, A_2 = PNAME, A_3 = BUDGET, A_4 = LOC$
- Attribute Usage Matrix

	A_1	A_2	A_3	A_4
q_1	1	0	1	0
q_2	0	1	1	0
q_3	0	1	0	1
q_4	0	0	1	1

- **Attribute Affinity Matrix**: Denotes the frequency of two attributes A_i and A_j with respect to a set of queries $Q = (q_1, \ldots, q_n)$:

$$aff(A_i, A_j) = \sum_{k:\, \substack{use(q_k, A_i)=1, \\ use(q_k, A_j)=1}} \left(\sum_{\text{sites } l} ref_l(q_k) acc_l(q_k) \right)$$

  where

  - $ref_l(q_k)$ is the cost (= number of accesses to (A_i, A_j)) of query q_K at site l
  - $acc_l(q_k)$ is the frequency of query q_k at site l

Vertical Fragmentation . . .

- **Example (contd.):** Let the cost of each query be $ref_l(q_k) = 1$, and the frequency $acc_l(q_k)$ of the queries be as follows:

$Site1$	$Site2$	$Site3$
$acc_1(q_1) = 15$	$acc_2(q_1) = 20$	$acc_3(q_1) = 10$
$acc_1(q_2) = 5$	$acc_2(q_2) = 0$	$acc_3(q_2) = 0$
$acc_1(q_3) = 25$	$acc_2(q_3) = 25$	$acc_3(q_3) = 25$
$acc_1(q_4) = 3$	$acc_2(q_4) = 0$	$acc_3(q_4) = 0$

- Attribute affinity matrix $aff(A_i, A_j) =$

	A_1	A_2	A_3	A_4
A_1	45	0	45	0
A_2	0	80	5	75
A_3	45	5	53	3
A_4	0	75	3	78

  – e.g., $aff(A_1, A_3) = \sum_{k=1}^{1} \sum_{l=1}^{3} acc_l(q_k) = acc_1(q_1) + acc_2(q_1) + acc_3(q_1) = 45$ (q_1 is the only query to access both A_1 and A_3)

Vertical Fragmentation . . .

- Take the attribute affinity matrix (AA) and reorganize the attribute orders to form clusters where the attributes in each cluster demonstrate high affinity to one another.
- **Bond energy algorithm (BEA)** has been suggested to be useful for that purpose for several reasons:
  - It is designed specifically to determine groups of similar items as opposed to a linear ordering of the items.
  - The final groupings are insensitive to the order in which items are presented.
  - The computation time is reasonable ($O(n^2)$, where n is the number of attributes)
- **BEA**:
  - Input: AA matrix
  - Output: Clustered AA matrix (CA)
  - Permutation is done in such a way to maximize the following **global affinity mesaure** (affinity of A_i and A_j with their neighbors):

$$AM = \sum_{i=1}^{n} \sum_{j=1}^{n} \textit{aff}(A_i, A_j)[\textit{aff}(A_i, A_{j-1}) + \textit{aff}(A_i, A_{j+1}) + \textit{aff}(A_{i-1}, A_j) + \textit{aff}(A_{i+1}, A_j)]$$

Vertical Fragmentation . . .

- **Example (contd.)**: Attribute Affinity Matrix CA after running the BEA

$$
\begin{array}{c} \\ A_1 \\ A_3 \\ A_2 \\ A_4 \end{array}
\begin{array}{c} \begin{array}{cccc} A_1 & A_3 & A_2 & A_4 \end{array} \\
\begin{bmatrix} 45 & 45 & 0 & 0 \\ 45 & 53 & 5 & 3 \\ 0 & 5 & 80 & 75 \\ 0 & 3 & 75 & 78 \end{bmatrix} \end{array}
$$

  - Elements with similar values are grouped together, and two clusters can be identified
  - An additional partitioning algorithm is needed to identify the clusters in CA
    - ∗ Usually more clusters and more than one candidate partitioning, thus additional steps are needed to select the best clustering.
  - The resulting fragmentation after partitioning (PNO is added in $PROJ_2$ explicilty as key):

$$PROJ_1 = \{PNO, BUDGET\}$$
$$PROJ_2 = \{PNO, PNAME, LOC\}$$

Correctness of Vertical Fragmentation

- Relation R is decomposed into fragments $R_1, R_2, \ldots, R_n$
  - e.g., $PROJ = \{PNO, BUDGET, PNAME, LOC\}$ into $PROJ_1 = \{PNO, BUDGET\}$ and $PROJ_2 = \{PNO, PNAME, LOC\}$
- **Completeness**
  - Guaranteed by the partitioning algortihm, which assigns each attribute in A to one partition
- **Reconstruction**
  - Join to reconstruct vertical fragments
  - $R = R_1 \bowtie \cdots \bowtie R_n = PROJ_1 \bowtie PROJ_2$
- **Disjointness**
  - Attributes have to be disjoint in VF. Two cases are distinguished:
    * If tuple IDs are used, the fragments are really disjoint
    * Otherwise, key attributes are replicated automatically by the system
    * e.g., PNO in the above example

Mixed Fragmentation

- In most cases simple horizontal or vertical fragmentation of a DB schema will not be sufficient to satisfy the requirements of the applications.
- **Mixed fragmentation (hybrid fragmentation)**: Consists of a horizontal fragment followed by a vertical fragmentation, or a vertical fragmentation followed by a horizontal fragmentation
- Fragmentation is defined using the selection and projection operations of relational algebra:

$$\sigma_p(\Pi_{A_1,\ldots,A_n}(R))$$

$$\Pi_{A_1,\ldots,A_n}(\sigma_p(R))$$

Replication and Allocation

- **Replication**: Which fragements shall be stored as multiple copies?
  - Complete Replication
    - ∗ Complete copy of the database is maintained in each site
  - Selective Replication
    - ∗ Selected fragments are replicated in some sites
- **Allocation**: On which sites to store the various fragments?
  - Centralized
    - ∗ Consists of a single DB and DBMS stored at one site with users distributed across the network
  - Partitioned
    - ∗ Database is partitioned into disjoint fragments, each fragment assigned to one site

Replication . . .

- Replicated DB
  - **fully replicated**: each fragment at each site
  - **partially replicated**: each fragment at some of the sites
- Non-replicated DB (= partitioned DB)
  - **partitioned**: each fragment resides at only one site
- Rule of thumb:
  - If $\frac{\text{read only queries}}{\text{update queries}} \geq 1$, then replication is advantageous, otherwise replication may cause problems

Replication . . .

- Comparison of replication alternatives

	Full-replication	Partial-replication	Partitioning
QUERY PROCESSING	Easy	← Same Difficulty →	
DIRECTORY MANAGEMENT	Easy or Non-existant	← Same Difficulty →	
CONCURRENCY CONTROL	Moderate	Difficult	Easy
RELIABILITY	Very high	High	Low
REALITY	Possible application	Realistic	Possible application

Fragment Allocation

- **Fragment allocation problem**
  - Given are:
    – fragments $F = \{F_1, F_2, ..., F_n\}$
    – network sites $S = \{S_1, S_2, ..., S_m\}$
    – and applications $Q = \{q_1, q_2, ..., q_l\}$
  - Find: the ”optimal” distribution of F to S

- **Optimality**
  - Minimal cost
    - ∗ Communication + storage + processing (read and update)
    - ∗ Cost in terms of time (usually)
  - Performance
    - ∗ Response time and/or throughput
  - Constraints
    - ∗ Per site constraints (storage and processing)

Fragment Allocation . . .

- **Required information**
  - Database Information
    - selectivity of fragments
    - size of a fragment
  - Application Information
    - RR_{ij}: number of read accesses of a query q_i to a fragment F_j
    - UR_{ij}: number of update accesses of query q_i to a fragment F_j
    - u_{ij}: a matrix indicating which queries updates which fragments,
    - r_{ij}: a similar matrix for retrievals
    - originating site of each query
  - Site Information
    - USC_k: unit cost of storing data at a site S_k
    - LPC_k: cost of processing one unit of data at a site S_k
  - Network Information
    - communication cost/frame between two sites
    - frame size

Fragment Allocation ...

- We present an **allocation model** which attempts to
  - minimize the total cost of processing and storage
  - meet certain response time restrictions
- General Form:

$$\min(\text{Total Cost})$$

  - subject to
    - response time constraint
    - storage constraint
    - processing constraint
- Functions for the total cost and the constraints are presented in the next slides.
- Decision variable x_{ij}

$$x_{ij} = \begin{cases} 1 & \text{if fragment } F_i \text{ is stored at site } S_j \\ 0 & \text{otherwise} \end{cases}$$

Fragment Allocation . . .

- The **total cost function** has two components: storage and query processing.

$$TOC = \sum_{S_k \in S} \sum_{F_j \in F} STC_{jk} + \sum_{q_i \in Q} QPC_i$$

  - **Storage cost** of fragment F_j at site S_k:

$$STC_{jk} = USC_k * size(F_i) * x_{ij}$$

    where USC_k is the unit storage cost at site k

  - **Query processing cost** for a query q_i is composed of two components:
    - composed of processing cost (PC) and transmission cost (TC)

$$QPC_i = PC_i + TC_i$$

Fragment Allocation ...

- **Processing cost** is a sum of three components:
  - access cost (AC), integrity contraint cost (IE), concurency control cost (CC)

$$PC_i = AC_i + IE_i + CC_i$$

  - **Access cost**:

$$AC_i = \sum_{s_k \in S} \sum_{F_j \in F} (UR_{ij} + RR_{ij}) * x_{ij} * LPC_k$$

    where LPC_k is the unit process cost at site k
  - **Integrity and concurrency costs**:
    * Can be similarly computed, though depends on the specific constraints

- **Note:** AC_i assumes that processing a query involves decomposing it into a set of subqueries, each of which works on a fragment, ...,
  - This is a very simplistic model
  - Does not take into consideration different query costs depending on the operator or different algorithms that are applied

Fragment Allocation . . .

- The **transmission cost** is composed of two components:
  - Cost of processing updates (TCU) and cost of processing retrievals (TCR)

$$TC_i = TCU_i + TCR_i$$

  - **Cost of updates**:
    - Inform all the sites that have replicas + a short confirmation message back

$$TCU_i = \sum_{S_k \in S} \sum_{F_j \in F} u_{ij} * (\text{update message cost} + \text{acknowledgment cost})$$

  - **Retrieval cost**:
    - Send retrieval request to all sites that have a copy of fragments that are needed + sending back the results from these sites to the originating site.

$$TCR_i = \sum_{F_j \in F} \min_{S_k \in S} *(\text{cost of retrieval request} + \text{cost of sending back the result})$$

Fragment Allocation . . .

- Modeling the **constraints**
  - **Response time** constraint for a query q_i

    $$\text{execution time of } q_i \leq \text{ max. allowable response time for } q_i$$

  - **Storage** constraints for a site S_k

    $$\sum_{F_j \in F} \text{storage requirement of } F_j \text{ at } S_k \leq \text{ storage capacity of } S_k$$

  - **Processing** constraints for a site S_k

    $$\sum_{q_i \in Q} \text{processing load of } q_i \text{ at site } S_k \leq \text{ processing capacity of} S_k$$

Fragment Allocation . . .

- **Solution Methods**
  - The complexity of this allocation model/problem is NP-complete
  - Correspondence between the allocation problem and similar problems in other areas
    - ∗ Plant location problem in operations research
    - ∗ Knapsack problem
    - ∗ Network flow problem
  - Hence, solutions from these areas can be re-used
  - Use different heuristics to reduce the search space
    - ∗ Assume that all candidate partitionings have been determined together with their associated costs and benefits in terms of query processing.
      - · The problem is then reduced to find the optimal partitioning and placement for each relation
    - ∗ Ignore replication at the first step and find an optimal non-replicated solution
      - · Replication is then handeled in a second step on top of the previous non-replicated solution.

Conclusion

- Distributed design decides on the placement of (parts of the) data and programs across the sites of a computer network
- On the abstract level there are two patterns: Top-down and Bottom-up
- On the detail level design answers two key questions: fragmentation and allocation/replication of data
  - Horizontal fragmentation is defined via the selection operation $\sigma_p(R)$
    - Rewrites the queries of each site in the conjunctive normal form and finds a minimal and complete set of conjunctions to determine fragmentation
  - Vertical fragmentation via the projection operation $\pi_A(R)$
    - Computes the attribute affinity matrix and groups "similar" attributes together
  - Mixed fragmentation is a combination of both approaches
- Allocation/Replication of data
  - Type of replication: no replication, partial replication, full replication
  - Optimal allocation/replication modelled as a cost function under a set of constraints
  - The complexity of the problem is NP-complete
  - Use of different heuristics to reduce the complexity