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 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
• 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 relation 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

⇒ 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.
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
- Example

Data

<table>
<thead>
<tr>
<th>ENO</th>
<th>ENAME</th>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>J. Doe</td>
<td>Elect. Eng.</td>
</tr>
<tr>
<td>E2</td>
<td>M. Smith</td>
<td>Syst. Anal.</td>
</tr>
<tr>
<td>E4</td>
<td>J. Miller</td>
<td>Programmer</td>
</tr>
<tr>
<td>E5</td>
<td>B. Casey</td>
<td>Syst. Anal.</td>
</tr>
<tr>
<td>E8</td>
<td>J. Jones</td>
<td>Syst. Anal.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ENO</th>
<th>PNO</th>
<th>RESP</th>
<th>DUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>P1</td>
<td>Manager</td>
<td>12</td>
</tr>
<tr>
<td>E2</td>
<td>P1</td>
<td>Analyst</td>
<td>24</td>
</tr>
<tr>
<td>E2</td>
<td>P2</td>
<td>Analyst</td>
<td>6</td>
</tr>
<tr>
<td>E3</td>
<td>P3</td>
<td>Consultant</td>
<td>10</td>
</tr>
<tr>
<td>E4</td>
<td>P3</td>
<td>Engineer</td>
<td>48</td>
</tr>
<tr>
<td>E4</td>
<td>P2</td>
<td>Programmer</td>
<td>18</td>
</tr>
<tr>
<td>E5</td>
<td>P2</td>
<td>Manager</td>
<td>24</td>
</tr>
<tr>
<td>E6</td>
<td>P4</td>
<td>Manager</td>
<td>48</td>
</tr>
<tr>
<td>E7</td>
<td>P3</td>
<td>Engineer</td>
<td>36</td>
</tr>
<tr>
<td>E8</td>
<td>P3</td>
<td>Manager</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
<th>TITLE</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Instrumentation</td>
<td>150000</td>
<td>Montreal</td>
<td>Elect. Eng.</td>
<td>40000</td>
</tr>
<tr>
<td>P2</td>
<td>Database Develop</td>
<td>135000</td>
<td>New York</td>
<td>Syst. Anal.</td>
<td>34000</td>
</tr>
<tr>
<td>P3</td>
<td>CAD/CAM</td>
<td>250000</td>
<td>New York</td>
<td>Mech. Eng.</td>
<td>27000</td>
</tr>
<tr>
<td>P4</td>
<td>Maintenance</td>
<td>310000</td>
<td>Paris</td>
<td>Programmer</td>
<td>24000</td>
</tr>
</tbody>
</table>

E-R Diagram

PAY

EMP

PROJ

ENQ, JNO, RESP, DUR

ENQ, JNO, TITLE

JNO, JNAME, BUDGET, LOC

L₁

L₂

L₃
• **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

<table>
<thead>
<tr>
<th>PROJ</th>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P1</td>
<td>Instrumentation</td>
<td>150000</td>
<td>Montreal</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>Database Develop</td>
<td>135000</td>
<td>New York</td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>CAD/CAM</td>
<td>250000</td>
<td>New York</td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>Maintenance</td>
<td>310000</td>
<td>Paris</td>
</tr>
<tr>
<td></td>
<td>P5</td>
<td>CAD/CAM</td>
<td>500000</td>
<td>Boston</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROJ₁</th>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Instrumentation</td>
<td>150000</td>
<td>Montreal</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>Database Develop</td>
<td>135000</td>
<td>New York</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROJ₂</th>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3</td>
<td>CAD/CAM</td>
<td>250000</td>
<td>New York</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>Maintenance</td>
<td>310000</td>
<td>Paris</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>CAD/CAM</td>
<td>500000</td>
<td>Boston</td>
<td></td>
</tr>
</tbody>
</table>
**Example (contd.):** Vertical fragmentation of PROJ relation

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

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Instrumentation</td>
<td>150000</td>
<td>Montreal</td>
</tr>
<tr>
<td>P2</td>
<td>Database Develop</td>
<td>135000</td>
<td>New York</td>
</tr>
<tr>
<td>P3</td>
<td>CAD/CAM</td>
<td>250000</td>
<td>New York</td>
</tr>
<tr>
<td>P4</td>
<td>Maintenance</td>
<td>310000</td>
<td>Paris</td>
</tr>
<tr>
<td>P5</td>
<td>CAD/CAM</td>
<td>500000</td>
<td>Boston</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PNO</th>
<th>BUDGET</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>150000</td>
</tr>
<tr>
<td>P2</td>
<td>135000</td>
</tr>
<tr>
<td>P3</td>
<td>250000</td>
</tr>
<tr>
<td>P4</td>
<td>310000</td>
</tr>
<tr>
<td>P5</td>
<td>500000</td>
</tr>
</tbody>
</table>
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 $\nabla$ 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
• **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:**

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

$$\sigma_{\text{BUDGET} \geq 200000}(\text{PROJ})$$
• **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'\}

\[
\begin{align*}
PROJ_1 &= \sigma_{LOC='Montreal'}(PROJ) \\
PNO &\quad PNAME &\quad BUDGET &\quad LOC \\
P1 &\quad \text{Instrumentation} &\quad 150000 &\quad \text{Montreal} \\
\end{align*}
\]

\[
\begin{align*}
PROJ_2 &= \sigma_{LOC='NewYork'}(PROJ) \\
PNO &\quad PNAME &\quad BUDGET &\quad LOC \\
P2 &\quad \text{Database Develop.} &\quad 135000 &\quad \text{New York} \\
P3 &\quad \text{CAD/CAM} &\quad 250000 &\quad \text{New York} \\
\end{align*}
\]

\[
\begin{align*}
PROJ_3 &= \sigma_{LOC='Paris'}(PROJ) \\
PNO &\quad PNAME &\quad BUDGET &\quad LOC \\
P4 &\quad \text{Maintenance} &\quad 310000 &\quad \text{Paris} \\
\end{align*}
\]

• 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:

\[
\begin{align*}
(LOC = 'Montreal') \land (BUDGET \leq 200000) \\
(LOC = 'Montreal') \land (BUDGET > 200000) \\
(LOC = 'NewYork') \land (BUDGET \leq 200000) \\
(LOC = 'NewYork') \land (BUDGET > 200000) \\
(LOC = 'Paris') \land (BUDGET \leq 200000) \\
(LOC = 'Paris') \land (BUDGET > 200000)
\end{align*}
\]
Example (contd.): Now, $PROJ_2$ will be split in two fragments

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

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Instrumentation</td>
<td>150000</td>
<td>Montreal</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4</td>
<td>Maintenance</td>
<td>310000</td>
<td>Paris</td>
</tr>
</tbody>
</table>

\[
PROJ_2 = \sigma_{LOC='NY'\land BUDGET<200000}(PROJ)
\]

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>Database Develop.</td>
<td>135000</td>
<td>New York</td>
</tr>
</tbody>
</table>

\[
PROJ_2' = \sigma_{LOC='NY'\land BUDGET\geq200000}(PROJ)
\]

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3</td>
<td>CAD/CAM</td>
<td>250000</td>
<td>New York</td>
</tr>
</tbody>
</table>

$\text{PROJ}_1$ and $\text{PROJ}_2$ 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:
**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:**

$$
\begin{align*}
PROJ_1 &= \Pi_{PNO, BUDGET}(PROJ) \\
PROJ_2 &= \Pi_{PNO, PNAME, LOC}(PROJ)
\end{align*}
$$

Vertical fragmentation has also been studied for (centralized) DBMS

- Smaller relations, and hence less page accesses
- e.g., MONET system
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 \( m \)th 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.
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 $Q = (q_1, \ldots, q_q)$ that will run on relation $R(A_1, \ldots, A_n)$

• **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 will run on the DB (consider also the 80-20 rule)
Vertical Fragmentation …

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

\[
PROJ(PNO, PNAME, BUDGET, LOC)
\]

and the following queries:

\[
\begin{align*}
q_1 &= \text{SELECT BUDGET FROM PROJ WHERE PNO=Value} \\
q_2 &= \text{SELECT PNAME,BUDGET FROM PROJ} \\
q_3 &= \text{SELECT PNAME FROM PROJ WHERE LOC=Value} \\
q_4 &= \text{SELECT SUM(BUDGET) FROM PROJ WHERE LOC =Value}
\end{align*}
\]

- Lets abbreviate \(A_1 = PNO, A_2 = PNAME, A_3 = BUDGET, A_4 = LOC\)

- Attribute Usage Matrix

\[
\begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1
\end{bmatrix}
\]
• **Attribute Affinity Matrix**: Denotes the frequency of two attributes $A_i$ and $A_j$ with respect to a set of queries $Q = (q_1, \ldots, q_n)$:

$$
\text{aff}(A_i, A_j) = \sum_{k: \text{use}(q_k, A_i) = 1, \text{use}(q_k, A_j) = 1} \left( \sum_{\text{sites } l} \text{ref}_l(q_k) \text{acc}_l(q_k) \right)
$$

where

- $\text{ref}_l(q_k)$ is the cost (= number of accesses to $(A_i, A_j)$) of query $q_K$ at site $l$
- $\text{acc}_l(q_k)$ 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:

<table>
<thead>
<tr>
<th>Site1</th>
<th>Site2</th>
<th>Site3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( acc_1(q_1) = 15 )</td>
<td>( acc_2(q_1) = 20 )</td>
<td>( acc_3(q_1) = 10 )</td>
</tr>
<tr>
<td>( acc_1(q_2) = 5 )</td>
<td>( acc_2(q_2) = 0 )</td>
<td>( acc_3(q_2) = 0 )</td>
</tr>
<tr>
<td>( acc_1(q_3) = 25 )</td>
<td>( acc_2(q_3) = 25 )</td>
<td>( acc_3(q_3) = 25 )</td>
</tr>
<tr>
<td>( acc_1(q_4) = 3 )</td>
<td>( acc_2(q_4) = 0 )</td>
<td>( acc_3(q_4) = 0 )</td>
</tr>
</tbody>
</table>

• Attribute affinity matrix \( \text{aff}(A_i, A_j) = \)

\[
\begin{pmatrix}
A_1 & A_2 & A_3 & A_4 \\
A_1 & 45 & 0 & 45 & 0 \\
A_2 & 0 & 80 & 5 & 75 \\
A_3 & 45 & 5 & 53 & 3 \\
A_4 & 0 & 75 & 3 & 78
\end{pmatrix}
\]

- e.g., \( \text{aff}(A_1, A_3) = \sum_{k=1}^{1} \sum_{l=1}^{3} acc_l(q_k) = acc_1(q_1) + acc_2(q_1) + acc_3(q_1) = 45 \)

(q_1 is the only query to access both \( A_1 \) and \( A_3 \)
• 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 measure (affinity of $A_i$ and $A_j$ with their neighbors):

$$AM = \sum_{i=1}^{n} \sum_{j=1}^{n} \text{aff}(A_i, A_j)[\text{aff}(A_i, A_{j-1}) + \text{aff}(A_i, A_{j+1}) + \text{aff}(A_{i-1}, A_j) + \text{aff}(A_{i+1}, A_j)]$$
• Example (contd.): Attribute Affinity Matrix $CA$ after running the BEA

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

- 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):

$$
PROJ_1 = \{PNO, BUDGET\} \\
PROJ_2 = \{PNO, PNAMES, LOC\}
$$
Correctness of Vertical Fragmentation

- Relation $R$ is decomposed into fragments $R_1, R_2, \ldots, R_n$
  - e.g., $\text{PROJ} = \{\text{PNO, BUDGET, PNAME, LOC}\}$ into $\text{PROJ}_1 = \{\text{PNO, BUDGET}\}$ and $\text{PROJ}_2 = \{\text{PNO, PNAME, LOC}\}$

- Completeness
  - Guaranteed by the partitioning algorithm, which assigns each attribute in $A$ to one partition

- Reconstruction
  - Join to reconstruct vertical fragments
  - $R = R_1 \bowtie \cdots \bowtie R_n = \text{PROJ}_1 \bowtie \text{PROJ}_2$

- 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., $\text{PNO}$ 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 fragments 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
• 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
- Comparison of replication alternatives

<table>
<thead>
<tr>
<th></th>
<th>Full-replication</th>
<th>Partial-replication</th>
<th>Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY PROCESSING</td>
<td>Easy</td>
<td></td>
<td>Same Difficulty</td>
</tr>
<tr>
<td>DIRECTORY MANAGEMENT</td>
<td>Easy or Non-existant</td>
<td></td>
<td>Same Difficulty</td>
</tr>
<tr>
<td>CONCURRENCY CONTROL</td>
<td>Moderate</td>
<td>Difficult</td>
<td>Easy</td>
</tr>
<tr>
<td>RELIABILITY</td>
<td>Very high</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>REALITY</td>
<td>Possible application</td>
<td>Realistic</td>
<td>Possible application</td>
</tr>
</tbody>
</table>
Fragment Allocation

• Fragment allocation problem
  – Given are:
    – fragments $F = \{F_1, F_2, \ldots, F_n\}$
    – network sites $S = \{S_1, S_2, \ldots, S_m\}$
    – and applications $Q = \{q_1, q_2, \ldots, q_l\}$
  – 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)
• 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
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** \( x_{ij} \)

\[
x_{ij} = \begin{cases} 
1 & \text{if fragment } F_i \text{ is stored at site } S_j \\
0 & \text{otherwise}
\end{cases}
\]
• 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 \( S_k \):

\[
STC_{jk} = USC_k \ast size(F_i) \ast x_{ij}
\]

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)

\[
QPC_i = PC_i + TC_i
\]
• **Processing cost** is a sum of three components:
  – access cost (AC), integrity constraint cost (IE), concurrency control cost (CC)

\[
P_{C_i} = A_{C_i} + I_{E_i} + C_{C_i}
\]

– **Access cost:**

\[
A_{C_i} = \sum_{s_k \in S} \sum_{F_j \in F} (U_{R_{ij}} + R_{R_{ij}}) \times x_{ij} \times L_{P_{C_k}}
\]

where \(L_{P_{C_k}}\) is the unit process cost at site \(k\)

– **Integrity and concurrency costs:**
  * Can be similarly computed, though depends on the specific constraints

• **Note:** \(A_{C_i}\) 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
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

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

- 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.

\[ TCR_i = \sum_{F_j \in F} \min_{S_k \in S} \ast (\text{cost of retrieval request + cost of sending back the result}) \]
• 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 $S_k$
    
    \[
    \sum_{q_i \in Q} \text{processing load of } q_i \text{ at site } S_k \leq \text{ processing capacity of } S_k
    \]
• 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 handled in a second step on top of the previous non-replicated solution.
• 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